MapMagic 2 World Generator

A node based infinite land generator

The manual could be found here:

https://gitlab.com/denispahunov/mapmagic/wikis/home

Quick start guide:

- Create the new MapMagic Graph asset by clicking Create in a Project View, and select MapMagic -> Ten
- Drag and drop the created Graph to scene
- Double-click on the Graph asset in the project view to open up the editor window (or select MapMagic obj
- In Editor Window use:
  - middle button to pan (or Alt+right button)
  - mouse wheel to zoom (or keyboard shift and "=" or "-")
  - left mouse button to drag generators and create connections
  - right mouse button to create/remove/duplicate/preview, etc
- See https://gitlab.com/denispahunov/mapmagic/wikis/Quick%20Start for more detail

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Terrains;
using MapMagic.Nodes;
using MapMagic.Nodes.MatrixGenerators;

using UnityEngine.Profiling;

namespace MapMagic.Brush
{
 public static class BrushOps
 {
  public static bool CheckHeightPixelSize (Terrain terrain, float refPixelSize)
  /// Checks if terrain pixel size matches reference pixel size
  {
   TerrainData data = terrain.terrainData;
   float pixelSize = data.size.x / (data.heightmapResolution-1);
   return !(refPixelSize > pixelSize+0.0001f  ||  refPixelSize < pixelSize-0.0001f);
```

```
  }


  public static CoordRect GetHeightPixelRect (Terrain terrain) => GetTerrainPixelRect(terrain, terrain.terrai


  public static CoordRect GetTerrainPixelRect (Terrain terrain, int resolution)
  {
   float terrainPixelSize = terrain.terrainData.size.x / (resolution-1);

    return new CoordRect(
     Mathf.RoundToInt(terrain.transform.position.x/terrainPixelSize),
     Mathf.RoundToInt(terrain.transform.position.z/terrainPixelSize),
     resolution,
     resolution);
  }


 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Products;
using MapMagic.Expose;
using System.Threading;
using System;

namespace MapMagic.Brush
{
 public interface IBrushRead
 {
  void ReadTerrains(TerrainCache[] terrainCaches, TileData tileData);
 }

 public interface IBrushWrite
 {
  void WriteTerrains(TerrainCache[] terrainCaches, CacheChange change, TileData tileData);
 }
```

```
[SelectionBase]
[ExecuteInEditMode] //to call onEnable, then it will subscribe to editor update
[HelpURL("https://gitlab.com/denispahunov/mapmagic/wikis/home")]
[DisallowMultipleComponent]
public class MapMagicBrush : MonoBehaviour
{
 public static readonly SemVer version = new SemVer(1,0,0);

 [NonSerialized] public bool draw;

 public Preset preset = null; //note brush is using copy. Can't create new instance (it's unityobj) here, so us
 public Preset sourcePreset = null; //to save preset with 'Save' button
 public Preset[] quickPresets = new Preset[0]; //quick selection
 public void AssignPreset (Preset newPreset)
 {
  if (newPreset == null) { Debug.Log("Selected Brush preset is null"); return; }
  sourcePreset = newPreset;
  preset = Instantiate(newPreset);
 }
 public void AssignQuickPreset (int num)
 {
  if (quickPresets.Length > num)
```

```
  AssignPreset(quickPresets[num]);
}

public Trace trace = new Trace(); //spacing tool

public Terrain[] terrains = new Terrain[0];
public Dictionary<Terrain,TerrainCache> terrainCaches = new Dictionary<Terrain,TerrainCache>();
public CacheChange cacheChange = new CacheChange();
public bool terrainsAdded = false; //called on OnEnable

public TuneStroke tuneStroke = new TuneStroke();

public Undo.Undo undo = new Undo.Undo();
public bool temp = false; //switching on new undo - otherwise it won't be recordered

//IMapMagic
public Graph Graph => preset?.graph;
public bool ContainsGraph (Graph graph) => preset?.graph==graph;
public TileData PreviewData { get; set; }

public Color mainColor = new Color(0, 0.5f, 1f, 1); //new Color(1,0.4f,0,1);
public Color falloffColor = new Color(0, 0.3f, 0.6f, 1); //new Color(0.6f, 0.1f, 0, 1); //alpha is 1
public float lineThickness = 5;

public bool guiBrush = true;
public bool guiPresets = false;
```

```
public bool guiGraph = false;
public bool guiProperties = false;
public bool guiTerrains;
public bool guiTuneStroke;
public bool guiSettings;

public int curUndoId = 0;  //to change something on undo and find out if this is a proper undo on undo redo
public int prevUndoId = 0;  //should be in object itself



public void OnEnable ()
{

 tuneStroke.brush = this;

 if (preset == null)
  preset = ScriptableObject.CreateInstance<Preset>();

 UpdateCaches();

 //updating cache on MapMagic change
 TerrainTile.OnTileApplied -= UpdateCache;
 TerrainTile.OnTileApplied += UpdateCache;
}

private void UpdateCache (TerrainTile tile, TileData data, StopToken stop) { UpdateCache(tile.draft?.terra
```

```
public void UpdateCaches ()
{
 for (int t=0; t<terrains.Length; t++)
 {
  TerrainCache cache;
  if (!terrainCaches.TryGetValue(terrains[t], out cache))
  {
   cache = new TerrainCache(terrains[t]);
   terrainCaches.Add(terrains[t], cache);
  }

  cache.UpdateCache();
 }
}


public void UpdateCache (Terrain terrain)
{
 if (!terrains.Contains(terrain))
  return;

 TerrainCache cache;
 if (!terrainCaches.TryGetValue(terrain, out cache))
 {
```

```
  cache = new TerrainCache(terrain);
  terrainCaches.Add(terrain, cache);
 }

 cache.UpdateCache();
}
```

```
public void Apply (Vector3 pos, bool isFirst=false)
/// Main fn to apply brush to terrain
/// isFirst: is this first (just clicked) stamp in stroke?
{
 Stamp stamp = new Stamp() {pos=(Vector2D)pos, radius=preset.radius, hardness=preset.hardness, mar

 Terrain[] stampTerrains = TerrainManager.GetTerrains(terrains, stamp.Min, stamp.Size);
 if (stampTerrains.Length == 0)
  return;

 TerrainCache[] stampTerrainCaches = new TerrainCache[stampTerrains.Length];
 for (int t=0; t<stampTerrains.Length; t++)
 {
  if (!terrainCaches.TryGetValue(stampTerrains[t], out TerrainCache cache))
   throw new Exception("No cache for terrain " + stampTerrains[t]);
  stampTerrainCaches[t] = cache;
```

```
}

TileData tileData = null;
//using (new Log.Timer("Area"))
{
 tileData = new TileData();
 tileData.area = stamp.GetArea(stampTerrains[0]);
 tileData.globals = new Globals(); //TODO: avoid creating it al the time
 tileData.globals.height = stampTerrains[0].terrainData.size.y;
 tileData.random = preset.graph.random;
 tileData.isPreview = true;
 tileData.isDraft = false;
}

//using (new Log.Timer("Override"))
{
 preset.ovd.SetIfContains("Position", typeof(Vector2D), pos);
 preset.ovd.SetIfContains("Position", typeof(Vector3), pos);
 preset.ovd.SetIfContains("Radius", typeof(float), preset.radius);
 preset.ovd.SetIfContains("Hardness", typeof(float), preset.hardness);
 preset.ovd.SetIfContains("PrevPosition", typeof(Vector2D), (Vector2D)trace.PrevStampPos);
 preset.ovd.SetIfContains("PrevPosition", typeof(Vector3), trace.PrevStampPos);
 preset.ovd.SetIfContains("CapturedPosition", typeof(Vector3), trace.capturedPosition);
 preset.ovd.SetIfContains("TerrainHeight", typeof(float), stampTerrains[0].terrainData.size.y);
 preset.ovd.SetIfContains("TerrainHeight", typeof(int), stampTerrains[0].terrainData.size.y);
 #if UNITY_EDITOR
```

```
 preset.ovd.SetIfContains("Shift", typeof(bool), Event.current.shift);
 preset.ovd.SetIfContains("Shift", typeof(int), Event.current.shift ? 1 : 0);
 preset.ovd.SetIfContains("Shift", typeof(float), Event.current.shift ? 1 : 0);
 #endif
}

//using (new Log.Timer("Read"))
 foreach (IBrushRead brushReadNode in BrushReadsOrdered(preset.graph))
  brushReadNode.ReadTerrains(stampTerrainCaches, tileData);

using (Log.Group("Generate"))
{
 StopToken stop = new StopToken();
 preset.graph.Generate(tileData, stop:stop, ovd:preset.ovd);
 preset.graph.Finalize(tileData, stop);
}

//using (new Log.Timer("Undo"))
{
 if (isFirst) undo.NewGroup(this);
 undo.Append(stampTerrains, stamp.Min, stamp.Size,
  readHeight: preset.graph.ContainsGeneratorOfType<BrushWriteHeight206>(),
  readSplats: preset.graph.ContainsGeneratorOfType<BrushWriteTextureSet>(),
  readGrass: preset.graph.ContainsGeneratorOfType<BrushWriteGrassSet>(),
  readTrees: preset.graph.ContainsGeneratorOfType<BrushWriteTrees>()
  );
```

```
 }

 //using (new Log.Timer("Write"))
  foreach (IBrushWrite brushWriteNode in BrushWritesOrdered(preset.graph))
   brushWriteNode.WriteTerrains(stampTerrainCaches, cacheChange, tileData);
}


private IEnumerable<IBrushRead> BrushReadsOrdered (Graph graph)
/// Equivalent of foreach GeneratorsOfType<IBrushRead> but heigts goes first (and checking enabled)
{
 foreach (IBrushRead heightNode in preset.graph.GeneratorsOfType<BrushReadHeight206>())
  yield return heightNode;

 foreach (IBrushRead brushNode in preset.graph.GeneratorsOfType<IBrushRead>())
  if (!(brushNode is BrushReadHeight206) && (brushNode as Generator).enabled)
   yield return brushNode;
}


private IEnumerable<IBrushWrite> BrushWritesOrdered (Graph graph)
/// Equivalent of foreach GeneratorsOfType<IBrushRead> but heigts goes first (and checking enabled)
{
 foreach (BrushWriteHeight206 heightNode in preset.graph.GeneratorsOfType<BrushWriteHeight206>())
  if (heightNode.enabled)
   yield return heightNode;
```

```
    foreach (IBrushWrite brushNode in preset.graph.GeneratorsOfType<IBrushWrite>())
     if (!(brushNode is BrushWriteHeight206) && (brushNode as Generator).enabled)
      yield return brushNode;
   }
  }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Terrains;
using MapMagic.Nodes;
using MapMagic.Nodes.MatrixGenerators;
using MapMagic.Expose;

using UnityEngine.Profiling;

namespace MapMagic.Brush
{
 [System.Serializable]
 [CreateAssetMenu(menuName = "MapMagic/Brush Preset", fileName = "Brush.asset", order = 124)]
 public class Preset : ScriptableObject
 {
  public Graph graph;

  public float radius = 100;
```

```
  public float hardness = 0.5f;
  public int margins = 0;
  public float spacing = 0.25f;

  public Override ovd = new Override();

  public bool guiShowUsedAutomatic = false;
  public bool guiShowAllAutomatic = false;

  public void SyncOvd ()
  {
   if (graph != null)
    ovd.Sync(graph.defaults);
  }

  public void CopyFrom (Preset other)
  {
   graph = other.graph;
   radius = other.radius;
   hardness = other.hardness;
   margins = other.margins;
   spacing = other.spacing;
   ovd = new Override(other.ovd);
  }
 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Products;
using MapMagic.Expose;
using System.Threading;
using System;

namespace MapMagic.Brush
{
 public class Stamp
 {
  public Vector2D pos;

  public float radius;
  public float hardness;
  public int margins; //TODO: make it world since it always affected by pixelSize. Or radius-related

  public Vector2D Min => pos-radius; //-margins
  public Vector2D Max => pos+radius; //+margins
```

```
  public Vector2D Size => new Vector2D(radius*2);




  public Area GetArea (Terrain closestTerrain)
  {
   CoordRect pixelRect = closestTerrain.PixelRect(pos-radius, (Vector2D)radius*2, TerrainControlType.Hei
   if (pixelRect.size.x < pixelRect.size.z) pixelRect.size.x = pixelRect.size.z; //ensuring pixelrect is square (ir
   if (pixelRect.size.z < pixelRect.size.x) pixelRect.size.z = pixelRect.size.x;

   Vector2D pixelSize = closestTerrain.PixelSize(TerrainControlType.Height);
   Vector2D worldPos = ((Vector2D)pixelRect.offset+0.5f) * pixelSize;  //+0.5 since world grid start from half
   Vector2D worldSize = ((Vector2D)pixelRect.size-1f) * pixelSize;  //and ends with half-pixel, so there is 1 p

   return new Area(worldPos, worldSize, pixelRect, margins);
  }



 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Products;
using MapMagic.Expose;
using System.Threading;
using System;

namespace MapMagic.Brush
{
 public class CacheChange
 {
  public CoordRect changeRect;

  [Flags] public enum Type { Height=1, Splats=2, Grass=4, Objects=8, Trees=16 };
  public Type type;

  public bool HasFlag (Type f) => (type & f) == f;
  public void AddFlag (Type f) => type = type | f;
  public bool HasAnyFlag => type!=0;
```

```
 public void AddRect (CoordRect rect)
 {
  if (changeRect.size.x==0  &&  changeRect.size.z==0) changeRect = rect;
  else changeRect = CoordRect.Combined(rect, changeRect);
 }

 public void Clear ()
 {
  type = 0;
  changeRect = new CoordRect(0,0,0,0);
 }
}


public class TerrainCache
{
 public Terrain terrain;

 public CoordRect rect;
 public Vector3 worldPos;
 public Vector3 worldSize;

 public Vector2D PixelSize => new Vector2D(worldSize.x/(rect.size.x-1), worldSize.z/(rect.size.z-1));

 private bool drawInstanced;
```

```
public TerrainCache (Terrain terrain)
{
 this.terrain = terrain;
 UpdateCache();
}

public void UpdateCache ()
{
 TerrainData terrainData = terrain.terrainData;

 worldPos = terrain.transform.position;
 worldSize = terrainData.size;

 int heightRes = terrainData.heightmapResolution;
 float terrainPixelSize = worldSize.x / (heightRes-1);

 rect = new CoordRect(
  Mathf.RoundToInt(worldPos.x/terrainPixelSize),
  Mathf.RoundToInt(worldPos.z/terrainPixelSize),
  heightRes,
  heightRes);

 drawInstanced = terrain.drawInstanced;
```

```
 LoadHeights();
 LoadTextures();
 LoadGrass();

 #if MM_DEBUG
 Debug.Log("Cache Updated");
 #endif
}

public void ApplyChanges (CacheChange change)
/// Saves all the changes to terrain
/// This should be called before rendering the new frame, not after each stamp (we can have several stam
{
 if (!change.HasAnyFlag  ||  change.changeRect.size.x==0  || change.changeRect.size.z==0)
  return;

 if (!CoordRect.IsIntersecting(change.changeRect, rect))
  return;

 using (Log.Group("ApplyChanges"))
 {
  using (Log.Group("ApplyHeights"))
  if (change.HasFlag(CacheChange.Type.Height)) ApplyHeights(change.changeRect);

  using (Log.Group("ApplyTextures"))
  if (change.HasFlag(CacheChange.Type.Splats)) ApplyTextures(change.changeRect);
```

```
  using (Log.Group("ApplyGrass"))
  if (change.HasFlag(CacheChange.Type.Grass)) ApplyGrass(change.changeRect);
 }
}

public void EndTrace ()
{
 terrain.drawInstanced = drawInstanced; //this is changed to false while applying heightmap to refresh visu
}

#region Heights

 public Matrix heights;
 private byte[] heightBytes;
 private Texture2D heightTex;
 private RenderTexture heightRenTex;

 public void LoadHeights ()
 /// Loads height data from whole terrain
 {
  float[,] heightsArr = terrain.terrainData.GetHeights(0,0,rect.size.x,rect.size.z);
  heights = new Matrix(rect);
  heights.ImportHeights(heightsArr);
 }
```

```
public void ReadHeights (Matrix matrix)  =>  Matrix.CopyIntersected(heights, matrix);

/// Fills matrix with loaded whole terrain data


public void WriteHeights (Matrix matrix)

/// Copies matrix data to loaded whole terrain data and marks changed rect

{

 matrix.Clamp01();

 Matrix.CopyIntersected(matrix, heights);

}


private void ApplyHeights (CoordRect applyRect)

/// Copies pixels within given applyRect from heights matrix to terrain on Apply Changes

{

 TerrainData terrainData = terrain.terrainData;


 CoordRect relApplyRect = new CoordRect(applyRect.offset - rect.offset, applyRect.size); //from here and

 CoordRect relFullRect = new CoordRect(Coord.zero, rect.size);


 CoordRect intersection = CoordRect.Intersected(relApplyRect, relFullRect);

 if (intersection.size.x<=0 || intersection.size.z<=0) return;


 //matrix to bytes

 if (heightBytes == null  ||  heightBytes.Length != intersection.size.x*intersection.size.z*4)

  heightBytes = new byte[intersection.size.x*intersection.size.z*4];


 float ushortEpsilon = 1f / 65535; //since setheights is using not full ushort range, but range-1
```

```
heights.ExportRawFloat(heightBytes, intersection.offset+heights.rect.offset, intersection.size, mult:0.5f-u

//bytes to texture2D
if (heightTex == null ||  heightTex.width != intersection.size.x  ||  heightTex.height != intersection.size.z)
 heightTex = new Texture2D(intersection.size.x, intersection.size.z, TextureFormat.RFloat, mipChain:fals

heightTex.LoadRawTextureData(heightBytes);
heightTex.Apply(updateMipmaps:false);

//texture2D to renderTexture
if (heightRenTex == null ||  heightRenTex.width != intersection.size.x  ||  heightRenTex.height != intersec
#if UNITY_2019_2_OR_NEWER
 heightRenTex = new RenderTexture(intersection.size.x, intersection.size.z, 32, RenderTextureFormat.R
#else
 heightRenTex = new RenderTexture(intersection.size.x, intersection.size.z, 32, RenderTextureFormat.R
#endif
Graphics.Blit(heightTex, heightRenTex);

//renderTexture to terrain
RenderTexture bacRenTex = RenderTexture.active;
RenderTexture.active = heightRenTex;

RectInt texRect = new RectInt(0,0, intersection.size.x, intersection.size.z);
terrain.drawInstanced = true;
terrainData.CopyActiveRenderTextureToHeightmap(texRect, new Vector2Int(intersection.offset.x, inters
//terrainData.DirtyHeightmapRegion(new RectInt(intersection.offset.x, intersection.offset.z, intersection.s
```

```
 //terrainData.DirtyTextureRegion("_Splat0", new RectInt(intersection.offset.x, intersection.offset.z, inters
 //terrainData.SetBaseMapDirty();
 //terrainData.UpdateDirtyRegion(intersection.offset.x, intersection.offset.z, intersection.size.x, intersectio
 //terrain.GetComponent<TerrainCollider>().enabled = false;
 //terrainData.SyncHeightmap(); //doesn't seems to make difference with DirtyHeightmapRegion, waiting
 //terrain.GetComponent<TerrainCollider>().enabled = true;
 //terrainData.UpdateDirtyRegion(intersection.offset.x, intersection.offset.z, intersection.size.x, intersectio
 //terrainData.DirtyTextureRegion(

 //terrain.heightmapPixelError = 1;

 RenderTexture.active = bacRenTex;
}

[Obsolete] private void ApplyHeights (Matrix matrix)
/// Copies matrix to terrain, while the matrix size is smalle than terrain
/// Tested and working, however not used. Just keeping as a reference
{
 //saving to cache
 Matrix.CopyIntersected(matrix, heights);

 //applying to terrain
 CoordRect intersection = CoordRect.Intersected(rect, matrix.rect);
 if (intersection.size.x<=0 || intersection.size.z<=0) return;

 //matrix to bytes
```

```
if (heightBytes == null  ||  heightBytes.Length != intersection.size.x*intersection.size.z*4)
 heightBytes = new byte[intersection.size.x*intersection.size.z*4];

float ushortEpsilon = 1f / 65535; //since setheights is using not full ushort range, but range-1
matrix.ExportRawFloat(heightBytes, intersection.offset, intersection.size, mult:0.5f-ushortEpsilon);

//bytes to texture2D
if (heightTex == null ||  heightTex.width != intersection.size.x  ||  heightTex.height != intersection.size.z)
 heightTex = new Texture2D(intersection.size.x, intersection.size.z, TextureFormat.RFloat, mipChain:fals

heightTex.LoadRawTextureData(heightBytes);
heightTex.Apply(updateMipmaps:false);

//texture2D to renderTexture
if (heightRenTex == null ||  heightRenTex.width != intersection.size.x  ||  heightRenTex.height != intersec
#if UNITY_2019_2_OR_NEWER
 heightRenTex = new RenderTexture(intersection.size.x, intersection.size.z, 32, RenderTextureFormat.R
#else
 heightRenTex = new RenderTexture(intersection.size.x, intersection.size.z, 32, RenderTextureFormat.R
#endif
Graphics.Blit(heightTex, heightRenTex);

//renderTexture to terrain
RenderTexture bacRenTex = RenderTexture.active;
RenderTexture.active = heightRenTex;
```

```
 RectInt texRect = new RectInt(0,0, intersection.size.x, intersection.size.z);
 terrain.terrainData.CopyActiveRenderTextureToHeightmap(texRect, new Vector2Int(intersection.offset.x
 //data.DirtyHeightmapRegion(texRect, TerrainHeightmapSyncControl.HeightAndLod); //or seems a bit fa
 //data.SyncHeightmap(); //doesn't seems to make difference with DirtyHeightmapRegion, waiting for rea

 RenderTexture.active = bacRenTex;
}

#endregion



#region Textures

 //public MatrixSet origTexMaps;
 public MatrixSet scaledTexMaps;

 //private MatrixSet partialScaledTexMaps; //for apply
 //private MatrixSet partialOrigTexMaps;
 //private const int partialScaleMargins = 5; //rescaling partial rect if it's more or less rect+margins

 public void LoadTextures ()
 /// Reads whole terrain textures and resizes them to fit current Rect
 {
  TerrainData terrainData =  terrain.terrainData;
  int splatRes = terrainData.alphamapResolution;
```

```
//loading
TerrainLayer[] layers = terrainData.terrainLayers;
float[,,] splats = terrainData.GetAlphamaps(0, 0, splatRes, splatRes);

CoordRect origSplatRect = new CoordRect(rect.offset.x, rect.offset.z, splatRes, splatRes);
MatrixSet origTexMaps = new MatrixSet(origSplatRect, worldPos, worldSize);
for (int p=0; p<layers.Length; p++)
{
 MatrixSet.Prototype prototype = new MatrixSet.Prototype(layers, p);
 if (prototype.Object == null)
  continue;

 if (!origTexMaps.TryGetValue(prototype, out Matrix matrix)) //adding if not contains
 {
  matrix = new Matrix(origSplatRect);
  origTexMaps[prototype] = matrix;
 }

 matrix.ImportSplats(splats, p);
}

//scaling
if (splatRes != rect.size.x  ||  splatRes != rect.size.z)
{
 scaledTexMaps = new MatrixSet(rect, worldPos, worldSize);
 MatrixSet.Resize(origTexMaps, scaledTexMaps, interpolate:true);
```

```
 }
 else
  scaledTexMaps = origTexMaps;
}



public void ReadTextures (MatrixSet matrixSet)  =>  MatrixSet.CopyIntersected(scaledTexMaps, matrixS
/// Fills matrixSet with loaded terrain data

public void WriteTextures (MatrixSet matrixSet)  =>  MatrixSet.CopyIntersected(matrixSet, scaledTexMap
/// Copies matrix data to loaded whole terrain data and marks changed rect



private static MatrixSet splatsApply;
private static float[,,] splats;

private void ApplyTextures (CoordRect applyRect)
/// Copies pixels within given applyRect from heights matrix to terrain on Apply Changes
{
 TerrainData terrainData =  terrain.terrainData;

 if (terrainData.alphamapLayers == 0)
 {
  if (scaledTexMaps.Count == 0) return;
  FillAll( MatrixSet.Prototype.NewLayer<TerrainLayer>(scaledTexMaps.GetPrototypeByNum(0)) );
  return;
```

```
}

int heightRes = terrainData.heightmapResolution;
int splatsRes = terrainData.alphamapResolution;

CoordRect relApplyRect = new CoordRect(applyRect.offset - rect.offset, applyRect.size); //from here and

//applyRect in splat coordinates
CoordRect splatsApplyRect = new CoordRect();
float scaleRatio = 1f*(splatsRes-1)/(heightRes-1); //using non-height alignedratio since interpolating NN a
splatsApplyRect.Min = Coord.Floor(relApplyRect.offset.x*scaleRatio, relApplyRect.offset.z*scaleRatio);
splatsApplyRect.Max = Coord.Ceil(relApplyRect.Max.x*scaleRatio, relApplyRect.Max.z*scaleRatio);

//MatrixSet splatsApply = new MatrixSet(splatsApplyRect, Vector3.zero, Vector3.zero);
//trying to re-use splatsApply to avoid constantly creating it
//on average creates new matrix 1 times of 10
if (splatsApply == null  ||  splatsApply.Count != scaledTexMaps.Count ||
 splatsApply.rect.size.x < splatsApplyRect.size.x  ||  splatsApply.rect.size.x > splatsApplyRect.size.x+3  |
 splatsApply.rect.size.z < splatsApplyRect.size.z  ||  splatsApply.rect.size.z > splatsApplyRect.size.z+3)
  splatsApply = new MatrixSet(splatsApplyRect, Vector3.zero, Vector3.zero);
splatsApply.SetOffset(splatsApplyRect.offset);

//resizing from scaledTexMaps to splatsApply
//using (Log.Group("CopyResized size:" + splatsApply.rect.size + " count:" + splatsApply.Count))
MatrixSet.CopyResized(src:scaledTexMaps, dst:splatsApply,
 srcRectPos: (Vector2D)splatsApply.rect.offset / scaleRatio + (Vector2D)scaledTexMaps.rect.offset, //sp
```

```
 srcRectSize: (Vector2D)splatsApply.rect.size / scaleRatio,
 dstRectPos: splatsApply.rect.offset,
 dstRectSize: splatsApply.rect.size);

//adding to terrain layers prototypes prototypes that are in set, but not yet present on terrain
TerrainLayer[] originalTerrainLayers = terrainData.terrainLayers;

TerrainLayer[] terrainLayers = originalTerrainLayers;
foreach (MatrixSet.Prototype prototype in splatsApply.Prototypes)
 terrainLayers = prototype.CheckAppendLayers(terrainLayers);

if (terrainLayers != originalTerrainLayers)
 terrain.terrainData.terrainLayers = terrainLayers;

//finding splats intersected rect
CoordRect intersection = CoordRect.Intersected(new CoordRect(0,0,splatsRes,splatsRes), splatsApply.
if (intersection.size.x<=0 || intersection.size.z<=0) return;

//trying to re-use 3d array
//float[,,] splats = new float[intersection.size.z, intersection.size.x, terrainLayers.Length]; //x and z sizes a
if (splats == null  ||  splats.GetLength(2) != terrainLayers.Length  ||
 splats.GetLength(0) != intersection.size.z  ||  splats.GetLength(1) != intersection.size.x)
  splats = new float[intersection.size.z, intersection.size.x, terrainLayers.Length];

//setting
using (Log.Group("Exporting"))
```

```
 for (int p=0; p<terrainLayers.Length; p++)
 {
  MatrixSet.Prototype prototype = new MatrixSet.Prototype(terrainLayers, p);

  Matrix matrix;
  bool isInSet = splatsApply.TryGetValue(prototype, out matrix);

  if (!isInSet) //clearing channel if it has no matrix layer in output
   ClearSplats(splatsApply.rect, splats, p);

  else
   matrix.ExportSplats(splats, intersection.offset, p);
 }

using (Log.Group("Setting x:" + splats.GetLength(0) + " z:" + splats.GetLength(1) + " count:" + splats.Ge
{
 Vector2D relativeOffset = (Vector2D)relApplyRect.offset / heightRes;
 Coord splatOffset = Coord.Round(relativeOffset * splatsRes);
 terrainData.SetAlphamaps(intersection.offset.x, intersection.offset.z, splats);
}

//re-importing cached matrixset on texture layers change
if (terrainLayers != originalTerrainLayers)
 LoadTextures();
}
```

```
private static void ClearSplats (CoordRect matrixRect, float[,] splats, int channel) => ClearSplats(matrixR

private static void ClearSplats (CoordRect matrixRect, float[,] splats, Coord splatsOffset, int channel)

///Same as Matrix.ExportSplats(ch), but just clearing one splats channel

{

 Coord splatsSize = new Coord(splats.GetLength(1), splats.GetLength(0));  //x and z swapped

 CoordRect splatsRect = new CoordRect(splatsOffset, splatsSize);


 CoordRect intersection = CoordRect.Intersected(matrixRect, splatsRect);

 Coord min = intersection.Min; Coord max = intersection.Max;


 for (int x=min.x; x<max.x; x++)

  for (int z=min.z; z<max.z; z++)

  {

   int matrixPos = (z-matrixRect.offset.z)*matrixRect.size.x + x - matrixRect.offset.x;

   int heightsPosZ = x - splatsRect.offset.x;

   int heightsPosX = z - splatsRect.offset.z;


   splats[heightsPosX, heightsPosZ, channel] = 0;

  }

}



private void FillAll (TerrainLayer layer)

{

 TerrainData terrainData =  terrain.terrainData;
```

```
 int heightRes = terrainData.heightmapResolution;
 terrainData.alphamapResolution = heightRes;

 float[,,] splats = new float[heightRes, heightRes, 1];
 for (int x=0; x<heightRes; x++)
  for (int z=0; z<heightRes; z++)
   splats[x,z,0] = 1;

 terrainData.terrainLayers = new TerrainLayer[] {layer};
 terrainData.SetAlphamaps(0,0,splats);

 LoadTextures(); //since changed layers
}

#endregion


#region Grass

 //public MatrixSet origGrassMaps;
 public MatrixSet scaledGrassMaps;

 //private MatrixSet partialScaledTexMaps; //for apply
 //private MatrixSet partialOrigTexMaps;
 //private const int partialScaleMargins = 5; //rescaling partial rect if it's more or less rect+margins
```

```
public void LoadGrass ()
/// Reads whole terrain textures and resizes them to fit current Rect
{
 TerrainData terrainData =  terrain.terrainData;
 int grassRes = terrainData.detailResolution;

 //loading
 DetailPrototype[] detLayers = terrainData.detailPrototypes;
 UnityEngine.Object[] textures = detLayers.Select(p => p.Object());

 CoordRect origGrassRect = new CoordRect(rect.offset.x, rect.offset.z, grassRes, grassRes);
 MatrixSet origGrassMaps = new MatrixSet(origGrassRect, worldPos, worldSize);

 for (int p=0; p<textures.Length; p++)
 {
  MatrixSet.Prototype prototype = new MatrixSet.Prototype(textures, p);

  if (!origGrassMaps.TryGetValue(prototype, out Matrix matrix)) //adding if not contains
  {
   matrix = new Matrix(origGrassRect);
   origGrassMaps[prototype] = matrix;
  }

  int[,] layer = terrainData.GetDetailLayer(0,0,grassRes,grassRes, p);
  matrix.ImportDetail(layer, density:origGrassMaps.PixelSize.x*origGrassMaps.PixelSize.z);
  //matrix will have values > 1
```

```
 }

 //scaling
 if (grassRes != rect.size.x  ||  grassRes != rect.size.z)
 {
  scaledGrassMaps = new MatrixSet(rect, worldPos, worldSize);
  MatrixSet.Resize(origGrassMaps, scaledGrassMaps, interpolate:false); //interpolated resize will clamp v
 }
 else
  scaledGrassMaps = origGrassMaps;
}



public void ReadGrass (MatrixSet matrixSet)  =>  MatrixSet.CopyIntersected(scaledGrassMaps, matrixS
/// Fills matrixSet with loaded terrain data

public void WriteGrass (MatrixSet matrixSet)  =>  MatrixSet.CopyIntersected(matrixSet, scaledGrassMap
/// Copies matrix data to loaded whole terrain data and marks changed rect



private static MatrixSet grassApply;
private static int[,] grass;
private static Noise grassNoise;

private void ApplyGrass (CoordRect applyRect)
/// Copies pixels within given applyRect from heights matrix to terrain on Apply Changes
```

```
{
 TerrainData terrainData =  terrain.terrainData;
 int heightRes = terrainData.heightmapResolution;
 int grassRes = terrainData.detailResolution;

 //adjusting grass res if grass is applied the first time
 if (terrainData.detailPrototypes.Length==0)  //no grass of any type added
 {
  grassRes = terrain.terrainData.heightmapResolution;
  terrain.terrainData.SetDetailResolution(terrain.terrainData.heightmapResolution, 16);
 }

 CoordRect relApplyRect = new CoordRect(applyRect.offset - rect.offset, applyRect.size); //from here and

 //applyRect in grass coordinates
 CoordRect grassApplyRect = new CoordRect();
 float scaleRatio = 1f*(grassRes-1)/(heightRes-1); //using non-height alignedratio since interpolating NN a
 grassApplyRect.Min = Coord.Floor(relApplyRect.offset.x*scaleRatio, relApplyRect.offset.z*scaleRatio);
 grassApplyRect.Max = Coord.Ceil(relApplyRect.Max.x*scaleRatio, relApplyRect.Max.z*scaleRatio);

 //MatrixSet grassApply = new MatrixSet(grassApplyRect, Vector3.zero, Vector3.zero);
 //trying to re-use grassApply to avoid constantly creating it
 //on average creates new matrix 1 times of 10
 if (grassApply == null  ||  grassApply.Count != scaledGrassMaps.Count ||
  grassApply.rect.size.x < grassApplyRect.size.x  ||  grassApply.rect.size.x > grassApplyRect.size.x+3  ||
  grassApply.rect.size.z < grassApplyRect.size.z  ||  grassApply.rect.size.z > grassApplyRect.size.z+3)
```

```
  grassApply = new MatrixSet(grassApplyRect, Vector3.zero, Vector3.zero);
grassApply.SetOffset(grassApplyRect.offset);

//resizing from scaledGrassMaps to grassApply
MatrixSet.CopyResized(src:scaledGrassMaps, dst:grassApply,
 srcRectPos: (Vector2D)grassApply.rect.offset / scaleRatio + (Vector2D)scaledGrassMaps.rect.offset, //g
 srcRectSize: (Vector2D)grassApply.rect.size / scaleRatio,
 dstRectPos: grassApply.rect.offset,
 dstRectSize: grassApply.rect.size);

//adding to terrain layers prototypes prototypes that are in set, but not yet present on terrain
DetailPrototype[] originalDetLayers = terrainData.detailPrototypes;
DetailPrototype[] detLayers = originalDetLayers;
foreach (MatrixSet.Prototype prototype in grassApply.Prototypes)
 detLayers = prototype.CheckAppendLayers(detLayers);

if (originalDetLayers != detLayers)
 terrain.terrainData.detailPrototypes = detLayers;

//finding grasss intersected rect
CoordRect intersection = CoordRect.Intersected(new CoordRect(0,0,grassRes,grassRes), grassApply.re
if (intersection.size.x<=0 || intersection.size.z<=0) return;

//trying to re-use array
if (grass == null  ||  grass.GetLength(0) != intersection.size.z  ||  grass.GetLength(1) != intersection.size.x
 grass = new int[intersection.size.z, intersection.size.x];
```

```
if (grassNoise == null)
 grassNoise = new Noise(123);

//setting
for (int p=0; p<detLayers.Length; p++)
{
 MatrixSet.Prototype prototype = new MatrixSet.Prototype(detLayers, p);

 Matrix matrix;
 bool isInSet = grassApply.TryGetValue(prototype, out matrix);

 if (!isInSet) //clearing channel if it has no matrix layer in output
 {
  for (int x=0; x<intersection.size.x; x++)
   for (int z=0; z<intersection.size.z; z++)
    grass[z,x] = 0;
 }

 else
 {
  float grassPixelSize = terrainData.size.x / (grassRes-1);
  matrix.ExportDetail(grass, intersection.offset, p, grassNoise, density:grassPixelSize*grassPixelSize);
 }

 terrain.terrainData.SetDetailLayer(intersection.offset.x, intersection.offset.z, p, grass);
```

```
    }
   }


  #endregion
 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Products;
using MapMagic.Expose;
using System.Threading;
using System;

namespace MapMagic.Brush
{
 public static class TerrainManager
 {

  public static bool TryAddTerrain (ref Terrain[] brushTerrains, Terrain terrain, out string error)
  /// Checks if terrain resolution matches brush terrains resolutions and add it if it does
  /// Or returns false and outs error if not
  {
   //first terrain
   if (brushTerrains.Length==0)
   {
```

```
 error =null;
 brushTerrains = new Terrain[] {terrain};
 return true;
}

//additional terrain
else
{
 if (brushTerrains.Contains(terrain))
  { error=null; return true; }

 if (CheckTerrain(brushTerrains[0], terrain, out string terrError))
 {
  ArrayTools.Add(ref brushTerrains, terrain);

  error = null;
  return true;
 }
 else
 {
  error = $"Could not add terrain {terrain.name}:\n\n{terrError}";
  return false;
 }
}
}
```

```
public static bool TryAddTerrains (ref Terrain[] brushTerrains, Terrain[] newTerrains, out string error)
/// Tries to add new terrains one by one
/// Skips if their resolution doesn't match brush terrains, will return false and give summary error in this cas
{
 Terrain refTerrain;
 if (brushTerrains.Length==0)
  refTerrain = newTerrains.FindBiggest(t => t.terrainData.heightmapResolution); //finds the terrain with big
 else
  refTerrain = brushTerrains[0]; //takes any brush terrain

 bool check = true;
 error = null;

 List<Terrain> addingTerrains = new List<Terrain>();
 foreach (Terrain terrain in newTerrains)
 {
  if (brushTerrains.Contains(terrain))
   continue;

  if (CheckTerrain(refTerrain, terrain, out string terrError))
   addingTerrains.Add(terrain);
  else
  {
   check = false;
```

```
   if (error == null)
    error = "Skipping adding these terrains:";

   error += $"\n\n{terrain.name}: {terrError}";
  }
 }

 ArrayTools.AddRange(ref brushTerrains, addingTerrains.ToArray());
 return check;
}

public static void ExcludeNullTerrains (ref Terrain[] brushTerrains)
/// Ensures that all brush terrains have the same size/resolution
/// Exclude terrains that does not and returns false
{
 if (brushTerrains.Length == 0)
  return;

 //excluding null terrains
 for (int i=brushTerrains.Length-1; i>=0; i--)
  if (brushTerrains[i] == null)
   ArrayTools.RemoveAt(ref brushTerrains, i);
}

public static bool ExcludeImproperTerrains (ref Terrain[] brushTerrains, out string error)
```

```
/// Ensures that all brush terrains have the same size/resolution
/// Exclude terrains that does not and returns false
{
 if (brushTerrains.Length <= 1)
  { error = null; return true; }

 bool check = true;
 error = null;

 Terrain refTerrain = brushTerrains[0];
 for (int i=brushTerrains.Length-1; i>=0; i--)
 {
  Terrain terrain = brushTerrains[i];
  bool checkTerrain = CheckTerrain(refTerrain, terrain, out string terrError);
  if (!checkTerrain)
  {
   check = false;
   ArrayTools.RemoveAt(ref brushTerrains, i);

   if (error == null)
    error = "MapMagic Brush terrains list resolution or size does not match:";

   error += $"\n\n{terrain.name}: {terrError}";
  }
 }
```

```
 if (!check)
  error += "\n\nExcluding these terrain from brush list. You can add them manually after changing the size

 return check;
}


private static bool CheckExcludeTerrains (Terrain refTerrain, ref Terrain[] newTerrains, out string terrError
/// Excluding terrains from newTerrains which resolution/size is not matching ref terrain
/// False if any of terrains was excluded
{
 terrError = null;
 bool check = true;

 for (int i=newTerrains.Length; i>=0; i--)
 {
  Terrain terrain = newTerrains[i];
  bool checkTerrain = CheckTerrain(refTerrain, terrain, out string checkTerrainError);
  if (!checkTerrain)
  {
   check = false;
   ArrayTools.RemoveAt(ref newTerrains, i);

   if (terrError == null)
    terrError = checkTerrainError;
   else
```

```
    terrError += "\n\n" + checkTerrainError;
   }
  }

 return check;
}


private static bool CheckTerrain (Terrain refTerrain, Terrain newTerrain, out string terrError)
/// Checks if size/resolution/etc match the reference terrain
{
 TerrainData refData = refTerrain.terrainData;
 TerrainData newData = newTerrain.terrainData;
 terrError = null;

 int refDetRes = refData.detailResolution;
 int newDetRes = newData.detailResolution;
 if (refDetRes!=newDetRes  &&  refDetRes!=newDetRes-1  &&  refDetRes!=newDetRes+1)
  terrError = $"Terrains detail resolution don't match: {refTerrain.name}:{refData.detailResolution} and {new

 int refAlphaRes = refData.alphamapResolution;
 int newAlphaRes = newData.alphamapResolution;
 if (refAlphaRes!=newAlphaRes  &&  refAlphaRes!=newAlphaRes-1  &&  refAlphaRes!=newAlphaRes+1)
  terrError = $"Terrains alpha maps (splats textures) resolution don't match: {refTerrain.name}:{refData.alp

 if (refData.heightmapResolution != newData.heightmapResolution)
```

```
  terrError = $"Terrains heighmap resolution don't match: {refTerrain.name}:{refData.heightmapResolution

 Vector3 refSize = refData.size;
 Vector3 newSize = newData.size;
 if (newSize.x < refSize.x-0.001f  ||  newSize.x > refSize.x+0.001f  ||
  newSize.z < refSize.z-0.001f  ||  newSize.z > refSize.z+0.001f)
   terrError = $"Terrains size don't match: {refTerrain.name}:{refSize.x},{refSize.z} and {newTerrain.name}

 if (terrError == null)
  return true;
 else
  return false;
}



public static bool CheckSplatResolution (Terrain terrain)
/// True if terrain splat resolution is PO2 + 1
{
 int terrainRes = terrain.terrainData.alphamapResolution;
 int potRes = Mathf.ClosestPowerOfTwo(terrainRes);
 return potRes+1 == terrainRes;
}

public const string splatResError = "Terrain splat resolution does not match heightmap resolution formula
```

```
public static void FixSplatResolution (Terrain terrain)
/// Rescales splat resolution so it is PO2 + 1
{
 TerrainData data = terrain.terrainData;
 int shouldBeRes = Mathf.ClosestPowerOfTwo(data.alphamapResolution) + 1;
 RescaleSplatResolution(data,shouldBeRes);
}



private static void RescaleSplatResolution (TerrainData data, int dstRes)
{
 int numLayers = data.alphamapLayers;
 int srcRes = data.alphamapResolution;
 if (srcRes == dstRes)
  return;

 float[,,] srcSplats = data.GetAlphamaps(0,0,srcRes,srcRes);
 Matrix[] dstMatrices = new Matrix[numLayers];
 for (int i=0; i<numLayers; i++)
 {
  Matrix srcMatrix = new Matrix(0,0,srcRes,srcRes);
  srcMatrix.ImportSplats(srcSplats,i);

  Matrix dstMatrix = new Matrix(0,0,dstRes,dstRes);
  //MatrixOps.Resize(srcMatrix, dstMatrix);
  Matrix.Resize(srcMatrix, dstMatrix); //this one will use NN interpolation, which is better when scaling 1 pi
```

```
  dstMatrices[i] = dstMatrix;
 }

 Matrix.NormalizeLayers(dstMatrices);

 float[,,] dstSplats = new float[dstRes,dstRes,numLayers];
 for (int i=0; i<numLayers; i++)
  dstMatrices[i].ExportSplats(dstSplats, i);

 data.alphamapResolution = dstRes;
 data.SetAlphamaps(0,0,dstSplats);
}


public static Terrain[] GetTerrains (Terrain[] allTerrains, Vector2D worldPos, Vector2D worldSize)
/// Gathers the array of terrains that are affected by this rect
/// Closest terrain goes first [0]
{
 if (allTerrains.Length == 0)
   return new Terrain[0];

 List<Terrain> stampTerrains = new List<Terrain>();

 Vector2D stampMin = worldPos;
 Vector2D stampMax = worldPos+worldSize;
```

```
Vector2D stampCenter = worldPos+worldSize/2;

float closestDist = float.MaxValue;
Terrain closestTerrain = null;

foreach (Terrain terrain in allTerrains)
{
 Vector3 terrainPos = terrain.transform.position;
 Vector3 terrainSize = terrain.terrainData.size;
 Vector3 terrainMax = terrainPos+terrainSize;

 if (stampMax.x < terrainPos.x  ||  stampMin.x > terrainMax.x  ||
  stampMax.z < terrainPos.z  ||  stampMin.z > terrainMax.z)
   continue;

 stampTerrains.Add(terrain);

 //if stamp is on this terrain - it should be 100% closest
 if (stampCenter.x > terrainPos.x  &&  stampCenter.x < terrainMax.x  &&
  stampCenter.z > terrainPos.z  &&  stampCenter.z < terrainMax.z)
 {
  closestTerrain = terrain;
  closestDist = 0;
 }

 //finding other closest
```

```
    else
    {
     Vector2D pDist = (Vector2D)terrainPos - stampCenter;
     Vector2D sDist = stampCenter - (Vector2D)terrainMax;
     float dist = Mathf.Max( Mathf.Max(pDist.x, pDist.z), Mathf.Max(sDist.x, sDist.z) );

     if (dist < closestDist)
      { closestDist=dist; closestTerrain=terrain; }
    }
   }

   //making closest terrain first in array
   int closestIndex = stampTerrains.IndexOf(closestTerrain);
   if (closestIndex != 0  &&  stampTerrains.Count != 0)
   {
    stampTerrains[closestIndex] = stampTerrains[0];
    stampTerrains[0] = closestTerrain;
   }

   //TODO: excluding terrains that have improper resolutions

   return stampTerrains.ToArray();
  }
 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Products;
using MapMagic.Expose;
using System.Threading;
using System;

namespace MapMagic.Brush
{
 public class Trace
 /// Performs the proper brush drawing on terrain with Start, Drag and Release
 /// Periodically stamps the brush on terrain using it's spacing
 {
  [NonSerialized] public bool isDrawing = false;
  [NonSerialized] public List<Vector3> framePoses = new List<Vector3>(); //stores position each frame to a
  [NonSerialized] public float length = 0; //current distance of the spacing line, to avoid recalculating it each
  [NonSerialized] public List<Vector3> stampPoses = new List<Vector3>();
  [NonSerialized] public Vector3 capturedPosition; //world position used for Constant or other presets
```

```
  public Vector3 LastPos => framePoses[framePoses.Count-1];
  public Vector3 LastStampPos => stampPoses[stampPoses.Count-1]; //position of last applid stamp
  public Vector3 PrevStampPos => stampPoses.Count>1 ? stampPoses[stampPoses.Count-2] : stampPos

  public Vector2D ApproxNextStep (float spacingDist)
  /// Approximate next step position to draw dashed line in scene view
  {
   Vector2D realDir = ((Vector2D)LastPos - (Vector2D)LastStampPos).Normalized; //next position based on
   Vector2D prevDir = stampPoses.Count >= 2 ?
    ((Vector2D)LastStampPos - (Vector2D)stampPoses[stampPoses.Count-2]).Normalized :
    realDir;

//   float percent = lastLength / spacingDist; //blending dirs based on distance passed
//   Vector2D dir = realDir*percent + prevDir*(1-percent);

//   return (Vector2D)LastPos + dir*(spacingDist-lastLength);
   return (new Vector2D());
  }


  public void Start (Vector3 pos, Action<Vector3,bool> stampFn, MapMagicBrush brush)
  {
   #if UNITY_EDITOR
   if (Event.current.control)
    { capturedPosition = pos; return; }
   #endif
```

```
 isDrawing = true;
 framePoses.Add(pos);
 stampPoses.Add(pos);
 stampFn(pos,true); //StampBrush(pos);

 //applying changes to terrain on first click
 foreach (Terrain terrain in brush.terrains)
 {
  TerrainCache terrainCache = brush.terrainCaches[terrain];
  terrainCache.ApplyChanges(brush.cacheChange);
 }

 brush.cacheChange.Clear(); //after each apply
}


public void Drag (Vector3 pos, float spacingDist, Action<Vector3,bool> stampFn, MapMagicBrush brush)
/// Checks if brush passed the spacing distance and applies it if needed (multiple times if really needed)
/// Called each frame when mouse button is pressed
/// spacingDist = radius*spacing
{
 if (!isDrawing)
  return;

 //saving previous data
```

```
float prevLength = length;
Vector3 prevPos = framePoses[framePoses.Count-1];
Vector3 moveDir = pos - prevPos;  moveDir.y = 0;
float segDist = moveDir.magnitude; //Mathf.Sqrt((pos.x-prevPos.x)*(pos.x-prevPos.x) + (pos.z-prevPos.z)
moveDir.Normalize();

//adding point to spacing poses list and update it's length
framePoses.Add(pos);
length += segDist;

//do need to apply stamp this frame?
int stampsShouldBeApplied = (int)(length/spacingDist) + 1;
int stampsApplied = stampPoses.Count;
int numStamps = stampsShouldBeApplied-stampsApplied;

//applying stamps if needed
float prevPassedLength = prevLength - stampsApplied*spacingDist; //distance since last stamp to last fra
for (int i=0; i<numStamps; i++)
{
 float currStampLength = i*spacingDist - prevPassedLength; //at what distance to prevPos should the sta

 Vector3 stampPos = prevPos + moveDir*currStampLength;
 stampPoses.Add(stampPos);
 stampFn(stampPos,false); //StampBrush(stampPos);
}
```

```
  //applying changes to terrain
  foreach (Terrain terrain in brush.terrains)
  {
   TerrainCache terrainCache = brush.terrainCaches[terrain];
   terrainCache.ApplyChanges(brush.cacheChange);
  }
  brush.cacheChange.Clear(); //after each apply
 }



 public void Release (MapMagicBrush brush)
 /// Called each frame when mouse button is not pressed
 /// Clears spacing path. Not included in StrokeSpacingBrush to make these two fn more readable
 {
  isDrawing = false;
  framePoses.Clear();
  stampPoses.Clear();
  length = 0;

  foreach (Terrain terrain in brush.terrains)
  {
   TerrainCache terrainCache = brush.terrainCaches[terrain];
   terrainCache.EndTrace();
  }
 }
}
```

```
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Products;
using System.Threading;
using System;

namespace MapMagic.Brush
{
 [System.Serializable]
 public class TuneStroke
 {
  public MapMagicBrush brush;

  public bool drawTune;

  public Stamp[] stamps = new Stamp[0];


  public void StampTune (Vector2D pos)
```

```
  {
   /*BrushStamp stamp = new BrushStamp() {pos=pos, radius=brush.radius, hardness=brush.hardness, ma

   Terrain closestTerrain = MapMagicBrush.GetClosestTerrain(brush.allTerrains, stamp.pos);
   TileData tileData = MapMagicBrush.GetTileData(closestTerrain, stamp, brush.graph);
   ReadData readData = new ReadData();

   foreach (IBrushRead brushReadNode in brush.graph.GeneratorsOfType<IBrushRead>())
    brushReadNode.ReadTerrains(brush.opTerrains, stamp, readData, tileData);

   readData.terrains = brush.opTerrains;

   ArrayTools.Add(ref stamps, stamp);
   ArrayTools.Add(ref originalReads, readData);*/
  }


  public void ApplyTune ()
  {
/*   //erasing previous stuff
   for (int s=0; s<stamps.Length; s++)
    originalReads[s].Apply();

   //applying new
   for (int s=0; s<stamps.Length; s++)
    brush.StampBrush(stamps[s]);*/
```

```
}



public void Clear ()
{
 /*for (int s=0; s<stamps.Length; s++)
  originalReads[s].Apply();

 stamps = new Stamp[0];
 originalReads = new ReadData[0];*/
}



public (Vector2D,float) GuiCircle ()
{
 Vector2D min = new Vector2D(float.MaxValue, float.MaxValue);
 Vector2D max = new Vector2D(float.MinValue, float.MinValue);

 for (int s=0; s<stamps.Length; s++)
 {
  Stamp stamp = stamps[s];
  if (stamp.pos.x < min.x) min.x = stamp.pos.x;
  if (stamp.pos.z < min.z) min.z = stamp.pos.z;
  if (stamp.pos.x > max.x) max.x = stamp.pos.x;
  if (stamp.pos.z > max.z) max.z = stamp.pos.z;
 }
```

```
    Vector2D center = (min+max) / 2;
    Vector2D size = (max-min);
    float radius = Mathf.Max(size.x, size.z) / 2;

    return (center,radius + stamps[0].radius);
   }
  }
 }
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Products;
using MapMagic.Expose;
using System.Threading;
using System;

[assembly: System.Runtime.CompilerServices.InternalsVisibleTo("MapMagic.Tests.Editor")]

namespace MapMagic.Brush.Undo
{
 public class TerrainUndoData
 {
  const int numChunksPerTerrain = 8;

  Matrix2D<float[,]> heightsChunks;
  Matrix2D<float[,,]> splatsChunks;
  TerrainLayer[] splatPrototypes;
  Matrix2D<int[][,]> grassChunks;
```

```
DetailPrototype[] grassPrototypes;
TreeInstance[] treeInstances;
TreePrototype[] treePrototypes;
//objects changes are recordered via UnityEditor.Undo in outputs

public void Read (Terrain terrain, Vector2D worldPos, Vector2D worldSize,
 bool readHeight, bool readSplats, bool readGrass, bool readTrees)
{
 TerrainData terrainData = terrain.terrainData;

 Vector2D terrainPos = (Vector2D)terrain.transform.position;
 Vector2D terrainSize = (Vector2D)terrain.terrainData.size;

 if (!Vector2D.Intersects(worldPos, worldSize, terrainPos, terrainSize))
  return;

 Coord chunksMin = (Coord)((worldPos-terrainPos) / terrainSize * numChunksPerTerrain);
 Coord chunksMax = (Coord)((worldPos+worldSize-terrainPos) / terrainSize * numChunksPerTerrain + 1)
 chunksMin.ClampByRect(new CoordRect(0,0,numChunksPerTerrain,numChunksPerTerrain));
 chunksMax.ClampByRect(new CoordRect(0,0,numChunksPerTerrain+1,numChunksPerTerrain+1));

 for (int x=chunksMin.x; x<chunksMax.x; x++)
  for (int z=chunksMin.z; z<chunksMax.z; z++)
  {
   Coord coord = new Coord(x,z);
```

```
if (readHeight  &&  heightsChunks?[x,z] == null)
{
 if (heightsChunks==null) heightsChunks = new Matrix2D<float[,]>(numChunksPerTerrain, numChunksP
 CoordRect pixRect = CoordToPixels(coord, terrainData.heightmapResolution, numChunksPerTerrain);
 float[,] heightData = terrainData.GetHeights(pixRect.offset.x, pixRect.offset.z, pixRect.size.x, pixRect.si
 heightsChunks[x,z] = heightData;
}

if (readSplats  &&  splatPrototypes == null)
 splatPrototypes = terrainData.terrainLayers;

if (readSplats  &&  splatsChunks?[x,z] == null)
{
 if (splatsChunks==null) splatsChunks = new Matrix2D<float[,,]>(numChunksPerTerrain, numChunksPe
 CoordRect pixRect = CoordToPixels(coord, terrainData.alphamapResolution, numChunksPerTerrain);
 float[,,] splatsData = terrainData.GetAlphamaps(pixRect.offset.x, pixRect.offset.z, pixRect.size.x, pixRe
 splatsChunks[x,z] = splatsData;
}

if (readGrass  &&  grassPrototypes == null)
 grassPrototypes = terrainData.detailPrototypes;

if (readGrass  &&  grassChunks?[x,z] == null)
{
 if (grassChunks==null) grassChunks = new Matrix2D<int[][,]>(numChunksPerTerrain, numChunksPerT
 CoordRect pixRect = CoordToPixels(coord, terrainData.detailResolution, numChunksPerTerrain);
```

```
    int layersCount = grassPrototypes.Length; //grassPrototypes already assigned at this moment
    int[][,] grassData = new int[layersCount][,];
    for (int i=0; i<layersCount; i++)
     grassData[i] = terrainData.GetDetailLayer(pixRect.offset.x, pixRect.offset.z, pixRect.size.x, pixRect.siz

    grassChunks[x,z] = grassData;
   }

   if (readTrees  &&  treeInstances == null)
    treeInstances = terrainData.treeInstances;

   if (readTrees  &&  treePrototypes == null)
    treePrototypes = terrainData.treePrototypes;
  }
}


public void Write (Terrain terrain)
{
 TerrainData terrainData = terrain.terrainData;

 if (splatPrototypes != null)
  terrainData.terrainLayers = splatPrototypes;

 if (grassPrototypes != null)
```

```
 terrainData.detailPrototypes = grassPrototypes;


for (int x=0; x<numChunksPerTerrain; x++)
 for (int z=0; z<numChunksPerTerrain; z++)
 {
  Coord coord = new Coord(x,z);

  if (heightsChunks != null  &&  heightsChunks[x,z] != null)
  {
   CoordRect pixRect = CoordToPixels(coord, terrainData.heightmapResolution, numChunksPerTerrain);
   terrainData.SetHeights(pixRect.offset.x, pixRect.offset.z, heightsChunks[x,z]);
  }

  if (splatsChunks != null  &&  splatsChunks[x,z] != null)
  {
   CoordRect pixRect = CoordToPixels(coord, terrainData.alphamapResolution, numChunksPerTerrain);
   terrainData.SetAlphamaps(pixRect.offset.x, pixRect.offset.z, splatsChunks[x,z]);
  }

  if (grassChunks != null  &&  grassChunks[x,z] != null)
  {
   CoordRect pixRect = CoordToPixels(coord, terrainData.detailResolution, numChunksPerTerrain);
   int[][,] grassData = grassChunks[x,z];
   for (int i=0; i<grassData.Length; i++)
    terrainData.SetDetailLayer(pixRect.offset.x, pixRect.offset.z, i,  grassData[i]);
  }
```

```
    }


  if (treePrototypes != null)
    terrainData.treePrototypes = treePrototypes;


  if (treeInstances != null)
    terrainData.treeInstances = treeInstances;
 }



 internal static CoordRect CoordToPixels (Coord coord, int resolution, int numChunks)
 //Converts chunk coord to pixel rect
 //Note that we don't need exact number of pixels in each chunk, come of them might have 64 pixels, some
 //Tested (MapMagicTester)
 {
  Vector2D minPixelFloat = (1f * resolution / numChunks) * (Vector2D)coord;
  Coord minPixel = (Coord)minPixelFloat;

  Vector2D maxPixelFloat = (1f * resolution / numChunks) * (Vector2D)(coord+1);
  Coord maxPixel = (Coord)maxPixelFloat;
  if (coord.x == numChunks-1) maxPixel.x = resolution;
  if (coord.z == numChunks-1) maxPixel.z = resolution;

  return new CoordRect(minPixel, maxPixel-minPixel);
 }
}
```

```
public class Undo
{
 public const string undoName = "Brush Stroke";
 public string lastUndoName;  //the last undo group name (kept to know it on UndoRedoPerformed)

 //private Stack<Set> sets = new Stack<Set>();  //we've got to remove first items from it, so using list instea
 [SerializeField] private List< Dictionary<Terrain,TerrainUndoData> > sets = new List< Dictionary<Terrain,T

 Terrain testTesrrain = null;

 #if UNITY_EDITOR
 ///Calling Undo
 ///Clone of Tools.GUI.Undo, except it's working with brush rather than gui
  public Undo ()
  {
   UnityEditor.Undo.undoRedoPerformed -= OnUndoRedoPerformed;
   UnityEditor.Undo.undoRedoPerformed += OnUndoRedoPerformed;
  }

  public void OnUndoRedoPerformed ()
  {
   string currGroupName = UnityEditor.Undo.GetCurrentGroupName();

   if (currGroupName == undoName || currGroupName == lastUndoName)
```

```
  // a bit hacky here. On undoRedoPerformed there is already no current group in stack, and no way to ge
  // so we store previous (before mm change) name and performing undo if this name is first in stack
  // TODO: use undo from MapMagicBrush with ids, it's more stable
  {
   Perform();

   if (currGroupName == lastUndoName)
    lastUndoName = null;
  }
 }
#endif

public void NewGroup (MapMagicBrush brush)
///Registering new undo (at the start of each stroke)
{
 /*if (sets.Count > 10)
 {
  List<Set> last10 = new List<Set>(sets);
 }*/

 sets.Add( new Dictionary<Terrain,TerrainUndoData>() );

 #if UNITY_EDITOR
  string currGroupName = UnityEditor.Undo.GetCurrentGroupName();
  if (currGroupName != undoName)
   lastUndoName = currGroupName;
```

```
  UnityEditor.Undo.RecordObject(brush, undoName);
  brush.temp = !brush.temp;
 #endif
}


public void Append (Terrain[] terrains, Vector2D worldPos, Vector2D worldSize,
 bool readHeight=false, bool readSplats=false, bool readGrass=false, bool readTrees=false)
{
 Dictionary<Terrain,TerrainUndoData> topSet = sets[sets.Count-1];

 foreach (Terrain terrain in terrains)
 {
  Vector2D terrainPos = (Vector2D)terrain.transform.position;
  Vector2D terrainSize = (Vector2D)terrain.terrainData.size;
  if (!Vector2D.Intersects(worldPos, worldSize, terrainPos, terrainSize))
   continue;

  TerrainUndoData undoData;
  if (!topSet.TryGetValue(terrain, out undoData))
  {
   undoData = new TerrainUndoData();
   topSet.Add(terrain, undoData);
  }
```

```
  undoData.Read(terrain, worldPos, worldSize,
    readHeight, readSplats, readGrass, readTrees);
 }

 testTesrrain = terrains[0];
}



public void Perform ()
{
 if (sets.Count == 0)
  return; //when two Brush components are used in scene - one will have no undo stack (but undo will be c

 Dictionary<Terrain,TerrainUndoData> topSet = sets[sets.Count-1];
 sets.RemoveAfter(sets.Count-2);

 foreach (var kvp in topSet)
 {
  Terrain terrain = kvp.Key;
  TerrainUndoData undoData = kvp.Value;

  undoData.Write(terrain);
 }

 foreach (MapMagicBrush brush in GameObject.FindObjectsOfType<MapMagicBrush>())
  brush.UpdateCaches();
```

```
 }
}



public class UndoBac
{
 private enum DataType { Height=0, Splats=1, Grass=2, Trees=3, Objects=4 };

 private class Set
 {
  private struct TerrainCoordType
  {
   public Terrain terrain;
   public Coord coord;
   public DataType type;

   public TerrainCoordType (Terrain terrain, Coord coord, DataType type)
   {
    this.terrain = terrain;
    this.coord = coord;
    this.type = type;
   }

   public override int GetHashCode () => terrain.GetHashCode() ^ (coord.GetHashCode() * 10 + (int)type);
  }
```

```
Dictionary<TerrainCoordType,object> dict = new Dictionary<TerrainCoordType, object>();


public object this[Terrain terrain, Coord coord, DataType type]
{
 get
 {
  TerrainCoordType tct = new TerrainCoordType(terrain, coord, type);

  if (dict.TryGetValue(tct, out object obj))
   return obj;
  else
   return null;
 }

 set
 {
  TerrainCoordType tct = new TerrainCoordType(terrain, coord, type);
  if (dict.ContainsKey(tct))
   dict[tct] = value;
  else
   dict.Add(tct, value);
 }
}

public IEnumerable<(Terrain,Coord,DataType,object)> Datas()
{
```

```
  foreach (var kvp in dict)
  {
   TerrainCoordType tct = kvp.Key;
   yield return (tct.terrain, tct.coord, tct.type, kvp.Value);
  }
 }
}
```

```
const int numChunksPerTerrain = 8; //number*number of chunks in terrain
const string undoName = "Brush Stroke";
public string lastUndoName;  //the last undo group name (kept to know it on UndoRedoPerformed)

//private Stack<Set> sets = new Stack<Set>();  //we've got to remove first items from it, so using list instea
[SerializeField] private List<Set> sets = new List<Set>();

#if UNITY_EDITOR
///Calling Undo
///Clone of Tools.GUI.Undo, except it's working with brush rather than gui
 public UndoBac ()
 {
  UnityEditor.Undo.undoRedoPerformed -= OnUndoRedoPerformed;
  UnityEditor.Undo.undoRedoPerformed += OnUndoRedoPerformed;
 }
```

```
 public void OnUndoRedoPerformed ()
 {
  string currGroupName = UnityEditor.Undo.GetCurrentGroupName();

  if (currGroupName == undoName || currGroupName == lastUndoName)
  // a bit hacky here. On undoRedoPerformed there is already no current group in stack, and no way to ge
  // so we store previous (before mm change) name and performing undo if this name is first in stack
  // TODO: use undo from MapMagicBrush with ids, it's more stable
  {
   Perform();

   if (currGroupName == lastUndoName)
    lastUndoName = null;
  }
 }
#endif

public void NewGroup (MapMagicBrush brush)
///Registering new undo (at the start of each stroke)
{
 /*if (sets.Count > 10)
 {
  List<Set> last10 = new List<Set>(sets);
 }*/

 Set newSet = new Set();
```

```
 sets.Add(newSet);

 #if UNITY_EDITOR
  string currGroupName = UnityEditor.Undo.GetCurrentGroupName();
  if (currGroupName != undoName)
   lastUndoName = currGroupName;

  UnityEditor.Undo.RecordObject(brush, undoName);
  brush.temp = !brush.temp;
 #endif
}



public void Append (Terrain[] terrains, Vector2D worldPos, Vector2D worldSize,
 bool readHeight=false, bool readSplats=false)
{
 foreach (Terrain terrain in terrains)
  using (Log.Group("Read All"))
 {
  TerrainData terrainData = terrain.terrainData;

  Vector2D terrainPos = (Vector2D)terrain.transform.position;
  Vector2D terrainSize = (Vector2D)terrain.terrainData.size;

  if (!Vector2D.Intersects(worldPos, worldSize, terrainPos, terrainSize))
   continue;
```

```
CoordRect intersectionChunks = terrain.PixelRect(worldPos, worldSize, numChunksPerTerrain);
Coord chunksMin = intersectionChunks.Min; Coord chunksMax = intersectionChunks.Max;

Set topSet = sets[sets.Count-1];

for (int x=chunksMin.x; x<chunksMax.x; x++)
 for (int z=chunksMin.z; z<chunksMax.z; z++)
 {
  Coord coord = new Coord(x,z);

  if (readHeight  &&  topSet[terrain, coord, DataType.Height] == null)
  {
   CoordRect heightPixels = CoordToPixels(coord, terrainData.heightmapResolution, numChunksPerTer
   float[,] heightData;
   using (Log.Group("Read heights"))
    heightData = terrainData.GetHeights(heightPixels.offset.x, heightPixels.offset.z, heightPixels.size.x, h
   //using (new Log.Timer("Read height texture"))
   // { RenderTexture htex = terrainData.heightmapTexture; }
   topSet[terrain, coord, DataType.Height] = heightData;
  }

  if (readSplats  &&  topSet[terrain, coord, DataType.Splats] == null)
  {
   CoordRect splatsPixels = CoordToPixels(coord, terrainData.alphamapResolution, numChunksPerTerr
   float[,,] splatsData;
```

```
      using (Log.Group("Read splats"))
       splatsData = terrainData.GetAlphamaps(splatsPixels.offset.x, splatsPixels.offset.z, splatsPixels.size.x
      //using (new Log.Timer("Read alpha texture"))
      // { Texture2D[] atex = terrainData.alphamapTextures; }
      topSet[terrain, coord, DataType.Splats] = splatsData;
     }
    }
  }
}

internal static CoordRect CoordToPixels (Coord coord, int resolution, int numChunks)
//Converts chunk coord to pixel rect
//Note that we don't need exact number of pixels in each chunk, come of them might have 64 pixels, some
//Tested (MapMagicTester)
{
 Vector2D minPixelFloat = (1f * resolution / numChunks) * (Vector2D)coord;
 Coord minPixel = (Coord)minPixelFloat;

 Vector2D maxPixelFloat = (1f * resolution / numChunks) * (Vector2D)(coord+1);
 Coord maxPixel = (Coord)maxPixelFloat;
 if (coord.x == numChunks-1) maxPixel.x = resolution;
 if (coord.z == numChunks-1) maxPixel.z = resolution;

 return new CoordRect(minPixel, maxPixel-minPixel);
}
```

```
public void Perform ()
{
 Set topSet = sets[sets.Count-1];
 sets.RemoveAfter(sets.Count-2);

 foreach ((Terrain terrain, Coord coord, DataType dataType, object data) in topSet.Datas())
 {
  TerrainData terrainData = terrain.terrainData;

  switch (dataType)
  {
   case DataType.Height:
    CoordRect heightPixels = CoordToPixels(coord, terrainData.heightmapResolution, numChunksPerTerr
    terrainData.SetHeights(heightPixels.offset.x, heightPixels.offset.z, (float[,])data);
    break;
   case DataType.Splats:
    CoordRect splatsPixels = CoordToPixels(coord, terrainData.alphamapResolution, numChunksPerTerra
    terrainData.SetAlphamaps(splatsPixels.offset.x, splatsPixels.offset.z, (float[,,])data);
    break;
  }
 }
}

//public static CoordRect RectByCoord
```

```
 }

}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Products;
using MapMagic.Expose;
using System.Threading;
using System;

[assembly: System.Runtime.CompilerServices.InternalsVisibleTo("MapMagic.Tests.Editor")]

namespace MapMagic.Brush
{
 public class TerrainApplyData
 {
  private RenderTexture heightTex;
  private Texture2D[] splatsTexs;
  private TerrainLayer[] splatsPrototypes;

  public void ReadHeight (TerrainData terrainData) => heightTex = terrainData.heightmapTexture;
```

```
 public void ReadSplats (TerrainData terrainData)
 {
  splatsTexs = terrainData.alphamapTextures;
  splatsPrototypes = terrainData.terrainLayers;
 }

 public void Apply (TerrainData terrainData)
 {
  //if (heightTex != null) terrainData.SetHeightmapTexture(heightTex);
 }
}

public class UndoFull
{
 const string undoName = "Brush Stroke";
 public string lastUndoName;  //the last undo group name (kept to know it on UndoRedoPerformed)

 //private Stack<Set> sets = new Stack<Set>();  //we've got to remove first items from it, so using list instea
 [SerializeField] private List< Dictionary<Terrain,TerrainApplyData> > sets = new List< Dictionary<Terrain,

 #if UNITY_EDITOR
 ///Calling Undo
 ///Clone of Tools.GUI.Undo, except it's working with brush rather than gui
  public UndoFull ()
  {
   UnityEditor.Undo.undoRedoPerformed -= OnUndoRedoPerformed;
```

```
 UnityEditor.Undo.undoRedoPerformed += OnUndoRedoPerformed;
}

public void OnUndoRedoPerformed ()
{
 string currGroupName = UnityEditor.Undo.GetCurrentGroupName();

 if (currGroupName == undoName || currGroupName == lastUndoName)
 // a bit hacky here. On undoRedoPerformed there is already no current group in stack, and no way to ge
 // so we store previous (before mm change) name and performing undo if this name is first in stack
 // TODO: use undo from MapMagicBrush with ids, it's more stable
 {
  Perform();

  if (currGroupName == lastUndoName)
   lastUndoName = null;
 }
}
#endif

public void NewGroup (MapMagicBrush brush)
///Registering new undo (at the start of each stroke)
{
 /*if (sets.Count > 10)
 {
  List<Set> last10 = new List<Set>(sets);
```

```
 }*/

 sets.Add( new Dictionary<Terrain,TerrainApplyData>() );

 #if UNITY_EDITOR
  string currGroupName = UnityEditor.Undo.GetCurrentGroupName();
  if (currGroupName != undoName)
   lastUndoName = currGroupName;

  UnityEditor.Undo.RecordObject(brush, undoName);
  brush.temp = !brush.temp;
 #endif
}


public void Append (Terrain[] terrains, Vector2D worldPos, Vector2D worldSize,
 bool readHeight=false, bool readSplats=false)
{
 foreach (Terrain terrain in terrains)
 {
  Dictionary<Terrain,TerrainApplyData> topSet = sets[sets.Count-1];
  if (topSet.ContainsKey(terrain))
   continue;

  TerrainData terrainData = terrain.terrainData;
```

```
  Vector2D terrainPos = (Vector2D)terrain.transform.position;
  Vector2D terrainSize = (Vector2D)terrain.terrainData.size;
  if (!Vector2D.Intersects(worldPos, worldSize, terrainPos, terrainSize))
   continue;

  TerrainApplyData applyData = new TerrainApplyData();
  if (readHeight) applyData.ReadHeight(terrainData);
  if (readSplats) applyData.ReadSplats(terrainData);
  topSet.Add(terrain, applyData);
 }
}


public void Perform ()
{
 Dictionary<Terrain,TerrainApplyData> topSet = sets[sets.Count-1];
 sets.RemoveAfter(sets.Count-2);

 foreach (var kvp in topSet)
 {
  TerrainData terrainData = kvp.Key.terrainData;
  TerrainApplyData applyData = kvp.Value;

  applyData.Apply(terrainData);
 }
```

```
    foreach (MapMagicBrush brush in GameObject.FindObjectsOfType<MapMagicBrush>())
      brush.UpdateCaches();
   }
  }
}
```

```
ï»¿
using UnityEngine;
using UnityEditor;
using UnityEngine.Profiling;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;

using MapMagic.Core;
using MapMagic.Terrains;
using MapMagic.Core.GUI;
using MapMagic.Terrains.GUI;

namespace MapMagic.Brush
{
 public partial class BrushInspector
 {
  const int numCorners = 64;
  static private PolyLine polyLine = new PolyLine(numCorners);
  static private PolyLine traceLine = new PolyLine(numCorners);
  static private PolyLine dashLine = new PolyLine(numCorners);
  static private Vector3[] corners = new Vector3[numCorners];
```

```
public void OnSceneGUI ()
{
 current = this;

 MapMagicBrush brush = (MapMagicBrush)target;
 lastBrush = brush;

 TerrainManager.ExcludeNullTerrains(ref brush.terrains);  //doing before each SceneGui since we can un

 if (brush.enabled  &&
  brush.draw  &&
  IsMouseInSceneView()  &&
  brush.terrains != null  &&
  brush.terrains.Length != 0)
 {
  DrawBrush(brush);

  //redrawing scene
  SceneView.lastActiveSceneView.Repaint();
 }

 if (brush.enabled  &&  Event.current.type == EventType.KeyDown)
 {
  bool keyPressed = false;

  switch (Event.current.keyCode)
```

```
  {
   case KeyCode.LeftBracket:  brush.preset.radius = brush.preset.radius / 1.25f;  keyPressed=true;  break
   case KeyCode.RightBracket:  brush.preset.radius = brush.preset.radius * 1.25f;  keyPressed = true;  bre

   case KeyCode.BackQuote: brush.AssignQuickPreset(0);  keyPressed = true;  break; //not tilde
   case KeyCode.Alpha1: brush.AssignQuickPreset(1);  keyPressed = true;  break;
   case KeyCode.Alpha2: brush.AssignQuickPreset(2);  keyPressed = true;  Event.current.Use();  break;
   case KeyCode.Alpha3: brush.AssignQuickPreset(3);  keyPressed = true;  break;
  }

  if (keyPressed)
   { Repaint(); UI.RepaintAllWindows(); }
 }
}


public static void DrawBrush (MapMagicBrush brush)
{
 //disabling selection
 HandleUtility.AddDefaultControl(GUIUtility.GetControlID(FocusType.Passive));

 //finding position
 Ray worldRay = HandleUtility.GUIPointToWorldRay(Event.current.mousePosition);

 HashSet<Terrain> possibleTerrains = new HashSet<Terrain>(brush.terrains);
 RaycastHit hit = TerrainAiming.GetAimedTerrainHit(worldRay, possibleTerrains, null);
```

```
Vector3 aimPos;
if (hit.collider == null) aimPos = TerrainAiming.GetAimPosAtZeroLevel(worldRay);
else aimPos = hit.point;



//drawing circles
if (brush.draw  &&  Event.current.type == EventType.Repaint)
{
 UnityEngine.Profiling.Profiler.BeginSample("Drawing Locks");

 DrawCircle(polyLine, aimPos, brush.preset.radius*brush.preset.hardness, brush.mainColor, brush.lineTh
 DrawCircle(polyLine, aimPos, brush.preset.radius, brush.falloffColor, brush.lineThickness, corners, brus

 UnityEngine.Profiling.Profiler.EndSample();
}

//pressing
if (brush.draw  &&
 Event.current.type==EventType.MouseDown  &&
 Event.current.button==0  &&
 !Event.current.alt)
  brush.trace.Start(aimPos, brush.Apply, brush);

//drawing
if (brush.draw  &&
```

```
 Event.current.type==EventType.MouseDrag  &&
 Event.current.button==0  &&
 !Event.current.alt)
  brush.trace.Drag(aimPos, brush.preset.spacing*brush.preset.radius, brush.Apply, brush);

//releasing
if (Event.current.rawType == EventType.MouseUp)
{
 brush.trace.Release(brush);

 //if (!brush.readStack.Empty)  //recording undo if brush stroked (anything in current stack)
 // BrushInspector.current.undo.RecordUndo(brush);

 if (Event.current.button==0) brush.tuneStroke.drawTune = false; //un-pressing tune stroke on mouse up

 foreach (Terrain terrain in brush.terrains) //updating collision/lods
  terrain.terrainData.SyncHeightmap();
}

//drawing trace
if (brush.trace.isDrawing)
{
// DrawTrace(traceLine, brush.trace.poses, falloffColor, brush.opTerrains);
 /*DrawDashLine(dashLine,
  start: brush.trace.LastPos,
  end: brush.trace.ApproxNextStep(brush.radius*brush.spacing),
```

```
   color: mainColor,
   dashSize: brush.radius/10,
   terrains: brush.opTerrains);*/
  Vector2D approxNextPos = brush.trace.ApproxNextStep(brush.preset.radius*brush.preset.spacing);

  //DrawTrace(traceLine, new Vector2D[]{brush.trace.LastPos, approxNextPos}, mainColor, brush.opTerra
  //DrawTrace(traceLine, new Vector2D[]{brush.trace.LastStampPos, brush.trace.LastPos}, falloffColor, br
 }

 //drawing tune circle
 if (brush.tuneStroke.stamps.Length != 0)
 {
  (Vector2D pos, float radius) = brush.tuneStroke.GuiCircle();
  DrawCircle(polyLine, (Vector3)pos, radius, brush.mainColor, brush.lineThickness, corners, brush.terrain
 }
}


private static void DrawTrace (PolyLine line, IList<Vector2D> points, Color color, Terrain[] terrains, int sta
{
 if (points.Count <= 1)
  return;

 Vector3[] flooredPoints = new Vector3[points.Count];
 for(int i=0; i<flooredPoints.Length; i++)
  flooredPoints[i] = (Vector3)points[i];
```

```
 FloorPoints(flooredPoints, terrains);

 line.DrawLine(flooredPoints, color, 8, zMode:PolyLine.ZMode.Overlay, offset:0.1f);
}

private static void DrawDashLine (PolyLine line, Vector2D start, Vector2D end, Color color, float dashSize
{
 float dist = (start-end).Magnitude;
 float numDashes = dist/dashSize;
 if (numDashes == 0)
  return;

 Vector3[][] dashes = new Vector3[(int)numDashes+1][];

 Vector3[] flooredStartEnd = new Vector3[] { (Vector3)start, (Vector3)end };
 FloorPoints(flooredStartEnd, terrains);

 for (int d=0; d<dashes.Length; d++)
 {
  float percent = d / numDashes;
  float ePercent = (d+0.5f) / numDashes;

  dashes[d] = new Vector3[] {
   flooredStartEnd[0]*percent + flooredStartEnd[1]*(1-percent),
```

```
   flooredStartEnd[0]*ePercent + flooredStartEnd[1]*(1-ePercent) };
 }

 line.DrawLine(dashes, color, 8, zMode:PolyLine.ZMode.Overlay, offset:0.1f);
}



private static float DrawCircle (PolyLine line, Vector3 center, float radius, Color color, float thickness, Vect
/// Return an average height BTW almost for free :)
/// Copy of lock circle, not reference since I'd like to be free to change it
{
 int numCorners = corners.Length;
 float step = 360f/(numCorners-1);

 for (int i=0; i<corners.Length; i++)
  corners[i] = new Vector3( Mathf.Sin(step*i*Mathf.Deg2Rad), 0, Mathf.Cos(step*i*Mathf.Deg2Rad) ) * rad

 FloorPoints(corners, terrains);

 line.DrawLine(corners, color, thickness, zMode:PolyLine.ZMode.Overlay, offset:0.1f);
 //Handles.DrawAAPolyLine(lineThickness, corners);

 //adjusting center height
 float heightSum = 0;
 for (int i=0; i<corners.Length; i++)
  heightSum += corners[i].y;
```

```
 return heightSum/(corners.Length-1);
}



private static void FloorPoints (Vector3[] points, Terrain[] terrains)
/// Floors the array of points to terrain
{
 Terrain prevTerrain = null;
 Rect prevRect = new Rect();

 for (int i=0; i<points.Length; i++)
 {
  Vector3 point = points[i];
  Vector2 point2D = new Vector2(point.x, point.z); //to find out if rects contains point

  //checking if the point lays within the same terrain first
  Terrain terrain = null;
  if (prevRect.Contains(point2D))
   terrain = prevTerrain;

  //finding proper terrain in all terrains in it's not in rect
  else
  {
   foreach (Terrain newTerrain in terrains)
   {
    if (newTerrain == null)
```

```
      continue;

    Vector3 terrainPos = newTerrain.transform.position;
    Vector3 terrainSize = newTerrain.terrainData.size;
    Rect terrainRect = new Rect(terrainPos.x, terrainPos.z, terrainSize.x, terrainSize.z);

    if (terrainRect.Contains(point2D))
    {
     terrain = newTerrain;
     prevTerrain = newTerrain;
     prevRect = terrainRect;
     break;
    }
   }
  }

  //sampling height
  if (terrain != null) points[i].y = terrain.SampleHeight(point);
 }
}


private static bool IsMouseInSceneView ()
{
 foreach (SceneView sceneView in SceneView.sceneViews)
  if (EditorWindow.mouseOverWindow == sceneView)
```

```
        return true;
      return false;
    }
  }

}
```

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Core.GUI;
using MapMagic.Expose.GUI;

namespace MapMagic.Nodes.GUI
{
 public static class BrushGraphTemplate
 {
   //empty graph is created viaCreateAssetMenuAttribute,
   //but unfortunately there's only one attribute per class

   [MenuItem("Assets/Create/MapMagic/Brush Graph", priority = 102)]
   static void MenuCreateMapMagicBrushGraph(MenuCommand menuCommand)
   {
    ProjectWindowUtil.StartNameEditingIfProjectWindowExists(0,
     ScriptableObject.CreateInstance<TmpCallbackRecieverBrush>(),
     "Brush Graph.asset",
```

```
  TexturesCache.LoadTextureAtPath("MapMagic/Icons/AssetBig"),
  null);
}

class TmpCallbackRecieverBrush : UnityEditor.ProjectWindowCallback.EndNameEditAction
{
 public override void Action(int instanceId, string pathName, string resourceFile)
 {
  Graph graph = CreateBrushErosionTemplate();
  graph.name = System.IO.Path.GetFileName(pathName);
  AssetDatabase.CreateAsset(graph, pathName);

  ProjectWindowUtil.ShowCreatedAsset(graph);

  GraphInspector.allGraphsGuids = new HashSet<string>(AssetDatabase.FindAssets("t:Graph"));
 }
}

public static Graph CreateBrushErosionTemplate ()
{
 Graph graph = GraphInspector.CreateInstance<Graph>();

 Brush.BrushReadHeight206 heightIn = (Brush.BrushReadHeight206)Generator.Create(typeof(Brush.Bru
 graph.Add(heightIn);
 heightIn.guiPosition = new Vector2(-270,-100);
```

```
MatrixGenerators.Spot210 fallof = (MatrixGenerators.Spot210)Generator.Create(typeof(MatrixGenerator
graph.Add(fallof);
fallof.guiPosition = new Vector2(-270,-20);

MatrixGenerators.Erosion200 erosion = (MatrixGenerators.Erosion200)Generator.Create(typeof(MatrixG
graph.Add(erosion);
erosion.iterations = 1;
erosion.guiPosition = new Vector2(-70,-220);
graph.Link(erosion, heightIn);

MatrixGenerators.Mask200 mask = (MatrixGenerators.Mask200)Generator.Create(typeof(MatrixGenera
graph.Add(mask);
mask.guiPosition = new Vector2(130,-100);
mask.invert = true;
graph.Link(erosion, mask.aIn);
graph.Link(heightIn, mask.bIn);
graph.Link(fallof, mask.maskIn);

Brush.BrushWriteHeight206 heightOut = (Brush.BrushWriteHeight206)Generator.Create(typeof(Brush.B
graph.Add(heightOut);
heightOut.guiPosition = new Vector2(350,-100);
graph.Link(mask, heightOut);

Expose.Exposed.Entry positionEntry = new Expose.Exposed.Entry(fallof.id, "position", "Position", typeof
graph.exposed.Add(positionEntry);
```

```
 Expose.Exposed.Entry radiusEntry = new Expose.Exposed.Entry(fallof.id, "radius", "Radius", typeof(floa
 graph.exposed.Add(radiusEntry);

 Expose.Exposed.Entry hardnessEntry = new Expose.Exposed.Entry(fallof.id, "hardness", "Hardness", ty
 graph.exposed.Add(hardnessEntry);

 graph.defaults.Add("Position", typeof(Vector2D), new Vector2D());
 graph.defaults.Add("Radius", typeof(float), 30f);
 graph.defaults.Add("Hardness", typeof(float), 0.5f);

 return graph;
}


public static Graph CreateBrushAddTemplate ()
{
 Graph graph = GraphInspector.CreateInstance<Graph>();
 graph.name = "AddTemplate";

 Brush.BrushReadHeight206 heightIn = (Brush.BrushReadHeight206)Generator.Create(typeof(Brush.Bru
 graph.Add(heightIn);
 heightIn.guiPosition = new Vector2(-400, -250);

 MatrixGenerators.Spot210 fallof = (MatrixGenerators.Spot210)Generator.Create(typeof(MatrixGenerator
 graph.Add(fallof);
 fallof.guiPosition = new Vector2(-400, -450);
```

```
MatrixGenerators.Blend200 blend = (MatrixGenerators.Blend200)Generator.Create(typeof(MatrixGenera
graph.Add(blend);
blend.guiPosition = new Vector2(-150, -350);
blend.layers = new MatrixGenerators.Blend200.Layer[2] { new MatrixGenerators.Blend200.Layer(), new
blend.layers[0].opacity = 1;

Brush.BrushWriteHeight206 heightOut = (Brush.BrushWriteHeight206)Generator.Create(typeof(Brush.B
graph.Add(heightOut);
heightOut.guiPosition = new Vector2(50, -350);



graph.Link(heightIn, blend.layers[0].inlet);
graph.Link(fallof, blend.layers[1].inlet);
graph.Link(blend, heightOut);



Group group = new Group();
graph.groups = new Group[] {group};
group.guiPos = new Vector2(-450, -520);
group.guiSize = new Vector2(670, 350);
group.name = "Template Graph";
group.comment = "Changing it will not be saved";



Expose.Exposed.Entry positionEntry = new Expose.Exposed.Entry(fallof.id, "position", "Position", typeof
```

```
    graph.exposed.Add(positionEntry);

    Expose.Exposed.Entry radiusEntry = new Expose.Exposed.Entry(fallof.id, "radius", "Radius", typeof(floa
    graph.exposed.Add(radiusEntry);

    Expose.Exposed.Entry hardnessEntry = new Expose.Exposed.Entry(fallof.id, "hardness", "Hardness", ty
    graph.exposed.Add(hardnessEntry);

    Expose.Exposed.Entry intensityEntry = new Expose.Exposed.Entry(fallof.id, "intensity", "Intensity*Radiu
    graph.exposed.Add(intensityEntry);



    graph.defaults.Add("Position", typeof(Vector2D), new Vector2D());
    graph.defaults.Add("Radius", typeof(float), 30f);
    graph.defaults.Add("Hardness", typeof(float), 0.5f);
    graph.defaults.Add("Intensity", typeof(float), 0.1f);
    graph.defaults.Add("TerrainHeight", typeof(float), 1);

    return graph;
   }
 }
}
```

```
ï»¿
using UnityEngine;
using UnityEditor;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;

using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Locks;
using MapMagic.Nodes.GUI; //to open graph


namespace MapMagic.Brush
{
 [CustomEditor(typeof(MapMagicBrush))]
 public partial class BrushInspector : Editor
 {
  public static BrushInspector current; //assigned on draw and removed after
  public static MapMagicBrush lastBrush; //last selected brush or any brush in scene (for graph nodes edito

  //public Undo undo = new Undo();
```

```
UI ui = new UI();
bool guiAbout = false;

//[RuntimeInitializeOnLoadMethod]
[UnityEditor.InitializeOnLoadMethod]
static void Initialize ()
{
 lastBrush = FindObjectOfType<MapMagicBrush>();
}

static void AssignCurrent (SceneView sceneView)
{
 //Debug.Log(Resources.FindObjectsOfTypeAll<MapMagicBrush>().Any());
 SceneView.duringSceneGui -= AssignCurrent;
}


public void OnEnable ()
{
 current = this;
 MapMagicBrush brush = (MapMagicBrush)target;
 lastBrush = brush;

 if (brush.preset.graph == null)
  brush.preset.graph = BrushGraphTemplate.CreateBrushAddTemplate();
```

```
//   UnityEditor.Undo.undoRedoPerformed -= OnUndoRedoPerformed; //done via undo itself
//   UnityEditor.Undo.undoRedoPerformed += OnUndoRedoPerformed;

   GraphWindow.OnGraphChanged -= RefreshTuneStroke;
   GraphWindow.OnGraphChanged += RefreshTuneStroke;

   brush.preset.SyncOvd();

   //adding initial terrains on first brush launch
   if (!brush.terrainsAdded)
   {
    Terrain[] terrains = brush.GetComponentsInChildren<Terrain>();
    TerrainManager.TryAddTerrains(ref brush.terrains, terrains, out string tmpErr); //no message for improp
    brush.terrainsAdded = true;
   }

   //checking all terrains size/resolution consistency since it could be changed while in the other inspector
   if (!(TerrainManager.ExcludeImproperTerrains(ref brush.terrains, out string error)))
    EditorUtility.DisplayDialog("MapMagic Brush", error, "OK");

  }

//  public void OnUndoRedoPerformed () => ((MapMagicBrush)target).undo.Perform();
  public void RefreshTuneStroke (Graph graph)
  {
/*   MapMagicBrush brush = (MapMagicBrush)target;
```

```
 if (brush.graph == graph  &&  brush.tuneStroke.stamps.Length != 0)
  brush.tuneStroke.ApplyTune();*/
}




public override void  OnInspectorGUI ()
{
 current = this;
 MapMagicBrush brush = (MapMagicBrush)target;

 if (ui.undo == null)
  ui.undo = new Den.Tools.GUI.Undo {
   undoObject = brush,
   undoName = "MapMagic Brush Value"
  };

 ui.Draw(DrawGUI, inInspector:true);

 //current = null;
}

public void DrawGUI ()
{
```

```
Cell.EmptyLinePx(4);

MapMagicBrush brush = (MapMagicBrush)target;
lastBrush = brush;

//Draw
using (Cell.LinePx(28))
{
 if (TerrainEditorEnabled)
  brush.draw = false; //disabling brush if were editing terrain

 Draw.CheckButton(ref brush.draw, visible:false);

 GUIStyle style = UI.current.textures.GetElementStyle(brush.draw ? "MapMagic/PinButtons/UniversalBut
 Draw.Element(style);

 Texture2D colorTex = UI.current.textures.GetColorizedTexture(brush.draw ? "MapMagic/PinButtons/Col
 GUIStyle colorStyle = UI.current.textures.GetElementStyle(colorTex);
 Draw.Element(colorStyle);

 Cell.EmptyRowPx(10);

 using (Cell.RowPx(30))
 {
  Texture2D icon = UI.current.textures.GetTexture("MapMagicBrush/DrawIcon");
  Draw.Icon(icon, scale:0.5f);
```

```
 }

 using (Cell.Row) Draw.Label("Draw", style:UI.current.styles.middleLabel);
 if (Cell.current.valChanged)
 {
  if (brush.draw  &&  brush.preset.graph == null)
  {
   brush.draw = false;
   EditorUtility.DisplayDialog("Error", "Can't enable brush draw since no graph is assigned", "OK");
   return;
  }

  if (brush.draw)
  {
   brush.UpdateCaches();
   TerrainEditorEnabled = false;
  }
 }
}

if (brush.draw  &&  brush.tuneStroke.drawTune) brush.tuneStroke.drawTune = false;
if (brush.draw  &&  brush.tuneStroke.stamps.Length!=0) brush.tuneStroke.stamps = new Stamp[0];

//Current Preset
Cell.EmptyLinePx(4);
using (Cell.LineStd)
```

```
using (new Draw.FoldoutGroup(ref brush.guiBrush, "Brush", isLeft:true))
 if (brush.guiBrush)
 {
  using (Cell.LineStd)
  {
   Preset newPreset = Draw.ObjectField(brush.sourcePreset, "Preset Source");
   if (Cell.current.valChanged)
    brush.AssignPreset(newPreset);
  }

  Cell.EmptyLinePx(6);
  using (Cell.LinePx(0))
   PresetInspector.DrawPreset(brush.preset);

  Cell.EmptyLinePx(4);
  using (Cell.LinePx(22))
  {
   Cell.EmptyRow();
   using (Cell.RowPx(80))
   {
    Cell.current.disabled = brush.sourcePreset==null;
    if (Draw.Button("Save"))
     brush.sourcePreset.CopyFrom(brush.preset);
   }
   using (Cell.RowPx(80))
   {
```

```
     if (Draw.Button("Save As..."))
     {
      ScriptableAssetExtensions.SaveAsset(brush.preset, filename:"BrushPreset", caption:"Save current p
      brush.AssignPreset(brush.preset); //this will create it's copy
     }
    }
   }
  }

//Quick Selection
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref brush.guiPresets, "Presets", isLeft:true))
  if (brush.guiPresets)
   PresetInspector.DrawQuickSelection(brush);

//Terrains
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref brush.guiTerrains, "Terrains", isLeft:true))
  if (brush.guiTerrains)
  {
   for (int t=0; t<brush.terrains.Length; t++)
    using (Cell.LineStd)
   {
    Terrain terrain = brush.terrains[t];
```

```
 using(Cell.RowPx(20))
 {
  if (!TerrainManager.CheckSplatResolution(terrain))
  {
   //Cell.current.tooltip = TerrainManager.
   if (Draw.Button(UI.current.textures.GetTexture("DPUI/Icons/Warning"), visible:false))
    DrawSplatResWarning(terrain);
  }
 }

 using (Cell.Row) Draw.ObjectField(terrain);

 using (Cell.RowPx(20))
  if (Draw.Button(UI.current.textures.GetTexture("DPUI/Icons/Remove"), visible:false))
   ArrayTools.RemoveAt(ref brush.terrains, t);
}

using (Cell.LineStd)
{
 using (Cell.RowPx(40))
  Draw.Label("Add");

 using (Cell.Row)
 {
  //using (Cell.LineStd)
```

```
// if (Draw.Button("Select"))
//  ScriptableAssetExtensions.ShowObjectSelector(typeof(Terrain), 12345, true, onClosed:TrySelectT

using (Cell.LineStd)
 if (Draw.Button("This Component"))
 {
  Terrain terrain = brush.GetComponent<Terrain>();
  if (terrain != null)
  {
   if (!TerrainManager.TryAddTerrain(ref brush.terrains, terrain, out string error))
    EditorUtility.DisplayDialog("MapMagic Brush", error, "OK");
   else if (!TerrainManager.CheckSplatResolution(terrain))
    DrawSplatResWarning(terrain);
  }
 }

using (Cell.LineStd)
 if (Draw.Button("Child Terrains"))
 {
  Terrain[] terrains = brush.GetComponentsInChildren<Terrain>();
  if (!TerrainManager.TryAddTerrains(ref brush.terrains, terrains, out string error))
    EditorUtility.DisplayDialog("MapMagic Brush", error, "OK");
 }

using (Cell.LineStd)
 if (Draw.Button("All Terrains"))
```

```
      {
       Terrain[] terrains = GameObject.FindObjectsOfType<Terrain>();
       if (!TerrainManager.TryAddTerrains(ref brush.terrains, terrains, out string error))
        EditorUtility.DisplayDialog("MapMagic Brush", error, "OK");
      }

     using (Cell.LineStd)
     {
      Terrain newTerrain = Draw.ObjectField<Terrain>(null, "Custom", allowSceneObject:true);
      if (newTerrain != null)
      {
       if (!TerrainManager.TryAddTerrain(ref brush.terrains, newTerrain, out string error))
        EditorUtility.DisplayDialog("MapMagic Brush", error, "OK");
       else if (!TerrainManager.CheckSplatResolution(newTerrain))
        DrawSplatResWarning(newTerrain);
      }
     }
    }
   }
  }

//Tune
/*Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref brush.guiTuneStroke, "Tune Stroke", isLeft:true))
  if (brush.guiTuneStroke)
```

```
{
 using (Cell.LinePx(26))
 {
  if (brush.tuneStroke.stamps.Length == 0)
   Draw.CheckButton(ref brush.tuneStroke.drawTune, "Draw Stroke");

  else
  {
   using (Cell.Row)
   {
    bool enabled = true;
    Draw.CheckButton(ref enabled, "Tune Stroke");
    if (!enabled) brush.tuneStroke.Clear();
   }

   using (Cell.RowPx(60))
    if (Draw.Button("Apply"))
    {
     brush.tuneStroke.ApplyTune();
     brush.tuneStroke.Clear();
    }
  }
 }

 using (Cell.LineStd) Draw.Label($"Stamps Count: {brush.tuneStroke.stamps.Length}");
}*/
```

```
//Settings
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref brush.guiSettings, "Settings", isLeft:true))
  if (brush.guiSettings)
  {
   using (Cell.LineStd) Draw.Field(ref brush.mainColor, "Hard Color");
   using (Cell.LineStd) Draw.Field(ref brush.falloffColor, "Falloff Color");
   using (Cell.LineStd) Draw.Field(ref brush.lineThickness, "Thickness");
  }

//About
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref guiAbout, "About", isLeft:true))
  if (guiAbout)
  {
   using (Cell.Line)
   {
    using (Cell.RowPx(100))
     Draw.Icon(UI.current.textures.GetTexture("MapMagic/Icons/AssetBig"), scale:0.5f);

    using (Cell.Row)
    {
```

```
        using (Cell.LineStd) Draw.Label("Brush " + MapMagicBrush.version.ToString());
        using (Cell.LineStd) Draw.Label("MapMagic " + Core.MapMagicObject.version.ToString());

        Cell.EmptyLinePx(10);

        using (Cell.LineStd) Draw.URL(" - Online Documentation", "https://gitlab.com/denispahunov/mapmagi
        using (Cell.LineStd) Draw.URL(" - Video Tutorials", url:"https://www.youtube.com/playlist?list=PL8fjbX
        using (Cell.LineStd) Draw.URL(" - Forum Thread", url:"https://forum.unity.com/threads/released-mapm
        using (Cell.LineStd) Draw.URL(" - Issues / Ideas", url:"http://mm2.idea.informer.com");
      }
     }
    }
}


private void DrawSplatResWarning (Terrain terrain)
{
 switch (EditorUtility.DisplayDialogComplex("MapMagic Brush", TerrainManager.splatResError, "Info", "Fi
 {
  case 0: Application.OpenURL("https://gitlab.com/denispahunov/mapmagic/-/wikis/Brush/Resolution"); br
  case 1: TerrainManager.FixSplatResolution(terrain); break;
 }
}


private void TrySelectTerrain (UnityEngine.Object obj)
```

```
/// Called on selecting any object in scene with object picker
{
 if (obj == null) return;
 if (!(obj is GameObject go)) return;

 Terrain terrain = go.GetComponent<Terrain>();
 if (terrain == null) return;

 MapMagicBrush brush = (MapMagicBrush)target;
 if (!TerrainManager.TryAddTerrain(ref brush.terrains, terrain, out string error))
  EditorUtility.DisplayDialog("MapMagic Brush", error, "OK");
 else if (!TerrainManager.CheckSplatResolution(terrain))
  DrawSplatResWarning(terrain);
}




/*[MenuItem ("GameObject/3D Object/MapMagic")]
public static MapMagicObject CreateMapMagic () { return CreateMapMagic(null); }

public static MapMagicObject CreateMapMagic (Graph graph)
{
 GameObject go = new GameObject();
 go.SetActive(false); //to avoid starting generate while graph not assigned
 go.name = "MapMagic";
```

```
    MapMagicObject mapMagic = go.AddComponent<MapMagicObject>();
    Selection.activeObject = mapMagic;

    mapMagic.graph = graph;
    go.SetActive(true);
    mapMagic.tiles.Pin( new Coord(0,0), false, mapMagic );

    //registering undo
    UnityEditor.Undo.RegisterCreatedObjectUndo (go, "MapMagic Create");
    EditorUtility.SetDirty(mapMagic);

//    Selection.activeGameObject = mapMagic.gameObject;
    //MapMagicWindow.Show(mapMagic.gens, mapMagic, asBiome:false);

    return mapMagic;
   }*/

   public static bool TerrainEditorEnabled
   {
    get
    {
     if (activeTerrainInspectorField == null)
      activeTerrainInspectorField = GetActiveTerrainInspectorField();
     return (int)activeTerrainInspectorField.GetValue(null) != 0;
    }
    set
```

```
 {
  if (activeTerrainInspectorField == null)
   activeTerrainInspectorField = GetActiveTerrainInspectorField();
  if (!value)
   activeTerrainInspectorField.SetValue(null, 0);
 }
}

 private static System.Reflection.FieldInfo GetActiveTerrainInspectorField ()
 {
  Type type = Type.GetType("UnityEditor.TerrainInspector, UnityEditor, Version=0.0.0.0, Culture=neutral, P
  return type.GetField("s_activeTerrainInspector", System.Reflection.BindingFlags.NonPublic | System.Ref
 }

 private static System.Reflection.FieldInfo activeTerrainInspectorField = null;

}//class



public class SplatResWarningWindow : EditorWindow
{
 UI ui = new UI();

 public MapMagicBrush brush;
 public Terrain terrain;
```

```
 public void OnGUI () => ui.Draw(DrawGUI, inInspector:false);


 public void DrawGUI ()
 {
  using (Cell.LinePx(100)) Draw.Label("Splat res");
  using (Cell.LineStd)
   if (Draw.Button("Fix"))
   {
    TerrainManager.FixSplatResolution(terrain);
    Close();
   }
 }



 public static void Show (MapMagicBrush brush, Terrain terrain)
 {
  SplatResWarningWindow window = new SplatResWarningWindow();
  window.position = new Rect(Screen.currentResolution.width/2-100, Screen.currentResolution.height/2-50
  window.brush = brush;
  window.terrain = terrain;
  window.ShowAuxWindow();
 }
}



public class PixelErrorWindow : EditorWindow
```

```
{
 UI ui = new UI();

 public void OnGUI () => ui.Draw(DrawGUI, inInspector:false);

 public void DrawGUI ()
 {
  using (Cell.LinePx(100)) Draw.Label("Splat res");
 }

 public static void ShowWindow ()
 {
  PixelErrorWindow window = new PixelErrorWindow();
  window.position = new Rect(Screen.currentResolution.width/2-100, Screen.currentResolution.height/2-50
  window.ShowAuxWindow();
 }
}

}//namespace
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes.GUI;
using MapMagic.Nodes;

namespace MapMagic.Brush
{
 public static class Editors
 {
  private static string[] terrainLayersNames;
  private static List<TerrainLayer> terrainLayers = new List<TerrainLayer>();
  private static HashSet<TerrainLayer> terrainLayersHash = new HashSet<TerrainLayer>();

  private static void RefreshTerrainLayers ()
  {
   //layers themselves
   MapMagicBrush brush = BrushInspector.lastBrush;
```

```
if (brush != null  &&  brush.preset.graph != null  &&  brush.preset.graph == GraphWindow.current.graph  &
 brush.terrains != null  &&  brush.terrains.Length != 0)
 {
  terrainLayers.Clear();
  terrainLayersHash.Clear();
  //do not add brush.opTerrains[0].terrainData.terrainLayers initially, since layer one could be duplicated h

  foreach (Terrain terrain in brush.terrains)
  {
   TerrainLayer[] curLayers = terrain.terrainData.terrainLayers;
   foreach (TerrainLayer layer in curLayers)
   {
    if (!terrainLayersHash.Contains(layer))
     { terrainLayers.Add(layer); terrainLayersHash.Add(layer); }
   }
  }
 }

//names
if (terrainLayersNames == null || terrainLayersNames.Length != terrainLayers.Count)
 terrainLayersNames = new string[terrainLayers.Count];

for (int i=0; i<terrainLayers.Count; i++)
 terrainLayersNames[i] = terrainLayers[i].name;
}
```

```
[Draw.Editor(typeof(BrushReadObjects))]
public static void DrawReadObjects (BrushReadObjects gen)
{
 using (Cell.LineStd) Draw.ToggleLeft(ref gen.specificPrefabs, "Only Specific Prefabs");

 if (gen.specificPrefabs)
  using (Cell.LineStd)
   ObjectsEditors.DrawObjectPrefabs(ref gen.prefabs, true, treeIcon:true);
}


[Draw.Editor(typeof(BrushWriteObjects))]
public static void DrawWriteObjects (BrushWriteObjects gen)
{
 using (Cell.LineStd)
  ObjectsEditors.DrawObjectPrefabs(ref gen.prefabs,multiPrefab:true, treeIcon:true);

 using (Cell.LinePx(0))
  using (Cell.Padded(2,2,0,0))
 {
  Cell.EmptyRowPx(4);

  using (Cell.LinePx(0))
   using (new Draw.FoldoutGroup(ref gen.guiProperties, "Properties"))
    if (gen.guiProperties)
```

```
    {
     using (Cell.LineStd) Draw.ToggleLeft(ref gen.instantiateClones, "As Clones");
    }

  Cell.EmptyRowPx(2);
  ObjectsEditors.DrawPositioningSettings(gen.posSettings, billboardRotWaring:true);
 }
}


[Draw.Editor(typeof(BrushTransferObjects))]
public static void DrawTransferObjects (BrushTransferObjects gen)
{
 using (Cell.LineStd) Draw.ToggleLeft(ref gen.specificPrefabs, "Only Specific Prefabs");

 if (gen.specificPrefabs)
  using (Cell.LineStd)
   ObjectsEditors.DrawObjectPrefabs(ref gen.prefabs, true, treeIcon:true);

 using (Cell.LinePx(0))
  using (Cell.Padded(2,2,0,0))
 {
  Cell.EmptyRowPx(2);
  ObjectsEditors.DrawPositioningSettings(gen.posSettings, billboardRotWaring:true, showRelativeHeight:
 }
}
```

```
[Draw.Editor(typeof(BrushReadTrees))]
public static void DrawReadTrees (BrushReadTrees gen)
{
 using (Cell.LineStd) Draw.ToggleLeft(ref gen.specificPrefabs, "Only Specific Prefabs");

 if (gen.specificPrefabs)
  using (Cell.LineStd)
   ObjectsEditors.DrawObjectPrefabs(ref gen.prefabs, true, treeIcon:true);
}


[Draw.Editor(typeof(BrushWriteTrees))]
public static void DrawWriteTrees (BrushWriteTrees gen)
{
 using (Cell.LineStd)
  ObjectsEditors.DrawObjectPrefabs(ref gen.prefabs,multiPrefab:true, treeIcon:true);

 using (Cell.LinePx(0))
  using (Cell.Padded(2,2,0,0))
 {
  //using (Cell.LineStd) Draw.ToggleLeft(ref gen.guiMultiprefab, "Multi-Prefab");

  Cell.EmptyRowPx(4);
```

```
  using (Cell.LinePx(0))
   using (new Draw.FoldoutGroup(ref gen.guiProperties, "Properties"))
    if (gen.guiProperties)
    {
     Cell.current.fieldWidth = 0.481f;
     using (Cell.LineStd) Draw.Field(ref gen.color, "Color");
     using (Cell.LineStd) Draw.Field(ref gen.lightmapColor, "Lightmap");
     using (Cell.LineStd) gen.bendFactor = Draw.Field(gen.bendFactor, "Bend Factor");
    }

  Cell.EmptyRowPx(2);
  ObjectsEditors.DrawPositioningSettings(gen.posSettings, billboardRotWaring:true);
 }
}

[Draw.Editor(typeof(BrushTransferTrees))]
public static void DrawTransferTrees (BrushTransferTrees gen)
{
 using (Cell.LineStd) Draw.ToggleLeft(ref gen.specificPrefabs, "Only Specific Prefabs");

 if (gen.specificPrefabs)
  using (Cell.LineStd)
   ObjectsEditors.DrawObjectPrefabs(ref gen.prefabs, true, treeIcon:true);

 using (Cell.LinePx(0))
  using (Cell.Padded(2,2,0,0))
```

```
  {
   using (Cell.LinePx(0))
    using (new Draw.FoldoutGroup(ref gen.guiProperties, "Properties"))
     if (gen.guiProperties)
     {
      Cell.current.fieldWidth = 0.481f;
      using (Cell.LineStd) Draw.Field(ref gen.color, "Color");
      using (Cell.LineStd) Draw.Field(ref gen.lightmapColor, "Lightmap");
      using (Cell.LineStd) gen.bendFactor = Draw.Field(gen.bendFactor, "Bend Factor");
     }

   Cell.EmptyRowPx(2);
   ObjectsEditors.DrawPositioningSettings(gen.posSettings, billboardRotWaring:true, showRelativeHeight:
  }
 }
}
}
```

```
ï»¿
using UnityEngine;
using UnityEditor;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;

using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Locks;
using MapMagic.Nodes.GUI; //to open graph


namespace MapMagic.Brush
{
 [CustomEditor(typeof(Preset))]
 public class PresetInspector : Editor
 {
  //private static readonly HashSet<string> automaticNames = new HashSet<string>() { "Radius", "Hardnes
  private static readonly Dictionary<string,Type> automaticNames = new Dictionary<string,Type>() {
   {"Radius", typeof(float)},
   {"Hardness", typeof(float)},
   //{"Margins", typeof(float)},
```

```
 {"Position", typeof(Vector3)},
 {"PrevPosition", typeof(Vector3)},
 {"CapturedPosition", typeof(Vector3)},
 {"TerrainHeight", typeof(float)},
 {"Shift", typeof(bool)} };

UI ui = new UI();

public void OnEnable ()
{
 Preset preset = (Preset)target;
 preset.SyncOvd();
}

public override void  OnInspectorGUI ()
{
 Preset preset = (Preset)target;

 if (ui.undo == null)
  ui.undo = new Den.Tools.GUI.Undo {
   undoObject = preset,
   undoName = "MapMagic Brush Value"
  };

 ui.Draw( ()=> DrawPreset(preset), inInspector:true );
```

```
}


#region Draw Preset


 public static void DrawPreset (Preset preset)
 {
  using (Cell.LineStd)
  {
   preset.graph = Draw.ObjectField(preset.graph, "Graph");
   preset.SyncOvd();
  }

  //default parameters
  Cell.EmptyLinePx(3);
  using (Cell.LineStd) preset.radius = Draw.Field(preset.radius, "Radius", min:0.1f);
  using (Cell.LineStd) preset.hardness = Draw.Field(preset.hardness, "Hardness", min:0, max:1);
  using (Cell.LineStd) preset.spacing =Draw.Field(preset.spacing, "Spacing", min:0.05f, max:5);
  //using (Cell.LineStd) Draw.Field(ref preset.margins, "Margins");

  //overridden parameters
  Cell.EmptyLinePx(3);
  for (int i=0; i<preset.ovd.Count; i++)
  {
   (string name, Type type, object obj) = preset.ovd.GetOverrideAt(i);
```

```
 if (automaticNames.ContainsKey(name)) //it's either radius, hardness, etc
  continue;

 using (Cell.LineStd)
 {
  obj = Draw.UniversalField(obj, type, name);

  if (Cell.current.valChanged)
   preset.ovd.SetOverrideAt(i, name, type, obj);
 }
}

//automatic parameters
Cell.EmptyLinePx(1);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref preset.guiShowUsedAutomatic, "Used Auto-Values", isLeft:false, pa
  if (preset.guiShowUsedAutomatic)
  {
   Cell.current.disabled = true;

   for (int i=0; i<preset.ovd.Count; i++)
   //like overridden parameters, but displaying only automatic
   {
    (string name, Type type, object obj) = preset.ovd.GetOverrideAt(i);

    if (automaticNames.ContainsKey(name)) //it's either radius, hardness, etc
```

```
        using (Cell.LineStd)
        {
         obj = Draw.UniversalField(obj, obj.GetType(), name); //using obj.GetType instead of 'type' otherwise


         if (Cell.current.valChanged)
          preset.ovd.SetOverrideAt(i, name, type, obj);
        }
       }
      }

  //all automatic list
  Cell.EmptyLinePx(1);
  using (Cell.LineStd)
   using (new Draw.FoldoutGroup(ref preset.guiShowAllAutomatic, "All Auto-Values", isLeft:false, padding
    if (preset.guiShowAllAutomatic)
    {
     foreach (var kvp in automaticNames)
      using (Cell.LineStd) Draw.DualLabel(kvp.Key, kvp.Value.Name);
    }
 }



#endregion
```

```
#region Quick Selection

 public static void DrawQuickSelection (MapMagicBrush brush)
 {
  using (Cell.LineStd) Draw.Label("Quick Selection:");

  LayersEditor.DrawLayers(
   brush.quickPresets.Length,
   onDraw:n => DrawQuickSelectionLayer(brush, n),
   onAdd:n => ArrayTools.Add(ref brush.quickPresets, element:null), //ArrayTools.Insert(ref brush.quickPr
   onMove:(n,m) => SwitchQuickSelectionLayers(brush, n, m),
   onRemove:n => ArrayTools.RemoveAt(ref brush.quickPresets, n) );
 }

 public static void DrawQuickSelectionLayer (MapMagicBrush brush, int n)
 {
  Preset preset = brush.quickPresets[n];

  //selection background
  if (brush.sourcePreset == preset)
  {
   GUIStyle style;
   if (selectedOffset==null || selectedOffset.top == 0)
    selectedOffset = new RectOffset(4,4,4,4); //can't create 4444 on serialize
   if (n==0) style = UI.current.textures.GetElementStyle("DPUI/Layers/SelectedTop", selectedOffset);
   else if (n == brush.quickPresets.Length-1) style = UI.current.textures.GetElementStyle("DPUI/Layers/Se
```

```
 else if (brush.quickPresets.Length == 1) style = UI.current.textures.GetElementStyle("DPUI/Layers/Sele
 else style = UI.current.textures.GetElementStyle("DPUI/Layers/SelectedMiddle", selectedOffset);

 Draw.Element(style);
}


Cell.EmptyLinePx(4);
using (Cell.LinePx(22))
{
 //layer icon
 Cell.EmptyRowPx(4);

 using (Cell.RowPx(20))
  Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Layer") );

 Cell.EmptyRowPx(4);

 //selection field
 using (Cell.Row)
 {
  Cell.EmptyRowPx(4);
  using (Cell.RowPx(20))
  {
   if (n < keyCodeNames.Length)
    Draw.Icon( UI.current.textures.GetTexture(keyCodeNames[n]) );
```

```
    }

    Cell.EmptyRowPx(4);

    using (Cell.Row)
     Draw.ObjectField(ref brush.quickPresets[n]);

    Cell.EmptyRowPx(4);

    //if (Draw.Button(visible:false))
    if (!UI.current.layout  &&  Event.current.isMouse  &&  Cell.current.Contains(UI.current.mousePos)  &&
    {
     brush.AssignPreset(preset);
     Event.current.Use();
    }
   }
  }
  Cell.EmptyLinePx(4);
 }

 private static void SwitchQuickSelectionLayers (MapMagicBrush brush, int n, int m)
 {
  if (brush.sourcePreset == brush.quickPresets[n])
   brush.sourcePreset = brush.quickPresets[m];

  else if(brush.sourcePreset == brush.quickPresets[m])
```

```
      brush.sourcePreset = brush.quickPresets[n];


     ArrayTools.Switch(brush.quickPresets, n, m);
    }


    private static readonly string[] keyCodeNames = new string[] {"DPUI/KeyCodes/KeyCode~", "DPUI/KeyC
     "DPUI/KeyCodes/KeyCode3", "DPUI/KeyCodes/KeyCode4", "DPUI/KeyCodes/KeyCode5", "DPUI/KeyC
     "DPUI/KeyCodes/KeyCode8", "DPUI/KeyCodes/KeyCode9", "DPUI/KeyCodes/KeyCode0" };
    public static RectOffset selectedOffset;// = new RectOffset(4,4,4,4);


   #endregion
  }
 }
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Terrains;
using MapMagic.Nodes;
using MapMagic.Nodes.MatrixGenerators;

using UnityEngine.Profiling;

namespace MapMagic.Brush
{
 [Serializable]
 [GeneratorMenu(
  menu = "Brush/Input",
  name = "Grass",
  section=2,
  colorType = typeof(MatrixSet),
  iconName="GeneratorIcons/BrushGrassIn",
  helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/Brush/GrassInOut")]
```

```
public class  BrushReadGrassSet : Generator, IOutlet<MatrixSet>,  IFnEnter<MatrixSet>, IBrushRead
{
 public string Name { get => "Grass"; set {} } //as function portal

 public override void Generate (TileData data, StopToken stop) { }

 public void ReadTerrains (TerrainCache[] terrainCaches, TileData tileData)
 {
  MatrixSet matrixSet = new MatrixSet(tileData.area.full.rect, tileData.area.full.worldPos, tileData.area.full.v

  foreach (TerrainCache terrainCache in terrainCaches)
   terrainCache.ReadGrass(matrixSet);

  tileData.StoreProduct(this, matrixSet);
 }


 private static void PerformRead (Terrain terrain,  MatrixSet matrixSet)
 /// Calls ReadSplats, but prepares per-channel matrices beforehand
 /// Rect needed to call with empty tlayeyrs-matrices dict
 {
  TerrainData terrainData =  terrain.terrainData;
  int resolution = terrainData.detailResolution;

  CoordRect terrainRect = terrain.PixelRect(resolution);
  CoordRect intersection = CoordRect.Intersected(terrainRect, matrixSet.rect);
```

```
  if (intersection.size.x<=0 || intersection.size.z<=0) return;

  DetailPrototype[] detLayers = terrainData.detailPrototypes;
  UnityEngine.Object[] textures = detLayers.Select(p => p.Object());

  for (int i=0; i<detLayers.Length; i++)
  {
   MatrixSet.Prototype prototype = new MatrixSet.Prototype(detLayers, i);

   if (!matrixSet.TryGetValue(prototype, out Matrix matrix)) //adding new matrix if it's not in set yet
   {
    matrix = new Matrix(matrixSet.rect);
    matrixSet[prototype] = matrix;
   }

   int[,] layer = terrainData.GetDetailLayer(
    intersection.offset.x-terrainRect.offset.x,
    intersection.offset.z-terrainRect.offset.z,
    intersection.size.x,
    intersection.size.z,
    i);

   matrix.ImportDetail(layer, intersection.offset, density:1);
  }
 }
}
```

```
[Serializable]
[GeneratorMenu(
 menu = "Brush/Output",
 name = "Grass",
 section=2,
 colorType = typeof(MatrixSet),
 iconName="GeneratorIcons/BrushGrassOut",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/Brush/GrassInOut")]
public class  BrushWriteGrassSet : Generator, IInlet<MatrixSet>,  IFnExit<MatrixSet>, IBrushWrite, IRelev
{
 public string Name { get => "Grass"; set {} } //as function portal



 public override void Generate (TileData data, StopToken stop) { }



 public void WriteTerrains (TerrainCache[] terrainCaches, CacheChange change, TileData tileData)
 {
  MatrixSet matrixSet = tileData.ReadInletProduct(this);

  foreach (TerrainCache terrainCache in terrainCaches)
   terrainCache.WriteGrass(matrixSet);

  change.AddRect(matrixSet.rect);
```

```
 change.AddFlag(CacheChange.Type.Grass);
}



public static void PerformWrite (Terrain terrain, MatrixSet matrixSet, Noise random)
{
 DetailPrototype[] originalDetLayers = terrain.terrainData.detailPrototypes;

 //adding to terrain prototypes that are in set, but not yet present on terrain
 DetailPrototype[] detLayers = originalDetLayers;
 foreach (MatrixSet.Prototype prototype in matrixSet.Prototypes)
  detLayers = prototype.CheckAppendLayers(detLayers);

 //finding intersection
 int resolution = terrain.terrainData.detailResolution;
 if (originalDetLayers.Length==0)  //no grass of any type added
 {
  resolution = terrain.terrainData.heightmapResolution;
  terrain.terrainData.SetDetailResolution(terrain.terrainData.heightmapResolution, 16);
 }
 CoordRect terrainRect = terrain.PixelRect(resolution);

 CoordRect intersection = CoordRect.Intersected(terrainRect, matrixSet.rect);
 intersection = CoordRect.Intersected(intersection, matrixSet.rect);
 if (intersection.size.x<=0 || intersection.size.z<=0) return;
```

```
//setting prototypes
if (originalDetLayers != detLayers) terrain.terrainData.detailPrototypes = detLayers;


//setting arrays
for (int p=0; p<detLayers.Length; p++)
{
 MatrixSet.Prototype prototype = new MatrixSet.Prototype(detLayers, p);


 Matrix matrix;
 bool isInSet = matrixSet.TryGetValue(prototype, out matrix);



 if (!isInSet) //clearing channel if it has no matrix layer in output
  continue; //not sure, it might require clearing something


 else
 {
  int[,] detailLayer = new int[intersection.size.z, intersection.size.x];
  matrix.ExportDetail(detailLayer, p, random, density:1);
  terrain.terrainData.SetDetailLayer(
   intersection.offset.x-terrainRect.offset.x,
   intersection.offset.z-terrainRect.offset.z,
   p,
   detailLayer);
 }
}
```

```
  }


 }


}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Terrains;
using MapMagic.Nodes;
using MapMagic.Nodes.MatrixGenerators;

using UnityEngine.Profiling;

namespace MapMagic.Brush
{
 [Serializable]
 [GeneratorMenu(
  menu = "Brush/Input",
  name = "Height",
  section=2,
  colorType = typeof(MatrixWorld),
  iconName="GeneratorIcons/BrushHeightIn",
  helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/Brush/HeightInOut")]
```

```
public class  BrushReadHeight206 : Generator, IOutlet<MatrixWorld>, IFnEnter<MatrixWorld>, IBrushRea
{
 public string Name { get => "Height"; set {} } //as function portal

 public override void Generate (TileData data, StopToken stop) { }

 public void ReadTerrains (TerrainCache[] terrainCaches, TileData tileData)
 {
  MatrixWorld matrix = new MatrixWorld(
   tileData.area.full.rect,
   tileData.area.full.worldPos,
   tileData.area.full.worldSize,
   tileData.globals.height);

  foreach (TerrainCache terrainCache in terrainCaches)
   terrainCache.ReadHeights(matrix);

  tileData.StoreProduct(this, matrix);
 }

 public static void ReadHeights (Terrain terrain, Matrix matrix)
 /// Reads height data from terrain to matrix within given pixel rect (origin/non-terrain rect)
 /// Saves in readData as well (if any)
 {
  TerrainData terrainData =  terrain.terrainData;
  int resolution = terrainData.heightmapResolution;
```

```
 CoordRect terrainRect = terrain.PixelRect(resolution);
 CoordRect intersection = CoordRect.Intersected(terrainRect, matrix.rect);
 if (intersection.size.x<=0 || intersection.size.z<=0) return;

 CoordRect heightsRect = intersection;  //terrain-relative rect (terrain zero is 0,0)
 heightsRect.offset -= terrainRect.offset;

 float[,] heights = terrainData.GetHeights(heightsRect.offset.x, heightsRect.offset.z, heightsRect.size.x, he

 matrix.ImportHeights(heights, intersection.offset);
}

public static void ReadTerrainTexture (Terrain terrain, Matrix matrix, Texture2D tempTex=null, RenderTex
{
 TerrainData terrainData =  terrain.terrainData;
 int resolution = terrainData.heightmapResolution;

 CoordRect terrainRect = terrain.PixelRect(resolution);
 CoordRect intersection = CoordRect.Intersected(terrainRect, matrix.rect);
 if (intersection.size.x<=0 || intersection.size.z<=0) return;

 CoordRect heightsRect = intersection;  //terrain-relative rect (terrain zero is 0,0)
 heightsRect.offset -= terrainRect.offset;

 RenderTexture terrainTex = terrain.terrainData.heightmapTexture;
```

```
  if (tempTex == null ||  tempTex.width != terrainTex.width  ||  tempTex.height != terrainTex.height)
   tempTex = new Texture2D(terrainTex.width,  terrainTex.height, TextureFormat.R16, mipChain:false, line

  //Graphics.CopyTexture(terrainTex, tempTex);
  RenderTexture bacRenTex = RenderTexture.active;
  RenderTexture.active = terrainTex;
  using (Log.Group("Read Pixels"))
  tempTex.ReadPixels(new Rect(0, 0, tempTex.width, tempTex.height), 0, 0);
  tempTex.Apply();
  RenderTexture.active = bacRenTex;

  byte[] bytes = tempTex.GetRawTextureData();
  //float ushortEpsilon = 1f / 65535; //since setheights is using not full ushort range, but range-1
  //matrix.ImportRawFloat(bytes, intersection.offset, intersection.size, mult:0.5f-ushortEpsilon);
  matrix.ImportRaw16(bytes, new Coord(0,0), new Coord(tempTex.width, tempTex.height));
  matrix.Multiply(2);

 // Color[] colors = tempTex.GetPixels();
 // matrix.ImportColors(colors, new Coord(0,0), new Coord(tempTex.width, tempTex.height), channel:0);
 }
}


[Serializable]
[GeneratorMenu(
```

```
 menu = "Brush/Output",
 name = "Height",
 section=2,
 colorType = typeof(MatrixWorld),
 iconName="GeneratorIcons/BrushHeightOut",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/Brush/HeightInOut")]
public class  BrushWriteHeight206 : Generator, IInlet<MatrixWorld>, IFnExit<MatrixWorld>, IBrushWrite, I
{
 public string Name { get => "Height"; set {} } //as function portal

 public override void Generate (TileData data, StopToken stop)
 {
  MatrixWorld matrix = data.ReadInletProduct(this);
  data.heights = matrix; //will need this while placing objects
 }


 public void WriteTerrains (TerrainCache[] terrainCaches, CacheChange change, TileData tileData)
 {
  MatrixWorld matrix = tileData.heights;

  foreach (TerrainCache terrainCache in terrainCaches)
   terrainCache.WriteHeights(matrix);

  change.AddRect(matrix.rect);
  change.AddFlag(CacheChange.Type.Height);
```

```
}


public static void WriteTerrain (Terrain terrain, Matrix matrix)
{
 CoordRect terrainRect = BrushOps.GetHeightPixelRect(terrain);

 CoordRect intersection = CoordRect.Intersected(terrainRect, matrix.rect);
 if (intersection.size.x<=0 || intersection.size.z<=0) return;

 float[,] heights = new float[intersection.size.x, intersection.size.z];
 matrix.ExportHeights(heights, intersection.offset);

 terrain.terrainData.SetHeightsDelayLOD(intersection.offset.x-terrainRect.offset.x, intersection.offset.z-ter
}


public static void WriteTerrainTexture (Terrain terrain, Matrix matrix, Texture2D tempTex=null, RenderTex
{
 TerrainData data = terrain.terrainData;

 CoordRect terrainRect = BrushOps.GetHeightPixelRect(terrain);

 CoordRect intersection = CoordRect.Intersected(terrainRect, matrix.rect);
 if (intersection.size.x<=0 || intersection.size.z<=0) return;
```

```
byte[] bytes = new byte[intersection.size.x*intersection.size.z*4];
float ushortEpsilon = 1f / 65535; //since setheights is using not full ushort range, but range-1
matrix.ExportRawFloat(bytes, intersection.offset, intersection.size, mult:0.5f-ushortEpsilon);

if (tempTex == null ||  tempTex.width != intersection.size.x  ||  tempTex.height != intersection.size.z)
 tempTex = new Texture2D(intersection.size.x, intersection.size.z, TextureFormat.RFloat, mipChain:false
tempTex.LoadRawTextureData(bytes);
tempTex.Apply(updateMipmaps:false);

if (renTex == null ||  tempTex.width != intersection.size.x  ||  tempTex.height != intersection.size.z)
#if UNITY_2019_2_OR_NEWER
 renTex = new RenderTexture(intersection.size.x, intersection.size.z, 32, RenderTextureFormat.RFloat, r
#else
 renTex = new RenderTexture(intersection.size.x, intersection.size.z, 32, RenderTextureFormat.RFloat);
#endif
Graphics.Blit(tempTex, renTex);

RenderTexture bacRenTex = RenderTexture.active;
RenderTexture.active = renTex;

RectInt texRect = new RectInt(0,0, intersection.size.x, intersection.size.z);
data.CopyActiveRenderTextureToHeightmap(texRect, new Vector2Int(intersection.offset.x, intersection.o
//data.DirtyHeightmapRegion(texRect, TerrainHeightmapSyncControl.HeightAndLod); //or seems a bit fas
//data.SyncHeightmap(); //doesn't seems to make difference with DirtyHeightmapRegion, waiting for read

RenderTexture.active = bacRenTex;
```

```
  }
 }


}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Terrains;
using MapMagic.Nodes;
//using MapMagic.Nodes.ObjectsGenerators; //not in objects module anymore

using UnityEngine.Profiling;

namespace MapMagic.Brush
{
 [Serializable]
 [GeneratorMenu(
  menu = "Brush/Input",
  name = "Objects",
  section=2,
  colorType = typeof(TransitionsList),
  iconName="GeneratorIcons/BrushObjectsIn",
  helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/Brush/ObjectsInOut")]
```

```
public class  BrushReadObjects : Generator, IOutlet<TransitionsList>,  IFnEnter<TransitionsList>, IBrushF
{
 public bool specificPrefabs = false;
 public GameObject[] prefabs = new GameObject[1];

 public string Name { get => "Objects"; set {} } //as function portal

 public override void Generate (TileData data, StopToken stop) { }

 public void ReadTerrains (TerrainCache[] terrainCaches, TileData tileData)
 {
  TransitionsList tlist = new TransitionsList();

  HashSet<GameObject> usedPrefabs = null;
  if (specificPrefabs)
  {
   usedPrefabs = new HashSet<GameObject>();
   usedPrefabs.AddRange(prefabs);
  }

  foreach (TerrainCache terrainCache in terrainCaches)
  {
   Terrain terrain = terrainCache.terrain;
   foreach (Transform tfm in ObjectsOps.RelatedTransforms(terrain, tileData.area.full.worldPos, tileData.ar
   {
    if (specificPrefabs)
```

```
    {
     GameObject prefab = ObjectsOps.GetPrefab(tfm);
     if (prefab == null  ||  !usedPrefabs.Contains(prefab))
      continue;
    }

    Transition trs = new Transition(tfm);
    tlist.Add(trs);
   }
  }

  tileData.StoreProduct(this, tlist);
 }
}


[Serializable]
[GeneratorMenu(
 menu = "Brush/Output",
 name = "Transfer Objects",
 section=2,
 colorType = typeof(TransitionsList),
 iconName="GeneratorIcons/BrushObjectsOut",
 disengageable = true,
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Textures")]
public class  BrushTransferObjects : Generator, IInlet<TransitionsList>,  IFnExit<TransitionsList>, IBrushW
```

```
{
 public bool specificPrefabs = false;
 public GameObject[] prefabs = new GameObject[1];
 public PositioningSettings posSettings = new PositioningSettings() { relativeHeight = false };

 public string Name { get => "Objects"; set {} } //as function portal


 public override void Generate (TileData data, StopToken stop) { }


 public void WriteTerrains (TerrainCache[] terrainCaches, CacheChange change, TileData tileData)
 {
  TransitionsList trnList = tileData.ReadInletProduct(this);
  if (trnList == null) return;

  HashSet<Transform> trnInstances = new HashSet<Transform>(); //transforms mentioned in trn instance
  for (int i=0; i<trnList.count; i++)
  {
   Transform instance = trnList.arr[i].instance;
   if (instance != null  &&  !trnInstances.Contains(instance))
    trnInstances.Add(instance);
  }

  HashSet<GameObject> usedPrefabs = null;
  if (specificPrefabs)
```

```
  {
   usedPrefabs = new HashSet<GameObject>();
   usedPrefabs.AddRange(prefabs);
  }

  Terrain[] terrains = terrainCaches.Select(t=>t.terrain);
  ObjectsOps.DestroyObjects(terrains, tileData.area.full.worldPos, tileData.area.full.worldSize,
   exceptObjs: trnInstances,
   onlyPrefabs: specificPrefabs ? usedPrefabs : null );

  ObjectsOps.TransferObjects(trnList,
   onlyPrefabs: specificPrefabs ? usedPrefabs : null );

  //change.AddRect(matrix.rect);
  change.AddFlag(CacheChange.Type.Objects);
 }
}


[Serializable]
[GeneratorMenu(
 menu = "Brush/Output",
 name = "Objects",
 section=2,
 colorType = typeof(TransitionsList),
 iconName="GeneratorIcons/BrushObjectsOut",
```

```
 disengageable = true,
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/Brush/ObjectsInOut")]
public class  BrushWriteObjects : Generator, IInlet<TransitionsList>, IFnExit<TransitionsList>, IBrushWrite
{
 //common settings
 public GameObject[] prefabs = new GameObject[1];
 public PositioningSettings posSettings = new PositioningSettings() {relativeHeight=false};

 //specific settings
 public bool instantiateClones = false;
 public bool guiProperties;

 public string Name { get => "Objects"; set {} } //as function portal


 public override void Generate (TileData data, StopToken stop) { }


 public void WriteTerrains (TerrainCache[] terrainCaches, CacheChange change, TileData tileData)
 {
  HashSet<GameObject> usedPrefabs = new HashSet<GameObject>();
  usedPrefabs.AddRange(prefabs);

  Terrain[] terrains = terrainCaches.Select(t=>t.terrain);
  ObjectsOps.DestroyObjects(terrains, tileData.area.full.worldPos, tileData.area.full.worldSize, onlyPrefabs
```

```
  TransitionsList tlist = tileData.ReadInletProduct(this);
  if (tlist == null) return;

  ObjectsOps.CreateObjects(terrains, tlist, prefabs, posSettings, tileData);

  //change.AddRect(matrix.rect);
  change.AddFlag(CacheChange.Type.Objects);
 }
}


public static class ObjectsOps
{
 public static IEnumerable<Transform> RelatedTransforms (Terrain terrain, Vector2D worldPos, Vector2D
 /// Iterates all transforms in possible objects parent that are within given world rect
 {
  //terrain children (goes first)
  foreach (Transform tfm in ChildRelatedTransforms(terrain.transform, worldPos, worldSize))
   yield return tfm;

  //upper level objects
  if (terrain.transform.parent != null)
   foreach (Transform par in terrain.transform.parent)
  {
   if (par == terrain) //ignore terrain itself
    continue;
```

```
  if (par.GetComponent<Terrain>() != null) //ignore draft terrains
   continue;

  foreach (Transform tfm in ChildRelatedTransforms(par, worldPos, worldSize))
   yield return tfm;
 }
}


private static IEnumerable<Transform> ChildRelatedTransforms (Transform parent, Vector2D worldPos, V
{
  //foreach (Transform tfm in parent) //sometimes returns null instead of transform

  int childCount = parent.childCount;
  for (int i=0; i<childCount; i++)
  {
   Transform tfm = parent.GetChild(i);

   Vector3 pos = tfm.position;
   if (pos.x < worldPos.x  ||  pos.x > worldPos.x+worldSize.x  || pos.z < worldPos.z  ||  pos.z > worldPos.z+
    continue;

   yield return tfm;
  }
}
```

```
public static void DestroyObjects (Terrain[] terrains, Vector2D worldPos, Vector2D worldSize, HashSet<T
/// If related object instance in scene is not contained in transitions list (trs.instance) it is removed
{
 List<Transform> objsToRemove = new List<Transform>();

 foreach (Terrain terrain in terrains)
  foreach (Transform tfm in RelatedTransforms(terrain, worldPos, worldSize))
  {
   if (exceptObjs != null  &&  exceptObjs.Contains(tfm))
    continue;

   if (onlyPrefabs != null)
   {
    GameObject prefab = GetPrefab(tfm);
    if (prefab == null  ||  !onlyPrefabs.Contains(prefab))
     continue;
   }

   objsToRemove.Add(tfm);
  }

 for (int i=objsToRemove.Count-1; i>=0; i--)
  GameObject.DestroyImmediate(objsToRemove[i].gameObject);
```

```
}



public static void TransferObjects (TransitionsList tlist, HashSet<GameObject> onlyPrefabs=null)
/// Moves/rotates/scales objects that are both in list and in scene
/// if heights is null then heights are not considered as relative
{
 for (int t=0; t<tlist.count; t++)
 {
  Transition trs = tlist.arr[t];

  Transform instance = trs.instance;
  if (instance == null)
   continue;

  if (onlyPrefabs != null)
  {
   GameObject prefab = GetPrefab(instance);
   if (prefab == null  ||  !onlyPrefabs.Contains(prefab))
    continue;
  }

  /*if (heights != null)
  {
   float terrainHeight = heights.GetWorldInterpolatedValue(trs.pos.x, trs.pos.z, roundToShort:true);
   terrainHeight *= heights.worldSize.y;
```

```
    trs.pos.y += terrainHeight;
   }*/

   #if UNITY_EDITOR
   UnityEditor.Undo.RecordObject(instance, Undo.Undo.undoName);
   #endif

   instance.position = trs.pos; //transformation.MultiplyPoint3x4(draft.pos); //actually MM generates world p
   instance.rotation = trs.rotation;
   instance.localScale = trs.scale;
  }
 }

public static void CreateObjects (Terrain[] terrains, TransitionsList trns, GameObject[] prefabs, MapMagic
 {
  Noise random = null;
  if (prefabs.Length > 1)
   random = new Noise(12345);

  (Vector3,Vector3,Transform)[] minMaxParents = GatherPerTerrainParents(terrains);

  for (int t=0; t<trns.count; t++)
  {
   Transition trn = trns.arr[t]; //using copy since it's changing in MoveRotateScale
```

```
Transform parent = GetParent(minMaxParents, trn.pos);
if (parent == null) continue; //do not spawn object if it is out of terrains

if (!data.area.active.Contains(trn.pos)) continue; //skipping out-of-active area

posSettings.MoveRotateScale(ref trn, data);

//selecting
GameObject prefab;
if (prefabs.Length == 1)
 prefab = prefabs[0];
else
{
 float rnd = random.Random(trn.hash);
 prefab = prefabs[ (int)(rnd*prefabs.Length) ];
}

//spawning
GameObject instance;
#if UNITY_EDITOR
if (!UnityEditor.EditorApplication.isPlaying &&
 UnityEditor.PrefabUtility.GetPrefabAssetType(prefab)!=UnityEditor.PrefabAssetType.NotAPrefab)  //if n
  instance = (GameObject)UnityEditor.PrefabUtility.InstantiatePrefab(prefab);
else
  instance = GameObject.Instantiate(prefab);
```

```
  UnityEditor.Undo.RegisterCreatedObjectUndo(instance, Undo.Undo.undoName);
  #else
  instance = GameObject.Instantiate(prefab);
  #endif

  instance.transform.parent = parent;
  instance.hideFlags = parent.gameObject.hideFlags;

  //moving
  Transform tfm = instance.transform;

  tfm.position = trn.pos; //transformation.MultiplyPoint3x4(draft.pos);


  if (posSettings.regardPrefabRotation) tfm.localRotation = trn.rotation * prefab.transform.rotation;
  else tfm.localRotation = trn.rotation;

  if (posSettings.regardPrefabScale) tfm.transform.localScale = trn.scale.x * prefab.transform.localScale;
  else tfm.transform.localScale = trn.scale;
 }
}


private static (Vector3,Vector3,Transform)[] GatherPerTerrainParents (Terrain[] terrains)
/// For each of the terrains finds it's min, max and transform parent object to spawn objects
{
```

```
(Vector3,Vector3,Transform)[] minMaxParents = new (Vector3,Vector3,Transform)[terrains.Length];

for (int t=0; t<minMaxParents.Length; t++)
{
 Terrain terrain = terrains[t];

 Vector3 terrainPos = terrain.transform.position;
 Vector3 terrainSize = terrain.terrainData.size;
 Vector3 terrainMax = terrainPos+terrainSize;

 Transform parent = terrain.transform;

 //mapmagic tile is preferable
 if (terrain.transform.parent != null)
  foreach (Transform par in terrain.transform.parent)
  {
   if (par.GetComponent<TerrainTile>() != null)
    parent = par;
  }

 minMaxParents[t] = (terrainPos, terrainMax, parent);
}

 return minMaxParents;
}
```

```
private static Transform GetParent ((Vector3,Vector3,Transform)[] minMaxParents, Vector3 pos)

{
 foreach ((Vector3 terrainMin, Vector3 terrainMax, Transform parent) in minMaxParents)
 {
  if (pos.x > terrainMin.x  &&  pos.x < terrainMax.x  &&
   pos.z > terrainMin.z  &&  pos.z < terrainMax.z)
    return parent;
 }

 return null;
}



public static GameObject GetPrefab (Transform tfm)
/// Returns a prefab of transform instance
{
 GameObject prefab = null;

 #if UNITY_EDITOR
 prefab = UnityEditor.PrefabUtility.GetCorrespondingObjectFromSource(tfm.gameObject);
 #endif

 return prefab;
}
```

```
 }

}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Terrains;
using MapMagic.Nodes;
using MapMagic.Nodes.MatrixGenerators;

using UnityEngine.Profiling;

namespace MapMagic.Brush
{
 [Serializable]
 [GeneratorMenu(
  menu = "Brush/Input",
  name = "Textures",
  section=2,
  colorType = typeof(MatrixSet),
  iconName="GeneratorIcons/BrushTexturesIn",
  helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/Brush/HeightInOut")]
```

```
public class  BrushReadTextureSet : Generator, IOutlet<MatrixSet>,  IFnEnter<MatrixSet>, IBrushRead
{
 public string Name { get => "Maps Set"; set {} } //as function portal

 public override void Generate (TileData data, StopToken stop) { }

 public void ReadTerrains (TerrainCache[] terrainCaches, TileData tileData)
 {
  MatrixSet matrixSet = new MatrixSet(tileData.area.full.rect, tileData.area.full.worldPos, tileData.area.full.v

  foreach (TerrainCache terrainCache in terrainCaches)
   terrainCache.ReadTextures(matrixSet);

  tileData.StoreProduct(this, matrixSet);
 }

 private static void PerformRead (Terrain terrain, MatrixSet matrixSet)
 /// Calls ReadSplats, but prepares per-channel matrices beforehand
 /// Rect needed to call with empty tlayeyrs-matrices dict
 {
  TerrainData terrainData =  terrain.terrainData;
  int resolution = terrainData.alphamapResolution;

  CoordRect terrainRect = terrain.PixelRect(resolution);
  CoordRect intersection = CoordRect.Intersected(terrainRect, matrixSet.rect);
  if (intersection.size.x<=0 || intersection.size.z<=0) return;
```

```
  TerrainLayer[] layers = terrainData.terrainLayers;
  float[,,] splats = terrainData.GetAlphamaps(
   intersection.offset.x-terrainRect.offset.x, intersection.offset.z-terrainRect.offset.z, intersection.size.x, inte


  for (int p=0; p<layers.Length; p++)
  {
   MatrixSet.Prototype prototype = new MatrixSet.Prototype(layers, p);


   if (!matrixSet.TryGetValue(prototype, out Matrix matrix))
   {
    matrix = new Matrix(matrixSet.rect);
    matrixSet[prototype] = matrix;
   }


   matrix.ImportSplats(splats, intersection.offset, p);
  }
 }
}



[Serializable]
[GeneratorMenu(
 menu = "Brush/Output",
 name = "Textures",
 section=2,
```

```
 colorType = typeof(MatrixSet),
 iconName="GeneratorIcons/BrushTexturesOut",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/Brush/HeightInOut")]
public class  BrushWriteTextureSet : Generator, IInlet<MatrixSet>,  IFnExit<MatrixSet>, IBrushWrite, IRele
{
 public string Name { get => "Maps Set"; set {} } //as function portal

 public override void Generate (TileData data, StopToken stop) { }



 public void WriteTerrains (TerrainCache[] terrainCaches, CacheChange change, TileData tileData)
 {
  MatrixSet matrixSet = tileData.ReadInletProduct(this);

  foreach (TerrainCache terrainCache in terrainCaches)
   terrainCache.WriteTextures(matrixSet);

  change.AddRect(matrixSet.rect);
  change.AddFlag(CacheChange.Type.Splats);
 }

 private static void ClearSplats (CoordRect matrixRect, float[,,] splats, int channel) => ClearSplats(matrixR
 private static void ClearSplats (CoordRect matrixRect, float[,,] splats, Coord splatsOffset, int channel)
 ///Same as Matrix.ExportSplats(ch), but just clearing one splats channel
 {
  Coord splatsSize = new Coord(splats.GetLength(1), splats.GetLength(0));  //x and z swapped
```

```
   CoordRect splatsRect = new CoordRect(splatsOffset, splatsSize);

   CoordRect intersection = CoordRect.Intersected(matrixRect, splatsRect);
   Coord min = intersection.Min; Coord max = intersection.Max;

   for (int x=min.x; x<max.x; x++)
    for (int z=min.z; z<max.z; z++)
    {
     int matrixPos = (z-matrixRect.offset.z)*matrixRect.size.x + x - matrixRect.offset.x;
     int heightsPosZ = x - splatsRect.offset.x;
     int heightsPosX = z - splatsRect.offset.z;

     splats[heightsPosX, heightsPosZ, channel] = 0;
    }
  }
 }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Terrains;
using MapMagic.Nodes;
//using MapMagic.Nodes.ObjectsGenerators; //not in objects module anymore

using UnityEngine.Profiling;

namespace MapMagic.Brush
{
 [Serializable]
 [GeneratorMenu(
  menu = "Brush/Input",
  name = "Trees",
  section=2,
  colorType = typeof(TransitionsList),
  iconName="GeneratorIcons/TreesIn",
  helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/Brush/TreesInOut")]
```

```
public class  BrushReadTrees : Generator, IOutlet<TransitionsList>,  IFnEnter<TransitionsList>, IBrushRe
{
 public bool specificPrefabs = false;
 public GameObject[] prefabs = new GameObject[1];

 public string Name { get => "Trees"; set {} } //as function portal


 public override void Generate (TileData data, StopToken stop) { }


 public void ReadTerrains (TerrainCache[] terrainCaches, TileData tileData)
 {
  TransitionsList tlist = new TransitionsList();

  foreach (TerrainCache terrainCache in terrainCaches)
  {
   HashSet<int> onlyIndexes = null;
   if (specificPrefabs) onlyIndexes = TreesOps.UsedPrefabsIndexes(terrainCache.terrain.terrainData.treeP

   TreesOps.TreesFromTerrain(terrainCache.terrain, tlist, tileData.area.full.worldPos, tileData.area.full.worl
  }

  //if (relativeHeight)
  // for (int i=0; i<tlist.count; i++)
  //  tlist.arr[i].pos.y -= BrushReadObjects.GetTerrainHeight(tlist.arr[i].pos, tileData); //terrain.SampleHeight
```

```
  tileData.StoreProduct(this, tlist);
 }
}


[Serializable]
[GeneratorMenu(
 menu = "Brush/Output",
 name = "Transfer Trees",
 section=2,
 colorType = typeof(TransitionsList),
 iconName="GeneratorIcons/BrushTreesOut",
 disengageable = true,
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/Brush/TreesInOut")]
public class  BrushTransferTrees : Generator, IInlet<TransitionsList>, IFnExit<TransitionsList>, IBrushWrit
{
 //common settings
 public bool specificPrefabs = false;
 public GameObject[] prefabs = new GameObject[1];
 public PositioningSettings posSettings = new PositioningSettings() { relativeHeight = false };
 public bool guiProperties;

 //specific settings
 public Color color = Color.white;
 public Color lightmapColor = Color.white;
```

```
public float bendFactor;



public string Name { get => "Move Trees"; set {} } //as function portal



public override void Generate (TileData data, StopToken stop) { }

public void WriteTerrains (TerrainCache[] terrainCaches, CacheChange change, TileData tileData)
{
 TransitionsList trnList = tileData.ReadInletProduct(this);
 if (trnList == null) return;

 //reading height if not loaded for relative pos
 if (tileData.heights == null  &&  posSettings.relativeHeight)
 {
  Area.Dimensions full = tileData.area.full;
  tileData.heights = new MatrixWorld(full.rect, full.worldPos, full.worldSize, tileData.globals.height);

  foreach (TerrainCache terrainCache in terrainCaches)
   BrushReadHeight206.ReadHeights(terrainCache.terrain, tileData.heights);
 }

 //clearing old/adding new trees
 foreach (TerrainCache terrainCache in terrainCaches)
 {
```

```
    Terrain terrain = terrainCache.terrain;

    TreeInstance[] trees = terrain.terrainData.treeInstances;
    TreePrototype[] prototypes = terrain.terrainData.treePrototypes;

    //clearing
    HashSet<int> onlyIndexes = null;
    if (specificPrefabs) onlyIndexes = TreesOps.UsedPrefabsIndexes(prototypes, prefabs);
    TreesOps.ClearTreesWithinRect(ref trees, tileData.area.active.worldPos, tileData.area.active.worldSize,

    //adding instances
    TreesOps.ReCreateTrees(ref trees, trnList, terrain, tileData, posSettings, color, onlyIndexes:onlyIndexes
    terrain.terrainData.treeInstances = trees;
  }

  //change.AddRect(matrix.rect);
  change.AddFlag(CacheChange.Type.Trees);
 }
}


[Serializable]
[GeneratorMenu(
 menu = "Brush/Output",
 name = "Write Trees",
 section=2,
```

```
 colorType = typeof(TransitionsList),
 iconName="GeneratorIcons/BrushTreesOut",
 disengageable = true,
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/Brush/TreesInOut")]
public class  BrushWriteTrees : Generator, IInlet<TransitionsList>, IFnExit<TransitionsList>, IBrushWrite, I
{
 //common settings
 public GameObject[] prefabs = new GameObject[1];
 public PositioningSettings posSettings = new PositioningSettings() { relativeHeight = false };

 //public bool guiMultiprefab;
 public bool guiProperties;

 //specific settings
 public bool useOriginalPrototype = false;
 public Color color = Color.white;
 public Color lightmapColor = Color.white;
 public float bendFactor;


 public string Name { get => "Trees"; set {} } //as function portal


 public override void Generate (TileData data, StopToken stop) { }

 public void WriteTerrains (TerrainCache[]  terrainCaches, CacheChange change, TileData tileData)
```

```
{
 TransitionsList trnList = tileData.ReadInletProduct(this);
 if (trnList == null) return;

 //reading height if not loaded for relative pos
 if (tileData.heights == null  &&  posSettings.relativeHeight)
 {
  Area.Dimensions full = tileData.area.full;
  tileData.heights = new MatrixWorld(full.rect, full.worldPos, full.worldSize, tileData.globals.height);

  foreach (TerrainCache terrainCache in terrainCaches)
   BrushReadHeight206.ReadHeights(terrainCache.terrain, tileData.heights);
 }

 //clearing old/adding new trees
 foreach (TerrainCache terrainCache in terrainCaches)
 {
  Terrain terrain = terrainCache.terrain;

  TreeInstance[] trees = terrain.terrainData.treeInstances;
  TreePrototype[] prototypes = terrain.terrainData.treePrototypes;

  //clearing
  HashSet<int> usedIndexes = TreesOps.UsedPrefabsIndexes(prototypes, prefabs);
  TreesOps.ClearTreesWithinRect(ref trees, tileData.area.active.worldPos, tileData.area.active.worldSize,
```

```
    //adding prototypes
    bool prototypesAdded = TreesOps.AddNewPrototypes(ref prototypes, prefabs, bendFactor);
    if (prototypesAdded)
     terrain.terrainData.treePrototypes = prototypes;

    //adding instances
    int[] prefabIndexes = TreesOps.PrefabsToIndexes(prototypes, prefabs);
    TreesOps.CreateTrees(ref trees, trnList, prefabIndexes, terrain, tileData, posSettings, color);
    terrain.terrainData.treeInstances = trees;
   }

   //change.AddRect(matrix.rect);
   change.AddFlag(CacheChange.Type.Trees);
  }
 }

public static class TreesOps
{
 public static void TreesFromTerrain (Terrain terrain, TransitionsList tlist, Vector2D worldPos, Vector2D wo
 {
  TerrainData data = terrain.terrainData;
  TreeInstance[] allTrees = data.treeInstances;
  TreePrototype[] treePrototypes = data.treePrototypes;

  //getting 0-1 range brush rect (since tree positions are relative to terain)
```

```
 Vector3 terrainPos = terrain.transform.position;
 Vector3 terrainSize = terrain.terrainData.size;
 Vector2D relPos = (Vector2D)(worldPos-terrainPos) / (Vector2D)terrainSize;
 Vector2D relSize = worldSize / (Vector2D)terrainSize;

 for (int i=0; i<allTrees.Length; i++)
 {
  Vector3 pos = allTrees[i].position;
  if (pos.x <= relPos.x || pos.x >= relPos.x+relSize.x ||
   pos.z <= relPos.z || pos.z >= relPos.z+relSize.z)
    continue;

  if (onlyIndexes != null  &&  !onlyIndexes.Contains(allTrees[i].prototypeIndex))
   continue;

  TreePrototype prototype = treePrototypes[allTrees[i].prototypeIndex];
  //not useing number since prototypes might change on apply

  tlist.Add(new Transition(allTrees[i], terrainPos, terrainSize, prototype.prefab.transform));
 }
}


public static void ClearTreesWithinRect (ref TreeInstance[] trees, Vector2D worldPos, Vector2D worldSize
/// Clears all of the trees within world rect
/// If onlyIndexes is not provided - clearing ALL trees
```

```
{
 //getting 0-1 range brush rect (since tree positions are relative to terrain)
 Vector3 terrainPos = terrain.transform.position;
 Vector3 terrainSize = terrain.terrainData.size;
 Vector2D relPos = (Vector2D)(worldPos-terrainPos) / (Vector2D)terrainSize;
 Vector2D relSize = worldSize / (Vector2D)terrainSize;

 //finding trees in rect
 bool[] isInRect = new bool[trees.Length];
 int inRectCount = 0;

 for (int i=0; i<trees.Length; i++)
 {
  if (onlyIndexes != null  &&  !onlyIndexes.Contains(trees[i].prototypeIndex))
   continue;

  Vector3 pos = trees[i].position;

  if (pos.x > relPos.x && pos.x < relPos.x+relSize.x &&
   pos.z > relPos.z && pos.z < relPos.z+relSize.z)
    { isInRect[i] = true; inRectCount++; }
 }

 //re-creating trees skipping rect
 if (inRectCount != 0)
 {
```

```
  TreeInstance[] newTrees = new TreeInstance[trees.Length-inRectCount];


  int c = 0;
  for (int i=0; i<trees.Length; i++)
  {
   if (isInRect[i]) continue;


   newTrees[c] = trees[i];
   c++;
  }


  trees = newTrees;
 }
}



public static void ReCreateTrees (ref TreeInstance[] trees, TransitionsList trnList,
 Terrain terrain, TileData data, PositioningSettings posSettings, Color color,
 HashSet<int> onlyIndexes=null)
/// Adding only those trees from tlist that have their instance prefab assigned (and setting their index to pr
/// Works within brush only (won't add it to other trees)
{
 //terrain position/size to calculate 0-1 range tree positions
 Vector3 terrainPos = terrain.transform.position;
 Vector3 terrainSize = terrain.terrainData.size;
```

```
//prefabs-indexes lut
Dictionary<GameObject,int> prefabToPrototypeIndex = new Dictionary<GameObject,int>();
TreePrototype[] prototypes = terrain.terrainData.treePrototypes;
for (int p=0; p<prototypes.Length; p++)
 prefabToPrototypeIndex.TryAdd(prototypes[p].prefab, p);

//adding
List<TreeInstance> dstInstances = new List<TreeInstance>(trnList.count);
for (int t=0; t<trnList.count; t++)
{
 Transition trn = trnList.arr[t]; //using copy since it's changing in MoveRotateScale

 Transform prefab = trn.instance;
 if (prefab == null)
  continue;

 int prototypeIndex;
 if (!prefabToPrototypeIndex.TryGetValue(prefab.gameObject, out prototypeIndex))
  continue;

 if (onlyIndexes != null  &&  !onlyIndexes.Contains(prototypeIndex))
  continue;

 //if (!data.area.active.Contains(trn.pos)) //skipping out-of-active area
 // continue;
 //otherwise it will remove the tree (not good for relocation)
```

```
  posSettings.MoveRotateScale(ref trn, data);

  TreeInstance tree = CreateTree(trn, prototypeIndex, terrainPos, terrainSize, color);

  if (tree.position.x < 0 || tree.position.z < 0 ||
   tree.position.x > 1 || tree.position.z > 1)
    continue; //this tree is now in the other terrain

  dstInstances.Add(tree);
 }

 ArrayTools.AddRange(ref trees, dstInstances.ToArray());
}


public static void CreateTrees (ref TreeInstance[] trees, TransitionsList trnList, int[] prefabsIndexes,
 Terrain terrain, TileData data, PositioningSettings posSettings, Color color)
/// Adds trnList trees to trees instances array
/// PrefabsIndexes is a list of gen prefabs, converted to prototype indees number as they used on terrain
{
 if (prefabsIndexes.Length == 0)
  return;

 //preparing noise to randomly select prefabs
 Noise random = null;
```

```
if (prefabsIndexes.Length > 1) //no need to randomize only prefab
 random = new Noise(12345);

//terrain position/size to calculate 0-1 range tree positions
Vector3 terrainPos = terrain.transform.position;
Vector3 terrainSize = terrain.terrainData.size;

//adding
List<TreeInstance> treeList = new List<TreeInstance>();

for (int t=0; t<trnList.count; t++)
{
 Transition trn = trnList.arr[t]; //using copy since it's changing in MoveRotateScale

 if (!data.area.active.Contains(trn.pos)) continue; //skipping out-of-active area

 posSettings.MoveRotateScale(ref trn, data);

 //selecting
 int index;
 if (prefabsIndexes.Length == 1)
  index = prefabsIndexes[0];
 else
 {
  float rnd = random.Random(trn.hash);
  index = prefabsIndexes[ (int)(rnd*prefabsIndexes.Length) ];
```

```
  }

  if (index < 0)
   continue;

  //spawning
  TreeInstance tree = CreateTree(trn, index, terrainPos, terrainSize, color);

  if (tree.position.x < 0 || tree.position.z < 0 ||
   tree.position.x > 1 || tree.position.z > 1)
    continue;

  treeList.Add(tree);
 }

 ArrayTools.AddRange(ref trees, treeList.ToArray());
}


private static TreeInstance CreateTree (Transition trn, int prototypeIndex, Vector3 terrainPos, Vector3 terr
/// Creates tree instance from floored/rotated/scaled transition using current gen parameters
{
 TreeInstance tree = new TreeInstance();

 tree.position.x = (trn.pos.x - terrainPos.x) / terrainSize.x; //trees should be in 0-1 range
 tree.position.z = (trn.pos.z - terrainPos.z) / terrainSize.z;
```

```
 tree.position.y = trn.pos.y / terrainSize.y;

 tree.rotation = trn.Yaw;
 tree.widthScale = trn.scale.x; // + trs.scale.z)/2;
 tree.heightScale = trn.scale.y;
 tree.prototypeIndex = prototypeIndex;
 tree.color = color;
 tree.lightmapColor = Color.white;

 return tree;
}



public static bool AddNewPrototypes (ref TreePrototype[] prototypes, GameObject[] prefabs, float bendFa
/// If prototypes do not contain prefab from prefabs - appends prototype with new prefab
{
 bool change = false;

 HashSet<GameObject> usedPrefabs = new HashSet<GameObject>();

 for (int i=0; i<prototypes.Length; i++)
  if (!usedPrefabs.Contains(prototypes[i].prefab))
   usedPrefabs.Add(prototypes[i].prefab);

 foreach (GameObject prefab in prefabs)
  if (!usedPrefabs.Contains(prefab))
```

```
  {
   ArrayTools.Add(ref prototypes, new TreePrototype() {prefab=prefab, bendFactor=bendFactor });
   usedPrefabs.Add(prefab); //in case it's twice in prefabs
   change = true;
  }

 return change;
}



public static Dictionary<GameObject,int> PrefabToPrototypeIndexLut (TreePrototype[] prototypes)
/// Creates a dictionary to convert prefab into a tree prototype index
/// Does not actually needs layers prefabs - will return all of the prefabs on terrain
{
 Dictionary<GameObject,int> lut = new Dictionary<GameObject, int>();

 for (int i=0; i<prototypes.Length; i++)
 {
  GameObject prefab = prototypes[i].prefab;
  if (!lut.ContainsKey(prefab))
   lut.Add(prefab, i);
 }

 return lut;
}
```

```
public static HashSet<int> UsedPrefabsIndexes (TreePrototype[] prototypes, GameObject[] prefabs)
/// Gets currently used prototype indexes for all prefab layers
/// If tree prototype index is in hashSet - then it's prefab is among generator prefabs
{
 HashSet<int> opIndexes = new HashSet<int>();

 foreach (GameObject prefab in prefabs)
 {
  int index = -1;

  for (int i=0; i<prototypes.Length; i++)
   if (prototypes[i].prefab == prefab)
   {
    index = i;
    break;
   }

  if (index >= 0)
   opIndexes.Add(index);
 }

 return opIndexes;
}
```

```
  public static int[] PrefabsToIndexes (TreePrototype[] prototypes, GameObject[] prefabs)
  /// For each of the prefabs finds an index matching in prototype
  {
   Dictionary<GameObject,int> lut = new Dictionary<GameObject, int>();

   for (int i=0; i<prototypes.Length; i++)
   {
    GameObject protPrefab = prototypes[i].prefab;
    if (!lut.ContainsKey(protPrefab))
     lut.Add(protPrefab, i);
   }

   int[] prefabIndexes = new int[prefabs.Length];
   for (int i=0; i<prefabs.Length; i++)
   {
    if (lut.TryGetValue(prefabs[i], out int index))
     prefabIndexes[i] = index;
    else
     prefabIndexes[i] = -1;
   }

   return prefabIndexes;
  }
 }
}
```

```
ï»¿#if MAPMAGIC2 //shouldn't work if MM assembly not compiled

using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Products;

#if CTS_PRESENT
using CTS;
#endif

namespace MapMagic.Nodes.MatrixGenerators {
 [System.Serializable]
 [GeneratorMenu(
  menu = "Map/Output",
  name = "CTS",
  section =2,
  drawButtons = false,
  colorType = typeof(MatrixWorld),
  iconName="GeneratorIcons/TexturesOut",
  helpLink = "https://gitlab.com/denispahunov/mapmagic/wikis/output_generators/Textures")]
 public class CTSOutput200 : BaseTexturesOutput<CTSOutput200.CTSLayer>
 {
  //public static CTS.CTSProfile ctsProfile;  //in globals
```

```
public class CTSLayer : BaseTextureLayer { }

public string[] guiTextureNames = null;



public override void Generate (TileData data, StopToken stop)
{
 //generating
 MatrixWorld[] dstMatrices = BaseGenerate(data, stop);

 //adding to finalize
 if (stop!=null && stop.stop) return;
 if (enabled)
 {
  for (int i=0; i<layers.Length; i++)
   data.StoreOutput(layers[i], typeof(CTSOutput200), layers[i],  dstMatrices[i]);
  data.MarkFinalize(Finalize, stop);
 }
 else
  data.RemoveFinalize(finalizeAction);
}

public override FinalizeAction FinalizeAction => finalizeAction; //should return variable, not create new
public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
```

```
{
 #if CTS_PRESENT
 if (data.globals.ctsProfile==null) return;

 //creating control textures
 if (stop!=null && stop.stop) return;
 data.GatherOutputs (typeof(CTSOutput200),
  out CTSLayer[] layers,
  out MatrixWorld[] matrices,
  out MatrixWorld[] masks,
  inSubs:true);
 float[] opacities = layers.Select(l=>l.Opacity);
 int[] channelNums = layers.Select(l=>l.channelNum);

 //purging if no outputs
 if (matrices.Length == 0)
 {
  if (stop!=null && stop.stop) return;
  data.MarkApply(CustomShaderOutput200.ApplyData.Empty);
  return;
 }

 //calculating number of textures
 int maxChannelNum = 0;
 foreach (int chNum in channelNums)
  if (chNum > maxChannelNum) maxChannelNum=chNum;
```

```
 int numTextures = maxChannelNum/4 + 1;

 //blending matrices
 Color[][] colors = CustomShaderOutput200.BlendMatrices(data.area.active.rect, matrices, masks, opaciti

 //pushing to apply
 if (stop!=null && stop.stop) return;
 //var applyData = new ApplyData() { colors = colors };
 var applyData = new ApplyData()
 {
  textureColors = colors,
  profile = (CTSProfile)data.globals.ctsProfile,
 };

 Graph.OnOutputFinalized?.Invoke(typeof(ApplyData), data, applyData, stop);
 data.MarkApply(applyData);

 #endif
}

public override void ClearApplied (TileData data, Terrain terrain)
{

}
```

```
#if CTS_PRESENT
public class ApplyData : IApplyData
{
 public CTSProfile profile;
 public Color[][] textureColors;

 public void Apply (Terrain terrain)
 {
  //saving enable state (since CTS switch to default on enabled when no profile assigned)
  bool activeState = terrain.gameObject.activeSelf;
  terrain.gameObject.SetActive(false);

  CompleteTerrainShader cts = terrain.GetComponent<CompleteTerrainShader>();
  if (cts == null) cts = terrain.gameObject.AddComponent<CompleteTerrainShader>();

  //firstly add splat textures (otherwise CTS will log error on profile assign in playmode)
  int resolution = (int)Mathf.Sqrt(textureColors[0].Length);
  TextureFormat texFormat = TextureFormat.RGBA32;
  for (int i=0; i<textureColors.Length; i++)
  {
   if (textureColors[i] == null) continue;

   Texture2D tex =
    i==0 ? cts.Splat1 :
    i==1 ? cts.Splat2 :
```

```
 i==2 ? cts.Splat3 :
 cts.Splat4;

if (tex==null || tex.width!=resolution || tex.height!=resolution || tex.format!=texFormat)
{
 if (tex!=null)
 {
  #if UNITY_EDITOR
  if (!UnityEditor.AssetDatabase.Contains(tex))
  #endif
   GameObject.DestroyImmediate(tex);
 }

 tex = new Texture2D(resolution, resolution, texFormat, false, true);
 tex.wrapMode = TextureWrapMode.Mirror; //to avoid border seams
 //tex.hideFlags = HideFlags.DontSave;
 //tex.filterMode = FilterMode.Point;

 if (i==0) cts.Splat1 = tex;
 else if (i==1) cts.Splat2 = tex;
 else if (i==2) cts.Splat3 = tex;
 else cts.Splat4 = tex;
}

tex.SetPixels(0,0,tex.width,tex.height,textureColors[i]);
tex.Apply();
```

```
    }

    //then asssign profile
    if (cts.Profile != profile) cts.Profile = profile;

    //enable
    terrain.gameObject.SetActive(activeState);
   }

   public void Read (Terrain terrain) { throw new System.NotImplementedException(); }

   public static ApplyData Empty {get{ return new ApplyData() { textureColors = new Color[0][] }; }}

   public int Resolution
   {get{
    if (textureColors.Length==0) return 0;
    else return (int)Mathf.Sqrt(textureColors[0].Length);
   }}
  }
  #endif
 }

}

#endif //MAPMAGIC2
```

```
ï»¿#if MAPMAGIC2 //shouldn't work if MM assembly not compiled

using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Core;  //used once to get tile size
using MapMagic.Products;
using MapMagic.Nodes.MatrixGenerators;


namespace MapMagic.Nodes.GUI
{

 public static class CTSEditor
 {
  private static string[] textureNames;

        [UnityEditor.InitializeOnLoadMethod]
  static void EnlistInMenu ()
  {
```

```
  CreateRightClick.generatorTypes.Add(typeof(CTSOutput200));
 }

 [Draw.Editor(typeof(MatrixGenerators.CTSOutput200))]
 public static void DrawCTS (MatrixGenerators.CTSOutput200 gen)
 {
  #if CTS_PRESENT

  CTS.CTSProfile profile = GraphWindow.current.mapMagic?.Globals.ctsProfile as CTS.CTSProfile;

  if (GraphWindow.current.mapMagic != null)
   using (Cell.LineStd)
   {
    CTS.CTSProfile newProfile = Draw.ObjectField(profile, "Profile");
    if (profile != newProfile)
    {
     GraphWindow.current.mapMagic.Globals.ctsProfile = newProfile;
     profile = newProfile;
    }
   }
  else
   using (Cell.LinePx(18+18)) Draw.Label("Not assigned to current \nMapMagic object");

//   using (Cell.LineStd)
//    if (Draw.Button("Update Shader"))
//     CTS_UpdateShader(ctsProfile, MapMagic.instance.terrainSettings.material);
```

```
 //populating texture names
 if (profile != null)
 {
  List<CTS.CTSTerrainTextureDetails> textureDetails = profile.TerrainTextures;
  if (textureNames==null || textureNames.Length!=textureDetails.Count) textureNames = new string[textur
  textureNames.Process(i=>textureDetails[i].m_name);
 }

 #else
 using (Cell.LinePx(60))
            Draw.Helpbox("CTS doesn't seem to be installed, or CTS compatibility is not enabled in settings'
 #endif

 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlinl
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
}

private static void DrawCTSLayer (Generator tgen, int num)
{
 CTSOutput200 gen = (CTSOutput200)tgen;
 CTSOutput200.CTSLayer layer = gen.layers[num];
 if (layer == null) return;

 #if CTS_PRESENT
 CTS.CTSProfile profile = GraphWindow.current.mapMagic?.Globals.ctsProfile as CTS.CTSProfile;
```

```
#endif

Cell.EmptyLinePx(3);
using (Cell.LinePx(28))
{
 //Cell.current.margins = new Padding(0,0,0,1); //1-pixel more padding from the bottom since layers are 1

 if (num!=0)
  using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, gen);
 else
  //disconnecting last layer inlet
  if (GraphWindow.current.graph.IsLinked(layer))
   GraphWindow.current.graph.UnlinkInlet(layer);

 Cell.EmptyRowPx(10);

 #if CTS_PRESENT

 //icon
 using (Cell.RowPx(28))
 {
  Texture2D icon = null;
  if (profile != null)
  {
   List<CTS.CTSTerrainTextureDetails> textureDetails = profile.TerrainTextures;
   if (layer.channelNum < textureDetails.Count)
```

```
   icon = textureDetails[layer.channelNum].Albedo;
 }
 Draw.TextureIcon(icon);
}

//channel selector
Cell.EmptyRowPx(5);
using (Cell.Row)
{
 Cell.EmptyLine();
 using (Cell.LineStd)
 {
  if (textureNames != null)
   Draw.PopupSelector(ref layer.channelNum, textureNames);
  else
   Draw.Field(ref layer.channelNum, "Channel");
 }
 Cell.EmptyLine();
}

#else
using (Cell.Row) Draw.Field(ref layer.channelNum, "Channel");
#endif

Cell.EmptyRowPx(10);
using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
```

```
 }
 Cell.EmptyLinePx(3);
}

      /*public static void DrawCTSShaderNameWarning ()
{
 Terrains.TerrainSettings settings = GraphWindow.current.mapMagic.terrainSettings;
 {
  using (Cell.LinePx(70))
  {
   //Cell.current.margins = new Padding(4);

   GUIStyle backStyle = UI.current.textures.GetElementStyle("DPUI/Backgrounds/Foldout");
   Draw.Element(backStyle);
   Draw.Element(backStyle);

   using (Cell.Row) Draw.Label("No CTS material \nis assigned as \nCustom Material in \nTerrain Setting

   using (Cell.RowPx(30))
    if (Draw.Button("Fix"))
    {
     Shader shader = Shader.Find("CTS/CTS Terrain Shader Basic");
     settings.material = new Material(shader);

     #if CTS_PRESENT
     if (ctsProfile != null) CTS_UpdateShader(ctsProfile, MapMagic.instance.terrainSettings.material);
```

```
        #endif

        GraphWindow.current.mapMagic.ApplyTerrainSettings();

        GraphWindow.RefreshMapMagic();
       }
     }
     Cell.EmptyLinePx(5);
    }
   }
*/
 }
}

#endif
```

```
ï»¿#if MAPMAGIC2 //shouldn't work if MM assembly not compiled

using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Products;
using MapMagic.Terrains;



namespace MapMagic.Nodes.MatrixGenerators
{
 [System.Serializable]
 [GeneratorMenu(
  menu = "Map/Output",
  name = "MegaSplat",
  section =2,
  drawButtons = false,
  colorType = typeof(MatrixWorld),
  iconName="GeneratorIcons/TexturesOut",
  helpLink = "https://gitlab.com/denispahunov/mapmagic/wikis/output_generators/Textures")]
 public class MegaSplatOutput200 : BaseTexturesOutput<MegaSplatOutput200.MegaSplatLayer>
 {
  public static float clusterNoiseScale = 0.05f;
```

```
private string[] clusterNames = new string[0];

public static bool smoothFallof = false;

//public Input wetnessIn = new Input(InoutType.Map);
//public Input puddlesIn = new Input(InoutType.Map);
//public Input displaceDampenIn = new Input(InoutType.Map);

public class MegaSplatLayer : BaseTextureLayer { } //inheriting empty to draw it's editor

public override void Generate (TileData data, StopToken stop)
{
 //generating
 MatrixWorld[] dstMatrices = BaseGenerate(data, stop);

 //adding to finalize
 if (stop!=null && stop.stop) return;
 if (enabled)
 {
  for (int i=0; i<layers.Length; i++)
   data.StoreOutput(layers[i], typeof(MegaSplatOutput200), layers[i],  dstMatrices[i]);
  data.MarkFinalize(Finalize, stop);
 }
 else
  data.RemoveFinalize(finalizeAction);
}
```

```
public override FinalizeAction FinalizeAction => finalizeAction; //should return variable, not create new
public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 #if __MEGASPLAT__
 if (data.globals.megaSplatTexList==null) return;

 //creating control textures
 if (stop!=null && stop.stop) return;
 data.GatherOutputs (typeof(MegaSplatOutput200),
  out MegaSplatLayer[] layers,
  out MatrixWorld[] matrices,
  out MatrixWorld[] masks,
  inSubs:true);
 float[] opacities = layers.Select(l=>l.Opacity);
 int[] channelNums = layers.Select(l=>l.channelNum);

 //purging if no outputs
 if (matrices.Length == 0)
 {
  if (stop!=null && stop.stop) return;
  data.MarkApply(CustomShaderOutput200.ApplyData.Empty);
  return;
 }
```

```
 Color[] controlColors = BlendMegaSplat(data.area, data.heights, data.globals.megaSplatTexList as Mega
  matrices, masks, opacities, channelNums);

 //pushing to apply
 if (stop!=null && stop.stop) return;
 //var applyData = new ApplyData() { colors = colors };
 var applyData = new CustomShaderOutput200.ApplyData()
 {
  textureColors = new Color[][] { controlColors },
  textureNames = new string[] { "_SplatControl" },
  textureFormat = TextureFormat.RGBA32
 };

 Graph.OnOutputFinalized?.Invoke(typeof(CustomShaderOutput200), data, applyData, stop);
 data.MarkApply(applyData);

 #endif
}


public override void ClearApplied (TileData data, Terrain terrain)
{

}
```

```
#if __MEGASPLAT__
public static Color[] BlendMegaSplat (Area area, Matrix heights, MegaSplatTextureList textureList,
 IList<Matrix> matrices, IList<Matrix> biomeMasks, IList<float> opacities, IList<int> channelNums,
 StopToken stop=null)
{
 int count = matrices.Count;
 CoordRect activeRect = area.active.rect;
 Color[] controlMap = new Color[activeRect.Count];

 //getting matrices rect
 CoordRect matrixRect = new CoordRect(0,0,0,0);
 for (int m=0; m<count; m++)
  if (matrices[m] != null) matrixRect = matrices[m].rect;

 //checking rect
 for (int m=0; m<count; m++)
  if (matrices[m] != null  &&  matrices[m].rect != matrixRect)
   throw new System.Exception("MapMagic: Matrix rect mismatch");
 for (int b=0; b<count; b++)
  if (biomeMasks[b] != null  &&  biomeMasks[b].rect != matrixRect)
   throw new System.Exception("MapMagic: Biome matrix rect mismatch");

 //preparing row re-use array
 float[] values = new float[count];

 //blending
```

```
for (int x=0; x<activeRect.size.x; x++)
 for (int z=0; z<activeRect.size.z; z++)
 {
  int matrixPosX = activeRect.offset.x + x;
  int matrixPosZ = activeRect.offset.z + z;
  int matrixPos = (matrixPosZ-matrixRect.offset.z)*matrixRect.size.x + matrixPosX - matrixRect.offset.x;

  int colorsPos = z*activeRect.size.x + x; //(z-colorsRect.offset.z)*colorsRect.size.x + x - colorsRect.offset

  // find highest two layers
  int botOutputIdx = 0;
  int topOutputIdx = 0;
  float botWeight = 0;
  float topWeight = 0;

  for (int i = 0; i<count; i++)
  {
   //value
   float val = matrices[i].arr[matrixPos];

   //multiply with biome
   Matrix biomeMask = biomeMasks[i];
   if (biomeMask != null) //no empty biomes in list (so no mask == root biome)
    val *= biomeMask.arr[matrixPos]; //if mask is not assigned biome was ignored, so only main outs with
```

```
 //clamp
 if (val < 0) val = 0; if (val > 1) val = 1;

 //finding if it's highest
 if (val > botWeight)
 {
  topWeight = botWeight;
  topOutputIdx = botOutputIdx;

  botWeight = val;
  botOutputIdx = i;
 }

 //or 2nd highest
 else if (val > topWeight)
 {
  topOutputIdx = i;
  topWeight = val;
 }
}

//converting layer index to texture index
int topClusterIdx = channelNums[topOutputIdx];
int botClusterIdx = channelNums[botOutputIdx];
```

```
Vector3 worldPos = area.active.CoordToWorld(x,z);
float heightRatio = heights!=null? heights.arr[matrixPos] : 0.5f; //0 is the bottom point, 1 is the maximum
Vector3 normal = new Vector3(0,1,0); //TODO: get normal from matrix

int topTexIdx = textureList.clusters[topClusterIdx].GetIndex(worldPos * clusterNoiseScale, normal, heigh
int botTexIdx = textureList.clusters[botClusterIdx].GetIndex(worldPos * clusterNoiseScale, normal, heigh

//swapping indexes to make topIdx always on top
/*if (botIdx > topIdx)
{
 int tempIdx = topIdx;
 topIdx = botIdx;
 botIdx = tempIdx;

 float tempWeight = topWeight;
 topWeight = botWeight;
 botWeight = tempWeight;
}*/

//finding blend
float totalWeight = topWeight + botWeight; if (totalWeight<0.01f) totalWeight = 0.01f; //Mathf.Max and C
float blend = botWeight / totalWeight;  if (blend>1) blend = 1;

//adjusting blend curve
if (smoothFallof) blend = (Mathf.Sqrt(blend) * (1-blend)) + blend*blend*blend;  //Magic secret formula! In
```

```
    //setting color
    controlMap[colorsPos] = new Color(botTexIdx / 255.0f, topTexIdx / 255.0f, 1.0f - blend, 1.0f);

    //params
    /*for (int i = 0; i<specialCount; i++)
    {
     float biomeVal = specialBiomeMasks[i]!=null? specialBiomeMasks[i].array[pos] : 1;

     if (specialWetnessMatrices[i]!=null) result.param[pos].b = specialWetnessMatrices[i].array[pos] * biome
     if (specialPuddlesMatrices[i]!=null)
     {
      result.param[pos].a = specialPuddlesMatrices[i].array[pos] * biomeVal;
      result.param[pos].r = 0.5f;
      result.param[pos].g = 0.5f;
     }
     if (specialDampeningMatrices[i]!=null) result.control[pos].a = specialDampeningMatrices[i].array[pos] *
    }*/
   }

  return controlMap;
 }
 #endif
}


}
```

```
#endif //MAPMAGIC2
```

```
ï»¿#if MAPMAGIC2 //shouldn't work if MM assembly not compiled

using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Core;  //used once to get tile size
using MapMagic.Products;
using MapMagic.Nodes.MatrixGenerators;


namespace MapMagic.Nodes.GUI
{

 public static class MegaSplatEditor
 {
  [UnityEditor.InitializeOnLoadMethod]
  static void EnlistInMenu ()
  {
   CreateRightClick.generatorTypes.Add(typeof(MegaSplatOutput200));
  }
```

```
[Draw.Editor(typeof(MegaSplatOutput200))]
public static void DrawMegaSplat (MegaSplatOutput200 gen)
{
 #if !__MEGASPLAT__
 using (Cell.LinePx(60))
  Draw.Helpbox("MegaSplat doesn't seem to be installed, or MicroSplat compatibility is not enabled in sett
 #endif
 if (GraphWindow.current.mapMagic != null  &&  GraphWindow.current.mapMagic is MapMagicObject ma
 {
  using (Cell.LineStd)
  {
   GeneratorDraw.DrawGlobalVar(ref mapMagicObject.terrainSettings.material, "Material");

   if (Cell.current.valChanged)
    mapMagicObject.ApplyTerrainSettings();
  }

  using (Cell.LineStd)
  {
   #if __MEGASPLAT__
   MegaSplatTextureList texList = mapMagicObject.globals.megaSplatTexList as MegaSplatTextureList;
   GeneratorDraw.DrawGlobalVar(ref texList, "TexList");
   mapMagicObject.globals.megaSplatTexList = texList;
   #endif
  }
```

```
 }
 else
  using (Cell.LinePx(18+18)) Draw.Label("Not assigned to current \nMapMagic object");

 using (Cell.LinePx(0)) CheckShader(gen);

 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlinl
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
}

private static void DrawCTSLayer (Generator tgen, int num)
{
 MegaSplatOutput200 gen = (MegaSplatOutput200)tgen;
 MegaSplatOutput200.MegaSplatLayer layer = gen.layers[num];
 if (layer == null) return;

 #if __MEGASPLAT__
 MegaSplatTextureList textureList = GraphWindow.current.mapMagic?.Globals.megaSplatTexList as Meg
 #endif

 Cell.EmptyLinePx(3);
 using (Cell.LinePx(28))
 {
  //Cell.current.margins = new Padding(0,0,0,1); //1-pixel more padding from the bottom since layers are 1

  if (num!=0)
```

```
 using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, gen);
else
 //disconnecting last layer inlet
 if (GraphWindow.current.graph.IsLinked(layer))
  GraphWindow.current.graph.UnlinkInlet(layer);

Cell.EmptyRowPx(10);

//icon
Texture2DArray icon = null;
int index = -2;

Material material = null;
if (GraphWindow.current.mapMagic != null  &&  GraphWindow.current.mapMagic is MapMagicObject ma
 material = mapMagicObject.terrainSettings.material;

if (material != null && material.HasProperty("_Diffuse"))
 icon = (Texture2DArray)material?.GetTexture("_Diffuse");

#if __MEGASPLAT__
if (textureList != null)
 //icon = textureList.clusters[num].previewTex; //preview textures doesnt seem to be working in recent ve
 index = textureList.clusters[num].indexes[0];
#endif

using (Cell.RowPx(28))
```

```
{
 if (icon != null && index >= 0)
  Draw.TextureIcon(icon, index);
}

//channel
Cell.EmptyRowPx(3);
using (Cell.Row)
{
 Cell.EmptyLine();
 using (Cell.LineStd)
 {
  Cell.current.fieldWidth = 0.4f;
  #if __MEGASPLAT__
  if (textureList!=null)
   Draw.PopupSelector(ref layer.channelNum, textureList.textureNames);
  else
   Draw.Field(ref layer.channelNum, "Channel");
  #else
  Draw.Field(ref layer.channelNum, "Channel");
  #endif
 }
 Cell.EmptyLine();
}

Cell.EmptyRowPx(10);
```

```
  using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
 }
 Cell.EmptyLinePx(3);
}



public static void CheckShader (MegaSplatOutput200 gen)
{
 if (GraphWindow.current.mapMagic == null  ||  !(GraphWindow.current.mapMagic is MapMagicObject ma

 Material mat = mapMagicObject.terrainSettings.material;
 if (mat==null || !mat.shader.name.Contains("MegaSplat"))
 {
  using (Cell.LinePx(50))
   using (Cell.Padded(3))
    Draw.Helpbox("The assigned material is not MegaSplat", UnityEditor.MessageType.Error);
 }
}

public static void CheckCustomSplatmaps (MegaSplatOutput200 gen)
{
 if (GraphWindow.current.mapMagic == null  ||  !(GraphWindow.current.mapMagic is MapMagicObject ma

 Material mat = mapMagicObject.terrainSettings.material;
 if (mat != null && !mat.HasProperty("_CustomControl0"))
 {
```

```
    using (Cell.LinePx(60))
     using (Cell.Padded(3))
      Draw.Helpbox("Use Custom Splatmaps is not enabled in the material", UnityEditor.MessageType.Error
   }
  }
 }
}

#endif
```

```
ï»¿#if MAPMAGIC2 //shouldn't work if MM assembly not compiled

using System;
using System.Collections;
using System.Collections.Generic;

using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Products;

#if __MICROSPLAT__
using JBooth.MicroSplat;
#endif

namespace MapMagic.Nodes.MatrixGenerators {
 [System.Serializable]
 [GeneratorMenu(
  menu = "Map/Output",
  name = "MicroSplat",
  section =2,
  drawButtons = false,
  colorType = typeof(MatrixWorld),
  iconName="GeneratorIcons/TexturesOut",
  helpLink = "https://gitlab.com/denispahunov/mapmagic/wikis/output_generators/Textures")]
```

```
public class MicroSplatOutput200 : BaseTexturesOutput<MicroSplatOutput200.MicroSplatLayer>
{
      //public static Material material;  //in globals
 //public static MicroSplatPropData propData;
 //public static bool assignComponent;

      public class MicroSplatLayer : BaseTextureLayer
{
  [NonSerialized] public TerrainLayer prototype = null; //used in case 'add std' enabled
 }


 public override void Generate (TileData data, StopToken stop)
{
  //generating
  MatrixWorld[] dstMatrices = BaseGenerate(data, stop);

  //adding to finalize
  if (stop!=null && stop.stop) return;
  if (enabled)
  {
   for (int i=0; i<layers.Length; i++)
    data.StoreOutput(layers[i], typeof(MicroSplatOutput200), layers[i],  dstMatrices[i]);
   data.MarkFinalize(Finalize, stop);
  }
  else
```

```
  data.RemoveFinalize(finalizeAction);
}

public override FinalizeAction FinalizeAction => finalizeAction; //should return variable, not create new
public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
        #if __MICROSPLAT__

 //creating control textures contents
 if (stop!=null && stop.stop) return;
 data.GatherOutputs (typeof(MicroSplatOutput200),
  out MicroSplatLayer[] layers,
  out MatrixWorld[] matrices,
  out MatrixWorld[] masks,
  inSubs:true);
 int[] channelNums = layers.Select(l=> l!=null ? l.channelNum : 0);
 float[] opacities = layers.Select(l=> l!=null ? l.Opacity : 0);

 //purging if no outputs
 if (matrices.Length == 0)
 {
  if (stop!=null && stop.stop) return;
  data.MarkApply(CustomShaderOutput200.ApplyData.Empty);
  return;
 }
```

```
IApplyData applyData = null;


//sorting matrices according to channel numbers
//don't sort beforehand since some matrices can take same channel num - in case of biomes
/*int maxChannel = 0;
for (int i=0; i<layers.Length; i++)
 if (layers[i].channelNum > maxChannel) maxChannel = layers[i].channelNum;

MatrixWorld[] sortedMatrices = new MatrixWorld[maxChannel+1];
MatrixWorld[] sortedMasks = new MatrixWorld[maxChannel+1];
MicroSplatLayer[] sortedLayers = new MicroSplatLayer[maxChannel+1];

for (int i=0; i<layers.Length; i++)
{
 int chNum = layers[i].channelNum;
 sortedLayers[chNum] = layers[i];  sortedMatrices[chNum] = matrices[i];  sortedMasks[chNum] = masks[i]
}*/


//custom splatmaps
if (data.globals.microSplatApplyType==Core.Globals.MicroSplatApplyType.Textures  ||
 data.globals.microSplatApplyType==Core.Globals.MicroSplatApplyType.Both)
{
 if (stop!=null && stop.stop) return;
 Color[][] colors = CustomShaderOutput200.BlendMatrices(data.area.active.rect, matrices, masks, opacit
```

```
string[] names = new string[colors.Length];
for (int i=0; i<names.Length; i++)
 names[i] = (data.globals.useCustomControlTextures ? "_CustomControl" : "_Control") + i.ToString();

//custom normal map
if (data.globals.microSplatNormals)
{
 (Matrix r, Matrix g, Matrix b) = MatrixOps.NormalsSet(data.heights, data.area.PixelSize.x, data.globals.h
 Color[] normColors = CustomShaderOutput200.MatricesToColors(data.area.active.rect, r, g, b, null);

 ArrayTools.Add(ref colors, normColors);
 ArrayTools.Add(ref names, "_PerPixelNormal");
}

if (stop!=null && stop.stop) return;
applyData = new ApplyCustomData()
{
 textureColors = colors,
 textureNames = names,
 textureFormat = TextureFormat.RGBA32,
 assignComponent = data.globals.assignComponent,
 propData = data.globals.microSplatPropData as MicroSplatPropData,
};

Graph.OnOutputFinalized?.Invoke(typeof(CustomShaderOutput200), data, applyData, stop);
data.MarkApply(applyData);
```

```
}

//standard splatmaps
if (data.globals.microSplatApplyType==Core.Globals.MicroSplatApplyType.Splats  ||
 data.globals.microSplatApplyType==Core.Globals.MicroSplatApplyType.Both)
{



 if (stop!=null && stop.stop) return;
 float[,,] splats3D = TexturesOutput200.BlendLayers(matrices, masks, data.area, channelNumbers:chann

 TerrainLayer[] tlayers = new TerrainLayer[splats3D.GetLength(2)];
 for (int i=0; i<layers.Length; i++)
 {
  int chNum = layers[i].channelNum;
  tlayers[chNum] = layers[i]?.prototype;
 }

 applyData = new ApplySplatsData() {
  splats = splats3D,
  prototypes = tlayers,
  assignComponent = data.globals.assignComponent,
  propData = data.globals.microSplatPropData as MicroSplatPropData};

 Graph.OnOutputFinalized?.Invoke(typeof(CustomShaderOutput200), data, applyData, stop);
 data.MarkApply(applyData);
```

```
 }

        #endif
}

public override void ClearApplied (TileData data, Terrain terrain)
{

}

public class ApplySplatsData : TexturesOutput200.ApplyData
{
 #if __MICROSPLAT__

 public bool assignComponent;
 public MicroSplatPropData propData;

 public override void Apply (Terrain terrain)
 {
  //checking microsplat component
  //this should be done before applying control since
  //microsplat removes template from terrain on disable (lod switch), so ensuring we have a material befor
  MicroSplatTerrain mso = null;
```

```
  if (assignComponent)
  {
   mso = CheckAssignMicroSplat(terrain);
   mso.propData = propData;
  }
  else if (terrain.materialTemplate == null) //prevents an error (materialTemplate is null) on disabling "Set C
  {
   MapMagic.Core.MapMagicObject mapMagic = terrain.transform.parent.parent.GetComponent<MapMag
   terrain.materialTemplate = mapMagic.terrainSettings.material;
  }

  base.Apply(terrain);

  if (assignComponent)
   mso.Sync();
 }

 #endif
}


public class ApplyCustomData : CustomShaderOutput200.ApplyData
{
 #if __MICROSPLAT__

 //in CustomShaderOutput200.ApplyData
```

```
//public Color[][] textureColors;
//public string[] textureNames;
//public string[] altTextureNames= null;

public bool assignComponent;
public MicroSplatPropData propData;
public Material materialTemplate; //source material assigned. Can't use terrain.materialTemplate since it

public override void Apply (Terrain terrain)
{
 //checking microsplat component
 //this should be done before applying control since
 //microsplat removes template from terrain on disable (lod switch), so ensuring we have a material befor
 MicroSplatTerrain mso = null;
 if (assignComponent)
 {
  mso = CheckAssignMicroSplat(terrain);
  mso.propData = propData;
 }
 else if (terrain.materialTemplate == null) //prevents an error (materialTemplate is null) on disabling "Set C
 {
  MapMagic.Core.MapMagicObject mapMagic = terrain.transform.parent.parent.GetComponent<MapMag
  terrain.materialTemplate = mapMagic.terrainSettings.material;
 }

 base.Apply(terrain);
```

```
 if (assignComponent)
  mso.Sync(); //this will create basemap and probably other useful stuff
}



#if UNITY_EDITOR
[UnityEditor.InitializeOnLoadMethod]
#endif
[RuntimeInitializeOnLoadMethod]
static void Subscribe ()
{
 MapMagic.Terrains.Weld.ReadEdgesCustom += ReadEdges;
 MapMagic.Terrains.Weld.WriteEdgesCustom += WriteEdges;
}

public static void ReadEdges (TileData thisData, MapMagic.Terrains.EdgesSet thisEdges)
{
 ApplyCustomData texturesData = thisData.ApplyOfType<ApplyCustomData>();
 if (texturesData != null && texturesData.textureColors!=null)
 {
  int numChs = texturesData.textureColors.Length * 4;
  if (thisEdges.controlEdges==null || thisEdges.controlEdges.Length != numChs)
   Array.Resize(ref thisEdges.controlEdges, numChs);

  for (int t=0; t<texturesData.textureColors.Length; t++)
```

```
   for (int i=0; i<4; i++)
  {
   int ch = t*4 + i;

   if (thisEdges.controlEdges[ch] == null)
    thisEdges.controlEdges[ch] = new MapMagic.Terrains.Edges(0,0);

   thisEdges.controlEdges[ch].ReadColors(texturesData.textureColors[t], i);
  }
 }
}

public static void WriteEdges (TileData thisData, MapMagic.Terrains.EdgesSet thisEdges)
{
 ApplyCustomData texturesData = thisData.ApplyOfType<ApplyCustomData>();
 if (texturesData != null && texturesData.textureColors!=null)
 {
  int numChs = texturesData.textureColors.Length * 4;
  if (thisEdges.controlEdges==null || thisEdges.controlEdges.Length != numChs)
   Array.Resize(ref thisEdges.controlEdges, numChs);

  for (int t=0; t<texturesData.textureColors.Length; t++)
   for (int i=0; i<4; i++)
  {
   int ch = t*4 + i;
   thisEdges.controlEdges[ch].WriteColors(texturesData.textureColors[t], i);
```

```
   }
  }
 }

 #endif
}


#if __MICROSPLAT__
public static MicroSplatTerrain CheckAssignMicroSplat (Terrain terrain)
{

 MicroSplatTerrain mso = terrain.GetComponent<MicroSplatTerrain>();
 if (mso == null) mso = terrain.gameObject.AddComponent<MicroSplatTerrain>();
 mso.terrain = terrain; //otherwise nullref on newly created tiles

 MapMagic.Core.MapMagicObject mapMagic = terrain.transform.parent.parent.GetComponent<MapMagi
 mso.templateMaterial = mapMagic.terrainSettings.material;
 if (terrain.materialTemplate == mso.templateMaterial || terrain.materialTemplate==null) //if material instan
 {
  mso.matInstance = new Material(mapMagic.terrainSettings.material);
  terrain.materialTemplate = mso.matInstance;
 }
 else
  mso.matInstance = terrain.materialTemplate;
```

```
 if (mso.keywordSO == null)
  mso.keywordSO = new MicroSplatKeywords();

 return mso;
}
#endif



public class TmpApplyData// : IApplyData
{
        #if __MICROSPLAT__

 public Color[][] colors; // TODO: use raw texture bytes

 public void Read (Terrain terrain) { throw new System.NotImplementedException(); }

 public void ApplyTmp (Terrain terrain)
 {
  //checking microsplat component
  MicroSplatTerrain mso = terrain.GetComponent<MicroSplatTerrain>();
  if (mso == null) mso = terrain.gameObject.AddComponent<MicroSplatTerrain>();
  mso.terrain = terrain; //otherwise nullref on newly created tiles
  mso.templateMaterial = terrain.materialTemplate;


  int numTextures = colors.Length;
```

```
if (numTextures==0) return;
int resolution = (int)Mathf.Sqrt(colors[0].Length);

for (int t=0; t<numTextures; t++)
{
 if (colors[t] == null) continue;

 Texture2D tex = GetTex(mso, t);
 if (tex==null || tex.width!=resolution || tex.height!=resolution || tex.format!=TextureFormat.RGBA32)
 {
  if (tex!=null)
  {
   #if UNITY_EDITOR
   if (!UnityEditor.AssetDatabase.Contains(tex))
   #endif
    GameObject.DestroyImmediate(tex);
  }

  tex = new Texture2D(resolution, resolution, TextureFormat.RGBA32, false, true);
  tex.wrapMode = TextureWrapMode.Mirror; //to avoid border seams
  tex.name = "CustomControl " + t;
  SetTex(mso, t, tex);
  //tex.hideFlags = HideFlags.DontSave;
 }

 tex.SetPixels(0,0, tex.width,tex.height, colors[t]);
```

```
  tex.Apply();
 }

 mso.Sync();
 //terrain.basemapDistance = 1000000;
}

public Texture2D GetTex (MicroSplatTerrain mso, int num)
{
 switch (num)
 {
  case 0: return mso.customControl0;
  case 1: return mso.customControl1;
  case 2: return mso.customControl2;
  case 3: return mso.customControl3;
  case 4: return mso.customControl4;
  case 5: return mso.customControl5;
  case 6: return mso.customControl6;
  case 7: return mso.customControl7;
  default: return null;
 }
}

public void SetTex (MicroSplatTerrain mso, int num, Texture2D tex)
{
 switch (num)
```

```
    {
     case 0: mso.customControl0 = tex; break;
     case 1: mso.customControl1 = tex; break;
     case 2: mso.customControl2 = tex; break;
     case 3: mso.customControl3 = tex; break;
     case 4: mso.customControl4 = tex; break;
     case 5: mso.customControl5 = tex; break;
     case 6: mso.customControl6 = tex; break;
     case 7: mso.customControl7 = tex; break;
    }
   }

   public static TmpApplyData Empty {get{ return new TmpApplyData() { colors = new Color[0][] }; }}

   public int Resolution
   {get{
    if (colors.Length==0) return 0;
    else return (int)Mathf.Sqrt(colors[0].Length);
   }}

          #endif
  }
 }
}

#endif //MAPMAGIC2
```

```
ï»¿#if MAPMAGIC2 //shouldn't work if MM assembly not compiled

using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Core;  //used once to get tile size
using MapMagic.Products;
using MapMagic.Nodes.MatrixGenerators;

#if __MICROSPLAT__
using JBooth.MicroSplat;
#endif

namespace MapMagic.Nodes.GUI
{

 public static class MicroSplatEditor
 {
  private static Texture2DArray lastIconsArr = null; //caching texture array to avoid fetching from MS each fr
  private static Material lastMicroSplatMat = null;
```

```
[UnityEditor.InitializeOnLoadMethod]
static void EnlistInMenu ()
{
 CreateRightClick.generatorTypes.Add(typeof(MicroSplatOutput200));
}



[Draw.Editor(typeof(MicroSplatOutput200))]
public static void DrawMicroSplat (MicroSplatOutput200 gen)
{
 #if !__MICROSPLAT__
 using (Cell.LinePx(60))
  Draw.Helpbox("MicroSplat doesn't seem to be installed, or MicroSplat compatibility is not enabled in sett
 #endif
 if (GraphWindow.current.mapMagic != null  &&  GraphWindow.current.mapMagic is MapMagicObject ma
  using (Cell.LineStd)
  {
   //Cell.current.fieldWidth = 0.5f;
   using (Cell.LineStd)
    GeneratorDraw.DrawGlobalVar(ref mapMagicObject.terrainSettings.material, "Material");

   using (Cell.LinePx(0))
    using (new Draw.FoldoutGroup(ref gen.guiAdvanced, "Advanced"))
    if (gen.guiAdvanced)
    {
```

```
using (Cell.LineStd)
{
 Cell.current.fieldWidth = 0.15f;
 GeneratorDraw.DrawGlobalVar(ref GraphWindow.current.mapMagic.Globals.assignComponent, "Set
}

#if __MICROSPLAT__
if (GraphWindow.current.mapMagic.Globals.assignComponent)
 using (Cell.LineStd)
  GraphWindow.current.mapMagic.Globals.microSplatPropData = GeneratorDraw.DrawGlobalVar<Mic
   GraphWindow.current.mapMagic.Globals.microSplatPropData==null ? null : (MicroSplatPropData)m
   "PropData");
#endif

//using (Cell.LineStd)
// Draw.Label("Apply Type");

using (Cell.LineStd)
 GeneratorDraw.DrawGlobalVar(ref mapMagicObject.globals.microSplatApplyType, "Apply Type");

using (Cell.LineStd)
{
 Cell.current.fieldWidth = 0.15f;
 Cell.current.disabled = mapMagicObject.globals.microSplatApplyType==Globals.MicroSplatApplyTyp
 GeneratorDraw.DrawGlobalVar(ref mapMagicObject.globals.useCustomControlTextures, "Custom Sp
}
```

```
    using (Cell.LineStd)
    {
     Cell.current.fieldWidth = 0.15f;
     Cell.current.disabled = mapMagicObject.globals.microSplatApplyType==Globals.MicroSplatApplyTyp
     GeneratorDraw.DrawGlobalVar(ref mapMagicObject.globals.microSplatNormals, "Add Normals Tex");
    }

    //using (Cell.LineStd)
    // { Cell.current.fieldWidth = 0.15f; GeneratorDraw.DrawGlobalVar(ref GraphWindow.current.mapMagic
   }

  if (Cell.current.valChanged)
   mapMagicObject.ApplyTerrainSettings();
 }
 else
  using (Cell.LinePx(18+18)) Draw.Label("Not assigned to current \nMapMagic object");

 using (Cell.LinePx(0)) CheckShader(gen);
 //using (Cell.LinePx(0)) CheckCustomSplatmaps(gen);

 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlinl
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
}

private static void DrawMicroSplatLayer (Generator tgen, int num)
```

```
{
 MicroSplatOutput200 gen = (MicroSplatOutput200)tgen;
 MicroSplatOutput200.MicroSplatLayer layer = gen.layers[num];
 if (layer == null) return;

 Material microSplatMat = null;
 if (GraphWindow.current.mapMagic != null   &&  GraphWindow.current.mapMagic is MapMagicObject mm
  microSplatMat = mmo.terrainSettings.material;

 Cell.EmptyLinePx(3);
 using (Cell.LinePx(28))
 {
  //Cell.current.margins = new Padding(0,0,0,1); //1-pixel more padding from the bottom since layers are 1

  if (num!=0)
   using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, gen);
  else
   //disconnecting last layer inlet
   if (GraphWindow.current.graph.IsLinked(layer))
    GraphWindow.current.graph.UnlinkInlet(layer);

  Cell.EmptyRowPx(10);

  //icon
  if (microSplatMat != lastMicroSplatMat  &&  microSplatMat != null  &&  microSplatMat.HasProperty("_Dif
  {
```

```
 lastIconsArr = (Texture2DArray)microSplatMat?.GetTexture("_Diffuse");
 lastMicroSplatMat = microSplatMat;
}

using (Cell.RowPx(28))
{
 if (lastIconsArr != null)
  Draw.TextureIcon(lastIconsArr, layer.channelNum);
}

Cell.EmptyRowPx(10);

//index
using (Cell.Row)
{
 Cell.EmptyLine();
 using (Cell.LineStd)
 {
  int newIndex = Draw.Field(layer.channelNum, "Index");
  if (newIndex >= 0 && (lastIconsArr==null || newIndex < lastIconsArr.depth))
  {
   layer.channelNum = newIndex;
   layer.prototype = null;
  }
 }
 Cell.EmptyLine();
```

```
  }

  //terrain layer (if enabled)
  if (GraphWindow.current.mapMagic!=null  &&
   GraphWindow.current.mapMagic is MapMagicObject mapMagicObject  &&
   mapMagicObject.globals.microSplatApplyType!=Globals.MicroSplatApplyType.Textures) //no need to cr
  {
   TerrainLayer tlayer = layer.prototype;
   if (tlayer == null)
    { tlayer = new TerrainLayer(); layer.prototype = tlayer; }
   if (tlayer.diffuseTexture == null)
    tlayer.diffuseTexture = lastIconsArr.GetTexture(num);
  }

  Cell.EmptyRowPx(10);
  using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
 }
 Cell.EmptyLinePx(3);
}

public static void CheckShader (MicroSplatOutput200 gen)
{
 if (GraphWindow.current.mapMagic == null   ||  !(GraphWindow.current.mapMagic is MapMagicObject ma

 Material mat = mapMagicObject.terrainSettings.material;
 if (mat==null || !mat.shader.name.Contains("MicroSplat"))
```

```
  {
    using (Cell.LinePx(50))
      using (Cell.Padded(3))
        Draw.Helpbox("The assigned material is not MicroSplat", UnityEditor.MessageType.Error);
  }
 }

 public static void CheckCustomSplatmaps (MicroSplatOutput200 gen)
 {
  if (GraphWindow.current.mapMagic == null   ||  !(GraphWindow.current.mapMagic is MapMagicObject ma

  Material mat = mapMagicObject.terrainSettings.material;
  if (mat != null && !mat.HasProperty("_CustomControl0"))
  {
    using (Cell.LinePx(60))
      using (Cell.Padded(3))
        Draw.Helpbox("Use Custom Splatmaps is not enabled in the material", UnityEditor.MessageType.Error
  }
 }
}
}

#endif
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;

using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Products;

#if !__MICROSPLAT__
namespace MapMagic.Nodes.MatrixGenerators {
 [System.Serializable]
 [GeneratorMenu(
  menu = "Map/Output",
  name = "MicroSplat",
  section =2,
  drawButtons = false,
  colorType = typeof(MatrixWorld),
  iconName="GeneratorIcons/TexturesOut",
  helpLink = "https://gitlab.com/denispahunov/mapmagic/wikis/output_generators/Textures")]
 public class MicroSplatOutput200 : BaseTexturesOutput<MicroSplatOutput200.MicroSplatLayer>
 {

  public override void Generate (TileData data, StopToken stop) { }
  public override void ClearApplied (TileData data, Terrain terrain) { }
```

```
        public class MicroSplatLayer : BaseTextureLayer
  {
   [NonSerialized] public TerrainLayer prototype = null; //used in case 'add std' enabled
  }

  public override FinalizeAction FinalizeAction => finalizeAction; //should return variable, not create new
  public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
  public static void Finalize (TileData data, StopToken stop) { }
 }
}
#endif
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Core;  //used once to get tile size
using MapMagic.Products;
using MapMagic.Nodes.MatrixGenerators;

#if !__MICROSPLAT__
namespace MapMagic.Nodes.GUI
{

 public static class MicroSplatEditor
 {
  [Draw.Editor(typeof(MicroSplatOutput200))]
  public static void DrawMicroSplat (MicroSplatOutput200 gen)
  {
   using (Cell.LinePx(60))
    Draw.Helpbox("MicroSplat doesn't seem to be installed, or MicroSplat compatibility is not enabled in sett

   using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlinl
```

```
   using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
  }

  private static void DrawMicroSplatLayer (Generator tgen, int num)
  {
   MicroSplatOutput200 gen = (MicroSplatOutput200)tgen;
   MicroSplatOutput200.MicroSplatLayer layer = gen.layers[num];
   if (layer == null) return;



   Cell.EmptyLinePx(3);
   using (Cell.LinePx(28))
   {
    if (num!=0)
     using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, gen);

    Cell.EmptyRow();

    using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
   }
   Cell.EmptyLinePx(3);
  }
 }
}
#endif
```

```
ï»¿#if MAPMAGIC2 //shouldn't work if MM assembly not compiled

using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Products;

namespace MapMagic.Nodes.MatrixGenerators
{

 [System.Serializable]
 [GeneratorMenu(
  menu = "Map/Output",
  name = "RTP",
  section =2,
  drawButtons = false,
  colorType = typeof(MatrixWorld),
  iconName = "GeneratorIcons/TexturesOut",
  helpLink = "https://gitlab.com/denispahunov/mapmagic/wikis/output_generators/Textures")]
 public class RTPOutput200 : BaseTexturesOutput<RTPOutput200.RTPLayer>
 {
  #if RTP
  public ReliefTerrain rtp = null;
  #endif
```

```
public class RTPLayer : BaseTextureLayer { }



public override void Generate (TileData data, StopToken stop)
{
 //generating
 MatrixWorld[] dstMatrices = BaseGenerate(data, stop);

 //adding to finalize
 if (stop!=null && stop.stop) return;
 if (enabled)
 {
  for (int i=0; i<layers.Length; i++)
   data.StoreOutput(layers[i], typeof(RTPOutput200), layers[i],  dstMatrices[i]);
  data.MarkFinalize(Finalize, stop);
 }
 else
  data.RemoveFinalize(finalizeAction);
}



public override FinalizeAction FinalizeAction => finalizeAction; //should return variable, not create new
public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 //purging if no outputs
```

```
  if (data.OutputsCount(typeof(RTPOutput200), inSubs:true) == 0)
  {
   if (stop!=null && stop.stop) return;
   data.MarkApply(CustomShaderOutput200.ApplyData.Empty);
   return;
  }

  //creating control textures contents
  Color[][] colors = null; //TODO: re-use colors array
//   CustomShaderOutput200.BlendControlTextures(ref colors, typeof(RTPOutput200), data);

  //pushing to apply
  if (stop!=null && stop.stop) return;
  var controlTexturesData = new CustomShaderOutput200.ApplyData() {
   textureColors = colors,
   textureFormat = TextureFormat.RGBA32,
   textureBaseMapDistance = 10000000, //no base map
   textureNames = new string[colors!=null ? colors.Length : 0] };

  for (int t=0; t<controlTexturesData.textureNames.Length; t++)
   controlTexturesData.textureNames[t] = "_Control" + (t+1);

  Graph.OnOutputFinalized?.Invoke(typeof(RTPOutput200), data, controlTexturesData, stop);
  data.MarkApply(controlTexturesData);
 }
```

```
  public override void ClearApplied (TileData data, Terrain terrain)
  {


  }
 }


}
#endif
```

```
ï»¿#if MAPMAGIC2 //shouldn't work if MM assembly not compiled

using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Core;  //used once to get tile size
using MapMagic.Products;
using MapMagic.Nodes.MatrixGenerators;


namespace MapMagic.Nodes.GUI
{

 public static class RTPEditor
 {
  #if RTP
  static string[] textureNames = null;
  #endif

   [UnityEditor.InitializeOnLoadMethod]
```

```
static void EnlistInMenu ()
{
 CreateRightClick.generatorTypes.Add(typeof(RTPOutput200));
}

[Draw.Editor(typeof(RTPOutput200))]
public static void DrawRTP (RTPOutput200 gen)
{
 #if RTP
 GeneratorEditors.UpdateMaterial();
 gen.rtp = GraphWindow.current.mapMagic?.GetComponent<ReliefTerrain>();

 if (gen.rtp != null)
 {
  if (textureNames==null || textureNames.Length!=gen.rtp.globalSettingsHolder.numLayers) textureNames
  textureNames.Process(i=>gen.rtp.globalSettingsHolder.splats[i]!=null ? gen.rtp.globalSettingsHolder.spl
 }

 using (Cell.Line)
 {
  //Cell.current.margins = new Padding(4);
  //DrawCustomMaterialWarning(MapMagic.instance.terrainSettings);
  DrawRTPComponentWarning();
 }

 #else
```

```
 using (Cell.LinePx(36)) Draw.Label("RTP is not installed or RTP \ncompatibility is disabled");
 #endif

 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlinl
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
}

private static void DrawRTPLayer (Generator tgen, int num)
{
 RTPOutput200 gen = (RTPOutput200)tgen;
 RTPOutput200.RTPLayer layer = gen.layers[num];
 if (layer == null) return;

 using (Cell.LinePx(32))
 {
  //Cell.current.margins = new Padding(0,0,0,1); //1-pixel more padding from the bottom since layers are 1

  if (num!=0)
   using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, gen);
  else
   //disconnecting last layer inlet
   if (GraphWindow.current.graph.IsLinked(layer))
    GraphWindow.current.graph.UnlinkInlet(layer);

  Cell.EmptyRowPx(10);
```

```
//icon
#if RTP
Texture2D icon = null;
if (gen.rtp != null)
{
 if (layer.channelNum < gen.rtp.globalSettingsHolder.splats.Length)
  icon = gen.rtp.globalSettingsHolder.splats[layer.channelNum];

 using (Cell.RowPx(28))
 {
  Cell.EmptyLinePx(2);
  using (Cell.Line) Draw.TextureIcon(icon);
  Cell.EmptyLinePx(2);
 }

 //channel selector
 Cell.EmptyRowPx(3);
 using (Cell.Row)
 {
  Cell.EmptyLine();
  using (Cell.LineStd) Draw.PopupSelector(ref layer.channelNum, textureNames);
  Cell.EmptyLine();
 }
}
else
#endif
```

```
   using (Cell.Row)
   {
    Cell.EmptyLine();
    using (Cell.LineStd) Draw.Field(ref layer.channelNum, "Channel");
    Cell.EmptyLine();
   }

  Cell.EmptyRowPx(10);
  using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
 }
}



public static void DrawRTPComponentWarning ()
{
 #if RTP
 if (GraphWindow.current.mapMagic == null)
  return;

 if (GraphWindow.current.mapMagic?.gameObject.GetComponent<ReliefTerrain>()==null || GraphWindo
 {
  using (Cell.LinePx(70))
  {
   GUIStyle backStyle = UI.current.textures.GetElementStyle("DPUI/Backgrounds/Foldout");

   using (Cell.Row)
```

```
    Draw.Label("RTP or Renderer \ncomponents are \nnot assigned to \nMapMagic object");


   using (Cell.RowPx(30))
    if (Draw.Button("Fix"))
   {
    if (GraphWindow.current.mapMagic.gameObject.GetComponent<Renderer>() == null)
    {
     MeshRenderer renderer = GraphWindow.current.mapMagic.gameObject.AddComponent<MeshRende
     renderer.enabled = false;
    }
    if (GraphWindow.current.mapMagic.gameObject.GetComponent<ReliefTerrain>() == null)
    {
     ReliefTerrain rtp = GraphWindow.current.mapMagic.gameObject.AddComponent<ReliefTerrain>();


     //filling empty splats
     Texture2D emptyTex = TextureExtensions.ColorTexture(4,4,new Color(0.5f, 0.5f, 0.5f, 1f));
     emptyTex.name = "Empty";
     rtp.globalSettingsHolder.splats = new Texture2D[] { emptyTex,emptyTex,emptyTex,emptyTex };
    }
   }
  }
   Cell.EmptyLinePx(5);
 }
 #endif
}
}
```

```
}


#endif
```

```
ï»¿using System;
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes;
using MapMagic.Nodes.GUI;

#if VEGETATION_STUDIO_PRO
using AwesomeTechnologies.Common;
using AwesomeTechnologies.VegetationSystem;
#endif

namespace MapMagic.VegetationStudio
{
 public static class VSProMapsOutEditor
 {
  #if VEGETATION_STUDIO_PRO //otherwise will log a warning
  static string[] maskNames = null;
  #endif
```

```
static string[] channelNames = new string[] { "Red", "Green", "Blue", "Alpha" }; //should be readonly


public static string[] objectNames = null;



[UnityEditor.InitializeOnLoadMethod]
static void EnlistInMenu ()
{
 CreateRightClick.generatorTypes.Add(typeof(VSProMapsOut));
}



[Draw.Editor(typeof(VSProMapsOut))]
public static void DrawVSProMapsOut (VSProMapsOut gen)
{
 #if VEGETATION_STUDIO_PRO
 VegetationPackagePro package = null;

 Cell.current.fieldWidth = 0.49f;

 using (Cell.LinePx(0))
 using (Cell.Padded(1,1,0,0))
 {
  if (GraphWindow.current.mapMagic != null)
  {
   VegetationSystemPro system = GraphWindow.current.mapMagic.Globals.vegetationSystem as Vegeta
```

```
 using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(ref system, "System");
 GraphWindow.current.mapMagic.Globals.vegetationSystem = system;

 package = GraphWindow.current.mapMagic.Globals.vegetationPackage as VegetationPackagePro;
 using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(ref package, "Package");
 GraphWindow.current.mapMagic.Globals.vegetationPackage = package;
}
else
 using (Cell.LinePx(18+18)) Draw.Label("Not assigned to current \nMapMagic object");

//populating masks list to choose from
if (package != null)
{
 int masksCount = package.TextureMaskGroupList.Count;
 if (maskNames == null || maskNames.Length != masksCount)
  maskNames = new string[masksCount];

 for (int i=0; i<masksCount; i++)
 {
  maskNames[i] = (i + 1).ToString() + ". " +
   package.TextureMaskGroupList[i].TextureMaskName + " - " +
   package.TextureMaskGroupList[i].TextureMaskType;
 }
}

Cell.EmptyLinePx(4);
```

```
 using (Cell.LineStd)
 {
  if (package != null)
   Draw.PopupSelector(ref gen.maskGroup, maskNames, "Group");
  else
   Draw.Field(ref gen.maskGroup, "Group");
 }
 using (Cell.LineStd)
  Draw.PopupSelector(ref gen.textureChannel, channelNames, "Channel");
}



Cell.EmptyLinePx(3);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref gen.guiAdvanced, "Advanced", isLeft:false, padding:1))
  if (gen.guiAdvanced)
  {
   if (GraphWindow.current.mapMagic != null)
   using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(
    ref GraphWindow.current.mapMagic.Globals.vegetationSystemCopy,
    "Copy VS");

   using (Cell.LineStd) Draw.Field(ref gen.outputLevel, "Out Level");
  }
```

```
    #else

    using (Cell.LinePx(76))
     Draw.Helpbox("Vegetation Studio Pro doesn't seem to be installed, or Vegetation Studio Pro compatibili

    using (Cell.LineStd)
     Draw.Field(ref gen.maskGroup, "Group");
    using (Cell.LineStd)
     Draw.PopupSelector(ref gen.textureChannel, channelNames, "Channel");

    #endif


   }
  }
 }
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Products;
using MapMagic.Terrains;
using MapMagic.Nodes;



#if !VEGETATION_STUDIO_PRO
namespace MapMagic.VegetationStudio
{
 [System.Serializable]
 [GeneratorMenu(
  //menu = "Map/Output",
  name = "VS Pro Maps",
  section =2,
  drawButtons = false,
  colorType = typeof(MatrixWorld))]
 public class VSProMapsOut : OutputGenerator, IInlet<MatrixWorld>
 {
  public OutputLevel outputLevel = OutputLevel.Main;
```

```
  public override OutputLevel OutputLevel { get{ return outputLevel; } }

  //[Val("Package", type = typeof(VegetationPackagePro))] public VegetationPackagePro package; //in glob

  public float density = 0.5f;

  public int maskGroup = 0;
  public int textureChannel = 0;

  public override void Generate (TileData data, StopToken stop) { }
  public override void ClearApplied (TileData data, Terrain terrain) { }
 }
}
#endif
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Products;
using MapMagic.Terrains;
using MapMagic.Nodes;



#if VEGETATION_STUDIO_PRO
using AwesomeTechnologies.VegetationSystem;
using AwesomeTechnologies.VegetationStudio;
using AwesomeTechnologies.Vegetation.Masks;
using AwesomeTechnologies.Vegetation.PersistentStorage;
#endif

namespace MapMagic.VegetationStudio
{
 [System.Serializable]
 [GeneratorMenu(
  menu = "Map/Output",
  name = "VS Pro Maps",
```

```
 section =2,
 drawButtons = false,
 colorType = typeof(MatrixWorld),
 helpLink = "https://gitlab.com/denispahunov/mapmagic/wikis/output_generators/Grass")]
public class VSProMapsOut : OutputGenerator, IInlet<MatrixWorld>
{
 public OutputLevel outputLevel = OutputLevel.Main;
 public override OutputLevel OutputLevel { get{ return outputLevel; } }

 //[Val("Package", type = typeof(VegetationPackagePro))] public VegetationPackagePro package; //in glob

 public float density = 0.5f;

 public int maskGroup = 0;
 public int textureChannel = 0;


 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
  MatrixWorld src = data.ReadInletProduct(this);
  if (src == null) return;
  if (data.globals.vegetationSystem == null || data.globals.vegetationPackage == null) return;

  if (enabled)
  {
```

```
  data.StoreOutput(this, typeof(VSProMapsOut), this, src);  //adding src since it's not changing
  data.MarkFinalize(Finalize, stop);
 }
 else
  data.RemoveFinalize(finalizeAction);
}



public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 #if VEGETATION_STUDIO_PRO

 //creating splats and prototypes arrays
 int layersCount = data.OutputsCount(finalizeAction, inSubs:true);
 int splatsSize = data.area.active.rect.size.x - 1; //-1 is a resolution fix from forums:
   //https://forum.unity.com/threads/released-mapmagic-2-infinite-procedural-land-generator.875470/page-
   //seems to be working, but probably there are some cases it's not

 //preparing texture colors
 VegetationPackagePro package = data.globals.vegetationPackage as VegetationPackagePro;
 Color[][] colors = new Color[package!=null ? package.TextureMaskGroupList.Count : 0][];
 for (int c=0; c<colors.Length; c++)
  colors[c] = new Color[splatsSize*splatsSize];
 int[] maskGroupNums = new int[colors.Length];
```

```
//filling colors
int i=0;
foreach ((VSProMapsOut output, MatrixWorld matrix, MatrixWorld biomeMask)
 in data.Outputs<VSProMapsOut,MatrixWorld,MatrixWorld>(typeof(VSProMapsOut), inSubs:true))
{
 if (matrix == null || package==null) continue;
 BlendLayer(colors, data.area, matrix, biomeMask, output.density, output.maskGroup, output.textureCha
 maskGroupNums[output.maskGroup] = output.maskGroup; //TODO: removed i/4, test this! (http://mm2.i
 i++;
}

//pushing to apply
if (stop!=null && stop.stop) return;
ApplyData applyData = new ApplyData() {
 srcSystem=data.globals.vegetationSystem as VegetationSystemPro,
 package=package,
 colors=colors,
 maskGroupNums=maskGroupNums,
 copyVS= data.globals.vegetationSystemCopy  };
Graph.OnOutputFinalized?.Invoke(typeof(VSProMapsOut), data, applyData, stop);
data.MarkApply(applyData);

#endif
}
```

```
public static void BlendLayer (Color[][] colors, Area area, MatrixWorld matrix, MatrixWorld biomeMask, flo
{
 Color[] cols = colors[maskGroup];
 int splatsSize = area.active.rect.size.x - 1; //-1 is a resolution fix from forums:
   //https://forum.unity.com/threads/released-mapmagic-2-infinite-procedural-land-generator.875470/page-
   //seems to be working, but probably there are some cases it's not
 int fullSize = area.full.rect.size.x;
 int margins = area.Margins;

 for (int x=0; x<splatsSize; x++)
  for (int z=0; z<splatsSize; z++)
  {
   if (stop!=null && stop.stop) return;

   int matrixPos = (z+margins)*fullSize + (x+margins);

   float val = matrix.arr[matrixPos];

   if (biomeMask != null) //no empty biomes in list (so no mask == root biome)
    val *= biomeMask.arr[matrixPos]; //if mask is not assigned biome was ignored, so only main outs with r

   val *= opacity;

   if (val < 0) val = 0; if (val > 1) val = 1;

   int colsPos = z*splatsSize + x;
```

```
   switch (textureChannel)
   {
    case 0: cols[colsPos].r += val; break;
    case 1: cols[colsPos].g += val; break;
    case 2: cols[colsPos].b += val; break;
    case 3: cols[colsPos].a += val; break;
   }
  }
}



public override void ClearApplied (TileData data, Terrain terrain)
{
 VegetationSystemPro system = VSProOps.GetCopyVegetationSystem(terrain);
 if (system == null) system = data.globals.vegetationSystem as VegetationSystemPro;
 if (system == null) system = GameObject.FindObjectOfType<VegetationSystemPro>();



}

#if VEGETATION_STUDIO_PRO
public class ApplyData : IApplyData
{
 public VegetationSystemPro srcSystem;
 public VegetationPackagePro package;
 public Color[][] colors;
```

```
public int[] maskGroupNums;
public bool copyVS;

public void Read (Terrain terrain)  { throw new System.NotImplementedException(); }

public void Apply (Terrain terrain)
{
 //updating system
 VegetationSystemPro copySystem = null;  //we'll need it to set up tile
 if (copyVS)
 {
  copySystem = VSProOps.GetCopyVegetationSystem(terrain);
  if (copySystem == null) copySystem = VSProOps.CopyVegetationSystem(srcSystem, terrain.transform.
  VSProOps.UpdateCopySystem(copySystem, terrain, package, srcSystem);
 }

 else
  VSProOps.UpdateSourceSystem(srcSystem, terrain);

 //applying
 Texture2D[] textures = WriteTextures(null, colors);
 VSProOps.SetTextures(
  copyVS ? copySystem : srcSystem,
  package, textures, maskGroupNums, terrain.GetWorldRect());

 //tile obj (serialization and disable purpose)
```

```
 Transform tileTfm = terrain.transform.parent;
 VSProMapsTile vsTile = tileTfm.GetComponent<VSProMapsTile>();
 if (vsTile == null) vsTile = tileTfm.gameObject.AddComponent<VSProMapsTile>();
 vsTile.system = copyVS ? copySystem : srcSystem;
 vsTile.package = package;
 vsTile.terrainRect = terrain.GetWorldRect();
 vsTile.textures = textures;
 vsTile.maskGroupNums = maskGroupNums;
 vsTile.masksApplied = true;
}

public static ApplyData Empty
{get{
 return new ApplyData() {
  colors = new Color[0][],
  maskGroupNums = new int[0]  };
}}

public int Resolution
{get{
 if (colors.Length==0) return 0;
 else return (int)Mathf.Sqrt(colors[0].Length);
}}
}
```

```
public static Texture2D[] WriteTextures (Texture2D[] oldTextures, Color[][] colors)
{
 int numTextures = colors.Length;
 if (numTextures==0) return new Texture2D[0];
 int resolution = (int)Mathf.Sqrt(colors[0].Length);

 Texture2D[] textures = new Texture2D[numTextures];

 //making textures of colors in coroutine
 for (int i=0; i<numTextures; i++)
 {
  //trying to reuse last used texture
  Texture2D tex;
  if (oldTextures != null  &&
   i < oldTextures.Length  &&
   oldTextures[i] != null &&
   oldTextures[i].width == resolution &&
   oldTextures[i].height == resolution)
    tex = oldTextures[i];

  else
  {
   #if UNITY_EDITOR
   if (textures[i] != null && !UnityEditor.AssetDatabase.Contains(textures[i]))
   #endif
    GameObject.DestroyImmediate(textures[i]);
```

```
      tex = new Texture2D(resolution, resolution, TextureFormat.RGBA32, true, true);
      tex.wrapMode = TextureWrapMode.Mirror; //to avoid border seams
    }

    tex.SetPixels(0,0,tex.width,tex.height,colors[i]);
    tex.Apply();

    textures[i] = tex;
   }

   return textures;
  }



  #endif
 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using MapMagic.Terrains;

#if VEGETATION_STUDIO_PRO
using AwesomeTechnologies.VegetationStudio;
using AwesomeTechnologies.VegetationSystem;
using AwesomeTechnologies.Vegetation.Masks;
using AwesomeTechnologies.Vegetation.PersistentStorage;
using AwesomeTechnologies.Billboards;
#endif

namespace MapMagic.VegetationStudio
{
 [ExecuteInEditMode]
 public class VSProMapsTile : MonoBehaviour
 /// Helper script to automatically add/remove and serialize textures and persistent storage data
 /// This will clear package mask on moving terrain and playmode disable as well
 {
  #if VEGETATION_STUDIO_PRO

  public VegetationSystemPro system;  //either source or clone system
  public VegetationPackagePro package;
```

```
public Rect terrainRect;


public Texture2D[] textures;
public int[] maskGroupNums;
[System.NonSerialized] public bool masksApplied;


#if UNITY_EDITOR
[UnityEditor.InitializeOnLoadMethod]
#endif
[RuntimeInitializeOnLoadMethod]
static void Subscribe ()  => TerrainTile.OnTileMoved += ClearOnMove;
static void ClearOnMove (TerrainTile tile)
{
 VSProMapsTile vsTile = tile.GetComponent<VSProMapsTile>();
 vsTile?.OnDisable();
}



//public void OnEnable ()  //VSPro has got to run OnEnable first (objects only)
public void Start ()
{
 if (package == null)
  return;


 if (!masksApplied) VSProOps.SetTextures(system, package, textures, maskGroupNums, terrainRect);
 masksApplied = true;
```

```
  }



  public void OnDisable ()
  {
   if (masksApplied) VSProOps.ClearTextures(package, terrainRect);
   masksApplied = false;
  }

  #endif
 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using MapMagic.Terrains;

#if VEGETATION_STUDIO_PRO
using AwesomeTechnologies.VegetationStudio;
using AwesomeTechnologies.VegetationSystem;
using AwesomeTechnologies.Vegetation.Masks;
using AwesomeTechnologies.Vegetation.PersistentStorage;
using AwesomeTechnologies.Billboards;
#endif

namespace MapMagic.VegetationStudio
{
 [ExecuteInEditMode]
 public static class VSProOps
 /// Helper script to automatically add/remove and serialize textures and persistent storage data
 {
  #if VEGETATION_STUDIO_PRO

  public static VegetationSystemPro CopyVegetationSystem (VegetationSystemPro src, Transform parent)
  /// Clones src VegetationSystemPro object as a child of parent
  {
```

```
 GameObject copiedSourceVSProGo = GameObject.Instantiate(src.gameObject, new Vector3(0, 0, 0), Q
 copiedSourceVSProGo.SetActive(true);
 copiedSourceVSProGo.name = "VegetationSystemPro Copy";
 copiedSourceVSProGo.transform.parent = parent;
 copiedSourceVSProGo.transform.position = Vector3.zero;

 foreach (Transform child in copiedSourceVSProGo.transform)
  GameObject.DestroyImmediate(child.gameObject);

 VegetationSystemPro copiedSource = copiedSourceVSProGo.GetComponent<VegetationSystemPro>()
 copiedSource.PersistentVegetationStorage.PersistentVegetationStoragePackage = null;
 return copiedSource;
}


public static UnityTerrain AddUnityTerrain (Terrain terrain)
/// Adds VSPro UnityTerrain required component to terrain
{
 UnityTerrain unityTerrain = terrain.gameObject.GetComponent<UnityTerrain>();
 if (unityTerrain == null) unityTerrain = terrain.gameObject.AddComponent<UnityTerrain>();
 unityTerrain.Terrain = terrain;
 unityTerrain.TerrainPosition = terrain.transform.position;
 return unityTerrain;
}
```

```
public static VegetationSystemPro GetTopVegetationSystem ()
/// Finds main system to be used as source for clones
{
 GameObject[] rootObjects = UnityEngine.SceneManagement.SceneManager.GetActiveScene().GetRoot


 foreach (GameObject rootObject in rootObjects)
  if (rootObject.GetComponent<VegetationStudioManager>())
  {
   int childCount = rootObject.transform.childCount;
   for (int c=0; c<childCount; c++)
   {
    Transform child = rootObject.transform.GetChild(c);
    VegetationSystemPro sys = child.GetComponent<VegetationSystemPro>();


    if (sys != null)
     return sys;
   }
  }


 return null;
}



public static VegetationSystemPro GetCopyVegetationSystem (Terrain terrain)
{
 Transform tileTfm = terrain.transform.parent;
```

```
 //VegetationSystemPro copySystem = tileTfm.GetComponentInChildren<VegetationSystemPro>();
 //seems not to be searching top level first, others then, but deep search from first object. Will iterate all ob

 foreach (Transform child in tileTfm)
  if (child.name == "VegetationSystemPro Copy")
  {
   VegetationSystemPro copySystem = child.GetComponent<VegetationSystemPro>();
   return copySystem;
  }

 return null;
}


public static VegetationSystemPro UpdateCopySystem (VegetationSystemPro copySystem, Terrain terra
{
 Transform tileTfm = terrain.transform.parent;

 //erasing legacy settings on copyVS change
 //checking if src system has any extents, and resetting them - otherwise refreshing clone systems will tak
 if (srcSystem.VegetationSystemBounds.extents.x > 1)
 {
  srcSystem.VegetationSystemBounds = new Bounds (Vector3.zero, Vector3.zero);
  srcSystem.RefreshVegetationSystem();
 }
```

```
if (srcSystem.VegetationStudioTerrainObjectList.Count != 0)
 srcSystem.RemoveAllTerrains();

//unityterrain
UnityTerrain unityTerrain = terrain.GetComponent<UnityTerrain>();  //VS Pro component added to terrain
if (unityTerrain == null) unityTerrain = VSProOps.AddUnityTerrain(terrain);
unityTerrain.TerrainPosition = terrain.transform.position;

//resetting clone package just in case
if (copySystem.VegetationPackageProList.Count == 0  ||  copySystem.VegetationPackageProList[0] != pa
{
 copySystem.VegetationPackageProList.Clear();
 copySystem.AddVegetationPackage(package);
}

//reset bounds
copySystem.AutomaticBoundsCalculation = false;
copySystem.VegetationSystemBounds = new Bounds (  //or use unityTerrain.TerrainBounds;
 terrain.terrainData.bounds.center + terrain.transform.position,
 terrain.terrainData.bounds.extents * 2);

//add terrain
if (copySystem.VegetationStudioTerrainObjectList.Count != 1  ||  copySystem.VegetationStudioTerrainOb
{
 if (copySystem.VegetationStudioTerrainObjectList.Count != 0) copySystem.RemoveAllTerrains();
 copySystem.AddTerrain(terrain.gameObject);  //will call RefreshVegetationStudioTerrains and RefreshV
```

```
    }

    //clearing storage
    if (copySystem.PersistentVegetationStorage.PersistentVegetationStoragePackage == null)
     copySystem.PersistentVegetationStorage.PersistentVegetationStoragePackage = ScriptableObject.Crea
     //clearing is done in apply

    //refresh system (takes >100 ms)
    UnityEngine.Profiling.Profiler.BeginSample("VegetationSystemPro.RefreshVegetationSystem");
//    copySystem.RefreshVegetationSystem();
    if (copySystem != null) //re-initializing
    {
     copySystem.enabled = false;
     copySystem.enabled = true; //to call private OnEnable
    }

    UnityEngine.Profiling.Profiler.EndSample();

    return copySystem;
   }


   public static void UpdateSourceSystem (VegetationSystemPro srcSystem, Terrain terrain)
   {
    //erasing legacy settings on copyVS change
    //returning extents after using per-terrain systems
```

```
VegetationSystemPro copySystem = VSProOps.GetCopyVegetationSystem(terrain);


if (srcSystem.VegetationSystemBounds.extents.x < 1)
{
 srcSystem.VegetationSystemBounds = new Bounds (
  terrain.transform.position + terrain.terrainData.size/2,
  terrain.terrainData.size*3.4f);
 srcSystem.RefreshVegetationSystem();
}


if (copySystem != null  &&
 srcSystem.PersistentVegetationStorage != null  &&
 srcSystem.PersistentVegetationStorage.PersistentVegetationStoragePackage == copySystem.Persister
  srcSystem.PersistentVegetationStorage.InitializePersistentStorage();


if (copySystem != null)
 GameObject.DestroyImmediate(copySystem.gameObject);


//unityterrain
UnityTerrain unityTerrain = terrain.GetComponent<UnityTerrain>();  //VS Pro component added to terrain
if (unityTerrain == null) unityTerrain = VSProOps.AddUnityTerrain(terrain);
unityTerrain.TerrainPosition = terrain.transform.position;


if (srcSystem != null  &&  !srcSystem.VegetationStudioTerrainList.Contains(unityTerrain))  //check if alrea
srcSystem.AddTerrain(terrain.gameObject);
```

```
 //refresh system (takes >100 ms)
 UnityEngine.Profiling.Profiler.BeginSample("VegetationSystemPro.RefreshVegetationSystem");
 srcSystem.RefreshVegetationSystem();
 UnityEngine.Profiling.Profiler.EndSample();
}



public static void SetTextures (VegetationSystemPro system, VegetationPackagePro package, Texture2D
{
 for (int i=0; i<textures.Length; i++)
 {
  Texture2D tex = textures[i];
  if (tex == null) continue;

  TextureMaskGroup maskGroup = package.TextureMaskGroupList[maskGroupNums[i]];

  //creating new mask only if the mask with the same rect doesn't exist
  TextureMask mask = maskGroup.TextureMaskList.Find(m => m.TextureRect == terrainRect);
  if (mask == null)
  {
   mask = new TextureMask { TextureRect = terrainRect };
   maskGroup.TextureMaskList.Add(mask);
  }

  mask.MaskTexture = tex;
 }
```

```
 //VegetationSystemPro system = GameObject.FindObjectOfType<VegetationSystemPro>();
 //if (system != null)
 // system.ClearCache(); //clearing cache causes flickering
 system.RefreshTerrainHeightmap();
}



public static void ClearTextures (VegetationPackagePro package, Rect terrainRect)
{
 if (package == null) return;

 foreach (TextureMaskGroup maskGroup in package.TextureMaskGroupList)
 {
  for (int i=maskGroup.TextureMaskList.Count-1; i>=0; i--)
  {
   TextureMask mask = maskGroup.TextureMaskList[i];
   if (mask.MaskTexture == null  ||  mask.TextureRect == terrainRect)
   {
    mask.Dispose();
    maskGroup.TextureMaskList.RemoveAt(i);
    //if (system != null) system.SelectedTextureMaskGroupTextureIndex = 0;
   }
  }
 }
```

```
 //if (system != null)
 // system.ClearCache();
}



public static void SetObjects (VegetationSystemPro system, List<Transition>[] allTransitions, string[] allIds
{
 PersistentVegetationStorage storage = system.PersistentVegetationStorage;

 //if (system.VegetationCellQuadTree == null) //could happen on scene loading or playmode change
 // system.RefreshVegetationSystem();
 if (!system.InitDone)
  { system.enabled = true; }

 for (int i=0; i<allTransitions.Length; i++)
 {
  if (allTransitions[i] == null) continue;
  foreach (Transition obj in allTransitions[i])
  {
   storage.AddVegetationItemInstance(
     allIds[i],
     obj.pos,
     obj.scale,
     obj.rotation,
     applyMeshRotation: true,
     vegetationSourceID: sourceId,
```

```
    distanceFalloff: 1,
    clearCellCache:true);
 }
}

//for (int i=0; i<system.VegetationCellList.Count; i++)
// system.VegetationCellList[i].ClearCache();

system.RefreshBillboards();

#if UNITY_EDITOR
if (storage.PersistentVegetationStoragePackage!=null)
 UnityEditor.EditorUtility.SetDirty(storage.PersistentVegetationStoragePackage);
UnityEditor.EditorUtility.SetDirty(system);
#endif
}


public static void FlushObjects (VegetationSystemPro system, Rect terrainRect, bool clearCache=true)
{
PersistentVegetationStorage storage = system.PersistentVegetationStorage;

PersistentVegetationStoragePackage storagePackage = system.PersistentVegetationStorage.Persistent
if (storagePackage == null) return;

List<VegetationCell> overlapCellList = new List<VegetationCell>();
```

```
system.VegetationCellQuadTree.Query(terrainRect, overlapCellList);


for (int i=0; i < overlapCellList.Count; i++)
{
 int cellIndex = overlapCellList[i].Index;

 //storagePackage.PersistentVegetationCellList[cellIndex].ClearCell();

 var infoList = storagePackage.PersistentVegetationCellList[cellIndex].PersistentVegetationInfoList;
 for (int j=0; j<infoList.Count; j++)
 {
  var itemList = infoList[j].VegetationItemList;

  for (int k=itemList.Count-1; k>=0; k--)
  {
   Vector3 pos = itemList[k].Position + system.VegetationSystemPosition;
   Vector2 pos2 = pos.V2();

   if (terrainRect.Contains(pos2))
   {
    itemList.RemoveAt(k);
    //storage.RemoveVegetationItemInstance(infoList[j].VegetationItemID, pos, 1, clearCellCache:false);
   }
  }
 }
```

```
    //VegetationItemIndexes indexes = VegetationSystemPro.GetVegetationItemIndexes(vegetationItemID);
    //system.ClearCache(overlapCellList[i],indexes.VegetationPackageIndex,indexes.VegetationItemIndex);



  }

  //if (clearCache)
  //{
  // for (int i=0; i<system.VegetationCellList.Count; i++)
  //  system.VegetationCellList[i].ClearCache();
  //}

  system.RefreshBillboards();
 }



 #endif
}
}
```

```
ï»¿using System;
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes;
using MapMagic.Nodes.GUI;

#if VEGETATION_STUDIO_PRO
using AwesomeTechnologies.Common;
using AwesomeTechnologies.VegetationSystem;
#endif

namespace MapMagic.VegetationStudio
{
 public static class VSProObjectsOutEditor
 {
  #if VEGETATION_STUDIO_PRO //otherwise will log a warning
  private static string[] objectNames;
  #endif
```

```
[UnityEditor.InitializeOnLoadMethod]
static void EnlistInMenu ()
{
 CreateRightClick.generatorTypes.Add(typeof(VSProObjectsOut));
}

[Draw.Editor(typeof(VSProObjectsOut))]
public static void DrawVSProObjectsOut (VSProObjectsOut gen)
{
 #if VEGETATION_STUDIO_PRO
 VegetationPackagePro package = null;

 if (GraphWindow.current.mapMagic != null)
 {
  VegetationSystemPro system = GraphWindow.current.mapMagic.Globals.vegetationSystem as Vegetat
  using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(ref system, "System");
  GraphWindow.current.mapMagic.Globals.vegetationSystem = system;

  package = GraphWindow.current.mapMagic.Globals.vegetationPackage as VegetationPackagePro;
  using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(ref package, "Package");
  GraphWindow.current.mapMagic.Globals.vegetationPackage = package;
 }
 else
  using (Cell.LinePx(18+18)) Draw.Label("Not assigned to current \nMapMagic object");
```

```
//filling object names array for popup
if (package != null)
{
 if (objectNames == null || objectNames.Length != package.VegetationInfoList.Count)
  objectNames = new string[package.VegetationInfoList.Count];
 for (int i=0; i<objectNames.Length; i++)
  objectNames[i] = package.VegetationInfoList[i].Name;
}
else objectNames = null;

if (GraphWindow.current.mapMagic == null)
 using (Cell.LineStd) Draw.Label("Graph is not in scene");
else if (package == null)
 using (Cell.LineStd) Draw.Label("No package assigned");

#else

using (Cell.LinePx(76))
 Draw.Helpbox("Vegetation Studio Pro doesn't seem to be installed, or Vegetation Studio Pro compatibili

#endif

using (Cell.LineStd)
 LayersEditor.DrawLayers(
  ref gen.layers,
  onDraw: n => DrawVSProObjectsLayer(gen, package, n),
```

```
  onCreate: n => new VSProObjectsOut.Layer() );



using (Cell.LinePx(0))
{
 Cell.EmptyRowPx(2);

 using (Cell.Row)
 {
  //height
  Cell.EmptyLinePx(2);
  using (Cell.LinePx(0))
   using (new Draw.FoldoutGroup(ref gen.guiHeight, "Height", padding:0))
    if (gen.guiHeight)
    {
     using (Cell.LineStd) Draw.ToggleLeft(ref gen.objHeight, "Object Height");
     using (Cell.LineStd) Draw.ToggleLeft(ref gen.relativeHeight, "Relative Height");
    }

  //rotation
  Cell.EmptyLinePx(2);
  using (Cell.LinePx(0))
   using (new Draw.FoldoutGroup(ref gen.guiRotation, "Rotation", padding:0))
    if (gen.guiRotation)
    {
     using (Cell.LineStd) Draw.ToggleLeft(ref gen.useRotation, "Use Rotation");
```

```
   using (Cell.LineStd) Draw.ToggleLeft(ref gen.takeTerrainNormal, "Terrain Normal");
   using (Cell.LineStd)
   {
    Cell.current.disabled = gen.takeTerrainNormal;
    Draw.ToggleLeft(ref gen.rotateYonly, "Rotate Y Only"); //
   }
   using (Cell.LineStd) Draw.ToggleLeft(ref gen.regardPrefabRotation, "Regard Prefab");
   using (Cell.LineStd)
   {
    Cell.EmptyRowPx(18);
    using (Cell.Row) Draw.Label("Rotation");
   }
  }

//scale
Cell.EmptyLinePx(2);
using (Cell.LinePx(0))
 using (new Draw.FoldoutGroup(ref gen.guiScale, "Scale", padding:0))
  if (gen.guiScale)
  {
   using (Cell.LineStd) Draw.ToggleLeft(ref gen.useScale, "Use Scale");
   using (Cell.LineStd) Draw.ToggleLeft(ref gen.scaleYonly, "Scale Y Only");
   using (Cell.LineStd) Draw.ToggleLeft(ref gen.regardPrefabScale, "Regard Prefab");
   using (Cell.LineStd)
   {
    Cell.EmptyRowPx(18);
```

```
    using (Cell.Row) Draw.Label("Scale");
   }
  }

Cell.EmptyLinePx(2);
using (Cell.LinePx(0))
 using (new Draw.FoldoutGroup(ref gen.guiAdvanced, "Advanced", isLeft:false, padding:0))
  if (gen.guiAdvanced)
  {
   using (Cell.LinePx(0))
   {
    //using (Cell.LineStd) Draw.Label("Biome Blend");
    using (Cell.LineStd)
    {
     //Cell.current.fieldWidth = 0.6f;
     Draw.Field(ref gen.biomeBlend, "Biome Blend");
    }
   }

   if (GraphWindow.current.mapMagic != null)
   using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(
    ref GraphWindow.current.mapMagic.Globals.vegetationSystemCopy,
    "Copy VS");

   using (Cell.LineStd) Draw.Field(ref gen.outputLevel, "Out Level");
  }
```

```
  }

  Cell.EmptyRowPx(2);
 }
}

public static void DrawVSProObjectsLayer (VSProObjectsOut gen, VegetationPackagePro package, int n
{
 if (n>=gen.layers.Length) return; //on layer remove

 VSProObjectsOut.Layer layer = gen.layers[n];

 #if !VEGETATION_STUDIO_PRO
 {
  using (Cell.LineStd) Draw.Label(" Item id: " + layer.id??"none");
  return;
 }

 #else

 if (package == null)
 {
  using (Cell.LineStd) Draw.Label(" Item id: " + layer.id??"none");
  return;
 }
```

```
int itemInfoIndex = package.VegetationInfoList.FindIndex(i => i.VegetationItemID == layer.id);
VegetationItemInfoPro itemInfo = itemInfoIndex>=0 ? package.VegetationInfoList[itemInfoIndex] : null;

Texture2D icon = null;
if (itemInfo != null)
{
 #if UNITY_EDITOR
 if (itemInfo.PrefabType == VegetationPrefabType.Mesh) icon = AssetPreviewCache.GetAssetPreview(it
 else icon = AssetPreviewCache.GetAssetPreview(itemInfo.VegetationTexture);
 #endif
}

Cell.EmptyLinePx(4);
using (Cell.LinePx(24))
{
 Cell.EmptyRowPx(4);

 using (Cell.RowPx(24))
  if (icon!=null)
   Draw.TextureIcon(icon);

 Cell.EmptyRowPx(2);

 using (Cell.Row)
 {
  Cell.EmptyLine();
```

```
      using (Cell.LineStd)
      {
       Draw.PopupSelector(ref itemInfoIndex, objectNames);
       if (Cell.current.valChanged)
        layer.id = package.VegetationInfoList[itemInfoIndex].VegetationItemID;
      }
      Cell.EmptyLine();
     }
     Cell.EmptyRowPx(4);
    }
    Cell.EmptyLinePx(4);

    #endif
   }
  }
 }
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Products;
using MapMagic.Terrains;
using MapMagic.Nodes;
using MapMagic.Nodes.ObjectsGenerators;



#if !VEGETATION_STUDIO_PRO
namespace MapMagic.VegetationStudio
{
 [System.Serializable]
 [GeneratorMenu(
  name = "VS Pro Objs",
  section =2,
  drawButtons = false,
  colorType = typeof(TransitionsList))]
 public class VSProObjectsOut : OutputGenerator, IInlet<TransitionsList>
 {
  public OutputLevel outputLevel = OutputLevel.Main;
```

```
public override OutputLevel OutputLevel { get{ return outputLevel; } }

public PositioningSettings posSettings = null; // new PositioningSettings(); //to load older output
public BiomeBlend biomeBlend = BiomeBlend.Random;

//[Val("Package", type = typeof(VegetationPackagePro))] public VegetationPackagePro package; //in glob

[System.Serializable]
public class Layer
{
 public string id; //= "d825a526-4ba2-4c8f-9f4d-3f855049718a";

 public string lastUsedName;
 public string lastUsedType;
}

public Layer[] layers = new Layer[] { new Layer() }; //do not use BaseObjectsOutput prefabs

public const byte VS_MM_id = 15; //15 for MapMagic, 18 for Voxeland

//moved to PositioningSettings, and thus outdated:
public bool objHeight = true;
public bool relativeHeight = true;
public bool guiHeight;
public bool useRotation = true;
public bool takeTerrainNormal = false;
```

```
  public bool rotateYonly = false;
  public bool regardPrefabRotation = false;
  public bool guiRotation;
  public bool useScale = true;
  public bool scaleYonly = false;
  public bool regardPrefabScale = false;
  public bool guiScale;

  public PositioningSettings CreatePosSettings () =>
   posSettings = new PositioningSettings() {
   objHeight=objHeight, relativeHeight=relativeHeight, guiHeight=guiHeight,
   useRotation=useRotation, takeTerrainNormal=takeTerrainNormal, rotateYonly=rotateYonly, regardPrefal
   useScale=useScale, scaleYonly=scaleYonly, regardPrefabScale=regardPrefabScale, guiScale=guiScale


  public override void Generate (TileData data, StopToken stop) { }

  public override void ClearApplied (TileData data, Terrain terrain) { }
 }
}
#endif
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Products;
using MapMagic.Terrains;
using MapMagic.Nodes;
using MapMagic.Nodes.ObjectsGenerators;

#if VEGETATION_STUDIO_PRO
using AwesomeTechnologies.VegetationSystem;
using AwesomeTechnologies.Vegetation.Masks;
using AwesomeTechnologies.Vegetation.PersistentStorage;
using AwesomeTechnologies.Billboards;
#endif

namespace MapMagic.VegetationStudio
{
 [System.Serializable]
 [GeneratorMenu(
  menu = "Objects/Outputs",
  name = "VS Pro Objs",
```

```
 section =2,
 drawButtons = false,
 colorType = typeof(TransitionsList),
 helpLink = "https://gitlab.com/denispahunov/mapmagic/wikis/output_generators/Grass")]
public class VSProObjectsOut : OutputGenerator, IInlet<TransitionsList>
{
 public OutputLevel outputLevel = OutputLevel.Main;
 public override OutputLevel OutputLevel { get{ return outputLevel; } }

 public PositioningSettings posSettings = null; // new PositioningSettings(); //to load older output
 public BiomeBlend biomeBlend = BiomeBlend.Random;

 //[Val("Package", type = typeof(VegetationPackagePro))] public VegetationPackagePro package; //in glob

 [System.Serializable]
 public class Layer
 {
  public string id; //= "d825a526-4ba2-4c8f-9f4d-3f855049718a";

  public string lastUsedName;
  public string lastUsedType;
 }

 public Layer[] layers = new Layer[] { new Layer() }; //do not use BaseObjectsOutput prefabs
```

```
public const byte VS_MM_id = 15; //15 for MapMagic, 18 for Voxeland


//moved to PositioningSettings, and thus outdated:
public bool objHeight = true;
public bool relativeHeight = true;
public bool guiHeight;
public bool useRotation = true;
public bool takeTerrainNormal = false;
public bool rotateYonly = false;
public bool regardPrefabRotation = false;
public bool guiRotation;
public bool useScale = true;
public bool scaleYonly = false;
public bool regardPrefabScale = false;
public bool guiScale;


public PositioningSettings CreatePosSettings () =>
 posSettings = new PositioningSettings() {
 objHeight=objHeight, relativeHeight=relativeHeight, guiHeight=guiHeight,
 useRotation=useRotation, takeTerrainNormal=takeTerrainNormal, rotateYonly=rotateYonly, regardPrefal
 useScale=useScale, scaleYonly=scaleYonly, regardPrefabScale=regardPrefabScale, guiScale=guiScale



public override void Generate (TileData data, StopToken stop)
{
 if (stop!=null && stop.stop) return;
```

```
 if (!enabled) { data.RemoveFinalize(finalizeAction); return; }

 TransitionsList trns = data.ReadInletProduct(this);
 if (trns == null) return;
 if (data.globals.vegetationSystem == null || data.globals.vegetationPackage == null) return;

 //adding to finalize
 if (stop!=null && stop.stop) return;
 if (enabled)
 {
  data.StoreOutput(this, typeof(VSProObjectsOut), this, trns);  //adding src since it's not changing
  data.MarkFinalize(Finalize, stop);
 }
 else
  data.RemoveFinalize(finalizeAction);
}


public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 #if VEGETATION_STUDIO_PRO

 if (stop!=null && stop.stop) return;
 Noise random = new Noise(data.random, 12345);
```

```
Dictionary<string, List<Transition>> idsObjs = new Dictionary<string, List<Transition>>();
//key is id ("d825a526-4ba2-4c8f-9f4d-3f855049718a"), while value is the list of objects positions

foreach ((VSProObjectsOut output, TransitionsList trns, MatrixWorld biomeMask)
 in data.Outputs<VSProObjectsOut,TransitionsList,MatrixWorld>(typeof(VSProObjectsOut), inSubs:true))
{
 if (stop!=null && stop.stop) return;

 if (trns == null) continue;
 if (biomeMask!=null  &&  biomeMask.IsEmpty()) continue;

 if (output.posSettings == null)
  output.posSettings = output.CreatePosSettings();

 foreach (Layer layer in output.layers)
  if (!idsObjs.ContainsKey(layer.id)) idsObjs.Add(layer.id, new List<Transition>());

 //objects
 for (int t=0; t<trns.count; t++)
 {
  Transition trn = trns.arr[t]; //using copy since it's changing in MoveRotateScale

  if (!data.area.active.Contains(trn.pos)) continue; //skipping out-of-active area
  if (PositioningSettings.SkipOnBiome(ref trn, output.biomeBlend, biomeMask, data.random)) continue;

  output.posSettings.MoveRotateScale(ref trn, data);
```

```
    float rnd = random.Random(trn.hash);
    int layerNum = (int)(rnd*output.layers.Length);
    string id = output.layers[layerNum].id;
    idsObjs[id].Add(trn);
   }
  }

  //pushing to apply
  if (stop!=null && stop.stop) return;

  List<Transition>[] allTrns = new List<Transition>[idsObjs.Count];
  string[] allIds = new string[idsObjs.Count];

  int i=0;
  foreach (var kvp in idsObjs)
  {
   allTrns[i] = kvp.Value;
   allIds[i] = kvp.Key;
   i++;
  }

  //adding coord to ids to clear tile on change
/*   string coordId = null;
  if (!data.globals.vegetationSystemCopy) //if copy - just clearing it's storage
  {
```

```
  Coord coord = data.area.Coord;
  coordId = $"_c{coord.x},{coord.z}";
  for (int j=0; j<allIds.Length; j++)
   allIds[j] = allIds[j] + coordId;
 }*/

 ApplyData applyData = new ApplyData() {
  trns=allTrns,
  ids=allIds,
  srcSystem=data.globals.vegetationSystem as VegetationSystemPro,
  package=data.globals.vegetationPackage as VegetationPackagePro,
  copyVS= data.globals.vegetationSystemCopy};
 Graph.OnOutputFinalized?.Invoke(typeof(ObjectsOutput), data, applyData, stop);
 data.MarkApply(applyData);

#endif
}


public override void ClearApplied (TileData data, Terrain terrain)
{

}

#if VEGETATION_STUDIO_PRO
public class ApplyData : IApplyData
```

```
{
 public VegetationSystemPro srcSystem;
 private VegetationSystemPro copySystem; //assigned from ApplyCopy only
 public VegetationPackagePro package; //to update copy system
 public bool copyVS;

 public List<Transition>[] trns;
 public string[] ids;

 public void Read (Terrain terrain)  { throw new System.NotImplementedException(); }

 public void Apply (Terrain terrain)
 {
  //checking persistent storage assigned (too much users missing this part)
  PersistentVegetationStorage storage = srcSystem.PersistentVegetationStorage;
  #if UNITY_EDITOR
  if (storage.PersistentVegetationStoragePackage == null)
   UnityEditor.EditorUtility.DisplayDialog("VSPro Storage not assigned",
    "VSPro stoarage asset is not assigned. Assign storage asset to VegetationSystemPro object -> Persist
    "and click Initialize Persistent Storage button.",
    "OK");
  #endif

  //moving all objects in case MM object is not placed in zero
  TerrainTile tile = terrain.transform.parent.GetComponent<TerrainTile>();
  Core.MapMagicObject mapMagic = tile.mapMagic;
```

```
Vector3 pos = mapMagic.transform.position;
if (pos != Vector3.zero)
{
 for (int i=0; i<trns.Length; i++)
  for (int j=0; j<trns[i].Count; j++)
  {
   Transition trn = trns[i][j];
   trn.pos += pos;
   trns[i][j] = trn;
  }
}

//getting approximate tile number (to clear storage on non-copy - hacky)
int tileNum = VS_MM_id;
if (!copyVS)
{
 foreach (TerrainTile atile in mapMagic.tiles.All())
 {
  if (atile == tile) break;
  tileNum++;
 }
 tileNum = tileNum % 255;
}

//updating system
VegetationSystemPro copySystem = null;  //we'll need it to set up tile
```

```
if (copyVS)
{
 copySystem = VSProOps.GetCopyVegetationSystem(terrain);
 if (copySystem == null) copySystem = VSProOps.CopyVegetationSystem(srcSystem, terrain.transform.
 VSProOps.UpdateCopySystem(copySystem, terrain, package, srcSystem);
 copySystem.PersistentVegetationStorage.InitializePersistentStorage();
}

else
{
 VSProOps.UpdateSourceSystem(srcSystem, terrain);

 foreach (string id in ids)
  storage.RemoveVegetationItemInstances(id, (byte)tileNum);
}

//applying
VSProOps.SetObjects(copyVS ? copySystem : srcSystem, trns, ids, (byte)tileNum);

//tile obj (serialization and disable purpose)
Transform tileTfm = terrain.transform.parent;
VSProObjectsTile vsTile = tileTfm.GetComponent<VSProObjectsTile>();
if (vsTile == null) vsTile = tileTfm.gameObject.AddComponent<VSProObjectsTile>();
vsTile.system = copyVS ? copySystem : srcSystem;
vsTile.package = package;
vsTile.terrainRect = terrain.GetWorldRect();
```

```
    vsTile.transitions = trns;
    vsTile.objectIds = ids;
    vsTile.objectApplied = true;
   }



   public static ApplyData Empty
   {get{
    return new ApplyData() {
     trns = new List<Transition>[0],
     ids = new string[0]  };
   }}

   public int Resolution {get{ return 0; }}
  }
  #endif
 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using MapMagic.Terrains;

#if VEGETATION_STUDIO_PRO
using AwesomeTechnologies.VegetationStudio;
using AwesomeTechnologies.VegetationSystem;
using AwesomeTechnologies.Vegetation.Masks;
using AwesomeTechnologies.Vegetation.PersistentStorage;
using AwesomeTechnologies.Billboards;
#endif

namespace MapMagic.VegetationStudio
{
 [ExecuteInEditMode]
 public class VSProObjectsTile : MonoBehaviour, ISerializationCallbackReceiver
 /// Helper script to automatically add/remove and serialize textures and persistent storage data
 /// This will clear objects on moving terrain and playmode disable as well
 {
  #if VEGETATION_STUDIO_PRO
  public VegetationSystemPro system;  //either source or clone system
  public VegetationPackagePro package;
  public Rect terrainRect;
```

```
public List<Transition>[] transitions;  //serialized via OnBeforeSerialize. Unity stores arrays, stores lists, bu
public string[] objectIds;
[System.NonSerialized] public bool objectApplied;



#if UNITY_EDITOR
[UnityEditor.InitializeOnLoadMethod]
#endif
[RuntimeInitializeOnLoadMethod]
static void Subscribe ()  => TerrainTile.OnTileMoved += ClearOnMove;
static void ClearOnMove (TerrainTile tile)
{
 VSProObjectsTile vsTile = tile.GetComponent<VSProObjectsTile>();
 vsTile?.OnDisable();
}



//public void OnEnable ()  //VSPro has got to run OnEnable first (objects only)
public void Start ()
{
 if (package == null)
  return;

 if (!objectApplied) VSProOps.SetObjects(system, transitions, objectIds, VSProObjectsOut.VS_MM_id);
 objectApplied = true;
```

```
}



public void OnDisable ()
{
 if (objectApplied) VSProOps.FlushObjects(system, terrainRect);
 objectApplied = false;
}

#endif



#region Serialization

 [System.Serializable]
 public struct SerList  { public List<Transition> list; }
 public SerList[] serTransitions;

 public void OnBeforeSerialize ()
 {
  #if VEGETATION_STUDIO_PRO
  serTransitions = new SerList[transitions.Length];
  for (int i=0; i<transitions.Length; i++)
   serTransitions[i].list = transitions[i];
  #endif
 }
```

```
    public void OnAfterDeserialize ()
    {
     #if VEGETATION_STUDIO_PRO
     transitions = new List<Transition>[serTransitions.Length];
     for (int i=0; i<transitions.Length; i++)
      transitions[i] = serTransitions[i].list;
     #endif
    }

   #endregion
  }
 }
```

```
ï»¿
using UnityEngine;
using System;
using System.Threading;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.Profiling;

using Den.Tools;

using MapMagic.Products;
using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Locks;

namespace MapMagic.Core
{
 public interface IMapMagic
 {
  Graph Graph {get;}
  void Refresh (bool clearAll);
  float GetProgress ();
  bool IsGenerating ();
  bool ContainsGraph (Graph graph);
  Globals Globals {get;}
 }
```

```
[SelectionBase]
[ExecuteInEditMode] //to call onEnable, then it will subscribe to editor update
[HelpURL("https://gitlab.com/denispahunov/mapmagic/wikis/home")]
[DisallowMultipleComponent]
public class MapMagicObject : MonoBehaviour, IMapMagic, ISerializationCallbackReceiver
{
 public static readonly SemVer version = new SemVer(2,1,12);

 //graph
 public Graph graph;
 public Graph Graph => graph;
 public bool guiOpenObjectSelector = false;  //forcing opening of object selector if mm was created with "L

 //terrains grid
 public bool guiTiles = true;
 public TerrainTileManager tiles = new TerrainTileManager() { allowMove=true };
 public Vector2D tileSize = new Vector2D(1000,1000);  //IDEA: move to tiles manager?

 [SerializeField] Coord previewCoord;
 [SerializeField] bool previewAssigned;

 public enum Resolution { _33=33, _65=65, _129=129, _257=257, _513=513, _1025=1025, _2049=2049 }
 public Resolution tileResolution = Resolution._513;
 public int tileMargins = 16;
```

```
public bool draftsInEditor = true;

public bool draftsInPlaymode = true;

public Resolution draftResolution = Resolution._65;

public int draftMargins = 2;


//locks

public bool guiLocks = false;

public Lock[] locks = new Lock[0];


public bool guiInfiniteTerrains = false;

public bool hideFarTerrains = true;

public int mainRange = 1;

public int DraftRange => tiles.generateRange;

public bool clearOnNodeRemove = true;


//terrain settings

public TerrainSettings terrainSettings = new TerrainSettings();  //pixel error, shadows, trees, grass, etc.


//outputs settings

public Globals globals = new Globals();

public Globals Globals => globals;


//mapmagic settings

public bool guiSettings = false;

public bool guiTileSettings = false;

public bool guiOutputsSettings = false;
```

```
public bool guiExposedVariables = false;

public bool guiTerrainSettings = false;

public bool guiDraftSettings = false;

public bool guiTreesGrassSettings = false;

public bool instantGenerate = true;

public bool saveIntermediate = true;

public int heightWeldMargins = 5;

public int splatsWeldMargins = 2;

public bool guiHideWireframe = false;

public bool guiThreads = false;


[NonSerialized] public bool guiDraggingField = false;  //DragDrop.group=="DragField", stored here since D


public bool applyColliders = true;

public bool setDirty; //registering change for undo. Inverting this value if Unity Undo does not see a chang


//world shift

public bool shift = false;

public int shiftThreshold = 4000;

public int shiftExcludeLayers = 0;


public static bool isPlaying = true;

//EditorApplication.isPlayingOrWillChangePlaymode does not work in thread

//switched via SetIsPlaying


public void OnEnable ()
```

```
{
 #if UNITY_EDITOR
 UnityEditor.EditorApplication.update -= EditorUpdate; //just in case OnDisabled was not called somehow
 UnityEditor.EditorApplication.update += EditorUpdate;

 isPlaying = UnityEditor.EditorApplication.isPlayingOrWillChangePlaymode;
 UnityEditor.EditorApplication.playModeStateChanged -= SetIsPlaying;
 UnityEditor.EditorApplication.playModeStateChanged += SetIsPlaying;
 #endif

 if (terrainSettings.material == null)
  terrainSettings.material = DefaultTerrainMaterial();

 //generating all tiles that were not generated previously
 StopGenerate();
 StartGenerateNonReady(); //executing in update, otherwise will not find obj pool
}


public void OnDisable ()
{
 #if UNITY_EDITOR
 UnityEditor.EditorApplication.update -= EditorUpdate;
 #endif
}
```

```
public void EditorUpdate ()
{
 #if UNITY_EDITOR
 if (!isPlaying) Update();
 #endif
}

public void Update ()
{
 tiles.Update((Vector3)tileSize, pinned:tiles.pinned, holder:this, distsOnly:!isPlaying); //distsOnly: only upd

 Den.Tools.Tasks.CoroutineManager.Update();
}

#if UNITY_EDITOR
public static void SetIsPlaying (UnityEditor.PlayModeStateChange m)
 { isPlaying = m==UnityEditor.PlayModeStateChange.EnteredPlayMode; }
#endif

public void ApplyTileSettings ()
{
 StopGenerate();

 foreach (TerrainTile tile in tiles.Tiles())
  tile.Resize();
```

```
 Refresh(clearAll:true);
}



public void ApplyTerrainSettings ()
{
 foreach (TerrainTile tile in tiles.All())
 {
  if (tile.main != null) terrainSettings.ApplyAll(tile.main.terrain);
  if (tile.draft != null) { terrainSettings.ApplyAll(tile.draft.terrain); tile.draft.terrain.groupingID = -1; }
 }
}

public Material DefaultTerrainMaterial ()
{
 #if UNITY_2019_2_OR_NEWER
 Shader shader = UnityEngine.Rendering.GraphicsSettings.renderPipelineAsset?.defaultTerrainMaterial?
 #else
 Shader shader = null;
 #endif

 if (shader == null) shader = Shader.Find("HDRP/TerrainLit");
 if (shader == null) shader = Shader.Find("Nature/Terrain/Standard");
 if (shader == null) shader = Shader.Find("Lightweight Render Pipeline/Terrain/Lit");

 return new Material(shader);
```

```
}

#region Preview

 public TerrainTile PreviewTile
 {get{
   TerrainTile previewTile = null;
   if (previewAssigned) previewTile = tiles[previewCoord];
   if (previewTile == null) previewTile = tiles.ClosestMain();
   return previewTile;
 }}

 public void AssignPreviewTile (TerrainTile tile)
 {
  previewCoord = tile.coord;
  previewAssigned = true;
 }

 public void ClearPreviewTile () => previewAssigned = false;

 public TileData PreviewData => PreviewTile?.main.data;
 public Terrain PreviewTerrain => PreviewTile?.main?.terrain;

 //these ones ignore the automatic closes tile:
 public TerrainTile AssignedPreviewTile => tiles[previewCoord];
 public TileData AssignedPreviewData => AssignedPreviewTile?.main.data;
```

```
 public Terrain AssignedPreviewTerrain => AssignedPreviewTile?.main?.terrain;


#endregion


#region IMapMagic


 public void Refresh (bool clearAll=false)
 /// Makes current mapMagic to generate changed
 /// If clearAll then clears all and generates from scratch
 {
  if (graph == null) return;


  ClearChanged(clearAll);


  if (instantGenerate)
   StartGenerate();
 }


 private void ClearChanged (bool clearAll=false)
 {
  foreach (TerrainTile tile in tiles.All())
  {
   if (clearAll)
   {
    tile.StopGenerate(); //this will reset tile tasks
```

```
   tile.main?.data?.Clear(inSubs:true);
   tile.draft?.data?.Clear(inSubs:true);
  }

  if (tile.main?.data!=null)
   graph.ClearChanged(tile.main.data, clearAll);

  if (tile.draft?.data!=null)
   graph.ClearChanged(tile.draft.data, clearAll);
 }
}

public bool ContainsGraph (Graph graph)
/// Does MM has graph assigned in any way (as a root graph or biome)
{
 if (this.graph == null) return false;
 if (this.graph.generators == null) return false; //avoiding breaking all if something went wrong on loading
 if (this.graph == graph  ||  this.graph.ContainsSubGraph(graph, recursively:true)) return true;
 return false;
}

[Obsolete] public void Purge (OutputGenerator outGen, bool main=true, bool draft=true)
{
 foreach (TerrainTile tile in tiles.Tiles())
 {
  if (tile.main?.data != null) outGen.ClearApplied(tile.main.data, tile.main.terrain);
```

```
  if (tile.draft?.data != null) outGen.ClearApplied(tile.draft.data, tile.draft.terrain);
 }
}

public void ResetTerrains ()
{
 if (clearOnNodeRemove)
  foreach (TerrainTile tile in tiles.Tiles())
   tile.ResetTerrain();
}



public void SwitchLods ()
{
 foreach (TerrainTile tile in tiles.All())
  tile.SwitchLod();
}

public void EnableEditorDrafts (bool enabled)
{
 foreach (TerrainTile tile in tiles.All())
 {
  if (enabled && tile.draft==null) tile.draft = new TerrainTile.DetailLevel(tile, isDraft:true);
  if (!enabled && tile.draft!=null) { tile.draft.Remove(); tile.draft = null; }
 }
}
```

```
public bool IsGenerating ()
/// Finds out if MapMagic is currently generating or applying terrains.
/// Much faster than GetProgress, could be called every frame
{
 //if (!Den.Tools.Tasks.CoroutineManager.IsWorking && !Den.Tools.Tasks.ThreadManager.IsWorking) re
 //else return true;
 //might be doing other routine operations (will be added later)

 foreach (TerrainTile tile in tiles.All())
  if (tile.IsGenerating)
   return true;

 return false;
}



public float GetProgress ()
/// Returns minimum and maximum of the generated tiles (excluding previews), in percent 0-1
{
 float generateComplexity = graph.GetGenerateComplexity();
 float applyComplexity = graph.GetApplyComplexity();

 float totalComplexity = 0;
 float totalComplete = 0;
```

```
 foreach (TerrainTile tile in tiles.All())
 {
  (float complete, float complexity) tileProgress = tile.GetProgress(graph, generateComplexity, applyCom


  totalComplete += tileProgress.complete;
  totalComplexity += tileProgress.complexity;
 }


 return totalComplete / totalComplexity;


}


    #endregion


    #region Start/Stop Generate


private void StartGenerate (bool main=true, bool draft=true)
/// Start generating all tiles (if the specified lod is enabled)
{
 if (graph == null)
  throw new Exception("MapMagic: Graph data is not assigned");


 if (draft || main)
  foreach (TerrainTile tile in tiles.All())
   tile.StartGenerate(graph, main, draft);  //enqueue all of chunks before starting generate
```

```
}



public void StartGenerate (TerrainTile tile, bool generateMain=true, bool generateLod=true)
/// Used by: Tile.OnChange
{
 if (instantGenerate)
  tile.StartGenerate(graph, generateMain:generateMain, generateLod:generateLod);
}



private void StartGenerateNonReady ()
/// Start generating all tiles if they are not already generated
{
 if (graph == null)
  throw new Exception("MapMagic: Graph data is not assigned");

 foreach (TerrainTile tile in tiles.All())
  if (!tile.Ready)
   tile.StartGenerate(graph);
}

private void StopGenerate ()
{
 if (graph != null)
  foreach (TerrainTile tile in tiles.All())
```

```
    tile.StopGenerate();
}

    #endregion


    #region Serialization

    public bool serializedMultithreading = true;
public int serializedMaxThreads = 3;
public bool serializedAutoMaxThreads = true;
public float serializedMaxApplyTime = 10;

public virtual void OnBeforeSerialize ()
{
 serializedMultithreading = Den.Tools.Tasks.ThreadManager.useMultithreading;
 serializedMaxThreads = Den.Tools.Tasks.ThreadManager.maxThreads;
 serializedAutoMaxThreads = Den.Tools.Tasks.ThreadManager.autoMaxThreads;
 serializedMaxApplyTime = Den.Tools.Tasks.CoroutineManager.timePerFrame;
}

public virtual void OnAfterDeserialize ()
{
 Den.Tools.Tasks.ThreadManager.useMultithreading = serializedMultithreading;
 Den.Tools.Tasks.ThreadManager.maxThreads = serializedMaxThreads;
 Den.Tools.Tasks.ThreadManager.autoMaxThreads = serializedAutoMaxThreads;
```

```
   Den.Tools.Tasks.CoroutineManager.timePerFrame = serializedMaxApplyTime;
  }
 #endregion


}



[Serializable]
public class Globals
{
 public Globals Clone() => this.MemberwiseClone() as Globals;

 public float height = 250;
 public Nodes.MatrixGenerators.HeightOutput200.Interpolation heightInterpolation = Nodes.MatrixGenerat
 public int heightSplit = 129;
 #if UNITY_2019_1_OR_NEWER
 public Nodes.MatrixGenerators.HeightOutput200.ApplyType heightMainApply = Nodes.MatrixGenerators.
 #else
 public Nodes.MatrixGenerators.HeightOutput200.ApplyType heightMainApply = Nodes.MatrixGenerators.
 #endif
 public Nodes.MatrixGenerators.HeightOutput200.ApplyType heightDraftApply = Nodes.MatrixGenerators.
 public int grassResDownscale = 1;
 public int grassResPerPatch = 16;
 #if UNITY_2022_1_OR_NEWER
 public DetailScatterMode grassScatterMode = DetailScatterMode.InstanceCountMode;
 #endif
```

```
  public int objectsNumPerFrame = 500;
  public int holesRes = 64;

  //public Material microSplatMaterial; //using terrain mat instead

  public string[] customControlTextureNames = new string[] { "_ControlTexture1" };
        public UnityEngine.Object ctsProfile;
  public UnityEngine.Object microSplatPropData;
  public bool microSplatTerrainDescriptor = false;
  public enum MicroSplatApplyType { Textures, Splats, Both }; //not flag for editor selection
  public MicroSplatApplyType microSplatApplyType = MicroSplatApplyType.Textures;
  public bool useCustomControlTextures = false;
  public bool microSplatNormals = false;
  public UnityEngine.Object megaSplatTexList;
  public UnityEngine.Object vegetationPackage;
  public bool assignComponent; //assign ms/ms/cts component to terrain

  public UnityEngine.Object vegetationSystem;
  public bool vegetationSystemCopy = true;
  public string sourceMultiPackageVsProTagName; //for compatibility with MultiPackage
 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace MapMagic.Core
{
 public class SettingsAsset : ScriptableObject
 {
  public static SettingsAsset instance;

  #if UNITY_EDITOR
  [UnityEditor.InitializeOnLoadMethod]
  #endif
  [RuntimeInitializeOnLoadMethod]
  static void Load ()
  {
   /*SettingsAsset loadedInstance = Resources.Load<SettingsAsset>("MapMagicSettings");

   if (loadedInstance==null)
   {
    loadedInstance =
   }*/
  }
 }
}
```

```
ï»¿using System;
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;

using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Products;
using MapMagic.Previews;

namespace MapMagic.Core.GUI
{
 //[EditoWindowTitle(title = "MapMagic Graph")]  //it's internal Unity stuff
 public class AboutWindow : EditorWindow
 {
  UI ui = new UI();

  public void OnGUI ()
  {
   ui.Draw(DrawGUI, inInspector:false);
```

```
}



public void DrawGUI ()
{
 using (Cell.Line)
  DrawAbout();
}

public static void DrawAbout ()
{
 using (Cell.RowPx(100))
  Draw.Icon(UI.current.textures.GetTexture("MapMagic/Icons/AssetBig"), scale:0.5f);

 using (Cell.Row)
 {
  string versionName = MapMagicObject.version.ToString();
  //versionName = versionName[0]+"."+versionName[1]+"."+versionName[2];
  using (Cell.LineStd) Draw.Label("MapMagic " + versionName);
  using (Cell.LineStd) Draw.Label("by Denis Pahunov");

  Cell.EmptyLinePx(10);

  using (Cell.LineStd) Draw.URL(" - Online Documentation", "https://gitlab.com/denispahunov/mapmagic/v
  using (Cell.LineStd) Draw.URL(" - Video Tutorials", url:"https://www.youtube.com/playlist?list=PL8fjbXLq
  using (Cell.LineStd) Draw.URL(" - Forum Thread", url:"https://forum.unity.com/threads/released-mapmag
```

```
    using (Cell.LineStd) Draw.URL(" - Issues / Ideas", url:"http://mm2.idea.informer.com");
   }
  }


  [MenuItem ("Window/MapMagic/About")]
  public static void ShowWindow ()
  {
   AboutWindow window = (AboutWindow)GetWindow(typeof (AboutWindow));

   Texture2D icon = TexturesCache.LoadTextureAtPath("MapMagic/Icons/Window");
   window.titleContent = new GUIContent("About MapMagic", icon);

   window.position = new Rect(100,100,300,200);
  }
 }
}
```

```
ï»¿using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;

using MapMagic.Core;
using Den.Tools.GUI;

namespace MapMagic.GUI.DocScreens
{
 //[EditoWindowTitle(title = "MapMagic Graph")]  //it's internal Unity stuff
 public class DocScreensWindow : EditorWindow
 {
  Texture2D docScreen;
  List<Texture2D> sources = new List<Texture2D>();

  string folder = "C:\\Unity\\Wiki\\MapMagicWiki";
  string file = "Noise_Intensity";
  int interval = 40;
  float ratio = 4;
  float zoom = 1;

  UI ui = new UI();

  public void OnGUI ()
  {
```

```
 ui.Draw(DrawGUI, inInspector:false);
}



public void DrawGUI ()
{
 using (Cell.LineStd) Draw.Field(ref folder, "Folder");
 using (Cell.LineStd) Draw.Field(ref file, "Name");
 using (Cell.LinePx(0))
 {
  using (Cell.LineStd) Draw.Field(ref interval, "Interval");
  using (Cell.LineStd) Draw.Field(ref ratio, "Ratio");
  using (Cell.LineStd) Draw.Field(ref zoom, "Zoom");

  if (Cell.current.valChanged)
   docScreen = Compile(sources);
 }

 using (Cell.LineStd)
  if (Draw.Button("Append")) Append(withGizmos:false);

 using (Cell.LineStd)
  if (Draw.Button("Append in With Gizmos")) Append(withGizmos:true);

 using (Cell.LineStd)
  if (Draw.Button("Remove Last")) { sources.RemoveAt(sources.Count-1); docScreen = Compile(sources)
```

```
 using (Cell.LineStd)
  if (Draw.Button("Save")) Save();

 using (Cell.LineStd)
  if (Draw.Button("Clear")  &&  EditorUtility.DisplayDialog("Clear DocScreens", $"Clear {file} without saving
   Clear();

 using (Cell.LineStd)
  if (Draw.Button("Save and Clear"))
  {
   Save();
   Clear();
  }

 using (Cell.LinePx(UI.current.editorWindow.position.width/ratio))
  Draw.Texture(docScreen);
}

public void Append (bool withGizmos=true)
{
 Texture2D screen = CaptureSceneView(withGizmos);
 sources.Add(screen);

 docScreen = Compile(sources);
}
```

```
private Texture2D Compile (List<Texture2D> sources)
{
 if (sources.Count==1) return sources[0];

 int height = (int)(sources[0].height / zoom);
 int width = (int)(height * ratio);
 int numIntervals = sources.Count-1;
 int pureWidth = width - numIntervals*interval;
 int screenWidth = pureWidth / sources.Count;

 //if (screenWidth > sources[0].width)
 // throw new System.Exception("Scene view ratio error: it's too high, make it wider");

 Texture2D tex = new Texture2D(width, height);
 Color[] empty = new Color[width*height];
 tex.SetPixels(empty);

 for (int s=0; s<sources.Count; s++)
 {
  //if (sources[s].height != height)
  // throw new System.Exception("Different source height");

  int middle = sources[s].width / 2;
  int blockWidth = Mathf.Min(screenWidth, sources[s].width);
```

```
  int start = middle - blockWidth /2;

  int blockHeight = Mathf.Min(height, sources[s].height);

  Color[] colors = sources[s].GetPixels(start, 0, blockWidth, blockHeight);
  tex.SetPixels(
   s*screenWidth + s*interval + screenWidth/2 - blockWidth/2,
   height/2 - blockHeight/2,
   blockWidth,
   blockHeight,
   colors);
 }

 tex.Apply();
 return tex;
}

public void Save ()
{
 docScreen = Compile(sources);
 byte[] bytes =docScreen.EncodeToPNG();
 string filename = $"{folder}\\{file}.png";
 if (System.IO.File.Exists(filename) && !EditorUtility.DisplayDialog("Overwrite", $"Overwrite {filename}?", "
  return;
 System.IO.File.WriteAllBytes(filename, bytes);
}
```

```
public void Clear()
{
 docScreen = null;
 sources.Clear();
 //name = "Generator_Value_V1_V2_V3";
}


public Texture2D CaptureSceneView (bool withGizmos = true)
{
 SceneView sv = SceneView.lastActiveSceneView;
 Camera cam = sv.camera;


 Texture2D gizmosTex = withGizmos ? CamTexToTexture(cam) : null; //previous (before render) cam tex

 cam.Render();
 Texture2D camTex = CamTexToTexture(cam);

 if (withGizmos)
 {
  Color[] gizmosColors = gizmosTex.GetPixels();
  Color[] camColors = camTex.GetPixels();

  for (int i=0; i<camColors.Length; i++)
```

```
    camColors[i] = camColors[i]*(1-gizmosColors[i].a) + gizmosColors[i]*gizmosColors[i].a;


   camTex.SetPixels(camColors);
   camTex.Apply();
  }


  return camTex;
 }


 public Texture2D CamTexToTexture (Camera cam)
 {
  RenderTexture rt = cam.targetTexture;


  Texture2D tex = new Texture2D(rt.width, rt.height, TextureFormat.RGBA32, false);


  RenderTexture prevRt = RenderTexture.active;
  RenderTexture.active = rt;
  tex.ReadPixels(new Rect(0, 0, rt.width, rt.height), 0, 0);
  tex.Apply();
  RenderTexture.active = prevRt;


  return tex;
 }



 #if MM_DOC
```

```
  [MenuItem ("Window/MapMagic/DocScreens")]
  #endif
  public static void ShowWindow ()
  {
   DocScreensWindow window = (DocScreensWindow)GetWindow(typeof (DocScreensWindow));

   Texture2D icon = TexturesCache.LoadTextureAtPath("MapMagic/Icons/Window");
   window.titleContent = new GUIContent("MapMagic DocScreens", icon);

   window.position = new Rect(100,100,300,200);
  }
 }
}
```

```
ï»¿
using UnityEngine;
using UnityEditor;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;

using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Locks;

using MapMagic.Nodes.GUI; //to open up editor window
using MapMagic.Terrains.GUI; //pin draw

namespace MapMagic.Core.GUI
{
 [CustomEditor(typeof(MapMagicObject))]
 public class MapMagicInspector : Editor
 {
  public static MapMagicInspector current; //assigned on draw and removed after

  public MapMagicObject mapMagic; //aka target
```

```
UI ui = new UI();


public int backgroundHeight = 0; //to draw type background

public int oldSelected = 0; //to repaint gui with new background if new type was selected



public PinDraw.SelectionMode selectionMode = PinDraw.SelectionMode.none;

public bool saveDraft = false;


public enum Pots { _64=64, _128=128, _256=256, _512=512, _1024=1024, _2048=2048, _4096=4096 };


bool guiAbout = false;


private RectOffset pinButtonsOverflow;


public static  Action<MapMagicObject, Vector3> OnClusterExported; //called in cluster module (not availa



[RuntimeInitializeOnLoadMethod, UnityEditor.InitializeOnLoadMethod]

static void Subscribe ()

{

 TerrainTile.OnLodSwitched += (TerrainTile t,bool m, bool d) => SceneView.RepaintAll();

}


public void OnEnable ()
```

```
{
 EditorHacks.SetIconForObject(target, TexturesCache.LoadTextureAtPath("MapMagic/Icons/Window"));
}



//when selected
public void OnSceneGUI ()
{
 //foreach (TerrainTile tile in mapMagic.tiles.All())
 // EditorUtility.SetSelectedRenderState(tile.ActiveTerrain.GetComponent<Terrain>(), EditorSelectedRend


 current = this;
 if (mapMagic == null) mapMagic = (MapMagicObject)target;
 if (!mapMagic.enabled) return;


 FrameDraw.DrawSceneGUI(mapMagic);


 if (mapMagic.guiTiles)
  PinDraw.DrawSceneGUI(mapMagic, ref selectionMode, saveDraft);


 if (mapMagic.guiLocks)
  LockDraw.DrawSceneGUI(mapMagic);


 current = null;
}
```

```
public override void  OnInspectorGUI ()
{
 current = this;
 mapMagic = (MapMagicObject)target;

 //waiting for object selector close
 if (Event.current.type == EventType.ExecuteCommand &&
  Event.current.commandName == "ObjectSelectorUpdated" &&
  ScriptableAssetExtensions.GetObjectSelectorId() == 12345)
   mapMagic.graph = (Graph)ScriptableAssetExtensions.GetObjectSelectorObject();

 if (ui.undo == null)
  ui.undo = new Den.Tools.GUI.Undo {
   undoObject = mapMagic,
   undoName = "MapMagic Inspector Value"
  };

 ui.Draw(DrawGUI, inInspector:true);

 current = null;
}

public void DrawGUI ()
{
 Cell.EmptyLinePx(4);
```

```
//graph
using (Cell.LinePx(20))
{
 using (Cell.RowPx(50)) Draw.Label("Graph");
 using (Cell.Row)
 {
  Graph oldGraph = mapMagic.graph;
  Draw.ObjectField(ref mapMagic.graph);

  if (Cell.current.valChanged && mapMagic.graph != null)
  {
   GraphWindow.current.UpdateRelatedMapMagic();
   GraphWindow.current.Repaint();
   mapMagic.Refresh();
  }
 }

 using (Cell.RowPx(22))
 {
  Cell.current.disabled = mapMagic.graph==null || mapMagic.graph.Equals(null);
  Texture2D openIcon = UI.current.textures.GetTexture("DPUI/Icons/FolderOpen");
  if (Draw.Button(icon:openIcon, iconScale:0.5f, visible:false))
   GraphWindow.Show(mapMagic.graph);
 }
}
```

```
//graph empty warning
if (mapMagic.graph==null || mapMagic.graph.Equals(null))
 using (Cell.LinePx(80))
 {
  using (Cell.LinePx(50)) Draw.Label("MapMagic graph is not assigned. \nEither create a new one or \nse

  using (Cell.LinePx(20))
  {
   using (Cell.Row) if (Draw.Button("Create Empty"))
   {
    Graph graph = Graph.Create();
    graph.OnBeforeSerialize();
    graph = ScriptableAssetExtensions.SaveAsset(graph, filename:"MapMagic Graph", caption:"Save Map
    mapMagic.graph = graph;
    mapMagic.tiles.Pin( new Coord(0,0), false, mapMagic );
   }

   using (Cell.Row) if (Draw.Button("Create Template"))
   {
    Graph graph = GraphTemplates.CreateTemplate();
    graph.OnBeforeSerialize();
    graph = ScriptableAssetExtensions.SaveAsset(graph, filename:"MapMagic Graph", caption:"Save Map
    mapMagic.graph = graph;
```

```
    mapMagic.tiles.Pin( new Coord(0,0), false, mapMagic );
   }

   using (Cell.Row) if (Draw.Button("Select"))
   {
    //showing object selector
    ScriptableAssetExtensions.ShowObjectSelector(typeof(Graph), 12345, false);
   }
  }

  return;
 }

//graph not loaded
else if (mapMagic.graph.generators == null)
{
 using (Cell.LineStd) Draw.Label("Could not load graph");
 using (Cell.LineStd) Draw.Label("See console for details");

 return;
}

//seed
/*using (Cell.LinePx(20)))
{
 Draw.Label("Seed", cell:UI.Empty(Size.RowPixels(50)));
```

```
 Draw.Field(ref mapMagic.seed, cell:UI.Empty(Size.row));
 UI.Empty(Size.RowPixels(68));
}*/

//generate
Cell.EmptyLinePx(4);

using (Cell.LinePx(30))
{
 //if (pinButtonsOverflow == null) pinButtonsOverflow = new RectOffset(0,0,1,2);
 GUIStyle style = UI.current.textures.GetElementStyle("MapMagic/PinButtons/Top", "MapMagic/PinButto

 using (Cell.Padded(0,0,-2,-1))
  if (Draw.Button("", style:style))
 {
  //graphUI.undo.Record(completeUndo:true); //won't record changed terrain data
  foreach (Terrain terrain in mapMagic.tiles.AllActiveTerrains())
   UnityEditor.Undo.RegisterFullObjectHierarchyUndo(terrain.terrainData, "RefreshAll");
  EditorUtility.SetDirty(mapMagic);

  mapMagic.Refresh(clearAll:true);
 }

 Draw.Label("Generate", style:UI.current.styles.middleCenterLabel);
 using (Cell.RowPx(30)) Draw.Icon(UI.current.textures.GetTexture("DPUI/Icons/RefreshAll"), scale:0.5f);
}
```

```
using (Cell.LinePx(22))
{
 GUIStyle style = UI.current.textures.GetElementStyle("MapMagic/PinButtons/Bottom", "MapMagic/PinBu

 using (Cell.Padded(0,0,-2,-1))
  if (Draw.Button("", style:style))
 {
  mapMagic.Refresh(clearAll:false);
 }

 Draw.Label("Generate Changed", style:UI.current.styles.middleCenterLabel);
 using (Cell.RowPx(30)) Draw.Icon(UI.current.textures.GetTexture("DPUI/Icons/Refresh"), scale:0.5f);
}

//Tiles
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref mapMagic.guiTiles, "Tiles", isLeft:true))
  if (mapMagic.guiTiles)
   PinDraw.DrawInspectorGUI(mapMagic, ref selectionMode, ref saveDraft);

//Locks
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref mapMagic.guiLocks, "Locks", isLeft:true))
```

```
  if (mapMagic.guiLocks)
   LockDraw.DrawInspectorGUI(mapMagic);

//Infinite Terrain
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref mapMagic.guiInfiniteTerrains, "Infinite Terrain (Playmode)", isLeft:tru
  if (mapMagic.guiInfiniteTerrains)
  {
   using (Cell.LineStd) Draw.ToggleLeft(ref mapMagic.tiles.generateInfinite, "Generate Infinite Terrain");

   using (Cell.LineStd) Draw.Field(ref mapMagic.mainRange, "Main Range");
   using (Cell.LineStd)
   {
    if (!mapMagic.draftsInPlaymode)
    {
     Cell.current.disabled = true;
     mapMagic.tiles.generateRange = mapMagic.mainRange;
    }

    Draw.Field(ref mapMagic.tiles.generateRange, "Drafts Range");
   }

   Cell.EmptyLinePx(4);
   using (Cell.LineStd) Draw.ToggleLeft(ref mapMagic.hideFarTerrains, "Hide Out-of-Range Terrains");
```

```
   Cell.EmptyLinePx(4);
   using (Cell.LineStd) Draw.Label("Generate Terrain Markers:");
   using (Cell.LineStd) Draw.Toggle(ref mapMagic.tiles.genAroundMainCam, "Around Main Camera");
   using (Cell.LineStd)
   {
    using (Cell.RowRel(1-Cell.current.fieldWidth))
     Draw.Label("Around Objects Tagged");

    using (Cell.RowRel(Cell.current.fieldWidth))
    {
     using (Cell.RowPx(20))
      Draw.Toggle(ref mapMagic.tiles.genAroundObjsTag);

     using (Cell.Row)
      mapMagic.tiles.genAroundTag = Draw.Field(
       mapMagic.tiles.genAroundTag,
       drawFn:(Rect rect, string oldVal) => { return EditorGUI.TagField(rect, (string)oldVal); } );
    }
   }
  }

//Size and Resolution
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref mapMagic.guiTileSettings, "Tile Settings", isLeft:true))
  if (mapMagic.guiTileSettings)
```

```
{
 using (Cell.LineStd)
 {
  float newSize = Draw.Field(mapMagic.tileSize.x, "Size");
  if (Cell.current.valChanged)
  {
   //UnityEditor.Undo.RegisterFullObjectHierarchyUndo(mapMagic.gameObject, "MapMagic Tile Size");
   mapMagic.tileSize.z = newSize;
   mapMagic.tileSize.x = newSize;
  }
 }

 Cell.EmptyLinePx(6);
 using (Cell.LineStd)
 {
  MapMagicObject.Resolution prevResolution = mapMagic.tileResolution;
  Draw.Field(ref mapMagic.tileResolution, "Main Resolution");
  if (Cell.current.valChanged && mapMagic.locks.Length!=0)
  {
   if (!EditorUtility.DisplayDialog("Resolution Change", "This will remove all of the Lock custom terrain da
    mapMagic.tileResolution = prevResolution;
  }
 }
 using (Cell.LineStd) Draw.Field(ref mapMagic.tileMargins, "Main Margins");

 Cell.EmptyLinePx(6);
```

```
if (mapMagic.draftsInEditor || mapMagic.draftsInPlaymode)
{
 using (Cell.LineStd) Draw.Field(ref mapMagic.draftResolution, "Draft Resolution");
 using (Cell.LineStd) Draw.Field(ref mapMagic.draftMargins, "Draft Margins");
}

Cell.EmptyLinePx(6);
using (Cell.LineStd) Draw.Label("Use Draft (low-detail) Terrains in:");
using (Cell.LineStd)
{
 Draw.ToggleLeft(ref mapMagic.draftsInEditor, "Editor");
 if (Cell.current.valChanged) mapMagic.EnableEditorDrafts(mapMagic.draftsInEditor);
}
using (Cell.LineStd)
{
 Draw.ToggleLeft(ref mapMagic.draftsInPlaymode, "Playmode");
 if (!mapMagic.draftsInPlaymode) mapMagic.tiles.generateRange = mapMagic.mainRange; //if just cha
}

if (Cell.current.valChanged)
{
 if (mapMagic.tileMargins < 2) mapMagic.tileMargins = 2;  //min margins is 2 for now
 if (mapMagic.draftMargins < 2) mapMagic.draftMargins = 2;

 mapMagic.ApplyTileSettings();
}
```

```
   Cell.EmptyLinePx(6);
   using (Cell.LineStd) Draw.ToggleLeft(mapMagic.clearOnNodeRemove, "Clear Generated on Node Re
  }

//Outputs settings
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref mapMagic.guiOutputsSettings, "Outputs Settings", isLeft:true))
  if (mapMagic.guiOutputsSettings)
  {
   using (Cell.LinePx(0))
   {
    Draw.Element(UI.current.styles.foldoutBackground);

    using (Cell.Padded(2,2,0,0))
    {
     Cell.EmptyLinePx(2);
     using (Cell.LineStd) Draw.Label("Height Output", style:UI.current.styles.boldLabel);

     Cell.EmptyLinePx(2);
     using (Cell.LineStd) Draw.Field(ref mapMagic.globals.height, "Height");
     using (Cell.LineStd) Draw.Field(ref mapMagic.globals.heightInterpolation, "Interpolate");

     Cell.EmptyLinePx(2);
     using (Cell.LineStd) Draw.Label("Apply Type");
```

```
  using (Cell.LineStd) Draw.Field(ref mapMagic.globals.heightMainApply, "Main");
  using (Cell.LineStd) Draw.Field(ref mapMagic.globals.heightDraftApply, "Draft");
  using (Cell.LineStd) Draw.Field(ref mapMagic.globals.heightSplit, "Split Frame");

  Cell.EmptyLinePx(2);
 }
}

#if CTS_PRESENT
Cell.EmptyLinePx(4);
using (Cell.LinePx(0))
{
 Draw.Element(UI.current.styles.foldoutBackground);

 using (Cell.Padded(2,2,0,0))
 {
  Cell.EmptyLinePx(2);
  using (Cell.LineStd) Draw.Label("CTS Output", style:UI.current.styles.boldLabel);
  using (Cell.LineStd) Draw.ObjectField(ref mapMagic.globals.ctsProfile, "CTS Profile");
  Cell.EmptyLinePx(2);
 }
}
#endif

#if __MICROSPLAT__
Cell.EmptyLinePx(4);
```

```
using (Cell.LinePx(0))
{
 Draw.Element(UI.current.styles.foldoutBackground);

 using (Cell.Padded(2,2,0,0))
 {
  Cell.EmptyLinePx(2);
  using (Cell.LineStd) Draw.Label("MicroSplat Output", style:UI.current.styles.boldLabel);

  Cell.EmptyLinePx(2);
  using (Cell.LineStd) Draw.ObjectField(ref mapMagic.globals.microSplatPropData, "Prop Data");
  using (Cell.LineStd) Draw.Toggle(ref mapMagic.globals.microSplatTerrainDescriptor, "Terrain Descrip
  using (Cell.LineStd) Draw.Field(ref mapMagic.globals.microSplatApplyType, "Apply Type");
  using (Cell.LineStd) Draw.Toggle(ref mapMagic.globals.useCustomControlTextures, "Use Custom Te
  using (Cell.LineStd) Draw.Toggle(ref mapMagic.globals.assignComponent, "Set Component");
  using (Cell.LineStd) Draw.ObjectField(ref mapMagic.globals.megaSplatTexList, "TexList");

  Cell.EmptyLinePx(2);
 }
}
#endif

#if __MEGASPLAT__
Cell.EmptyLinePx(4);
using (Cell.LinePx(0))
{
```

```
 Draw.Element(UI.current.styles.foldoutBackground);

 using (Cell.Padded(2,2,0,0))
 {
  Cell.EmptyLinePx(2);
  using (Cell.LineStd) Draw.Label("MegaSplat Output", style:UI.current.styles.boldLabel);
  using (Cell.LineStd) Draw.ObjectField(ref mapMagic.globals.megaSplatTexList, "TexList");
  using (Cell.LineStd) Draw.Toggle(ref mapMagic.globals.assignComponent, "Set Component");
  Cell.EmptyLinePx(2);
 }
}
#endif

#if VEGETATION_STUDIO_PRO
Cell.EmptyLinePx(4);
using (Cell.LinePx(0))
{
 Draw.Element(UI.current.styles.foldoutBackground);

 using (Cell.Padded(2,2,0,0))
 {
  Cell.EmptyLinePx(2);
  using (Cell.LineStd) Draw.Label("VSPro Output", style:UI.current.styles.boldLabel);

  Cell.EmptyLinePx(2);
  using (Cell.LineStd) Draw.Field(ref mapMagic.globals.height, "Height");
```

```
  using (Cell.LineStd) Draw.Field(ref mapMagic.globals.heightInterpolation, "Interpolate");

  Cell.EmptyLinePx(2);
  using (Cell.LineStd) Draw.ObjectField(ref mapMagic.globals.vegetationPackage, "Vegetation Package
  using (Cell.LineStd) Draw.ObjectField(ref mapMagic.globals.vegetationSystem, "Vegetation system");
  using (Cell.LineStd) Draw.Toggle(ref mapMagic.globals.vegetationSystemCopy, "Copy System");
  using (Cell.LineStd) Draw.Field(ref mapMagic.globals.sourceMultiPackageVsProTagName, "Source M

  Cell.EmptyLinePx(2);
 }
}
#endif

Cell.EmptyLinePx(4);
using (Cell.LinePx(0))
{
 Draw.Element(UI.current.styles.foldoutBackground);

 using (Cell.Padded(2,2,0,0))
 {
  Cell.EmptyLinePx(2);
  using (Cell.LineStd) Draw.Label("Grass Output", style:UI.current.styles.boldLabel);

  Cell.EmptyLinePx(2);
  using (Cell.LineStd) Draw.Field(ref mapMagic.globals.grassResDownscale, "Resolution Downscale");
  using (Cell.LineStd) Draw.Field(ref mapMagic.globals.grassResPerPatch, "Resolution per Patch");
```

```
  #if UNITY_2022_2_OR_NEWER
  using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(ref mapMagic.globals.grassScatterMode, "Scatte
  #endif

  Cell.EmptyLinePx(2);
 }
}

Cell.EmptyLinePx(4);
using (Cell.LinePx(0))
{
 Draw.Element(UI.current.styles.foldoutBackground);

 using (Cell.Padded(2,2,0,0))
 {
  Cell.EmptyLinePx(2);
  using (Cell.LineStd) Draw.Label("Objects Output", style:UI.current.styles.boldLabel);
  using (Cell.LineStd) Draw.Field(ref mapMagic.globals.objectsNumPerFrame, "Num Per Frame");
  Cell.EmptyLinePx(2);
 }
}

if (Cell.current.valChanged)
{
 mapMagic.Refresh(clearAll:true);
```

```
  }
 }



//Exposed variables
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref mapMagic.guiExposedVariables, "Exposed Variables", isLeft:true))
  if (mapMagic.guiExposedVariables)
   Expose.GUI.OverrideInspector.DrawStaticOverride(mapMagic.graph.defaults);



//Terrain Settings
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref mapMagic.guiTerrainSettings, "Terrain Properties", isLeft:true))
  if (mapMagic.guiTerrainSettings)
  {
   Cell.EmptyLinePx(6);
   using (Cell.LinePx(0))
   {
    using (Cell.LineStd)
    {
     using (Cell.RowRel(1-Cell.current.fieldWidth-0.08f)) Draw.Label("Auto Connect");
     using (Cell.RowRel(0.08f)) Draw.Toggle(ref mapMagic.terrainSettings.allowAutoConnect);
     using (Cell.RowRel(Cell.current.fieldWidth)) Draw.Field(ref mapMagic.terrainSettings.groupingID);
```

```
 }
 //using (Cell.LineStd) Draw.Field(ref mapMagic.terrainSettings.pixelError, "Pixel Error");
 using (Cell.LineStd)
 {
  using (Cell.RowRel(1-Cell.current.fieldWidth-0.08f)) Draw.Label("Base Map Distance");
  using (Cell.RowRel(0.08f)) Draw.Toggle(ref mapMagic.terrainSettings.showBaseMap);
  using (Cell.RowRel(Cell.current.fieldWidth)) Draw.Field(ref mapMagic.terrainSettings.baseMapDist);
 }
 using (Cell.LineStd) Draw.Field(ref mapMagic.terrainSettings.baseMapResolution, "Base Map Resolut

 Draw.Class(mapMagic.terrainSettings, category:"Terrain");

 if (Cell.current.valChanged) mapMagic.ApplyTerrainSettings();
}

Cell.EmptyLinePx(6);
using (Cell.LineStd) Draw.Field(ref mapMagic.terrainSettings.material, "Material Template");

Cell.EmptyLinePx(6);
using (Cell.LineStd)
{
 Draw.ToggleLeft(ref mapMagic.guiHideWireframe, "Hide Selection Outline"); //
 if (Cell.current.valChanged) mapMagic.transform.ToggleDisplayWireframe(mapMagic.guiHideWirefram
}

//copy
```

```
    Cell.EmptyLinePx(6);
    using (Cell.LineStd) Draw.ToggleLeft(ref mapMagic.terrainSettings.copyLayersTags, "Copy Layers to T
    using (Cell.LineStd) Draw.ToggleLeft(ref mapMagic.terrainSettings.copyLayersTags, "Copy Tags to Te
    using (Cell.LineStd) Draw.ToggleLeft(ref mapMagic.terrainSettings.copyComponents, "Copy Compone

    if (Cell.current.valChanged)
     mapMagic.ApplyTerrainSettings();
   }



//Trees, Details and Grass Settings
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref mapMagic.guiTreesGrassSettings, "Trees, Details and Grass Proper
  if (mapMagic.guiTreesGrassSettings)
  {
    Draw.Class(mapMagic.terrainSettings, category:"Trees");
    Cell.EmptyLinePx(5);
    Draw.Class(mapMagic.terrainSettings, category:"Grass");
    Cell.EmptyLinePx(5);
    Draw.Class(mapMagic.terrainSettings, category:"WindTint");

    if (Cell.current.valChanged) mapMagic.ApplyTerrainSettings();
   }

//Multithreading
```

```
Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref mapMagic.guiThreads, "Multithreading", isLeft:true))
  if (mapMagic.guiThreads)
  {
   using (Cell.LineStd) Draw.Toggle(ref Den.Tools.Tasks.ThreadManager.useMultithreading, "Use Multith
   using (Cell.LineStd) Draw.Toggle(ref Den.Tools.Tasks.ThreadManager.autoMaxThreads, "Auto Max Th
   using (Cell.LineStd)
   {
    Cell.current.disabled = Den.Tools.Tasks.ThreadManager.autoMaxThreads;
    Draw.Field(ref Den.Tools.Tasks.ThreadManager.maxThreads, "Max Threads");
   }

   Cell.EmptyLinePx(5);

   using (Cell.LineStd)
    Draw.Field(ref Den.Tools.Tasks.CoroutineManager.timePerFrame, "Apply Time per Frame");

   Cell.EmptyLinePx(5);

   using (Cell.LineStd) Draw.Toggle(ref mapMagic.instantGenerate, "Instant Generate");

   Cell.EmptyLinePx(5);

   using (Cell.LinePx(32)) Draw.Helpbox("Multitheading values are shared between all MapMagic objects
```

```
    }

  //About
  Cell.EmptyLinePx(4);
  using (Cell.LineStd)
   using (new Draw.FoldoutGroup(ref guiAbout, "About", isLeft:true))
    if (guiAbout)
    {
     using (Cell.Line)
      AboutWindow.DrawAbout();
    }
}
```

```
public bool GetDebug ()
{
 #if UNITY_EDITOR
 UnityEditor.BuildTargetGroup buildGroup = UnityEditor.EditorUserBuildSettings.selectedBuildTargetGrou
 string defineSymbols = UnityEditor.PlayerSettings.GetScriptingDefineSymbolsForGroup(buildGroup);

 return (defineSymbols.Contains("WDEBUG;") || defineSymbols.EndsWith("WDEBUG"));
 #else
 return false;
```

```
 #endif
}

public void SetDebug (object debug) { SetDebug((bool)debug); }
public void SetDebug (bool debug)
{
 #if UNITY_EDITOR
 UnityEditor.BuildTargetGroup buildGroup = UnityEditor.EditorUserBuildSettings.selectedBuildTargetGrou
 string defineSymbols = UnityEditor.PlayerSettings.GetScriptingDefineSymbolsForGroup(buildGroup);

 if (debug)
 {
  defineSymbols += (defineSymbols.Length!=0? ";" : "") + "WDEBUG";
 }
 else
 {
  defineSymbols = defineSymbols.Replace("WDEBUG","");
  defineSymbols = defineSymbols.Replace(";;", ";");
 }

 UnityEditor.PlayerSettings.SetScriptingDefineSymbolsForGroup(buildGroup, defineSymbols);
 #endif
}


[MenuItem ("GameObject/3D Object/MapMagic")]
```

```
  public static MapMagicObject CreateMapMagic () { return CreateMapMagic(null); }


  public static MapMagicObject CreateMapMagic (Graph graph)
  {
   GameObject go = new GameObject();
   go.SetActive(false); //to avoid starting generate while graph not assigned
   go.name = "MapMagic";
   MapMagicObject mapMagic = go.AddComponent<MapMagicObject>();
   Selection.activeObject = mapMagic;


   mapMagic.graph = graph;
   go.SetActive(true);
   mapMagic.tiles.Pin( new Coord(0,0), false, mapMagic );


   //registering undo
   UnityEditor.Undo.RegisterCreatedObjectUndo (go, "MapMagic Create");
   EditorUtility.SetDirty(mapMagic);


//   Selection.activeGameObject = mapMagic.gameObject;
   //MapMagicWindow.Show(mapMagic.gens, mapMagic, asBiome:false);


   return mapMagic;
  }


 }//class
```

```
}//namespace
```

ï»¿

```
//using a different assembly to make it compile independently from MapMagic or other assets
#if UNITY_EDITOR

using System;
using System.IO;
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;
//using System.Reflection;
using UnityEngine.Profiling;
using UnityEditor.Compilation;

//using Plugins;
//using Plugins.GUI;

//using MapMagic.Core;
//using MapMagic.Nodes;
//using MapMagic.Products;
//using MapMagic.Previews;

namespace MapMagic.GUI
{
```

```
//[EditoWindowTitle(title = "MapMagic Settings")]  //it's internal Unity stuff
public class SettingsWindow : EditorWindow
{
 private struct Settings : ICloneable
 {
  public bool native;
  public bool debug;
  public bool experimental;

  public bool cts;
  public bool megaSplat;
  public bool microSplat;
  public bool rtp;
  public bool vsPro;

  public bool autoRef;

  public object Clone() => this.MemberwiseClone();
  public void DisableCompatibility ()
   { cts=false; megaSplat=false; microSplat=false; rtp=false; vsPro=false; }
 }

 private Settings current;
 private Settings changed;

 //UI ui = new UI(); //using standard editor since settings supposed to be compiled before everything else (
```

```
#if UNITY_EDITOR
[UnityEditor.InitializeOnLoadMethod]
#endif
static void InitializeSettings ()
/// Initializes settings on first MM import
{
 BuildTargetGroup group = EditorUserBuildSettings.selectedBuildTargetGroup;
 string symbols = PlayerSettings.GetScriptingDefineSymbolsForGroup(group);

 //removing beta marks
 if (symbols.Contains("_MAPMAGIC_BETA"))
  ToggleKeyword(false, "_MAPMAGIC_BETA", ref symbols);
 if (symbols.Contains("_MMNATIVE"))
  ToggleKeyword(false, "_MMNATIVE", ref symbols);

 if (!symbols.Contains("MAPMAGIC2"))
 {
  ToggleKeyword(true, "MAPMAGIC2", ref symbols); //for voxeland and other plugins compatibility

  #if UNITY_EDITOR_WIN || UNITY_EDITOR_OSX
  ToggleKeyword(true, "MM_NATIVE", ref symbols);
  #endif

  PlayerSettings.SetScriptingDefineSymbolsForGroup(group, symbols);
 }
```

```
 //ShowNet20Notification();
}


public void OnEnable ()
{
 current = ReadCurrentSettings();
 changed = (Settings)current.Clone();
}

public void OnGUI ()
{
 //ui.Draw(DrawGUI);
 DrawGUI();
}

public void DrawGUI ()
{
 if (EditorApplication.isCompiling)
  EditorGUILayout.HelpBox("Compiling scripts. Please wait until compilation is finished", MessageType.N

 //using (Cell.LineStd)
 EditorGUI.BeginDisabledGroup(EditorApplication.isCompiling);
 using (new EditorGUILayout.HorizontalScope())
 {
```

```
EditorGUILayout.Space();
using (new EditorGUILayout.VerticalScope())
{
 BuildTargetGroup group = EditorUserBuildSettings.selectedBuildTargetGroup;
 string symbols = PlayerSettings.GetScriptingDefineSymbolsForGroup(group);

 EditorGUILayout.Space();

 EditorGUILayout.LabelField("Scripting Define Symbols");
 DrawToggle(current.native, ref changed.native, "C++ Native Code");
 DrawToggle(current.debug, ref changed.debug, "Debug Mode");
 DrawToggle(current.experimental, ref changed.experimental, "Experimental Features");

 //Cell.EmptyLinePx(5);
 EditorGUILayout.Space();

 EditorGUI.BeginDisabledGroup(!changed.autoRef);
 if (!changed.autoRef)
  changed.DisableCompatibility();
 EditorGUILayout.LabelField("Compatibility");
 DrawToggle(current.cts, ref changed.cts, "CTS 2019");
 DrawToggle(current.megaSplat, ref changed.megaSplat, "MegaSplat");
 DrawToggle(current.microSplat, ref changed.microSplat, "MicroSplat");
 DrawToggle(current.rtp, ref changed.rtp, "Relief Terrain Pack");
 DrawToggle(current.vsPro, ref changed.vsPro, "Vegetation Studio Pro");
 EditorGUI.EndDisabledGroup();
```

```
//Cell.EmptyLinePx(4);
EditorGUILayout.Space();

#if UNITY_2019_2_OR_NEWER
DrawToggle(current.autoRef, ref changed.autoRef, "Assemblies Auto Ref");
EditorGUILayout.HelpBox("Enable MM assemblies Auto Reference for compatibility with these or custo
#endif

EditorGUILayout.Space();
EditorGUILayout.LabelField("*: Fields marked with asterisk are not applied yet.");

EditorGUILayout.Space();
using (new EditorGUILayout.HorizontalScope())
{
 //EditorGUILayout.Space(180);
 //doesnt work in Unity 2019

 EditorGUILayout.Space();
 EditorGUILayout.Space();
 EditorGUILayout.Space();

 if (GUILayout.Button("Apply"))
 {
  ApplySettings(changed);
  current = ReadCurrentSettings();
```

```
    }

    if (GUILayout.Button("Cancel"))
     Close();
   }
  }
  EditorGUILayout.Space();


  //EditorWindow.focusedWindow.Repaint();
  //AssetDatabase.Refresh();
 }
 EditorGUI.EndDisabledGroup();
}

static void DrawToggle (bool currEnabled, ref bool newEnabled, string label)
{
 if (newEnabled != currEnabled)
  label += "*";

 newEnabled = EditorGUILayout.ToggleLeft(label, newEnabled);
}

static Settings ReadCurrentSettings ()
```

```
{
 Settings settings = new Settings();

 BuildTargetGroup group = EditorUserBuildSettings.selectedBuildTargetGroup;
 string symbols = PlayerSettings.GetScriptingDefineSymbolsForGroup(group);

 settings.native = symbols.Contains("MM_NATIVE");
 settings.debug = symbols.Contains("MM_DEBUG");
 settings.experimental = symbols.Contains("MM_EXPERIMENTAL");

 settings.cts = symbols.Contains("CTS_PRESENT");
 settings.megaSplat = symbols.Contains("__MEGASPLAT__");
 settings.microSplat = symbols.Contains("__MICROSPLAT__");
 settings.rtp = symbols.Contains("RTP");
 settings.vsPro = symbols.Contains("VEGETATION_STUDIO_PRO");

 settings.autoRef =
  GetAutoRef("MapMagic") &&
  GetAutoRef("MapMagic.Editor") &&
  GetAutoRef("Den.Tools") &&
  GetAutoRef("Den.Tools.Editor");

 return settings;
}
```

```
static void ApplySettings (Settings settings)
{
 BuildTargetGroup group = EditorUserBuildSettings.selectedBuildTargetGroup;
 string symbols = PlayerSettings.GetScriptingDefineSymbolsForGroup(group);

 ToggleKeyword(settings.native, "MM_NATIVE", ref symbols);
 ToggleKeyword(settings.debug, "MM_DEBUG", ref symbols);
 ToggleKeyword(settings.experimental, "MM_EXPERIMENTAL", ref symbols);

 ToggleKeyword(settings.cts, "CTS_PRESENT", ref symbols);
 ToggleKeyword(settings.megaSplat, "__MEGASPLAT__", ref symbols);
 ToggleKeyword(settings.microSplat, "__MICROSPLAT__", ref symbols);
 ToggleKeyword(settings.rtp, "RTP", ref symbols);
 ToggleKeyword(settings.vsPro, "VEGETATION_STUDIO_PRO", ref symbols);

 SetAutoRef("MapMagic", settings.autoRef);
 SetAutoRef("MapMagic.Editor", settings.autoRef);
 SetAutoRef("Den.Tools", settings.autoRef);
 SetAutoRef("Den.Tools.Editor", settings.autoRef);

 PlayerSettings.SetScriptingDefineSymbolsForGroup(group, symbols);
}


static void ToggleKeyword (bool val, string keyword, ref string symbols)
{
```

```
 //enabling
 if (val  &&  !symbols.Contains(keyword+";")  &&  !symbols.EndsWith(keyword))
 {
  symbols += (symbols.Length!=0? ";" : "") + keyword;
  Debug.Log(keyword + " Enabled");
 }

 //disabling
 if (!val  &&  (symbols.Contains(keyword+";")  ||  symbols.EndsWith(keyword)))
 {
  symbols = symbols.Replace(keyword,"");
  symbols = symbols.Replace(";;", ";");
  Debug.Log(keyword + " Disabled");
 }
}



static bool GetAutoRef (string assName)
{
 string asmdef = CompilationPipeline.GetAssemblyDefinitionFilePathFromAssemblyName(assName);
 using (StreamReader reader = new StreamReader(asmdef))
 {
  string asmdefText = reader.ReadToEnd();
  if (asmdefText.Contains("\"autoReferenced\": false"))
   return false;
  else return true;
```

```
 }
}

static void SetAutoRef (string assName, bool val)
{
 string asmdef = CompilationPipeline.GetAssemblyDefinitionFilePathFromAssemblyName(assName);
 string asmdefText;
 using (StreamReader reader = new StreamReader(asmdef))
  asmdefText = reader.ReadToEnd();

 bool getVal = asmdefText.Contains("\"autoReferenced\": false");
 if (val == getVal)
  return;

 string newText = "\"autoReferenced\": " + (val? "true":"false");
 if (asmdefText.Contains("\"autoReferenced\": false"))
  asmdefText = asmdefText.Replace("\"autoReferenced\": false", newText);
 else if (asmdefText.Contains("\"autoReferenced\": true"))
  asmdefText = asmdefText.Replace("\"autoReferenced\": true", newText);
 else
  asmdefText = asmdefText.Replace("}", newText + "\n}");

 using (StreamWriter writer = new StreamWriter(asmdef))
  writer.Write(asmdefText);

 AssetDatabase.Refresh();
```

```
}



[MenuItem ("Window/MapMagic/Settings")]
public static void ShowWindow ()
{
 SettingsWindow window = (SettingsWindow)GetWindow(typeof (SettingsWindow));

 Texture2D icon = Resources.Load("MapMagic/Icons/Window") as Texture2D;
 window.titleContent = new GUIContent("MapMagic Settings", icon);

 window.position = new Rect(100,100,300,340);
}

public static void ShowNet20Notification ()
{
 //#if !NET_STANDARD_2_0 won't work since editor is always NET_4
 if (PlayerSettings.GetApiCompatibilityLevel(EditorUserBuildSettings.selectedBuildTargetGroup) != ApiCo
  EditorUtility.DisplayDialog("MapMagic API Compatibility Warning", "MapMagic requires .NET 4.x API Co
   "Do you want to switch compatibility level now? \n\n"+
   "You can switch compatibility level manually in Project Settings -> Player -> Api Compatibility Level",
   "Switch to .NET 4.x",
   "Cancel"))
    PlayerSettings.SetApiCompatibilityLevel(EditorUserBuildSettings.selectedBuildTargetGroup, ApiCompa

}
```

```
 }

}


#endif
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine;

using Den.Tools;
using MapMagic.Nodes;

namespace MapMagic.Expose
{
 public static class Assigner
 {
  public static IUnit CopyAndAssign (IUnit unit, Exposed exp, Override ovd)
  {
   IUnit copy = unit.ShallowCopy();

   //overriding fields
   AssignOverrideToFields(copy, exp, ovd);

   //checking biome sub-override table
   if (copy is IBiome biome  &&  biome.Override != null)
   {
    biome.Override = new Override(biome.Override); //copying override table in copy too
    AssignOverrideToOverride(exp.EntriesById(unit.Id), biome.Override, ovd);
   }
```

```
//overriding layers
if (copy is IMultiLayer multi)  // or has overrided layer (to assign new
{
 bool overrideLayer = false;
 foreach (IUnit layer in multi.Layers)
  if (IsExposed(layer,exp))
   overrideLayer = true;

 if (overrideLayer)
 {
  //copy layers collection (since some will be re-assigned)
  ICollection<IUnit> srcLayers = multi.Layers;
  IUnit[] dstLayers = new IUnit[srcLayers.Count];
  srcLayers.CopyTo(dstLayers, 0);

  //copy/assign layers in copied collection
  for (int l=0; l<dstLayers.Length; l++)
   dstLayers[l] = CopyAndAssign(dstLayers[l], exp, ovd);

  //apply new collection itself
  multi.Layers = dstLayers;
 }
}

return copy;
```

```
}



public static bool IsExposed (IUnit unit, Exposed exp)
/// If element or any of it's layers exposed
{
 if (exp == null  ||  exp.Count == 0) //nothing exposed in exp
  return false;

 if (exp.Contains(unit.Id))
  return true;

 if (unit is IMultiLayer multi)  // or has overrided layer (to assign new
 {
  foreach (IUnit layer in multi.Layers)
   if (IsExposed(layer, exp)) //recursively, not with Contains
    return true;
 }

 return false;
}



private static void AssignOverrideToFields (IUnit unit, Exposed exp, Override ovd)
/// Assigns overriden values to standard (field) generator/layer
{
```

```
Type type = unit.GetType();
foreach (Exposed.Entry entry in exp.EntriesById(unit.Id))
{
 FieldInfo fieldInfo = type.GetField(entry.name);
 if (fieldInfo == null)
  continue; //might happen in biome, which has not fields, but ovd exposed

 if (fieldInfo.FieldType != entry.type  &&  !fieldInfo.FieldType.IsArray)
  throw new Exception("Recorded field type doesn't match generator field type. Possibly generator has ch



 object val;
 Calculator.Vector vec = entry.calculator.Calculate(ovd);

 if (entry.channel == -1) //non-channel
  val = vec.Convert(entry.type);
 else
 {
  object origVal = fieldInfo.GetValue(unit); //loading original value to take other channels
  val = vec.ConvertToChannel(origVal, entry.channel, entry.type);
 }

 if (entry.arrIndex == -1) //non-array
  fieldInfo.SetValue(unit, val);
 else if (unit is Nodes.MatrixGenerators.Blend200 blendGen) //special case for blend node
  blendGen.layers[entry.arrIndex].opacity = (float)val;
```

```
  else //array
  {
   Array arr = (Array)fieldInfo.GetValue(unit);
   arr.SetValue(val, entry.arrIndex);
   fieldInfo.SetValue(unit, arr);
  }
 }
}


private static void AssignOverrideToOverride (IEnumerable<Exposed.Entry> entries, Override ovdBase, O
/// Analog of AssignOverrideToFields, but overrides sub-graph defaults instead of generator fields
{
 foreach (Exposed.Entry entry in entries)
 {
  if (!ovdBase.TryGetValue(entry.name, out Type origType, out object origVal))
   continue;
   //do not override values that were not exposed

  if (origType != entry.type)
   continue;
   //do not override if they are of the same type



  object val;
  Calculator.Vector vec = entry.calculator.Calculate(ovdOverride);
```

```
    if (entry.channel == -1) //non-channel
     val = vec.Convert(entry.type);
    else
     val = vec.ConvertToChannel(origVal, entry.channel, entry.type);

    ovdBase[entry.name] = val;
   }
  }

 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine;

using Den.Tools;

[assembly:System.Runtime.CompilerServices.InternalsVisibleTo("MapMagic.Tests.Editor")]

namespace MapMagic.Expose
{
 //non-serialized (serialize string expressions instead)
 public partial class Calculator
 {
  public enum DataType { Null, Operator, Reference, Number };
  public DataType dataType;

  //operator
  public char action;
  public Calculator left;
  public Calculator right;

  //final variable
  public string reference;
  public float number; //'42' in expression
```

```
public string error; //here is written string that could not be parsed



public Vector Calculate (Dictionary<string,object> refVals=null)
/// Performs recursive calculations
{
 if (dataType == DataType.Operator)
 {
  Vector leftVal = left.Calculate(refVals);
  Vector rightVal = right.Calculate(refVals);
  return Action(leftVal, rightVal, action);
 }

 else if (dataType == DataType.Reference)
 {
  if (refVals.TryGetValue(reference, out object obj))
   return new Vector(obj);

  else
   return (Vector)0;
 }

 else if (dataType == DataType.Number)
  return (Vector)number;
```

```
 else
  throw new Exception("Could not perform calculation: operator type is Null");
}



public Vector Calculate (Override ovd)
/// The same but with Override
{
 if (dataType == DataType.Operator)
 {
  Vector leftVal = left.Calculate(ovd);
  Vector rightVal = right.Calculate(ovd);
  return Action(leftVal, rightVal, action);
 }

 else if (dataType == DataType.Reference)
 {
  if (ovd.TryGetValue(reference, out Type type, out object val))
   return new Vector(val);

  else
   return (Vector)0;
 }

 else if (dataType == DataType.Number)
  return (Vector)number;
```

```
 else
  throw new Exception("Could not perform calculation: operator type is Null");
}


public static Vector Action (Vector left, Vector right, char action)
{
 switch (action)
 {
  case '+': return left+right;
  case '-': return left-right;
  case '*': return left*right;
  case '/': return left/right;
  case '^': return left^right;
  case '.': return (Vector)left[(int)right];

  default: return new Vector();
 }
}


#region Parsing

 public static Calculator Parse (string str, bool prevActionIsDot=false)
```

```
/// Reads string and converts to CalcVar
/// prevActionIsDot just to properly treat channel .x
{
 Calculator var = new Calculator();

 //if string contains actions - filling left/right operators recursively
 if (ContainsAction(str))
 {
  //looking most low-priority action
  char minChr = ' ';
  int minPos = -1;
  int minPriority = int.MaxValue;

  foreach ((char chr, int pos, int priority) in StringActions(str))
  {
   if (priority < minPriority)
    { minChr = chr; minPos = pos; minPriority = priority; }
  }

  if (minPos == -1  ||  minChr == ' ')
   return null;

  //creating operator
  string str1 = str.Substring(0, minPos);
  string str2 = str.Substring(minPos + 1); //after action
```

```
 var.dataType = DataType.Operator;
 var.action = minChr;

 var.left = Parse(str1);
 var.right = Parse(str2, prevActionIsDot:var.action=='.');

 return var;
}



//trying to load channel
if (prevActionIsDot)
{
 int channel = StringToChannel(str);
 if (channel >= 0)
 {
  var.dataType = Calculator.DataType.Number;
  var.number = channel;

  return var;
 }
}



//trying to load reference
string reference = StringToReference(str);
```

```
if (reference != null)
{
 var.dataType = Calculator.DataType.Reference;
 var.reference = reference;

 return var;
}



//trying to load number
string numStr = str.Replace('(',' ').Replace(')',' '); //getting rid of ()
bool isNum = float.TryParse(numStr, out float num);

if (isNum)
{
 var.dataType = DataType.Number;
 var.number = num;

 return var;
}



//could not parse
var.dataType = Calculator.DataType.Null;
if (str.Length == 0 || str == " ")
 var.error = "Empty expression member";
```

```
 else
  var.error = "Could not parse '" + str + "'";
 return var;
}



internal static bool ContainsAction (string str)
{
 if (str.IndexOfAny(actionChars) >= 0)
  return true;

 else
 {
  int indexOfDot = str.IndexOf('.');
  if (indexOfDot >= 0  &&  indexOfDot < str.Length-1  &&  !char.IsDigit(str[indexOfDot+1]))
   return true;

  else
   return false;
 }

}
private static readonly char[] actionChars = new char[] { '+', '-', '*', '/', '^' };



internal static IEnumerable<(char chr, int pos, int priority)> StringActions (string str)
```

```
/// Iterates all +-*/ in string, and returns their [num] and execution priority (high eecuted first)
{
 int order = 0;
 int bracket = 0;

 for (int i=0; i<str.Length; i++)
 {
  char chr = str[i];

  bool isAction = false;
  bool isMult = false;
  bool isPow = false;
  bool isDot = false;

  if (chr=='(') bracket++;
  if (chr==')') bracket--;

  if (chr=='+' || chr=='-') isAction=true;
  if (chr=='*' || chr=='/') { isAction=true; isMult=true; }
  if (chr=='^') { isAction=true; isPow=true; }
  if (chr=='.' && i<str.Length-1 && !char.IsDigit(str[i+1])) { isAction=true; isDot=true; }

  if (isAction)
  {
   int priority = bracket*10000000 + (isDot ? 1000000 : 0) + (isPow ? 100000 : 0) + (isMult ? 10000 : 0) + (
   yield return (chr, i, priority);
```

```
   order--;
  }
 }
}



internal static string StringToReference (string str)
/// Checks whether string is possible reference and formats it properly if so (null if starts with digit could n
{
 string result = null;
 bool ended = false; //space or bracket or other non letter/digit symbol met

 foreach (char chr in str)
 {
  if (result == null) //starting read
  {
   if (chr == ' '  ||  chr == '(')  //skipping opening spaces and brackets
    continue;

   if (Char.IsDigit(chr))  //first is digit - error
    return null;

   if (Char.IsLetter(chr))
    result = "" + chr;
  }
```

```
  else //already reading
  {
   if (Char.IsLetterOrDigit(chr))
   {
    if (ended)
     return null;
    else
     result += chr;
   }

   else if (chr == ' '  ||  chr == ')')
    ended = true;

   else //if not letter or digit
    return null;
  }
 }

 return result;
}

internal static int StringToChannel (string str)
{
 int ch = -1;

 foreach (char chr in str)
```

```
 {
  if (chr == ' ') continue;

  if (ch >= 0) //if already read char, and following isn't space
   return -1;

  if (chr == 'x' || chr == 'r')
   { ch = 0; continue; }
  if (chr == 'y' || chr == 'g')
   { ch = 1; continue; }
  if (chr == 'z' || chr == 'b')
   { ch = 2; continue; }
  if (chr == 'w' || chr == 'a')
   { ch = 3; continue; }

  return -1;
 }

 return ch;
}



public bool CheckValidity (out string error)
/// Checks if it was loaded correctly, no nulls for left/right and reference
{
 error = null;
```

```
if (dataType == DataType.Null)
 { error = this.error; return false; }

if (dataType == DataType.Operator)
{
 if (!(action=='+' || action=='-' || action=='*' || action=='/' || action=='^' || action=='.'))
  { error = "Unknown operator type " + action; return false; }

 if (left == null)
  { error = "Left operator not defined"; return false; }

 if (right == null)
  { error = "Right operator not defined"; return false; }

 bool leftValid = left.CheckValidity(out string leftError);
 if (!leftValid) { error = leftError; return false; }

 bool rightValid = right.CheckValidity(out string rightError);
 if (!rightValid) { error = rightError; return false; }
}

if (dataType == DataType.Reference  &&  reference == null)
 { error = "Reference not defined"; return false; }

return true;
```

```
}



public string CheckOverrideAssign (Override ovd)
/// Checks that override contains all the calculator (and it's subs) references
/// Returns the first occurrence of reference that is not in override. Null if all assigned
{
 if (dataType == DataType.Reference)
 {
  if (!ovd.Contains(reference)  ||  !ovd.Contains(reference))
   return reference;
 }

 if (dataType == DataType.Operator)
 {
  string leftRef = left.CheckOverrideAssign(ovd);
  if (leftRef != null) return leftRef;

  string rightRef = right.CheckOverrideAssign(ovd);
  if (rightRef != null) return rightRef;
 }

 return null;
}
```

```
public IEnumerable<string> AllRefenrences ()
{
 if (dataType == DataType.Reference)
  yield return reference;

 if (dataType == DataType.Operator)
 {
  foreach (string leftReference in left.AllRefenrences())
   yield return leftReference;

  foreach (string rightReference in right.AllRefenrences())
   yield return rightReference;
 }
}


public bool ContainsReference (string reference)
{
 if (dataType == DataType.Reference)
  return this.reference == reference;

 if (dataType == DataType.Operator)
 {
  if (left.ContainsReference(reference)) return true;
  if (right.ContainsReference(reference)) return true;
 }
```

```
 return false;
}

/* public string CheckOverrideType (Override ovd)
/// Checks that overridden values (have the same type as calculator refs) are floats or ints
/// Returns the first occurrence of reference whose type don't match
/// Skips references not in override
{
 if (dataType == DataType.Reference)
 {
  if (!ovd.TryGetValue(reference, out object val, out Type type))
   return null;

  if (type != typeof(float)  &&  type != typeof(int))
   return reference;
 }

 if (dataType == DataType.Operator)
 {
  string leftRef = left.CheckOverrideType(ovd);
  if (leftRef != null) return leftRef;

  string rightRef = right.CheckOverrideType(ovd);
  if (rightRef != null) return rightRef;
 }
```

```
 return null;
}*/

#endregion



#region ToString

 public override string ToString ()
 {
  if (dataType == DataType.Operator)
  {
   bool leftBrackets = left.dataType == DataType.Operator;
   bool rightBrackets = right.dataType == DataType.Operator;

   /*if ((action == '+' || action == '-')  &&  (left.action == '+' || left.action == '-')) leftBrackets = false;
   if ((action == '*' || action == '/')  &&  (left.action == '*' || left.action == '/')) leftBrackets = false;
   if ((action == '+' || action == '-')  &&  (left.action == '*' || left.action == '/')) leftBrackets = false;
   if ((action == '+' || action == '-')  &&  (right.action == '*' || right.action == '/')) rightBrackets = false;*/

   if (!leftBrackets && !rightBrackets)
    return $"{left.ToString()} {action} {right.ToString()}";
   else if (leftBrackets && !rightBrackets)
    return $"({left.ToString()}) {action} {right.ToString()}";
   else if (!leftBrackets && rightBrackets)
```

```
      return $"{left.ToString()} {action} ({right.ToString()})";
     else
      return $"({left.ToString()}) {action} ({right.ToString()})";
    }

    else if (dataType == DataType.Reference)
     return reference;

    else
     return number.ToString();
   }

  #endregion

 }

}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine;

using Den.Tools;
using MapMagic.Nodes;

namespace MapMagic.Expose
{
 [Serializable]
 public class Exposed : ISerializationCallbackReceiver
 {
  [Serializable]
  public class Entry : ISerializationCallbackReceiver
  {
   public ulong id;
   public string name;
   public int channel = -1;
   public int arrIndex = -1;

   public string expression; //serializes calculator and stores expression in an original form

   [NonSerialized] public Type type;  //always matches the field type. Evel if one channel of Vector3 expose
   [NonSerialized] public Calculator calculator;
```

```
public Entry () {} //for serializer

public Entry (ulong id, string name, int channel, int arrIndex, string expression, Type type, Calculator calc
 { this.id = id; this.name=name; this.channel=channel; this.arrIndex=arrIndex; this.expression=expression

public Entry (ulong id, string name, int channel, int arrIndex, string expression, Type type)
 { this.id = id; this.name=name; this.channel=channel; this.arrIndex=arrIndex; this.expression=expression

public Entry (ulong id, string name, string expression, Type type, Calculator calculator)
 { this.id = id; this.name=name; this.channel=-1; this.expression=expression; this.type=type; this.calculato

public Entry (ulong id, string name, string expression, Type type)
 { this.id = id; this.name=name; this.channel=-1; this.expression=expression; this.type=type; calculator=C


#region Serialize

 [SerializeField] private string serType;

 public void OnBeforeSerialize ()
 {
  serType = type.AssemblyQualifiedName;
 }
```

```
  public void OnAfterDeserialize ()
  {
   if (serType==null) return;
   type = Type.GetType(serType);
   calculator = Calculator.Parse(expression);
  }

 #endregion
}

[NonSerialized] Dictionary<ulong,Entry[]> idToExpressions = new Dictionary<ulong,Entry[]>();
 //used as multi-level dict: id -> name -> channel



public Entry this[ulong id, string name, int channel=-1, int arrIndex=-1]
/// Returns null if entry is not listed
{get{
 Entry[] entries;
 if (!idToExpressions.TryGetValue(id, out entries))
  return null;

 else
 {
  Entry exp = entries.FindMember(e => e.name==name && e.channel==channel && e.arrIndex==arrIndex
  return exp;
 }
```

```
}}


public Entry this[Entry entry]
{set{
 Entry[] entries = idToExpressions[entry.id];
 int num = entries.Find(e => e.name==entry.name && e.channel==entry.channel && e.arrIndex==entry.ar


 if (num >= 0)
  entries[num] = value;
 else
  throw new Exception ("Value is not contained in array");
}}


public Entry[] this[ulong id]
/// Returns null if entry is not listed
{get{
 if (!idToExpressions.TryGetValue(id, out Entry[] entries))
  return entries;
 else
  return null;
}}


public string GetExpression (ulong id, string name, int channel=-1, int arrIndex=-1)
```

```
/// If we don't want to deal with entries
{
 Entry entry = this[id, name, channel, arrIndex];
 return entry!=null ? entry.expression : null;
}

public Calculator GetCalculator (ulong id, string name, int channel=-1, int arrIndex=-1)
/// If we don't want to deal with entries
{
 Entry entry = this[id, name, channel, arrIndex];
 return entry!=null ? entry.calculator : null;
}

public Type GetType (ulong id, string name, int channel=0)
/// If we don't want to deal with entries
{
 Entry entry = this[id,name,channel];
 return entry!=null ? entry.type : null;
}


public void Add (Entry entry, bool overwrite=false)
/// If overwrite could be used as ForceAdd
{
 Entry[] entries;
 if (!idToExpressions.TryGetValue(entry.id, out entries))
```

```
 {
  entries = new Entry[1];
  entries[0] = entry;
  idToExpressions.Add(entry.id, entries);
 }
 else
 {
  int num = entries.Find(e => e.name==entry.name && e.channel==entry.channel && e.arrIndex==entry.ar
  if (num >= 0)
  {
   if (overwrite) entries[num] = entry;
   else
    throw new Exception ("Can't add value since it's already in array");
  }
  else
  {
   ArrayTools.Add(ref entries, entry);
   idToExpressions[entry.id] = entries; //ArrayTools ref does not update dictionary
  }
 }
}



public void AddRange (Entry[] entries, bool overwrite=false)
/// Adds exposed values for one gen only
{
```

```
 foreach (Entry entry in entries)
  Add(entry, overwrite);
}



public void AddRange (Exposed other, bool overwrite=false)
/// Adds exposed values from other graph (for duplicating or copy/paste generators)
{
 foreach (Entry[] entries in other.idToExpressions.Values)
  foreach (Entry entry in entries)
   Add(entry, overwrite);
}



public bool Remove (ulong id, string name, int channel, int arrIndex)
/// Unexpose one entry only
/// True if entry was really removed
{
 Entry[] exps;
 if (!idToExpressions.TryGetValue(id, out exps))
  return false;

 else
 {
  int num = exps.Find(e => e.name==name && e.channel==channel && e.arrIndex==arrIndex);
  if (num >= 0)
```

```
  {
   ArrayTools.RemoveAt(ref exps, num);
   idToExpressions[id] = exps; //ref from dict doesn't follow

   return true;
  }
 }

 return false;
}



public bool Remove (ulong id, string name, int arrIndex)
/// Unexpose all entries with the same name, no matter of their channel
{
 Entry[] exps;
 if (!idToExpressions.TryGetValue(id, out exps))
  return false;

 else
 {
  bool found = false;
  while (true) //there could be several channels with this name
  {
   int num = exps.Find(e => e.name==name && e.arrIndex==arrIndex);
```

```
   if (num >= 0)
   {
    ArrayTools.RemoveAt(ref exps, num);
    idToExpressions[id] = exps; //ref from dict doesn't follow
    found = true;
   }

   else
    break;
  }

  return found;
 }
}



public bool Remove (ulong id)
/// Unexpose all of the gen (on gen remove or fn graph reassign)
/// /// True if entry was really removed
{
 if (idToExpressions.ContainsKey(id))
 {
  idToExpressions.Remove(id);
  return true;
 }
```

```
 return false;
}

public bool Contains (ulong id) => idToExpressions.ContainsKey(id);

public bool Contains (ulong id, string name) //contains any channel with given name
{
 Entry[] entries;
 if (!idToExpressions.TryGetValue(id, out entries))
  return false;
 else
  return entries.Contains(e => e.name==name);
}

public bool Contains (ulong id, string name, int channel, int arrIndex)
{
 Entry[] entries;
 if (!idToExpressions.TryGetValue(id, out entries))
  return false;
 else
 {
  int num = entries.Find(e => e.name==name && e.channel==channel && e.arrIndex==arrIndex);
  return num >= 0;
 }
}
```

```
public int Count
{get{
 int count = 0;
 foreach (Entry[] entries in idToExpressions.Values)
  count += entries.Length;
 return count;
}}



public IEnumerable<Entry> EntriesById (ulong id)
{
 if (!idToExpressions.TryGetValue(id, out Entry[] exps))
  yield break;

 foreach (Entry entry in exps)
  yield return entry;
}

public IEnumerable<Entry> EntiriesByReference (string reference)
{
 foreach (Entry[] exps in idToExpressions.Values)
  foreach (Entry entry in exps)
  {
   if (entry.calculator.ContainsReference(reference))
    yield return entry;
```

```
  }


}

public IEnumerable<Entry> AllEntries ()
{
 foreach (Entry[] exps in idToExpressions.Values)
  foreach (Entry entry in exps)
   yield return entry;
}

public IEnumerable<ulong> AllIds ()
{
 foreach (ulong key in idToExpressions.Keys)
  yield return key;
}

public IEnumerable<IUnit> AllUnits (Graph graph)
{
 foreach (IUnit unit in graph.AllUnits())
  if (idToExpressions.ContainsKey(unit.Id))
   yield return unit;
}



public void ReplaceIds (Dictionary<ulong,ulong> oldNewIds)
```

```
/// Changes the ids of exposed values (when chnging gen id on duplicating or copy/paste generators)
{
 ulong[] oldIds = oldNewIds.Keys.ToArray();
 ulong[] newIds = oldNewIds.Values.ToArray();

 for (int i=0; i<oldIds.Length; i++)
 {
  ulong oldId = oldIds[i];
  ulong newId = newIds[i];

  if (idToExpressions.TryGetValue(oldId, out Entry[] entries))
  {
   if (idToExpressions.ContainsKey(newId))
    throw new Exception("ReplaceIds: idToExpressions already contains this newid: " + newId);

   foreach (Entry entry in entries)
    entry.id = newId;

   idToExpressions.Remove(oldId);
   idToExpressions.Add(newId, entries);
  }
 }
}

public void RemoveUnused (Graph graph)
```

```
/// Clears exposed expressions that do not have generator related with them
{
 HashSet<ulong> unused = new HashSet<ulong>(idToExpressions.Keys);

 foreach (IUnit unit in graph.AllUnits())
  unused.Remove(unit.Id);

 foreach (ulong uid in unused)
  idToExpressions.Remove(uid);
}

public IEnumerable<IUnit> UnitsByReference (Graph graph, string reference)
/// Iterates generators in graph that has expose with this variable name
{
 foreach (IUnit unit in graph.AllUnits())
 {
  if (idToExpressions.TryGetValue(unit.Id, out Entry[] exps))
  {
   foreach (Entry entry in exps)
    if (entry.calculator.ContainsReference(reference))
    {
     yield return unit;
     break;
    }
  }
```

```
 }
}


#region Serialization

 [SerializeField] private Entry[] serEntries;

 public void OnBeforeSerialize ()
 {
  serEntries = new Entry[Count];

  int i=0;
  foreach (Entry entry in AllEntries())
  {
   serEntries[i] = entry;
   i++;
  }
 }

 public void OnAfterDeserialize ()
 {
  idToExpressions.Clear(); //just in case

  if (serEntries==null)
   return;
```

```
    foreach (Entry entry in serEntries)
      Add(entry);
   }

  #endregion
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine;

using Den.Tools;
using MapMagic.Nodes;

namespace MapMagic.Expose
{
 public class Loader
 /// Simplified Calculator version to store non-expression objects
 /// Non-serialized (serialize string expressions instead)
 {
  public string reference;

  public string typeName;
  private Type type;

  public string error;


  /*public object Load (Override ovd)
  {
   if (ovd!=null  &&  ovd.TryGetFloat(reference, out float val)  &&  )
```

```
    return val;
  }*/
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine;

using Den.Tools;
using Den.Tools.Serialization;
using MapMagic.Nodes;

namespace MapMagic.Expose
{
 [Serializable]
 public class Override : ISerializationCallbackReceiver
 {
  [NonSerialized] private DictionaryOrdered<string,ObjType> dict = new DictionaryOrdered<string,ObjType
   //dictionary of structs - for each value type is determined by is/GetType
   //for unity objects using UnityObject

  private struct ObjType
  {
   public object obj;
   public Type type;

   public ObjType (object obj, Type type) { this.obj=obj; this.type=type; }
  }
```

```
public Override () { }
public Override (Override src) { dict = new DictionaryOrdered<string,ObjType>(src.dict); }



public object this[string key]
{
 get => dict[key].obj;
 set => dict[key] = new ObjType(value, value.GetType());
}



public bool TryGetValue (string name, out Type type, out object obj)
{
 if (dict.TryGetValue(name, out ObjType ot))
 {
  obj = ot.obj; type = ot.type;
  return true;
 }

 else
 {
  obj=default; type=null;
  return false;
 }
```

```
}

public void Add (string name, Type type, object val) => dict.Add(name, new ObjType(val, type));

public void AddDefault (string name, Type type)
/// Adding default value (checking it's not null for value types)
{
 object val = default;
 if (type.IsValueType)
  val = Activator.CreateInstance(type);

 Add(name, type, val);
}

public void SetOrAdd (string name, Type type, object val)
/// Adds value if it's not in dict
{
 if (dict.ContainsKey(name))
  dict[name] = new ObjType(val, type);
 else
  dict.Add(name, new ObjType(val, type));
}

public void SetIfContains (string name, Type type, object val)
/// Sets only if name key is in dict and type match
```

```
{
 if (dict.TryGetValue(name, out ObjType ot)  &&  ot.type == type)
  dict[name] = new ObjType(val, type);
}

public void RemoveAt (int num) => dict.RemoveAt(num);

public void Switch (int n1, int n2) => dict.Switch(n1, n2);

public int Count => dict.Count;

public bool Contains (string name) => dict.Contains(name);

public void Clear () => dict.Clear();


public (string,Type,object) GetOverrideAt (int num)
/// Reads name, dataand object at specified index
{
 string name = ((IList<string>)dict)[num]; //default call returns value. Treating dict as list to get key
 ObjType ot = dict[name];

 return (name, ot.type, ot.obj);
}

public void SetOverrideAt (int num, string name, Type type, object obj)
```

```
/// Re-writes previous name, data and object at specified index
{
 string prevName = ((IList<string>)dict)[num];

 if (name != prevName) //removing entry completely
 {
  dict.RemoveAt(num);
  dict.Add(name, new ObjType(obj,type));
 }

 else
  dict[prevName] = new ObjType(obj,type);
}



public void AddExposed (Exposed.Entry entry, Graph graph=null)
/// Adds all references from entry expression (there could be several). Tries to assign the generator fieldva
/// Does nothing if reference already assigned
{
 // assigning type of float for exposed channels
 Type assignType;
 if (entry.channel >= 0)
 {
  if (entry.type == typeof(Coord))
   assignType = typeof(int);
  else
```

```
  assignType = typeof(float);
}
else
 assignType = entry.type;

//getting default value
object defaultVal = null;
if (graph != null)
{
 defaultVal = GetFieldValue(graph, entry.id, entry.name);

 //convert to vec and back to set channel and just to make sure it's of proper type
 Calculator.Vector vec = new Calculator.Vector(defaultVal);
 if (entry.channel >= 0)  //picking the right channel if it's defined
  vec.Unify(entry.channel);
 defaultVal = vec.Convert(assignType);
}

//assigning
while (true) //could be several variables in one expression (yep, all of them will get default value)
{
 string assignName = entry.calculator.CheckOverrideAssign(this);
 if (assignName == null) //nothing more to assign
  break;

 if (defaultVal!=null)
```

```
   graph.defaults.Add(assignName, assignType, defaultVal);
  else
   graph.defaults.AddDefault(assignName, assignType);
 }
}


public void AddAllExposed (Exposed exp, Graph graph=null)
/// Adds all exposed values (from graph)
{
 foreach (Exposed.Entry entry in exp.AllEntries())
  AddExposed(entry, graph);
}


public void RemoveAllUnused (Exposed exp)
/// Removes all values that are not used in Expose
{
 HashSet<string> allReferences = new HashSet<string>();

 foreach (Exposed.Entry entry in exp.AllEntries())
  foreach (string reference in entry.calculator.AllRefenrences())
   allReferences.Add(reference);

 List<string> refsToRemove = new List<string>();
```

```
 foreach (string reference in dict.Keys)
  if (!allReferences.Contains(reference))
   refsToRemove.Add(reference);

 foreach (string reference in refsToRemove)
  dict.Remove(reference);
}



private static object GetFieldValue (Graph graph, ulong id, string name)
/// Gets generator original field value
/// To find AddExposed default
{
 IUnit unit = null;
 foreach (IUnit aunit in graph.AllUnits())
  if (aunit.Id == id)
   { unit = aunit; break; }

 if (unit == null)
  return null;

 FieldInfo field = unit.GetType().GetField(name);
 if (field == null)
  return null;

 object val = field.GetValue(unit);
```

```
 return val;
}



public void Sync (Override ovd)
/// Adds new overriden values (with ovd values) and removes values which are not in ovd
/// Do not change values if they already exist
{
 for (int i=dict.Count-1; i>=0; i--)
 {
  string name = ((IList<string>)dict)[i];
  Type type = dict[name].type;

  if (!ovd.dict.Contains(name)  ||  ovd.dict[name].type!=type) //removing if improper type too
   RemoveAt(i);
 }

 for (int i=0; i<ovd.dict.Count; i++)
 {
  string ovdName = ((IList<string>)ovd.dict)[i];

  if (!dict.Contains(ovdName))
   dict.Add(ovdName, ovd.dict[ovdName]);
 }
}
```

```
 #region Serialization

  [SerializeField] private string serializedDict;
  [SerializeField] private UnityEngine.Object[] unityObjects;
  //name orders is serialized on itself

  public void OnBeforeSerialize ()
  {
   serializedDict = Den.Tools.Serialization.Serializer.Serialize(dict, out unityObjects);
  }

  public void OnAfterDeserialize ()
  {
   if (serializedDict != null  &&  unityObjects != null)
    dict = (DictionaryOrdered<string,ObjType>)Den.Tools.Serialization.Serializer.Deserialize(serializedDict,

   #if UNITY_2021_2_OR_NEWER
   // Unity 2021.2 Beta cannot use re-serialized dictionary. Cannot find key that definitely in it. Probably sor
   // This one re-creates dictionary from scratch
   dict.ReCreateDictionary();
   #endif
  }

 #endregion
}
```

```
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine;

using Den.Tools;

namespace MapMagic.Expose
{
 public partial class Calculator
 {
  public struct Vector
  /// Uniform interface for all Vector2, Vector3, Coord, etc
  /// Calculator performs operations not on floats, ints, etc, but on these vectors instead
  {
   public float x;
   public float y;
   public float z;
   public float w;

   public UnityEngine.Object uobj;  //yep, vector has a unity object. Just to allow Calculator output it

   private Vector (float x, float y, float z, float w) { this.x=x; this.y=y; this.z=z; this.w=w; uobj=null; }
```

```
public static explicit operator Vector(float f) => new Vector(f, f, f, f);

public static explicit operator Vector(int i) => new Vector(i, i, i, i);

public static explicit operator Vector(double d) => new Vector((float)d, (float)d, (float)d, (float)d);

public static explicit operator Vector(Vector2 v) => new Vector(v.x, v.y, v.y, 0); //for Vector2 and Vector2D

public static explicit operator Vector(Vector2D v) => new Vector(v.x, v.z, v.z, 0);

public static explicit operator Vector(Coord c) => new Vector(c.x, c.z, c.z, 0);

public static explicit operator Vector(Vector3 v) => new Vector(v.x, v.y, v.z, 0);

public static explicit operator Vector(Vector4 v) => new Vector(v.x, v.y, v.z, v.w);

public static explicit operator Vector(Color c) => new Vector(c.r, c.g, c.b, c.a);

public static explicit operator Vector(bool b) => b ? new Vector(1,1,1,1) : new Vector(0,0,0,0);

public static explicit operator Vector(UnityEngine.Object o)

{

 Vector vec = o!=null ? new Vector(1,1,1,1) : new Vector(0,0,0,0);

 vec.uobj = o;

 return vec;

}


public static explicit operator float(Vector v) => v.x;

public static explicit operator int(Vector v) => Mathf.RoundToInt(v.x);

public static explicit operator double(Vector v) => v.x;

public static explicit operator Vector2(Vector v) => new Vector2(v.x, v.y);

public static explicit operator Vector2D(Vector v) => new Vector2D(v.x, v.z);

public static explicit operator Coord(Vector v) => new Coord( Mathf.RoundToInt(v.x), Mathf.RoundToInt(v

public static explicit operator Vector3(Vector v) => new Vector3(v.x, v.y, v.z);

public static explicit operator Vector4(Vector v) => new Vector4(v.x, v.y, v.z, v.w);

public static explicit operator Color(Vector v) => new Color(v.x, v.y, v.z, v.w);
```

```
public static explicit operator bool(Vector v) => v.x>0.00001f;
public static explicit operator UnityEngine.Object(Vector v)
{
 if (Mathf.Abs(v.x)<0.00001f &&  Mathf.Abs(v.y)<0.00001f  &&  Mathf.Abs(v.z)<0.00001f  &&  Mathf.Abs(v
  return null;
 else return v.uobj;
}



public Vector (object obj)
{
 switch (obj)
 {
  case float fobj: this = (Vector)fobj; break;
  case int iobj: this = (Vector)iobj; break;
  case double dobj: this = (Vector)dobj; break;
  case Vector2 v2obj: this = (Vector)v2obj; break;
  case Vector2D v2dobj: this = (Vector)v2dobj; break;
  case Coord cdobj: this = (Vector)cdobj; break;
  case Vector3 v3obj: this = (Vector)v3obj; break;
  case Vector4 v4obj: this = (Vector)v4obj; break;
  case Color cobj: this = (Vector)cobj; break;
  case bool bobj: this = (Vector)bobj; break;
  case UnityEngine.Object uobj: this = (Vector)uobj; break;
  default: this=new Vector(); break;
 }
```

```
}

public object Convert (Type type)
/// Casts Vector to given type
/// Takes a specified channel for int, float and bool
/// dynamic requires CSharp assembly
{
 if (type==typeof(float)) return (float)this;
 if (type==typeof(int)) return (int)this;
 if (type==typeof(double)) return (double)this;
 if (type==typeof(Vector2)) return (Vector2)this;
 if (type==typeof(Vector2D)) return (Vector2D)this;
 if (type==typeof(Coord)) return (Coord)this;
 if (type==typeof(Vector3)) return (Vector3)this;
 if (type==typeof(Vector4)) return (Vector4)this;
 if (type==typeof(Color)) return (Color)this;
 if (type==typeof(bool)) return (bool)this;
 if (typeof(UnityEngine.Object).IsAssignableFrom(type)) return (UnityEngine.Object)this;
 if (typeof(Enum).IsAssignableFrom(type)) return (int)this;
 return null;
}

public object ConvertToChannel (object wholeVal, int channel, Type type)
/// Sets the channel of Vector3, Vector4 to vec value, converted to float
/// Actually converts object to vec, sets channel, then converts back
```

```
{
 Vector wholeVec = new Vector(wholeVal);
 wholeVec[channel] = (float)this;
 return wholeVec.Convert(type);
}



public void Unify (int channel)
/// Makes all 4 channels = specified channel
/// Then can convert to float - this way can take one channel
{
 float val = this[channel];
 x = val; y=val; z=val; w=val;
}



public float this[int i]
{
 get {
  switch (i)
  {
   case 0: return x;
   case 1: return y;
   case 2: return z;
   case 3: return w;
   default: return 0;
```

```
  }
 }

 set {
  switch (i)
  {
   case 0: x = value; break;
   case 1: y = value; break;
   case 2: z = value; break;
   case 3: w = value; break;
  }
 }
}

public static Vector operator + (Vector c1, Vector c2) { c1.x+=c2.x; c1.y+=c2.y; c1.z+=c2.z; c1.w+=c2.w;
public static Vector operator - (Vector c1, Vector c2) { c1.x-=c2.x; c1.y-=c2.y; c1.z-=c2.z; c1.w-=c2.w; retu
public static Vector operator * (Vector c1, Vector c2) { c1.x*=c2.x; c1.y*=c2.y; c1.z*=c2.z; c1.w*=c2.w; ret
public static Vector operator / (Vector c1, Vector c2)
{
 if (c2.x!=0) c1.x/=c2.x;
 if (c2.y!=0) c1.y/=c2.y;
 if (c2.z!=0) c1.z/=c2.z;
 if (c2.w!=0) c1.w/=c2.w;
 return c1;
}
public static Vector operator ^ (Vector c1, Vector c2)
```

```
    { c1.x=(float)Math.Pow(c1.x,c2.x); c1.y=(float)Math.Pow(c1.y,c2.y); c1.z=(float)Math.Pow(c1.z,c2.z); c1.w
    //not XOR, but exponent


  }
 }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
//using Den.Tools.Segs;
using Den.Tools.Splines;
using MapMagic.Core;  //used once to get tile size
using MapMagic.Products;
using MapMagic.Expose.GUI;
using MapMagic.Expose;

namespace MapMagic.Nodes.GUI
{
 public static class CellExpose
 {
  public static void Expose (this Cell cell, ulong unitId, string fieldName, Type fieldType, int chNum=-1, int a
  /// Adds right-click menu to cell and draw exposed field over standard one if cell is exposed
  /// Automatically does so for all vector cells (maintaining their channel numbers)
  /// Checks whether this field is exposed only if genExposed=true (genExposed is false on drawing class to
  {
```

```
//fast skipping (since it will be called in every field)
if (!genExposed && UI.current.mouseButton!=1)
 return;

//if exposed - overdrawing exposed fields
bool fieldExposed = genExposed  &&  GraphWindow.current.graph.exposed.Contains(unitId, fieldName,

if (fieldExposed && !UI.current.layout)
{
 foreach (Cell subCell in cell.SubCellsRecursively(includeSelf:true))
 {
  subCell.inactive = true; //turning off controls like field drag
  //making all cells disable recursively since they won't make it AFTER deactivating them
  //could not be made inside special.HasFlag because flag is set only on repaint - when it's too late to ma

  if (subCell.special.HasFlag(Cell.Special.Field))
  {
   subCell.Activate();
   ExposedField(unitId, fieldName, fieldType, chNum, arrIndex);
   subCell.Dispose();
  }
 }
}

//adding to right-click menu
if (UI.current.mouseButton==1)
```

```
 UI.current.cellObjs.ForceAdd(new RightClickExpose(unitId, fieldName, fieldType, chNum, arrIndex), cell



 //if vector - calling per-channel expose cell first
 if (cell.special.HasFlag(Cell.Special.Vector) && chNum < 0)
 {
  foreach (Cell subCell in cell.SubCellsRecursively())
  {
   if (subCell.special.HasFlag(Cell.Special.VectorX))
    subCell.Expose(unitId, fieldName, fieldType, 0);

   if (subCell.special.HasFlag(Cell.Special.VectorY))
    subCell.Expose(unitId, fieldName, fieldType, 1);

   if (subCell.special.HasFlag(Cell.Special.VectorZ))
    subCell.Expose(unitId, fieldName, fieldType, 2);

   if (subCell.special.HasFlag(Cell.Special.VectorW))
    subCell.Expose(unitId, fieldName, fieldType, 3);
  }
 }
}



public static bool ExposableClass (object genObj, ulong genId, string category=null)
// Synonym of Draw.Class but enhanced with expose
```

```
/// True if drawn any field
{
 Graph graph = GraphWindow.current.graph;
 bool genExposed = graph.exposed!=null ? graph.exposed.Contains(genId) : false;  //has this generator e

 void ExposeCellAction (FieldInfo field, Cell cell)
 {
  if (genExposed || UI.current.mouseButton==1)
   cell.Expose(genId, field.Name, field.FieldType);
 }

 return Draw.Class(genObj, category, ExposeCellAction);
}


/// Drawing field of a value that is currently exposed
private static void ExposedField (ulong genId, string fieldName, Type fieldType, int chNum, int arrIndex)
{
 if (UI.current.layout) //will need field widt, and what's the point to do it in layout?
  return;

 Graph graph = GraphWindow.current.graph;

 float fieldWidth = Cell.current.finalSize.x; //we are not in layout

 string label = graph.exposed.GetExpression(genId, fieldName, channel:chNum, arrIndex:arrIndex);
```

```
    float labelWidth = UI.current.styles.label.CalcSize( new GUIContent(label) ).x;
    if (labelWidth > fieldWidth-20)
     label = "...";

    Draw.Label(label, UI.current.styles.field);

    Vector2 center = Cell.current.InternalCenter;
    Vector2 iconPos = new Vector2(center.x + fieldWidth/2 - 10, center.y);
    Draw.Icon(UI.current.textures.GetTexture("DPUI/Icons/Expose"), iconPos, scale:0.5f);

    //if (Draw.Button(visible:false))  //cell inactive
    if (UI.current.mouseButton==0 && Cell.current.Contains(UI.current.mousePos))
     ExposeWindow.ShowWindow(graph, genId, fieldName, fieldType, chNum, arrIndex);

    #if UNITY_EDITOR
//   Rect rect = Cell.current.GetRect(UI.current.scrollZoom);
//   UnityEditor.EditorGUIUtility.AddCursorRect(rect, UnityEditor.MouseCursor.Zoom);
    #endif
  }
 }
}
```

```
ï»¿using System;
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;

using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Expose;
using MapMagic.Products;
using MapMagic.Previews;

namespace MapMagic.Expose.GUI
{
 //[EditoWindowTitle(title = "MapMagic Graph")]  //it's internal Unity stuff
 [Serializable]
 public class ExposeWindow : EditorWindow
 {
  [SerializeField] private Graph graph;
  [SerializeField] private Exposed.Entry entry;
```

```
[SerializeField] private UI ui = new UI();

private static readonly Dictionary<int,string> chToName = new Dictionary<int, string> { {-1,"None"}, {0,"X"

public void OnGUI () => ui.Draw(DrawParams, inInspector:false);

private void DrawParams ()
{
 if (entry==null || graph==null) //happens after re-compile
  Close();

 using (Cell.Padded(5,5,5,5))
 {
  Cell.current.fieldWidth = 0.7f;

  //using (Cell.LineStd) Draw.DualLabel("Generator", genName);
  using (Cell.LineStd) Draw.DualLabel("Node Id", entry.id.ToString());
  using (Cell.LineStd) Draw.DualLabel("Field", $"{entry.name.Nicify()} ({entry.type})");
  using (Cell.LineStd) Draw.DualLabel("Channel", chToName[entry.channel]);
  using (Cell.LineStd) Draw.DualLabel("Array Index", entry.arrIndex==-1 ? "N/A" : entry.arrIndex.ToString(

  Cell.EmptyLinePx(10);

  //expression field
  using (Cell.LineStd)
```

```
 Draw.Label("Expression:");

using (Cell.LineStd)
 Draw.Field(ref entry.expression);

//parsings/checking expression (every frame, since we can add or remove overrided variables from grap
string error = null;
string warning = null;
string result = null;
string assign = null; //value name to assign override to graph with button

if (entry.expression.Length == 0)
 result = "Non exposed";

else
{
 entry.calculator = Calculator.Parse(entry.expression);

 if (!entry.calculator.CheckValidity(out string anyError))
  error = anyError; //error assigned only if non-valid

 else
 {
  assign = entry.calculator.CheckOverrideAssign(graph.defaults);
  if (assign != null)
   warning = $"Graph variable '{assign}' is not assigned. \nUsing 0 instead.";
```

```
 /* else  //currently type can't be wroong
  {
   warning = calculator.CheckOverrideType(graph.defaults);
   if (warning != null)
    warning = $"Graph variable '{warning}' has the wrong type. \nUsing 0 instead.";
  }*/

  Type assignedType = entry.channel>=0 ? typeof(float) : entry.type;
  result = $"All okay. Default result: {entry.calculator.Calculate(graph.defaults).Convert(assignedType)}";
 }
}

Cell.EmptyLinePx(10);

//errors/warnings
using (Cell.LinePx(46))
{
 using (Cell.RowPx(20))
 {
  using (Cell.LinePx(16))
  {
   if (error != null) Draw.Icon(UI.current.textures.GetTexture("DPUI/Icons/Error"));
   else if (warning != null) Draw.Icon(UI.current.textures.GetTexture("DPUI/Icons/Warning"));
   else Draw.Icon(UI.current.textures.GetTexture("DPUI/Icons/Okay"));
  }
  Cell.EmptyLine();
```

```
 }

 using (Cell.Row)
 {
  if (error != null) Draw.Label(error, style:UI.current.styles.topLabel);
  else if (warning != null) Draw.Label(warning, style:UI.current.styles.topLabel);
  else Draw.Label(result, style:UI.current.styles.topLabel);
 }

 //assign button
 if (assign != null)
  using (Cell.RowPx(60))
  {
   using (Cell.LinePx(22))
    if (Draw.Button("Assign"))
    {
     graph.defaults.AddExposed(entry,graph);
     OverrideInspector.RefreshMapMagic(entry.name);
    }

   Cell.EmptyLine();
  }
}

//Cell.EmptyLinePx(10);
```

```
//okay/cancel
using (Cell.LinePx(22))
{
 Cell.EmptyRow();

 using (Cell.RowPx(70))
 {
  Cell.current.disabled = error!=null; //could not be pressed while expression is not valid

  if (Draw.Button("OK"))
  {
   if (entry.expression.Length == 0)
    graph.exposed.Remove(entry.id, entry.name, entry.channel, entry.arrIndex);

   else
   {
    //removing all channels if exposing vector
    if (entry.channel < 0)
    {
     graph.exposed.Remove(entry.id, entry.name, 0, entry.arrIndex);
     graph.exposed.Remove(entry.id, entry.name, 1, entry.arrIndex);
     graph.exposed.Remove(entry.id, entry.name, 2, entry.arrIndex);
     graph.exposed.Remove(entry.id, entry.name, 3, entry.arrIndex);
    }

    //removing base vector if exposing channel
```

```
      else
       graph.exposed.Remove(entry.id, entry.name, -1, entry.arrIndex);

      //exposing
      graph.exposed.Add(entry, overwrite:true);
     }

     Close();
     EditorWindow.focusedWindow?.Repaint();
    }
   }

   Cell.EmptyRowPx(10);

   using (Cell.RowPx(70))
    if (Draw.Button("Cancel"))
     Close();
  }
 }
}


private void DrawExpression ()
/// Draws expression line with check string
{
```

```
}




public void ParseCheckExpression ()
{


}

public static void ShowWindow (Graph graph, ulong genId, string fieldName, Type fieldType, int channel,
{
 Vector2 mousePos = Event.current.mousePosition + UI.current.editorWindow.position.position; //before c

 ExposeWindow window = GetWindow<ExposeWindow>();
 if (window != null) window.Close(); //otherwise it will turn utility window in a tab
 window = ScriptableObject.CreateInstance(typeof(ExposeWindow)) as ExposeWindow;

 window.graph = graph;
 window.entry = graph.exposed[genId, fieldName, channel, arrIndex];
 if (window.entry == null  ||  window.entry.type != fieldType)
  window.entry = new Exposed.Entry(
   id: genId,
   name: fieldName,
   type: fieldType,
   channel: channel,
   arrIndex: arrIndex,
```

```
      expression: "" );


    window.titleContent = new GUIContent("Expose " + fieldName.Nicify());
    window.ShowUtility();


    Vector2 windowSize = new Vector2(310, 220);
    window.position = new Rect(
     mousePos - windowSize/2,
     windowSize);
   }
  }

}//namespace
```

```
ï»¿using System;
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;

using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Expose;
using MapMagic.Products;
using MapMagic.Previews;
using MapMagic.Nodes.GUI;

namespace MapMagic.Expose.GUI
{
 public static class OverrideInspector
 {
  public static void DrawLayeredOverride (Graph graph)
  /// Drawing override in layers-style (can add, remove or switch)
  /// For graph defaults
  {
```

```
 Override ovd = graph.defaults;


 using (Cell.LinePx(0))
  LayersEditor.DrawLayers(
   ovd.Count,
   onDraw:n => DrawLayer(ovd, n),
   onAdd:n => NewOverrideWindow.ShowWindow(ovd,n),
   onRemove:n => ovd.RemoveAt(n),
   onMove:(n1,n2) => ovd.Switch(n1,n2) );


 using (Cell.LinePx(20))
 {
  using (Cell.Row)
   if (Draw.Button("Add All Exposed"))
    graph.defaults.AddAllExposed(graph.exposed, graph);


  using (Cell.Row)
   if (Draw.Button("Remove Unused"))
    graph.defaults.RemoveAllUnused(graph.exposed);
 }
}



public static void DrawLayer (Override ovd, int num)
{
 (string name, Type type, object obj) = ovd.GetOverrideAt(num);
```

```
 Cell.EmptyLinePx(2);
 using (Cell.LineStd)
 {
  Cell.EmptyRowPx(2);

  using (Cell.RowPx(20)) Draw.Icon(UI.current.textures.GetTexture("DPUI/Icons/Layer"));
  using (Cell.Row)
  {
   obj = Draw.UniversalField(obj, type, name);

   if (Cell.current.valChanged)
   {
    ovd.SetOverrideAt(num, name, type, obj);
    RefreshMapMagic(name);
   }
  }

  Cell.EmptyRowPx(2);
 }
 Cell.EmptyLinePx(2);
}



public static void DrawStaticOverride (Override ovd)
/// For inspector use only: will make it re-generate on change, so don't use in function
```

```
{
 for (int i=0; i<ovd.Count; i++)
 {
  (string name, Type type, object obj) = ovd.GetOverrideAt(i);

  using (Cell.LineStd)
  {
   obj = Draw.UniversalField(obj, type, name);

   if (Cell.current.valChanged)
   {
    ovd.SetOverrideAt(i, name, type, obj);
    RefreshMapMagic(name);
   }
  }
 }
}



public static void RefreshMapMagic (string reference)
/// Makes MM generate if override changed
/// Automaticaly finds current mm and it's graph
{
 IMapMagic mm = GraphWindow.current.mapMagic;
 if (mm == null)
  return;
```

```
  Graph mmGraph = mm.Graph;
  if (mmGraph == null)
   return;

  foreach (IUnit unit in mmGraph.exposed.UnitsByReference(mmGraph, reference))
   if (unit is Generator gen)
    gen.version++;

  GraphWindow.current?.RefreshMapMagic();
 }
}



public class NewOverrideWindow : EditorWindow
{
 public Override ovd;
 private UI ui = new UI();

 public int num;
 private string varName = "";
 private Type varType = typeof(float);

 private static Type[] types = new Type[] {
  typeof(float), typeof(int), typeof(bool), typeof(Vector2D), typeof(Vector3),
  typeof(Texture2D), typeof(TerrainLayer)  };
```

```
private static string[] typeNames = new string[] {  //to avoid gathering names for drop-down
 "Float", "Int", "Boolean", "Vector2D", "Vector3",
 "Texture", "Terrain Layer" };

//public override void OnGUI(Rect rect) => ui.Draw(DrawParams);
//public override Vector2 GetWindowSize() =>  new Vector2(200, 150);
public void OnGUI () => ui.Draw(DrawParams, inInspector:false);

private void DrawParams ()
{
 using (Cell.Padded(5,5,5,5))
 {
  using (Cell.LineStd) Draw.Field(ref varName, "Name");
  using (Cell.LineStd) Draw.PopupSelector(ref varType, types, typeNames, "Type");

  Cell.EmptyLine();

  using (Cell.LinePx(22))
  {
   Cell.EmptyRow();

   using (Cell.RowPx(70))
   {
    Cell.current.disabled = name.Length!=0; //could not be pressed while expression is not valid

    if (Draw.Button("OK"))
```

```
    {
     ovd.Add(varName, varType, varType.IsValueType ? Activator.CreateInstance(varType) : null);

     Close();
     GraphWindow.current?.RefreshMapMagic();
     Extensions.GetInspectorWindow()?.Repaint();
    }
   }

   Cell.EmptyRowPx(10);

   using (Cell.RowPx(70))
    if (Draw.Button("Cancel"))
     Close();
  }
 }
}

public static void ShowWindow (Override ovd, int num)
{
 Vector2 curWinPos = focusedWindow!=null ? focusedWindow.position.position : new Vector2(0,0);
 Vector2 mousePos = Event.current.mousePosition + curWinPos; //before opening window

 NewOverrideWindow window = ScriptableObject.CreateInstance(typeof(NewOverrideWindow)) as NewO

 window.ovd = ovd;
```

```
    window.num = num;
    window.titleContent = new GUIContent("New Variable");
    window.ShowUtility();

    Vector2 windowSize = new Vector2(200, 100);
    window.position = new Rect(
     new Vector2(Screen.currentResolution.width, Screen.currentResolution.height)/2 - windowSize/2,
     windowSize);
   }
  }
 }
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes.GUI;
using MapMagic.Nodes.Biomes;

namespace MapMagic.Nodes.GUI
{
 public static class BiomesEditors
 {
  [Draw.Editor(typeof(RefBiome))]
  public static void DrawRefBiome (RefBiome gen)
  {
   using (Cell.Padded(1,1,0,0))
   {
    using (Cell.LineStd)
    {
     Draw.ObjectField(ref gen.subGraph, "Graph");
```

```
   if (Cell.current.valChanged)
    GraphWindow.current?.RefreshMapMagic();
  }

  using (Cell.LineStd)
   if (Draw.Button("Open") && gen.subGraph!=null)
    UI.current.DrawAfter( ()=> GraphWindow.current.OpenBiome(gen.subGraph) );
 }
}


[Draw.Editor(typeof(BiomesSet200))]
public static void BiomesSetEditor (BiomesSet200 gen)
{
 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlinl
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
}


private static void DrawBiomeLayer (Generator tgen, int num)
{
 BiomesSet200 gen = (BiomesSet200)tgen;
 BiomeLayer layer = gen.layers[num];
 if (layer == null) return;
```

```
 Cell.EmptyLinePx(3);
 using (Cell.LinePx(18))
 {
  if (num!=0)
   using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, gen);
  else
   //disconnecting last layer inlet
   if (GraphWindow.current.graph.IsLinked(layer))
    GraphWindow.current.graph.UnlinkInlet(layer);

  Cell.EmptyRowPx(12);

  Texture2D biomeIcon = UI.current.textures.GetTexture("MapMagic/Icons/Biomes");
  using (Cell.RowPx(14)) Draw.Icon(biomeIcon);

  using (Cell.Row) GeneratorDraw.SubGraph(layer, ref layer.graph);

  Cell.EmptyRowPx(10);

  using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
 }
 Cell.EmptyLinePx(3);
}


[Draw.Editor(typeof(Whittaker200))]
```

```
public static void DrawWhittaker (Whittaker200 gen)
{
 //using (Cell.LineStd)
 // using (Cell.Padded(1,1,0,0))
 //  Draw.Field(ref gen.sharpness, "Sharpness");

 foreach (WhittakerLayer layer in gen.layers)
 {
  Cell.EmptyLinePx(2);
  using (Cell.LinePx(0))
  {
   //outlet
   using (Cell.Full)
   {
    using (Cell.LineStd)
    {
     Cell.EmptyRow();
     using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
    }
    Cell.EmptyLine();
   }

   //layer itself
   using (Cell.Full)
    using (Cell.Padded(3,3,0,0))
    {
```

```
    if (layer.guiExpanded) Draw.Element(UI.current.styles.foldoutBackground);


    using (Cell.LineStd)
    {
     Cell.EmptyRowPx(2);
     using (Cell.Row) Draw.FoldoutLeft(ref layer.guiExpanded, layer.name);
    }


    if (layer.guiExpanded)
    {
     using (Cell.LinePx(0))
     using (Cell.Padded(2,2,0,0))
     {
      using (Cell.LineStd)
       GeneratorDraw.SubGraph(layer, ref layer.graph, refreshOnGraphChange:false); //this has loop prote


      using (Cell.LineStd) Draw.Field(ref layer.opacity, "Influence");
      //using (Cell.LineStd) Draw.Field(ref layer.opacity, "Opacity");
     }
     Cell.EmptyLinePx(3);
    }
   }
  }
  Cell.EmptyLinePx(2);
 }
}
```

```
 }
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes.GUI;
using MapMagic.Nodes.Biomes;

namespace MapMagic.Nodes.GUI
{
 public static class FunctionsEditors
 {
  [Draw.Editor(typeof(Function210), cat="Header")]
  public static void DrawFunctionHeader (Function210 fn)
  {
   RefreshOnSubgraphChange(fn); //since it's first encounter fn gui appears checking for internal graph cha
   DrawInletsOutlets(fn);
  }

  [Draw.Editor(typeof(Loop210), cat="Header")]
```

```
public static void DrawLoopHeader (Loop210 fn)
{
 RefreshOnSubgraphChange(fn);
 DrawInletsOutlets(fn);
}



private static void DrawInletsOutlets (BaseFunctionGenerator fn)
{
 using (Cell.LinePx(0))
 {
  using (Cell.Row)
  {
   for (int i=0; i<fn.inlets.Length; i++)
   {
    using (Cell.LineStd)
    {
     using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(fn.inlets[i], fn);
     Cell.EmptyRowPx(8);
     using (Cell.Row) Draw.Label(fn.inlets[i].Name);
    }
   }
  }

  Cell.EmptyRowPx(10);
```

```
  using (Cell.Row)
  {
   for (int i=0; i<fn.outlets.Length; i++)
   {
    using (Cell.LineStd)
    {
     using (Cell.Row) Draw.Label(fn.outlets[i].Name);
     Cell.EmptyRowPx(8);
     using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(fn.outlets[i]);
    }
   }
  }
 }

 if (fn.guiSameInletsName != null)
  using (Cell.LinePx(54))
   Draw.Label($"Two or more inlets \nhave the same name \n{fn.guiSameInletsName}");

 if (fn.guiSameOutletsName != null)
  using (Cell.LinePx(54))
   Draw.Label($"Two or more outlets \nhave the same name \n{fn.guiSameOutletsName}");
}


[Draw.Editor(typeof(Function210))]
public static void DrawFunction (Function210 fn)
```

```
{
 using (Cell.Padded(1,1,0,0))
 {
  Cell.EmptyLinePx(2);
  using (Cell.LineStd)
  {
   Graph prevGraph = fn.subGraph;
   GeneratorDraw.SubGraph(fn, ref fn.subGraph, refreshOnGraphChange:false);

   if (prevGraph != fn.subGraph  &&  fn.subGraph != null)
   {
    fn.ovd = new Expose.Override(fn.subGraph.defaults);
    GraphWindow.current?.RefreshMapMagic();
   }
  }

  bool genExposed = GraphWindow.current.graph.exposed.Contains(fn.id);

  Cell.EmptyLinePx(5);
  if (fn.subGraph != null)
  {
   for (int i=0; i<fn.ovd.Count; i++)
   {
    (string name, Type type, object obj) = fn.ovd.GetOverrideAt(i);

    using (Cell.LineStd)
```

```
      {
       obj = Draw.UniversalField(obj, type, name);
       Cell.current.Expose(fn.id, name, type, genExposed:genExposed);

       if (Cell.current.valChanged)
       {
        fn.ovd.SetOverrideAt(i, name, type, obj);

//        foreach (IUnit unit in fn.subGraph.exposed.UnitsByReference(fn.subGraph, name))
//         if (unit is Generator gen)
//          mm.Clear(gen);
       }
      }
     }
    }

    Cell.EmptyLinePx(2);
   }
  }

  [Draw.Editor(typeof(Loop210))]
  public static void DrawLoop (Loop210 loop)
  {
   using (Cell.Padded(1,1,0,0))
   {
```

```
Cell.EmptyLinePx(2);
using (Cell.LineStd)
{
 Graph prevGraph = loop.subGraph;
 GeneratorDraw.SubGraph(loop, ref loop.subGraph, refreshOnGraphChange:false);

 if (prevGraph != loop.subGraph  &&  loop.subGraph != null)
 {
  loop.ovd = new Expose.Override(loop.subGraph.defaults);
  GraphWindow.current?.RefreshMapMagic();
 }
}

bool genExposed = GraphWindow.current.graph.exposed.Contains(loop.id);

Cell.EmptyLinePx(5);
using (Cell.LineStd)
{
 Draw.Field(ref loop.iterations, "Iterations");
 Cell.current.Expose(loop.id, "iterations", typeof(int), genExposed:genExposed);
}

Cell.EmptyLinePx(4);
if (loop.subGraph != null)
 for (int i=0; i<loop.ovd.Count; i++)
 {
```

```
    (string name, Type type, object obj) = loop.ovd.GetOverrideAt(i);


    using (Cell.LineStd)
    {
     obj = Draw.UniversalField(obj, type, name);
     Cell.current.Expose(loop.id, name, type, genExposed:genExposed);


     if (Cell.current.valChanged)
      loop.ovd.SetOverrideAt(i, name, type, obj);
    }
   }


  Cell.EmptyLinePx(2);
 }
}



private static void RefreshOnSubgraphChange (BaseFunctionGenerator fn)
/// Checks if fn subgraph changed and performs complex operations if it's so
{
 ulong subGraphVersion = fn.subGraph ? fn.subGraph.IdsVersions() : 0;
 if (fn.guiPrevGraphVersion == subGraphVersion)
  return;


 //removing all if there is no graph
 if (fn.subGraph == null)
```

```
{
 if (fn.inlets.Length != 0  ||  fn.outlets.Length != 0)
 {
  fn.inlets = new FnInlet<object>[0];
  fn.outlets = new FnOutlet<object>[0];
  GraphWindow.current.graph.UnlinkGenerator(fn);
 }

 if (fn.ovd.Count != 0)
  fn.ovd.Clear();
}

else
{
 //syncing inlets/outlets
 SyncLayersPortals<IFnInlet<object>, IFnEnter<object>>(ref fn.inlets, fn, fn.subGraph, GraphWindow.cu
 SyncLayersPortals<IFnOutlet<object>, IFnExit<object>>(ref fn.outlets, fn, fn.subGraph, GraphWindow.c

 //refreshing override
 fn.ovd.Sync(fn.subGraph.defaults);

 //checking same inlet/outlet names
 fn.guiSameInletsName = CheckSameLayersNames(fn.inlets);
 fn.guiSameOutletsName = CheckSameLayersNames(fn.outlets);
}
```

```
 fn.guiPrevGraphVersion = subGraphVersion;
}



public static bool SyncLayersPortals<TL,TP> (ref TL[] layers, Generator gen, Graph subGraph, Graph par
 where TL:IFnLayer<object>
 where TP:IFnPortal<object>
 /// TL is layer type, TP is portal type
/// Synchronizes function inlets/outlets with function portals used in internal graph
/// ParentGraph to unlink link on remove, gen - to assign layer Gen and Id
{
 List<TL> newLayers = null;
 lock (layers) //SyncInlets may change inputs
 {
  //gathering all internal portals by name
  Dictionary<string,TP> namesPortals = new Dictionary<string,TP>();
  foreach (TP portal in subGraph.GeneratorsOfType<TP>())
  {
   HashSet<TP> portalSet = new HashSet<TP>();
   namesPortals.Add(portal.Name, portal);
  }



  //skipping if there is no change
  bool noChange = true;
```

```
if (namesPortals.Count != layers.Length)
 noChange = false;

foreach (IFnLayer<object> layer in layers)
{
 if (!namesPortals.ContainsKey(layer.Name))
  { noChange = false; break; }
}

if (noChange)
 return false;


//copying only layers with portals to newarray
newLayers = new List<TL>();

foreach (TL layer in layers)
{
 string name = layer.Name;
 if (!namesPortals.TryGetValue(name, out TP portal))
 {
  //unlinking removed inlet from graph
  if (layer is IInlet<object> inlet) parentGraph.UnlinkInlet(inlet);
  if (layer is IOutlet<object> outlet) parentGraph.UnlinkOutlet(outlet);

  continue;
```

```
  }

  newLayers.Add(layer);
  namesPortals.Remove(name);
 }


 //creating layers for portals left
 foreach (TP portal in namesPortals.Values)
 {
  //Type genericType = portal.GetType().BaseType.GetGenericArguments()[0]; //might have IFNPortal no
  Type genericType = null;
  foreach(Type iType in portal.GetType().GetInterfaces())
  {
   if (typeof(IFnPortal<object>).IsAssignableFrom(iType))
    genericType = iType.GetGenericArguments()[0];
  }

  TL layer = CreateLayer<TL>(gen, genericType);

  layer.Name = portal.Name;
  newLayers.Add(layer);
 }
}

layers = newLayers.ToArray();
```

```
 return true;
}


private static TL CreateLayer<TL> (Generator gen, Type genericType) where TL:IFnLayer<object>
{
 Type layerBaseType;
 if (typeof(TL).IsAssignableFrom(typeof(FnInlet<>))) layerBaseType = typeof(FnInlet<>);
 else layerBaseType = typeof(FnOutlet<>);

 Type layerType = layerBaseType.GetGenericTypeDefinition().MakeGenericType(genericType);

 object layerObj = Activator.CreateInstance(layerType);
 TL layer = (TL)layerObj;

 layer.Id = Id.Generate();
 layer.SetGen(gen);

 return layer;
}


private static string CheckSameLayersNames<TL> (TL[] layers) where TL: class, IFnLayer<object>
/// It two outlets have the same name returns this name
/// Returns null if no same name found
{
```

```
   foreach(TL outlet1 in layers)
    foreach(TL outlet2 in layers) //faster than creating hashset and no garbage
     if (outlet1 != outlet2  &&  outlet1.Name == outlet2.Name) return outlet1.Name;
   return null;
  }
 }
}
```

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Products;
using Den.Tools.GUI;

namespace MapMagic.Nodes.Biomes
{
 public class BiomeLayer : IBiome, IInlet<MatrixWorld>, IOutlet<MatrixWorld>
 {
  public float Opacity { get; set; }

  public Generator Gen { get { return gen; } private set { gen = value;} }
  public Generator gen; //property is not serialized
  public void SetGen (Generator gen) => this.gen=gen;

  public ulong id; //properties not serialized
  public ulong Id { get{return id;} set{id=value;} }
  public ulong LinkedOutletId { get; set; }  //if it's inlet. Assigned every before each clear or generate
  public ulong LinkedGenId { get; set; }
```

```
 public IUnit ShallowCopy() => (BiomeLayer)this.MemberwiseClone();

 public Expose.Override Override { get{return null;} set{} }

 public Graph graph;
 public Graph SubGraph => graph;
 public Graph AssignedGraph => graph;
 public TileData SubData (TileData parent) => parent.GetSubData(id);

 public BiomeLayer () => Opacity=1;
}


[Serializable]
[GeneratorMenu (menu="Biomes", name ="Biomes Set", iconName="GeneratorIcons/Biome", priority = 1,
public class BiomesSet200 : Generator, IMultiInlet, IMultiLayer, ICustomComplexity, ICustomClear, IPrepa
{
 public BiomeLayer[] layers = new BiomeLayer[0];
 public IList<IUnit> Layers { get => layers; set => layers=ArrayTools.Convert<BiomeLayer,IUnit>(value); }
 public void SetLayers(object[] ls) => layers = Array.ConvertAll(ls, i=>(BiomeLayer)i);
 public bool Inversed => true;
 public bool HideFirst => true;

 public IEnumerable<IInlet<object>> Inlets()
 {
  foreach (BiomeLayer layer in layers)
```

```
  yield return layer;
 //TODO: return layers
}

public IEnumerable<IBiome> Biomes()
{
 foreach (BiomeLayer layer in layers)
  yield return layer;
}

public IEnumerable<(IInlet<MatrixWorld>,Graph,TileData)> InletsSubgraphsDatas(TileData data)
/// Iterates in active generated layers (ones that have subgraph assigned and data generated) and returns
{
 foreach (BiomeLayer layer in layers)
 {
  Graph subGraph = layer.SubGraph;
  if (subGraph == null) continue;

  TileData subData = layer.SubData(data);
  if (subData == null) continue; //in case biome has not been generated ever yet, but dragging field

  yield return (layer, subGraph, subData);
 }

}
```

```
public float Complexity
{get{
 float sum = 0;
 foreach (BiomeLayer layer in layers)
  if (layer.graph != null)
   sum += layer.graph.GetGenerateComplexity();
 return sum;
}}

public float Progress (TileData data)
{
 float sum = 0;
 foreach (BiomeLayer layer in layers)
 {
  if (layer.graph == null) continue;

  TileData subData = layer.SubData(data);
  if (subData == null) continue;

  sum += layer.graph.GetGenerateProgress(subData);
 }
 return sum;
}

public void Prepare (TileData data, Terrain terrain)
```

```
{
 foreach (BiomeLayer layer in layers)
 {
  if (layer.graph == null) continue;

  TileData subData = data.CreateLoadSubData(layer.Id);

  layer.graph.Prepare(subData, terrain);
 }
}

public override void Generate (TileData data, StopToken stop)
{
 #if MM_DEBUG
 Log.Add("Biome start (draft:" + data.isDraft + " gen:" + id);
 #endif

 if (layers.Length == 0) return;
 BiomeLayer[] layersCopy = layers.Copy(); //layers count can be changed during generate

 //reading/copying products
 MatrixWorld[] dstMatrices = new MatrixWorld[layersCopy.Length];
 float[] opacities = new float[layersCopy.Length];

 if (stop!=null && stop.stop) return;
```

```
for (int i=0; i<layersCopy.Length; i++)
{
 if (stop!=null && stop.stop) return;

 MatrixWorld srcMatrix = data.ReadInletProduct(layersCopy[i]);
 if (srcMatrix != null) dstMatrices[i] = new MatrixWorld(srcMatrix);
 else dstMatrices[i] = new MatrixWorld(data.area.full.rect, (Vector3)data.area.full.worldPos, (Vector3)data

 opacities[i] = layersCopy[i].Opacity;
}

//normalizing
if (stop!=null && stop.stop) return;
dstMatrices.FillNulls(() => new MatrixWorld(data.area.full.rect, (Vector3)data.area.full.worldPos, (Vector3
dstMatrices[0].Fill(1);
Matrix.BlendLayers(dstMatrices, opacities);

//saving products
if (stop!=null && stop.stop) return;
for (int i=0; i<layersCopy.Length; i++)
 data.StoreProduct(layersCopy[i], dstMatrices[i]);

//generating biomes
for (int i=0; i<layersCopy.Length; i++)
{
 if (stop!=null && stop.stop) return;
```

```
 BiomeLayer layer = layersCopy[i];

 MatrixWorld mask;
 if (data.biomeMask == null)
  mask = dstMatrices[i]; //no need to copy for first-level biome
 else
 {
  mask = new MatrixWorld(dstMatrices[i]);
  mask.Multiply(data.biomeMask);
 }

 Graph subGraph = layer.SubGraph;
 if (subGraph == null) continue;

 //TileData subData = data.GetSubData(layer.Id);
 //if (subData == null) subData = data.CreateSubData(layer.Id, mask);
 //subData.mask = mask;
 //SubData could be created at prepare stage (Whittaker has prepare), but I have not tested re-assigning
 TileData subData = data.CreateLoadSubData(layer.Id, mask);

 layer.graph.Generate(subData, stop:stop, ovd:layer.graph.defaults);
}

#if MM_DEBUG
Log.Add("Biome generated (draft:" + data.isDraft + " gen:" + id);
```

```
 #endif
}



public void OnClearing (Graph graph, TileData data, ref bool isReady, bool totalRebuild=false)
/// Will be called at least once when ClearChanged graph (no matter ready or not)
/// Returns true if changed
/// Inlets are already cleared to this moment, but not this node itself
/// Iterating in sub-graph made with this
{
 // What should be cleared and when:
 // - On this graph modification (inlet change):   this node (done by default), all subgraph outputs (for biom
 // - Subgraph modification (any subgraph node change): this node, all subgraph relevants
 // - Exposed values change (this node change):   this node (done by default), exp related subgraph node,

 //clarifying whether this generator changed directly or recursively
 bool versionChanged = data.VersionChanged(this);
 bool thisChanged = !isReady && versionChanged;
 bool inletChanged = !isReady && !versionChanged;

 //iterating subgraphs/subdatas
 foreach (BiomeLayer layer in this.layers)
 {
  Graph subGraph = layer.SubGraph;
  if (subGraph == null) continue;
```

```
TileData subData = layer.SubData(data);
if (subData == null) continue; //in case biome has not been generated ever yet, but dragging field

//resetting exposed related nodes on this node change
if (thisChanged)
{
 //TODO: reset only changed generators
 foreach (IUnit expUnit in subGraph.exposed.AllUnits(subGraph))
  subData.ClearReady((Generator)expUnit);
}

//resetting outputs/relevants on inlet or this changed
if (inletChanged || thisChanged)
{
 foreach (Generator relGen in subGraph.RelevantGenerators(data.isDraft))
  subData.ClearReady(relGen);
}

//iterating in sub-graph after
subGraph.ClearChanged(subData);

//at the end clearing this if any subgraph relevant changed
if (isReady)
{
 foreach (Generator relGen in subGraph.RelevantGenerators(data.isDraft))
  if (!subData.IsReady(relGen))
```

```
        isReady = false;
      }
    }
  }
 }
}
```

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using MapMagic.Products;
using MapMagic.Expose;



namespace MapMagic.Nodes.Biomes
{
 public abstract class BaseFunctionGenerator : Generator
 /// Separate class mainly for gui purpose, to not distinguish between fn, loop and cluster
 {
  public IFnInlet<object>[] inlets = new IFnInlet<object>[0];
  public IEnumerable<IInlet<object>> Inlets() => inlets;

  public IFnOutlet<object>[] outlets = new IFnOutlet<object>[0];
  public IEnumerable<IOutlet<object>> Outlets() => outlets;

  public Graph subGraph;
  public Graph SubGraph => subGraph;
  public TileData SubData (TileData parent) => parent.GetSubData(id);
```

```
 public Override ovd = new Override();
 public Override Override { get => ovd; set => ovd=value; }

 [NonSerialized] public ulong guiPrevGraphVersion = 0;
 [NonSerialized] public string guiSameInletsName = null;
 [NonSerialized] public string guiSameOutletsName = null;
}



[Serializable]
[GeneratorMenu (menu="Functions", name ="Function", iconName="GeneratorIcons/Function", priority = 1
public class Function210 : BaseFunctionGenerator, IMultiInlet, IMultiOutlet, IPrepare, IBiome, ICustomCon
//relevant since it could be used as biome
{
 public float Complexity => subGraph!=null ? subGraph.GetGenerateComplexity() : 0;
 public float Progress (TileData data)
 {
  if (subGraph == null) return 0;

  TileData subData = SubData(data);
  if (subData == null) return 0;

  return subGraph.GetGenerateProgress(subData);
 }
```

```
public void Prepare (TileData data, Terrain terrain)
{
 if (subGraph == null) return;

 TileData subData = data.CreateLoadSubData(id);

 subGraph.Prepare(subData, terrain);
}

public override void Generate (TileData data, StopToken stop)
{
 if (stop!=null && stop.stop) return;
 if (subGraph == null) return;

 TileData subData = data.CreateLoadSubData(id);

 //sending inlet products to sub-graph enters
 if (stop!=null && stop.stop) return;
 for (int i=0; i<inlets.Length; i++)
 {
  IFnInlet<object> inlet = inlets[i];
  object product = data.ReadInletProduct(inlet);

  IFnEnter<object> fnEnter = (IFnEnter<object>)inlet.GetInternalPortal(subGraph);
  subData.StoreProduct(fnEnter, product);
```

```
  subData.MarkReady(fnEnter.Id, fnEnter.Gen.version); //TODO:check
 }

 //generating
 if (stop!=null && stop.stop) return;
 subGraph.Generate(subData, stop:stop, ovd:ovd); //with fn override, not graph defaults

 //returning products back from sub-graph exists
 if (stop!=null && stop.stop) return;
 for (int o=0; o<outlets.Length; o++)
 {
  IFnOutlet<object> outlet = outlets[o];
  IFnExit<object> fnExit = (IFnExit<object>)outlet.GetInternalPortal(subGraph);
  object product = subData.ReadInletProduct(fnExit);

  data.StoreProduct(outlet, product);
 }
}

public void OnClearing (Graph graph, TileData data, ref bool isReady, bool totalRebuild=false)
{
 // What should be cleared and when:
 // - On this graph modification (inlet change):   this node (done by default), all subgraph outputs (for biom
 // - Subgraph modification (any subgraph node change): this node, all subgraph relevants
 // - Exposed values change (this node change):   this node (done by default), exp related subgraph node,
```

```
//clarifying whether this generator changed directly or recursively
bool versionChanged = data.VersionChanged(this);
bool thisChanged = !isReady && versionChanged;
bool inletChanged = !isReady && !versionChanged;

if (subGraph == null) return;
TileData subData = data.CreateLoadSubData(id);

//resetting exposed related nodes on this node change
if (thisChanged)
{
 //TODO: reset only changed generators
 foreach (IUnit expUnit in subGraph.exposed.AllUnits(subGraph))
  subData.ClearReady((Generator)expUnit);
}

//resetting outputs/relevants on inlet or this changed
if (inletChanged || thisChanged)
{
 foreach (Generator relGen in subGraph.RelevantGenerators(data.isDraft))
  subData.ClearReady(relGen);
}

//iterating in sub-graph after
subGraph.ClearChanged(subData);
```

```
  //at the end clearing this if any subgraph relevant changed
  if (isReady)
  {
   foreach (Generator relGen in subGraph.RelevantGenerators(data.isDraft))
    if (!subData.IsReady(relGen))
     isReady = false;
  }
 }
}

[GeneratorMenu (name ="Function Outdated", iconName="GeneratorIcons/Function", priority = 1, colorTy
public class Function200 : Generator
{
 public Graph srcGraph;

 public override void Generate (TileData data, StopToken stop) { }

 public Function210 Update ()
 {
  Function210 fn = Generator.Create(typeof(Function210)) as Function210;
  fn.subGraph = srcGraph;
  return fn;
 }
}
}
```

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using MapMagic.Products;
using MapMagic.Expose;



namespace MapMagic.Nodes.Biomes
{
 public interface IFnLayer<out T> : IUnit where T:class
 {
  string Name { get; set; }
  ulong PortalId { get; set; } //fn portal node id in internal graph. TODO: switch to it instead of names
  IFnPortal<T> GetInternalPortal (Graph graph);
 }

 public interface IFnInlet<out T> : IFnLayer<T>, IInlet<T> where T:class {}

 public interface IFnOutlet<out T> : IFnLayer<T>, IOutlet<T> where T:class {}



 [Serializable]
```

```
public class FnInlet<T> : IFnInlet<T> where T: class
{
 [SerializeField] private string name; //properties not serialized
 public string Name { get=>name; set=>name=value; }

 [SerializeField] private Generator gen;
 public Generator Gen { get=>gen; private set=>gen=value; }
 public void SetGen (Generator gen) => Gen=gen;

 public ulong id;
 public ulong Id { get{return id;} set{id=value;} }

 public ulong portalId;
 public ulong PortalId { get{return portalId;} set{portalId=value;} }
 public ulong LinkedOutletId { get; set; }  //if it's inlet. Assigned every before each clear or generate
 public ulong LinkedGenId { get; set; }

 public IUnit ShallowCopy() => (Generator)this.MemberwiseClone();

 public FnInlet () { }


 public IFnPortal<T> GetInternalPortal (Graph graph)
 {
  foreach (IFnEnter<T> fnInput in graph.GeneratorsOfType<IFnEnter<T>>())
   if (fnInput.Name == Name)
```

```
    return fnInput;
  return null;
 }
}


[Serializable]
public class FnOutlet<T> : IFnOutlet<T> where T:class
{
 [SerializeField] private string name; //properties not serialized
 public string Name { get=>name; set=>name=value; }

 [SerializeField] private Generator gen;
 public Generator Gen { get=>gen; private set=>gen=value; }
 public void SetGen (Generator gen) => Gen=gen;

 public ulong id;
 public ulong Id { get{return id;} set{id=value;} }

 public ulong portalId;
 public ulong PortalId { get{return portalId;} set{portalId=value;} }

 public IUnit ShallowCopy() => (FnOutlet<T>)this.MemberwiseClone();

 public FnOutlet () { }
```

```
  public IFnPortal<T> GetInternalPortal (Graph graph)
  {
   foreach (IFnExit<T> fnInput in graph.GeneratorsOfType<IFnExit<T>>())
    if (fnInput.Name == Name)
     return fnInput;
   return null;
  }
 }

}
```

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Products;

namespace MapMagic.Nodes
{
 [GeneratorMenu (menu = "Functions/Enter", menuName = "Map", name = "Enter", iconName = "Generato
 [GeneratorMenu (menu = "Functions/Exit", menuName = "Map", name = "Exit", iconName = "GeneratorIco
 [GeneratorMenu (menu = "Functions/Enter", menuName = "Objects", name = "Enter", iconName = "Gener
 [GeneratorMenu (menu = "Functions/Exit", menuName = "Objects", name = "Exit", iconName = "Generato
 //[GeneratorMenu (menu = "Functions/Enter", menuName = "Spline", name = "Enter", iconName = "Gener
 //[GeneratorMenu (menu = "Functions/Exit",  menuName = "Spline", name = "Exit", iconName = "Generato
 [GeneratorMenu (menu = "Functions/Enter", menuName = "Spline", name = "Enter", iconName = "Genera
 [GeneratorMenu (menu = "Functions/Exit",  menuName = "Spline", name = "Exit", iconName = "Generator
 //no way to initialize portals in modules since they require FnEnter and FnExit class


 //gui interfaces
 //public interface IFnPortal<out T>
 //{
```

```
// string Name { get; set; }
//}

//public interface IFnEnter<out T> : IFnPortal<T>, IOutlet<T>  where T: class { }  //to use objects of type IF
//public interface IFnExit<out T> : IFnPortal<T>, IInlet<T>, IRelevant where T: class { } //fnExit is always ge
//interfaces required in draw editor, so they are stored in portals.cs, not module


[Serializable]
public class FnEnter<T> : Generator, IFnEnter<T>, IOutlet<T>, IRelevant where T: class
{
 [Val("Name")] public string name = "Input";
 public string Name { get{return name;} set{name=value;} }

 public override void Generate (TileData data, StopToken stop) {}
}


[Serializable]
public class FnExit<T> : Generator, IInlet<T>, IOutlet<T>, IFnExit<T>, IRelevant where T: class
{
 [Val("Name")] public string name = "Output";
 public string Name { get{return name;} set{name=value;} }

 public override void Generate (TileData data, StopToken stop)
 {
```

```
    if (stop!=null && stop.stop) return;


    //just passing link products to this
    object product = data.ReadInletProduct(this);
    data.StoreProduct(this, product);
   }
  }
 }
```

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Products;
using MapMagic.Expose;



namespace MapMagic.Nodes.Biomes
{
 [Serializable]
 [GeneratorMenu (menu="Functions", name ="Loop", iconName="GeneratorIcons/Function", priority = 1, co
 public class Loop210 : BaseFunctionGenerator, IMultiInlet, IMultiOutlet, IPrepare, IBiome, ICustomComple
 /// Function which executed several times
 /// Inlets and Outlets should have same name to transfer results for next iteration
 {
  public int iterations = 1;



  private TileData GetSubData (int i, TileData parentData)
  {
   if (parentData == null)
```

```
  return null;

 ulong subId = (ulong)(id*10000 + (ulong)i); //id of not existing node
 return SubData(parentData);
}


private IEnumerable<TileData> SubDatas (TileData parentData)
{
 if (parentData == null)
  yield break;

 for (int i=0; i<iterations; i++)
  yield return GetSubData(i, parentData);
}


public float Complexity => subGraph!=null ? subGraph.GetGenerateComplexity()*iterations : 0;
public float Progress (TileData data)
{
 if (subGraph == null)
  return 0;

 float totalProgress = 0;
 foreach (TileData subData in SubDatas(data))
  totalProgress += subData!=null ? subGraph.GetGenerateProgress(subData) : 0;
```

```
 return totalProgress;
}



public void Prepare (TileData parentData, Terrain terrain)
{
 if (subGraph == null)
  return;

 foreach (TileData subData in SubDatas(parentData))
  subGraph.Prepare(subData, terrain);
}



public override void Generate (TileData parentData, StopToken stop)
{
 if (stop!=null && stop.stop) return;
 if (subGraph == null) return;

 for (int it=0; it<iterations; it++)
 {
  TileData subData = GetSubData(it,parentData);

  //sending inlet products to sub-graph enters
  if (stop!=null && stop.stop) return;
```

```
for (int i=0; i<inlets.Length; i++)
{
 IFnInlet<object> inlet = inlets[i];
 object product = parentData.ReadInletProduct(inlet);

 IFnEnter<object> fnEnter = (IFnEnter<object>)inlet.GetInternalPortal(subGraph);
 subData.StoreProduct(fnEnter, product);
 subData.MarkReady(fnEnter.Id, fnEnter.Gen.version);
}

//overriding loop iteration
if (stop!=null && stop.stop) return;
ovd.SetOrAdd("LoopIteration", typeof(int), it);

//generating
if (stop!=null && stop.stop) return;
subGraph.Generate(subData, stop:stop, ovd:ovd); //with fn override, not graph defaults

//returning products back from sub-graph exists
if (stop!=null && stop.stop) return;
for (int o=0; o<outlets.Length; o++)
{
 IFnOutlet<object> outlet = outlets[o];
 IFnExit<object> fnExit = (IFnExit<object>)outlet.GetInternalPortal(subGraph);
 object product = subData.ReadInletProduct(fnExit);
```

```
      parentData.StoreProduct(outlet, product);
    }
   }
  }

 }
}
```

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Products;
using Den.Tools.GUI;

namespace MapMagic.Nodes
{
 [Serializable]
 //[GeneratorMenu (menu="Biomes", name ="Ref Biome", priority = 1)]
 public class RefBiome : Generator, IMultiInlet, IBiome, ICustomComplexity
 {
  //could be Inlet<mask> but do so since mask is not mandatory
  [Val("Mask", "Inlet")] public readonly Inlet<MatrixWorld> maskIn = new Inlet<MatrixWorld>();
  public IEnumerable<IInlet<object>> Inlets() { yield return maskIn; }

  public Graph subGraph;
  public Graph SubGraph => subGraph;
  public Graph AssignedGraph => subGraph;
  public TileData SubData (TileData parent) => parent.GetSubData(id);
```

```
  public Expose.Override Override { get{return null;} set{} }


  public float Complexity => subGraph!=null ? subGraph.GetGenerateComplexity() : 0;
  public float Progress (TileData data)
  {
/*   TileData subData = data.subDatas[this];
   if (subGraph == null  ||  subData == null) return 0;
   return subGraph.GetGenerateProgress(subData);*/
return 0;
  }


  private TileData GetSubData (TileData parentData)
  {
/*   TileData usedData = parentData.subDatas[this];
   if (usedData == null)
   {
    usedData = new TileData(parentData);
    parentData.subDatas[this] = usedData;
   }
   return usedData;*/
return null;
  }


  public override void Generate (TileData data, StopToken stop)
  {
/*   if (stop!=null && stop.stop) return;
```

```
    if (subGraph == null) return;


    MatrixWorld mask = data.ReadInletProduct(maskIn);

    GetSubData(data).SetBiomeMask(mask, data.currentBiomeMask);  */

   }


  }
}
```

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Products;
using Den.Tools.GUI;

namespace MapMagic.Nodes.Biomes
{
 public class WhittakerLayer : IBiome, IOutlet<MatrixWorld>
 {
  public string name;
  public float opacity;
  public float influence;
  public string diagramName;

  public Generator Gen { get { return gen; } private set { gen = value;} }
  public Generator gen; //property is not serialized
  public void SetGen (Generator gen) => this.gen=gen;

  public ulong id; //properties not serialized
  public ulong Id { get{return id;} set{id=value;} }
```

```
  public ulong LinkedOutletId { get; set; }  //if it's inlet. Assigned every before each clear or generate
  public ulong LinkedGenId { get; set; }

  public IUnit ShallowCopy() => (WhittakerLayer)this.MemberwiseClone();

  public Expose.Override Override { get{return null;} set{} }

  public Graph graph;
  public Graph SubGraph => graph;
  public Graph AssignedGraph => graph;
  public TileData SubData (TileData parent) => parent.GetSubData(id);

  public WhittakerLayer () { opacity=1; }
  public WhittakerLayer (string name) { opacity=1; this.name=name; }
  public WhittakerLayer (string name, string diagramName) { opacity=1; this.name=name; this.diagramNam

/*  public TileData GetSubData (TileData parentData)
  {
   TileData usedData = parentData.subDatas[this];
   if (usedData == null)
   {
    usedData = new TileData(parentData);
    parentData.subDatas[this] = usedData;
   }
   return usedData;
  }*/
```

```
 public bool guiExpanded;
}

[Serializable]
[GeneratorMenu (menu="Biomes", name ="Whittaker", iconName="GeneratorIcons/Whittaker", priority = 1
public class Whittaker200 : Generator, IPrepare, IMultiInlet, ICustomComplexity, IMultiLayer, ICustomClea
{
 public static MatrixAsset[] diagramAssets;

 [Val("Heat", "Inlet")] public readonly Inlet<MatrixWorld> temperatureIn = new Inlet<MatrixWorld>();
 [Val("Moisture ", "Inlet")] public readonly Inlet<MatrixWorld> moistureIn = new Inlet<MatrixWorld>();
 public IEnumerable<IInlet<object>> Inlets () { yield return temperatureIn; yield return moistureIn; }

 [Val("Sharpness")] public float sharpness = 0.6f;

 //[Val(name="Temperature Limit")] public float brightness = 0f;
 //[Val(name="Contrast")] public float contrast = 1f;

 public WhittakerLayer[] layers = new WhittakerLayer[]
 {
  new WhittakerLayer("Tropic Rainforest", "TropicalRainforest"),
  new WhittakerLayer("Mild Rainforest", "TemperateRainforest"),

  new WhittakerLayer("Tropic Forest", "TropicalForest"),
  new WhittakerLayer("Mild Forest", "TemperateForest"),
```

```
  new WhittakerLayer("Taiga", "Taiga"),

  new WhittakerLayer("Savanna", "Savanna"),
  new WhittakerLayer("Grassland", "Grassland"),
  new WhittakerLayer("Tundra", "Tundra"),

  new WhittakerLayer("Hot Desert", "Desert"),
  new WhittakerLayer("Cold Desert", "ColdDesert")
 };

 public IList<IUnit> Layers { get => layers; set {return;} } //don't set anything
 public void SetLayers(object[] ls) {return;}
 public bool Inversed => false;
 public bool HideFirst => false;


// public ICollection<IUnit> Layerss { get => Layers; }
// public void SetLayers(object[] ls) => layers = Array.ConvertAll(ls, i=>(BiomeLayer)i);
//IMultiLayer

 public IEnumerable<IOutlet<object>> Outlets ()
 {
  foreach (WhittakerLayer layer in layers)
   yield return layer;
 }
```

```
public IEnumerable<IBiome> Biomes()
{
 foreach (WhittakerLayer layer in layers)
  yield return layer;
}

public float Complexity
{get{
 float sum = 0;
 foreach (WhittakerLayer layer in layers)
  if (layer.graph != null)
   sum += layer.graph.GetGenerateComplexity();
 return sum;
}}

public float Progress (TileData data)
{
 float sum = 0;
 foreach (WhittakerLayer layer in layers)
 {
  if (layer.graph == null) continue;

  TileData subData = layer.SubData(data);
  if (subData == null) continue;

  sum += layer.graph.GetGenerateProgress(subData);
```

```
 }
 return sum;
}



public void Prepare (TileData data, Terrain terrain)
{
 foreach (WhittakerLayer layer in layers)
 {
  if (layer.graph == null) continue;

  TileData subData = layer.SubData(data);
  if (subData == null) continue;

  layer.graph.Prepare(subData, terrain);
 }

 if (diagramAssets == null)
 {
  diagramAssets = new MatrixAsset[10];
  int i=0;
  foreach (WhittakerLayer layer in layers)
  {
   diagramAssets[i] =  Resources.Load<MatrixAsset>("MapMagic/Whittaker/" + layer.diagramName);
   i++;
  }
```

```
 }
}



public override void Generate (TileData data, StopToken stop)
{
 //reading inputs
 if (stop!=null && stop.stop) return;
 MatrixWorld temperatureMatrix = data.ReadInletProduct(temperatureIn);
 MatrixWorld moistureMatrix = data.ReadInletProduct(moistureIn);
 if (temperatureMatrix == null || moistureMatrix == null || !enabled) return;

 if (diagramAssets == null)
  throw new Exception("Could not find Whittaker diagrams. Possibly generator is not initialized");
 Coord diagramSize = diagramAssets[0].matrix.rect.size;

 //creating biome masks
 if (stop!=null && stop.stop) return;
 MatrixWorld[] masks = new MatrixWorld[10];
 for (int i=0; i<masks.Length; i++)
  masks[i] = new MatrixWorld(temperatureMatrix.rect, temperatureMatrix.worldPos, temperatureMatrix.wo

 if (stop!=null && stop.stop) return;
 Coord min = temperatureMatrix.rect.Min; Coord max = temperatureMatrix.rect.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
```

```
 {
  int pos = temperatureMatrix.rect.GetPos(x,z);
  float temperature = temperatureMatrix.arr[pos] * diagramSize.x;
  float moisture = moistureMatrix.arr[pos] * diagramSize.z;

  if (temperature < 0) temperature = 0; if (temperature > diagramSize.x) temperature = diagramSize.x;
  if (moisture < 0) moisture = 0; if (moisture > diagramSize.z) moisture = diagramSize.z;

  for (int m=0; m<masks.Length; m++)
  {
   float val = diagramAssets[m].matrix.GetInterpolated(temperature, moisture);
   val -= sharpness/2;
   if (val < 0) val = 0;
   masks[m].arr[pos] = val;
  }
 }

//and opacities
if (stop!=null && stop.stop) return;
float[] opacities = new float[masks.Length];
int t=0;
foreach (WhittakerLayer layer in layers)
 { opacities[t] = layer.opacity; t++; }

if (stop!=null && stop.stop) return;
Matrix.NormalizeLayers(masks, opacities, allowBelowOne:false);
```

```
//saving products
if (stop!=null && stop.stop) return;
t=0;
foreach (WhittakerLayer layer in layers)
{
 data.StoreProduct(layer.Id, masks[t]);
 t++;
}

//generating biomes
for (int i=0; i<layers.Length; i++)
{
 if (stop!=null && stop.stop) return;

 WhittakerLayer layer = layers[i];

 MatrixWorld mask;
 if (data.biomeMask == null)
  mask = masks[i]; //no need to copy for first-level biome
 else
 {
  mask = new MatrixWorld(masks[i]);
  mask.Multiply(data.biomeMask);
 }
```

```
  Graph subGraph = layer.SubGraph;
  if (subGraph == null) continue;
  TileData subData = data.CreateLoadSubData(layer.Id, mask);

  if (mask.MaxValue() > 0.0001f)
    layer.graph.Generate(subData, stop:stop, ovd:layer.graph.defaults);
 }
}

public void OnClearing (Graph graph, TileData data, ref bool isReady, bool totalRebuild=false)
{
 // What should be cleared and when:
 // - On this graph modification (inlet change):   this node (done by default), all subgraph outputs (for biom
 // - Subgraph modification (any subgraph node change): this node, all subgraph relevants
 // - Exposed values change (this node change):   this node (done by default), exp related subgraph node,

 //clarifying whether this generator changed directly or recursively
 bool versionChanged = data.VersionChanged(this);
 bool thisChanged = !isReady && versionChanged;
 bool inletChanged = !isReady && !versionChanged;

 //iterating subgraphs/subdatas
 foreach (WhittakerLayer layer in layers)
 {
  Graph subGraph = layer.SubGraph;
  if (subGraph == null) return;
```

```
TileData subData = layer.SubData(data);
if (subData == null) return; //in case biome has not been generated ever yet, but dragging field

//resetting exposed related nodes on this node change
if (thisChanged)
{
 //TODO: reset only changed generators
 foreach (IUnit expUnit in subGraph.exposed.AllUnits(subGraph))
  subData.ClearReady((Generator)expUnit);
}

//resetting outputs/relevants on inlet or this changed
if (inletChanged || thisChanged)
{
 foreach (Generator relGen in subGraph.RelevantGenerators(data.isDraft))
  subData.ClearReady(relGen);
}

//iterating in sub-graph after
subGraph.ClearChanged(subData);

//at the end clearing this if any subgraph relevant changed
if (isReady)
{
 foreach (Generator relGen in subGraph.RelevantGenerators(data.isDraft))
```

```
      if (!subData.IsReady(relGen))
       isReady = false;
    }
   }
  }
 }
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes.GUI;
using MapMagic.Nodes.MatrixGenerators;

namespace MapMagic.Nodes.GUI
{
 public static class GeneratorEditors
 {
  [Draw.Editor(typeof(MatrixGenerators.Blur200))]
  public static void DrawSelector (MatrixGenerators.Blur200 gen)
  {
   //drawing standard class values

   using (Cell.Padded(1,1,0,0))
   {
    //radius warning
```

```
  MapMagicObject mapMagic = GraphWindow.current.mapMagic as MapMagicObject;
  if (mapMagic != null)
  {
   float pixelSize = mapMagic.tileSize.x / (int)mapMagic.tileResolution;
   if (gen.blur * gen.downsample > mapMagic.tileMargins * pixelSize)
   {
    Cell.EmptyLinePx(2);
    using (Cell.LinePx(40))
     using (Cell.Padded(2,2,0,0))
     {
      Draw.Element(UI.current.styles.foldoutBackground);
      using (Cell.LinePx(15)) Draw.Label("Current setup can");
      using (Cell.LinePx(15)) Draw.Label("create tile seams");
      using (Cell.LinePx(15)) Draw.URL("More", url:"https://gitlab.com/denispahunov/mapmagic/-/wikis/Tile_
     }
    Cell.EmptyLinePx(2);
   }
  }
 }
}



[Draw.Editor(typeof(MatrixGenerators.Curve200))]
public static void DrawCurve (MatrixGenerators.Curve200 gen)
{
 using (Cell.LinePx(GeneratorDraw.nodeWidth)) //square cell
```

```
  //using (Timer.Start("DrawCurve"))
 {
  Draw.Rect(new Color(1,1,1,0.5f)); //background

  using (Cell.Padded(5))
   CurveDraw.DrawCurve(gen.curve, gen.histogram);
 }
}


[Draw.Editor(typeof(MatrixGenerators.Levels200))]
public static void DrawLevels (MatrixGenerators.Levels200 gen)
{
 using (Cell.LinePx( (GeneratorDraw.nodeWidth-10)/2 )) //square internal grid cell
  //using (Timer.Start("DrawLevels"))
 {
  Draw.Rect(new Color(1,1,1,0.5f));

  using (Cell.Padded(1,1,7,0))
   LevelsDraw.DrawLevels(ref gen.inMin, ref gen.inMax, ref gen.gamma, ref gen.outMin, ref gen.outMax, g
 }

 using (Cell.LineStd)
 {
  if (!gen.guiParams)
   Draw.Rect(new Color(1,1,1,0.5f));
```

```
using (new Draw.FoldoutGroup(ref gen.guiParams, "Parameters", isLeft:true))
 if (gen.guiParams)
 {
  using (Cell.LineStd) {
   Draw.Field(ref gen.inMin, "In Low");
   Cell.current.Expose(gen.id, "inMin", typeof(float));
   Draw.AddFieldToCellObj(typeof(MatrixGenerators.Levels200), "inMin"); }
  using (Cell.LineStd) {
   Draw.Field(ref gen.gamma, "Gamma");
   Cell.current.Expose(gen.id, "gamma", typeof(float));
   Draw.AddFieldToCellObj(typeof(MatrixGenerators.Levels200), "gamma"); }
  using (Cell.LineStd) {
   Draw.Field(ref gen.inMax, "In High");
   Cell.current.Expose(gen.id, "inMax", typeof(float));
   Draw.AddFieldToCellObj(typeof(MatrixGenerators.Levels200), "inMax"); }

  Cell.EmptyLinePx(5);

  using (Cell.LineStd) {
   Draw.Field(ref gen.outMin, "Out Low");
   Cell.current.Expose(gen.id, "outMin", typeof(float));
   Draw.AddFieldToCellObj(typeof(MatrixGenerators.Levels200), "outMin"); }
  using (Cell.LineStd) {
   Draw.Field(ref gen.outMax, "Out High");
   Cell.current.Expose(gen.id, "outMax", typeof(float));
```

```
    Draw.AddFieldToCellObj(typeof(MatrixGenerators.Levels200), "outMax"); }
   }
 }
}


[Draw.Editor(typeof(MatrixGenerators.Selector200))]
public static void DrawSelector (MatrixGenerators.Selector200 gen)
{
 using (Cell.Padded(1,1,0,0))
 {
  using (Cell.LineStd) Draw.Field(ref gen.rangeDet, "Set Range");
  using (Cell.LineStd)
  {
   Draw.Field(ref gen.units, "Units");
   Cell.current.Expose(gen.id, "units", typeof(int));
   Draw.AddFieldToCellObj(typeof(MatrixGenerators.Selector200), "units");
  }

  if (gen.rangeDet == MatrixGenerators.Selector200.RangeDet.MinMax)
  {
   using (Cell.LineStd)
   {
    Draw.Field(ref gen.from, "From");
    Cell.current.Expose(gen.id, "from", typeof(Vector2));
    Draw.AddFieldToCellObj(typeof(MatrixGenerators.Selector200), "from");
```

```
 }
 using (Cell.LineStd)
 {
  Draw.Field(ref gen.to, "To");
  Cell.current.Expose(gen.id, "to", typeof(Vector2));
  Draw.AddFieldToCellObj(typeof(MatrixGenerators.Selector200), "to");
 }
}
else
{
 float from = (gen.from.x + gen.from.y)/2;
 float to = (gen.to.x + gen.to.y)/2;
 float transition = (gen.from.y - gen.from.x);

 using (Cell.LineStd)
 {
  Draw.Field(ref from, "From");
  Draw.AddFieldToCellObj(typeof(MatrixGenerators.Selector200), "from"); //not a single value, but the or
 }
 using (Cell.LineStd)
 {
  Draw.Field(ref to, "To");
  Draw.AddFieldToCellObj(typeof(MatrixGenerators.Selector200), "to");
 }
 using (Cell.LineStd) Draw.Field(ref transition, "Transition");
```

```
    gen.from.x = from-transition/2;
    gen.from.y = from+transition/2;
    gen.to.x = to-transition/2;
    gen.to.y = to+transition/2;
   }
  }
}


[Draw.Editor(typeof(MatrixGenerators.HeightOutput200))]
public static void HeightOutputEditor (MatrixGenerators.HeightOutput200 heightOut)
{
 using (Cell.Padded(1,1,0,0))
 {
  using (Cell.LinePx(0))
  {
   Cell.current.fieldWidth = 0.4f;

   if (GraphWindow.current.mapMagic != null)
   {
    using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(ref GraphWindow.current.mapMagic.Globals.heigh
    using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(ref GraphWindow.current.mapMagic.Globals.heigh
   }
   else
    using (Cell.LinePx(18+18)) Draw.Label("Not assigned to current \nMapMagic object");
```

```
   using (Cell.LineStd) Draw.Field(ref heightOut.outputLevel, "Out Level");
  }
 }
}

[Draw.Editor(typeof(MatrixGenerators.HolesOutput2112))]
public static void HeightOutputEditor (MatrixGenerators.HolesOutput2112 holesOut)
{
 using (Cell.Padded(1,1,0,0))
 {
  using (Cell.LinePx(0))
  {
   Cell.current.fieldWidth = 0.4f;

   if (GraphWindow.current.mapMagic != null)
   {
    //using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(ref GraphWindow.current.mapMagic.Globals.hole
    //there's no way to change holes resolution with script yey. It's always height-1.
   }
   else
    using (Cell.LinePx(18+18)) Draw.Label("Not assigned to current \nMapMagic object");
  }
 }
}
```

```
[Draw.Editor(typeof(MatrixGenerators.UnityCurve200))]
public static void CurveGeneratorEditor (MatrixGenerators.UnityCurve200 gen)
{
 using (Cell.LinePx(GeneratorDraw.nodeWidth+4)) //don't really know why 4
  using (Cell.Padded(5))
  {
   Draw.AnimationCurve(gen.curve);
   Draw.AddFieldToCellObj(typeof(MatrixGenerators.UnityCurve200), "curve");
  }
}

[Draw.Editor(typeof(Blend200))]
public static void BlendGeneratorEditor (Blend200 gen)
{
 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true);
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
}

private static void DrawBlendLayer (Generator tgen, int num)
{
 Blend200 gen = (Blend200)tgen;
 Blend200.Layer layer = gen.layers[num];

 Cell.EmptyLinePx(2);

 using (Cell.LineStd)
```

```
{
 using (Cell.RowPx(0))
  GeneratorDraw.DrawInlet(layer.inlet, gen);
 Cell.EmptyRowPx(10);

 using (Cell.RowPx(20)) Draw.Icon(UI.current.textures.GetTexture("DPUI/Icons/Layer"));

 if (num == 0)
 {
  layer.algorithm = MatrixGenerators.Blend200.BlendAlgorithm.add;
  using (Cell.Row) Draw.Label("Background");
 }

 else
 {
  if (!layer.guiExpanded)
  {
   using (Cell.Row) {
    layer.algorithm = Draw.Field(layer.algorithm); }
    //Draw.AddFieldToCellObj(typeof(MatrixGenerators.Blend200.Layer), "algorithm"); }
    //could not be exposed since it's layer value, not generator one
   using (Cell.RowPx(20)) layer.guiExpanded = Draw.LayerChevron(layer.guiExpanded);
  }

  else
  {
```

```
  using (Cell.Row)
  {
   using (Cell.LineStd) {
    layer.algorithm = Draw.Field(layer.algorithm); }
    //Draw.AddFieldToCellObj(typeof(MatrixGenerators.Blend200.Layer), "algorithm"); }
   using (Cell.LineStd)
   {
    Draw.FieldDragIcon(ref layer.opacity, UI.current.textures.GetTexture("DPUI/Icons/Opacity"));
    Cell.current.Expose(gen.id, "layers", typeof(float), arrIndex:num);
    //Draw.AddFieldToCellObj(typeof(MatrixGenerators.Blend200.Layer), "opacity"); }
   }
  }

  using (Cell.RowPx(20))
   using (Cell.LineStd) layer.guiExpanded = Draw.LayerChevron(layer.guiExpanded);
 }

 /*using (Cell.RowPx(35))
 {
  //Draw.Field(ref layer.opacity);
  Draw.FieldDragIcon(ref layer.opacity, UI.current.textures.GetTexture("DPUI/Icons/Opacity"));
 }*/
}

Cell.EmptyRowPx(3);
}
```

```
 Cell.EmptyLinePx(2);
}

[Draw.Editor(typeof(Normalize200))]
public static void NormalizeGeneratorEditor (Normalize200 gen)
{
 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlinl
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
}

private static void DrawNormalizeLayer (Generator tgen, int num)
{
 Normalize200 gen = (Normalize200)tgen;
 Normalize200.NormalizeLayer layer = gen.layers[num];

 if (layer == null) return;

 using (Cell.LinePx(20))
 {
  if (num!=0)
   using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, gen);
  else
   //disconnecting last layer inlet
   if (GraphWindow.current.graph.IsLinked(layer))
```

```
    GraphWindow.current.graph.UnlinkInlet(layer);


  Cell.EmptyRowPx(10);


  using (Cell.RowPx(73))
  {
   if (num==0) Draw.Label("Background");
   else Draw.Label("Layer " + num);
  }


  using (Cell.RowPx(10)) Draw.Icon(UI.current.textures.GetTexture("DPUI/Icons/Opacity"));
  using (Cell.Row) layer.Opacity = Draw.Field(layer.Opacity);


  Cell.EmptyRowPx(10);
  using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
 }
}



[Draw.Editor(typeof(MatrixGenerators.GrassOutput200))]
public static void DrawGrassOutput (MatrixGenerators.GrassOutput200 grassOut)
{
 using (Cell.Padded(1,1,0,0))
 {
  //Cell.current.margins = new Padding(4,4,4,4);
```

```
using (Cell.LinePx(0))
{
 Cell.current.fieldWidth = 0.6f;

 using (Cell.LineStd) Draw.Field(ref grassOut.renderMode, "Mode");

 if (grassOut.renderMode == MatrixGenerators.GrassOutput200.GrassRenderMode.MeshUnlit || grassO
 {
  using (Cell.LineStd) grassOut.prototype.prototype = Draw.Field(grassOut.prototype.prototype, "Object"
  grassOut.prototype.prototypeTexture = null; //otherwise this texture will be included to build even if not
  grassOut.prototype.usePrototypeMesh = true;
 }
 else
 {
  using (Cell.LineStd) grassOut.prototype.prototypeTexture = Draw.Field(grassOut.prototype.prototypeTe
  grassOut.prototype.prototype = null; //otherwise this object will be included to build even if not displaye
  grassOut.prototype.usePrototypeMesh = false;
 }
 switch (grassOut.renderMode)
 {
  case MatrixGenerators.GrassOutput200.GrassRenderMode.Grass: grassOut.prototype.renderMode =
  case MatrixGenerators.GrassOutput200.GrassRenderMode.Billboard: grassOut.prototype.renderMode
  case MatrixGenerators.GrassOutput200.GrassRenderMode.MeshVertexLit: grassOut.prototype.render
  case MatrixGenerators.GrassOutput200.GrassRenderMode.MeshUnlit: grassOut.prototype.renderMod
 }
```

```
 #if UNITY_2021_2_OR_NEWER
 if (Cell.current.valChanged)
  grassOut.prototype.useInstancing = false;
 #endif
}

using (Cell.LinePx(0))
{
 Cell.current.fieldWidth = 0.4f;

 using (Cell.LineStd) Draw.Field(ref grassOut.density, "Density");
 using (Cell.LineStd) grassOut.prototype.dryColor = Draw.Field(grassOut.prototype.dryColor, "Dry");
 using (Cell.LineStd) grassOut.prototype.healthyColor = Draw.Field(grassOut.prototype.healthyColor, "H

 Vector2 temp = new Vector2(grassOut.prototype.minWidth, grassOut.prototype.maxWidth);
 using (Cell.LineStd) Draw.Field(ref temp, "Width", xName:"Min", yName:"Max", xyWidth:25);
 grassOut.prototype.minWidth = temp.x; grassOut.prototype.maxWidth = temp.y;

 temp = new UnityEngine.Vector2(grassOut.prototype.minHeight, grassOut.prototype.maxHeight);
 using (Cell.LineStd) Draw.Field(ref temp, "Height", xName:"Min", yName:"Max", xyWidth:25);
 grassOut.prototype.minHeight = temp.x; grassOut.prototype.maxHeight = temp.y;

 Cell.EmptyLinePx(2);
 using (Cell.LinePx(0))
  using (new Draw.FoldoutGroup(ref grassOut.guiAdvanced, "Advanced", padding:0))
   if (grassOut.guiAdvanced)
```

```
{
 using (Cell.LineStd) grassOut.prototype.noiseSpread = (float)Draw.Field(grassOut.prototype.noiseSpr

 #if UNITY_2021_2_OR_NEWER
 using (Cell.LineStd)
 {
  bool instancing = Draw.ToggleLeft(grassOut.prototype.useInstancing, "Use Instancing");

  if (Cell.current.valChanged && instancing && grassOut.prototype.renderMode != DetailRenderMode.V
  {
   string text = "Unity DetailPrototype.useInstancing documentation states that this setting is only effect
   instancing = UnityEditor.EditorUtility.DisplayDialog("Enable Instancing", text, "Yes, enable instancing
  }

  grassOut.prototype.useInstancing = instancing;

 }
 #endif

 #if UNITY_2022_2_OR_NEWER
 using (Cell.LineStd)
  grassOut.prototype.useDensityScaling = Draw.ToggleLeft(grassOut.prototype.useDensityScaling, "U
 #endif

 if (GraphWindow.current.mapMagic != null  &&  GraphWindow.current.mapMagic is MapMagicObject
 {
```

```
        using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(ref mapMagicObject.globals.grassResDownscal
        using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(ref mapMagicObject.globals.grassResPerPatch,

        #if UNITY_2022_2_OR_NEWER
        using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(ref mapMagicObject.globals.grassScatterMode,
        #endif
       }
      }
   }
  }
}



[Draw.Editor(typeof(TexturesOutput200))]
public static void TexturesGeneratorEditor (TexturesOutput200 gen)
{
 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlinl
 using (Cell.LinePx(0))  GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
}

private static void DrawTexturesLayer (Generator tgen, int num)
{
 TexturesOutput200 texOut = (TexturesOutput200)tgen;
 TexturesOutput200.TextureLayer layer = texOut.layers[num];
 if (layer == null) return;
```

```
Cell.EmptyLinePx(3);
using (Cell.LinePx(28))
{
 if (num!=0)
  using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, texOut);
 else
  //disconnecting last layer inlet
  if (GraphWindow.current.graph.IsLinked(layer))
   GraphWindow.current.graph.UnlinkInlet(layer);

 Cell.EmptyRowPx(10);

 Texture2D tex = layer.prototype!=null ? layer.prototype.diffuseTexture : UI.current.textures.GetTexture("
 using (Cell.RowPx(28)) Draw.TextureIcon(tex);

 using (Cell.Row)
 {
  Cell.current.trackChange = false;
  Draw.EditableLabel(ref layer.name);
 }

 using (Cell.RowPx(20))
 {
  Cell.current.trackChange = false;
  Draw.LayerChevron(num, ref texOut.guiExpanded);
 }
```

```
 Cell.EmptyRowPx(10);
 using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
}
Cell.EmptyLinePx(2);

if (texOut.guiExpanded == num)
using (Cell.Line)
{
 Cell.EmptyRowPx(2);

 using (Cell.Row)
 {
  using (Cell.LinePx(0))
   using (Cell.Padded(1,0,0,0))
  {
   //using (Cell.LineStd) layer.Opacity = Draw.Field(layer.Opacity, "Opacity");

   using (Cell.LineStd)
   {
    Draw.ObjectField(ref layer.prototype, "Layer");
    Cell.current.Expose(layer.id, "prototype", typeof(TerrainLayer));
   }

   if (layer.name == "Layer"  &&  layer.prototype != null)
    layer.name = layer.prototype.name;
```

```
}

if (layer.prototype != null)
{
 Cell.EmptyLinePx(2);

 using (Cell.LineStd)
  using (new Draw.FoldoutGroup(ref layer.guiProperties, "Properties"))
   if (layer.guiProperties)
 {
  //textures
  using (Cell.LineStd)
  {
   Texture2D tex = layer.prototype.diffuseTexture;
   Draw.Field(ref tex, "Diffuse");
   if (Cell.current.valChanged)
   {
    if (layer.prototype.diffuseTexture.name == "WrColorPlaceholder2x2")
     GameObject.DestroyImmediate(layer.prototype.diffuseTexture); // removing temporary color texture
    layer.prototype.diffuseTexture = tex;
   }
  }

  using (Cell.LineStd)
  {
   Texture2D tex = layer.prototype.normalMapTexture;
```

```
 Draw.Field(ref tex, "Normal");
 if (Cell.current.valChanged)
  layer.prototype.normalMapTexture = tex;
}

using (Cell.LineStd)
{
  Texture2D tex = layer.prototype.maskMapTexture;
  Draw.Field(ref tex, "Mask");
  if (Cell.current.valChanged)
   layer.prototype.maskMapTexture = tex;
}

//color (after texture)
if (layer.prototype.diffuseTexture == null)
{
 layer.prototype.diffuseTexture = TextureExtensions.ColorTexture(2,2,layer.color);
 layer.prototype.diffuseTexture.name = "WrColorPlaceholder2x2";
}

if (layer.prototype.diffuseTexture.name == "WrColorPlaceholder2x2")
{
 using (Cell.LineStd)
 {
  using (Cell.LineStd) Draw.Field(ref layer.color, "Color");
  if (Cell.current.valChanged) layer.prototype.diffuseTexture.Colorize(layer.color);
```

```
    }
   }


   using (Cell.LineStd) layer.prototype.specular = Draw.Field(layer.prototype.specular, "Specular");
   using (Cell.LineStd) layer.prototype.smoothness = Draw.Field(layer.prototype.smoothness, "Smooth");
   using (Cell.LineStd) layer.prototype.metallic = Draw.Field(layer.prototype.metallic, "Metallic");
   using (Cell.LineStd) layer.prototype.normalScale = Draw.Field(layer.prototype.normalScale, "N. Scale"
  }


  using (Cell.LineStd)
   using (new Draw.FoldoutGroup(ref layer.guiTileSettings, "Tile Settings"))
    if (layer.guiTileSettings)
  {
   using (Cell.LineStd) layer.prototype.tileSize = Draw.Field(layer.prototype.tileSize, "Size");
   using (Cell.LineStd) layer.prototype.tileOffset = Draw.Field(layer.prototype.tileOffset, "Offset");
  }


  if (layer.guiTileSettings)
   Cell.EmptyLinePx(3);
 }



}


/*using (UI.FoldoutGroup(ref layer.guiRemapping, "Remapping", inspectorOffset:0, margins:0))
```

```
 if (layer.guiTileSettings)
 {
  using (Cell.LineStd)
  {
   Draw.Label("Red", cell:UI.Empty(Size.row));
   layer.prototype.diffuseRemapMin.x = Draw.Field(layer.prototype.diffuseRemapMin.x, cell:UI.Empty(Siz
  }
 }*/

 Cell.EmptyRowPx(2);
 }
}


[Draw.Editor(typeof(MatrixGenerators.CustomShaderOutput200))]
public static void DrawCustomShaderOutput (MatrixGenerators.CustomShaderOutput200 gen)
{
 MatrixGenerators.CustomShaderOutput200 cso = (MatrixGenerators.CustomShaderOutput200)gen;
 string[] controlTextureNames = MatrixGenerators.CustomShaderOutput200.controlTextureNames;
 string[] controlTextureChannelNames = MatrixGenerators.CustomShaderOutput200.controlTextureName

 int texturesCount = MatrixGenerators.CustomShaderOutput200.controlTextureNames.Length;
 using (Cell.LineStd) Draw.Field(ref texturesCount, "Textures Count");
 if (texturesCount != MatrixGenerators.CustomShaderOutput200.controlTextureNames.Length)
  ArrayTools.Resize(ref controlTextureNames, texturesCount, i=> "_ControlTexture"+i);
```

```
 using (Cell.LineStd) Draw.Label("Texture Names:");
 for (int i=0; i<controlTextureNames.Length; i++)
  using (Cell.LinePx(20))
  {
   Cell.current.fieldWidth = 0.9f;
   Draw.Field(ref controlTextureNames[i], i+":");
  }

 if (controlTextureNames==null || controlTextureChannelNames.Length!=controlTextureNames.Length*4)
  controlTextureChannelNames = new string[controlTextureNames.Length*4];
 for (int i=0; i<controlTextureNames.Length; i++)
 {
  controlTextureChannelNames[i*4] = controlTextureNames[i] + " R";
  controlTextureChannelNames[i*4+1] = controlTextureNames[i] + " G";
  controlTextureChannelNames[i*4+2] = controlTextureNames[i] + " B";
  controlTextureChannelNames[i*4+3] = controlTextureNames[i] + " A";
 }

 //using (Cell.Line)
 // DrawCustomMaterialWarning();

 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlinl
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
}

private static void DrawCustomShaderLayer (Generator tgen, int num)
```

```
{
 CustomShaderOutput200 gen = (CustomShaderOutput200)tgen;
 CustomShaderOutput200.CustomShaderLayer layer = gen.layers[num];
 if (layer == null) return;

 using (Cell.LinePx(32))
 {
  if (num!=0)
   using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, gen);
  else
   //disconnecting last layer inlet
   if (GraphWindow.current.graph.IsLinked(layer))
    GraphWindow.current.graph.UnlinkInlet(layer);

  Cell.EmptyRowPx(10);

  using (Cell.Row) Draw.PopupSelector(ref layer.channelNum, MatrixGenerators.CustomShaderOutput20

  Cell.EmptyRowPx(10);
  using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
 }
}

[Draw.Editor(typeof(DirectTexturesOutput200))]
public static void DrawDirectTexturesOutput (DirectTexturesOutput200 gen)
```

```
{
 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlinl
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
}

private static void DrawDirectTexturesLayer (Generator tgen, int num)
{
 DirectTexturesOutput200 gen = (DirectTexturesOutput200)tgen;
 DirectTexturesOutput200.DirectTexturesLayer layer = gen.layers[num];
 if (layer == null) return;

 Cell.EmptyLinePx(2);
 using (Cell.LineStd)
 {
  using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, gen);

  Cell.EmptyRowPx(10);

  using (Cell.Row)
  {
   //Cell.current.trackChange = false;
   Draw.EditableLabel(ref layer.name);
  }

  using (Cell.RowPx(55))
  {
```

```
   //Cell.current.trackChange = false;
   Draw.PopupSelector(ref layer.channelNum, channelLabels);
  }

  Cell.EmptyRowPx(4);
  //using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
 }
 Cell.EmptyLinePx(2);
}

private static readonly string[] channelLabels = new string[4]{ "Red", "Green", "Blue", "Alpha" };



[Draw.Editor(typeof(DirectMatricesOutput200))]
public static void DrawDirectMapsOutput (DirectMatricesOutput200 gen)
{
 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlinl
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
}

private static void DrawDirectMatricesLayer (Generator tgen, int num)
{
 DirectMatricesOutput200 gen = (DirectMatricesOutput200)tgen;
 DirectMatricesOutput200.DirectMatricesLayer layer = gen.layers[num];
 if (layer == null) return;
```

```
 using (Cell.LinePx(28))
 {
  using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, gen);


  Cell.EmptyRowPx(10);


  using (Cell.Row)
  {
   //Cell.current.trackChange = false;
   Draw.EditableLabel(ref layer.name);
  }


  Cell.EmptyRowPx(4);
  //using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
 }
 Cell.EmptyLinePx(2);
}



//public static void DrawMicroSplatLayer (LayersGenerator.Layer target, Graph graph, Generator gen, int
//in Compatibility/Editor

[Draw.Editor(typeof(Placeholders.InletOutletPlaceholder))]
[Draw.Editor(typeof(Placeholders.InletPlaceholder))]
[Draw.Editor(typeof(Placeholders.OutletPlaceholder))]
[Draw.Editor(typeof(Placeholders.Placeholder))]
```

```
public static void DrawPlaceholder (Placeholders.GenericPlaceholder placeholder)
{
 using (Cell.LinePx(80))
  Draw.Helpbox ("Generator type not found. It might be a custom generator, or a generator from the packa
}



/*[Draw.Editor(typeof(Placeholders.InletOutletPlaceholder), cat="Header")]
public static void DrawPlaceholderHeader (Placeholders.GenericPlaceholder placeholder)
{
 using (Cell.LinePx(0))
 {
  using (Cell.Row)
  {
   foreach (IInlet<object> inlet in placeholder.inlets)
   {
    if (inlet == null) continue;
    using (Cell.LineStd)
    {
     using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(inlet, placeholder);
     Cell.EmptyRowPx(8);
     //using (Cell.Row) Draw.Label(fn.inlets[i].Name);
    }
   }
  }
 }
```

```
}*/



#region Warnings


 /*public static void DrawCustomMaterialWarning ()
 {
  Terrains.TerrainSettings settings = GraphWindow.current.mapMagic.terrainSettings;
  if (settings.materialType != Terrain.MaterialType.Custom)
  {
   using (Cell.LinePx(56))
   {
    //Cell.current.margins = new Padding(4);

    GUIStyle backStyle = UI.current.textures.GetElementStyle("DPUI/Backgrounds/Foldout");
    Draw.Element(backStyle);
    Draw.Element(backStyle);

    using (Cell.Row) Draw.Label("Material Type \nis not switched \nto Custom.");

    using (Cell.RowPx(30))
     if (Draw.Button("Fix"))
     {
      settings.materialType = Terrain.MaterialType.Custom;
      GraphWindow.current.mapMagic.ApplyTerrainSettings();
```

```
      GraphWindow.current.mapMagic.ClearAllNodes();
      GraphWindow.current.mapMagic.StartGenerate();
     }
   }
   Cell.EmptyLinePx(5);
  }
}*/


public static void DrawMegaSplatShaderNameWarning ()
{
 if (GraphWindow.current.mapMagic == null  ||  !(GraphWindow.current.mapMagic is MapMagicObject ma

 Terrains.TerrainSettings settings = mapMagicObject.terrainSettings;
 {
  using (Cell.LinePx(70))
  {
   //Cell.current.margins = new Padding(4);

   GUIStyle backStyle = UI.current.textures.GetElementStyle("DPUI/Backgrounds/Foldout");
   Draw.Element(backStyle);
   Draw.Element(backStyle);

   using (Cell.Row) Draw.Label("No MegaSplat material \nis assigned as \nCustom Material in \nTerrain S

   using (Cell.RowPx(30))
```

```
    if (Draw.Button("Fix"))
    {
     Shader shader = ReflectionExtensions.CallStaticMethodFrom("Assembly-CSharp-Editor", "SplatArray
     settings.material = new Material(shader);
     settings.material.EnableKeyword("_TERRAIN");

     mapMagicObject.ApplyTerrainSettings();

     GraphWindow.current.RefreshMapMagic();
    }
  }
  Cell.EmptyLinePx(5);
 }
}


public static void DrawMegaSplatAssignedTextureArraysWarning ()
{
 if (GraphWindow.current.mapMagic == null  ||  !(GraphWindow.current.mapMagic is MapMagicObject ma

 Terrains.TerrainSettings settings = mapMagicObject.terrainSettings;
 {
  using (Cell.LinePx(70))
  {
   //Cell.current.margins = new Padding(4);
```

```
    GUIStyle backStyle = UI.current.textures.GetElementStyle("DPUI/Backgrounds/Foldout");
    Draw.Element(backStyle);
    Draw.Element(backStyle);

    using (Cell.Row) Draw.Label("Material has \nno Albedo/Height \nTexture Array \nassigned");

    using (Cell.RowPx(30))
     if (Draw.Button("Fix"))
     {
      Shader shader = ReflectionExtensions.CallStaticMethodFrom("Assembly-CSharp-Editor", "SplatArray
      settings.material = new Material(shader);
      settings.material.EnableKeyword("_TERRAIN");

      mapMagicObject.ApplyTerrainSettings();

      GraphWindow.current.RefreshMapMagic();
     }
   }
   Cell.EmptyLinePx(5);
  }
}


public static void UpdateMaterial ()
{
 if (GraphWindow.current.mapMagic == null  ||  !(GraphWindow.current.mapMagic is MapMagicObject ma
```

```
    Renderer renderer = mapMagicObject.gameObject.GetComponent<Renderer>();
    if (renderer == null) return;


    if (mapMagicObject.terrainSettings.material != renderer.sharedMaterial)
    {
     mapMagicObject.terrainSettings.material = renderer.sharedMaterial;
     mapMagicObject.ApplyTerrainSettings();
    }
   }
  #endregion
 }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Products;
using MapMagic.Terrains;

namespace MapMagic.Nodes.MatrixGenerators
{
 [System.Serializable]
 [GeneratorMenu(
  menu = "Map/Output",
  name = "Grass",
  section =2,
  drawButtons = false,
  colorType = typeof(MatrixWorld),
  iconName="GeneratorIcons/GrassOut",
  helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Grass")]
 public class GrassOutput200 : OutputGenerator, IInlet<MatrixWorld>
 {
  public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
```

```
  public OutputLevel outputLevel = OutputLevel.Main;
  public override OutputLevel OutputLevel { get{ return outputLevel; } }

  public float density = 0.5f; //number of meshes per map pixel. Should not be confused with opacity, it's no
  public DetailPrototype prototype = new DetailPrototype() { dryColor = new Color(0.95f, 1f, 0.65f), healthyC

  public enum GrassRenderMode { Grass, Billboard, MeshVertexLit, MeshUnlit };
  public GrassRenderMode renderMode;

  public override void Generate (TileData data, StopToken stop)
  {
   //loading source
   if (stop!=null && stop.stop) return;
   MatrixWorld src = data.ReadInletProduct(this);
   if (src == null) return;

//   if (!enabled)
//    { data.finalize.Remove(finalizeAction, this); return; }

   //adding to finalize
   if (stop!=null && stop.stop) return;
   if (enabled)
   {
    data.StoreOutput(this, typeof(GrassOutput200), this, src);
    data.MarkFinalize(Finalize, stop);
   }
```

```
 else
  data.RemoveFinalize(finalizeAction);
}


public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 //creating splats and prototypes arrays
 int layersCount = data.OutputsCount(typeof(GrassOutput200), inSubs:true);
 int splatsSize = data.area.active.rect.size.x;

 int[][,] detailArr = new int[layersCount][,];
 DetailPrototype[] prototypes = new DetailPrototype[layersCount];

 //filling arrays
 int i=0;
 foreach ((GrassOutput200 output, MatrixWorld product, MatrixWorld biomeMask)
  in data.Outputs<GrassOutput200,MatrixWorld,MatrixWorld>(typeof(GrassOutput200), inSubs:true))
 {
  if (stop!=null && stop.stop) return;

  detailArr[i] = CreateDetailLayer(product, biomeMask, output.density*product.PixelSize.x*product.PixelSi
  prototypes[i] = output.prototype;
  i++;
 }
```

```
 //pushing to apply
 if (stop!=null && stop.stop) return;
 ApplyData applyData = new ApplyData() {detailLayers=detailArr, detailPrototypes=prototypes, patchResc

 #if UNITY_2022_2_OR_NEWER
 applyData.scatterMode = data.globals.grassScatterMode;
 #endif

 Graph.OnOutputFinalized?.Invoke(typeof(GrassOutput200), data, applyData, stop);
 data.MarkApply(applyData);
}


private static int[,] CreateDetailLayer (Matrix matrix, Matrix biomeMask, float density, int randomNum, Tile
{
 //Matrix matrix = (Matrix)output.product;
 //Matrix biomeMask = output.biomeMask;
 //GrassLayer layer = (GrassLayer)output.layer;

 int downscaleRatio = data.globals.grassResDownscale;
 int arrSize = (data.area.active.rect.size.x-1) / downscaleRatio + 1;
 int fullSize = data.area.full.rect.size.x;
 int margins = data.area.Margins;

 int[,] detail = new int[arrSize, arrSize];
```

```
if (matrix == null) return detail;


for (int x = 0; x < arrSize; x++)
 for (int z = 0; z < arrSize; z++)
 {
  if (stop!=null && stop.stop) return null;

  int pos = (z*downscaleRatio+margins)*fullSize + (x*downscaleRatio+margins);
  float val = matrix.arr[pos];

  //interpolating value since detail resolution is 512, while height is 513
  //val += matrix.arr[pos+1] + matrix.arr[pos+fullSize] + matrix.arr[pos+fullSize+1]; //margins should preve
  //val /= 4;

  //or using minimal interpolation (creates better visual effect - grass isn't growing where it should not)
  float val1 = matrix.arr[pos+1];    if (val1<val) val=val1;
  float val2 = matrix.arr[pos+fullSize];  if (val2<val) val=val2;
  float val3 = matrix.arr[pos+fullSize+1]; if (val3<val) val=val3;

  //multiply with biome
  if (biomeMask != null) //no empty biomes in list (so no mask == root biome)
   val *= biomeMask.arr[pos]; //if mask is not assigned biome was ignored, so only main outs with mask==

  if (val < 0) val = 0; if (val > 1) val = 1;
```

```
    //the number of bushes in pixel
    val *= density*downscaleRatio*downscaleRatio;

    //random
    float rnd = data.random.Random(randomNum, x,z);

    //converting to integer with random
    int intVal = (int)val;
    float remain = val - intVal;
    if (remain>rnd) intVal++;

    detail[z, x] = intVal;
   }

  return detail;
}


public class ApplyData : IApplyData
{
  public int[][,] detailLayers;
  public DetailPrototype[] detailPrototypes;
  public int patchResolution = 16;

  #if UNITY_2022_2_OR_NEWER
  public DetailScatterMode scatterMode = DetailScatterMode.InstanceCountMode;
```

```
#endif

//public CoordRect rect; //storing both offset and size (in case the layers length is 0)



public void Apply (Terrain terrain)
{
 if (terrain==null || terrain.Equals(null) || terrain.terrainData==null) return; //chunk removed during apply

 #if UNITY_2022_2_OR_NEWER
 terrain.terrainData.SetDetailScatterMode(scatterMode);
 #endif

 int resolution = detailLayers[0].GetLength(1);
 terrain.terrainData.SetDetailResolution(resolution, patchResolution);

 terrain.terrainData.detailPrototypes = detailPrototypes; //this was after SetDetalLayer. Check if work with

 for (int i=0; i<detailLayers.Length; i++)
  terrain.terrainData.SetDetailLayer(0, 0, i, detailLayers[i]);
}

public static ApplyData Empty
 {get{ return new ApplyData() { detailLayers=new int[0][,], detailPrototypes=new DetailPrototype[0] }; }}

public int Resolution => detailLayers.Length != 0 ? detailLayers[0].GetLength(0) : 0;
```

```
#if UN_MapMagic
public void ApplyUNature (Terrain terrain)
/// Just in case I'll need to return compatibility
{

 uNatureGrassTuple uNatureTuple = null;

 if (FoliageCore_MainManager.instance != null)
 {
  uNatureTuple = (uNatureGrassTuple)dataBox;
  grassTuple = uNatureTuple.tupleInformation;
 }
 else
 {
  //Debug.LogError("uNature_MapMagic extension is enabled but no foliage manager exists on the scene
  //yield break;
  grassTuple = (TupleSet<int[][,], DetailPrototype[]>)dataBox;
 }

 int[][,] details = grassTuple.item1;
 DetailPrototype[] prototypes = grassTuple.item2;

 //resolution
 int resolution = details[0].GetLength(1);
 terrain.terrainData.SetDetailResolution(resolution, patchResolution);
```

```
 if (FoliageCore_MainManager.instance != null)
 {
  UNMapMagic_Manager.RegisterGrassPrototypesChange(prototypes);
 }

 //prototypes
 terrain.terrainData.detailPrototypes = prototypes;

 if (FoliageCore_MainManager.instance != null)
 {
  UNMapMagic_Manager.ApplyGrassOutput(uNatureTuple);
 }
 else
 {
  //Debug.LogError("uNature_MapMagic extension is enabled but no foliage manager exists on the scene
  //yield break;
  for (int i = 0; i < details.Length; i++)
  {
   terrain.terrainData.SetDetailLayer(0, 0, i, details[i]);
  }
 }
}
#endif
}
```

```
  public override void ClearApplied (TileData data, Terrain terrain)
  {
    TerrainData terrainData = terrain.terrainData;
    Vector3 terrainSize = terrainData.size;

    terrainData.detailPrototypes = new DetailPrototype[0];
    terrainData.SetDetailResolution(32, 32);
  }
 }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Terrains;

#if UN_MapMagic
using uNature.Core.Extensions.MapMagicIntegration;
using uNature.Core.FoliageClasses;
#endif



namespace MapMagic.Nodes.MatrixGenerators
{
/*
 [System.Serializable]
 [GeneratorMenu (menu="Map/Input", name ="Height In", section=1, disengageable = true)]
 public class HeightInput : StandardGenerator, IOutlet<MatrixWorld>, IPrepare
 {
  [Val("Use Initial TerrainData Holder")] public bool useInitial;
```

```
[Val("Subtract MapMagic Changes")] public bool useDelta;

public override IEnumerable<Inlet> Inlets () { yield break; }



public void Prepare (TileData data, Terrain terrain)
{
 InitialTerrainData initial = null;
 TerrainHeightData heightData = new TerrainHeightData();

 if (useInitial)
  initial = terrain.GetComponent<InitialTerrainData>();

 //using initial data holder
 //if (initial != null && !initial.Equals(null))
 // heightData.heights2D = initial.initialHeights;

 //reading terrain directly
 else
  heightData.Read(terrain);

 data.SetPrepare(this, heightData);
}

public override void GenerateProduct (TileData data, StopToken stop)
{
```

```
//getting terrain reads
TerrainHeightData heightData = data.GetPrepare<TerrainHeightData>(this);

int heightRes = heightData.heights2D.GetLength(0) - 1;
CoordRect centerRect = data.area.active.rect;
MatrixWorld matrix;

//no interpolation
if (heightRes == centerRect.size.x)
{
 matrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSize);
 FillMatrixWithFloat2D(matrix, centerRect, heightData.heights2D);
}

//with resize
else
{
 float sizeFactor = 1f * heightRes / centerRect.size.x;
 Matrix scaledMatrix = new Matrix(data.area.full.rect * sizeFactor);
 CoordRect scaledCenterRect = centerRect * sizeFactor;

 FillMatrixWithFloat2D(scaledMatrix, scaledCenterRect, heightData.heights2D);

 Matrix rescaledMatrix = new Matrix(scaledMatrix);
 rescaledMatrix.Resize(data.area.full.rect);
 matrix = new MatrixWorld(rescaledMatrix.rect, data.area.full.worldPos, data.area.full.worldSize, rescaled
```

```
 }

 //subtracting mapmagic apply
 if (useDelta)
 {
  Matrix lastAppliedHeight = data.heights; //TODO: use a special applied data (asset? for serialization)
  if (matrix.rect == lastAppliedHeight.rect)
  {
   for (int i=0; i<matrix.arr.Length; i++)
   {
    float delta = lastAppliedHeight.arr[i] - matrix.arr[i];
   }
  }
 }

 data.products[this] = ,matrix);
}



public void FillMatrixWithFloat2D (Matrix matrix, CoordRect rect, float[,] heights2D)
/// Fills the rect area of the matrix with heights2D array. Area outsize rect is filled with edge values
{
 Coord min = matrix.rect.Min; Coord max = matrix.rect.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
```

```
   {
    int ax = x - rect.offset.x;
    int az = z - rect.offset.z;

    if (ax<0) ax = 0; if (ax>rect.size.x) ax = rect.size.x;
    if (az<0) az = 0; if (az>rect.size.z) az = rect.size.z;

    float val = heights2D[az,ax];
    matrix[x,z] = val;
   }
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map/Input", name ="Splats In", section=1, disengageable = true)]
public class SplatsInput : StandardGenerator, IOutlet<MatrixWorld>, IPrepare, ISubGraph
{
 [Val("Channel")] public int channel;

 public override IEnumerable<Inlet> Inlets () { yield break; }


 public void Prepare (TileData data, Terrain terrain)
 {
  TerrainSplatData splatsData = new TerrainSplatData();
```

```
 splatsData.Read(terrain);

 data.SetPrepare(this, splatsData);
}

public override void GenerateProduct (TileData data, StopToken stop)
{
 //getting terrain reads
 TerrainSplatData splatsData = data.GetPrepare<TerrainSplatData>(this);

 int splatsRes = splatsData.splats.GetLength(0);
 CoordRect centerRect = data.area.active.rect;
 MatrixWorld matrix;

 //no interpolation
 if (splatsRes == centerRect.size.x)
 {
  matrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSize);
  FillMatrixWithFloat3D(matrix, centerRect, splatsData.splats, channel);
 }

 //with resize
 else
 {
  float sizeFactor = 1f * splatsRes / centerRect.size.x;
```

```
  Matrix scaledMatrix = new Matrix (data.area.full.rect * sizeFactor);
  CoordRect scaledCenterRect = centerRect * sizeFactor;

  FillMatrixWithFloat3D(scaledMatrix, scaledCenterRect, splatsData.splats, channel);

  Matrix rescaledMatrix = new Matrix(scaledMatrix);
  rescaledMatrix.Resize(data.area.full.rect);
  matrix = new MatrixWorld(rescaledMatrix.rect, data.area.full.worldPos, data.area.full.worldSize, rescalec
 }

 data.products[this] = ,matrix);
}

public void FillMatrixWithFloat3D (Matrix matrix, CoordRect rect, float[,,] splats, int ch)
/// Fills the rect area of the matrix with heights2D array. Area outsize rect is filled with edge values
{
 if (ch >= splats.GetLength(2)) return;

 Coord min = matrix.rect.Min; Coord max = matrix.rect.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int ax = x - rect.offset.x;
   int az = z - rect.offset.z;
```

```
    if (ax<0) ax = 0; if (ax>rect.size.x-1) ax = rect.size.x-1;
    if (az<0) az = 0; if (az>rect.size.z-1) az = rect.size.z-1;

    float val = splats[az,ax, ch];
    matrix[x,z] = val;
   }
  }
 }
*/
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Terrains;

using UnityEngine.Profiling;

#if UN_MapMagic
using uNature.Core.Extensions.MapMagicIntegration;
using uNature.Core.FoliageClasses;
#endif


namespace MapMagic.Nodes.MatrixGenerators
{
 [Serializable]
 [GeneratorMenu(
  menu = "Map/Output",
  name = "Height",
```

```
 section=2,
 colorType = typeof(MatrixWorld),
 iconName="GeneratorIcons/HeightOut",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Height")]
public class  HeightOutput200 : OutputGenerator, IInlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 public OutputLevel outputLevel = OutputLevel.Draft | OutputLevel.Main;
 public override OutputLevel OutputLevel { get{ return outputLevel; } }

 //public float height; //stored in graph
 public enum Interpolation { None, Smooth, Scale2X, Scale4X };
 //public Interpolation interpolation; //stored in globals
 //public int splitInFrames = 4; //stored in globals

 public enum ApplyType { SetHeights, SetHeightsDelayLOD, TextureToHeightmap }
 //public ApplyType mainApply = ApplyType.TextureToHeightmap;
 //public ApplyType draftApply = ApplyType.SetHeights;
 //in globals

 public bool guiApplyType = false;


 public override void Generate (TileData data, StopToken stop)
 {
```

```
   //loading source
   if (stop!=null && stop.stop) return;
   MatrixWorld src = data.ReadInletProduct(this);
   if (src == null) return;
//   if (!enabled) { data.finalize.Remove(finalizeAction, this); return; }

   //adding to finalize
   if (stop!=null && stop.stop) return;
   if (enabled)
   {
    data.StoreOutput(this, typeof(HeightOutput200), this, src);  //adding src since it's not changing
    data.MarkFinalize(Finalize, stop);
   }
   else
    data.RemoveFinalize(finalizeAction);

   #if MM_DEBUG
   Log.Add("Height generated (id:" + id + " draft:" + data.isDraft + ")");
   #endif
  }


  public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
  public static void Finalize (TileData data, StopToken stop)
  {
   //blending all biomes in data.height matrix
```

```
if (data.heights == null ||
 data.heights.rect.size != data.area.full.rect.size ||
 data.heights.worldPos != (Vector3)data.area.full.worldPos ||
 data.heights.worldSize != (Vector3)data.area.full.worldSize)
  data.heights = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSize, dat
data.heights.worldSize.y = data.globals.height;
data.heights.Fill(0);

foreach ((HeightOutput200 output, MatrixWorld product, MatrixWorld biomeMask)
 in data.Outputs<HeightOutput200,MatrixWorld,MatrixWorld> (typeof(HeightOutput200), inSubs:true) )
{
 if (data.heights == null) //height output not generated or received null result
  return;

 for (int a=0; a<data.heights.arr.Length; a++)
 {
  if (stop!=null && stop.stop) return;

  float val = product.arr[a];
  float biomeVal = biomeMask!=null ? biomeMask.arr[a] : 1;

  data.heights.arr[a] += val * biomeVal;
 }
}

//determining resolutions
```

```
if (stop!=null && stop.stop) return;
Interpolation interpolation = data.globals.heightInterpolation;
int upscale = GetUpscale(interpolation);
int margins =  data.area.Margins;
int matrixRes = (data.heights.rect.size.x - margins*2 - 1)*upscale + margins*2*upscale + 1;

//creating upscaled/blurred height matrix
if (stop!=null && stop.stop) return;
Matrix matrix;
switch (interpolation)
{
 default: matrix = data.heights; break;
 case Interpolation.Smooth:
  matrix = new Matrix(data.heights);
  MatrixOps.GaussianBlur(matrix, 0.5f);
  break;
 //case Interpolation.Scale2X:
 // matrix = new Matrix( new CoordRect(data.heights.rect.offset, new Coord(matrixRes)) );
 // MatrixOps.UpscaleFast(data.heights, matrix);
 // MatrixOps.GaussianBlur(matrix, 0.5f); //upscaleFast interpolates linear, so each new vert is exactly be
 // break;
 //nah, summary effect is better with classic resize
 case Interpolation.Scale4X: case Interpolation.Scale2X:
  matrix = new Matrix( new CoordRect(data.heights.rect.offset, new Coord(matrixRes)) );
  MatrixOps.Resize(data.heights, matrix);
  break;
```

```
}

//clamping heights to 0-1 (otherwise culing issues can occur)
matrix.Clamp01();

//2Darray resolution and
int arrRes = matrix.rect.size.x - margins*upscale*2;

//splits number (used for SetHeightsDelayLOD and Texture)
int splitSize = data.globals.heightSplit;
int numSplits = arrRes / splitSize;
if (arrRes % splitSize != 0) numSplits++;

//getting apply data
ApplyType applyType = data.isDraft ? data.globals.heightDraftApply : data.globals.heightMainApply;
IApplyData applyData;

if (applyType == ApplyType.SetHeights)
{
 float[,] heights2Dfull = new float[arrRes,arrRes];
 matrix.ExportHeights(heights2Dfull, matrix.rect.offset + margins*upscale);
 applyData = new ApplySetData() {heights2D=heights2Dfull, height=data.globals.height};
}

else if (applyType == ApplyType.SetHeightsDelayLOD)
{
```

```
 float[][,] height2DSplits = new float[numSplits][,];

 int offset = 0;
 for (int i=0; i<numSplits; i++)
 {
  int spaceLeft = arrRes - offset;
  int currSplitSize = Mathf.Min(splitSize, arrRes-offset);

  float[,] heights2D = new float[currSplitSize, arrRes];

  Coord heights2Dcoord = new Coord(
   matrix.rect.offset.x + margins*upscale,
   matrix.rect.offset.z + margins*upscale + offset );

  matrix.ExportHeights(heights2D, heights2Dcoord);

  height2DSplits[i] = heights2D;

  offset += currSplitSize;
 }

 applyData = new ApplySplitData() {heights2DSplits=height2DSplits, height=data.globals.height};
}

#if UNITY_2019_1_OR_NEWER
else //if TextureToHeightmap
```

```
 {
  byte[] bytes = new byte[arrRes*arrRes*4];
  float ushortEpsilon = 1f / 65535; //since setheights is using not full ushort range, but range-1
  matrix.ExportRawFloat(bytes, matrix.rect.offset+margins*upscale, new Coord(arrRes,arrRes), mult:0.5f-
  //not coord(margins) since matrix rect has -margins offset
  //somehow requires halved values

  applyData = new ApplyTexData() { res=arrRes, margins=margins, splitSize=splitSize, height=data.globa
 }
 #else
 else
  throw new Exception("Unknown Height Output apply type.\n For Unity 2018.4 or older choose either Set
 #endif

 //pushing to apply
 if (stop!=null && stop.stop) return;
 Graph.OnOutputFinalized?.Invoke(typeof(HeightOutput200), data, applyData, stop);
 data.MarkApply(applyData);

 #if MM_DEBUG
 Log.Add("HeightOut Finalized");
 #endif
}

private static int GetUpscale (Interpolation interpolation)
// Could be done via enum, but using this for compatibility reasons
```

```
{
 switch (interpolation)
 {
  case Interpolation.Scale2X: return 2;
  case Interpolation.Scale4X: return 4;
  default: return 1;
 }
}



public override void ClearApplied (TileData data, Terrain terrain)
{
 TerrainData terrainData = terrain.terrainData;
 Vector3 terrainSize = terrainData.size;

 terrainData.heightmapResolution = 33;
 terrain.groupingID = terrainData.heightmapResolution;
 terrainData.size = terrainSize;

 data.heights = null;
}

public interface IApplyHeightData : IApplyData { } //common type for all height applies

public class ApplySetData : IApplyData, IApplyHeightData
{
```

```
public float[,] heights2D;
public float height;
public Coord offset;  //a partial rect to avoid reading-writing all of the terrain. Size is the size of the array.

public void Read (Terrain terrain)
{
 int heightRes = terrain.terrainData.heightmapResolution;
 Read(terrain, new CoordRect(0,0,heightRes,heightRes));
}

public void Read (Terrain terrain, CoordRect rect)
{
 heights2D = terrain.terrainData.GetHeights(rect.offset.x, rect.offset.z, rect.size.x, rect.size.z);
 offset = rect.offset;
}

public void Apply (Terrain terrain)
{
 if (terrain==null || terrain.Equals(null) || terrain.terrainData==null) return; //chunk removed during apply
 TerrainData data = terrain.terrainData;

 //no resize algorithm
 Vector3 terrainSize = data.size;
 data.heightmapResolution = heights2D.GetLength(0);
 terrain.groupingID = data.heightmapResolution; //groups all the terrains of the same resolution
 data.size = new Vector3(terrainSize.x, height, terrainSize.z);
```

```
  data.SetHeights(offset.x, offset.z, heights2D);
  terrain.Flush();

  #if MM_DEBUG
  Log.Add("HeightOut Applied Set");
  #endif
 }

 public static ApplySetData Empty
  {get{ return new ApplySetData() { heights2D = new float[33,33] }; }}

 public int Resolution {get{ return heights2D.GetLength(0); }}
}


public class ApplySplitData : IApplyDataRoutine, IApplyHeightData
{
 public float[][,] heights2DSplits;
 public float height;

 public void Apply (Terrain terrain)
 {
  Profiler.BeginSample("Apply Height Splits " + terrain.transform.parent.name);

  if (terrain==null || terrain.Equals(null) || terrain.terrainData==null) return; //chunk removed during apply
```

```
 TerrainData data = terrain.terrainData;

 FastHeightmapResize(terrain, heights2DSplits[0].GetLength(1), new Vector3(data.size.x, height, data.si
 terrain.groupingID = data.heightmapResolution;

 int offset = 0;
 for (int i=0; i<heights2DSplits.Length; i++)
 {
  data.SetHeights(0, offset, heights2DSplits[i]);
  offset += heights2DSplits[i].GetLength(0);
 }

 terrain.Flush();

 Profiler.EndSample();
}


public IEnumerator ApplyRoutine (Terrain terrain)
{
 TerrainData data = terrain.terrainData;
 {
  Profiler.BeginSample("Apply ResizeHeightmap " + terrain.transform.parent.name);
  FastHeightmapResize(terrain, heights2DSplits[0].GetLength(1), new Vector3(data.size.x, height, data.s
  terrain.groupingID = data.heightmapResolution;
  Profiler.EndSample();
```

```
}
yield return null;

int offset = 0;
for (int i=0; i<heights2DSplits.Length; i++)
{
 Profiler.BeginSample("Apply HeightAddStrip " + offset + " " + terrain.transform.parent.name);

 data.SetHeightsDelayLOD(0, offset, heights2DSplits[i]);
 offset += heights2DSplits[i].GetLength(0);

 Profiler.EndSample();

 yield return null;
}



{
 Profiler.BeginSample("Apply ApplyDelayedHeightmapModification " + terrain.transform.parent.name);
 #if UNITY_2019_1_OR_NEWER
 terrain.terrainData.SyncHeightmap();
 #else
 terrain.ApplyDelayedHeightmapModification();
 #endif
 Profiler.EndSample();
}
```

```
 yield return null;


 {
  Profiler.BeginSample("Apply Flush " + terrain.transform.parent.name);
  terrain.Flush();
  terrain.terrainData.size = terrain.terrainData.size; //this will recalculate terrain bounding box in 2017.1
  Profiler.EndSample();
 }
 yield return null;
}



public static void FastHeightmapResize (Terrain terrain, int res, Vector3 size)
/// Resizing terrain the standard way is extremely slow. Even when creating a new terrain
{
 TerrainData data = terrain.terrainData;
 if ((data.size - size).sqrMagnitude > 0.01f || data.heightmapResolution != res)
 {
  if (res <= 64) //brute force
  {
   data.heightmapResolution = res;
   data.size = new Vector3(size.x, size.y, size.z);
  }


  else //setting res 64, re-scaling to 1/64, and then changing res
  {
```

```
    data.heightmapResolution = 65;
    terrain.Flush(); //otherwise unity crashes without an error
    int resFactor = (res - 1) / 64;
    data.size = new Vector3(size.x / resFactor, size.y, size.z / resFactor);
    data.heightmapResolution = res;
   }
  }
 }



 public static ApplySplitData Empty
  {get{ return new ApplySplitData() { heights2DSplits = new float[][,]{ new float[65,65] } }; }}

 public int Resolution
 {get{
  if (heights2DSplits.Length==0) return 0;
  else return heights2DSplits[0].GetLength(1);
 }}
}

#if UNITY_2019_1_OR_NEWER
public class ApplyTexData : IApplyDataRoutine, IApplyHeightData
{
 public int res;
 public int margins;
 public int splitSize;
```

```
public float height;

public byte[] texBytes;

private static Texture2D tempTex;
private static RenderTexture renTex;



public void Apply (Terrain terrain)
{
 if (terrain==null || terrain.Equals(null) || terrain.terrainData==null) return; //chunk removed during apply
 TerrainData data = terrain.terrainData;

 RectInt texRect = new RectInt(0,0,res,res);

 if (tempTex == null ||  tempTex.width != res)
  tempTex = new Texture2D(res, res, TextureFormat.RFloat, mipChain:false, linear:true);
 tempTex.LoadRawTextureData(texBytes);
 tempTex.Apply(updateMipmaps:false);

 if (renTex == null ||  renTex.width != res)
 #if UNITY_2019_2_OR_NEWER
  renTex = new RenderTexture(res,res,32, RenderTextureFormat.RFloat, mipCount:0);
 #else
  renTex = new RenderTexture(res,res,32, RenderTextureFormat.RFloat);
 #endif
```

```
 Graphics.Blit(tempTex, renTex);

 //no resize algorithm
 Vector3 terrainSize = data.size;
 data.heightmapResolution = res;
 terrain.groupingID = res;
 data.size = new Vector3(terrainSize.x, height, terrainSize.z);

 RenderTexture bacRenTex = RenderTexture.active;
 RenderTexture.active = renTex;

 data.CopyActiveRenderTextureToHeightmap(texRect, texRect.min, TerrainHeightmapSyncControl.None
 data.DirtyHeightmapRegion(texRect, TerrainHeightmapSyncControl.HeightAndLod); //or seems a bit fas
 //data.SyncHeightmap(); //doesn't seems to make difference with DirtyHeightmapRegion, waiting for rea

 RenderTexture.active = bacRenTex;
}

public IEnumerator ApplyRoutine (Terrain terrain)
{
 if (terrain==null || terrain.Equals(null) || terrain.terrainData==null) yield break; //chunk removed during ap
 TerrainData data = terrain.terrainData;

 Profiler.BeginSample("PrepareTexApply");
 RectInt texRect = new RectInt(0,0,res,res);
```

```
if (tempTex == null ||  tempTex.width != res)
 tempTex = new Texture2D(res, res, TextureFormat.RFloat, mipChain:false, linear:true);
Profiler.BeginSample("Load Tex");
tempTex.LoadRawTextureData(texBytes);
tempTex.Apply(updateMipmaps:false);
Profiler.EndSample();

if (renTex == null ||  renTex.width != res)
#if UNITY_2019_2_OR_NEWER
 renTex = new RenderTexture(res,res,32, RenderTextureFormat.RFloat, mipCount:0);
#else
 renTex = new RenderTexture(res,res,32, RenderTextureFormat.RFloat);
#endif

Profiler.BeginSample("Blit Tex");
Graphics.Blit(tempTex, renTex);
Profiler.EndSample();

//no resize algorithm
Profiler.BeginSample("Change Res");
Vector3 terrainSize = data.size;
//data.heightmapResolution = res;
data.size = new Vector3(terrainSize.x, height, terrainSize.z);
ApplySplitData.FastHeightmapResize(terrain, res, new Vector3(terrainSize.x, height, terrainSize.z));
terrain.groupingID = res;
Profiler.EndSample();
```

```
//splitting in parts
int numSplits = (int)(res/splitSize);
if (numSplits*splitSize < res) numSplits++;
Profiler.EndSample();

for (int sx = 0; sx<numSplits; sx++)
 for (int sz = 0; sz<numSplits; sz++)
 {
  Profiler.BeginSample("Apply Heightmap");

  Profiler.BeginSample("Set active rendertex");
  RenderTexture bacRenTex = RenderTexture.active;
  RenderTexture.active = renTex;
  Profiler.EndSample();

  RectInt rect = new RectInt(sx*splitSize, sz*splitSize, splitSize, splitSize);
  rect.xMax = Mathf.Min(rect.xMax, res);
  rect.yMax = Mathf.Min(rect.yMax, res);

  Profiler.BeginSample("Copy to heightmap");
  data.CopyActiveRenderTextureToHeightmap(rect, rect.min, TerrainHeightmapSyncControl.None);
  Profiler.EndSample();

  Profiler.BeginSample("Dirty heightmap");
  data.DirtyHeightmapRegion(rect, TerrainHeightmapSyncControl.HeightAndLod); //or seems a bit faster
```

```
    Profiler.EndSample();

    Profiler.BeginSample("SyncHeightmap");
    //data.SyncHeightmap(); //doesn't seems to make difference with DirtyHeightmapRegion, waiting for rea
    Profiler.EndSample();

    Profiler.BeginSample("Returning rendertex");
    RenderTexture.active = bacRenTex;
    Profiler.EndSample();

    Profiler.EndSample();

    yield return null;
   }

  #if MM_DEBUG
  Log.Add("HeightOut Applied Texture");
  #endif
 }

 public static ApplyTexData Empty
  {get{ return new ApplyTexData() { texBytes=new byte[16], height=0, splitSize = 5 }; }}

 public int Resolution {get{ return (int)Mathf.Sqrt(texBytes.Length/4); }}
}
#endif
```

```
 }

}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Terrains;

using UnityEngine.Profiling;

#if UN_MapMagic
using uNature.Core.Extensions.MapMagicIntegration;
using uNature.Core.FoliageClasses;
#endif


namespace MapMagic.Nodes.MatrixGenerators
{
 [Serializable]
 [GeneratorMenu(
  menu = "Map/Output",
  name = "Holes",
```

```
 section=2,
 colorType = typeof(MatrixWorld),
 iconName="GeneratorIcons/HeightOut",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Holes")]
public class  HolesOutput2112 : OutputGenerator, IInlet<MatrixWorld>
{
 [Val(name="Tolerance")] public float tolerance = 0.5f;

 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 public OutputLevel outputLevel = OutputLevel.Draft | OutputLevel.Main;
 public override OutputLevel OutputLevel { get{ return outputLevel; } }

 public override void Generate (TileData data, StopToken stop)
 {
  //loading source
  if (stop!=null && stop.stop) return;
  MatrixWorld src = data.ReadInletProduct(this);
  if (src == null) return;

  //adding to finalize
  if (stop!=null && stop.stop) return;
  if (enabled)
  {
   data.StoreOutput(this, typeof(HolesOutput2112), this, src);  //adding src since it's not changing
   data.MarkFinalize(Finalize, stop);
```

```
 }
 else
  data.RemoveFinalize(finalizeAction);
}



public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 int fullSize = data.area.full.rect.size.x;
 int activeSize = data.area.active.rect.size.x;
 int margins = data.area.Margins;
 int resolution = activeSize-1; //data.globals.holesRes;
 //there's no way to change holes resolution with script yey. It's always height-1.
 //otherwise everything is ready to use it

 bool[,] holes2D = new bool[resolution, resolution];

 foreach ((HolesOutput2112 output, MatrixWorld product, MatrixWorld biomeMask)
  in data.Outputs<HolesOutput2112,MatrixWorld,MatrixWorld> (typeof(HolesOutput2112), inSubs:true) )
 {
  if (product == null)
   continue;

  for (int x = 0; x < activeSize; x++)
   for (int z = 0; z < activeSize; z++)
```

```
  {
   if (stop!=null && stop.stop) return;

   int pos = (z+margins)*fullSize + (x+margins);
   float val = product.arr[pos];
   float biomeVal = biomeMask!=null ? biomeMask.arr[pos] : 1;

   int holesX = (int)(1f*x/activeSize * resolution + 0.5f);
   int holesZ = (int)(1f*z/activeSize * resolution + 0.5f);
   if (val*biomeVal > output.tolerance)
    holes2D[holesZ,holesX] = false;
   else
    holes2D[holesZ,holesX] = true;
  }
 }

 //pushing to apply
 if (stop!=null && stop.stop) return;
 ApplyData applyData = new ApplyData() {holes2D=holes2D};
 Graph.OnOutputFinalized?.Invoke(typeof(HolesOutput2112), data, applyData, stop);
 data.MarkApply(applyData);
}


public class ApplyData : IApplyData
{
```

```
 public bool[,] holes2D;

 public void Apply (Terrain terrain)
 {
  if (terrain==null || terrain.Equals(null) || terrain.terrainData==null) return; //chunk removed during apply
  TerrainData data = terrain.terrainData;

  //data.holesResolution = holes2D.GetLength(0);  //there's no way to change holes resolution with script y
  data.SetHoles(0, 0, holes2D);
  terrain.Flush();
 }

 public static ApplyData Empty
  {get{ return new ApplyData() { holes2D = new bool[33,33] }; }}

 public int Resolution {get{ return holes2D.GetLength(0); }}
}

public override void ClearApplied (TileData data, Terrain terrain)
{
 TerrainData terrainData = terrain.terrainData;
 Vector3 terrainSize = terrainData.size;

 int res = terrainData.holesResolution;
 bool[,] holes = new bool[res,res];
```

```
    for (int x = 0; x < res; x++)
     for (int z = 0; z < res; z++)
       holes[z,x] = true;


    terrainData.SetHoles(0,0,holes);


   }
  }
 }
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;

namespace MapMagic.Nodes.MatrixGenerators
{
 [System.Serializable]
 [GeneratorMenu (menu="Map/Initial", name ="Constant", iconName="GeneratorIcons/Constant", disengag
  codeFile = "MatrixInitial", codeLine = 17,
  helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Constant")]
 public class Constant200 : Generator, IOutlet<MatrixWorld>
 {
  public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
  [Val("Level", min:0)] public float level;

  public override void Generate (TileData data, StopToken stop)
  {
   MatrixWorld matrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSiz
```

```
  matrix.Fill(level);
  data.StoreProduct(this, matrix);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map/Initial", name ="Noise", iconName="GeneratorIcons/Noise", disengageable
 codeFile = "MatrixInitial", codeLine = 36,
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Noise")]
public class Noise200 : Generator, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 public enum Type { Unity=0, Linear=1, Perlin=2, Simplex=3 };
 [Val("Type")] public Type type = Type.Perlin;

 [Val("Seed")]  public int seed = 12345;
 [Val("Intensity")] public float intensity = 1f;
 [Val("Size")]  public float size = 200f;
 [Val("Detail")]  public float detail = 0.5f;
 [Val("Turbulence")] public float turbulence = 0f;
 [Val("Offset")]  public Vector2D offset = new Vector2D(0,0);


 public override void Generate (TileData data, StopToken stop)
 {
```

```
 MatrixWorld matrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSiz
 Noise noise = new Noise(data.random, seed);

        #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 if (type!=Type.Unity) //obviously, no native for unity noise
  GeneratorNoise200(matrix, noise, stop,
   (int)type, intensity, size, detail, turbulence, offset.x, offset.z,
   matrix.worldPos.x, matrix.worldPos.z, matrix.worldSize.x, matrix.worldSize.z);
 else
  Noise(matrix, noise, stop, (int)type, intensity, size, detail, turbulence, offset);
 #else
 Noise(matrix, noise, stop, (int)type, intensity, size, detail, turbulence, offset);
       #endif

 data.StoreProduct(this, matrix);
}

[DllImport("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "GeneratorNoise2
private static extern void GeneratorNoise200(Matrix matrix, Noise noise, StopToken stop,
       int type, float intensity, float size, float detail, float turbulence, float offsetX, float offsetZ,
 float worldRectPosX, float worldRectPosZ, float worldRectSizeX, float worldRectSizeZ);

private static void Noise (MatrixWorld matrix, Noise noise, StopToken stop,
 int type, float intensity, float size, float detail, float turbulence, Vector2D offset)
{
 int iterations = (int)Mathf.Log(size,2) + 1; //+1 max size iteration
```

```
 Coord min = matrix.rect.Min; Coord max = matrix.rect.Max;
 for (int x=min.x; x<max.x; x++)
 {
  for (int z=min.z; z<max.z; z++)
  {
   Vector2D relativePos = new Vector2D (
    (float)(x - matrix.rect.offset.x) / (matrix.rect.size.x-1),
    (float)(z - matrix.rect.offset.z) / (matrix.rect.size.z-1) );

   Vector2D worldPos = new Vector2D (
    relativePos.x*matrix.worldSize.x + matrix.worldPos.x,
    relativePos.z*matrix.worldSize.z + matrix.worldPos.z );

      float val = noise.Fractal(worldPos.x+offset.x, worldPos.z+offset.z, size, iterations, detail,turbulence,(in
   val *= intensity;

   if (val < 0) val = 0; //before mask?
   if (val > 1) val = 1;

   matrix[x,z] += val;
  }

  if (stop!=null && stop.stop) return; //checking stop every x line
 }
}
```

```
}



[System.Serializable]
[GeneratorMenu (menu="Map/Initial", name ="Voronoi", iconName="GeneratorIcons/Voronoi", disengagea
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Voronoi")]
public class Voronoi200 : Generator, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Intensity")] public float intensity = 1f;
 [Val("Cell Size", min:0)] public int cellSize = 50;
 [Val("Uniformity")] public float uniformity = 0;
 [Val("Seed")] public int seed = 12345;
 public enum BlendType { flat, closest, secondClosest, cellular, organic }
 [Val("Blend Type")] public BlendType blendType = BlendType.cellular;



 public override void Generate (TileData data, StopToken stop)
 {
  MatrixWorld matrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSiz
  Voronoi(matrix, new Noise(data.random,seed), stop);
  data.StoreProduct(this, matrix);
 }



 public void Voronoi (MatrixWorld matrix, Noise random, StopToken stop=null)
```

```
{
 Vector3 matrixPos = matrix.worldPos;
 Vector3 matrixSize = matrix.worldSize;

 CoordRect rect = CoordRect.WorldToPixel((Vector2D)matrixPos, (Vector2D)matrixSize, (Vector2D)cellSi
 matrixPos = new Vector3(rect.offset.x*cellSize, matrixPos.y, rect.offset.z*cellSize);  //modifying worldPos
 matrixSize = new Vector3(rect.size.x*cellSize, matrixSize.y, rect.size.z*cellSize);

 rect.offset -= 1; rect.size += 2; //leaving 1-cell margins
 matrixPos.x -= cellSize; matrixPos.z -= cellSize;
 matrixSize.x += cellSize*2; matrixSize.z += cellSize*2;

 PositionMatrix posMatrix = new PositionMatrix(rect, matrixPos, matrixSize);

 posMatrix.Scatter(uniformity, random);
 posMatrix = posMatrix.Relaxed();

 float relativeIntensity = intensity * (matrix.worldSize.x / cellSize) * 0.05f;

 Coord min = matrix.rect.Min; Coord max = matrix.rect.Max;
 for (int x=min.x; x<max.x; x++)
 {
  if (stop!=null && stop.stop) return; //checking stop every x line

  for (int z=min.z; z<max.z; z++)
  {
```

```
 //Vector3 worldPos = matrix.PixelToWorld(x,z);

 Vector2D relativePos = new Vector2D (
  (float)(x - matrix.rect.offset.x) / (matrix.rect.size.x-1),
  (float)(z - matrix.rect.offset.z) / (matrix.rect.size.z-1) );

 Vector2D worldPos = new Vector2D (
  relativePos.x*matrix.worldSize.x + matrix.worldPos.x,
  relativePos.z*matrix.worldSize.z + matrix.worldPos.z );

 Vector3 closest; Vector3 secondClosest;
 float minDist; float secondMinDist;
 posMatrix.GetTwoClosest((Vector3)worldPos, out closest, out secondClosest, out minDist, out secondM

 float val = 0;
 switch (blendType)
 {
  case BlendType.flat: val = closest.y; break;
  case BlendType.closest: val = minDist / (matrix.worldSize.x*16); break;  //(MapMagic.instance.resolutio
  case BlendType.secondClosest: val = secondMinDist / (matrix.worldSize.x*16); break;
  case BlendType.cellular: val = (secondMinDist-minDist) / (matrix.worldSize.x*16); break;
  case BlendType.organic: val = (secondMinDist+minDist)/2 / (matrix.worldSize.x*16); break;
 }

 matrix[x,z] += val*relativeIntensity;
}
```

```
  }
 }
}

[System.Serializable]
[GeneratorMenu (menu="Map/Initial", name ="Simple Form", iconName="GeneratorIcons/SimpleForm", di
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/SimpleForm")]
public class SimpleForm200 : Generator, IOutlet<MatrixWorld>, ISceneGizmo
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 public enum FormType { GradientX, GradientZ, Pyramid, Cone }
 [Val("Type")]  public FormType type = FormType.Cone;
 [Val("Intensity")] public float intensity = 1;
 [Val("Scale")]  public float scale = 1;
 [Val("Offset")]  public Vector2 offset;
 [Val("Wrap")]  public CoordRect.TileMode wrap = CoordRect.TileMode.Tile;


 public bool hideDefaultToolGizmo { get; set; }
 public bool drawOffsetScaleGizmo = false;

 public override void Generate (TileData data, StopToken stop)
 {
  MatrixWorld matrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSiz
  SimpleForm(matrix, (Vector2D)offset, scale * (Vector2D)data.area.active.worldSize, stop); //size is chunk
  matrix.Clamp01();
```

```
 data.StoreProduct(this, matrix);
}

public void SimpleForm (MatrixWorld matrix, Vector2D formOffset, Vector2D formSize, StopToken stop=n
{
 Vector2D center = formSize/2 + formOffset;
 float radius = Mathf.Min(formSize.x,formSize.z) / 2f;

 Coord min = matrix.rect.Min; Coord max = matrix.rect.Max;

 for (int x=min.x; x<max.x; x++)
 {
  if (stop!=null && stop.stop) return;

  for (int z=min.z; z<max.z; z++)
  {
   //Vector3 worldPos = matrix.PixelToWorld(x,z);

   Vector2D relativePos = new Vector2D (
    (float)(x - matrix.rect.offset.x) / (matrix.rect.size.x-1),
    (float)(z - matrix.rect.offset.z) / (matrix.rect.size.z-1) );

   Vector2D worldPos = new Vector2D (
    relativePos.x*matrix.worldSize.x + matrix.worldPos.x,
    relativePos.z*matrix.worldSize.z + matrix.worldPos.z );
```

```
   Vector2D formPos = Tile(worldPos, formOffset, formSize, wrap);


   float val = 0;
   switch (type)
   {
    case FormType.GradientX:
     val = (formPos.x-formOffset.x) / formSize.x;
     break;
    case FormType.GradientZ:
     val = (formPos.z-formOffset.z) / formSize.z;
     break;
    case FormType.Pyramid:
     float valX = (formPos.x-formOffset.x) / formSize.x; if (valX > 1-valX) valX = 1-valX;
     float valZ = (formPos.z-formOffset.z) / formSize.z; if (valZ > 1-valZ) valZ = 1-valZ;
     val = valX<valZ? valX*2 : valZ*2;
     break;
    case FormType.Cone:
     val = 1 - ((center-formPos).Magnitude)/radius;
     if (val<0) val = 0;
     break;
   }


   matrix[x,z] = val*intensity;
  }
 }
}
```

```
public Vector2D Tile (Vector2D pos, Vector2D rectOffset, Vector2D rectSize, CoordRect.TileMode tileMod
/// Tiling pos in a 0-size rect
/// Similar to CoordRect.Tile
{
 pos.x -= rectOffset.x; //tile requires 0-based coordinates
 pos.z -= rectOffset.z;

 switch (tileMode)
 {
  case CoordRect.TileMode.Clamp:
   if (pos.x < 0) pos.x = 0;
   if (pos.x >= rectSize.x) pos.x = rectSize.x - 1;
   if (pos.z < 0) pos.z = 0;
   if (pos.z >= rectSize.z) pos.z = rectSize.z - 1;
   break;

  case CoordRect.TileMode.Tile:
   pos.x = pos.x % rectSize.x;
   if (pos.x < 0) pos.x= rectSize.x + pos.x;
   pos.z = pos.z % rectSize.z;
   if (pos.z < 0) pos.z= rectSize.z + pos.z;
   break;

  case CoordRect.TileMode.PingPong:
   pos.x = pos.x % (rectSize.x*2);
```

```
    if (pos.x < 0) pos.x = rectSize.x*2 + pos.x;
    if (pos.x >= rectSize.x) pos.x = rectSize.x*2 - pos.x - 1;

    pos.z = pos.z % (rectSize.z*2);
    if (pos.z<0) pos.z=rectSize.z*2 + pos.z;
    if (pos.z>=rectSize.z) pos.z = rectSize.z*2 - pos.z - 1;
    break;
  }

  pos.x += rectOffset.x;
  pos.z += rectOffset.z;

  return pos;
}

/*[Val]
public void OnGUI_SetInScene (object graphBox)
{
  UI.Empty(Size.LinePixels(5));
  UI.CheckButton(ref drawOffsetScaleGizmo, "Set In Scene");
}*/

public void DrawGizmo ()
{
  //hideDefaultToolGizmo = drawOffsetScaleGizmo;
  //if (drawOffsetScaleGizmo)
```

```
  // GeneratorUI.DrawNodeOffsetSize(ref offset, ref scale, nodeToChange:this);
 }
}

[System.Serializable]
[GeneratorMenu (menu="Map/Initial", name ="Import", iconName="GeneratorIcons/Import", disengageable
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Import")]
public class Import200 : Generator, IOutlet<MatrixWorld>, ISceneGizmo
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Map", priority = 10, type = typeof(MatrixAsset))] public MatrixAsset matrixAsset;

 [Val("Wrap Mode", priority = 4)] public CoordRect.TileMode wrapMode = CoordRect.TileMode.Clamp;
 [Val("Scale", priority = 3)]  public float scale = 1;
 [Val("Offset", priority = 2)]  public Vector2 offset;


 public bool hideDefaultToolGizmo { get; set; }
 public bool drawOffsetScaleGizmo = false;

 static Import200()
 {
  MatrixAsset.OnReloaded += OnMatrixAssetReloaded_ReGenerate;
 }
```

```
public static void OnMatrixAssetReloaded_ReGenerate (MatrixAsset ma)
/// If this matrixAsset is used then clearing this node in all related mapmagics and starting generate
{
 MapMagicObject[] mapMagics = GameObject.FindObjectsOfType<MapMagicObject>();
 foreach (MapMagicObject mapMagic in mapMagics)
 {
  bool containsMa = false;
  if (mapMagic.graph != null)
   foreach (Import200 gen in mapMagic.graph.GeneratorsOfType<Import200>())
   {
    if (gen.matrixAsset == ma)
    {
     gen.version++;

     containsMa = true;
     break;
    }
   }

  if (containsMa)
   mapMagic.Refresh();
 }
}



public override void Generate (TileData data, StopToken stop)
```

```
{
 if (stop!=null && stop.stop) return;
 if (matrixAsset == null || matrixAsset.matrix==null || matrixAsset.matrix.rect.Count==0  || !enabled) { data.

 //preparing matrices (note that their world coordinates use the same coordsys)
 MatrixWorld dstMatrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.world

 MatrixWorld srcMatrix = new MatrixWorld(
  matrixAsset.matrix,
  (Vector2D)offset,
  data.area.active.worldSize * scale, //since it should be equal to acvtive area when scale 1
  data.globals.height);

 if (srcMatrix.rect.size == data.area.active.rect.size  &&  scale<1.0001f  &&  scale>0.999f)
 //if active resolution matches src matrix (common case when importing map exported from this terrain)
  Matrix.ReadMatrix(srcMatrix, dstMatrix, wrapMode);

 else if (srcMatrix.PixelSize.x >= dstMatrix.PixelSize.x)
  ImportWithEnlarge(srcMatrix, dstMatrix, wrapMode, stop);

 else
  ImportWithDownscale(srcMatrix, dstMatrix, wrapMode, stop);

 data.StoreProduct(this, dstMatrix);
}
```

```
public void ImportWorldInerpolated (MatrixWorld src, MatrixWorld dst, StopToken stop)
/// Takes a part of raw (src) and expands it to fill tile (dst)
{
 Coord min = dst.rect.Min; Coord max = dst.rect.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   Vector3 worldPos = dst.PixelToWorld(x,z);
   Coord srcPos = src.WorldToPixel(worldPos.x, worldPos.z);

   if (!src.rect.Contains(srcPos))
    continue;

   dst[x,z] = src.GetWorldInterpolatedValue(worldPos.x, worldPos.z); //src[srcPos];
  }
}



public static void ImportWithEnlarge (MatrixWorld src, MatrixWorld dst, CoordRect.TileMode wrapMode, S
/// Takes a part of raw (src) and expands it to fill tile (dst)
/// The new function, but doesn't work for some reason (no offset)
{
 //finding read area - dst rect in src coordsys
 Vector2D worldRatio = (Vector2D)dst.worldSize / (Vector2D)src.worldSize;
 Vector2D rectRatio = (Vector2D)dst.rect.size / (Vector2D)src.rect.size;
```

```
 Vector2D ratio = worldRatio / rectRatio;
 Vector2D readPos = (Vector2D)dst.rect.offset * ratio  -  (Vector2D)src.worldPos / src.PixelSize;
 Vector2D readSize = (Vector2D)dst.rect.size * ratio; //was /ratio, but we need dst-1 for size

 //int pixelMarg = Mathf.Max((int)(0.5f/ratio.x), 2); //2 initially, but increases for big scale values  //can't reso
 int pixelMarg = (int)(0.5f/ratio.x); //2 initially, but increases for big scale values
 CoordRect readRect = new CoordRect(Coord.Floor(readPos)-pixelMarg, (Coord)readSize+pixelMarg*2);

 if (stop!=null && stop.stop) return;
 Matrix readMatrix = new Matrix(readRect);
 Matrix.ReadMatrix(src, readMatrix, wrapMode);

 if (stop!=null && stop.stop) return;
 MatrixOps.Upsize(readMatrix, readPos, readSize, dst);
}


public static void ImportWithDownscale (MatrixWorld src, MatrixWorld dst, CoordRect.TileMode wrapMod
/// Downsizes raw (src) and copies it to tile (dst)
{
 //src (raw) rect in tile pixels coordsys
 CoordRect dstPartRect = dst.WorldRectToPixels((Vector2D)src.worldPos, (Vector2D)src.worldSize);
  //using matrix size = full area here

 //creating a small matrix
```

```
  Matrix smallMatrix = new Matrix(dstPartRect);


  if (stop!=null && stop.stop) return;
  MatrixOps.Downsize(src, smallMatrix);


  //writing small matrix to dst
  if (stop!=null && stop.stop) return;
  Matrix.ReadMatrix(smallMatrix, dst, wrapMode);
 }


 /*[Val]
 public void OnGUI_SetInScene (object graphBox)
 {
  UI.Empty(Size.LinePixels(5));
  UI.CheckButton(ref drawOffsetScaleGizmo, "Set In Scene");
 }*/


 public void DrawGizmo ()
 {
  //hideDefaultToolGizmo = drawOffsetScaleGizmo;
  //if (drawOffsetScaleGizmo)
  // GeneratorUI.DrawNodeOffsetSize(ref offset, ref scale, nodeToChange:this);
 }
}
```

```
[Serializable]
[GeneratorMenu(
 menu = "Map/Initial",
 name = "Spot",
 section=2,
 colorType = typeof(MatrixWorld),
 iconName="GeneratorIcons/Spot",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Spot")]
public class Spot210 : Generator, IOutlet<MatrixWorld>
/// Creates a single round spot
/// Analog of Objects.Stroke, but made to make brush possible without objects
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Intensity")] public float intensity = 1;
 [Val("Position")] public Vector2D position;
 [Val("Radius")] public float radius = 30;
 [Val("Hardness")] public float hardness = 0.5f;



 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
  MatrixWorld matrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSiz
  float pixelSize= matrix.PixelSize.x;
  Vector2D pixelPos = position/pixelSize;
  float pixelRadius = radius/pixelSize;
```

```
  matrix.Stroke(pixelPos, pixelRadius, hardness);

  if (intensity < 0.999f  ||  intensity > 1.001f)
   matrix.Multiply(intensity);

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, matrix);
 }
}


[System.Serializable]
//[GeneratorMenu (menu="Map/Test", name ="World To Pixel", iconName="GeneratorIcons/Constant", dis
//[GeneratorMenu (menu=null, name ="World To Pixel", iconName="GeneratorIcons/Constant", disengage
public class WorldToPixelTest : Generator, IOutlet<MatrixWorld>, IPrepare
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Level")] public float level;
 [Val("Grid")] public float grid;
 [Val("Transform", allowSceneObject =true)] public Transform tfm;
 [Val("Interpolated")] public bool interpolated;

 private Vector3 pos;

 public void Prepare (TileData data, Terrain terrain)
```

```
{
 pos = tfm.position;
}

public override void Generate (TileData data, StopToken stop)
{
 MatrixWorld matrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSiz

 for (int x=matrix.rect.offset.x; x<matrix.rect.offset.x+matrix.rect.size.x; x++)
  for (int z=matrix.rect.offset.z; z<matrix.rect.offset.z+matrix.rect.size.z; z++)
  {
   matrix[x,z] = grid * (x%2) * (z%2);
  }

 if (interpolated)
 {
  Vector3 vec = matrix.WorldToPixelInterpolated(pos.x, pos.z);
  Coord coord = Coord.Floor((Vector2D)vec);
  matrix[coord] = level;
  matrix[coord.x+1, coord.z] = level*0.75f;
  matrix[coord.x, coord.z+1] = level*0.5f;
  matrix[coord.x+1, coord.z+1] = level*0.25f;

  Vector3 retVec = matrix.PixelToWorld(vec.x, vec.z);
  DebugGizmos.DrawDot("WorldToPixelTest", retVec, 6, Color.green);
 }
```

```
  else
  {
   Coord coord = matrix.WorldToPixel(pos.x, pos.z);
   matrix[coord] = level;

   Vector3 retVec = matrix.PixelToWorld(coord.x, coord.z);
   DebugGizmos.DrawDot("WorldToPixelTest", retVec, 6, Color.green);
  }

  data.StoreProduct(this, matrix);
 }
}



[System.Serializable]
#if MM_DEBUG
[GeneratorMenu (menu="Map/Test", name ="MultiProperties", iconName="GeneratorIcons/Constant", dise
//[GeneratorMenu (menu=null, name ="MultiProperties", iconName="GeneratorIcons/Constant", disengage
#endif
public class MultiPropertiesTest : Generator, IOutlet<MatrixWorld>, IPrepare
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Bool")] public bool boolean;
 [Val("Vector2")] public Vector2 vector2;
 [Val("Vector3")] public Vector3 vector3;
 [Val("Vector4")] public Vector4 vector4;
```

```
  [Val("Color")] public Color color;
  [Val("Texture")] public Texture2D texture;
  [Val("Terrain Layer")] public TerrainLayer terrainLayer;

  public void Prepare (TileData data, Terrain terrain)
  {

  }

  public override void Generate (TileData data, StopToken stop)
  {
   Debug.Log("Bool " + boolean.ToString() + "\n" +
    "Vector2 " + vector2.ToString() + "\n" +
    "Vector3 " + vector3.ToString() + "\n" +
    "Vector4 " + vector4.ToString() + "\n" +
    "Color " + color.ToString() + "\n" +
    "Texture " + (texture!=null ? "assigned" : "null") + "\n" +  //ToString could not be called not in main thread
    "Terrain Layer " + (terrainLayer!=null ? "assigned" : "null") );
  }
 }
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;

namespace MapMagic.Nodes.MatrixGenerators
{

 [System.Serializable]
 [GeneratorMenu (menu="Map/Modifiers", name ="Curve", iconName="GeneratorIcons/Curve", disengagea
  helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Curve")]
 public class Curve200 : Generator, IInlet<MatrixWorld>, IOutlet<MatrixWorld>
 {
  public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

  public Curve curve = new Curve( new Vector2(0,0), new Vector2(1,1) );

  [NonSerialized] public float[] histogram = null;
  public const int histogramSize = 256;
```

```
public override void Generate (TileData data, StopToken stop)
{
 if (stop!=null && stop.stop) return;
 MatrixWorld src = data.ReadInletProduct(this);
 if (src == null) return;
 if (!enabled) { data.StoreProduct(this, src); return; }

 if (stop!=null && stop.stop) return;
 if (data.isPreview)
  histogram = src.Histogram(histogramSize, max:1, normalize:true);

 if (stop!=null && stop.stop) return;
 MatrixWorld dst = new MatrixWorld(src);

 if (stop!=null && stop.stop) return;
 curve.Refresh(updateLut:true);

 if (stop!=null && stop.stop) return;
 //for (int i=0; i<dst.arr.Length; i++) dst.arr[i] = curve.EvaluateLuted(dst.arr[i]);
 dst.UniformCurve(curve.lut);

 if (stop!=null && stop.stop) return;
 data.StoreProduct(this, dst);
}
```

```
}



/*[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="nonExisting", iconName="GeneratorIcons/Curve", disen
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Curve")]
public class NonExisting200 : Generator, IInlet<MatrixWorld>, IOutlet<MatrixWorld>
{
 [Val(name="Intensity")] public float brightness = 0f;
 [Val(name="Contrast")] public float contrast = 1f;

 public Curve curve = new Curve( new Vector2(0,0), new Vector2(1,1) );

 [NonSerialized] public float[] histogram = null;
 public const int histogramSize = 256;



 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
  MatrixWorld src = data.ReadInletProduct(this);
  if (src == null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }

  if (stop!=null && stop.stop) return;
  if (data.isPreview)
```

```
   histogram = src.Histogram(histogramSize, max:1, normalize:true);


  if (stop!=null && stop.stop) return;
  MatrixWorld dst = new MatrixWorld(src);


  if (stop!=null && stop.stop) return;
  curve.Refresh(updateLut:true);


  if (stop!=null && stop.stop) return;
  //for (int i=0; i<dst.arr.Length; i++) dst.arr[i] = curve.EvaluateLuted(dst.arr[i]);
  dst.UniformCurve(curve.lut);


  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, dst);
 }
}
*/


[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="Levels", iconName="GeneratorIcons/Levels", disengage
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Levels")]
public class Levels200 : Generator, IInlet<MatrixWorld>, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 //public Vector2 min = new Vector2(0,0);
```

```
//public Vector2 max = new Vector2(1,1);

public float inMin = 0;

public float inMax = 1;

public float gamma = 1f; //min/max bias. 0 for min 2 for max, 1 is straight curve


public float outMin = 0;

public float outMax = 1;


[NonSerialized] public float[] histogram = null;

public const int histogramSize = 256;


public bool guiParams = false;



public override void Generate (TileData data, StopToken stop)

{

 if (stop!=null && stop.stop) return;

 MatrixWorld src = data.ReadInletProduct(this);

 if (src == null) return;

 if (!enabled) { data.StoreProduct(this, src); return; }


 if (stop!=null && stop.stop) return;

 if (data.isPreview)

  histogram = src.Histogram(histogramSize, max:1, normalize:true);


 if (stop!=null && stop.stop) return;
```

```
  MatrixWorld dst = new MatrixWorld(src);

  if (stop!=null && stop.stop) return;
  dst.Levels(inMin, inMax, gamma, outMin, outMax);

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, dst);
 }
}

[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="Contrast", iconName="GeneratorIcons/Contrast", disen
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Contrast")]
public class Contrast200 : Generator, IInlet<MatrixWorld>, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val(name="Intensity")] public float brightness = 0f;
 [Val(name="Contrast")] public float contrast = 1f;

 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
  MatrixWorld src = data.ReadInletProduct(this);
  if (src == null) return;
```

```
  if (!enabled) { data.StoreProduct(this, src); return; }

  if (stop!=null && stop.stop) return;
  //src.Histogram(256, max:1, normalize:true);
  //if (data.isPreview)
  // histogram = src.Histogram(histogramSize, max:1, normalize:true);

  if (stop!=null && stop.stop) return;
  MatrixWorld dst = new MatrixWorld(src);

  if (stop!=null && stop.stop) return;
  dst.BrighnesContrast(brightness, contrast);

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="Unity Curve", iconName="GeneratorIcons/UnityCurve",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/UnityCurve")]
public class UnityCurve200 : Generator, IMultiInlet, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Inlet", "Inlet")] public readonly IInlet<MatrixWorld> srcIn = new Inlet<MatrixWorld>();
```

```
[Val("Mask", "Inlet")] public readonly IInlet<MatrixWorld> maskIn = new Inlet<MatrixWorld>();

public IEnumerable<IInlet<object>> Inlets() { yield return srcIn; yield return maskIn; }


public AnimationCurve curve = new AnimationCurve( new Keyframe[] { new Keyframe(0,0,1,1), new Keyfr
public Vector2 min = new Vector2(0,0);
public Vector2 max = new Vector2(1,1);


public override void Generate (TileData data, StopToken stop)
{
 if (stop!=null && stop.stop) return;
 MatrixWorld src = data.ReadInletProduct(srcIn);
 MatrixWorld mask = data.ReadInletProduct(maskIn);
 if (src == null) return;
 if (!enabled) { data.StoreProduct(this, src); return; }

 //preparing output
 if (stop!=null && stop.stop) return;
 MatrixWorld dst = new MatrixWorld(src);

 //curve
 if (stop!=null && stop.stop) return;
 AnimCurve c = new AnimCurve(curve);
 for (int i=0; i<dst.arr.Length; i++) dst.arr[i] = c.Evaluate(dst.arr[i]);

 //mask
 if (stop!=null && stop.stop) return;
```

```
  if (mask != null) dst.InvMix(src,mask);


  data.StoreProduct(this, dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="Mask", iconName="GeneratorIcons/MapMask", disenga
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/MapMask")]
public class Mask200 : Generator, IMultiInlet, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Input A", "Inlet")] public readonly Inlet<MatrixWorld> aIn = new Inlet<MatrixWorld>();
 [Val("Input B", "Inlet")] public readonly Inlet<MatrixWorld> bIn = new Inlet<MatrixWorld>();
 [Val("Mask", "Inlet")] public readonly Inlet<MatrixWorld> maskIn = new Inlet<MatrixWorld>();
 public IEnumerable<IInlet<object>> Inlets () { yield return aIn; yield return bIn; yield return maskIn; }


 [Val("Invert")] public bool invert = false;



 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
  MatrixWorld matrixA = data.ReadInletProduct(aIn);
  MatrixWorld matrixB = data.ReadInletProduct(bIn);
```

```
  MatrixWorld mask = data.ReadInletProduct(maskIn);
  if (matrixA == null || matrixB == null) return;
  if (!enabled || mask == null) { data.StoreProduct(this, matrixA); return; }

  if (stop!=null && stop.stop) return;
  MatrixWorld dst = new MatrixWorld(matrixA);

  if (stop!=null && stop.stop) return;
  dst.Mix(matrixB, mask, 0, 1, invert, false, 1);

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="Blend", iconName="GeneratorIcons/Blend", disengagea
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Blend")]
public class Blend200 : Generator, IMultiInlet, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 public class Layer
 {
  public readonly Inlet<MatrixWorld> inlet = new Inlet<MatrixWorld>();
```

```
 public BlendAlgorithm algorithm = BlendAlgorithm.add;
 public float opacity = 1;
 public bool guiExpanded = false;
}

public Layer[] layers = new Layer[] { new Layer(), new Layer() };
public Layer[] Layers => layers;
public void SetLayers(object[] ls) => layers = Array.ConvertAll(ls, i=>(Layer)i);

public IEnumerable<IInlet<object>> Inlets()
{
 for (int i=0; i<layers.Length; i++)
  yield return layers[i].inlet;
}

public override void Generate (TileData data, StopToken stop)
{
 if (stop!=null && stop.stop) return;
 if (!enabled) return;
 MatrixWorld matrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSiz

 if (stop!=null && stop.stop) return;

 if (stop!=null && stop.stop) return;
  for (int i = 0; i < layers.Length; i++)
 {
```

```
  Layer layer = layers[i];
  if (layer.inlet == null) continue;

  MatrixWorld blendMatrix = data.ReadInletProduct(layer.inlet);
  if (blendMatrix == null) continue;

  Blend(matrix, blendMatrix, layer.algorithm, layer.opacity);
 }

 data.StoreProduct(this, matrix);
}


public enum BlendAlgorithm {
 mix=0,
 add=1,
 subtract=2,
 multiply=3,
 divide=4,
 difference=5,
 min=6,
 max=7,
 overlay=8,
 hardLight=9,
 softLight=10}
```

```
 public static void Blend (Matrix m1, Matrix m2, BlendAlgorithm algorithm, float opacity=1)
 {
  switch (algorithm)
  {
   case BlendAlgorithm.mix: default: m1.Mix(m2, opacity); break;
   case BlendAlgorithm.add: m1.Add(m2, opacity); break;
   case BlendAlgorithm.subtract: m1.Subtract(m2, opacity); break;
   case BlendAlgorithm.multiply: m1.Multiply(m2, opacity); break;
   case BlendAlgorithm.divide: m1.Divide(m2, opacity); break;
   case BlendAlgorithm.difference: m1.Difference(m2, opacity); break;
   case BlendAlgorithm.min: m1.Min(m2, opacity); break;
   case BlendAlgorithm.max: m1.Max(m2, opacity); break;
   case BlendAlgorithm.overlay: m1.Overlay(m2, opacity); break;
   case BlendAlgorithm.hardLight: m1.HardLight(m2, opacity); break;
   case BlendAlgorithm.softLight: m1.SoftLight(m2, opacity); break;
  }
 }
}



[System.Serializable]
[GeneratorMenu (
 menu="Map/Modifiers",
 name ="Normalize",
 disengageable = true,
 iconName="GeneratorIcons/Normalize",
```

```
 drawInlets = false,
 drawOutlet = false,
 colorType = typeof(MatrixWorld),
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Normalize")]
public class Normalize200 : Generator, IMultiInlet, IMultiOutlet
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 public class NormalizeLayer : IInlet<MatrixWorld>, IOutlet<MatrixWorld>
 {
  public float Opacity { get; set; }

  public Generator Gen { get; private set; }
  public void SetGen (Generator gen) => Gen=gen;
  public NormalizeLayer (Generator gen) { this.Gen = gen; }
  public NormalizeLayer () { Opacity = 1; }

  public ulong id; //properties not serialized
  public ulong Id { get{return id;} set{id=value;} }
  public ulong LinkedOutletId { get; set; }  //if it's inlet. Assigned every before each clear or generate
  public ulong LinkedGenId { get; set; }

  public IUnit ShallowCopy() => (NormalizeLayer)this.MemberwiseClone();
 }

 public NormalizeLayer[] layers = new NormalizeLayer[0];
```

```
public NormalizeLayer[] Layers => layers;

public void SetLayers(object[] ls) => layers = Array.ConvertAll(ls, i=>(NormalizeLayer)i);



public IEnumerable<IInlet<object>> Inlets()

{

 for (int i=0; i<layers.Length; i++)

  yield return layers[i];

}

public IEnumerable<IOutlet<object>> Outlets()

{

 for (int i=0; i<layers.Length; i++)

  yield return layers[i];

}

public override void Generate (TileData data, StopToken stop)

{

 if (layers.Length == 0) return;

 //reading/copying products

 MatrixWorld[] dstMatrices = new MatrixWorld[layers.Length];

 float[] opacities = new float[layers.Length];

 if (stop!=null && stop.stop) return;

 for (int i=0; i<layers.Length; i++)
```

```
  {
   if (stop!=null && stop.stop) return;

   MatrixWorld srcMatrix = data.ReadInletProduct(layers[i]);
   if (srcMatrix != null) dstMatrices[i] = new MatrixWorld(srcMatrix);
   else dstMatrices[i] = new MatrixWorld(data.area.full.rect, (Vector3)data.area.full.worldPos, (Vector3)data

   opacities[i] = layers[i].Opacity;
  }

  //normalizing
  if (stop!=null && stop.stop) return;
  dstMatrices.FillNulls(() => new MatrixWorld(data.area.full.rect, (Vector3)data.area.full.worldPos, (Vector3
  dstMatrices[0].Fill(1);
  Matrix.BlendLayers(dstMatrices, opacities);

  //saving products
  if (stop!=null && stop.stop) return;
  for (int i=0; i<layers.Length; i++)
   data.StoreProduct(layers[i], dstMatrices[i]);
 }
}

[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="Blur", iconName="GeneratorIcons/Blur", disengageable
```

```
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Blur")]
public class Blur200 : Generator, IInlet<MatrixWorld>, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Downsample")] public float downsample = 10f;
 [Val("Blur")] public float blur = 3f;

 public override void Generate (TileData data, StopToken stop)
 {
  MatrixWorld src = data.ReadInletProduct(this);
  if (src == null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }

  MatrixWorld dst = new MatrixWorld(src);

  int rrDownsample = (int)(downsample / Mathf.Sqrt(dst.PixelSize.x));
  float rrBlur = blur / dst.PixelSize.x;

  if (rrDownsample > 1)
   MatrixOps.DownsampleBlur(src, dst, rrDownsample, rrBlur);
  else
   MatrixOps.GaussianBlur(src, dst, rrBlur);

  data.StoreProduct(this, dst);
 }
}
```

```
[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="Cavity", iconName="GeneratorIcons/Cavity", disengage
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Cavity")]
public class Cavity200 : Generator, IInlet<MatrixWorld>, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 public enum CavityType { Convex=0, Concave=1, Both=2 }
 [Val("Type")]  public CavityType type = CavityType.Convex;
 [Val("Intensity")] public float intensity = 3;
 [Val("Spread")]  public float spread = 10; //actually the pixel size (in world units) of the lowerest mipmap. S

 public override void Generate (TileData data, StopToken stop)
 {
  MatrixWorld src = data.ReadInletProduct(this);
  if (src == null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }

  if (stop!=null && stop.stop) return;
  MatrixWorld dst = new MatrixWorld(src.rect, src.worldPos, src.worldSize);

  if (stop!=null && stop.stop) return;
  Cavity(src, dst, type, intensity, spread, src.PixelSize.x, data.area.active.worldSize, stop);

  if (stop!=null && stop.stop) return;
```

```
 data.StoreProduct(this, dst);
}

public static void Cavity (Matrix src, Matrix dst,
 CavityType cavityType, float intensity, float spread,
 float pixelSize, Vector2D worldSize, StopToken stop)
{
 MatrixOps.Cavity(src, dst); //produces the map with range -1 - 1
 dst.Multiply(1f / Mathf.Pow(pixelSize, 0.25f));

 float minResolution = worldSize.x / spread;  //area worldsize / (spread = min pixel size)
 float downsample = Mathf.Log(src.rect.size.x, 2);
 downsample -= Mathf.Log(minResolution, 2);

 if (stop!=null && stop.stop) return;
 MatrixOps.OverblurMipped(dst, downsample:Mathf.Max(0,downsample), escalate:1.5f);

 if (stop!=null && stop.stop) return;
 dst.Multiply(intensity*100f);

 switch (cavityType)
 {
  case CavityType.Convex: dst.Invert(); break;
  //case CavityType.Concave: break;
  case CavityType.Both: dst.Invert(); dst.Multiply(0.5f); dst.Add(0.5f); break;
 }
```

```
  dst.Clamp01();

  //blending 50% map if downsample doesn't allow cavity here (for drafts or low-res)
  if (stop!=null && stop.stop) return;
  if (downsample < 0f)
  {
   float subsample = -downsample/4;
   if (subsample > 1) subsample = 1;

   float avg = dst.Average();
   dst.Fill(avg, subsample);
  }
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="Slope", iconName="GeneratorIcons/Slope", disengagea
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Slope")]
public class Slope200 : Generator, IInlet<MatrixWorld>, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("From")]   public float from = 30;
 [Val("To")]    public float to = 90;
 [Val("Smooth Range")] public float range = 30f;
```

```
public override void Generate (TileData data, StopToken stop)
{
 MatrixWorld src = data.ReadInletProduct(this);
 if (src==null) return;
 if (!enabled) { data.StoreProduct(this, src); return; }

 if (stop!=null && stop.stop) return;
 MatrixWorld dst = Slope(src, data.globals.height, from, to, range);

 if (stop!=null && stop.stop) return;
 data.StoreProduct(this, dst);
}

public static MatrixWorld Slope (MatrixWorld heights, float height, float from, float to, float range) =>
 Slope(heights, heights.worldPos, heights.worldSize, height, from, to, range);

public static MatrixWorld Slope (Matrix heights, Vector3 worldPos, Vector3 worldSize, float height, float fr
{
 //delta map
 MatrixWorld delta = new MatrixWorld(heights.rect, worldPos, worldSize);
 MatrixOps.Delta(heights, delta);

 //slope map
 float minAng0 = from-range/2;
```

```
float minAng1 = from+range/2;
float maxAng0 = to-range/2;
float maxAng1 = to+range/2;

float pixelSize = 1f * worldSize.x / heights.rect.size.x; //using the terain-height relative values

float minDel0 = Mathf.Tan(minAng0*Mathf.Deg2Rad) * pixelSize / height;
float minDel1 = Mathf.Tan(minAng1*Mathf.Deg2Rad) * pixelSize / height;
float maxDel0 = Mathf.Tan(maxAng0*Mathf.Deg2Rad) * pixelSize / height;
float maxDel1 = Mathf.Tan(maxAng1*Mathf.Deg2Rad) * pixelSize / height;

//dealing with 90-degree
if (maxAng0 > 89.9f) maxDel0 = 20000000;
if (maxAng1 > 89.9f) maxDel1 = 20000000;

if (from < 0.00001f) { minDel0=-1; minDel1=-1; }
//not right, but intuitive - if user wants to mask from 0 don't add gradient here

//ignoring min if it is zero
//if (steepness.x<0.0001f) { minDel0=0; minDel1=0; }

delta.SelectRange(minDel0, minDel1, maxDel0, maxDel1);

return delta;
}
```

```
}



[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="Selector", iconName="GeneratorIcons/Selector", diseng
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Selector")]
public class Selector200 : Generator, IInlet<MatrixWorld>, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on


 public enum RangeDet { Transition, MinMax}
 public RangeDet rangeDet = RangeDet.Transition;
 public enum Units { Map, World }
 public Units units = Units.Map;
 public Vector2 from = new Vector2(0.4f, 0.6f);
 public Vector2 to = new Vector2(1f, 1f);


 public override void Generate (TileData data, StopToken stop)
 {
  MatrixWorld src = data.ReadInletProduct(this);
  if (src==null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }


  if (stop!=null && stop.stop) return;
  MatrixWorld dst = new MatrixWorld(src);
```

```
  if (stop!=null && stop.stop) return;
  Select(dst, from, to, inWorldUnits:units==Units.World, worldHeight:data.globals.height);

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, dst);
 }

 public static void Select (MatrixWorld dst, Vector2 from, Vector2 to, bool inWorldUnits, float worldHeight)
 {
  float min0 = from.x;  if (inWorldUnits) min0 /= worldHeight;
  float min1 = from.y;  if (inWorldUnits) min1 /= worldHeight;
  float max0 = to.x;    if (inWorldUnits) max0 /= worldHeight;
  float max1 = to.y;    if (inWorldUnits) max1 /= worldHeight;
  dst.SelectRange(min0, min1, max0, max1);
 }
}


[System.Serializable]
[GeneratorMenu (
 menu="Map/Modifiers",
 name ="Terrace",
 iconName="GeneratorIcons/Terrace",
 disengageable = true,
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Terrace")]
public class Terrace200 : Generator, IInlet<MatrixWorld>, IOutlet<MatrixWorld>
```

```
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Seed")]   public int seed = 12345;
 [Val("Num")]   public int num = 10;
 [Val("Uniformity")] public float uniformity = 0.5f;
 [Val("Steepness")]  public float steepness = 0.5f;
 //[Val("Intensity")]  public float intensity = 1f;


 public override void Generate (TileData data, StopToken stop)
 {
  MatrixWorld src = data.ReadInletProduct(this);
  if (src == null || num <= 1) return;
  if (!enabled) { data.StoreProduct(this, src); return; }

  MatrixWorld dst = new MatrixWorld(src);
  float[] terraceLevels = TerraceLevels(new Noise(data.random,seed), num, uniformity);

  if (stop!=null && stop.stop) return;
  dst.Terrace(terraceLevels, steepness);

  data.StoreProduct(this, dst);
 }
```

```
 public static float[] TerraceLevels (Noise random, int num, float uniformity)
 {
  //creating terraces
  float[] terraces = new float[num];

  float step = 1f / (num-1);
  for (int t=1; t<num; t++)
   terraces[t] = terraces[t-1] + step;

  for (int i=0; i<10; i++)
   for (int t=1; t<num-1; t++)
   {
    float rndVal = random.Random(i);
    rndVal = terraces[t-1] +  rndVal*(terraces[t+1]-terraces[t-1]);
    terraces[t] = terraces[t]*uniformity + rndVal*(1-uniformity);
   }

  return terraces;
 }
}


[System.Serializable]
[GeneratorMenu (
 menu="Map/Modifiers",
 name ="Direction",
```

```
 iconName="GeneratorIcons/Direction",
 disengageable = true,
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Direction")]
public class Direction210 : Generator, IInlet<MatrixWorld>, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Hor Angle", min=-360, max=360)] public float horAngle = 0;
 [Val("Vert Angle", min=-89.99f, max=89.99f)] public float vertAngle = 0;
 [Val("Intensity", min =0)] public float intensity = 1;
 [Val("Wrapping", min=-1, max=1)] public float wrapping = 0;

 public override void Generate (TileData data, StopToken stop)
 {
  MatrixWorld src = data.ReadInletProduct(this);
  if (src == null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }

  Vector3 dir = new Vector3(Mathf.Sin(horAngle*Mathf.Deg2Rad), Mathf.Tan(vertAngle*Mathf.Deg2Rad),
  dir = dir.normalized;

  MatrixWorld dst = new MatrixWorld(src.rect, src.worldPos, src.worldSize);

  if (stop!=null && stop.stop) return;
  MatrixOps.NormalsDir(src, dst, dir, data.area.PixelSize.x, data.globals.height, intensity, wrapping);
```

```
  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, dst);
 }
}


[System.Serializable]
[GeneratorMenu (
 menu="Map/Modifiers",
 name ="Ledge",
 iconName="GeneratorIcons/Ledge",
 disengageable = true,
 advancedOptions = true,
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Ledge")]
public class Ledge210 : Generator, IInlet<MatrixWorld>, IOutlet<MatrixWorld>, IMultiInlet
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Level")]  public float level = 50;
 [Val("Contour Blur")] public float contourBlur = 3;
 [Val("Height")]  public float height = 10;
 [Val("Steep")]  public float steep = 30; //aka width

 [Val("Top Shoulder", "Advanced")] public float topShoulder = 2f;
 [Val("Bottom Shoulder", "Advanced")] public float bottomShoulder = 2f;
 [Val("Smooth", "Advanced")] public bool smooth = true;
```

```
[Val("Mask", "Inlet")] public readonly Inlet<MatrixWorld> heightMaskIn = new Inlet<MatrixWorld>();
public IEnumerable<IInlet<object>> Inlets()  { yield return heightMaskIn; }

public override void Generate (TileData data, StopToken stop)
{
 MatrixWorld src = data.ReadInletProduct(this);
 if (src == null) return;
 if (!enabled) { data.StoreProduct(this, src); return; }

 if (stop!=null && stop.stop) return;

 MatrixWorld heightMask = data.ReadInletProduct(heightMaskIn);

 Matrix ledgeMask = LedgeMask(src, contourBlur);

 MatrixWorld dst = new MatrixWorld(src.rect, src.worldPos, src.worldSize);
 /*LedgeStep(src, mask, dst,
   minFrom: (level - height*steep/2) / data.globals.height,
   maxFrom: (level + height*steep/2) / data.globals.height,
   minTo: (level - height/2) / data.globals.height,
   maxTo: (level + height/2) / data.globals.height,
   bottomShoulder, topShoulder);*/

 LedgeStep(src, ledgeMask, heightMask, dst,
  level/data.globals.height, height/data.globals.height, steep,
```

```
  bottomShoulder, topShoulder, smooth);


 if (stop!=null && stop.stop) return;
 data.StoreProduct(this, dst);
}

public static Matrix LedgeMask (Matrix src, float blur)
{
 Matrix mask;

 if (blur > 2)
 {
  mask = new Matrix(src);
  MatrixOps.DownsampleBlur(mask, (int)blur, 1.75f);
 }
 else if (blur > 0.001f)
 {
  mask = new Matrix(src);
  MatrixOps.GaussianBlur(mask, blur);
 }
 else
  mask = src;

 return mask;
}
```

```
/*  #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
   [DllImport("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "GeneratorLedge
   private static extern void LedgeStep (Matrix src, Matrix mask, Matrix dst,
    float minFrom, float maxFrom, float minTo, float maxTo,
    float bottomShoulder, float topShoulder);
  #else */
   public static void LedgeStep (Matrix src, Matrix mask, Matrix intensity, Matrix dst,
    float minFrom, float maxFrom, float minTo, float maxTo,
    float bottomShoulder, float topShoulder)
   {
    for (int i=0; i<dst.arr.Length; i++)
    {
     float heightVal = src.arr[i];
     float maskVal = mask.arr[i];

     if (maskVal < minFrom)
     {
      float p = heightVal/minFrom;
      p = ((p/bottomShoulder) + (bottomShoulder-1)/(bottomShoulder)) * p  +  p * (1-p); //inverse of (p/should
      dst.arr[i] = minTo * p;
     }

     else if (maskVal > maxFrom)
     {
      float p = ((heightVal-maxFrom)/(1-maxFrom));
      p = (p/topShoulder)*(1-p) + p*p; //Mathf.Pow(p,topShoulder);
```

```
      dst.arr[i] = maxTo + (1-maxTo) * p;
     }


     else
     {
      float p = (heightVal-minFrom) / (maxFrom-minFrom);
      float ip = 1-p;
      float tp = ((p/topShoulder) + (topShoulder-1)/(topShoulder)) * p  +  p * (1-p);
      float lp = (p/bottomShoulder)*(1-p) + p*p;
      float bp = 3*p*p - 2*p*p*p;


      p = lp*(1-bp)  +  tp*bp;


      dst.arr[i] = minTo + (maxTo-minTo)*p;
     }
    }
   }
//  #endif



  public static void LedgeStep (Matrix src, Matrix mask,  Matrix intensity, Matrix dst,
   float level, float height, float steep,
   float bottomShoulder, float topShoulder,
   bool smooth=true)
  {
   for (int i=0; i<dst.arr.Length; i++)
```

```
{
 float intensityVal = intensity!=null ? intensity.arr[i] : 1;
 float heightVal = src.arr[i];
 float maskVal = mask.arr[i];

 float start = level - 1f/steep*intensityVal;
 float end = level + 1f/steep*intensityVal;

 float startShoulder = start - (height*intensityVal)/2*bottomShoulder;
 float endShoulder = end + (height*intensityVal)/2*topShoulder;

 //ledge itself
 if (maskVal > start  &&  maskVal < end)
 {
  float p = (maskVal-start) / (end-start);

  if (smooth)
   p = 3*p*p - 2*p*p*p;
  //p = 3*p*p - 2*p*p*p;
  //p = 6*p*p*p*p*p - 15*p*p*p*p + 10*p*p*p;
  dst.arr[i] = (heightVal - (height*intensityVal)/2)*(1-p)  +  (heightVal + (height*intensityVal)/2)*p;
 }

 //bottom shoulder
 else if (maskVal > startShoulder  &&  maskVal < level)
 {
```

```
      float p = (maskVal-startShoulder) / (start-startShoulder);

      if (smooth)
        p = 3*p*p - 2*p*p*p;

      dst.arr[i] = heightVal - (height*intensityVal)/2*p;
     }

     //top shoulder
     else if (maskVal < endShoulder  &&  maskVal > level)
     {
      float p = (endShoulder-maskVal) / (endShoulder-end);

      if (smooth)
        p = 3*p*p - 2*p*p*p;

      dst.arr[i] = heightVal + (height*intensityVal)/2*p;
     }

     //everything else
     else
      dst.arr[i] = heightVal;
    }
   }
}
```

```
[System.Serializable]
[GeneratorMenu (
 menu="Map/Modifiers",
 name ="Beach",
 iconName="GeneratorIcons/Beach",
 disengageable = true,
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Beach")]
public class Beach210 : Generator, IInlet<MatrixWorld>, IMultiInlet, IOutlet<MatrixWorld>, IMultiOutlet
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Level")] public float level = 50;
 [Val("Contour Relax")] public float relax = 100;
 [Val("Size")] public float size = 40;
 [Val("Height")] public float height = 5;
 [Val("Sand Tex Blur")] public float sandBlur = 5f;

 [Val("Mask", "Inlet")] public readonly Inlet<MatrixWorld> beachMaskIn = new Inlet<MatrixWorld>();
 [Val("Sand", "Inlet")] public readonly Outlet<MatrixWorld> sandMaskOut = new Outlet<MatrixWorld>();

 public IEnumerable<IInlet<object>> Inlets()  { yield return beachMaskIn; }
 public IEnumerable<IOutlet<object>> Outlets()  { yield return sandMaskOut; }


 public override void Generate (TileData data, StopToken stop)
```

```
{
 MatrixWorld src = data.ReadInletProduct(this);
 if (src == null) return;
 if (!enabled) { data.StoreProduct(this, src); return; }

 MatrixWorld beachMask = data.ReadInletProduct(beachMaskIn);

 if (stop!=null && stop.stop) return;
 Matrix shoreSpread = PrepareShoreMask(src,
   level/data.globals.height,
   size/data.area.PixelSize.x,
   height/data.globals.height,
   relax, stop);

 if (stop!=null && stop.stop) return;
 MatrixWorld dst = new MatrixWorld(src.rect, src.worldPos, src.worldSize);
 BeachHeight(src, dst, shoreSpread, beachMask,
   level/data.globals.height,
   height/data.globals.height);

 if (stop!=null && stop.stop) return;
 MatrixWorld sand = new MatrixWorld(src.rect, src.worldPos, src.worldSize);
 BeachSand(src, dst, sand, level/data.globals.height, sandBlur/data.globals.height);

 if (stop!=null && stop.stop) return;
 dst.Max(src);
```

```
 if (stop!=null && stop.stop) return;
 data.StoreProduct(this, dst);
 data.StoreProduct(sandMaskOut, sand);
}

public static Matrix PrepareShoreMask (Matrix src, float level, float size, float height, float relax, StopToke
/// Relaxes the shore line and creates the beach mask
{
 Matrix shore = new Matrix(src);
 shore.Select(level+height/2);

 //relaxing shore line: moving beach inside
 Matrix insShore = new Matrix(shore.rect);
  if (stop!=null && stop.stop) return null;

 shore.InvertOne();
 MatrixOps.SpreadLinear(shore, insShore, 1f/relax, diagonals:true, quarters:true);
 insShore.Select(0.01f);
 insShore.InvertOne();
  if (stop!=null && stop.stop) return null;

 MatrixOps.GaussianBlur(insShore, 3.75f); //just to smoothen remaining edges
 insShore.Select(0.5f);
  if (stop!=null && stop.stop) return null;
```

```
  //and then outside
  shore.Fill(0);
  MatrixOps.SpreadLinear(insShore, shore, 1f/relax, diagonals:true, quarters:true);
  shore.Select(0.01f);
   if (stop!=null && stop.stop) return null;

  //now standard shore spreading
  Matrix shoreSpread = insShore; shoreSpread.Fill(0); //re-using matrix // = new Matrix(src.rect);
  MatrixOps.SpreadLinear(shore, shoreSpread, 1f/size, diagonals:true, quarters:true);  //0.5 above water, 0
  shoreSpread.Clamp01();
   if (stop!=null && stop.stop) return null;

  //blurring a bit (to avoid spread artifacts)
  //MatrixOps.DownsampleBlur(shoreSpread, smooth, 1.75f);
  MatrixOps.GaussianBlur(shoreSpread, 3.75f);

  return shoreSpread;
 }

/*  #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
  [DllImport("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "GeneratorBeach
  private static extern void BeachHeight (Matrix src, Matrix dst, Matrix shoreSpread, Matrix mask, float leve
 #else*/
  public static void BeachHeight (Matrix src, Matrix dst, Matrix shoreSpread, Matrix mask, float level, float h
  /// Generates beach height using spread & mask. Should be combined with original height with Max
  {
```

```
    for (int i=0; i<dst.count; i++)
    {
     float heightVal = src.arr[i];
     float shoreVal = shoreSpread.arr[i];
     float maskVal = mask != null ? 1-(1-mask.arr[i])*(1-mask.arr[i]) : 1;

     if (shoreVal > 0.999f)
      dst.arr[i] = level+height/2;

     if (shoreVal < 0.0001f)
      dst.arr[i] = src.arr[i];

     float percent = shoreVal * maskVal;
     float val = level-height/2 + percent*height;

     float bottomPercent = (0.5f-percent) * 2; //0->1, 0.5->0
     if (bottomPercent < 0) bottomPercent = 0;
     bottomPercent = 3*bottomPercent*bottomPercent - 2*bottomPercent*bottomPercent*bottomPercent;
     val = val*(1-bottomPercent) + heightVal*bottomPercent;

     dst.arr[i] = val;
    }
   }
//  #endif

  #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
```

```
 [DllImport("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "GeneratorBeach
 private static extern void BeachSand (Matrix heights, Matrix beach, Matrix sand, float waterLevel, float m
#else
 public static void BeachSand (Matrix heights, Matrix beach, Matrix sand, float waterLevel, float maxDelta)
 /// Creates sand mask based on beach and original height
 {
  for (int i=0; i<heights.count; i++)
  {
   float heightVal = heights.arr[i];
   float beachVal = beach.arr[i];

   //for above water
   if (beachVal > waterLevel)
   {
    float delta = beachVal - heightVal;
    float percent = delta / maxDelta;
    if (percent > 1) percent = 1;
    if (percent < 0) percent = 0;

    sand.arr[i] = percent;
   }

   //for shallow underwater
   else if (beachVal > waterLevel-maxDelta/2)
   {
    if (beachVal > heightVal+0.0001f)
```

```
      sand.arr[i] = 1;
     else
      sand.arr[i] = 0;
    }

    //deep - always in sand
    else
     sand.arr[i] = 1;
   }
  }
 #endif

}


[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="Erosion", iconName="GeneratorIcons/Erosion", disenga
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Erision")]
public class Erosion200 : Generator, IInlet<MatrixWorld>, IOutlet<MatrixWorld>, ICustomComplexity
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Iterations")]   public int iterations = 3;
 [Val("Durability")] public float terrainDurability=0.9f;
 //[Val("Erosion")]
  public float erosionAmount=1f;
```

```
[Val("Sediment")]  public float sedimentAmount=0.75f;

[Val("Fluidity")] public int fluidityIterations=3;

[Val("Relax")]  public float relax=0.0f;


[DllImport ("NativePlugins", EntryPoint = "SetOrder")]

private static extern void SetOrder (float[] refArr, int[] orderArr, int length);


[DllImport ("NativePlugins", EntryPoint = "MaskBorders")]

private static extern int MaskBorders (int[] orderArr, CoordRect matrixRect);


[DllImport ("NativePlugins", EntryPoint = "CreateTorrents")]

private static extern int CreateTorrents (float[] heights, int[] order, float[] torrents, CoordRect matrixRect);


[DllImport ("NativePlugins", EntryPoint = "Erode")]

private static extern int Erode (float[] heights, float[] torrents, float[] mudflow, int[] order, CoordRect matrixR

 float erosionDurability = 0.9f, float erosionAmount = 1, float sedimentAmount = 0.5f);


[DllImport ("NativePlugins", EntryPoint = "TransferSettleMudflow")]

private static extern int TransferSettleMudflow(float[] heights, float[] mudflow, float[] sediments, int[] order,


public float Complexity {get{ return iterations*2; }}

public float Progress (TileData data) { return data.GetProgress(this); }


public override void Generate (TileData data, StopToken stop)

{
```

```
 #if MM_DEBUG
 Log.Add("Generating Erosion (draft:" + data.isDraft + " pos:" + data.area.active.worldPos);
 #endif

 MatrixWorld src = data.ReadInletProduct(this);
 if (src == null) return;
 if (!enabled || iterations <= 0) { data.StoreProduct(this, src); return; }

 MatrixWorld dst = new MatrixWorld(src);
 Erosion(dst, data.isDraft, data, iterations, terrainDurability, erosionAmount, sedimentAmount, fluidityItera
  if (stop!=null && stop.stop) return;

 data.StoreProduct(this, dst);
}


public static void Erosion (MatrixWorld dstHeight, bool isDraft, TileData data,
 int iterations, float terrainDurability, float erosionAmount, float sedimentAmount, int fluidityIterations, float
 ICustomComplexity thisGen, StopToken stop=null)
{
 //allocating temporary matrices
 Matrix2D<int> order = new Matrix2D<int>(dstHeight.rect);
 Matrix torrents = new Matrix(dstHeight.rect);
 Matrix mudflow = new Matrix(dstHeight.rect);
 Matrix sediment = new Matrix(dstHeight.rect);
```

```
int curIterations = iterations;
int curFluidity = fluidityIterations;

if (isDraft)
{
 curIterations = iterations/3;
 curFluidity = fluidityIterations/3;
}

//calculate erosion
for (int i=0; i<curIterations; i++)
{
 Den.Tools.Erosion.SetOrder(dstHeight, order);
 if (stop!=null && stop.stop) return;

 Den.Tools.Erosion.MaskBorders(order);
 if (stop!=null && stop.stop) return;

 Den.Tools.Erosion.CreateTorrents(dstHeight, order, torrents);
 if (stop!=null && stop.stop) return;

 Den.Tools.Erosion.Erode(dstHeight, torrents, mudflow, order, terrainDurability, erosionAmount, sedimen
 if (stop!=null && stop.stop) return;

 Den.Tools.Erosion.TransferSettleMudflow(dstHeight, mudflow, sediment, order, curFluidity);
 if (stop!=null && stop.stop) return;
```

```
    if (relax>0.0001f  &&  i!=curIterations-1)
      MatrixOps.GaussianBlur(dstHeight, relax/(i+1));
    if (stop!=null && stop.stop) return;

    data.SetProgress(thisGen, i*2);
   }
  }
 }

[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="Sediment", iconName="GeneratorIcons/Sediment", dise
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Sediment")]
public class Sediment210 : Generator, IMultiInlet, IMultiOutlet
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Original", "Outlet")] public readonly Inlet<MatrixWorld> origIn = new Inlet<MatrixWorld>();
 [Val("Eroded", "Outlet")] public readonly Inlet<MatrixWorld> erodedIn = new Inlet<MatrixWorld>();
 public IEnumerable<IInlet<object>> Inlets()  { yield return origIn; yield return erodedIn; }

 [Val("Cliff", "Outlet")] public readonly Outlet<MatrixWorld> cliffOut = new Outlet<MatrixWorld>();
 [Val("Sediment", "Outlet")] public readonly Outlet<MatrixWorld> sedimentOut = new Outlet<MatrixWorld>(
 public IEnumerable<IOutlet<object>> Outlets()  { yield return cliffOut; yield return sedimentOut; }
```

```
[Val("Cliff")] public float cliffIntensity = 1f;
[Val("Sediment")] public float sedimentIntensity = 1f;



public override void Generate (TileData data, StopToken stop)
{
 MatrixWorld orig = data.ReadInletProduct(origIn);
 MatrixWorld eroded = data.ReadInletProduct(erodedIn);
 if (orig == null || eroded == null) return;

 MatrixWorld cliff = new MatrixWorld(orig.rect, orig.worldPos, orig.worldSize);
 MatrixWorld sediment = new MatrixWorld(orig.rect, orig.worldPos, orig.worldSize);
  if (stop!=null && stop.stop) return;
 CliffSediment(orig, eroded, cliff, sediment);
  if (stop!=null && stop.stop) return;

 data.StoreProduct(cliffOut, cliff);
 data.StoreProduct(sedimentOut, sediment);
}



public void CliffSediment (MatrixWorld orig, MatrixWorld eroded, MatrixWorld cliff, MatrixWorld sediment)
///Painting with cliff or sediment depending on the erosion change (delta) value
///Determining whether it's sediment or cliff by comparing original and eroded inclines (if more inclined - th
{
 MatrixWorld origInclines = cliff;  //writing incline to temporary matrices. Will be overwritten anyways
```

```
  MatrixOps.Delta(orig, origInclines);

  MatrixWorld erodedInclines = sediment;
  MatrixOps.Delta(eroded, erodedInclines);

  for (int i=0; i<orig.count; i++)
  {
   float heightDelta = orig.arr[i] - eroded.arr[i];
   if (heightDelta < 0) heightDelta = 0;

   float inclineDelta = erodedInclines.arr[i] - origInclines.arr[i];

   if (inclineDelta > 0) //if incline increased - using cliff
   {
    cliff.arr[i] = inclineDelta*1000*cliffIntensity/orig.PixelSize.x;
    sediment.arr[i] = 0;
   }

   else //if lowered - using sediment
   {
    sediment.arr[i] = heightDelta*1000*sedimentIntensity/orig.PixelSize.x; //sediment takes delta
    cliff.arr[i] = 0;
   }
  }
 }
}
```

```
[Serializable]
[GeneratorMenu(
 menu = "Map/Modifiers",
 name = "Parallax",
 section=2,
 colorType = typeof(MatrixWorld),
 iconName="GeneratorIcons/Parallax",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Parallax")]
public class Parallax210 : Generator, IInlet<MatrixWorld>, IMultiInlet, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Intensity X", "Inlet")] public readonly Inlet<MatrixWorld> intensityInX = new Inlet<MatrixWorld>();
 [Val("Intensity Z", "Inlet")] public readonly Inlet<MatrixWorld> intensityInZ = new Inlet<MatrixWorld>();
 public virtual IEnumerable<IInlet<object>> Inlets () { yield return intensityInX; yield return intensityInZ; }

 [Val("Offset")] public Vector2D offset;
 public enum Interpolation { None, Always, OnTransitions }
 [Val("Interpolation")] public Interpolation interpolation = Interpolation.Always;

 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
  MatrixWorld map = data.ReadInletProduct(this);
```

```
   if (map == null) return;
   if (!enabled) { data.StoreProduct(this,map); return; }

   MatrixWorld intensityX = data.ReadInletProduct(intensityInX);
   MatrixWorld intensityZ = data.ReadInletProduct(intensityInZ);

   if (stop!=null && stop.stop) return;
   Vector2D pixelDir = offset / map.PixelSize;
   MatrixWorld result = new MatrixWorld(map);
   result.Parallax(pixelDir, map, intensityX, intensityZ, (int)interpolation);

   if (stop!=null && stop.stop) return;
   data.StoreProduct(this, result);
  }
 }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using MapMagic.Products;

namespace MapMagic.Nodes.MatrixGenerators
{

 /*[System.Serializable]
 [GeneratorMenu (menu="Map/Input", name ="Textures In", section=1, disengageable = true)]
 public class TexturesInput : Generator, IOutlet<MatrixWorld>, ITerrainReader
 {
  [Val(name="Channel")] public int channel = 0;

  public void CheckReadTerrain (Terrain terrain, Results results)
  {
   if (results.terrainReads.ContainsKey(typeof(SplatData))) return; //already read

   SplatData data = new SplatData();
   data.ReadFromTerrain(terrain);
   results.terrainReads.Add(typeof(SplatData), data);
  }
```

```
public override void Generate (Results results, Area area, int seed, StopCallback stop)
{
 if (!enabled) { results.SetProduct(this, null); return; }  //should set anything to mark as generated

 SplatData data = null;
 if (results.terrainReads.ContainsKey(typeof(SplatData))) data = (SplatData)results.terrainReads[typeof(S
 if (data==null) { results.SetProduct(this, null); return; }

 if (stop!=null && stop(0)) return;

 MatrixWorld matrix = new MatrixWorld(area.full.resolution, area.full.position, area.full.size);
 Floats3DtoMatrix(data.splats3D, channel, matrix, area);

 if (stop!=null && stop(0)) return;
 results.SetProduct(this, matrix);
}

public void Floats3DtoMatrix (float[,,] splats3D, int channel, Matrix matrix, Area area)
{
 int splatsResolution = splats3D.GetLength(0);
 int margins = area.Margins;

 //simple case if resolution match
 if (area.active.resolution == splatsResolution)
 {
  for (int x=0; x<splatsResolution; x++)
```

```
    for (int z=0; z<splatsResolution; z++)
    {
     float val = splats3D[z,x, channel];
     matrix.array[(z+margins)*matrix.rect.size.x + x+margins] = val; //do not use matrix[x,z] since x/z are 0-b
    }

   //TODO: fill margins
  }

  //interpolated if resolution doesn't match
  else
  {
   Matrix tmpMatrix = new Matrix( new Coord(0,0), new Coord(splatsResolution,splatsResolution) );

   for (int x=0; x<splatsResolution; x++)
    for (int z=0; z<splatsResolution; z++)
     tmpMatrix.array[z*splatsResolution + x] = splats3D[z,x, channel];

   Debug.Log("could not read heightmap - resolutions mismatch");

   //TODO: interpolated
  }
 }
}*/

}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Products;
using MapMagic.Terrains;

using UnityEngine.Profiling;

namespace MapMagic.Nodes.MatrixGenerators {
 [System.Serializable]
 public class BaseTextureLayer : IInlet<MatrixWorld>, IOutlet<MatrixWorld>
 {
  public string name = "Layer";
  public int channelNum = 0; //for RTP, CTS and custom

  [SerializeField] private float opacity = 1;
  public float Opacity { get=>opacity; set=>opacity=value; }

  public Generator Gen { get { return gen; } private set { gen = value;} }
  public Generator gen; //property is not serialized
  public void SetGen (Generator gen) => this.gen=gen;
```

```
  public ulong id; //properties not serialized
  public ulong Id { get{return id;} set{id=value;} }
  public ulong LinkedOutletId { get; set; }  //if it's inlet. Assigned every before each clear or generate
  public ulong LinkedGenId { get; set; }

  public IUnit ShallowCopy() => (BaseTextureLayer)this.MemberwiseClone();
 }



 [System.Serializable]
 public abstract class BaseTexturesOutput<L> : OutputGenerator, IMultiLayer, IMultiInlet, IMultiOutlet  whe
 {
  public OutputLevel outputLevel = OutputLevel.Draft | OutputLevel.Main;
  public override OutputLevel OutputLevel { get{ return outputLevel; } }


  public L[] layers = new L[0];
  public IList<IUnit> Layers { get => layers; set => layers=ArrayTools.Convert<L,IUnit>(value); }
//  public virtual void SetLayers(object[] ls) => layers = Array.ConvertAll(ls, i=>(L)i);
  public virtual bool Inversed => true;
  public virtual bool HideFirst => true; //not for all - direct matrices and textures do not hide first layer

  public IEnumerable<IInlet<object>> Inlets()
  {
   for (int i=0; i<layers.Length; i++)
```

```
  yield return layers[i];
}

public IEnumerable<IOutlet<object>> Outlets()
{
 for (int i=0; i<layers.Length; i++)
  yield return layers[i];
}

public MatrixWorld[] BaseGenerate (TileData data, StopToken stop)
/// Reads inlets, normalizes, writes outputs
/// But not sending to finalize
{
 if (layers.Length == 0) return null;

 //reading/copying products
 MatrixWorld[] dstMatrices = new MatrixWorld[layers.Length];
 float[] opacities = new float[layers.Length];

 if (stop!=null && stop.stop) return null;
 for (int i=0; i<layers.Length; i++)
 {
  if (stop!=null && stop.stop) return null;

  MatrixWorld srcMatrix = data.ReadInletProduct(layers[i]);
  if (srcMatrix != null) dstMatrices[i] = new MatrixWorld(srcMatrix);
```

```
   else dstMatrices[i] = new MatrixWorld(data.area.full.rect, (Vector3)data.area.full.worldPos, (Vector3)data

    opacities[i] = layers[i].Opacity;
  }

  //normalizing
  if (stop!=null && stop.stop) return null;
  dstMatrices.FillNulls(() => new MatrixWorld(data.area.full.rect, (Vector3)data.area.full.worldPos, (Vector3
  dstMatrices[0].Fill(1);
  Matrix.BlendLayers(dstMatrices, opacities);

  //saving products
  if (stop!=null && stop.stop) return null;
  for (int i=0; i<layers.Length; i++)
    data.StoreProduct(layers[i], dstMatrices[i]);

  return dstMatrices;
 }

 public abstract FinalizeAction FinalizeAction { get; }

 //public abstract void Purge (TileData data, Terrain terrain);
}

[System.Serializable]
```

```
[GeneratorMenu(
 menu = "Map/Output",
 name = "Textures",
 section =2,
 drawButtons = false,
 colorType = typeof(MatrixWorld),
 iconName="GeneratorIcons/TexturesOut",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Textures")]
public class TexturesOutput200 : BaseTexturesOutput<TexturesOutput200.TextureLayer>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 public class TextureLayer : BaseTextureLayer
 {
  [Val("Layer", cat:"Layer")] public TerrainLayer prototype; // = new TerrainLayer() {  tileSize=new Vector2(

  public Color color = new Color(0.75f, 0.75f, 0.75f, 1);

  public bool guiProperties;
  public bool guiRemapping;
  public bool guiTileSettings;
 }

 [SerializeField] public int guiExpanded;
```

```
public override void Generate (TileData data, StopToken stop)
{
 Log.Add("Textures");

 //generating
 MatrixWorld[] dstMatrices = BaseGenerate(data, stop);

 //adding to finalize
 if (stop!=null && stop.stop) return;
 if (enabled)
 {
  for (int i=0; i<layers.Length; i++)
   data.StoreOutput(layers[i], typeof(TexturesOutput200), layers[i].prototype,  dstMatrices[i]);
  data.MarkFinalize(Finalize, stop);
 }
 else
  data.RemoveFinalize(finalizeAction);
}

public override FinalizeAction FinalizeAction => finalizeAction; //should return variable, not create new
public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 //preparing arrays
 if (stop!=null && stop.stop) return;
 data.GatherOutputs (typeof(TexturesOutput200),
```

```
 out TerrainLayer[] prototypes,
 out MatrixWorld[] matrices,
 out MatrixWorld[] masks,
 inSubs:true);

//creating splats and prototypes arrays
if (stop!=null && stop.stop) return;
float[,,] splats3D = BlendLayers(matrices, masks, data.area, stop:stop);

//pushing to apply
if (stop!=null && stop.stop) return;
ApplyData applyData = new ApplyData() { splats=splats3D, prototypes=prototypes };
Graph.OnOutputFinalized?.Invoke(typeof(TexturesOutput200), data, applyData, stop);
data.MarkApply(applyData);

#if MM_DEBUG
Log.Add("TexturesOut Finalized");
#endif
}

public static float[,,] BlendLayers (IList<Matrix> matrices, IList<Matrix> masks, Area area, IList<int> chann
/// If channelNumbers are not provided - blending Texture Output style. If provided - blending MicroSplat s
{
 int fullSize = area.full.rect.size.x;
 int activeSize = area.active.rect.size.x;
 int margins = area.Margins;
```

```
int count;
if (channelNumbers != null)
{
 int maxChannelNum = 0;
 foreach (int chNum in channelNumbers)
  if (chNum > maxChannelNum) maxChannelNum=chNum;

 count = maxChannelNum + 1;
}
else count = matrices.Count;

float[,,] splats3D = new float[activeSize, activeSize, count];

for (int x=0; x<activeSize; x++)
{
 if (stop!=null && stop.stop) return null;
 for (int z=0; z<activeSize; z++)
 {
  //int pos = (z+margins-area.full.rect.offset.z)*area.full.rect.size.x + x+margins - area.full.rect.offset.x;
  int pos = area.full.rect.GetPos(x+area.full.rect.offset.x+margins, z+area.full.rect.offset.z+margins);

  float sum = 0;
  for (int i=0; i<count; i++)
  {
   float val = matrices[i]!=null ? matrices[i].arr[pos] : 0;
```

```
    val *= masks[i]==null ? 1 : masks[i].arr[pos];
    sum += val;
   }

   if (sum != 0)
    for (int i=0; i<count; i++)
   {
    float val = matrices[i]!=null ? matrices[i].arr[pos] : 0;
    val *= masks[i]==null ? 1 : masks[i].arr[pos];
    val /= sum;

    if (val < 0) val = 0; if (val > 1) val = 1;

    int chNum;
    if (channelNumbers != null) chNum = channelNumbers[i];
    else chNum = i;

    splats3D[z,x,chNum] += val;
   }
  }
 }

 return splats3D;
}

public class ApplyData : IApplyData
```

```
{
 public float[,,] splats;
 public TerrainLayer[] prototypes;

 public virtual void Apply (Terrain terrain)
 {
  Profiler.BeginSample("Apply Textures " + terrain.transform.name);

  if (terrain==null || terrain.Equals(null) || terrain.terrainData==null) return; //chunk removed during apply
  TerrainData data = terrain.terrainData;

  //setting resolution
  int size = splats.GetLength(0);
  if (data.alphamapResolution != size) data.alphamapResolution = size;

  terrain.terrainData.terrainLayers = prototypes; //in 2017 seems that alphamaps should go first
  terrain.terrainData.SetAlphamaps(0,0,splats);

  Profiler.EndSample();

  #if MM_DEBUG
  Log.Add("TexturesOut Applied");
  #endif
 }

 public static ApplyData Empty
```

```
  {get{
   return new ApplyData() {
    splats = new float[64,64,0],
    prototypes = new TerrainLayer[0] };
  }}

  public int Resolution
  {get{
   if (splats==null) return 0;
   else return splats.GetLength(0);
  }}
 }

 public override void ClearApplied (TileData data, Terrain terrain)
 {
  TerrainData terrainData = terrain.terrainData;
  terrainData.terrainLayers = new TerrainLayer[0];
  terrainData.alphamapResolution = 32;
 }
}


[System.Serializable]
[GeneratorMenu(
 menu = "Map/Output",
 name = "Custom Material",
```

```
 section =2,
 drawButtons = false,
 colorType = typeof(MatrixWorld),
 iconName="GeneratorIcons/TexturesOut",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Textures")]
public class CustomShaderOutput200 : BaseTexturesOutput<CustomShaderOutput200.CustomShaderLa
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on


 public class CustomShaderLayer : BaseTextureLayer { } //inheriting empty class just to draw it's editor


 public static string[] controlTextureNames = new string[] { "_ControlTexture1" };


 public static string[] controlTexturePossibleNames = new string[] { "_ControlTexture1", "_ControlTexture2"
  "_ControlTexture3", "_ControlTexture4", "_ControlTexture5", "_ControlTexture6", "_ControlTexture7",
  "_ControlTexture8", "_ControlTexture9", "_ControlTexture10", "_ControlTexture11", "_ControlTexture12"


 public override void Generate (TileData data, StopToken stop)
 {
  //generating
  MatrixWorld[] dstMatrices = BaseGenerate(data, stop);

  //adding to finalize
  if (stop!=null && stop.stop) return;
  if (enabled)
```

```
 {
  for (int i=0; i<layers.Length; i++)
   data.StoreOutput(layers[i], typeof(CustomShaderOutput200), layers[i],  dstMatrices[i]);
  data.MarkFinalize(Finalize, stop);
 }
 else
  data.RemoveFinalize(finalizeAction);
}



public override FinalizeAction FinalizeAction => finalizeAction; //should return variable, not create new
public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 //preparing arrays
 if (stop!=null && stop.stop) return;
 data.GatherOutputs (typeof(CustomShaderOutput200),
  out (int chNum, float opacity)[] prototypes,
  out MatrixWorld[] matrices,
  out MatrixWorld[] masks,
  inSubs:true);
 float[] opacities = prototypes.Select(p => p.opacity);
 int[] chNums = prototypes.Select(p => p.chNum);

 //purging if no outputs
 if (matrices.Length == 0)
```

```
 {
  if (stop!=null && stop.stop) return;
  data.MarkApply(ApplyData.Empty);
  return;
 }

 //creating control textures contents
 Color[][] colors = BlendMatrices(data.area.active.rect, matrices, masks, opacities, chNums, normalize:tru

 //pushing to apply
 if (stop!=null && stop.stop) return;
 var controlTexturesData = new ApplyData() {
  textureColors = colors,
  textureFormat = TextureFormat.RGBA32,
  textureBaseMapDistance = 10000000, //no base map
  textureNames = (string[])controlTextureNames.Clone() };

 Graph.OnOutputFinalized?.Invoke(typeof(CustomShaderOutput200), data, controlTexturesData, stop);
 data.MarkApply(controlTexturesData);
}


public static Color[][] BlendMatrices (CoordRect colorsRect, IList<Matrix> matrices, IList<Matrix> biomeM
/// Reads matrices and fills normalized values to colors using masks
/// TODO: use raw texture bytes
/// TODO: bring to matrix
```

```
{
 int texturesCount;
 int maxChannelNum = 0;
 foreach (int chNum in channelNums)
  if (chNum > maxChannelNum) maxChannelNum=chNum;
 texturesCount = maxChannelNum/4 + 1;

 Color[][] colors = new Color[texturesCount][];

 int matrixCount = matrices.Count;

 //getting matrices rect
 CoordRect matrixRect = new CoordRect(0,0,0,0);
 for (int m=0; m<matrixCount; m++)
  if (matrices[m] != null) matrixRect = matrices[m].rect;

 //checking rect
 for (int m=0; m<matrixCount; m++)
  if (matrices[m] != null  &&  matrices[m].rect != matrixRect)
   throw new Exception("MapMagic: Matrix rect mismatch");
 for (int b=0; b<matrixCount; b++)
  if (biomeMasks[b] != null  &&  biomeMasks[b].rect != matrixRect)
   throw new Exception("MapMagic: Biome matrix rect mismatch");

 //preparing row re-use array
 float[] values = new float[texturesCount*4];
```

```
//blending
for (int x=0; x<colorsRect.size.x; x++)
 for (int z=0; z<colorsRect.size.z; z++)
 {
  int matrixPosX = colorsRect.offset.x + x;
  int matrixPosZ = colorsRect.offset.z + z;
  int matrixPos = (matrixPosZ-matrixRect.offset.z)*matrixRect.size.x + matrixPosX - matrixRect.offset.x;

  int colorsPos = z*colorsRect.size.x + x; //(z-colorsRect.offset.z)*colorsRect.size.x + x - colorsRect.offset

  float sum = 0;

  //resetting values
  for (int m=0; m<values.Length; m++)
   values[m] = 0;

  //getting values
  for (int m=0; m<matrixCount; m++)
  {
   Matrix matrix = matrices[m];
   if (matrix == null)
    continue;

   float val = matrix.arr[matrixPos];
```

```
 //multiply with biome
 Matrix biomeMask = biomeMasks[m];
 if (biomeMask != null) //no empty biomes in list (so no mask == root biome)
  val *= biomeMask.arr[matrixPos]; //if mask is not assigned biome was ignored, so only main outs with

 //clamp
 if (val < 0) val = 0; if (val > 1) val = 1;

 sum += val;
 values[channelNums[m]] += val;
}

//normalizing and writing to colors
for (int m=0; m<values.Length; m++)
{
 float val = values[m];

 if (normalize) val = sum!=0 ? val/sum : 0;

 int texNum = m / 4;
 int chNum = m % 4;

 if (colors[texNum] == null) colors[texNum] = new Color[colorsRect.size.x*colorsRect.size.z];

 switch (chNum)
 {
```

```
    case 0: colors[texNum][colorsPos].r += val; break;
    case 1: colors[texNum][colorsPos].g += val; break;
    case 2: colors[texNum][colorsPos].b += val; break;
    case 3: colors[texNum][colorsPos].a += val; break;
   }
  }
 }

 return colors;
}



public static Color[] MatricesToColors (CoordRect colorsRect, Matrix rMatrix, Matrix gMatrix, Matrix bMatr
/// Just creates a texture from matrices without blending
{
 CoordRect matrixRect = rMatrix.rect;
 Color[] colors = new Color[colorsRect.size.x*colorsRect.size.z];
 Color color = new Color();

 Coord min = colorsRect.Min; Coord max = colorsRect.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
 {
  int matrixPos = (z-matrixRect.offset.z)*matrixRect.size.x + x - matrixRect.offset.x;
  int colorsPos =  (z-colorsRect.offset.z)*colorsRect.size.x + x - colorsRect.offset.x;
```

```
  color.r = rMatrix.arr[matrixPos];
  if (gMatrix != null) color.g = gMatrix.arr[matrixPos];
  if (bMatrix != null) color.b = bMatrix.arr[matrixPos];
  if (aMatrix != null) color.a = aMatrix.arr[matrixPos];

  colors[colorsPos] = color;
 }

 return colors;
}



public class ApplyData : IApplyData
{
 public Color[][] textureColors; // TODO: use raw texture bytes
 public string[] textureNames;
 public string[] altTextureNames= null; //to let MicroSplat work with _Control0 and _CustomControl0
 public TextureFormat textureFormat;
 public float textureBaseMapDistance; //most custom shaders change the base distance using their profile



 public virtual void Apply (Terrain terrain)
 {
  if (textureColors==null) return;
  int numTextures = textureColors.Length;
```

```
if (numTextures==0) return;
int resolution = (int)Mathf.Sqrt(textureColors[0].Length);

//MaterialPropertyBlock matProps = new MaterialPropertyBlock();

//assigning material props via MaterialPropertySerializer to make them serializable
MaterialPropertySerializer matPropSerializer = terrain.GetComponent<MaterialPropertySerializer>();
if (matPropSerializer == null)
 matPropSerializer = terrain.gameObject.AddComponent<MaterialPropertySerializer>();


for (int i=0; i<textureColors.Length; i++)
{
 if (textureColors[i] == null) continue;

 string texName = null;
 if (i<textureNames.Length) texName = textureNames[i];

 Texture2D tex = matPropSerializer.GetTexture(textureNames[i]);
 if (tex==null || tex.width!=resolution || tex.height!=resolution || tex.format!=textureFormat)
 {
  if (tex!=null)
  {
   #if UNITY_EDITOR
   if (!UnityEditor.AssetDatabase.Contains(tex))
   #endif
```

```
    GameObject.DestroyImmediate(tex);
   }

   tex = new Texture2D(resolution, resolution, textureFormat, false, true);
   tex.name = texName;
   tex.wrapMode = TextureWrapMode.Mirror; //to avoid border seams
   //tex.hideFlags = HideFlags.DontSave;
   //tex.filterMode = FilterMode.Point;

   matPropSerializer.SetTexture(textureNames[i], tex);
  }

  tex.SetPixels(0,0,tex.width,tex.height,textureColors[i]);
  tex.Apply();

  //if (texName != null) matPropSerializer.SetTexture(texName, tex);
  if (texName != null) terrain.materialTemplate.SetTexture(texName, tex);
 }

 matPropSerializer.Apply();

 terrain.basemapDistance = textureBaseMapDistance;
}

public static ApplyData Empty
{get{
```

```
   return new ApplyData() {
    textureColors = new Color[0][],
    textureNames = new string[0]  };
  }}

  public int Resolution
  {get{
   if (textureColors.Length==0) return 0;
   else return (int)Mathf.Sqrt(textureColors[0].Length);
  }}
 }

 public override void ClearApplied (TileData data, Terrain terrain)
 {

 }
}

[System.Serializable]
[GeneratorMenu(
 menu = "Map/Output",
 name = "Direct Textures",
 section =2,
 drawButtons = false,
 colorType = typeof(MatrixWorld),
```

```
 iconName="GeneratorIcons/TexturesOut",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Textures")]
public class DirectTexturesOutput200 : BaseTexturesOutput<DirectTexturesOutput200.DirectTexturesLay
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on


 public class DirectTexturesLayer : BaseTextureLayer { } //inheriting empty class just to draw it's editor


 public override bool HideFirst => false;


 public override void Generate (TileData data, StopToken stop)
 {
  //reading products
  MatrixWorld[] matrices = new MatrixWorld[layers.Length];
  for (int i=0; i<layers.Length; i++)
  {
   if (stop!=null && stop.stop) return;
   matrices[i] = data.ReadInletProduct(layers[i]);
  }


  //adding to finalize
  if (stop!=null && stop.stop) return;
  if (enabled)
  {
   for (int i=0; i<layers.Length; i++)
    data.StoreOutput(layers[i], typeof(DirectTexturesOutput200), (layers[i].name, layers[i].channelNum, laye
```

```
  data.MarkFinalize(Finalize, stop);
 }
 else
  data.RemoveFinalize(finalizeAction);
}



public override FinalizeAction FinalizeAction => finalizeAction; //should return variable, not create new
public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 //preparing arrays
 if (stop!=null && stop.stop) return;
 data.GatherOutputs (typeof(DirectTexturesOutput200),
  out (string name, int chNum, float opacity)[] prototypes,
  out MatrixWorld[] matrices,
  out MatrixWorld[] masks,
  inSubs:true);
 string[] names = prototypes.Select(p => p.name);
 float[] opacities = prototypes.Select(p => p.opacity);
 int[] chIndexes = prototypes.Select(p => p.chNum);

 //purging if no outputs
 if (matrices.Length == 0)
 {
  if (stop!=null && stop.stop) return;
```

```
 data.MarkApply(ApplyData.Empty);
 return;
}

//calculating number of textures, creating name->textureNum lut
Dictionary<string,int> nameToNum = new Dictionary<string, int>();
foreach (string name in names)
 if (!nameToNum.ContainsKey(name))
  nameToNum.Add(name, nameToNum.Count);

//creating control textures contents
Color[][] colors = new Color[nameToNum.Count][];
string[] colorNames = new string[nameToNum.Count]; //texture names, in order corresponding to colors
for (int m=0; m<matrices.Length; m++)
{
 int textureNum = nameToNum[names[m]];

 colorNames[textureNum] = names[m];

 if (matrices[m] != null)
 {
  if (colors[textureNum] == null)
   colors[textureNum] = new Color[data.area.active.rect.size.x * data.area.active.rect.size.z];

  matrices[m].ExportColors(colors[textureNum], data.area.active.rect.offset, data.area.active.rect.size, ch
 }
```

```
 }

 //pushing to apply
 if (stop!=null && stop.stop) return;
 var controlTexturesData = new ApplyData() {
  textureColors = colors,
  textureNames = colorNames,
  textureFormat = TextureFormat.RGBA32 };

 Graph.OnOutputFinalized?.Invoke(typeof(DirectTexturesOutput200), data, controlTexturesData, stop);
 data.MarkApply(controlTexturesData);
}



public class ApplyData : IApplyData
{
 public Color[][] textureColors;
 public string[] textureNames;
 public TextureFormat textureFormat;

 public virtual void Apply (Terrain terrain)
 {
  if (textureColors==null  ||  textureColors.Length==0  ||  textureColors.AllNull()) return;
  int resolution = (int)Mathf.Sqrt(textureColors.Any().Length);

  DirectTexturesHolder holder = terrain.GetComponent<DirectTexturesHolder>();
```

```
if (holder == null)
 holder = terrain.gameObject.AddComponent<DirectTexturesHolder>();

//preparing textures
DictionaryOrdered<string,Texture2D> newDict = new DictionaryOrdered<string,Texture2D>(textureNam
newDict.TakeMatchingValuesFrom(holder.textures);

for (int i=0; i<textureColors.Length; i++)
{
 if (textureColors[i] == null) continue;

 string texName = textureNames[i];
 Texture2D tex = newDict[texName];

 CheckTexture(ref tex, resolution, textureFormat);
 tex.name = textureNames[i];
 tex.wrapMode = TextureWrapMode.Mirror; //to avoid border seams

 tex.SetPixels(0,0,tex.width,tex.height,textureColors[i]);
 tex.Apply();

 newDict[texName] = tex; //it could be created from null
}

holder.textures = newDict;
holder.position = (Vector2D)terrain.transform.position;
```

```
 holder.size = (Vector2D)terrain.terrainData.size;
}



public static void CheckTexture (ref Texture2D tex, int resolution, TextureFormat format)
///Checks if texture has this resolution and format, and if not removes it and creates a new one
{
 if (tex==null)
 {
  tex = new Texture2D(resolution, resolution, format, false, true);
  return;
 }

 if (tex.width!=resolution || tex.height!=resolution || tex.format!=format)
 {
  #if UNITY_EDITOR
  if (!UnityEditor.AssetDatabase.Contains(tex))
  #endif
   GameObject.DestroyImmediate(tex);

  tex = new Texture2D(resolution, resolution, format, false, true);
 }
}



public static ApplyData Empty
```

```
 {get{
  return new ApplyData() {
   textureColors = new Color[0][],
   textureNames = new string[0]  };
 }}

 public int Resolution
 {get{
  if (textureColors.Length==0) return 0;
  else return (int)Mathf.Sqrt(textureColors[0].Length);
 }}
}

 public override void ClearApplied (TileData data, Terrain terrain)
 {

 }
}


[System.Serializable]
[GeneratorMenu(
 menu = "Map/Output",
 name = "DirectMatrices",
 section =2,
 drawButtons = false,
```

```
 colorType = typeof(MatrixWorld),
 iconName="GeneratorIcons/TexturesOut",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Textures")]
public class DirectMatricesOutput200 : BaseTexturesOutput<DirectMatricesOutput200.DirectMatricesLaye
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 public class DirectMatricesLayer : BaseTextureLayer { } //inheriting empty class just to draw it's editor

 public override bool HideFirst => false;

 public override void Generate (TileData data, StopToken stop)
 {
  //reading products
  MatrixWorld[] matrices = new MatrixWorld[layers.Length];
  for (int i=0; i<layers.Length; i++)
  {
   if (stop!=null && stop.stop) return;
   matrices[i] = data.ReadInletProduct(layers[i]);
  }

  //adding to finalize
  if (stop!=null && stop.stop) return;
  if (enabled)
  {
   for (int i=0; i<layers.Length; i++)
```

```
   data.StoreOutput(layers[i], typeof(DirectMatricesOutput200), layers[i].name, matrices[i]);
  data.MarkFinalize(Finalize, stop);
 }
 else
  data.RemoveFinalize(finalizeAction);
}



public override FinalizeAction FinalizeAction => finalizeAction; //should return variable, not create new
public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 //preparing arrays
 if (stop!=null && stop.stop) return;
 data.GatherOutputs (typeof(DirectMatricesOutput200),
  out string[] names,
  out MatrixWorld[] matrices,
  out MatrixWorld[] masks,
  inSubs:true);

 //purging if no outputs
 if (matrices.Length == 0)
 {
  if (stop!=null && stop.stop) return;
  data.MarkApply(ApplyData.Empty);
  return;
```

```
}

//writing dict
DictionaryOrdered<string,MatrixWorld> dict = new DictionaryOrdered<string,MatrixWorld>();
for (int m=0; m<matrices.Length; m++)
{
 if (matrices[m] == null)
  continue;

 MatrixWorld matrixCopy = new MatrixWorld(matrices[m]); //to multyply with biome, and to keep it indepe

 if (masks[m] != null)
  matrixCopy.Multiply(masks[m]);

 matrixCopy.Crop(data.area.active.rect);
 Vector2D pixelSize = matrixCopy.PixelSize;
 matrixCopy.worldPos = (Vector3)data.area.active.worldPos;
 matrixCopy.worldSize = (Vector3)data.area.active.worldSize;
 matrixCopy.worldSize.y = matrices[m].worldSize.y;

 if (dict.ContainsKey(names[m])) dict[names[m]] = matrixCopy;
 else dict.Add(names[m], matrixCopy);
}

//pushing to apply
if (stop!=null && stop.stop) return;
```

```
 var controlTexturesData = new ApplyData() {dict = dict};

 Graph.OnOutputFinalized?.Invoke(typeof(DirectMatricesOutput200), data, controlTexturesData, stop);
 data.MarkApply(controlTexturesData);
}



public class ApplyData : IApplyData
{
 public DictionaryOrdered<string,MatrixWorld> dict;

 public virtual void Apply (Terrain terrain)
 {
  DirectMatricesHolder holder = terrain.GetComponent<DirectMatricesHolder>();
  if (holder == null)
   holder = terrain.gameObject.AddComponent<DirectMatricesHolder>();

  holder.maps = dict;
 }


 public static ApplyData Empty
 {
  get => new ApplyData() { dict = new DictionaryOrdered<string,MatrixWorld>() };
 }
```

```
  public int Resolution
  {get{
   if (dict.Count==0) return 0;
   else return dict[0].rect.size.x;
  }}
 }

 public override void ClearApplied (TileData data, Terrain terrain)
 {

 }
}

}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes.GUI;
using MapMagic.Nodes.MatrixSetsGenerators;

namespace MapMagic.Nodes.GUI
{
 public static class MatrixSetsEditors
 {

  [Draw.Editor(typeof(Add210))]
  public static void DrawAddGen (Add210 gen)
  {
   using (Cell.Padded(1,1,0,0))
   {
    Cell.current.fieldWidth = 0.6f;
```

```
using (Cell.RowPx(35))
 using (Cell.LinePx(35))
  using (Cell.Padded(4,2,4,2))
 {
  Texture2D tex;
  if (gen.prototypeType == PrototypeType.TerrainLayer  &&  gen.terrainLayer != null)
   tex = gen.terrainLayer.diffuseTexture;
  if (gen.prototypeType == PrototypeType.GrassTexture  &&  gen.texture != null)
   tex = gen.texture;
  else
   tex = UI.current.textures.GetTexture("DPUI/Backgrounds/Empty");

  Draw.TextureIcon(tex);
 }

using (Cell.Row)
{
 Cell.current.fieldWidth = 0.58f;

 using (Cell.LineStd) Draw.Field(ref gen.prototypeType, "Type");

 using (Cell.LineStd)
 {
  if (gen.prototypeType == PrototypeType.TerrainLayer)
  {
   Draw.ObjectField(ref gen.terrainLayer, "Layer");
```

```
  Cell.current.Expose(gen.id, "terrainLayer", typeof(TerrainLayer));
 }

 else if (gen.prototypeType == PrototypeType.GrassTexture)
 {
  Draw.ObjectField(ref gen.texture, "Texture");
  Cell.current.Expose(gen.id, "texture", typeof(Texture2D));
 }

 else
 {
  Draw.ObjectField(ref gen.prefab, "Prefab");
  Cell.current.Expose(gen.id, "prefab", typeof(GameObject));
 }
}

using (Cell.LineStd)
{
 Draw.Field(ref gen.instanceNum, "Num");
 Cell.current.Expose(gen.id, "instanceNum", typeof(int));
}

using (Cell.LineStd)
{
 Draw.Field(ref gen.mode, "Mode");
 Cell.current.Expose(gen.id, "mode", typeof(int));
```

```
   }

   using (Cell.LineStd)
   {
    Draw.Field(ref gen.opacity, "Opacity");
    Cell.current.Expose(gen.id, "opacity", typeof(float));
   }

   /*using (Cell.LineStd)
   {
    Draw.ToggleLeft(ref gen.normalize, "Normalize");
    Draw.AddFieldToCellObj(typeof(Add210), "normalize");
   }*/
  }
 }
}


[Draw.Editor(typeof(Pick210))]
public static void DrawPickGen (Pick210 gen)
{
 using (Cell.Padded(1,1,0,0))
 {
  Cell.current.fieldWidth = 0.57f;

  using (Cell.RowPx(35))
```

```
 using (Cell.LinePx(35))
  using (Cell.Padded(4,2,4,2))
 {
  Texture2D tex;
  if (gen.prototypeType == PrototypeType.TerrainLayer  &&  gen.terrainLayer != null)
   tex = gen.terrainLayer.diffuseTexture;
  if (gen.prototypeType == PrototypeType.GrassTexture  &&  gen.texture != null)
   tex = gen.texture;
  else
   tex = UI.current.textures.GetTexture("DPUI/Backgrounds/Empty");

  Draw.TextureIcon(tex);
 }

using (Cell.Row)
{
 using (Cell.LineStd) Draw.Field(ref gen.prototypeType, "Type");

 using (Cell.LineStd)
 {
  if (gen.prototypeType == PrototypeType.TerrainLayer)
  {
   Draw.ObjectField(ref gen.terrainLayer, "Layer");
   Cell.current.Expose(gen.id, "terrainLayer", typeof(TerrainLayer));
  }
```

```
 else if (gen.prototypeType == PrototypeType.GrassTexture)
 {
  Draw.ObjectField(ref gen.texture, "Texture");
  Cell.current.Expose(gen.id, "texture", typeof(Texture2D));
 }

 else
 {
  Draw.ObjectField(ref gen.prefab, "Prefab");
  Cell.current.Expose(gen.id, "prefab", typeof(GameObject));
 }
}

using (Cell.LineStd)
{
 Draw.Field(ref gen.instanceNum, "Num");
 Cell.current.Expose(gen.id, "instanceNum", typeof(int));
}

using (Cell.LineStd)
{
 Draw.ToggleLeft(ref gen.createIfNotExists, "Create if Empty");
 Cell.current.Expose(gen.id, "createIfNotExists", typeof(bool));
 Draw.AddFieldToCellObj(typeof(Pick210), "createIfNotExists");
}
```

```
   using (Cell.LineStd)
   {
    Draw.Field(ref gen.opacity, "Opacity");
    Cell.current.Expose(gen.id, "opacity", typeof(float));
   }
  }
 }
}


[Draw.Editor(typeof(Create210))]
public static void DrawAssembleGen (Create210 gen)
{
 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlinl
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito

 Cell.EmptyLinePx(2);

 using (Cell.LineStd)
  using (Cell.Padded(1,1,0,0))
   Draw.Field(ref gen.normalize, "Normalize");

 Cell.EmptyLinePx(2);
}

private static void DrawAssembleLayer (Generator tgen, int num)
```

```
{
 Create210 gen = (Create210)tgen;
 Create210.Layer layer = gen.layers[num];

 if (layer == null) return;

 using (Cell.LinePx(20))
 {
  Cell.current.fieldWidth = 0.52f;

  using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, gen);

  Cell.EmptyRowPx(7);

  using (Cell.RowPx(25))
   using (Cell.LinePx(25+2))
    using (Cell.Padded(2,2,4,2))
    {
     Texture2D tex;
     if (layer.prototypeType == PrototypeType.TerrainLayer  &&  layer.terrainLayer != null)
      tex = layer.terrainLayer.diffuseTexture;
     if (layer.prototypeType == PrototypeType.GrassTexture  &&  layer.texture != null)
      tex = layer.texture;
     else
      tex = UI.current.textures.GetTexture("DPUI/Backgrounds/Empty");
```

```
   Draw.TextureIcon(tex);
  }

using (Cell.Row)
{
 Cell.EmptyLinePx(2);

 if (gen.guiExpanded == num)
 {
  using (Cell.LineStd) Draw.Field(ref layer.prototypeType, "Type");

  using (Cell.LineStd)
  {
   if (layer.prototypeType == PrototypeType.TerrainLayer)
   {
    Draw.ObjectField(ref layer.terrainLayer, "Layer");
    Cell.current.Expose(layer.id, "terrainLayer", typeof(TerrainLayer));
   }

   else if (layer.prototypeType == PrototypeType.GrassTexture)
   {
    Draw.ObjectField(ref layer.texture, "Texture");
    Cell.current.Expose(layer.id, "texture", typeof(Texture2D));
    Draw.AddFieldToCellObj(typeof(Add210), "texture");
   }
```

```
   else
   {
    Draw.ObjectField(ref layer.prefab, "Prefab");
    Cell.current.Expose(layer.id, "prefab", typeof(GameObject));
   }
  }

  using (Cell.LineStd)
  {
   Draw.Field(ref layer.instanceNum, "Num");
   Cell.current.Expose(layer.id, "instanceNum", typeof(int));
  }

  using (Cell.LineStd) layer.Opacity = Draw.Field(layer.Opacity, "Opacity");
 }

 else
  Draw.Label(layer.Object!=null ? layer.Object.name : "Empty");
}

using (Cell.RowPx(20))
 using (Cell.LinePx(25))
 {
  Cell.current.trackChange = false;
  Draw.LayerChevron(num, ref gen.guiExpanded);
 }
```

```
 }
}



[Draw.Editor(typeof(Combine210))]
public static void DrawCombineGen (Combine210 gen)
{
 using (Cell.LinePx(0))
 {
  using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true, unlin
  using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito

  //using (Cell.Padded(1,1,0,0))
  {
   using (Cell.LineStd)
   {
    Draw.Toggle(ref gen.normalize, "Normalize");
    Cell.current.Expose(gen.id, "normalize", typeof(bool));
   }
  }
 }
}

private static void DrawCombineLayer (Generator tgen, int num)
{
 Combine210 gen = (Combine210)tgen;
```

```
   Combine210.Layer layer = gen.layers[num];


   if (layer == null) return;


   using (Cell.LinePx(20))
   {
    using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(layer, gen);


    Cell.EmptyRowPx(10);


    using (Cell.RowPx(73)) Draw.Label("Layer " + num);
   }
  }

 }
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;

namespace MapMagic.Nodes.MatrixSetsGenerators
{
 [System.Serializable]
 [GeneratorMenu (menu="Map Set", name ="Curve", iconName="GeneratorIcons/Curve", disengageable =
  helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Curve")]
 public class Curve200 : Generator, IInlet<MatrixSet>, IOutlet<MatrixSet>
 {
  public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

  public Curve curve = new Curve( new Vector2(0,0), new Vector2(1,1) );

  public override void Generate (TileData data, StopToken stop)
  {
   if (stop!=null && stop.stop) return;
```

```
  MatrixSet src = data.ReadInletProduct(this);
  if (!enabled) { data.StoreProduct(this, src); return; }

  curve.Refresh(updateLut:true);

  MatrixSet dst = new MatrixSet(src);

  for (int i=0; i<dst.Count; i++)
  {
   if (stop!=null && stop.stop) return;
   dst[i].UniformCurve(curve.lut);
  }

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map Set", name ="Levels", iconName="GeneratorIcons/Levels", disengageable
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Levels")]
public class Levels200 : Generator, IInlet<MatrixSet>, IOutlet<MatrixSet>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
```

```
public float inMin = 0;

public float inMax = 1;

public float gamma = 1f; //min/max bias. 0 for min 2 for max, 1 is straight curve


public float outMin = 0;

public float outMax = 1;


public bool guiParams = false;



public override void Generate (TileData data, StopToken stop)

{

 if (stop!=null && stop.stop) return;

 MatrixSet src = data.ReadInletProduct(this);

 if (!enabled) { data.StoreProduct(this, src); return; }


 if (stop!=null && stop.stop) return;

 MatrixSet dst = new MatrixSet(src);


 for (int i=0; i<dst.Count; i++)

 {

  if (stop!=null && stop.stop) return;

  dst[i].Levels(inMin, inMax, gamma, outMin, outMax);

 }


 if (stop!=null && stop.stop) return;
```

```
  data.StoreProduct(this, dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map Set", name ="Contrast", iconName="GeneratorIcons/Contrast", disengagea
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Contrast")]
public class Contrast200 : Generator, IInlet<MatrixSet>, IOutlet<MatrixSet>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val(name="Intensity")] public float brightness = 0f;
 [Val(name="Contrast")] public float contrast = 1f;


 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
  MatrixSet src = data.ReadInletProduct(this);
  if (!enabled) { data.StoreProduct(this, src); return; }

  if (stop!=null && stop.stop) return;
  MatrixSet dst = new MatrixSet(src);

  for (int i=0; i<dst.Count; i++)
  {
```

```
    if (stop!=null && stop.stop) return;
    dst[i].BrighnesContrast(brightness, contrast);
   }

   if (stop!=null && stop.stop) return;
   data.StoreProduct(this, dst);
  }
 }

[System.Serializable]
[GeneratorMenu (menu="Map Set", name ="Unity Curve", iconName="GeneratorIcons/UnityCurve", disen
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/UnityCurve")]
public class UnityCurve200 : Generator, IMultiInlet, IOutlet<MatrixSet>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Inlet", "Inlet")] public readonly IInlet<MatrixSet> srcIn = new Inlet<MatrixSet>();
 [Val("Mask", "Inlet")] public readonly IInlet<MatrixWorld> maskIn = new Inlet<MatrixWorld>();
 public IEnumerable<IInlet<object>> Inlets() { yield return srcIn; yield return maskIn; }

 public AnimationCurve curve = new AnimationCurve( new Keyframe[] { new Keyframe(0,0,1,1), new Keyfr
 public Vector2 min = new Vector2(0,0);
 public Vector2 max = new Vector2(1,1);

 public override void Generate (TileData data, StopToken stop)
 {
```

```
if (stop!=null && stop.stop) return;
MatrixSet src = data.ReadInletProduct(srcIn);
MatrixWorld mask = data.ReadInletProduct(maskIn);
if (src == null) return;
if (!enabled) { data.StoreProduct(this, src); return; }

//preparing output
if (stop!=null && stop.stop) return;
MatrixSet dst = new MatrixSet(src);

//curve
for (int i=0; i<dst.Count; i++)
{
 if (stop!=null && stop.stop) return;
 AnimCurve c = new AnimCurve(curve);
 float[] arr = dst[i].arr;
 for (int p=0; p<arr.Length; p++)
  arr[p] = c.Evaluate(arr[p]);
}

//mask
if (stop!=null && stop.stop) return;
if (mask != null)
{
 for (int i=0; i<dst.Count; i++)
  dst[i].InvMix(src[i],mask);
```

```
  }

  data.StoreProduct(this, dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map Set", name ="Mask", iconName="GeneratorIcons/MapMask", disengageabl
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Mask")]
public class Mask200 : Generator, IMultiInlet, IOutlet<MatrixSet>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Input A", "Inlet")] public readonly Inlet<MatrixSet> aIn = new Inlet<MatrixSet>();
 [Val("Input B", "Inlet")] public readonly Inlet<MatrixSet> bIn = new Inlet<MatrixSet>();
 [Val("Mask", "Inlet")] public readonly Inlet<MatrixWorld> maskIn = new Inlet<MatrixWorld>();
 public IEnumerable<IInlet<object>> Inlets () { yield return aIn; yield return bIn; yield return maskIn; }

 [Val("Invert")] public bool invert = false;


 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
  MatrixSet setA = data.ReadInletProduct(aIn);
  MatrixSet setB = data.ReadInletProduct(bIn);
```

```
  MatrixWorld mask = data.ReadInletProduct(maskIn);
  if (setA == null || setB == null) return;
  if (!enabled || mask == null) { data.StoreProduct(this, setA); return; }

  if (stop!=null && stop.stop) return;
  MatrixSet dst = new MatrixSet(setA);
  //dst.SyncPrototypes(setA.Prototypes); //already has prototypes
  dst.SyncPrototypes(setB.Prototypes);

  foreach (MatrixSet.Prototype prototype in dst.Prototypes)
  {
   Matrix matrixB = setB[prototype];
   if (matrixB == null)
    continue;

   dst[prototype].Mix(matrixB, mask, 0, 1, invert, false, 1);
  }

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, dst);
 }
}


/*[System.Serializable]
[GeneratorMenu (menu="Map/Modifiers", name ="Blend", iconName="GeneratorIcons/Blend", disengagea
```

```
public class Blend200 : Generator, IMultiInlet, IOutlet<MatrixWorld>
{
 public class Layer
 {
  public readonly Inlet<MatrixWorld> inlet = new Inlet<MatrixWorld>();
  public BlendAlgorithm algorithm = BlendAlgorithm.add;
  public float opacity = 1;
  public bool guiExpanded = false;
 }

 public Layer[] layers = new Layer[] { new Layer(), new Layer() };
 public Layer[] Layers => layers;
 public void SetLayers(object[] ls) => layers = Array.ConvertAll(ls, i=>(Layer)i);

 public IEnumerable<IInlet<object>> Inlets()
 {
  for (int i=0; i<layers.Length; i++)
   yield return layers[i].inlet;
 }

 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
  if (!enabled) return;
  MatrixWorld matrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSiz
```

```
 if (stop!=null && stop.stop) return;


 if (stop!=null && stop.stop) return;
  for (int i = 0; i < layers.Length; i++)
 {
  Layer layer = layers[i];
  if (layer.inlet == null) continue;


  MatrixWorld blendMatrix = data.ReadInletProduct(layer.inlet);
  if (blendMatrix == null) continue;


  Blend(matrix, blendMatrix, layer.algorithm, layer.opacity);
 }


 data.StoreProduct(this, matrix);
}



public enum BlendAlgorithm {
 mix=0,
 add=1,
 subtract=2,
 multiply=3,
 divide=4,
 difference=5,
 min=6,
```

```
  max=7,
  overlay=8,
  hardLight=9,
  softLight=10}

 public static void Blend (Matrix m1, Matrix m2, BlendAlgorithm algorithm, float opacity=1)
{
  switch (algorithm)
  {
   case BlendAlgorithm.mix: default: m1.Mix(m2, opacity); break;
   case BlendAlgorithm.add: m1.Add(m2, opacity); break;
   case BlendAlgorithm.subtract: m1.Subtract(m2, opacity); break;
   case BlendAlgorithm.multiply: m1.Multiply(m2, opacity); break;
   case BlendAlgorithm.divide: m1.Divide(m2, opacity); break;
   case BlendAlgorithm.difference: m1.Difference(m2, opacity); break;
   case BlendAlgorithm.min: m1.Min(m2, opacity); break;
   case BlendAlgorithm.max: m1.Max(m2, opacity); break;
   case BlendAlgorithm.overlay: m1.Overlay(m2, opacity); break;
   case BlendAlgorithm.hardLight: m1.HardLight(m2, opacity); break;
   case BlendAlgorithm.softLight: m1.SoftLight(m2, opacity); break;
  }
 }
}*/


/*[System.Serializable]
```

```
[GeneratorMenu (
 menu="Map/Modifiers",
 name ="Normalize",
 disengageable = true,
 helpLink ="https://gitlab.com/denispahunov/mapmagic/wikis/map_generators/normalize",
 iconName="GeneratorIcons/Normalize",
 drawInlets = false,
 drawOutlet = false,
 colorType = typeof(MatrixWorld))]
public class Normalize200 : Generator, IMultiInlet, IMultiOutlet
{
 public class NormalizeLayer : IInlet<MatrixWorld>, IOutlet<MatrixWorld>
 {
  public float Opacity { get; set; }

  public Generator Gen { get; private set; }
  public void SetGen (Generator gen) => Gen=gen;
  public NormalizeLayer (Generator gen) { this.Gen = gen; }
  public NormalizeLayer () { Opacity = 1; }

  public ulong id; //properties not serialized
  public ulong Id { get{return id;} set{id=value;} }
  public ulong LinkedOutletId { get; set; }  //if it's inlet. Assigned every before each clear or generate
  public ulong LinkedGenId { get; set; }

  public IUnit ShallowCopy() => (NormalizeLayer)this.MemberwiseClone();
```

```
}

public NormalizeLayer[] layers = new NormalizeLayer[0];
public NormalizeLayer[] Layers => layers;
public void SetLayers(object[] ls) => layers = Array.ConvertAll(ls, i=>(NormalizeLayer)i);



public IEnumerable<IInlet<object>> Inlets()
{
 for (int i=0; i<layers.Length; i++)
  yield return layers[i];
}

public IEnumerable<IOutlet<object>> Outlets()
{
 for (int i=0; i<layers.Length; i++)
  yield return layers[i];
}

public override void Generate (TileData data, StopToken stop)
{
 if (layers.Length == 0) return;

 //reading/copying products
 MatrixWorld[] dstMatrices = new MatrixWorld[layers.Length];
 float[] opacities = new float[layers.Length];
```

```
 if (stop!=null && stop.stop) return;
 for (int i=0; i<layers.Length; i++)
 {
  if (stop!=null && stop.stop) return;

  MatrixWorld srcMatrix = data.ReadInletProduct(layers[i]);
  if (srcMatrix != null) dstMatrices[i] = new MatrixWorld(srcMatrix);
  else dstMatrices[i] = new MatrixWorld(data.area.full.rect, (Vector3)data.area.full.worldPos, (Vector3)data

  opacities[i] = layers[i].Opacity;
 }

 //normalizing
 if (stop!=null && stop.stop) return;
 dstMatrices.FillNulls(() => new MatrixWorld(data.area.full.rect, (Vector3)data.area.full.worldPos, (Vector3
 dstMatrices[0].Fill(1);
 Matrix.BlendLayers(dstMatrices, opacities);

 //saving products
 if (stop!=null && stop.stop) return;
 for (int i=0; i<layers.Length; i++)
  data.StoreProduct(layers[i], dstMatrices[i]);
 }
}*/
```

```
[System.Serializable]
[GeneratorMenu (menu="Map Set", name ="Blur", iconName="GeneratorIcons/Blur", disengageable = true
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Blur")]
public class Blur200 : Generator, IInlet<MatrixSet>, IOutlet<MatrixSet>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Downsample")] public float downsample = 10f;
 [Val("Blur")] public float blur = 3f;

 public override void Generate (TileData data, StopToken stop)
 {
  MatrixSet src = data.ReadInletProduct(this);
  if (src == null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }

  MatrixSet dst = new MatrixSet(src);

  int rrDownsample = (int)(downsample / Mathf.Sqrt(dst.PixelSize.x));
  float rrBlur = blur / dst.PixelSize.x;

  for (int m=0; m<src.Count; m++)
  {
   if (stop!=null && stop.stop) return;

   if (rrDownsample > 1)
```

```
    MatrixOps.DownsampleBlur(src[m], dst[m], rrDownsample, rrBlur);
   else
    MatrixOps.GaussianBlur(src[m], dst[m], rrBlur);
  }

  data.StoreProduct(this, dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map Set", name ="Cavity", iconName="GeneratorIcons/Cavity", disengageable =
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Cavity")]
public class Cavity200 : Generator, IInlet<MatrixSet>, IOutlet<MatrixSet>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Type")]  public MatrixGenerators.Cavity200.CavityType type = MatrixGenerators.Cavity200.CavityTy
 [Val("Intensity")] public float intensity = 3;
 [Val("Spread")]  public float spread = 10; //actually the pixel size (in world units) of the lowerest mipmap. S

 public override void Generate (TileData data, StopToken stop)
 {
  MatrixSet src = data.ReadInletProduct(this);
  if (src == null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }
```

```
  if (stop!=null && stop.stop) return;
  MatrixSet dst = new MatrixSet(src.rect, src.worldPos, src.worldSize, src.Prototypes); //empty matrices, no

  for (int m=0; m<src.Count; m++)
  {
   if (stop!=null && stop.stop) return;
   MatrixGenerators.Cavity200.Cavity(src[m], dst[m], type, intensity, spread, src.PixelSize.x, data.area.activ
  }

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map Set", name ="Slope", iconName="GeneratorIcons/Slope", disengageable =
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Slope")]
public class Slope200 : Generator, IInlet<MatrixSet>, IOutlet<MatrixSet>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("From")]   public float from = 30;
 [Val("To")]    public float to = 90;
 [Val("Smooth Range")] public float range = 30f;

 public override void Generate (TileData data, StopToken stop)
```

```
 {
  MatrixSet src = data.ReadInletProduct(this);
  if (src==null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }

  MatrixSet dst = new MatrixSet(src.rect, src.worldPos, src.worldSize, src.Prototypes); //empty matrices, no

  for (int m=0; m<src.Count; m++)
  {
   if (stop!=null && stop.stop) return;
   dst[m] = MatrixGenerators.Slope200.Slope(src[m], src.worldPos, src.worldSize, data.globals.height, from
  }

  data.StoreProduct(this, dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map Set", name ="Selector", iconName="GeneratorIcons/Selector", disengageal
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Selector")]
public class Selector200 : Generator, IInlet<MatrixSet>, IOutlet<MatrixSet>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 public enum RangeDet { Transition, MinMax}
```

```
 public RangeDet rangeDet = RangeDet.Transition;
 public enum Units { Map, World }
 public Units units = Units.Map;
 public Vector2 from = new Vector2(0.4f, 0.6f);
 public Vector2 to = new Vector2(1f, 1f);

 public override void Generate (TileData data, StopToken stop)
 {
  MatrixSet src = data.ReadInletProduct(this);
  if (src==null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }

  if (stop!=null && stop.stop) return;
  MatrixSet dst = new MatrixSet(src);

  for (int m=0; m<src.Count; m++)
  {
   if (stop!=null && stop.stop) return;
   MatrixWorld dstW = new MatrixWorld(dst[m], dst.worldPos, dst.worldSize);
   MatrixGenerators.Selector200.Select(dstW, from, to, inWorldUnits:units==Units.World, worldHeight:data
  }

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, dst);
 }
}
```

```
[System.Serializable]
[GeneratorMenu (
 menu="Map Set",
 name ="Terrace",
 iconName="GeneratorIcons/Terrace",
 disengageable = true,
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Terrace")]
public class Terrace200 : Generator, IInlet<MatrixSet>, IOutlet<MatrixSet>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Seed")]   public int seed = 12345;
 [Val("Num")]   public int num = 10;
 [Val("Uniformity")] public float uniformity = 0.5f;
 [Val("Steepness")]  public float steepness = 0.5f;
 //[Val("Intensity")]  public float intensity = 1f;


 public override void Generate (TileData data, StopToken stop)
 {
  MatrixSet src = data.ReadInletProduct(this);
  if (src == null || num <= 1) return;
  if (!enabled) { data.StoreProduct(this, src); return; }

  MatrixSet dst = new MatrixSet(src);
```

```
  for (int m=0; m<src.Count; m++)
  {
   if (stop!=null && stop.stop) return;
   float[] terraceLevels = MatrixGenerators.Terrace200.TerraceLevels(new Noise(data.random,seed), num

   if (stop!=null && stop.stop) return;
   dst[m].Terrace(terraceLevels, steepness);
  }

  data.StoreProduct(this, dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Map Set", name ="Erosion", iconName="GeneratorIcons/Erosion", disengageabl
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Erosion")]
public class Erosion200 : Generator, IInlet<MatrixSet>, IOutlet<MatrixSet>, ICustomComplexity
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Iterations")]   public int iterations = 3;
 [Val("Durability")] public float terrainDurability=0.9f;
 //[Val("Erosion")]
  public float erosionAmount=1f;
 [Val("Sediment")]  public float sedimentAmount=0.75f;
```

```
 [Val("Fluidity")] public int fluidityIterations=3;
 [Val("Relax")]  public float relax=0.0f;

 public float Complexity {get{ return iterations*2; }}
 public float Progress (TileData data) { return data.GetProgress(this); }



 public override void Generate (TileData data, StopToken stop)
 {
  MatrixSet src = data.ReadInletProduct(this);
  if (src == null) return;
  if (!enabled || iterations <= 0) { data.StoreProduct(this, src); return; }

  MatrixSet dst = new MatrixSet(src);

  for (int m=0; m<src.Count; m++)
  {
   if (stop!=null && stop.stop) return;
   MatrixWorld dstW = new MatrixWorld(dst[m], dst.worldPos, dst.worldSize);
   MatrixGenerators.Erosion200.Erosion(dstW, data.isDraft, data, iterations, terrainDurability, erosionAmou
  }

  data.StoreProduct(this, dst);
 }
}
```

```
[Serializable]
[GeneratorMenu(
 menu = "Map Set",
 name = "Parallax",
 section=2,
 colorType = typeof(MatrixSet),
 iconName="GeneratorIcons/Parallax",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Parallax")]
public class Parallax210 : Generator, IInlet<MatrixSet>, IMultiInlet, IOutlet<MatrixSet>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
 [Val("Intensity X", "Inlet")] public readonly Inlet<MatrixWorld> intensityInX = new Inlet<MatrixWorld>();
 [Val("Intensity Z", "Inlet")] public readonly Inlet<MatrixWorld> intensityInZ = new Inlet<MatrixWorld>();
 public virtual IEnumerable<IInlet<object>> Inlets () { yield return intensityInX; yield return intensityInZ; }

 [Val("Offset")] public Vector2D offset;
 public enum Interpolation { None, Always, OnTransitions }
 [Val("Interpolation")] public Interpolation interpolation = Interpolation.Always;

 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
  MatrixSet maps = data.ReadInletProduct(this);
  if (maps == null) return;
  if (!enabled) { data.StoreProduct(this,maps); return; }
```

```
   MatrixWorld intensityX = data.ReadInletProduct(intensityInX);
   MatrixWorld intensityZ = data.ReadInletProduct(intensityInZ);

   if (stop!=null && stop.stop) return;
   Vector2D pixelOffset = offset / (Vector2D)data.area.full.PixelSize;
   MatrixSet results = new MatrixSet(maps.rect, maps.worldPos, maps.worldSize, maps.Prototypes);

   for (int m=0; m<maps.Count; m++)
   {
    if (stop!=null && stop.stop) return;
    results.GetMatrixByNum(m).Parallax(pixelOffset, maps.GetMatrixByNum(m), intensityX, intensityZ, (int)i
   }

   if (stop!=null && stop.stop) return;
   data.StoreProduct(this, results);
  }
 }
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Core;
using MapMagic.Products;

namespace MapMagic.Nodes.MatrixSetsGenerators
{
 public enum PrototypeType { TerrainLayer, GrassTexture, GrassPrefab };



 [Serializable]
 [GeneratorMenu(
  menu = "Map Set",
  name = "Add",
  colorType = typeof(MatrixSet),
  iconName="GeneratorIcons/MapSetAdd",
  helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixSetGenerators/Add")]
 public class Add210 : Generator, IInlet<MatrixSet>, IMultiInlet, IOutlet<MatrixSet>
 {
```

```
public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

[Val("Add", "Inlet")] public readonly Inlet<MatrixWorld> addIn = new Inlet<MatrixWorld>();
public IEnumerable<IInlet<object>> Inlets () { yield return addIn; }

public PrototypeType prototypeType = PrototypeType.TerrainLayer;
public TerrainLayer terrainLayer = null;
public Texture2D texture = null;
public GameObject prefab = null;
public int instanceNum = 0;
public enum Mode { Set, Add, AddNormalize };
public Mode mode = Mode.Add;
//public bool normalize = false;
public float opacity = 1;

public MatrixSet.Prototype Prototype
{get{
 switch (prototypeType)
 {
  case PrototypeType.TerrainLayer: return new MatrixSet.Prototype(terrainLayer, instanceNum);
  case PrototypeType.GrassPrefab: return new MatrixSet.Prototype(prefab, instanceNum);
  case PrototypeType.GrassTexture: default: return new MatrixSet.Prototype(texture, instanceNum);
 }
}}
```

```
public override void Generate (TileData data, StopToken stop)
{
 if (stop!=null && stop.stop) return;
 MatrixSet src = data.ReadInletProduct(this);

 MatrixWorld add = data.ReadInletProduct(addIn);
 if (!enabled || (texture==null && terrainLayer==null && prefab==null) || add==null)
 {
  data.StoreProduct(this, src);
  return;
 }

 if (opacity < 0.99f  ||  opacity > 1.01f)
 {
  add = new MatrixWorld(add);
  add.Multiply(opacity);
 }

 if (stop!=null && stop.stop) return;
 MatrixSet dst;
 if (src==null) dst = new MatrixSet(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSize, dat
 else dst = new MatrixSet(src);

 if (mode == Mode.Set)
  dst[Prototype] = add;
 else
```

```
   dst.Append(add, Prototype, normalized:mode==Mode.AddNormalize);


   if (stop!=null && stop.stop) return;
   data.StoreProduct(this, dst);
  }
 }


[Serializable]
[GeneratorMenu(
 menu = "Map Set",
 name = "Pick",
 colorType = typeof(MatrixSet),
 iconName="GeneratorIcons/MapSetPick",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixSetGenerators/Pick")]
public class Pick210 : Generator, IInlet<MatrixSet>, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on


 [Val("Add", "Inlet")] public readonly Inlet<MatrixWorld> addIn = new Inlet<MatrixWorld>();
 public IEnumerable<IInlet<object>> Inlets () { yield return addIn; }


 public PrototypeType prototypeType = PrototypeType.TerrainLayer;


 public TerrainLayer terrainLayer = null;
 public Texture2D texture = null;
```

```
public GameObject prefab = null;
public int instanceNum = 0;
public bool createIfNotExists = true;
public float opacity = 1;

public MatrixSet.Prototype Prototype
{get{
 switch (prototypeType)
 {
  case PrototypeType.TerrainLayer: return new MatrixSet.Prototype(terrainLayer, instanceNum);
  case PrototypeType.GrassPrefab: return new MatrixSet.Prototype(prefab, instanceNum);
  case PrototypeType.GrassTexture: default: return new MatrixSet.Prototype(texture, instanceNum);
 }
}}

public override void Generate (TileData data, StopToken stop)
{
 if (stop!=null && stop.stop) return;
 MatrixSet src = data.ReadInletProduct(this);
 if (src == null) return;

 Matrix pick = src[Prototype];
 if (pick == null) //prototype not assigned
 {
  if (createIfNotExists) pick = new Matrix(data.area.full.rect);
  else return;
```

```
  }

  MatrixWorld pickW = new MatrixWorld(pick, src.worldPos, src.worldSize);

  if (opacity < 0.99f  ||  opacity > 1.01f)
  {
   pickW = new MatrixWorld(pickW);
   pickW.Multiply(1f/opacity); //opacity value inverted in pick
  }

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, pickW);
 }
}


[Serializable]
[GeneratorMenu(
 menu = "Map Set",
 name = "Create",
 colorType = typeof(MatrixSet),
 iconName="GeneratorIcons/MapSetCreate",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixSetGenerators/Create")]
public class Create210 : Generator, IMultiInlet, IOutlet<MatrixSet>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
```

```
public class Layer : IInlet<MatrixWorld>
{
 public PrototypeType prototypeType = PrototypeType.TerrainLayer;
 public TerrainLayer terrainLayer = null;
 public Texture2D texture = null;
 public GameObject prefab = null;
 public int instanceNum = 0;
 public float Opacity { get; set; }

 public Generator Gen { get; private set; }
 public void SetGen (Generator gen) => Gen=gen;
 public Layer (Generator gen) { this.Gen = gen; }
 public Layer () { Opacity = 1; }

 public ulong id; //properties not serialized
 public ulong Id { get{return id;} set{id=value;} }
 public ulong LinkedOutletId { get; set; }  //if it's inlet. Assigned every before each clear or generate
 public ulong LinkedGenId { get; set; }

 public IUnit ShallowCopy() => (Layer)this.MemberwiseClone();

 public MatrixSet.Prototype Prototype
 {get{
  switch (prototypeType)
  {
```

```
   case PrototypeType.TerrainLayer: return new MatrixSet.Prototype(terrainLayer, instanceNum);
   case PrototypeType.GrassPrefab: return new MatrixSet.Prototype(prefab, instanceNum);
   case PrototypeType.GrassTexture: default: return new MatrixSet.Prototype(texture, instanceNum);
  }
 }}

 public UnityEngine.Object Object
 {get{
  if (prototypeType==PrototypeType.TerrainLayer) return texture;
  if (prototypeType==PrototypeType.GrassTexture) return texture;
  else return prefab;
 }}
}

public Layer[] layers = new Layer[0];
public Layer[] Layers => layers;
public void SetLayers(object[] ls) => layers = Array.ConvertAll(ls, i=>(Layer)i);
public int guiExpanded;

public IEnumerable<IInlet<object>> Inlets()
{
 for (int i=0; i<layers.Length; i++)
  yield return layers[i];
}
```

```
public enum Normalize { None, Sum, Layered }

public Normalize normalize = Normalize.None;


public override void Generate (TileData data, StopToken stop)

{

 if (!enabled || layers.Length == 0) return;


 //reading/copying products

 MatrixWorld[] dstMatrices = new MatrixWorld[layers.Length];

 float[] opacities = new float[layers.Length];


 if (stop!=null && stop.stop) return;

 for (int i=0; i<layers.Length; i++)

 {

  if (stop!=null && stop.stop) return;


  MatrixWorld srcMatrix = data.ReadInletProduct(layers[i]);


  if (normalize == Normalize.None)

   dstMatrices[i] = srcMatrix;


  else

  {

   if (srcMatrix != null) dstMatrices[i] = new MatrixWorld(srcMatrix);

   else dstMatrices[i] = new MatrixWorld(data.area.full.rect, (Vector3)data.area.full.worldPos, (Vector3)dat

  }
```

```
 opacities[i] = layers[i].Opacity;
}

//normalizing
if (stop!=null && stop.stop) return;
if (normalize != Normalize.None)
{
 dstMatrices.FillNulls(() => new MatrixWorld(data.area.full.rect, (Vector3)data.area.full.worldPos, (Vector3

 if (normalize == Normalize.Sum)
  Matrix.NormalizeLayers(dstMatrices, opacities);

 if (normalize == Normalize.Layered)
 {
  dstMatrices[0].Fill(1);
  Matrix.BlendLayers(dstMatrices, opacities);
 }
}

//assembling
if (stop!=null && stop.stop) return;
MatrixSet set = new MatrixSet(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSize, data.g
for (int i=0; i<layers.Length; i++)
 set.Append(dstMatrices[i], layers[i].Prototype, normalized:false);
 //no need to normalize since it's already was normalized
```

```
  //storing products
  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, set);
 }
}


[Serializable]
[GeneratorMenu(
 menu = "Map Set",
 name = "Combine",
 colorType = typeof(MatrixSet),
 iconName="GeneratorIcons/MapSetCombine",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixSetGenerators/Combine")]
public class Combine210 : Generator, IMultiInlet, IOutlet<MatrixSet>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 public class Layer : IInlet<MatrixSet>
 {
  public Generator Gen { get; private set; }
  public void SetGen (Generator gen) => Gen=gen;
  public Layer (Generator gen) { this.Gen = gen; }
  public Layer () { } //for editor
```

```
 public ulong id; //properties not serialized
 public ulong Id { get{return id;} set{id=value;} }
 public ulong LinkedOutletId { get; set; }  //if it's inlet. Assigned every before each clear or generate
 public ulong LinkedGenId { get; set; }

 public IUnit ShallowCopy() => (Layer)this.MemberwiseClone();
}

public Layer[] layers = new Layer[0];
public Layer[] Layers => layers;
public void SetLayers(object[] ls) => layers = Array.ConvertAll(ls, i=>(Layer)i);

public IEnumerable<IInlet<object>> Inlets()
{
 for (int i=0; i<layers.Length; i++)
  yield return layers[i];
}

public bool normalize = false;


public override void Generate (TileData data, StopToken stop)
{
 if (!enabled || layers.Length == 0) return;

 //gathering all matrix and prototypes
```

```
List<Matrix> matricesList = new List<Matrix>();
List<MatrixSet.Prototype> prototypesList = new List<MatrixSet.Prototype>();

if (stop!=null && stop.stop) return;
foreach (Layer layer in layers)
{
 MatrixSet layerSet = data.ReadInletProduct(layer);
 if (layerSet == null) continue;

 for (int m=0; m<layerSet.Count; m++)
 {
  matricesList.Add(layerSet.GetMatrixByNum(m));
  prototypesList.Add(layerSet.GetPrototypeByNum(m));
 }
}

Matrix[] matrices = matricesList.ToArray();
MatrixSet.Prototype[] prototypes = prototypesList.ToArray();

//copying to normalize
if (normalize)
 for (int m=0; m<matrices.Length; m++)
  matrices[m] = new Matrix(matrices[m]);

//normalizing (including copy)
if (normalize)
```

```
    Matrix.NormalizeLayers(matrices);


   //assembling new set
   MatrixSet set = new MatrixSet(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSize, data.g
   for (int i=0; i<layers.Length; i++)
    set.Append(matrices[i], prototypes[i], normalized:false);
    //no need to normalize since it's already was normalized


   if (stop!=null && stop.stop) return;
   data.StoreProduct(this, set);
  }
 }
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes.GUI;
using MapMagic.Nodes.ObjectsGenerators;

namespace MapMagic.Nodes.GUI
{
 public static partial class ObjectsEditors
 {
  /*[Draw.Editor(typeof(ObjectsGenerators.Random200))]
  public static void DrawOutdated (ObjectsGenerators.Random200 gen)
  {
   using (Cell.Padded(1,1,0,0))
   {
    //Draw.Element(UI.current.styles.foldoutBackground);
    using (Cell.LinePx(70))
     Draw.Label("This node is outdated \nand should not be used. \nTry replacing it with the \nnew node with
```

```
 }
}*/

/*[Draw.Editor(typeof(ObjectsGenerators.Scatter200))]
public static void DrawScatter (ObjectsGenerators.Scatter200 gen)
{
 //drawing original values

 using (Cell.Padded(1,1,0,0))
 {
  Cell.EmptyLinePx(2);
  using (Cell.LinePx(0))
   using (new Draw.FoldoutGroup(ref gen.guiAdvanced, "Advanced"))
   if (gen.guiAdvanced)
   {
    using (Cell.LineStd) Draw.Field(ref gen.relax, "Relax");
    using (Cell.LineStd) Draw.Field(ref gen.additionalMargins, "Add Margin");
   }
  Cell.EmptyLinePx(2);
 }
}*/

private static void CheckDrawRadiusWarning (float genRadius)
{
  MapMagicObject mapMagic = GraphWindow.current.mapMagic as MapMagicObject;
  if (mapMagic != null)
```

```
  {
   float pixelSize = mapMagic.tileSize.x / (int)mapMagic.tileResolution;
   if (genRadius > mapMagic.tileMargins * pixelSize)
   {
    Cell.EmptyLinePx(2);
    using (Cell.LinePx(40))
     using (Cell.Padded(2,2,0,0))
     {
      Draw.Element(UI.current.styles.foldoutBackground);
      using (Cell.LinePx(15)) Draw.Label("Current setup can");
      using (Cell.LinePx(15)) Draw.Label("create tile seams");
      using (Cell.LinePx(15)) Draw.URL("More", url:"https://gitlab.com/denispahunov/mapmagic/-/wikis/Tile_
     }
    Cell.EmptyLinePx(2);
   }
  }
}


[Draw.Editor(typeof(ObjectsGenerators.Spread200))]
public static void DrawSpread (ObjectsGenerators.Spread200 gen)
{
 using (Cell.Padded(1,1,0,0))
 {
  using (Cell.LineStd) Draw.ToggleLeft(ref gen.retainOriginals, "Retain Originals");
  using (Cell.LineStd)
  {
```

```
   Draw.Field(ref gen.seed, "Seed");
   Cell.current.Expose(gen.id, "seed", typeof(int));
  }

  Cell.EmptyLinePx(5);
  using (Cell.LineStd)
  {
   Draw.Field(ref gen.growth, "Growth", "Min", "Max", 25);
   Cell.current.Expose(gen.id, "growth", typeof(Vector2));
  }

  Cell.EmptyLinePx(5);
  using (Cell.LineStd)
  {
   Draw.Field(ref gen.distance, "Dist", "Min", "Max", 25);
   Cell.current.Expose(gen.id, "distance", typeof(Vector2));
  }

  Cell.EmptyLinePx(5);
  using (Cell.LineStd)
  {
   Draw.Field(ref gen.sizeFactor, "Size Factor");
   Cell.current.Expose(gen.id, "sizeFactor", typeof(float));
  }
 }
}
```

```
[Draw.Editor(typeof(ObjectsGenerators.Adjust200))]
public static void DrawObjectsAdjust (ObjectsGenerators.Adjust200 adj)
{
 using (Cell.Padded(1,1,0,0))
 {
  if (!adj.useRandom)
  {
   using (Cell.LinePx(0))
   using (Cell.Padded(2,0,0,0))
   {
    using (Cell.LineStd)
    {
     Draw.IconField(ref adj.height.x, "Height", UI.current.textures.GetTexture("DPUI/Icons/Height"));
     Cell.current.Expose(adj.id, "height", typeof(Vector2));
    }

    using (Cell.LineStd)
    {
     Draw.IconField(ref adj.offsetFront.x, "Front", UI.current.textures.GetTexture("DPUI/Icons/Front"));
     Cell.current.Expose(adj.id, "front", typeof(Vector2));
    }

    using (Cell.LineStd)
    {
```

```
   Draw.IconField(ref adj.offsetRight.x, "Right", UI.current.textures.GetTexture("DPUI/Icons/Right"));
   Cell.current.Expose(adj.id, "right", typeof(Vector2));
  }

  using (Cell.LineStd)
  {
   Draw.IconField(ref adj.rotation.x, "Rotation", UI.current.textures.GetTexture("DPUI/Icons/Rotate"));
   Cell.current.Expose(adj.id, "rotation", typeof(Vector2));
  }

  using (Cell.LineStd)
  {
   Draw.IconField(ref adj.scale.x, "Scale", UI.current.textures.GetTexture("DPUI/Icons/Scale"));
   Cell.current.Expose(adj.id, "scale", typeof(Vector2));
  }
 }
}

else
{
 using (Cell.LineStd)
 {
  Cell.current.fieldWidth = 0.6f;
  using (Cell.RowPx(16)) Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Height") );
  using (Cell.RowRel(0.5f))
   Draw.FieldDragIcon(ref adj.height.x);
```

```
 using (Cell.RowRel(0.5f))
  Draw.FieldDragIcon(ref adj.height.y);
 Cell.current.Expose(adj.id, "height", typeof(Vector2));
}

using (Cell.LineStd)
{
 Cell.current.fieldWidth = 0.6f;
 using (Cell.RowPx(16)) Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Front") );
 using (Cell.RowRel(0.5f))
  Draw.FieldDragIcon(ref adj.offsetFront.x);
 using (Cell.RowRel(0.5f))
  Draw.FieldDragIcon(ref adj.offsetFront.y);
 Cell.current.Expose(adj.id, "front", typeof(Vector2));
}

using (Cell.LineStd)
{
 Cell.current.fieldWidth = 0.6f;
 using (Cell.RowPx(16)) Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Right") );
 using (Cell.RowRel(0.5f))
  Draw.FieldDragIcon(ref adj.offsetRight.x);
 using (Cell.RowRel(0.5f))
  Draw.FieldDragIcon(ref adj.offsetRight.y);
 Cell.current.Expose(adj.id, "right", typeof(Vector2));
}
```

```
 using (Cell.LineStd)
 {
  Cell.current.fieldWidth = 0.6f;
  using (Cell.RowPx(16)) Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Rotate") );
  using (Cell.RowRel(0.5f))
   Draw.FieldDragIcon(ref adj.rotation.x);
  using (Cell.RowRel(0.5f))
   Draw.FieldDragIcon(ref adj.rotation.y);
  Cell.current.Expose(adj.id, "rotation", typeof(Vector2));
 }

 using (Cell.LineStd)
 {
  Cell.current.fieldWidth = 0.6f;
  using (Cell.RowPx(16)) Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Scale") );
  using (Cell.RowRel(0.5f))
   Draw.FieldDragIcon(ref adj.scale.x);
  using (Cell.RowRel(0.5f))
   Draw.FieldDragIcon(ref adj.scale.y);
  Cell.current.Expose(adj.id, "scale", typeof(Vector2));
 }

}

using (Cell.LineStd)
```

```
  {
   using (Cell.Row) Draw.Label("Random Range");
   using (Cell.RowPx(18)) Draw.Toggle(ref adj.useRandom);
  }

  if (adj.useRandom)
   using (Cell.LineStd)
   {
    Draw.Field(ref adj.seed, "Seed");
    Cell.current.Expose(adj.id, "seed", typeof(int));
   }

  using (Cell.LineStd)
  {
   Draw.Field(ref adj.sizeFactor, "Size Factor");
   Cell.current.Expose(adj.id, "sizeFactor", typeof(float));
  }

  Cell.EmptyLinePx(5);

  using (Cell.LineStd) Draw.Field(ref adj.relativeness, "Relativity");
 }
}

[Draw.Editor(typeof(ObjectsGenerators.Flatten200))]
```

```
public static void DrawFlatten (ObjectsGenerators.Flatten200 gen)
{
 //drawing standard inspector (radius, hardness)

 using (Cell.Padded(1,1,0,0))
 {
  Cell.EmptyLinePx(3);

  using (Cell.LineStd) Draw.ToggleLeft(ref gen.noiseFallof, "Use Noise on Falloff");

  if (gen.noiseFallof)
  {
   using (Cell.LineStd)
   {
    Draw.Field(ref gen.noiseAmount, "Amount");
    Cell.current.Expose(gen.id, "noiseAmount", typeof(float));
   }
   using (Cell.LineStd)
   {
    Draw.Field(ref gen.noiseSize, "Size");
    Cell.current.Expose(gen.id, "noiseSize", typeof(float));
   }
  }

  //radius warning
  MapMagicObject mapMagic = GraphWindow.current.mapMagic as MapMagicObject;
```

```
  if (mapMagic != null)
  {
   float pixelSize = mapMagic.tileSize.x / (int)mapMagic.tileResolution;
   if (gen.radius > mapMagic.tileMargins * pixelSize)
   {
    Cell.EmptyLinePx(2);
    using (Cell.LinePx(40))
     using (Cell.Padded(2,2,0,0))
     {
      Draw.Element(UI.current.styles.foldoutBackground);
      using (Cell.LinePx(15)) Draw.Label("Current setup can");
      using (Cell.LinePx(15)) Draw.Label("create tile seams");
      using (Cell.LinePx(15)) Draw.URL("More", url:"https://gitlab.com/denispahunov/mapmagic/-/wikis/Tile_
     }
    Cell.EmptyLinePx(2);
   }
  }
 }
}



[Draw.Editor(typeof(ObjectsGenerators.Stroke200))]
public static void DrawStroke (ObjectsGenerators.Stroke200 gen)
{
 //drawing standard inspector (radius, hardness)
```

```
 using (Cell.Padded(1,1,0,0))
 {
  Cell.EmptyLinePx(3);

  using (Cell.LineStd) Draw.ToggleLeft(ref gen.noiseFallof, "Use Noise on Falloff");

  if (gen.noiseFallof)
  {
   using (Cell.LineStd)
   {
    Draw.Field(ref gen.noiseAmount, "Amount");
    Cell.current.Expose(gen.id, "noiseAmount", typeof(float));
   }

   using (Cell.LineStd)
   {
    Draw.Field(ref gen.noiseSize, "Size");
    Cell.current.Expose(gen.id, "noiseSize", typeof(float));
   }
  }

  CheckDrawRadiusWarning(gen.radius);
 }
}
```

```
[Draw.Editor(typeof(ObjectsGenerators.Split200))]
public static void SplitGeneratorEditor (ObjectsGenerators.Split200 gen)
{
 Cell.EmptyLinePx(5);
 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:true);
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:true, layerEdito
}

private static void DrawSplitLayer (Generator tgen, int num)
{
 Split200 splitGen = (Split200)tgen;
 Split200.SplitLayer layer = splitGen.layers[num];
 if (layer == null) return;

 using (Cell.LinePx(0))
 {
  Cell.EmptyLinePx(2);
  using (Cell.LineStd)
  {
   using (Cell.Row) Draw.EditableLabel(ref layer.name);

   using (Cell.RowPx(20)) Draw.LayerChevron(num, ref splitGen.guiExpanded);

   Cell.EmptyRowPx(10);
   using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet(layer);
```

```
 }
 Cell.EmptyLinePx(2);
}

if (splitGen.guiExpanded == num)
 using (Cell.LinePx(0))
  using (Cell.Padded(1,1,0,0))
{
 Cell.EmptyLinePx(5);
 using (Cell.LineStd) Draw.Field(ref layer.chance, "Probability");
 Cell.EmptyLinePx(5);

 using (Cell.LineStd) Draw.ToggleLeft(ref layer.heightConditionActive, "Height Condition");
 if (layer.heightConditionActive)
 {
  using (Cell.LineStd)
  {
   using (Cell.RowRel(0.5f)) Draw.FieldDragIcon(ref layer.heightCondition.x);
   using (Cell.RowRel(0.5f)) Draw.FieldDragIcon(ref layer.heightCondition.y);
  }
  Cell.EmptyLinePx(5);
 }

 using (Cell.LineStd) Draw.ToggleLeft(ref layer.rotationConditionActive, "Rotation Condition");
 if (layer.rotationConditionActive)
 {
```

```
   using (Cell.LineStd)
   {
    using (Cell.RowRel(0.5f)) Draw.FieldDragIcon(ref layer.rotationCondition.x);
    using (Cell.RowRel(0.5f)) Draw.FieldDragIcon(ref layer.rotationCondition.y);
   }
   Cell.EmptyLinePx(5);
  }

  using (Cell.LineStd) Draw.ToggleLeft(ref layer.scaleConditionActive, "Scale Condition");
  if (layer.scaleConditionActive)
  {
   using (Cell.LineStd)
   {
    using (Cell.RowRel(0.5f)) Draw.FieldDragIcon(ref layer.scaleCondition.x);
    using (Cell.RowRel(0.5f)) Draw.FieldDragIcon(ref layer.scaleCondition.y);
   }
   Cell.EmptyLinePx(5);
  }
 }
}



[Draw.Editor(typeof(ObjectsGenerators.ObjectsOutput))]
public static void DrawObjectsOutput (ObjectsGenerators.ObjectsOutput gen)
{
 if (gen.posSettings == null) gen.posSettings = ObjectsOutput.CreatePosSettings(gen);
```

```
using (Cell.LineStd)
 DrawObjectPrefabs(ref gen.prefabs, gen.guiMultiprefab, treeIcon:false);

using (Cell.LinePx(0))
 using (Cell.Padded(2,2,0,0))
{
 using (Cell.LineStd) Draw.ToggleLeft(ref gen.guiMultiprefab, "Multi-Prefab");

 Cell.EmptyRowPx(4);

 using (Cell.LinePx(0))
  using (new Draw.FoldoutGroup(ref gen.guiProperties, "Properties"))
   if (gen.guiProperties)
   {
    Cell.current.fieldWidth = 0.481f;
    using (Cell.LineStd) Draw.Field(ref gen.seed, "Seed");
    using (Cell.LineStd) Draw.ToggleLeft(ref gen.allowReposition, "Use Pool");
    using (Cell.LineStd) Draw.ToggleLeft(ref gen.instantiateClones, "As Clones");
    using (Cell.LineStd) Draw.Field(ref gen.biomeBlend, "Biome Blend");

    if (GraphWindow.current.mapMagic != null)
     using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(ref GraphWindow.current.mapMagic.Globals.obje
   }

 Cell.EmptyRowPx(2);
```

```
  DrawPositioningSettings(gen.posSettings, billboardRotWaring:true);
 }
}


[Draw.Editor(typeof(ObjectsGenerators.TreesOutput))]
public static void DrawTreesOutput (ObjectsGenerators.TreesOutput gen)
{
 if (gen.posSettings == null) gen.posSettings = TreesOutput.CreatePosSettings(gen);

 using (Cell.LineStd)
  DrawObjectPrefabs(ref gen.prefabs, gen.guiMultiprefab, treeIcon:true);

 using (Cell.LinePx(0))
  using (Cell.Padded(2,2,0,0))
 {
  using (Cell.LineStd) Draw.ToggleLeft(ref gen.guiMultiprefab, "Multi-Prefab");

  Cell.EmptyRowPx(4);

  using (Cell.LinePx(0))
   using (new Draw.FoldoutGroup(ref gen.guiProperties, "Properties"))
    if (gen.guiProperties)
    {
     Cell.current.fieldWidth = 0.481f;
     using (Cell.LineStd) Draw.Field(ref gen.seed, "Seed");
```

```
    using (Cell.LineStd) Draw.Field(ref gen.color, "Color");
    using (Cell.LineStd) Draw.Field(ref gen.lightmapColor, "Lightmap");
    using (Cell.LineStd) gen.bendFactor = Draw.Field(gen.bendFactor, "Bend Factor");
    using (Cell.LineStd) Draw.Field(ref gen.biomeBlend, "Biome Blend");
   }

  Cell.EmptyRowPx(2);
  DrawPositioningSettings(gen.posSettings, billboardRotWaring:true);
 }
}



[Draw.Editor(typeof(ObjectsGenerators.Rarefy200))]
public static void DrawRarefyGenerator (ObjectsGenerators.Rarefy200 gen)
{
 using (Cell.LinePx(0))
  using (Cell.Padded(1,1,0,0))
 {
  using (Cell.LineStd) Draw.Field(ref gen.distance, "Distance");
  using (Cell.LineStd) Draw.Field(ref gen.sizeFactor, "Size Factor");
  using (Cell.LineStd) Draw.Toggle(ref gen.self, "Use Self");
 }

 using (Cell.LinePx(0))
 LayersEditor.DrawLayers(ref gen.layers,
  onDraw: num =>
```

```
{
 if (num>=gen.layers.Length) return; //on layer remove
 int iNum = gen.layers.Length-1 - num;

 Cell.EmptyLinePx(2);
 using (Cell.LineStd)
 {
  using (Cell.RowPx(0))
   GeneratorDraw.DrawInlet(gen.layers[iNum].inlet, gen);
  Cell.EmptyRowPx(10);
  using (Cell.RowPx(20)) Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Layer") );

  using (Cell.Row)
  {
   using (Cell.RowPx(43)) Draw.Label("Dist");
   using (Cell.Row) Draw.FieldDragIcon(ref gen.layers[iNum].distance);

   Cell.current.Expose(gen.layers[iNum].id, "distance", typeof(float));
  }


  Cell.EmptyRowPx(2);
 }
 Cell.EmptyLinePx(2);
},
onCreate: num => new ObjectsGenerators.Rarefy200.Layer(gen) );
```

```
}



[Draw.Editor(typeof(ObjectsGenerators.Positions200))]
public static void DrawPositionsGenerator (ObjectsGenerators.Positions200 gen)
{
 using (Cell.LinePx(0))
 LayersEditor.DrawLayers(ref gen.positions,
  onDraw: num =>
  {
   if (num>=gen.positions.Length) return; //on layer remove
   int iNum = gen.positions.Length-1 - num;

   Cell.EmptyLinePx(2);
   using (Cell.LineStd)
   {
    Cell.current.fieldWidth = 0.7f;
    Cell.EmptyRowPx(2);
    using (Cell.RowPx(15)) Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Layer") );
    using (Cell.Row)
    {
     using (Cell.LineStd) Draw.Field(ref gen.positions[num].x, "X");
     using (Cell.LineStd) Draw.Field(ref gen.positions[num].y, "Y");
     using (Cell.LineStd) Draw.Field(ref gen.positions[num].z, "Z");

     Cell.current.Expose(gen.id, "positions", typeof(Vector3), arrIndex:num);
```

```
    }
    Cell.EmptyRowPx(2);
   }
   Cell.EmptyLinePx(2);
  } );
}


[Draw.Editor(typeof(ObjectsGenerators.Combine200))]
public static void DrawCombineGenerator (ObjectsGenerators.Combine200 gen)
{
 using (Cell.LinePx(0))
 LayersEditor.DrawLayers(ref gen.inlets,
  onDraw: num =>
  {
   if (num>=gen.inlets.Length) return; //on layer remove
   int iNum = gen.inlets.Length-1 - num;

   Cell.EmptyLinePx(2);
   using (Cell.LineStd)
   {
    using (Cell.RowPx(0))
     GeneratorDraw.DrawInlet(gen.inlets[iNum], gen);
    Cell.EmptyRowPx(10);
    using (Cell.RowPx(15)) Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Layer") );
    using (Cell.Row) Draw.Label("Layer " + iNum);
```

```
    }
    Cell.EmptyLinePx(2);
  },
  onCreate: num => new Inlet<TransitionsList>() );
}


[Draw.Editor(typeof(ObjectsGenerators.Stamp200))]
public static void DrawStampGenerator (ObjectsGenerators.Stamp200 gen)
{
 //drawing standard class values

 using (Cell.Padded(1,1,0,0))
 {
  Cell.current.fieldWidth = 0.4f;

  //Draw.Class(gen, "Custom");
  CellExpose.ExposableClass(gen, gen.id, "Custom");

  /*using (Cell.LineStd) Draw.Field(ref gen.size, "Size");
  using (Cell.LineStd) Draw.Field(ref gen.intensity, "Intensity");
  using (Cell.LineStd) Draw.Field(ref gen.sizeFactor, "Size Factor");
  using (Cell.LineStd) Draw.Field(ref gen.intensityFactor, "Intensity Factor");
  using (Cell.LineStd) Draw.Field(ref gen.blendType, "Blend Type");

  using (Cell.LineStd) Draw.ToggleLeft(ref gen.useRotation, "Use Rotation");
```

```
  using (Cell.LineStd) Draw.ToggleLeft(ref gen.useFalloff, "Use Falloff");*/


  if (gen.useFalloff)
  // using (Cell.LineStd) Draw.Field(ref gen.hardness, "Hardness");
  // Draw.Class(gen, "UseFallof");
   CellExpose.ExposableClass(gen, gen.id, "UseFallof");


  CheckDrawRadiusWarning(gen.size);
 }
}



[Draw.Editor(typeof(ObjectsGenerators.GetByTag211))]
public static void DrawTagGenerator(ObjectsGenerators.GetByTag211 gen)
{
 //drawing standard class values


 using (Cell.Padded(1, 1, 0, 0))
 {
  //Cell.current.fieldWidth = 0.4f;


  using (Cell.LineStd)
  {
   using (Cell.RowRel(1 - Cell.current.fieldWidth))
    Draw.Label("Tag");
```

```
      using (Cell.RowRel(Cell.current.fieldWidth))
       gen.tag = Draw.Field(gen.tag, drawFn: (Rect rect, string oldVal) => { return UnityEditor.EditorGUI.TagF
     }


     using (Cell.LineStd)
      Draw.Field(ref gen.additionalMargins, "Add. Margins");
    }
   }
  }
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Products;

namespace MapMagic.Nodes.ObjectsGenerators
{
 [System.Serializable]
 [GeneratorMenu (menu="Objects/Initial", name ="Positions", iconName="GeneratorIcons/Position", diseng
 public class Positions200 : Generator, IOutlet<TransitionsList>
 {
  public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

  public Vector3[] positions= new Vector3[1];

  public override void Generate (TileData data, StopToken stop)
  {
   if (!enabled) return;

   TransitionsList trns = new TransitionsList();
   for (int p=0; p<positions.Length; p++)
```

```
  {
    Transition trs = new Transition(positions[p].x, positions[p].y, positions[p].z);
    trns.Add(trs);
  }

  data.StoreProduct(this, trns);
 }

}


[System.Serializable]
[GeneratorMenu (menu="Objects/Initial",
 name ="Scatter",
 iconName="GeneratorIcons/Scatter",
 disengageable = true,
 advancedOptions = true,
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/ObjectsGenerators/Scatter")]
public class Scatter200 : Generator, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Seed")]   public int seed = 12345;
 //      public enum Algorithm { Random, SquareCells, HexCells };
 //[Val("Algorithm")]  public Algorithm algorithm = Algorithm.SquareCells;
 [Val("Density")]  public float density = 10; //doesn't guarantee that chunk 1km*1km will have this number o
```

```
[Val("Uniformity")]  public float uniformity = 0.1f;


[Val("Relax", "Advanced")] public float relax = 0.5f;
[Val("Add.Margin", "Advanced")] public float additionalMargins = 0;


public override void Generate (TileData data, StopToken stop)
{
 if (!enabled) return;
 if (density==0) { data.StoreProduct(this, new TransitionsList()); return; }


 Noise random = new Noise(data.random, seed);
 TransitionsList trns = Scatter(
  (Vector3)data.area.full.worldPos - new Vector3(additionalMargins, 0, additionalMargins),
  (Vector3)data.area.full.worldSize + new Vector3(additionalMargins*2, 0, additionalMargins*2),
  random);


 data.StoreProduct(this, trns);
}


public TransitionsList Scatter (Vector3 worldPos, Vector3 worldSize, Noise random)
{
 float cellSize = 1000 / Mathf.Sqrt(density);


 CoordRect rect = CoordRect.WorldToPixel((Vector2D)worldPos, (Vector2D)worldSize, (Vector2D)cellSiz
 worldPos = new Vector3(rect.offset.x*cellSize, worldPos.y, rect.offset.z*cellSize);  //modifying worldPos/s
 worldSize = new Vector3(rect.size.x*cellSize, worldSize.y, rect.size.z*cellSize);
```

```
  rect.offset -= 1; rect.size += 2; //leaving 1-cell margins
  worldPos.x -= cellSize; worldPos.z -= cellSize;
  worldSize.x += cellSize*2; worldSize.z += cellSize*2;

  PositionMatrix posMatrix = new PositionMatrix(rect, worldPos, worldSize);
  posMatrix.Scatter(uniformity, random);
  posMatrix = posMatrix.Relaxed(relax);

  //DebugGizmos.DrawCoordRect("posMatrix", posMatrix.rect,  posMatrix.worldPos, posMatrix.worldSize);

  return posMatrix.ToTransitionsList();
 }
}


[System.Serializable]
[GeneratorMenu (menu="Objects/Initial", name ="Random", iconName="GeneratorIcons/Random", diseng
 colorType = typeof(TransitionsList),
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/ObjectsGenerators/Random")]
public class Random207 : Generator, IInlet<MatrixWorld>, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Seed")]   public int seed = 12345;
 [Val("Density")]  public float density = 10;
```

```
[Val("Uniformity")]  public float uniformity = 0.1f;



public override void Generate (TileData data, StopToken stop)
{
 if (!enabled) return;
 MatrixWorld probMatrix = data.ReadInletProduct(this);

 Noise random = new Noise(data.random, seed);

 float square = data.area.active.worldSize.x * data.area.active.worldSize.z; //note using the real size, dens
 float count = square*(density/1000000); //number of items per terrain

 PosTab posTab = new PosTab((Vector3)data.area.full.worldPos, (Vector3)data.area.full.worldSize, 16);
 RandomScatter((int)count, uniformity, (Vector3)data.area.full.worldPos, (Vector3)data.area.full.worldSize
 TransitionsList transitions = posTab.ToTransitionsList();

 data.StoreProduct(this, transitions);
}



public static void RandomScatter (int count, float uniformity, Vector3 offset, Vector3 size, PosTab posTab
{
 int candidatesNum = (int)(uniformity*100);
 if (candidatesNum < 1) candidatesNum = 1;
```

```
for (int i=0; i<count; i++)
{
 if (stop!=null && stop.stop) return;

 float bestCandidateX = 0;
 float bestCandidateZ = 0;
 float bestDist = 0;

 for (int c=0; c<candidatesNum; c++)
 {
  float candidateX = (offset.x+1) + (rnd.Random((int)posTab.pos.x, (int)posTab.pos.z, i*candidatesNum+c
  float candidateZ = (offset.z+1) + (rnd.Random((int)posTab.pos.x, (int)posTab.pos.z, i*candidatesNum+c

  //checking if candidate is the furthest one
  Transition closest = posTab.Closest(candidateX, candidateZ, minDist:0.001f);
  float dist = (closest.pos.x-candidateX)*(closest.pos.x-candidateX) + (closest.pos.z-candidateZ)*(closest.

  //distance to the edge
  float bd = (candidateX-offset.x)*2; if (bd*bd < dist) dist = bd*bd;
  bd = (candidateZ-offset.z)*2; if (bd*bd < dist) dist = bd*bd;
  bd = (offset.x+size.x-candidateX)*2; if (bd*bd < dist) dist = bd*bd;
  bd = (offset.z+size.z-candidateZ)*2; if (bd*bd < dist) dist = bd*bd;

  //probability
  if (prob != null)
  {
```

```
      float probValue = prob.GetWorldInterpolatedValue(candidateX, candidateZ);
      dist *= probValue;
     }

     if (dist>bestDist) { bestDist=dist; bestCandidateX = candidateX; bestCandidateZ = candidateZ; }
    }

    if (bestDist>0.001f) //adding only if some suitable candidate found
    {
     Transition trs = new Transition(bestCandidateX, bestCandidateZ);
     posTab.Add(trs);
    }
   }
   posTab.Flush();
  }
}

[System.Serializable]
[GeneratorMenu (menu=null, name ="Random", iconName="GeneratorIcons/Random", disengageable = t
 colorType = typeof(TransitionsList),
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/ObjectsGenerators/Random")]
public class Random200 : Random207,  IInlet<MatrixWorld>
{
 //outdated, but has inlet and could not be removed
}
```

```
[System.Serializable]
[GeneratorMenu(menu = "Objects/Initial", name = "Get by Tag", iconName = "GeneratorIcons/Position", di
 colorType = typeof(TransitionsList),
 //advancedOptions = true,
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/ObjectsGenerators/GetByTag")]
public class GetByTag211 : Generator, IOutlet<TransitionsList>, IPrepare
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 public string tag;
 [Val("Add.Margin", "Advanced")] public float additionalMargins = 0;

 public void Prepare (TileData data, Terrain terrain)
 {
  GameObject[] objs = GameObject.FindGameObjectsWithTag(tag);

  Vector3[] poses = objs.Select(p => p.transform.position);
  Quaternion[] rotations = objs.Select(p => p.transform.rotation);
  Vector3[] scales = objs.Select(p => p.transform.localScale);
  (Vector3[], Quaternion[], Vector3[]) result = (poses, rotations, scales);

  data.StorePrepare(id, result);
 }

 public override void Generate(TileData data, StopToken stop)
```

```
{
 if (!enabled) return;

 Vector3 rectPos = (Vector3)data.area.full.worldPos - new Vector3(additionalMargins, 0, additionalMargin
 Vector3 rectSize = (Vector3)data.area.full.worldSize + new Vector3(additionalMargins * 2, 0, additionalM
 Vector3 min = rectPos;
 Vector3 max = rectPos + rectSize;

 TransitionsList trns = new TransitionsList();

 (Vector3[] p, Quaternion[] r, Vector3[] s) readResult = ((Vector3[], Quaternion[], Vector3[]))data.ReadPrep
 Vector3[] poses = readResult.p;
 Quaternion[] rotations = readResult.r;
 Vector3[] scales = readResult.s;

for (int i=0; i<poses.Length; i++)
{
  Vector3 pos = poses[i];

  if (pos.x > min.x && pos.z > min.z &&
   pos.x < max.x && pos.z < max.z)
   {
    Transition trs = new Transition(pos.x, pos.z);
    trs.rotation = rotations[i];
    trs.scale = scales[i];
```

```
        trns.Add(trs);
      }
    }

    data.StoreProduct(this, trns);
  }
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Terrains;
using MapMagic.Nodes;
using MapMagic.Nodes.ObjectsGenerators;

namespace MapMagic.Locks
{
 public class ObjectsData : ILockData
 {
  //public ObjectsPool.Prototype[] lockPrototypes;
  //public List<Transition>[] lockTransitions;
  //do not use them, but clearing a glade to place locked objects

  public Transform lockParent;
  public Dictionary<GameObject,Vector3> lockedObjsPoses = new Dictionary<GameObject,Vector3>(); //lo
  public Dictionary<GameObject,Vector3> adjustedObjsPoses = new Dictionary<GameObject,Vector3>(); /

  public Vector2D center; //local terrain coordsys
  public float radius;
  public float transition;
```

```
public float terrainSize;
public float terrainHeight;

Vector2D min; //terrain local too
Vector2D max;

bool heightChanged;



public void Read (Terrain terrain, Lock lk)
{
 if (terrain.name == "Draft Terrain") return;

 Vector2D terrainPos = (Vector2D)terrain.transform.position;

 center = (Vector2D)lk.worldPos - terrainPos;
 radius = lk.worldRadius;
 transition = lk.worldTransition;
 terrainSize = terrain.terrainData.size.x;
 terrainHeight = terrain.terrainData.size.y;

 min = center - radius - transition;
 max = center + radius + transition;

 //finding locked parent if it exists
 string lockParentName = $"LockedObjects {lk.guiName}";
```

```
if (lockParent == null)
{
 Transform tileTfm = terrain.transform.parent;

 int tileChildCount = tileTfm.childCount;
 for (int c=0; c<tileChildCount; c++)
 {
  Transform child = tileTfm.GetChild(c);
  if (child.name == lockParentName || //name match
   ((Vector2D)child.localPosition == center  &&  child.name.StartsWith("LockedObject")))  //or position ma
    { lockParent = child; break; }
 }
}

//creating locked objects parent if it wasn't found
if (lockParent == null)
{
 GameObject go = new GameObject();
 go.transform.parent = terrain.transform.parent;
 go.transform.localPosition = (Vector3)center;
 lockParent = go.transform;
}

lockParent.gameObject.name = lockParentName;

//filling objects list
```

```
lockedObjsPoses.Clear();
adjustedObjsPoses.Clear();
int childCount = lockParent.childCount;
for (int i=0; i<childCount; i++)
{
 Transform child = lockParent.GetChild(i);
 lockedObjsPoses.Add(child.gameObject, child.position - (Vector3)terrainPos);
}

//de-pooling objects in range from the pool
ObjectsPool pool = terrain.transform.parent.GetComponentInChildren<ObjectsPool>();
if (pool != null)
{
 for (int i=pool.transform.childCount-1; i>=0; i--)
 {
  Transform child = pool.transform.GetChild(i);

  Vector3 pos = child.localPosition;
  if (pos.x < min.x || pos.x > max.x ||
   pos.z < min.z || pos.z > max.z) continue;

  float dist = (center.x-pos.x)*(center.x-pos.x) + (center.z-pos.z)*(center.z-pos.z);

  if (dist < (radius+transition)*(radius+transition))
  {
   pool.Depool(child.gameObject);
```

```
    if (dist < radius*radius) //adding only objects within lock center
    {
     lockedObjsPoses.Add(child.gameObject, child.transform.position - (Vector3)terrainPos);
     child.parent = lockParent;
    }
    else
     GameObject.DestroyImmediate(child.gameObject);
   }
  }
 }

 //ensuring all objects are parented to lockParent
 foreach (GameObject obj in lockedObjsPoses.Keys)
  obj.transform.parent = lockParent;
  //if (obj.transform.parent == lockParent)
  // throw new Exception("Does not belong to lock parent");
}


public void WriteInThread (IApplyData applyData)
{
 if (! (applyData is ObjectsOutput.ApplyObjectsData applyObjsData) ) return;

 //clearing a glade to place locked objs
 for (int p=0; p<applyObjsData.transitions.Length; p++)
```

```
 {
  List<Transition> transitionList = applyObjsData.transitions[p];
  int transitionsCount = transitionList.Count;
  for (int t=transitionList.Count-1; t>=0; t--)
  {
   Vector3 pos = transitionList[t].pos;
   if (pos.x < min.x || pos.x > max.x ||
    pos.z < min.z || pos.z > max.z) continue;

   float dist = (center.x-pos.x)*(center.x-pos.x) + (center.z-pos.z)*(center.z-pos.z);

   if (dist < (radius+transition)*(radius+transition))
    transitionList.RemoveAt(t);
  }
 }
}

public void WriteInApply (Terrain terrain, bool resizeTerrain=false)
{
 if (!heightChanged) return;
 if (terrain.name == "Draft Terrain") return;

 Dictionary<GameObject,Vector3> lockedPoeseDict = adjustedObjsPoses.Count == 0 ?
  lockedObjsPoses : adjustedObjsPoses;
  //using the adjusted poses dict if height has been applied
```

```
 int childCount = lockParent.childCount;
 for (int i=0; i<childCount; i++)
 {
  Transform child = lockParent.GetChild(i);

  if (lockedPoeseDict.TryGetValue(child.gameObject, out Vector3 pos))
   child.position = new Vector3(child.position.x, pos.y, child.position.z);
 }

 heightChanged = false;
}


public void ApplyHeightDelta (Matrix srcHeights, Matrix dstHeights)
{
 heightChanged = true;

 Vector2D relRectStart = center-(radius+transition);
 float relRectSize = radius*2+transition*2;

 adjustedObjsPoses.Clear();

 foreach (var kvp in lockedObjsPoses)
 {
  Vector3 pos = kvp.Value;
  GameObject obj = kvp.Key;
```

```
  Vector2D relPos = ((Vector2D)pos-center) / relRectSize;  //relative to lock circle
  relPos += 0.5f; //relative to lock matrix
  Vector2D matrixPos = new Vector2D(
   relPos.x*srcHeights.rect.size.x + srcHeights.rect.offset.x,
   relPos.z*srcHeights.rect.size.z + srcHeights.rect.offset.z);

  float hSrc = srcHeights.GetRelative(relPos.x, relPos.z);
  float hDst = dstHeights.GetRelative(relPos.x, relPos.z);
  float heightDelta = hDst-hSrc;

  pos.y += heightDelta*terrainHeight;

  adjustedObjsPoses.Add(obj, pos);
 }
}


public void ResizeFrom (ILockData src) {  }


private static void EnlistObjsWithinRange (Vector3 center, float range, Transform parent, List<(GameObje
{
 int childCount = parent.childCount;
 for (int i=0; i<childCount; i++)
 {
```

```
    Transform child = parent.GetChild(i);
    Vector3 pos = child.position;
    if (pos.x < center.x-range || pos.x > center.x+range ||
     pos.z < center.z-range || pos.z > center.z+range) continue;

    float dist = Mathf.Sqrt( (center.x-pos.x)*(center.x-pos.x) + (center.z-pos.z)*(center.z-pos.z) );

    if (dist < range)
    {
     GameObject childObj = child.gameObject;
     //if (!list.Contains(childObj))
      list.Add( (childObj, childObj.transform.position.y) );
    }
   }
  }
 }

}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Products;

namespace MapMagic.Nodes.ObjectsGenerators
{
 [System.Serializable]
 [GeneratorMenu (menu="Objects/Modifiers", name ="Adjust", iconName="GeneratorIcons/Adjust", diseng
 public class Adjust200 : Generator, IMultiInlet, IOutlet<TransitionsList>
 {
  public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

  [Val("Input", "Inlet")]  public readonly Inlet<TransitionsList> input = new Inlet<TransitionsList>();
  [Val("Intensity", "Inlet")] public readonly Inlet<MatrixWorld> intensityIn = new Inlet<MatrixWorld>();
  public IEnumerable<IInlet<object>> Inlets () { yield return input; yield return intensityIn; }

  public bool useRandom = false;
  public int seed = 12345;
  public float sizeFactor = 0;
```

```
public enum Relativeness { absolute, relative };

public Relativeness relativeness = Relativeness.relative;


//public enum Randomness { equally, range };

//public Randomness randomness = Randomness.equally;


public Vector2 offsetFront = Vector2.zero;

public Vector2 offsetRight = Vector2.zero;

public Vector2 height = Vector2.zero;  //x is min, y is max

public Vector2 rotation = Vector2.zero;

public Vector2 scale = Vector2.one;



public override void Generate (TileData data, StopToken stop)

{

 TransitionsList src = data.ReadInletProduct(input);

 if (src == null) return;

 if (!enabled) { data.StoreProduct(this, src); return; }


 TransitionsList dst = new TransitionsList(src);


 MatrixWorld intensityMatrix = data.ReadInletProduct(intensityIn);

 Noise rnd = useRandom ? new Noise(data.random, seed) : null;


 for (int t=0; t<dst.count; t++)

 {
```

```
  if (stop!=null && stop.stop) return;
  Adjust(ref dst.arr[t], intensityMatrix, rnd);
 }

 data.StoreProduct(this, dst);
}


public void Adjust (ref Transition trn, MatrixWorld intensityMatrix, Noise rnd)
{
 //generating random vals
 float heightRnd, rotRnd, scaleRnd, frontRnd, rightRnd;
 if (rnd != null)
 {
  heightRnd = rnd.Random(trn.hash, 0);
  heightRnd = height.x + heightRnd*(height.y-height.x);

  rotRnd = rnd.Random(trn.hash, 1);
  rotRnd = rotation.x + rotRnd*(rotation.y-rotation.x);

  scaleRnd = rnd.Random(trn.hash, 2);
  scaleRnd = scale.x + scaleRnd*(scale.y-scale.x);

  frontRnd = rnd.Random(trn.hash, 3); //goes last for compatibility random
  frontRnd = scale.x + frontRnd*(offsetFront.y-offsetFront.x);
```

```
 rightRnd = rnd.Random(trn.hash, 3);
 rightRnd = scale.x + frontRnd*(offsetRight.y-offsetRight.x);
}
else
{
 heightRnd = height.x;
 rotRnd = rotation.x;
 scaleRnd = scale.x;
 frontRnd = offsetFront.x;
 rightRnd = offsetRight.x;
}

//calculating intensity
float intensity = 1;
if (intensityMatrix != null)
{
 if (!intensityMatrix.ContainsWorldValue(trn.pos.x, trn.pos.z)) intensity = 0;
 else intensity = intensityMatrix.GetWorldValue(trn.pos.x, trn.pos.z);
}



if (relativeness == Relativeness.relative)
{
 //scale is not affected by sizeFactor
 trn.scale *= scaleRnd * intensity;
```

```
    //everything else does
    intensity = intensity*(1-sizeFactor) + intensity*trn.scale.x*sizeFactor;

    (Vector2D front, Vector2D right) = trn.FrontRight2D;
    trn.pos += (Vector3)front * frontRnd * intensity;
    trn.pos += (Vector3)right * rightRnd * intensity;
    trn.pos.y += heightRnd * intensity; // / height; //not multiplying in output
    trn.Yaw = trn.Yaw  +  rotRnd * intensity;
   }
   else
   {
    trn.scale = new Vector3(1,1,1) * scaleRnd * intensity;

    intensity = intensity*(1-sizeFactor) + intensity*trn.scale.x*sizeFactor;

    (Vector2D front, Vector2D right) = trn.FrontRight2D;
    trn.pos += (Vector3)front * frontRnd * intensity;
    trn.pos += (Vector3)right * rightRnd * intensity;
    trn.pos.y = heightRnd * intensity; // / height;
    trn.Yaw = rotRnd * intensity;
   }
  }
 }

[System.Serializable]
```

```
[GeneratorMenu(menu = "Objects/Modifiers", name = "Randomize", iconName = "GeneratorIcons/Random
helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/ObjectsGenerators/Randomize")]
public class Randomize211 : Generator, IInlet<TransitionsList>, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Seed")] public int seed = 12345;

 public override void Generate(TileData data, StopToken stop)
 {
  TransitionsList src = data.ReadInletProduct(this);
  if (src == null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }

  TransitionsList dst = new TransitionsList(src);
  Noise random = new Noise(data.random, seed);

  for (int t = 0; t < dst.count; t++)
  {
   if (stop != null && stop.stop) return;
   dst.arr[t].id = t;
   dst.arr[t].hash = random.RandomInt(t);
  }

  data.StoreProduct(this, dst);
 }
```

```
}



[System.Serializable]
[GeneratorMenu (menu="Objects/Modifiers", name ="Mask", iconName="GeneratorIcons/ObjectsMask", d
public class Mask200 : Generator, IMultiInlet, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Masked", "Inlet")] public readonly Inlet<TransitionsList> srcIn = new Inlet<TransitionsList>();
 [Val("Unmasked", "Inlet")] public readonly Inlet<TransitionsList> invIn = new Inlet<TransitionsList>();
 [Val("Mask", "Inlet")] public readonly Inlet<MatrixWorld> maskIn = new Inlet<MatrixWorld>();
 public IEnumerable<IInlet<object>> Inlets () { yield return srcIn; yield return invIn; yield return maskIn; }

 [Val("Seed")] public int seed = 12345;
 [Val("Invert")] public bool invert = false;



 public override void Generate (TileData data, StopToken stop)
 {
  TransitionsList src = data.ReadInletProduct(srcIn);
  TransitionsList inv = data.ReadInletProduct(invIn);
  if (src == null  &&  inv == null) return;
  if (!enabled) { data.StoreProduct(this, src ?? inv); return; }

  MatrixWorld mask = data.ReadInletProduct(maskIn);
```

```
 if (mask == null) { data.StoreProduct(this, src); return; }

 TransitionsList dst = new TransitionsList();
 Noise random = new Noise(data.random, seed);

 if (src != null) Mask(src, dst, mask, random, invert, stop);
 if (inv != null) Mask(inv, dst, mask, random, !invert, stop);

 data.StoreProduct(this, dst);
}


public static void Mask (TransitionsList src, TransitionsList dst, MatrixWorld mask, Noise random, bool inv
{
 for (int t=0; t<src.count; t++)
 {
  if (stop!=null && stop.stop) return;

  Vector3 pos = src.arr[t].pos;

  if (pos.x <= mask.worldPos.x  ||   pos.x >= mask.worldPos.x+mask.worldSize.x  ||
   pos.z <= mask.worldPos.z  ||   pos.z >= mask.worldPos.z+mask.worldSize.z)
    continue; //do remove out of range objects?

  float val = mask.GetWorldValue(pos.x, pos.z);
  float rnd = random.Random(src.arr[t].hash);
```

```
   if (val<rnd && invert) dst.Add(src.arr[t]);
   if (val>=rnd && !invert) dst.Add(src.arr[t]);
  }
 }
}
```

```
[System.Serializable]
[GeneratorMenu (menu="Objects/Modifiers", name ="Lerp", iconName="GeneratorIcons/ObjectsMask", di
public class Lerp210 : Generator, IMultiInlet, IOutlet<TransitionsList>
/// Interpolates positions,rotations,scales of objects with the same id
/// Adds to result only objects with same ids
/// Designed to use instead Move's intensity mask, but what is the purpose if it's done the same? However
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Original", "Inlet")] public readonly Inlet<TransitionsList> srcIn = new Inlet<TransitionsList>();
 [Val("Modified", "Inlet")] public readonly Inlet<TransitionsList> modIn = new Inlet<TransitionsList>();
 [Val("Mask", "Inlet")]  public readonly Inlet<MatrixWorld> maskIn = new Inlet<MatrixWorld>();
 public virtual IEnumerable<IInlet<object>> Inlets () { yield return srcIn; yield return modIn; yield return mas

 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
```

```
 TransitionsList src = data.ReadInletProduct(srcIn);  if (src == null) return;
 TransitionsList mod = data.ReadInletProduct(modIn);  if (mod == null) return;
 MatrixWorld intensity = data.ReadInletProduct(maskIn);
  if (!enabled || intensity == null) { data.StoreProduct(this, mod); return; }

 if (stop!=null && stop.stop) return;
 TransitionsList dst = new TransitionsList(src.count);
 Lerp(src, mod, dst, intensity);

 if (stop!=null && stop.stop) return;
 data.StoreProduct(this, dst);
}

public void Lerp (TransitionsList list1, TransitionsList list2, TransitionsList dst, MatrixWorld mask)
{
 Dictionary<int,Transition> idToList1 = new Dictionary<int,Transition>();
 for (int t=0; t<list1.count; t++)
  idToList1.Add(list1.arr[t].id, list1.arr[t]);

 for (int t=0; t<list2.count; t++)
 {
  Transition trs2 = list2.arr[t];

  Transition trs1;
  if (!idToList1.TryGetValue(trs2.id, out trs1))
   continue;
```

```
   if (trs1.pos.x <= mask.worldPos.x  ||  trs1.pos.x >= mask.worldPos.x+mask.worldSize.x  ||
    trs1.pos.z <= mask.worldPos.z  ||  trs1.pos.z >= mask.worldPos.z+mask.worldSize.z)
     continue;

   float percent = mask.GetWorldInterpolatedValue(trs1.pos.x, trs1.pos.z); //or use trs2 or medium

   trs1.pos = trs1.pos*(1-percent) + trs2.pos*percent;
   trs1.rotation = Quaternion.Lerp(trs1.rotation, trs2.rotation, percent);
   trs1.scale = trs1.scale*percent + trs2.scale*(1-percent);

   dst.Add(trs1);
  }
 }
}



[System.Serializable]
[GeneratorMenu (menu="Objects/Modifiers", name ="Floor", iconName=null, disengageable = true, helpLi
public class Floor200 : Generator, IMultiInlet, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Input", "Inlet")]  public readonly Inlet<TransitionsList> srcIn = new Inlet<TransitionsList>();
 [Val("Height", "Inlet")] public readonly Inlet<MatrixWorld> heightIn = new Inlet<MatrixWorld>();
```

```
public enum Relativity{ absolute, relative };
[Val("Relativity")] public Relativity relativity = Relativity.absolute;

public IEnumerable<IInlet<object>> Inlets () { yield return srcIn; yield return heightIn; }


public override void Generate (TileData data, StopToken stop)
{
 TransitionsList src = data.ReadInletProduct(srcIn);
 if (src == null) return;
 if (!enabled) { data.StoreProduct(this, src); return; }

 MatrixWorld heights = data.ReadInletProduct(heightIn);

 TransitionsList dst = new TransitionsList(src);
 for (int t=0; t<dst.count; t++)
 {
  if (stop!=null && stop.stop) return;
  Floor(ref dst.arr[t], heights);
 }

 data.StoreProduct(this, dst);
}
```

```
 public void Floor (ref Transition trn, MatrixWorld heights)
 {
  if (heights == null)
   { trn.pos.y = 0; return; }

  if (trn.pos.x <= heights.worldPos.x  ||  trn.pos.x >= heights.worldPos.x +heights.worldSize.x ||
   trn.pos.z <= heights.worldPos.z  ||  trn.pos.z >= heights.worldPos.z +heights.worldSize.z)
    return;

  float terrainHeight = heights.GetWorldInterpolatedValue(trn.pos.x, trn.pos.z);
  if (terrainHeight > 1) terrainHeight = 1;
  terrainHeight *= heights.worldSize.y;

  if (relativity == Relativity.relative)
   trn.pos.y += terrainHeight;
  else
   trn.pos.y = terrainHeight;
 }
}


[System.Serializable]
[GeneratorMenu (menu="Objects/Modifiers", name ="Split", iconName="GeneratorIcons/Split", colorType
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/ObjectsGenerators/Split")]
public class Split200 : Generator, IInlet<TransitionsList>, IMultiLayer
{
```

```
public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on


public class SplitLayer : IOutlet<TransitionsList>
{
 public string name = "Object Layer";

 public bool heightConditionActive = false;
 public Vector2 heightCondition = new Vector2(0,1);

 public bool rotationConditionActive = false;
 public Vector2 rotationCondition = new Vector2(0,360);

 public bool scaleConditionActive = false;
 public Vector2 scaleCondition = new Vector2(0,100);

 public float chance = 1;

 public Generator Gen { get; private set; }
 public void SetGen (Generator gen) => Gen=gen;

 public ulong id; //properties not serialized
 public ulong Id { get{return id;} set{id=value;} }

 public IUnit ShallowCopy() => (SplitLayer)this.MemberwiseClone();
}
```

```
public SplitLayer[] layers = new SplitLayer[0];

public IList<IUnit> Layers { get => layers; set {return;} } //don't set anything
public void SetLayers(object[] ls) => layers = Array.ConvertAll(ls, i=>(SplitLayer)i);
public bool Inversed => true;
public bool HideFirst => false;

public int guiExpanded = -1;

public enum MatchType { layered, random };
[Val("Match")] public MatchType matchType = MatchType.random;
[Val("Seed")] public int seed = 12345;

public override void Generate (TileData data, StopToken stop)
{
 if (stop!=null && stop.stop) return;
 TransitionsList src = data.ReadInletProduct(this);
 if (src == null) return;
 if (!enabled) return;

 Noise random = new Noise(data.random, seed);
 bool[] match = new bool[layers.Length];

 //creating dst
 TransitionsList[] dst = new TransitionsList[layers.Length];
 for (int i=0; i<dst.Length; i++)
```

```
  dst[i] = new TransitionsList();

 //splitting
 for (int t=0; t<src.count; t++)
 {
  if (stop!=null && stop.stop) return;

  int layerNum = PickObjLayer(src.arr[t], random, match);
  if (layerNum >= 0)
   dst[layerNum].Add(src.arr[t]);
 }

 //setting results
 for (int i=0; i<dst.Length; i++)
 {
  if (stop!=null && stop.stop) return;
  data.StoreProduct(layers[i], dst[i]);
 }
}

private int PickObjLayer (Transition trs, Noise random, bool[] match=null)
{
 if (match==null) match = new bool[layers.Length];

 //finding suitable objects (and sum of chances btw. And last object for non-random)
 int matchesNum = 0; //how many layers have a suitable obj
```

```
float chanceSum = 0;
int lastLayerNum = 0;

for (int i=0; i<layers.Length; i++)
{
 SplitLayer layer = (SplitLayer)layers[i];
 float yaw = (trs.Yaw+360) % 360;

 bool heightMatch = !layer.heightConditionActive || (trs.pos.y >= layer.heightCondition.x && trs.pos.y <= l
 bool rotationMatch = !layer.rotationConditionActive || (yaw >= layer.rotationCondition.x && yaw <= layer.
 bool scaleMatch = !layer.scaleConditionActive || (trs.scale.x >= layer.scaleCondition.x && trs.scale.x <=

 if (heightMatch && rotationMatch && scaleMatch)
 {
  match[i] = true;

  matchesNum ++;
  chanceSum += layer.chance;
  lastLayerNum = i;
 }
 else match[i] = false;
}

//if no matches detected - continue withous assigning obj
if (matchesNum == 0) return -1;
```

```
  //if one match - assigning last obj
  else if (matchesNum == 1 || matchType == MatchType.layered) return lastLayerNum;

  //selecting layer at random
  else if (matchesNum > 1 && matchType == MatchType.random)
  {
   float randomVal = random.Random(trs.hash);
   randomVal *= chanceSum;
   chanceSum = 0;

   for (int i=0; i<layers.Length; i++)
   {
    if (!match[i]) continue;

    SplitLayer layer = (SplitLayer)layers[i];
    if (randomVal > chanceSum  &&  randomVal < chanceSum + layer.chance) return i;
    chanceSum += layer.chance;
   }
  }

  return -1;
 }
}

[System.Serializable]
```

```
[GeneratorMenu (menu="Objects/Modifiers", name ="Rarefy", iconName="GeneratorIcons/Rarefy", diseng
public class Rarefy200 : Generator, IInlet<TransitionsList>, IMultiInlet, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on


 [Serializable]
 public struct Layer
 {
  public Inlet<TransitionsList> inlet;
  public ulong id;
  [Val("Dist")] public float distance;
  [Val("SF")] public float sizeFactor;

  public Layer (Generator gen)
  {
   id = Den.Tools.Id.Generate();
   inlet = new Inlet<TransitionsList>();
   distance = 1;
   sizeFactor = 0;
  }
 }

 public Layer[] layers = new Layer[0];
 public IEnumerable<IInlet<object>> Inlets ()
 {
  for (int i=0; i<layers.Length; i++)
```

```
  yield return layers[i].inlet;
}

public bool self = true;
public float distance = 1;
public float sizeFactor = 0;



public override void Generate (TileData data, StopToken stop)
{
 TransitionsList src = data.ReadInletProduct(this);
 if (src == null) return;
 if (!enabled) { data.StoreProduct(this, src); return; }

 PosTab srcTab = new PosTab((Vector3)data.area.full.worldPos, (Vector3)data.area.full.worldSize, 16);
 srcTab.Add(src);

 (PosTab posTab, float distance, float sizeFactor)[] subtrahends = new (PosTab,float,float)[layers.Length]
 for (int i=0; i<subtrahends.Length; i++)
 {
  TransitionsList trns = data.ReadInletProduct(layers[i].inlet);
  if (trns == null) continue;

  subtrahends[i].posTab = new PosTab((Vector3)data.area.full.worldPos, (Vector3)data.area.full.worldSize
  subtrahends[i].posTab.Add(trns);
```

```
  subtrahends[i].distance = layers[i].distance;
  subtrahends[i].sizeFactor = layers[i].sizeFactor;
 }

 if (stop!=null && stop.stop) return;
 PosTab dst = Rarefied(srcTab, subtrahends, stop);
 data.StoreProduct(this, dst.ToTransitionsList());
}



private PosTab Rarefied (PosTab src, (PosTab posTab, float distance, float sizeFactor)[] subtrahends, St
{
 PosTab dst = new PosTab(src.pos, src.size, src.resolution);

 foreach (Transition obj in src.All())
 {
  if (stop!=null && stop.stop) return dst;

  float thisRange =  distance*(1-sizeFactor) + distance*obj.scale.x*sizeFactor; //thisDistance*(1 - (obj.scal

  if (self && dst.IsAnyObjInRange(obj.pos.x, obj.pos.z, thisRange+thisRange)) continue;

  bool remove = false;
  for (int s=0; s<subtrahends.Length; s++)
   if (subtrahends[s].posTab!=null && subtrahends[s].posTab.IsAnyObjInRange(obj.pos.x, obj.pos.z, thisR
    { remove = true; break; }
```

```
   if (remove) continue;


    dst.Add(obj);
   }
   dst.Flush();


   return dst;
  }
 }


[System.Serializable]
[GeneratorMenu (menu="Objects/Modifiers", name ="Combine", iconName="GeneratorIcons/Combine", d
public class Combine200 : Generator, IMultiInlet, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on


 public Inlet<TransitionsList>[] inlets = new Inlet<TransitionsList>[2] { new Inlet<TransitionsList>(), new Inl


 public IEnumerable<IInlet<object>> Inlets ()
  { for (int i=0; i<inlets.Length; i++) yield return inlets[i]; }


 public override void Generate (TileData data, StopToken stop)
 {
  if (!enabled) return;
```

```
  TransitionsList dst = new TransitionsList();
  bool defined = false;

  for (int i=0; i<inlets.Length; i++)
  {
   TransitionsList src = data.ReadInletProduct(inlets[i]);
   if (src != null)
   {
    dst.Add(src);
    defined = true;
   }
  }

  if (!defined) data.StoreProduct(this, null);
  else data.StoreProduct(this, dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Objects/Modifiers", name ="Spread", iconName="GeneratorIcons/Spread", diser
public class Spread200 : Generator, IInlet<TransitionsList>, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 public bool retainOriginals = true;
```

```
public int seed = 12345;
public UnityEngine.Vector2 growth = new UnityEngine.Vector2(2,3);
public UnityEngine.Vector2 distance = new UnityEngine.Vector2(1,2);
public float sizeFactor = 0;



public override void Generate (TileData data, StopToken stop)
{
 TransitionsList src = data.ReadInletProduct(this);
 if (src == null) return;
 if (!enabled) { data.StoreProduct(this, src); return; }

 TransitionsList dst = new TransitionsList(); //capacity src.count
 Noise random = new Noise(data.random, seed);

 for (int t=0; t<src.count; t++)
  Spread(src.arr[t], dst, random);

 if (retainOriginals) dst.Add(src);

 data.StoreProduct(this, dst);
}



private void Spread (Transition trn, TransitionsList spreadedList, Noise random)
{
```

```
//calculating number of propagate objects
float rnd = random.Random(trn.hash);
float num = growth.x + rnd*(growth.y-growth.x);
num = num*(1-sizeFactor) + num*trn.scale.x*sizeFactor;
num = Mathf.Round(num);

//creating objs
for (int n=0; n<num; n++)
{
 float angRnd = random.Random(trn.hash, n*2);
 float distRnd = rnd = random.Random(trn.hash, n*2+1);
 float angle = angRnd * Mathf.PI*2; //in radians
 UnityEngine.Vector2 direction = new UnityEngine.Vector2(Mathf.Sin(angle), Mathf.Cos(angle) );
 float dist = distance.x + distRnd*(distance.y-distance.x);
 dist = dist*(1-sizeFactor) + dist*trn.scale.x*sizeFactor;

 float posX = trn.pos.x + direction.x*dist;
 //if (posX <= dst.rect.offset.x+1.01f) posX = dst.rect.offset.x+1.01f;
 //if (posX >= dst.rect.offset.x+dst.rect.size.x-1.01f) posX = dst.rect.offset.x+dst.rect.size.x-1.01f;

 float posZ = trn.pos.z + direction.y*dist;
 //if (posZ <= dst.rect.offset.z+1.01f) posZ = dst.rect.offset.z+1.01f;
 //if (posZ >= dst.rect.offset.z+dst.rect.size.z-1.01f) posZ = dst.rect.offset.z+dst.rect.size.z-1.01f;

 Transition newPos = new Transition() {
  pos = new Vector3(posX, trn.pos.y, posZ),
```

```
    rotation = trn.rotation,
    scale = trn.scale,
    hash = (trn.hash<<1) + n};
   spreadedList.Add(newPos); //with auto id
  }
 }
}
```

```
//TODO: could be unified with stamp
/*[System.Serializable]
[GeneratorMenu (menu="Objects", name ="Blob", disengageable = true, helpLink = "https://gitlab.com/der
public class BlobGenerator : Generator, IOutlet<MatrixWorld>
{
 [Val(name="Objects", priority=2)] public readonly Inlet<PosTab> objectsIn = new Inlet<PosTab>();
 [Val(name="Canvas", priority=2)] public readonly Inlet<MatrixWorld> canvasIn = new Inlet<MatrixWorld>(
 [Val(name="Mask", priority=2)]  public readonly Inlet<MatrixWorld> maskIn = new Inlet<MatrixWorld>();

 [Val(name="Intensity")]  public float intensity = 1f;
 [Val(name="Radius")]  public float radius = 10;
 [Val(name="Size Factor")] public float sizeFactor = 0;
 [Val(name="Fallof")]  public AnimationCurve fallof = new AnimationCurve( new Keyframe[] { new Keyfram
 [Val(name="Noise Amount")] public float noiseAmount = 0.1f;
```

```
[Val(name="Noise Size")] public float noiseSize = 100;
[Val(name="Safe Borders")] public int safeBorders = 0;

public override void Generate (TileData data, StopToken stop)
{
 //getting inputs
 PosTab objects = results.GetProduct<PosTab>(objectsIn);
 MatrixWorld src = results.GetProduct<MatrixWorld>(canvasIn);

 if (objects==null) { results.SetProduct(this, null); return; }  //should set anything to mark as generated

 //preparing output
 MatrixWorld dst;
 if (src != null) dst = (MatrixWorld)src.Clone();
 else dst = new MatrixWorld(area.full.resolution, area.full.position, area.full.size);

 foreach (Transition obj in objects.AllObjs())
  DrawBlob(dst, obj.pos.x, obj.pos.z, intensity, radius, fallof, noiseAmount, noiseSize);

 MatrixWorld mask = results.GetProduct<MatrixWorld>(maskIn);
 if (mask != null) MatrixWorld.Mask(src, dst, mask);
 if (safeBorders != 0) MatrixWorld.SafeBorders(src, dst, safeBorders);

 //setting output
 if (stop!=null && stop(0)) return;
 results.SetProduct(this, dst);
```

```
}

public override void Clear (TileData data, StopToken stop)
{
 data.products.Remove(this);
 data.ready.CheckRemove(this);
}

public override bool IsReady (TileData data, StopToken stop)
{
 return data.products.Exists(this);
}

public static void DrawBlob (MatrixWorld canvas, float posX, float posZ, float val, float radius, AnimationC
{
 Coord mapCoord = new Coord(canvas.WorldToMap(posX), canvas.WorldToMap(posZ));
 int mapRadius = canvas.WorldToMap(radius);
 CoordRect blobRect = new CoordRect(mapCoord-mapRadius, new Coord(mapRadius*2 + 1, mapRadius

 Curve curve = new Curve(fallof);
 InstanceRandom noise = new InstanceRandom(noiseSize, 512, 12345, 123); //TODO: use normal noise

 CoordRect intersection = CoordRect.Intersected(canvas.rect, blobRect);
 Coord center = blobRect.Center;
 Coord min = intersection.Min; Coord max = intersection.Max;
 for (int x=min.x; x<max.x; x++)
```

```
   for (int z=min.z; z<max.z; z++)
  {
   //float dist = Coord.Distance(center, new Coord(x,z));
   float distX = canvas.MapToWorld(x) - posX;
   float distZ = canvas.MapToWorld(z) - posZ;
   float dist = Mathf.Sqrt(distX*distX + distZ*distZ);

   float percent = curve.Evaluate(1f - dist/radius);
   float result = percent;

   if (noiseAmount > 0.001f)
   {
    float maxNoise = percent; if (percent > 0.5f) maxNoise = 1-percent;
    result += (noise.Fractal(x,z)*2 - 1) * maxNoise * noiseAmount;
   }

   //canvas[x,z] = Mathf.Max(result*val, canvas[x,z]);
   canvas[x,z] = val*result + canvas[x,z]*(1-result);
  }
 }
}*/


[System.Serializable]
[GeneratorMenu (
 menu="Objects/Modifiers",
```

```
 name ="Flatten",
 iconName="GeneratorIcons/Flatten",
 disengageable = true,
 colorType = typeof(TransitionsList),
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/ObjectsGenerators/Flatten")]
public class Flatten200 : Generator, IMultiInlet, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Positions", "Inlet")] public readonly Inlet<TransitionsList> positionsIn = new Inlet<TransitionsList>();
 [Val("Heights", "Inlet")] public readonly Inlet<MatrixWorld> heightsIn = new Inlet<MatrixWorld>();

 [Val("Radius")]   public float radius = 1;
 [Val("Hardness")]  public float hardness = 0.5f;
 [Val("Size Factor")] public float sizeFactor = 0;

 public bool noiseFallof = false;
 public float noiseAmount = 1f;
 public float noiseSize = 10;


 public IEnumerable<IInlet<object>> Inlets () { yield return positionsIn; yield return heightsIn; }

 public override void Generate (TileData data, StopToken stop)
 {
  TransitionsList positions = data.ReadInletProduct(positionsIn);
```

```
MatrixWorld heights = data.ReadInletProduct(heightsIn);
if (heights == null) return;
if (positions == null) { data.StoreProduct(this, heights); return; }
if (!enabled) return;

heights = new MatrixWorld(heights);

Noise random = null;
if (noiseFallof) random = new Noise(data.random, 12345);

for (int t=0; t<positions.count; t++)
{
 if (stop!=null && stop.stop) return;
 Transition obj = positions.arr[t];

 //if (obj.pos.x < heights.worldPos.x || obj.pos.x > heights.worldPos.x+heights.worldSize.x ||
 // obj.pos.z < heights.worldPos.z || obj.pos.z > heights.worldPos.z+heights.worldSize.z)
 //  continue;

 StampLevel(heights,
  level: heights.GetWorldInterpolatedValue(obj.pos.x, obj.pos.z),
  center: (Vector2D)obj.pos,
  radius: radius*(1-sizeFactor) + radius*obj.scale.y*sizeFactor,
  hardness: hardness,
  noise:random, noiseAmount:noiseAmount, noiseSize:noiseSize);
}
```

```
 data.StoreProduct(this, heights);
}


public static void StampLevel (MatrixWorld matrix, float level, Vector2D center, float radius, float hardness
 Noise noise=null, float noiseAmount=0, float noiseSize=20)
{
 Vector2D mapCenter = (Vector2D)matrix.WorldToPixelInterpolated(center.x, center.z);
 float mapRadius = matrix.WorldDistToPixelInterpolated(radius);
 CoordRect stampRect = new CoordRect(mapCenter, mapRadius);

 CoordRect intersection = CoordRect.Intersected(matrix.rect, stampRect);
 Coord min = intersection.Min; Coord max = intersection.Max;

 Coord coord = new Coord(); //temporary coord to call GetFallof

for (int x=min.x; x<max.x; x++)
 for (int z=min.z; z<max.z; z++)
{
 coord.x = x;
 coord.z = z;

 float falloff = coord.GetInterpolatedFalloff(mapCenter, mapRadius, hardness, smooth:2);
 if (falloff < 0.00001f) continue;
```

```
   if (noise != null)
   {
    float maxNoise = falloff; if (falloff > 0.5f) maxNoise = 1-falloff;
    falloff += (noise.Fractal(x,z,noiseSize)*2 - 1) * maxNoise * noiseAmount;
   }

   int pos = (z-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;  //coord.GetPos
   matrix.arr[pos] = level*falloff + matrix.arr[pos]*(1-falloff);
  }
 }
}


[System.Serializable]
[GeneratorMenu (
 menu="Objects/Modifiers",
 name ="Stroke",
 iconName="GeneratorIcons/Flatten",
 disengageable = true,
 colorType = typeof(TransitionsList),
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/ObjectsGenerators/Stroke")]
public class Stroke200 : Generator, IInlet<TransitionsList>, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Radius")]   public float radius = 1;
```

```
[Val("Hardness")]  public float hardness = 0.5f;
[Val("Size Factor")] public float sizeFactor = 0;

public bool noiseFallof = false;
public float noiseAmount = 10f;
public float noiseSize = 5f;



public override void Generate (TileData data, StopToken stop)
{
 TransitionsList positions = data.ReadInletProduct(this);
 if (positions == null  ||  !enabled) return;

 MatrixWorld matrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSiz

 Noise random = null;
 if (noiseFallof) random = new Noise(data.random, 12345);

 for (int t=0; t<positions.count; t++)
 {
  if (stop!=null && stop.stop) return;
  Transition obj = positions.arr[t];

  StampMax(matrix,
   center: (Vector2D)obj.pos,
   radius: radius*(1-sizeFactor) + radius*obj.scale.y*sizeFactor,
```

```
    hardness: hardness,
    noise:random, noiseAmount:noiseAmount, noiseSize:noiseSize);
  }

  matrix.Clamp01(); //for test purpose on issue https://forum.unity.com/threads/released-mapmagic-2-infinit
  data.StoreProduct(this, matrix);
 }



 private static void StampMax (MatrixWorld matrix, Vector2D center, float radius, float hardness,
  Noise noise=null, float noiseAmount=0, float noiseSize=20)
  /// Nearly copy of StampLevel with the other apply algorithm
 {
  Vector2D mapCenter = (Vector2D)matrix.WorldToPixelInterpolated(center.x, center.z);
  float mapRadius = matrix.WorldDistToPixelInterpolated(radius);
  CoordRect stampRect = new CoordRect(mapCenter, mapRadius);

  CoordRect intersection = CoordRect.Intersected(matrix.rect, stampRect);
  Coord min = intersection.Min; Coord max = intersection.Max;

  Coord coord = new Coord(); //temporary coord to call GetFallof

  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
  {
   coord.x = x;
```

```
    coord.z = z;

    float falloff = coord.GetInterpolatedFalloff(mapCenter, mapRadius, hardness, smooth:1);
    if (falloff < 0.00001f) continue;

    if (noise != null)
    {
     float maxNoise = falloff; if (falloff > 0.5f) maxNoise = 1-falloff;
     falloff += (noise.Fractal(x,z,noiseSize)*2 - 1) * maxNoise * noiseAmount;
    }

    int pos = (z-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;  //coord.GetPos
    if (falloff>matrix.arr[pos]) matrix.arr[pos] = falloff;
   }
  }
 }


[System.Serializable]
[GeneratorMenu (
 menu="Objects/Modifiers",
 name ="Stamp",
 iconName="GeneratorIcons/Stamp",
 disengageable = true,
 colorType = typeof(TransitionsList),
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/ObjectsGenerators/Stamp")]
```

```
public class Stamp200 : Generator, IMultiInlet, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Positions", "Inlet")]  public readonly Inlet<TransitionsList> positionsIn = new Inlet<TransitionsList>();
 [Val("Stamp", "Inlet")]   public readonly Inlet<MatrixWorld> stampIn = new Inlet<MatrixWorld>();

 [Val("Size", "Custom")] public float size = 1;
 [Val("Intensity", "Custom")] public float intensity = 1;
 [Val("Hardness", "UseFallof")] public float hardness = 0.8f;
 [Val("Size Factor", "Custom")] public float sizeFactor = 1;
 [Val("Intensity Factor", "Custom")] public float intensityFactor = 1;

 public enum BlendType { Max, Add }
 public BlendType blendType = BlendType.Max;

 [Val("Use Rotation", "Custom", isLeft = true)] public bool useRotation = true;
 [Val("Use Falloff", "Custom", isLeft = true)] public bool useFalloff = false;


 public IEnumerable<IInlet<object>> Inlets () { yield return positionsIn; yield return stampIn; }

 public override void Generate (TileData data, StopToken stop)
 {
  TransitionsList positions = data.ReadInletProduct(positionsIn);
  MatrixWorld stamp = data.ReadInletProduct(stampIn);
```

```
 if (positions == null || stamp == null) return;
 if (!enabled) return;

 MatrixWorld dst = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldSize, d

 for (int t=0; t<positions.count; t++)
 {
  if (stop!=null && stop.stop) return;

  float radius = size/2;
  float currRadius = radius*(1-sizeFactor) + radius*positions.arr[t].scale.y*sizeFactor;
  if (useRotation) currRadius *= 1.414213562373095f; //it will be then reduced in stamp

  StampMatrix(dst, stamp,
   stampRect: data.area.active.rect,
   center:positions.arr[t].pos,
   rotation: positions.arr[t].rotation,
   radius: currRadius,
   hardness: hardness,
   intensity: intensity*(1-intensityFactor) + intensity*positions.arr[t].scale.y*intensityFactor,
   blendAdditive: blendType==BlendType.Add,
   useRotation:useRotation, useFalloff:useFalloff);
 }

 data.StoreProduct(this, dst);
}
```

```
private static void StampMatrix (MatrixWorld matrix, MatrixWorld stamp, CoordRect stampRect,
 Vector3 center, Quaternion rotation, float radius, float hardness, float intensity, bool blendAdditive,
 bool useRotation, bool useFalloff)
{
 Vector2D mapCenter = (Vector2D)matrix.WorldToPixelInterpolated(center.x, center.z);
 float mapRadius = matrix.WorldDistToPixelInterpolated(radius);
 CoordRect strokeRect = new CoordRect(mapCenter, mapRadius);

 CoordRect intersection = CoordRect.Intersected(matrix.rect, strokeRect);
 Coord min = intersection.Min; Coord max = intersection.Max;

 Coord coord = new Coord(); //temporary coord to call GetFallof

 Vector2D rotDirection = new Vector2D(0,1);
 if (useRotation)
  rotDirection = (Vector2D)(rotation * new Vector3(1,0,0));

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
 {
  //stamp relative coordinates
  float stampPercentX = 1f * (x - strokeRect.offset.x) / (strokeRect.size.x-1); //(size-1)
  float stampPercentZ = 1f * (z - strokeRect.offset.z) / (strokeRect.size.z-1);
```

```
if (stampPercentX > 0.999f) stampPercentX = 0.999f; //to avoid reading last pixel which could be Simple
if (stampPercentZ > 0.999f) stampPercentZ = 0.999f;

//falloff
float falloff = 1;
if (useFalloff)
{
 coord.x = x; coord.z = z;

 falloff = coord.GetInterpolatedFalloff(mapCenter, mapRadius / (useRotation ? 1.414213562373095f : 1)
 if (falloff < 0.00001f) continue;
}

//value
float value = 0;
if (useRotation)
{
 stampPercentX = (stampPercentX-0.5f)*1.414213562373095f + 0.5f;  //reducing stamp square to make
 stampPercentZ = (stampPercentZ-0.5f)*1.414213562373095f + 0.5f;

 (stampPercentX, stampPercentZ) = GetRelativeRotatedCoords(stampPercentX, stampPercentZ, rotDire
}

float fx = stampPercentX*stampRect.size.x + stampRect.offset.x;
float fz = stampPercentZ*stampRect.size.z + stampRect.offset.z;
value = stamp.GetFloored(fx,fz);
```

```
  //applying
  value *= intensity;
  int pos = (z-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;  //coord.GetPos
  if (blendAdditive)
   matrix.arr[pos] += value*falloff;
  else //max
   if (value*falloff > matrix.arr[pos])
    matrix.arr[pos] = value*falloff;
 }
}

private static (float,float) GetRelativeRotatedCoords (float sx, float sz, Vector2D rotationDirection, Vector2
/// Rotating relative (0-1) coordinates across pivot (0-1 too)
{
 float cx = sx - rotationPivot.x;  float cz = sz - rotationPivot.z;

 Vector2D vx = rotationDirection * cx;
 Vector2D vz = new Vector2D(rotationDirection.z, -rotationDirection.x) * cz;
 Vector2D v = vx+vz;

 v.x += rotationPivot.x;  v.z += rotationPivot.z;

 if (v.x < 0) v.x = 0; if (v.x > 0.999f) v.x = 0.999f; //to avoid reading last pixel which could be Simple Form's
 if (v.z < 0) v.z = 0; if (v.z > 0.999f) v.z = 0.999f;
```

```
  return (v.x, v.z);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Objects/Modifiers", name ="Forest", iconName="GeneratorIcons/Forest", diseng
public class Forest200 : Generator, IMultiInlet, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Seedlings", "Inlet")]  public readonly Inlet<TransitionsList> seedlingsIn = new Inlet<TransitionsList>(
 [Val("Other Trees", "Inlet")] public readonly Inlet<TransitionsList> otherTreesIn = new Inlet<TransitionsLis
 [Val("Soil", "Inlet")]   public readonly Inlet<MatrixWorld> soilIn = new Inlet<MatrixWorld>();
 public IEnumerable<IInlet<object>> Inlets () { yield return seedlingsIn; yield return otherTreesIn; yield retu

 [Val("Years")]   public int years = 100;
 [Val("Density")]  public float density = 10000; //max trees per 1*1km
 [Val("Fecundity")]  public float fecundity = 0.5f;
 [Val("Seed Dist")]  public float seedDist = 15;
 [Val("Reproductive Age")]public float reproductiveAge = 10;
 [Val("Survival Rate")] public float survivalRate = 0.95f;
 [Val("Life Age")]  public float lifeAge = 100;
 [Val("Size Is Age")] public bool sizeIsLife = true;
 [Val("Seed")]   public int seed = 12345;
```

```
public override void Generate (TileData data, StopToken stop)
{
 TransitionsList seedlings = data.ReadInletProduct(seedlingsIn);
 if (seedlings == null) return;
 if (!enabled) { data.StoreProduct(this, seedlings); return; }

 TransitionsList otherTrees = data.ReadInletProduct(otherTreesIn);
 MatrixWorld soil = data.ReadInletProduct(soilIn);

 Noise random = new Noise(data.random, seed);

 TransitionsList forest = Forest(seedlings, otherTrees, soil, (Vector3)data.area.full.worldPos, (Vector3)dat

 data.StoreProduct(this, forest);
}

public TransitionsList Forest (TransitionsList seedlings, TransitionsList otherTrees,
 MatrixWorld soil, Vector3 worldPos, Vector3 worldSize, Noise random, StopToken stop=null)
{
 float cellSize = 1000 / Mathf.Sqrt(density);

 CoordRect rect = CoordRect.WorldToPixel((Vector2D)worldPos, (Vector2D)worldSize, (Vector2D)cellSiz
 worldPos = new Vector3(rect.offset.x*cellSize, worldPos.y, rect.offset.z*cellSize);  //modifying worldPos/s
 worldSize = new Vector3(rect.size.x*cellSize, worldSize.y, rect.size.z*cellSize);
```

```
PositionMatrix forest = new PositionMatrix(rect, worldPos, worldSize);
forest.Scatter(0, random, maxHeight:0);
forest = forest.Relaxed();

PositionMatrix otherForest = new PositionMatrix(rect, worldPos, worldSize);
if (otherTrees != null)
 otherForest.AddTransitionsList(otherTrees, 1); //using custom height as tree age

//growing
for (int y=0; y<years; y++)
{
 //filling seedlings - each iteration (except the last one) to make them persistent
// forest.AddTransitionsList(seedlings, reproductiveAge+1); //with custom height
 if (y==0)
  TransitionsToMatrix(forest, seedlings, sizeIsLife);

 //generating
 Coord min = forest.rect.Min; Coord max = forest.rect.Max;
 for (int x=min.x; x<max.x; x++)
 {
  if (stop!=null && stop.stop) return null; //checking stop every x line

  for (int z=min.z; z<max.z; z++)
  {
   float age = forest.GetHeight(x,z);
```

```
if (age < 0.5f) continue;  //no tree in this cell
int iAge = (int)age - 1; //to be used in random instead of year (for brush compatibility)

//growing tree
forest.SetHeight(x,z, ++age);

//killing the tree
float curSurvivalRate = survivalRate;
if (soil != null)
{
 Vector3 wpos = forest[x,z];
 if (!soil.ContainsWorldValue(wpos.x, wpos.z)) curSurvivalRate = 0;
 else curSurvivalRate *= soil.GetWorldValue(wpos.x, wpos.z);
}
if (age > lifeAge || random.Random(x,z,iAge,0) > curSurvivalRate)
 forest.SetHeight(x,z, 0);

//breeding the tree
//TODO: use id random
if (age > reproductiveAge && random.Random(x,z,iAge,1) < fecundity)
{
 float angleRad = random.Random(x,z,iAge,2) * 6.283f;
 float dist = random.Random(x,z,iAge,3) * seedDist/forest.cellSize + 1;

 int nx = (int)(x + Mathf.Sin(angleRad)*dist);
 int nz = (int)(z + Mathf.Cos(angleRad)*dist);
```

```
     if (forest.rect.Contains(nx, nz) && forest.GetHeight(nx,nz)<0.5f && otherForest.GetHeight(nx,nz)<0.01f
      forest.SetHeight(nx,nz, 1);
    }
   }
  }
 }

 //return forest.ToTransitionsList(minHeight:0.5f);
 //using method copy with some custom changes

 TransitionsList trsList = new TransitionsList(); //capacity rect.size.x * rect.size.z
 MatrixToTransitions(forest, trsList, sizeIsLife);

 return trsList;
}



private void TransitionsToMatrix (PositionMatrix matrix, TransitionsList trns, bool scaleIsAge)
{
 for (int t=0; t<trns.count; t++)
 {
  if (trns.arr[t].pos.x < matrix.worldPos.x || trns.arr[t].pos.x > matrix.worldPos.x+matrix.worldSize.x ||
   trns.arr[t].pos.z < matrix.worldPos.z || trns.arr[t].pos.z > matrix.worldPos.z+matrix.worldSize.z)
    continue;
```

```
  float currAge = scaleIsAge ?
   trns.arr[t].scale.y :
   reproductiveAge+1;

  float prevAge; //wring the oldest tree
  Coord coord = matrix.GetCoord(trns.arr[t].pos);
  prevAge = matrix.GetHeight(coord.x, coord.z);

  if (currAge > prevAge)
   matrix.SetPosition( new Vector3(trns.arr[t].pos.x, currAge, trns.arr[t].pos.z) );
 }
}

private static void MatrixToTransitions (PositionMatrix matrix, TransitionsList trns, bool scaleIsAge)
{
 Coord rmin = matrix.rect.Min; Coord rmax = matrix.rect.Max;
 for (int x=rmin.x; x<rmax.x; x++)
  for (int z=rmin.z; z<rmax.z; z++)
  {
   Vector3 pos = matrix[x,z];
   if (pos.y < 1.5f) continue; //removing trees on improper soil that were appeared for 1 year

   Transition trs = new Transition(pos.x, pos.z);
   if (scaleIsAge) { trs.scale.x=pos.y; trs.scale.y=pos.y; trs.scale.z=pos.y; }
   trs.hash = x*2000 + z; //to make hash independent from grid size
```

```
    trns.Add(trs);
   }
 }

}

[System.Serializable]
[GeneratorMenu (menu="Objects/Modifiers", name ="Slide", iconName="GeneratorIcons/Slide", disengag
public class SlideGenerator200 : Generator, IMultiInlet, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Input", "Inlet")]  public readonly Inlet<TransitionsList> srcIn = new Inlet<TransitionsList>();
 [Val("Stratum", "Inlet")]  public readonly Inlet<MatrixWorld> stratumIn = new Inlet<MatrixWorld>();

 [Val("Blur")]  public int smooth = 2;
 [Val("Iterations")] public int iterations = 100;
 [Val("Move Factor")] public float moveFactor = 0.2f;
 [Val("Stop Slope")] public float stopSlope = 10;


 public IEnumerable<IInlet<object>> Inlets () { yield return srcIn; yield return stratumIn; }


 public override void Generate (TileData data, StopToken stop)
```

```
{
 TransitionsList src = data.ReadInletProduct(srcIn);
 MatrixWorld stratum = data.ReadInletProduct(stratumIn);
 if (src == null) return;
 if (!enabled || stratum == null) { data.StoreProduct(this, src); return; }

 TransitionsList dst = new TransitionsList(src);

 //preparing matrix mip for smoothing
 if (smooth != 0)
 {
  Matrix[] mips = MatrixOps.GenerateMips(stratum, smooth);
  Matrix mip = mips[mips.Length-1];

  stratum = new MatrixWorld(mip, stratum.worldPos, stratum.worldSize);
 }

 //finding stop slope (in 0-1 height difference, same as slope gen)
 float stopDelta = Mathf.Tan(stopSlope*Mathf.Deg2Rad) * data.area.PixelSize.x / data.globals.height;

 int resIndepIterations = (int)(iterations/512f * data.area.active.rect.size.x);

 for (int t=0; t<dst.count; t++)
  Slide(ref dst.arr[t], stratum, resIndepIterations, moveFactor, stopDelta, data.globals.height);
 //TODO: test resolution independance
```

```
 data.StoreProduct(this, dst);
}



public static void Slide (ref Transition trn, MatrixWorld stratum, int iterations, float moveFactor, float stopD
{
 for (int i=0; i<iterations; i++)
 {
  //flooring coordiantes
  //TODO: use built-in matrix operations
  Coord pos = stratum.WorldToPixel(trn.pos.x, trn.pos.z);

  if (!stratum.rect.Contains(pos.x, pos.z, 1.0001f)) break;

  float heightMXMZ = stratum[pos.x, pos.z];
  float heightPXMZ = stratum[pos.x+1, pos.z];
  float heightMXPZ = stratum[pos.x, pos.z+1];
  float heightPXPZ = stratum[pos.x+1, pos.z+1];

  float xNormal1 = heightMXPZ-heightPXPZ; //Mathf.Atan(heightPXPZ-heightMXPZ) / halfPi;
  float xNormal2 = heightMXMZ-heightPXMZ; //Mathf.Atan(heightPXMZ-heightMXMZ) / halfPi;
  float zNormal1 = heightPXMZ-heightPXPZ; //Mathf.Atan(heightPXPZ-heightPXMZ) / halfPi;
  float zNormal2 = heightMXMZ-heightMXPZ; //Mathf.Atan(heightMXPZ-heightMXMZ) / halfPi;

  //finding incline tha same way as the slope generator
  float xDelta1 = xNormal1>0? xNormal1 : -xNormal1; float xDelta2 = xNormal2>0? xNormal2 : -xNormal2
```

```
    float zDelta1 = zNormal1>0? zNormal1 : -zNormal1; float zDelta2 = zNormal2>0? zNormal2 : -zNormal2
    float delta = xDelta>zDelta? xDelta : zDelta; //because slope generator uses additive blend

    if (delta < stopDelta) continue;

    float xNormal = (xNormal1+xNormal2)/2f;
    float zNormal = (zNormal1+zNormal2)/2f; //TODO: use smooth interpolation

    trn.pos.x += xNormal*(terrainHeight*moveFactor);
    trn.pos.z += zNormal*(terrainHeight*moveFactor);
   }
  }
 }

[Serializable]
[GeneratorMenu(
 menu = "Objects/Modifiers",
 name = "Move",
 section=2,
 colorType = typeof(TransitionsList),
 iconName="GeneratorIcons/Move",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Move")]
public class Move210 : Generator, IInlet<TransitionsList>, IMultiInlet, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
```

```
[Val("Intensity", "Inlet")]  public readonly Inlet<MatrixWorld> intensityIn = new Inlet<MatrixWorld>();
public IEnumerable<IInlet<object>> Inlets () { yield return intensityIn; }

[Val("Direction")] public Vector2D direction;

public override void Generate (TileData data, StopToken stop)
{
 if (stop!=null && stop.stop) return;
 TransitionsList src = data.ReadInletProduct(this);
 if (src == null) return;

 MatrixWorld intensity = data.ReadInletProduct(intensityIn);
 if (!enabled || intensity == null) { data.StoreProduct(this, src); return; }

 if (stop!=null && stop.stop) return;
 TransitionsList dst = new TransitionsList(src);
 for (int t=0; t<dst.count; t++)
 {
  Transition trs = src.arr[t];

  if (trs.pos.x <= intensity.worldPos.x  ||  trs.pos.x >= intensity.worldPos.x+intensity.worldSize.x  ||
   trs.pos.z <= intensity.worldPos.z  ||  trs.pos.z >= intensity.worldPos.z+intensity.worldSize.z)
    continue;

  float percent = intensity.GetWorldInterpolatedValue(trs.pos.x, trs.pos.z);
```

```
    dst.arr[t].pos += direction*percent;
   }

   if (stop!=null && stop.stop) return;
   data.StoreProduct(this, dst);
  }
 }


[Serializable]
[GeneratorMenu(
 menu = "Objects/Modifiers",
 name = "Shrink Scale",
 section=2,
 colorType = typeof(TransitionsList),
 iconName="GeneratorIcons/ShrinkScale",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/ShrinkScale")]
public class ShrinkScale210 : Generator, IInlet<TransitionsList>, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Multiply")] public float multiply = 0.001f;
 [Val("Invert")] public bool invert;

 public override void Generate (TileData data, StopToken stop)
```

```
 {
  if (stop!=null && stop.stop) return;
  TransitionsList src = data.ReadInletProduct(this);
  if (src == null) return;

  if (stop!=null && stop.stop) return;
  TransitionsList dst = new TransitionsList(src);

  if (stop!=null && stop.stop) return;
  if (!invert)
   for (int t=0; t<dst.count; t++)
    dst.arr[t].scale = Vector3.one + dst.arr[t].scale*multiply;
  else
   for (int t=0; t<dst.count; t++)
    dst.arr[t].scale = (dst.arr[t].scale - Vector3.one)/multiply;


  if (stop!=null && stop.stop) return;
  data.StoreProduct(this, dst);
 }
}


[Serializable]
[GeneratorMenu(
 menu = "Objects/Modifiers",
```

```
 name = "Safe Borders",
 section=2,
 colorType = typeof(TransitionsList),
 iconName="GeneratorIcons/SafeBorders",
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/ShrinkScale")]
public class SafeBorders218 : Generator, IInlet<TransitionsList>, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Edge Dist")] public float edgeDist = 10f;

 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
  TransitionsList src = data.ReadInletProduct(this);
  if (src == null) return;

  Vector2D safeMin = data.area.active.worldPos + edgeDist;
  Vector2D safeMax = data.area.active.worldPos + data.area.active.worldSize - edgeDist;

  TransitionsList dst = new TransitionsList();

  for (int t=0; t<src.count; t++)
  {
   if (stop!=null && stop.stop) return;
```

```
    Vector3 pos = src.arr[t].pos;


    if (pos.x <= safeMin.x  ||   pos.x >= safeMax.x  ||
     pos.z <= safeMin.z  ||   pos.z >= safeMax.z)
      continue;


    dst.Add(src.arr[t]);
   }


   if (stop!=null && stop.stop) return;
   data.StoreProduct(this, dst);
  }
 }

}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Products;
using MapMagic.Terrains;

namespace MapMagic.Nodes.ObjectsGenerators
{

 [System.Serializable]
 public abstract class BaseObjectsOutput : OutputGenerator
 // doesn't have to be Generator, but we can't inherit from both BaseObjectsOutput and Generator
 // should be mostly replaced with PositioningSettings
 {
  public string name = "(Empty)";
  public GameObject[] prefabs = new GameObject[1];

  public bool guiMultiprefab;
  public bool guiProperties;
```

```
 public BiomeBlend biomeBlend = BiomeBlend.Random;

 //outdated:
 public bool objHeight = true;
 public bool relativeHeight = true;

 public bool useRotation = true; //in base since tree could also be rotated. Not the imposter ones, but anyw
 public bool takeTerrainNormal = false;
 public bool rotateYonly = false;
 public bool regardPrefabRotation = false;

 public bool useScale = true;
 public bool scaleYonly = false;
 public bool regardPrefabScale = false;

 public enum BiomeBlend { Sharp, Random, Scale, Pure }
}


[System.Serializable]
[GeneratorMenu(menu = "Objects/Outputs", name = "Objects", section=2, colorType = typeof(TransitionsL
public class ObjectsOutput : OutputGenerator, IInlet<TransitionsList>, IPrepare
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 //common settings
```

```
public GameObject[] prefabs = new GameObject[1];
public PositioningSettings posSettings = null; // new PositioningSettings(); //to load older output
public BiomeBlend biomeBlend = BiomeBlend.Random;

public OutputLevel outputLevel = OutputLevel.Main;
public override OutputLevel OutputLevel { get{ return outputLevel; } }

public bool guiMultiprefab;
public bool guiProperties;

//specific settings
public bool allowReposition = true;
public bool instantiateClones = false;
public int seed = 12345;

//moved to PositioningSettings, and thus outdated:
public bool objHeight = true;
public bool relativeHeight = true;
public bool guiHeight;
public bool useRotation = true;
public bool takeTerrainNormal = false;
public bool rotateYonly = false;
public bool regardPrefabRotation = false;
public bool guiRotation;
public bool useScale = true;
public bool scaleYonly = false;
```

```
public bool regardPrefabScale = false;
public bool guiScale;

public static PositioningSettings CreatePosSettings (ObjectsOutput output)
{
 PositioningSettings ps = new PositioningSettings();
 ps.objHeight=output.objHeight; ps.relativeHeight=output.relativeHeight; ps.guiHeight=output.guiHeight;
 ps.useRotation=output.useRotation; ps.takeTerrainNormal=output.takeTerrainNormal; ps.rotateYonly=ou
 ps.useScale=output.useScale; ps.scaleYonly=output.scaleYonly; ps.regardPrefabScale=output.regardPr
 return ps;
}


public void Prepare (TileData data, Terrain terrain)
{
 //resetting modified objects to real nulls - otherwise they won't appear in thread
 for (int p=0; p<prefabs.Length; p++)
  if ((UnityEngine.Object)prefabs[p] == (UnityEngine.Object)null)  //if (prefabs[p] == null)
   prefabs[p] = null;
}

public List<ObjectsPool.Prototype> GetPrototypes ()
{
 List<ObjectsPool.Prototype> prototypes = new List<ObjectsPool.Prototype>();
 for (int p=0; p<prefabs.Length; p++)
  if (!prefabs[p].IsNull())  //if (prefabs[p] != null)
```

```
      prototypes.Add (new ObjectsPool.Prototype() {
       prefab = prefabs[p],
       allowReposition = allowReposition,
       instantiateClones = instantiateClones,
       regardPrefabRotation = posSettings.regardPrefabRotation,
       regardPrefabScale = posSettings.regardPrefabScale } );
    return prototypes;
   }

   public override void Generate (TileData data, StopToken stop)
   {
    if (stop!=null && stop.stop) return;
//    if (!enabled) { data.finalize.Remove(finalizeAction, this); return; }
    TransitionsList trns = data.ReadInletProduct(this);

    //adding to finalize
    if (enabled)
    {
     data.StoreOutput(this, typeof(ObjectsOutput), this, trns);  //adding src since it's not changing
     data.MarkFinalize(Finalize, stop);
    }
    else
     data.RemoveFinalize(finalizeAction);
   }
```

```
#if UNITY_EDITOR
[UnityEditor.InitializeOnLoadMethod]
#endif
[RuntimeInitializeOnLoadMethod]
static void Subscribe () => Graph.OnOutputFinalized += FinalizeIfHeightFinalized;
static void FinalizeIfHeightFinalized (Type type, TileData tileData, IApplyData applyData, StopToken stop)
{
 if (type == typeof(MatrixGenerators.HeightOutput200))
  tileData.MarkFinalize(finalizeAction, stop);
}

public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 if (stop!=null && stop.stop) return;

 //List<ObjectsPool.Prototype> objPrototypesList = new List<ObjectsPool.Prototype>();
 //List<List<Transition>> objTransitionsList = new List<List<Transition>>();
 //List<(ObjectsPool.Prototype prot, List<Transition> trns)> allObjsList = new List<(ObjectsPool.Prototype,
 Dictionary<ObjectsPool.Prototype, List<Transition>> objs = new Dictionary<ObjectsPool.Prototype, List<

 foreach ((ObjectsOutput output, TransitionsList trns, MatrixWorld biomeMask)
  in data.Outputs<ObjectsOutput,TransitionsList,MatrixWorld>(typeof(ObjectsOutput), inSubs:true))
 {
  if (stop!=null && stop.stop) return;
```

```
if (trns == null) continue;
if (biomeMask!=null  &&  biomeMask.IsEmpty()) continue;

if (output.posSettings == null)
 output.posSettings = CreatePosSettings(output);

List<ObjectsPool.Prototype> prototypes = output.GetPrototypes();
if (prototypes.Count == 0) continue;

foreach (ObjectsPool.Prototype prot in prototypes)
 if (!objs.ContainsKey(prot)) objs.Add(prot, new List<Transition>());

Noise random = new Noise(data.random, output.seed);

//objects
for (int t=0; t<trns.count; t++)
{
 Transition trn = trns.arr[t]; //using copy since it's changing in MoveRotateScale

 if (!data.area.active.Contains(trn.pos)) continue; //skipping out-of-active area
 if (PositioningSettings.SkipOnBiome(ref trn, output.biomeBlend, biomeMask, data.random)) continue; //

 output.posSettings.MoveRotateScale(ref trn, data);

 trn.pos -= (Vector3)data.area.active.worldPos; //objects pool use local positions
```

```
    //float rnd = random.Random(trs.hash);
    //int listNum = transitionsCount + (int)(rnd*output.prefabs.Length);
    //objTransitionsList[listNum].Add(trsCpy);
    //objsList.[listNum].Add(trsCpy);

    float rnd = random.Random(trn.hash);
    ObjectsPool.Prototype prototype = prototypes[ (int)(rnd*prototypes.Count) ];
    objs[prototype].Add(trn);
   }
  }

  //pushing to apply
  if (stop!=null && stop.stop) return;
  ApplyObjectsData applyData = new ApplyObjectsData() {
   prototypes=objs.Keys.ToArray(),
   transitions=objs.Values.ToArray(),
   terrainHeight = data.globals.height,
   objsPerIteration = data.globals.objectsNumPerFrame};
  Graph.OnOutputFinalized?.Invoke(typeof(ObjectsOutput), data, applyData, stop);
  data.MarkApply(applyData);
 }


 public class ApplyObjectsData : IApplyDataRoutine
 {
  public ObjectsPool.Prototype[] prototypes;
```

```
 public List<Transition>[] transitions;
 public float terrainHeight; //to get relative object height (since all of the terrain data is 0-1). //TODO: mayb
 public int objsPerIteration=500;

 public void Apply(Terrain terrain)
 {
  ObjectsPool pool = terrain.transform.parent.GetComponent<TerrainTile>().objectsPool;
  pool.Reposition(prototypes, transitions);
 }

 public IEnumerator ApplyRoutine (Terrain terrain)
 {
  ObjectsPool pool = terrain.transform.parent.GetComponent<TerrainTile>().objectsPool;

  IEnumerator e = pool.RepositionRoutine(prototypes, transitions, objsPerIteration);
  while (e.MoveNext()) { yield return null; }
 }

 public int Resolution {get{ return 0; }}
}


public override void ClearApplied (TileData data, Terrain terrain)
{
 if (posSettings == null)
  posSettings = CreatePosSettings(this);
```

```
  TerrainData terrainData = terrain.terrainData;
  Vector3 terrainSize = terrainData.size;

  ObjectsPool pool = terrain.transform.parent.GetComponent<TerrainTile>().objectsPool;
  List<ObjectsPool.Prototype> prototypes = GetPrototypes();
  pool.ClearPrototypes(prototypes.ToArray());
 }
}


[System.Serializable]
[GeneratorMenu(menu = "Objects/Outputs", name = "Trees", section=2, colorType = typeof(TransitionsLis
public class TreesOutput : OutputGenerator, IInlet<TransitionsList>, IPrepare
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 //common settings
 public GameObject[] prefabs = new GameObject[1];
 public PositioningSettings posSettings = null; // new PositioningSettings(); //to load older output
 public BiomeBlend biomeBlend = BiomeBlend.Random;

 public OutputLevel outputLevel = OutputLevel.Main;
 public override OutputLevel OutputLevel { get{ return outputLevel; } }

 public bool guiMultiprefab;
```

```
public bool guiProperties;

//specific settings
public int seed = 12345;
public Color color = Color.white;
public Color lightmapColor = Color.white;
public float bendFactor;

//moved to PositioningSettings, and thus outdated:
public bool objHeight = true;
public bool relativeHeight = true;
public bool guiHeight;
public bool useRotation = true;
public bool takeTerrainNormal = false;
public bool rotateYonly = false;
public bool regardPrefabRotation = false;
public bool guiRotation;
public bool useScale = true;
public bool scaleYonly = false;
public bool regardPrefabScale = false;
public bool guiScale;

public static PositioningSettings CreatePosSettings (TreesOutput output)
{
 PositioningSettings ps = new PositioningSettings();
 ps.objHeight=output.objHeight; ps.relativeHeight=output.relativeHeight; ps.guiHeight=output.guiHeight;
```

```
 ps.useRotation=output.useRotation; ps.takeTerrainNormal=output.takeTerrainNormal; ps.rotateYonly=ou
 ps.useScale=output.useScale; ps.scaleYonly=output.scaleYonly; ps.regardPrefabScale=output.regardPr
 return ps;
}

public void Prepare (TileData data, Terrain terrain)
{
 //resetting modified objects to real nulls - otherwise they won't appear in thread
 for (int p=0; p<prefabs.Length; p++)
  if ((UnityEngine.Object)prefabs[p] == (UnityEngine.Object)null)  //if (prefabs[p] == null)
   prefabs[p] = null;
}

public List<TreePrototype> GetPrototypes ()
{
 List<TreePrototype> prototypes = new List<TreePrototype>();
 for (int p=0; p<prefabs.Length; p++)
  if (!prefabs[p].IsNull())  //if (prefabs[p] != null)
   prototypes.Add (new TreePrototype() { prefab = prefabs[p], bendFactor = bendFactor } );
 return prototypes;
}


public override void Generate (TileData data, StopToken stop)
{
 if (stop!=null && stop.stop) return;
```

```
//   if (!enabled)
//    { data.finalize.Remove(finalizeAction, this); return; }

   TransitionsList trns = data.ReadInletProduct(this);

   //adding to finalize
   if (stop!=null && stop.stop) return;
   if (enabled)
   {
    data.StoreOutput(this, typeof(TreesOutput), this, trns);  //adding src since it's not changing
    data.MarkFinalize(Finalize, stop);
   }
   else
    data.RemoveFinalize(finalizeAction);
  }


  #if UNITY_EDITOR
  [UnityEditor.InitializeOnLoadMethod]
  #endif
  [RuntimeInitializeOnLoadMethod]
  static void Subscribe () => Graph.OnOutputFinalized += FinalizeIfHeightFinalized;
  static void FinalizeIfHeightFinalized (Type type, TileData tileData, IApplyData applyData, StopToken stop)
  {
   if (type == typeof(MatrixGenerators.HeightOutput200))
    tileData.MarkFinalize(finalizeAction, stop);
```

```
}

public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 if (stop!=null && stop.stop) return;

 List<TreeInstance> instancesList = new List<TreeInstance>();
 List<TreePrototype> prototypesList = new List<TreePrototype>();

 int prototypesCount = 0; //the total number of prototypes added to give unique index for trees

 foreach ((TreesOutput output, TransitionsList trns, MatrixWorld biomeMask)
  in data.Outputs<TreesOutput,TransitionsList,MatrixWorld>(typeof(TreesOutput), inSubs:true))
 {
  if (stop!=null && stop.stop) return;

  if (trns == null) continue;
  if (biomeMask!=null  &&  biomeMask.IsEmpty()) continue;

  if (output.posSettings == null)
   output.posSettings = CreatePosSettings(output);

  Noise random = new Noise(data.random, output.seed);

  //prototypes
```

```
//TODO: use GetPrototypes (and skip RemoveNullPrototypes)
TreePrototype[] prototypesArr = new TreePrototype[output.prefabs.Length];
for (int p=0; p<output.prefabs.Length; p++)
 prototypesArr[p] = new TreePrototype() { prefab =  output.prefabs[p], bendFactor = output.bendFactor };
prototypesList.AddRange(prototypesArr);

//instances
for (int t=0; t<trns.count; t++)
{
 Transition trn = trns.arr[t]; //using copy since it's changing in MoveRotateScale

 if (!data.area.active.Contains(trn.pos)) continue; //skipping out-of-active area
 if (PositioningSettings.SkipOnBiome(ref trn, output.biomeBlend, biomeMask, data.random)) continue;

 output.posSettings.MoveRotateScale(ref trn, data);

 float rnd = random.Random(trn.hash);
 int index = (int)(rnd*output.prefabs.Length);

 TreeInstance tree = new TreeInstance();

 tree.position = (trn.pos - (Vector3)data.area.active.worldPos) / data.area.active.worldSize.x;
 if (tree.position.x < 0 || tree.position.z < 0 ||
  tree.position.x > 1 || tree.position.z > 1)
   continue;
```

```
    tree.position.y = trn.pos.y / data.globals.height; //trees should be in 0-1 range

    tree.rotation = trn.Yaw;
    tree.widthScale = trn.scale.x; // + trs.scale.z)/2;
    tree.heightScale = trn.scale.y;
    tree.prototypeIndex = prototypesCount + index;
    tree.color = output.color;
    tree.lightmapColor = output.lightmapColor;

    instancesList.Add(tree);
   }

   prototypesCount += output.prefabs.Length;
  }

  //RemoveNullPrototypes(prototypesList, instancesList); //could not be executed in thread

  //pushing to apply
  if (stop!=null && stop.stop) return;
  ApplyTreesData applyData = new ApplyTreesData() { treePrototypes=prototypesList.ToArray(), treeInsta
  Graph.OnOutputFinalized?.Invoke(typeof(TreesOutput), data, applyData, stop);
  data.MarkApply(applyData);
 }


 public class ApplyTreesData : IApplyData
```

```
{
 public TreeInstance[] treeInstances;  //tree positions use 0-1 range (percent relatively to terrain)
 public TreePrototype[] treePrototypes;

 public void Read (Terrain terrain)
 {
  TerrainData data = terrain.terrainData;
  treeInstances = data.treeInstances;
  treePrototypes = data.treePrototypes;
 }

 public void Apply (Terrain terrain)
 {
  if (treePrototypes.Contains( p=>p.prefab==null ))
   RemoveNullPrototypes(ref treePrototypes, ref treeInstances);

  if (treePrototypes.Length == 0  &&  terrain.terrainData.treeInstanceCount != 0)
  {
   terrain.terrainData.treeInstances = new TreeInstance[0]; //setting instances first
   terrain.terrainData.treePrototypes = new TreePrototype[0];
  }

  terrain.terrainData.treePrototypes = treePrototypes;
  terrain.terrainData.treeInstances = treeInstances;
 }
```

```
 public int Resolution {get{ return 0; }}
}



public static void RemoveNullPrototypes (List<TreePrototype> prototypes, List<TreeInstance> instances)
{
 Dictionary<int,int> indexToOptimized = new Dictionary<int, int>();

 int originalPrototypesCount = prototypes.Count;
 int counter = 0;
 for (int p=0; p<originalPrototypesCount; p++)
  if (prototypes[p].prefab != null)
  {
   indexToOptimized.Add(p,counter);
   counter++;
  }

 for (int p=originalPrototypesCount-1; p>=0; p--)
  if (prototypes[p].prefab == null)
   prototypes.RemoveAt(p);

 for (int i=instances.Count-1; i>=0; i--)
 {
  if (!indexToOptimized.TryGetValue(instances[i].prototypeIndex, out int optimizedIndex))
   instances.RemoveAt(i);
```

```
  else if (instances[i].prototypeIndex != optimizedIndex)
  {
   TreeInstance instance = instances[i];
   instance.prototypeIndex = optimizedIndex;
   instances[i] = instance;
  }
 }
}

public static void RemoveNullPrototypes (ref TreePrototype[] prototypes, ref TreeInstance[] instances)
{
 List<TreePrototype> prototypesList = new List<TreePrototype>(prototypes);
 List<TreeInstance> instancesList = new List<TreeInstance>(instances);
 RemoveNullPrototypes(prototypesList, instancesList);
 prototypes = prototypesList.ToArray();
 instances = instancesList.ToArray();
}

public override void ClearApplied (TileData data, Terrain terrain)
{
 if (posSettings == null)
  posSettings = CreatePosSettings(this);

 TerrainData terrainData = terrain.terrainData;
 TreePrototype[] prototypes = terrainData.treePrototypes;
 TreeInstance[] instances = terrainData.treeInstances;
```

```
List<TreePrototype> newPrototypes = new List<TreePrototype>();
Dictionary<int,int> prototypeNumsLut = new Dictionary<int, int>();  //old -> new prototype number. If not c
for (int num=0; num<prototypes.Length; num++)
{
 bool contains = false;  //if terrain tree prototype contains in this generator
 for (int p=0; p<prefabs.Length; p++)
 {
  if (prototypes[num].prefab == prefabs[p] && prototypes[num].bendFactor < bendFactor+0.0001f && prot
  {
   contains = true;
   break;
  }
 }

 if (!contains)
 {
  prototypeNumsLut.Add(num, newPrototypes.Count);
  newPrototypes.Add(prototypes[num]);
 }
}

List<TreeInstance> newInstances = new List<TreeInstance>();
for (int i=0; i<instances.Length; i++)
{
 if (prototypeNumsLut.TryGetValue(instances[i].prototypeIndex, out int newIndex))
```

```
    {
      TreeInstance instance = instances[i];
      instance.prototypeIndex = newIndex;
      newInstances.Add(instance);
    }
   }

   terrainData.treeInstances = newInstances.ToArray();
   terrainData.treePrototypes = newPrototypes.ToArray();
  }
 }
}
```

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using MapMagic.Products;

namespace MapMagic.Nodes
{
 [GeneratorMenu (menu = "Objects/Portals", name = "Enter", iconName = "GeneratorIcons/PortalIn", lookL
 [GeneratorMenu (menu = "Objects/Portals", name ="Exit", iconName = "GeneratorIcons/PortalOut", lookLi
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Terrains;
using MapMagic.Nodes;
using MapMagic.Nodes.ObjectsGenerators;

namespace MapMagic.Locks
{
 public class TreesData : ILockData
 {
  public TreeInstance[] lockInstances;  //tree positions use 0-1 range (percent relatively to terrain)
  public TreePrototype[] lockPrototypes;

  public Vector2D center; //int 0-1 range relatively to terrain
  public float radius;
  public float transition;


  public void Read (Terrain terrain, Lock lk)
  {
   Vector3 terrainPos = terrain.transform.position;
```

```
 TerrainData terrainData = terrain.terrainData;
 Vector3 terrainSize = terrainData.size;

 center = new Vector2D(
  (lk.worldPos.x-terrainPos.x)/terrainSize.x,
  (lk.worldPos.z-terrainPos.z)/terrainSize.z);
 radius = lk.worldRadius/terrainSize.x;
 transition = lk.worldTransition/terrainSize.x;

 lockInstances = TreesInRange(terrainData.treeInstances, center, radius+transition).ToArray();
 lockPrototypes = terrainData.treePrototypes;
}


public void WriteInThread (IApplyData applyData)
{
 if (! (applyData is TreesOutput.ApplyTreesData applyTreesData) ) return;

 TreeInstance[] generatedInstances = applyTreesData.treeInstances;
 TreePrototype[] generatedPrototypes = applyTreesData.treePrototypes;

 UnifyPrototypes(ref generatedPrototypes, ref generatedInstances, ref lockPrototypes, ref lockInstances);

 List<TreeInstance> newTrees = TreesOutRange(generatedInstances, center, radius+transition); //clearin
 newTrees.AddRange(lockInstances);
```

```
 applyTreesData.treeInstances = newTrees.ToArray();
 applyTreesData.treePrototypes = lockPrototypes;
}

public void WriteInApply (Terrain terrain, bool resizeTerrain=false)
{
 if (lockInstances == null || lockPrototypes == null) return; // Don't perform lock if nothing is stored

 terrain.terrainData.treePrototypes = lockPrototypes;

 List<TreeInstance> trees = TreesOutRange(terrain.terrainData.treeInstances, center, radius+transition);
 trees.AddRange(lockInstances);
 terrain.terrainData.treeInstances = trees.ToArray();
}

public void ApplyHeightDelta (Matrix srcHeights, Matrix dstHeights)
{
 Vector2D relRectStart = center-(radius+transition);
 float relRectSize = radius*2+transition*2;

 for (int i=0; i<lockInstances.Length; i++)
 {
  Vector2D relPos = ((Vector2D)lockInstances[i].position - relRectStart) / relRectSize;
  Vector2D matrixPos = new Vector2D(
   relPos.x*srcHeights.rect.size.x + srcHeights.rect.offset.x,
   relPos.z*srcHeights.rect.size.z + srcHeights.rect.offset.z);
```

```
  float hSrc = srcHeights.GetInterpolated(matrixPos.x, matrixPos.z);
  float hDst = dstHeights.GetInterpolated(matrixPos.x, matrixPos.z);
  float heightDelta = hDst-hSrc;

  lockInstances[i].position.y += heightDelta;
 }
}

public void ResizeFrom (ILockData src) {  }


private static List<TreeInstance> TreesInRange (TreeInstance[] srcInstances, Vector2D center, float radiu
{
 Vector2D min = new Vector2D(center.x-radius, center.z-radius);
 Vector2D max = new Vector2D(center.x+radius, center.z+radius);

 List<TreeInstance> dstInstances = new List<TreeInstance>();
 for (int i=0; i<srcInstances.Length; i++)
 {
  TreeInstance instance = srcInstances[i];

  Vector3 pos = instance.position;
  if (pos.x < min.x || pos.z < min.z ||
   pos.x > max.x || pos.z > max.z) continue;
```

```
  float dist = Mathf.Sqrt((pos.x-center.x)*(pos.x-center.x) + (pos.z-center.z)*(pos.z-center.z));
  if (dist > radius) continue;

  dstInstances.Add(instance);
 }

 return dstInstances;
}

private static List<TreeInstance> TreesOutRange (TreeInstance[] srcInstances, Vector2D center, float rad
/// TreesInRange inverted
{
 List<TreeInstance> dstInstances = new List<TreeInstance>();
 for (int i=0; i<srcInstances.Length; i++)
 {
  TreeInstance instance = srcInstances[i];

  Vector3 pos = instance.position;
  float dist = Mathf.Sqrt((pos.x-center.x)*(pos.x-center.x) + (pos.z-center.z)*(pos.z-center.z));
  if (dist < radius)
   continue;

  dstInstances.Add(instance);
 }

 return dstInstances;
```

```
}


private static void UnifyPrototypes (ref TreePrototype[] basePrototypes, ref TreeInstance[] baseInstances,
        ref TreePrototype[] addPrototypes, ref TreeInstance[] addInstances)
/// Makes both datas prototypes arrays equal, and the layers arrays relevant to prototypes (empty arrays)
/// Safe per-channel blend could be performed after this operation
{
 //guard if prototypes have not been changed
 if (ArrayTools.MatchExactly(basePrototypes, addPrototypes)) return;

 //creating array of unified prototypes
 List<TreePrototype> unifiedPrototypes = new List<TreePrototype>();
 unifiedPrototypes.AddRange(basePrototypes); //do not change the base prototypes order
 for (int p=0; p<addPrototypes.Length; p++)
 {
  if (!unifiedPrototypes.Contains(addPrototypes[p]))
   unifiedPrototypes.Add(addPrototypes[p]);
 }

 //lut to convert prototypes indexes
 Dictionary<int,int> baseToUnifiedIndex = new Dictionary<int, int>();
 Dictionary<int,int> addToUnifiedIndex = new Dictionary<int, int>();

 for (int p=0; p<basePrototypes.Length; p++)
  baseToUnifiedIndex.Add(p, unifiedPrototypes.IndexOf(basePrototypes[p]));  //should be 1,2,3,4,5, but do
```

```
   for (int p=0; p<addPrototypes.Length; p++)
    addToUnifiedIndex.Add(p, unifiedPrototypes.IndexOf(addPrototypes[p]));

   //re-creating base data
   for (int i=0; i<baseInstances.Length; i++)
    baseInstances[i].prototypeIndex = baseToUnifiedIndex[ baseInstances[i].prototypeIndex ];

   //re-creating add data
   for (int i=0; i<addInstances.Length; i++)
    addInstances[i].prototypeIndex = addToUnifiedIndex[ addInstances[i].prototypeIndex ];

   //saving prototypes
   baseProtototypes = unifiedPrototypes.ToArray();
   addPrototypes = unifiedPrototypes.ToArray();
  }
 }
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes.GUI;

namespace MapMagic.Nodes.GUI
{
 public class SplineEditors
 {
  [Draw.Editor(typeof(SplinesGenerators.Stamp200))]
  public static void DrawStamp (SplinesGenerators.Stamp200 gen)
  {
   using (Cell.Padded(1,1,0,0))
   {
    using (Cell.LineStd) Draw.Field(ref gen.algorithm, "Algorithm");

    if (gen.algorithm == SplinesGenerators.Stamp200.Algorithm.Flatten || gen.algorithm == SplinesGenerato
    {
```

```
 using (Cell.LineStd)
 {
  Draw.Field(ref gen.flatRange, "Flat Range");
  Cell.current.Expose(gen.id, "flatRange", typeof(float));
 }
 using (Cell.LineStd)
 {
  Draw.Field(ref gen.blendRange, "Blend Range");
  Cell.current.Expose(gen.id, "blendRange", typeof(float));
 }
}

if (gen.algorithm == SplinesGenerators.Stamp200.Algorithm.Detail || gen.algorithm == SplinesGenerator
{
 using (Cell.LineStd)
 {
  Draw.Field(ref gen.detailRange, "Detail Range");
  Cell.current.Expose(gen.id, "detailRange", typeof(float));
 }
 using (Cell.LineStd)
 {
  Draw.Field(ref gen.detail, "Detail Size");
  Cell.current.Expose(gen.id, "detail", typeof(float));
 }
}
```

```
  using (Cell.LineStd)
  {
   Draw.Field(ref gen.fallof, "Fallof");
   Cell.current.Expose(gen.id, "fallof", typeof(float));
  }
 }
}



[Draw.Editor(typeof(SplinesGenerators.Manual210))]
public static void DrawManualGenerator (SplinesGenerators.Manual210 gen)
{
 using (Cell.LinePx(0))
 LayersEditor.DrawLayers(ref gen.positions,
  onDraw: num =>
  {
   if (num>=gen.positions.Length) return; //on layer remove
   int iNum = gen.positions.Length-1 - num;

   Cell.EmptyLinePx(2);
   using (Cell.LineStd)
   {
    Cell.current.fieldWidth = 0.7f;
    Cell.EmptyRowPx(2);
    using (Cell.RowPx(15)) Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Layer") );
    using (Cell.Row)
```

```
    {
     using (Cell.LineStd) Draw.Field(ref gen.positions[num].x, "X");
     using (Cell.LineStd) Draw.Field(ref gen.positions[num].y, "Y");
     using (Cell.LineStd) Draw.Field(ref gen.positions[num].z, "Z");
    }

    Cell.current.Expose(gen.id, "positions", typeof(Vector3), arrIndex:num);

    Cell.EmptyRowPx(2);
   }
   Cell.EmptyLinePx(2);
  } );
}


[Draw.Editor(typeof(SplinesGenerators.Combine217))]
public static void BlendGeneratorEditor (SplinesGenerators.Combine217 gen)
{
 using (Cell.LinePx(20)) GeneratorDraw.DrawLayersAddRemove(gen, ref gen.layers, inversed:false);
 using (Cell.LinePx(0)) GeneratorDraw.DrawLayersThemselves(gen, gen.layers, inversed:false, layerEdit
}

private static void DrawCombineLayer (Generator tgen, int num)
{
 SplinesGenerators.Combine217 gen = (SplinesGenerators.Combine217)tgen;
 SplinesGenerators.Combine217.Layer layer = gen.layers[num];
```

```
 Cell.EmptyLinePx(2);

 using (Cell.LineStd)
 {
  using (Cell.RowPx(0))
   GeneratorDraw.DrawInlet(layer.inlet, gen);
  Cell.EmptyRowPx(10);

  using (Cell.RowPx(20)) Draw.Icon(UI.current.textures.GetTexture("DPUI/Icons/Layer"));

  using (Cell.Row) Draw.Label("Spline " + num);

  Cell.EmptyRowPx(3);
 }


 Cell.EmptyLinePx(2);
}


[Draw.Editor(typeof(SplinesGenerators.Adjust2113))]
public static void DrawObjectsAdjust (SplinesGenerators.Adjust2113 adj)
{
 using (Cell.Padded(1,1,0,0))
 {
```

```
if (!adj.useRandom)
{
 using (Cell.LinePx(0))
 using (Cell.Padded(2,0,0,0))
 {
  using (Cell.LineStd)
  {
   Draw.IconField(ref adj.height.x, "Height", UI.current.textures.GetTexture("DPUI/Icons/Height"));
   Cell.current.Expose(adj.id, "height", typeof(Vector2));
  }

  /*using (Cell.LineStd)
  {
   Draw.IconField(ref adj.offsetFront.x, "Front", UI.current.textures.GetTexture("DPUI/Icons/Front"));
   Cell.current.Expose(adj.id, "front", typeof(Vector2));
  }

  using (Cell.LineStd)
  {
   Draw.IconField(ref adj.offsetRight.x, "Right", UI.current.textures.GetTexture("DPUI/Icons/Right"));
   Cell.current.Expose(adj.id, "right", typeof(Vector2));
  }*/
 }
}

else
```

```
{
 using (Cell.LineStd)
 {
  Cell.current.fieldWidth = 0.6f;
  using (Cell.RowPx(16)) Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Height") );
  using (Cell.RowRel(0.5f))
   Draw.FieldDragIcon(ref adj.height.x);
  using (Cell.RowRel(0.5f))
   Draw.FieldDragIcon(ref adj.height.y);
  Cell.current.Expose(adj.id, "height", typeof(Vector2));
 }

 /*using (Cell.LineStd)
 {
  Cell.current.fieldWidth = 0.6f;
  using (Cell.RowPx(16)) Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Front") );
  using (Cell.RowRel(0.5f))
   Draw.FieldDragIcon(ref adj.offsetFront.x);
  using (Cell.RowRel(0.5f))
   Draw.FieldDragIcon(ref adj.offsetFront.y);
  Cell.current.Expose(adj.id, "front", typeof(Vector2));
 }

 using (Cell.LineStd)
 {
  Cell.current.fieldWidth = 0.6f;
```

```
  using (Cell.RowPx(16)) Draw.Icon( UI.current.textures.GetTexture("DPUI/Icons/Right") );
  using (Cell.RowRel(0.5f))
   Draw.FieldDragIcon(ref adj.offsetRight.x);
  using (Cell.RowRel(0.5f))
   Draw.FieldDragIcon(ref adj.offsetRight.y);
  Cell.current.Expose(adj.id, "right", typeof(Vector2));
 }*/

}

using (Cell.LineStd)
{
 using (Cell.Row) Draw.Label("Random Range");
 using (Cell.RowPx(18)) Draw.Toggle(ref adj.useRandom);
}

if (adj.useRandom)
 using (Cell.LineStd)
 {
  Draw.Field(ref adj.seed, "Seed");
  Cell.current.Expose(adj.id, "seed", typeof(int));
 }

Cell.EmptyLinePx(5);

using (Cell.LineStd) Draw.Field(ref adj.relativeness, "Relativity");
```

```
  }
 }


 }
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.Splines;
using Den.Tools.Matrices;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Products;

namespace MapMagic.Nodes.SplinesGenerators
{
 public abstract partial class Pathfinding
 {
  protected const float maxWeight = 1000000;
  protected const int maxIterations = 1000000;

  public float distanceFactor = 1;
  public float elevationFactor = 5;
  public float straightenFactor = 1;
  public float maxElevation = 10000;
 }
```

```
public class FixedListPathfinding : Pathfinding
{
 private class FixedList
 {
  public int[] arr;
  public int count;

  public FixedList (int capacity) { arr = new int[capacity]; }
  public FixedList (int[] arr) { this.arr = arr; }
 }

 private FixedList changedPoses = new FixedList(10000);
 private FixedList newChangedPoses = new FixedList(10000); //to swap with changedcoords each iteratio



 public Coord[] FindPathDijkstra (Coord from, Coord to, MatrixWorld heights, Matrix weights, Matrix2D<Co
 /// Returns null if path could not be found (in manhattan dist * 2 cells)
 {
  weights.Fill(maxWeight+1);
  dirs.Fill(new Coord());

  //calculating weights
  weights[to] = 0;

  int fromPos = heights.rect.GetPos(from);
  int toPos = heights.rect.GetPos(to);
```

```
changedPoses.arr[0] = toPos;
changedPoses.count = 1;

int maxIterations = Coord.DistanceManhattan(from, to) * 2;
int counter = 0;
bool pathFound = false;
Coord min = heights.rect.Min; Coord max = heights.rect.Max;
for (int i=0; i<maxIterations; i++)
{
 for (int c=0; c<changedPoses.count; c++)
 {
  int pos = changedPoses.arr[c];
  Coord coord = heights.rect.GetCoord(pos);
  CalcNearWeights(coord, heights, weights, dirs, changedPosesFixedList:newChangedPoses);
  counter ++;
 }

 if (weights.arr[fromPos] < maxWeight) { pathFound = true; break; }

 FixedList tempList = changedPoses;
 changedPoses = newChangedPoses;
 newChangedPoses = tempList;
 newChangedPoses.count = 0;
}
if (!pathFound) return null;
```

```
 //drawing path
 List<Coord> path = new List<Coord>();
 path.Add(from);
 Coord curr = from;
 for (int i=0; i<1000; i++)
 {
  curr -= dirs[curr];
  path.Add(curr);

  if (curr == to) break;
 }

 /*for (int i=0; i<path.Count; i++)
  weights[path[i]] = -1;
 weights.Multiply(0.005f);
 DebugGizmos.ToMatrixPreviewCopyMainthread(weights);*/

 return path.ToArray();
}


private void CalcNearWeights (Coord coord, MatrixWorld heights, Matrix weights, Matrix2D<Coord> dirs,
{
 Coord rectMin = heights.rect.offset; Coord rectMax = heights.rect.offset + heights.rect.size;
 if (coord.x < rectMin.x  ||  coord.x > rectMax.x-1  ||
```

```
 coord.z < rectMin.z  ||  coord.z > rectMax.z-1) return;

float pixelSize = heights.worldSize.x/heights.rect.size.x; //heights.PixelSize.x;

int pos = (coord.z-heights.rect.offset.z)*heights.rect.size.x + coord.x - heights.rect.offset.x;
float thisHeight = heights.arr[pos];
float thisWeight = weights.arr[pos];
if (thisWeight > maxWeight) return;

for (int nx=-1; nx<=1; nx++)
 for (int nz=-1; nz<=1; nz++)
 {
  if (nx==0 && nz==0) continue;

  if ((coord.x==rectMin.x && nx==-1) || (coord.x==rectMax.x-1 && nx==1) ||
   (coord.z==rectMin.z && nz==-1) || (coord.z==rectMax.z-1 && nz==1)) continue;

  int nPos = pos + heights.rect.size.x*nz + nx;
  float nWeight = weights.arr[nPos];
  float nNewWeight = CalcWeight(pos, nx, nz, heights, weights, dirs);

  if (nNewWeight < nWeight)
  {
   weights.arr[nPos] = nNewWeight;
   dirs.arr[nPos].x = nx;
   dirs.arr[nPos].z = nz;
```

```
      if (changedPosesFixedList!=null)
      {
       changedPosesFixedList.arr[changedPosesFixedList.count] = nPos;
       changedPosesFixedList.count++;
      }
     }
    }
   }

}


 public class ListPathfinding : Pathfinding
 /// Keeps the list of changed cell coordinates. The simplest implementation.
 {

  public Coord[] FindPathAstarFastList (Coord from, Coord to, MatrixWorld heights, Matrix weights, Matrix2
  {
int statIterations = 0;
int statDistEvaluations = 0;
int statDistChecks = 0;

   Matrix priorities = new Matrix(heights.rect);
   priorities.Fill(-1);
```

```
weights.Fill(maxWeight+1);
dirs.Fill(new Coord());

//calculating weights
weights[to] = 0;
priorities[to] = Coord.DistanceManhattan(from, to) * distanceFactor;

List<int> changedPoses = new List<int>(10000);
int fromPos = heights.rect.GetPos(from);
int toPos = heights.rect.GetPos(to);
changedPoses.Add(toPos);
Coord closestCoord = new Coord();

Coord min = heights.rect.Min; Coord max = heights.rect.Max;
for (int i=0; i<maxIterations; i++)
{
 //finding theoretically closest coord in changed
 int closestNum = 0;
 float closestPriority = float.MaxValue;

 int changedPosesCount = changedPoses.Count;
 for (int p=0; p<changedPosesCount; p++)
 {
  int pos = changedPoses[p];
  if (pos<0) continue;
```

```
 float priority = priorities.arr[pos];
 if (priority < 0)
 {
  Coord coord = heights.rect.GetCoord(pos);
  priority = Coord.DistanceManhattan(from, coord)*distanceFactor  +  weights.arr[pos];
  priorities.arr[pos] = priority;
  statDistEvaluations++;
 }

 if (priority < closestPriority)
 {
  closestPriority = priority;
  closestNum = p;
 }

 statDistChecks++;
}

closestCoord = heights.rect.GetCoord( changedPoses[closestNum] );
changedPoses[closestNum] = -1;

CalcNearWeights(closestCoord, heights, weights, dirs, changedPosesList:changedPoses);
statIterations++;

//if (changedPoses.Contains(fromPos)) { Debug.Log("AstarFastList " + i); break; }
```

```
 if (weights.arr[fromPos] < maxWeight) break;
}

Debug.Log("AstarFastList iterations:" + statIterations +
 " distChacks:" + statDistChecks +
 " distEvaluations:" + statDistEvaluations +
 " changedPosesCount:" + changedPoses.Count);

//drawing path
List<Coord> path = new List<Coord>();
path.Add(from);
Coord curr = from;
for (int i=0; i<1000; i++)
{
 curr -= dirs[curr];
 path.Add(curr);

 if (curr == to) break;
}

//debugging path
//weights.Multiply(0.005f);
//for (int i=0; i<path.Count; i++)
// weights[path[i]] = 2;
//DebugGizmos.ToMatrixPreviewCopyMainthread(weights);
```

```
 //debugging changed poses
 Matrix weightsCpy = new Matrix(weights);
 weightsCpy.Multiply(0.01f);
 weightsCpy.arr[fromPos] = 0f;
 weightsCpy.arr[toPos] = 0f;
 weightsCpy[closestCoord] = 0.999f;
 for (int j=0; j<changedPoses.Count; j++)
 {
  if (changedPoses[j] < 0) continue;
  weightsCpy.arr[changedPoses[j]] = -1;
 }
 //weightsCpy[closestCoord] = 0.999f;
 DebugGizmos.ToMatrixPreview(weightsCpy);

 return path.ToArray();
}
```

```
private void CalcNearWeights (Coord coord, MatrixWorld heights, Matrix weights, Matrix2D<Coord> dirs,
{
 if (coord.x <= heights.rect.offset.x  ||  coord.x >= heights.rect.offset.x+heights.rect.size.x-1  ||
  coord.z <= heights.rect.offset.z  ||  coord.z >= heights.rect.offset.z+heights.rect.size.z-1) return;
```

```
float pixelSize = heights.worldSize.x/heights.rect.size.x; //heights.PixelSize.x;

int pos = (coord.z-heights.rect.offset.z)*heights.rect.size.x + coord.x - heights.rect.offset.x;
float thisHeight = heights.arr[pos];
float thisWeight = weights.arr[pos];
if (thisWeight > maxWeight) return;

for (int nx=-1; nx<=1; nx++)
 for (int nz=-1; nz<=1; nz++)
 {
  if (nx==0 && nz==0) continue;

  int nPos = pos + heights.rect.size.x*nz + nx;
  float nWeight = weights.arr[nPos];
  float nNewWeight = CalcWeight(pos, nx, nz, heights, weights, dirs);


  if (nNewWeight < nWeight)
  {
   weights.arr[nPos] = nNewWeight;
   dirs.arr[nPos].x = nx;
   dirs.arr[nPos].z = nz;

   if (changedPosesList!=null && !changedPosesList.Contains(nPos))
   {
    int changedPosesCount = changedPosesList.Count;
```

```
      bool added = false;
      for (int c=0; c<changedPosesCount; c++)
       if (changedPosesList[c]<0)
       {
        changedPosesList[c] = nPos;
        added = true;
       }
      if (!added) changedPosesList.Add(nPos);
     }
    }
   }
 }
}


public class HashSetPathfinding : Pathfinding
{
 public Coord[] FindPathAstar (Coord from, Coord to, MatrixWorld heights, Matrix weights, Matrix2D<Coor
 {
  //Matrix priorities = new Matrix(heights.rect);
  //priorities.Fill(-1);

  weights.Fill(maxWeight+1);
  dirs.Fill(new Coord());

  //calculating weights
```

```
weights[to] = 0;
//priorities[to] = Coord.DistanceManhattan(from, to) * distanceFactor;

HashSet<Coord> changedCoords = new HashSet<Coord>();
changedCoords.Add(to);

Coord min = heights.rect.Min; Coord max = heights.rect.Max;
for (int i=0; i<100000000; i++)
{
 //finding theoretically closest coord in changed
 Coord closestCoord = to;
 float closestPriority = float.MaxValue;
 foreach (Coord c in changedCoords)
 {
  int pos = (c.z-heights.rect.offset.z)*heights.rect.size.x + c.x - heights.rect.offset.x;
  float priority = Coord.DistanceManhattan(from, c) * distanceFactor  +  weights.arr[pos]; // priorities[c];

  if (priority < closestPriority)
  {
   closestPriority = priority;
   closestCoord.x = c.x;
   closestCoord.z = c.z;

  }
 }
```

```
 Coord coord = closestCoord;
 changedCoords.Remove(closestCoord);

 CalcNearWeights(coord, heights, weights, dirs, changedCoords);

 if (changedCoords.Contains(from)) { Debug.Log("Astar " + i); break; }
}


//drawing path
List<Coord> path = new List<Coord>();
path.Add(from);
Coord curr = from;
for (int i=0; i<1000; i++)
{
 curr -= dirs[curr];
 path.Add(curr);

 if (curr == to) break;
}

for (int i=0; i<path.Count; i++)
 weights[path[i]] = -1;
weights.Multiply(0.005f);

return path.ToArray();
```

```
}


public Coord[] FindPathDijkstraHashSet (Coord from, Coord to, MatrixWorld heights, Matrix weights, Matr
{
 weights.Fill(maxWeight+1);
 dirs.Fill(new Coord());

 //calculating weights
 weights[to] = 0;

 HashSet<Coord> changedCoords = new HashSet<Coord>();
 changedCoords.Add(to);

 HashSet<Coord> newChangedCoords = new HashSet<Coord>(); //to swap with changedcoords each ite

 int counter = 0;
 Coord min = heights.rect.Min; Coord max = heights.rect.Max;
 for (int i=0; i<1000; i++)
 {
  int changedCoordsCount = changedCoords.Count;
  foreach (Coord c in changedCoords)
  {
   CalcNearWeights(c, heights, weights, dirs, newChangedCoords);
   counter ++;
  }
```

```
 if (changedCoords.Contains(from)) break;

 HashSet<Coord> tmp = changedCoords;
 changedCoords = newChangedCoords;
 newChangedCoords = tmp;
 newChangedCoords.Clear();
}

Debug.Log("DijkstraHashSet " + counter);

//drawing path
List<Coord> path = new List<Coord>();
path.Add(from);
Coord curr = from;
for (int i=0; i<1000; i++)
{
 curr -= dirs[curr];
 path.Add(curr);

 if (curr == to) break;
}

for (int i=0; i<path.Count; i++)
 weights[path[i]] = -1;
weights.Multiply(0.005f);
```

```
 DebugGizmos.ToMatrixPreviewCopyMainthread(weights);

 return path.ToArray();
}

public Coord[] FindPathDijkstraListTemp (Coord from, Coord to, MatrixWorld heights, Matrix weights, Mat
{
 weights.Fill(maxWeight+1);
 dirs.Fill(new Coord());

 //calculating weights
 weights[to] = 0;

 HashSet<Coord> changedCoords = new HashSet<Coord>();
 changedCoords.Add(to);

 HashSet<Coord> newChangedCoords = new HashSet<Coord>(); //to swap with changedcoords each ite

 Coord min = heights.rect.Min; Coord max = heights.rect.Max;
 for (int i=0; i<1000; i++)
 {
  int changedCoordsCount = changedCoords.Count;

  for (int j=0; j<changedCoordsCount; j++)
  {
```

```
  Coord closestCoord = to;
  foreach (Coord c in changedCoords)
   { closestCoord = c; break; }

  CalcNearWeights(closestCoord, heights, weights, dirs, newChangedCoords);
  changedCoords.Remove(closestCoord);

  if (changedCoords.Count==0) break;
  if (newChangedCoords.Contains(from)) break;
 }

 if (changedCoords.Contains(from)) break;

 HashSet<Coord> tmp = changedCoords;
 changedCoords = newChangedCoords;
 newChangedCoords = tmp;
 newChangedCoords.Clear();
}


//drawing path
List<Coord> path = new List<Coord>();
path.Add(from);
Coord curr = from;
for (int i=0; i<1000; i++)
{
```

```
 curr -= dirs[curr];
 path.Add(curr);

 if (curr == to) break;
}

for (int i=0; i<path.Count; i++)
 weights[path[i]] = -1;
weights.Multiply(0.005f);
DebugGizmos.ToMatrixPreviewCopyMainthread(weights);

return path.ToArray();
}



private void CalcNearWeights (Coord coord, MatrixWorld heights, Matrix weights, Matrix2D<Coord> dirs,
{
 if (coord.x <= heights.rect.offset.x  ||  coord.x >= heights.rect.offset.x+heights.rect.size.x-1  ||
  coord.z <= heights.rect.offset.z  ||  coord.z >= heights.rect.offset.z+heights.rect.size.z-1) return;

 float pixelSize = heights.worldSize.x/heights.rect.size.x; //heights.PixelSize.x;

 int pos = (coord.z-heights.rect.offset.z)*heights.rect.size.x + coord.x - heights.rect.offset.x;
 float thisHeight = heights.arr[pos];
 float thisWeight = weights.arr[pos];
```

```
  if (thisWeight > maxWeight) return;


  for (int nx=-1; nx<=1; nx++)
   for (int nz=-1; nz<=1; nz++)
   {
    if (nx==0 && nz==0) continue;

    int nPos = pos + heights.rect.size.x*nz + nx;
    float nWeight = weights.arr[nPos];
    float nNewWeight = CalcWeight(pos, nx, nz, heights, weights, dirs);


    if (nNewWeight < nWeight)
    {
     weights.arr[nPos] = nNewWeight;
     dirs.arr[nPos].x = nx;
     dirs.arr[nPos].z = nz;
    }
   }
 }

}


public class CellPathfinding : Pathfinding
/// Uses the matrix of combined values (weight, dir, etc)
```

```
{
 private struct PathCell
 {
  public float weight;
  public Coord dir;
  public bool active;
  public float weightLeft;
 }


 public Coord[] FindPathAstar (Coord from, Coord to, MatrixWorld heights, Matrix weights, Matrix2D<Coor
 {
  Matrix2D<PathCell> cells = new Matrix2D<PathCell>(heights.rect);

  for (int i=0; i<cells.arr.Length; i++)
  {
   cells.arr[i].weight = maxWeight+1;
   cells.arr[i].dir.x = 0;
   cells.arr[i].dir.z = 0;
   cells.arr[i].active = false;
  }

  Coord min = heights.rect.Min; Coord max = heights.rect.Max;

  for (int i=0; i<1000; i++)
  {
```

```
 //finding closest active cell both to from and to
 Coord closestCoord = new Coord();
 float closestSum = float.MaxValue;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
 {
  int pos = (z-heights.rect.offset.z)*heights.rect.size.x + x - heights.rect.offset.x;
  if (!cells.arr[pos].active) continue;

  float sum = cells.arr[pos].weight + cells.arr[pos].weightLeft;
  if (sum < closestSum)
  {
   closestSum = sum;
   closestCoord.x = x;
   closestCoord.z = z;
  }
 }

 //calculating weight to neigbour cells
 CalcNearWeights(closestCoord, heights, cells, null, from);

 if (cells[from].active) break;
}

//drawing path
```

```
 List<Coord> path = new List<Coord>();
 path.Add(from);
 Coord curr = from;
 for (int i=0; i<1000; i++)
 {
  curr -= cells[curr].dir;
  path.Add(curr);

  if (curr == to) break;
 }

 return path.ToArray();
}



public Coord[] FindPathDijkstra (Coord from, Coord to, MatrixWorld heights, Matrix weights, Matrix2D<Co
{
 Matrix2D<PathCell> cells = new Matrix2D<PathCell>(heights.rect);

 for (int i=0; i<cells.arr.Length; i++)
 {
  cells.arr[i].weight = maxWeight+1;
  cells.arr[i].dir.x = 0;
  cells.arr[i].dir.z = 0;
  cells.arr[i].active = false;
 }
```

```
var toCell = cells[to];
toCell.active = true;
toCell.weight = 0;
cells[to] = toCell;

Coord min = heights.rect.Min; Coord max = heights.rect.Max;
for (int i=0; i<1000; i++)
{
 Matrix2D<PathCell> newCells = new Matrix2D<PathCell>(cells);

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
 {
  int pos = (z-heights.rect.offset.z)*heights.rect.size.x + x - heights.rect.offset.x;
  if (!cells.arr[pos].active) continue;

  CalcNearWeights(new Coord(x,z), heights, cells, newCells, from);
 }

 cells = newCells;
}

//drawing path
List<Coord> path = new List<Coord>();
path.Add(from);
```

```
 Coord curr = from;
 for (int i=0; i<1000; i++)
 {
  curr -= cells[curr].dir;
  path.Add(curr);

  if (curr == to) break;
 }

 //for (int i=0; i<cells.arr.Length; i++)
 // weights.arr[i] = cells.arr[i].weight * 0.001f;
 //DebugGizmos.ToMatrixPreviewCopyMainthread(weights);

 return path.ToArray();
}


private void CalcNearWeights (Coord coord, MatrixWorld heights, Matrix2D<PathCell> cells,  Matrix2D<P
{
 int pos = (coord.z-heights.rect.offset.z)*heights.rect.size.x + coord.x - heights.rect.offset.x;
 cells.arr[pos].active = false;

 if (coord.x <= heights.rect.offset.x  ||  coord.x >= heights.rect.offset.x+heights.rect.size.x-1  ||
  coord.z <= heights.rect.offset.z  ||  coord.z >= heights.rect.offset.z+heights.rect.size.z-1) return;

 float pixelSize = heights.worldSize.x/heights.rect.size.x; //heights.PixelSize.x;
```

```
float thisHeight = heights.arr[pos];
float thisWeight = cells.arr[pos].weight;
if (thisWeight > maxWeight) return;

for (int nx=-1; nx<=1; nx++)
 for (int nz=-1; nz<=1; nz++)
 {
  if (nx==0 && nz==0) continue;

  int nPos = pos + heights.rect.size.x*nz + nx;
  float nWeight = cells.arr[nPos].weight;
  float nHeight = heights.arr[nPos];


  //calculating factors
  float diagonalFactor = 1;
  if (nx*nz!=0) diagonalFactor = 1.414213562373f;

  float elevation = nHeight - thisHeight;
  if (elevation < 0) elevation = -elevation;
  elevation *= heights.worldSize.y;
  elevation = elevation/(pixelSize*diagonalFactor)*0.8f + elevation/diagonalFactor*0.2f;

  float xDirDelta = cells.arr[pos].dir.x-nx;
  float zDirDelta = cells.arr[pos].dir.z-nz;
```

```
float dirDelta = xDirDelta*xDirDelta + zDirDelta*zDirDelta;



//passability
bool passable = elevation <= maxElevation;



//new weight
float nNewWeight;
if (passable)
 nNewWeight = thisWeight +
  diagonalFactor*distanceFactor +
  elevation*elevation*elevationFactor +
  dirDelta*straightenFactor;
else
 nNewWeight = maxWeight+1;



if (nNewWeight < nWeight)
{
 newCells.arr[nPos].weight = nNewWeight;
 newCells.arr[nPos].active = true;
 newCells.arr[nPos].dir.x = nx;
 newCells.arr[nPos].dir.z = nz;
 newCells.arr[nPos].weightLeft = Coord.DistanceManhattan(from, new Coord(coord.x+nx, coord.z+nz))
}
```

```
   }
  }
 }

public partial class Pathfinding
{
 protected float CalcWeight (int pos, int nx, int nz, MatrixWorld heights, Matrix weights, Matrix2D<Coord> c
 {
  int nPos = pos + heights.rect.size.x*nz + nx;
  float nWeight = weights.arr[nPos];
  float nHeight = heights.arr[nPos];
  float pixelSize = heights.worldSize.x/heights.rect.size.x;

  //calculating factors
  float diagonalFactor = 1;
  if (nx*nz!=0) diagonalFactor = 1.414213562373f;

  float elevation = nHeight - heights.arr[pos]; //thisHeight;
  if (elevation < 0) elevation = -elevation;
  elevation *= heights.worldSize.y;
  elevation = elevation/(pixelSize*diagonalFactor)*0.8f + elevation/diagonalFactor*0.2f;

  float xDirDelta = dirs.arr[pos].x-nx;
  float zDirDelta = dirs.arr[pos].z-nz;
  float dirDelta = xDirDelta*xDirDelta + zDirDelta*zDirDelta;
```

```
    //passability
    bool passable = elevation <= maxElevation;



    //new weight
    float nNewWeight;
    if (passable)
     nNewWeight = weights.arr[pos] +
      diagonalFactor*distanceFactor +
      elevation*elevation*elevationFactor +
      dirDelta*straightenFactor;
    else
     nNewWeight = maxWeight+1;

    return nNewWeight;
   }
  }
 }
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;

using Den.Tools;
using Den.Tools.Splines;
using Den.Tools.Matrices;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Products;

namespace MapMagic.Nodes.SplinesGenerators
{
 [System.Serializable]
 [GeneratorMenu (
  menu="Spline/Standard",
  name ="Manual",
  iconName="GeneratorIcons/Constant",
  colorType = typeof(SplineSys),
  disengageable = true,
  helpLink ="https://gitlab.com/denispahunov/mapmagic/wikis/map_generators/constant")]
 public class Manual210 : Generator, IOutlet<SplineSys>
 {
  public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
```

```
 public Vector3[] positions = new Vector3[1];


 public override void Generate (TileData data, StopToken stop)
 {
  if (!enabled) return;


  SplineSys spline = new SplineSys();
  Line line = new Line();
  line.SetNodes(positions);
  spline.AddLine(line);


  data.StoreProduct(this, spline);
 }
}



[System.Serializable]
[GeneratorMenu (
 menu="Spline/Standard",
 name ="Interlink",
 iconName="GeneratorIcons/Constant",
 colorType = typeof(SplineSys),
 disengageable = true,
 helpLink ="https://gitlab.com/denispahunov/mapmagic/wikis/map_generators/constant")]
public class Interlink200 : Generator, IInlet<TransitionsList>, IOutlet<SplineSys>
```

```
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on


 [Val("Input", "Inlet")]  public readonly Inlet<TransitionsList> input = new Inlet<TransitionsList>();


 [Val("Iterations")]  public int iterations = 8;
 [Val("Max Links")]  public int maxLinks = 4;
 [Val("Within Tile")] public bool withinTile = true;


 public enum Clamp { Off, Full, Active }
 [Val("Clamp")] public Clamp clamp;



 public IEnumerable<IInlet<object>> Inlets () { yield return input; }


 public override void Generate (TileData data, StopToken stop)
 {
  TransitionsList objs = data.ReadInletProduct(this);
  if (objs == null || !enabled) return;


  SplineSys spline = new SplineSys();


  //creating hash set
  PosTab posTab;
  if (withinTile)
   posTab = new PosTab((Vector3)data.area.active.worldPos, (Vector3)data.area.active.worldSize, 16);
```

```
 else
 {
  Vector3 min = objs.Min();  min -= Vector3.one;
  Vector3 max = objs.Max();  max += Vector3.one;
  posTab = new PosTab(min, max-min, 16);
 }
 posTab.Add(objs);

 if (stop != null && stop.stop) return;
 GabrielGraph(posTab, spline, maxLinks:maxLinks, triesPerObj:iterations);

 if (stop != null && stop.stop) return;
 if (clamp == Clamp.Full) spline.Clamp((Vector3)data.area.full.worldPos, (Vector3)data.area.full.worldSize
 if (clamp == Clamp.Active) spline.Clamp((Vector3)data.area.active.worldPos, (Vector3)data.area.active.w

 if (stop != null && stop.stop) return;
 data.StoreProduct(this,  spline);
}

private struct LinkIds { public int id1; public int id2; public LinkIds(int i1, int i2) {id1=i1; id2=i2;} }

public static void GabrielGraph ( PosTab objs, SplineSys splineSys, int maxLinks=4, int triesPerObj=8 )
{
 triesPerObj = Mathf.Min(objs.totalCount-1, triesPerObj);
```

```
Dictionary<int,int[]> closestMap = new Dictionary<int, int[]>();


//bool CheckIfWithin (Transition trs)
//{
// return  trs.pos.x >= worldPos.x  &&  trs.pos.x <= worldPos.x+worldSize.x  &&
//   trs.pos.z >= worldPos.z  &&  trs.pos.z <= worldPos.x+worldSize.z;
//}


//filling closest map
foreach (Transition trs in objs.All())
{
 int[] closestIds = new int[triesPerObj];
 closestMap.Add(trs.id, closestIds);

 float minDist = 0.001f;
 for (int i=0; i<triesPerObj; i++)
 {
  Transition closest = objs.Closest(trs.pos.x, trs.pos.z, minDist); //, filterFn:CheckIfWithin);

  float curDistSq = (trs.pos.x-closest.pos.x)*(trs.pos.x-closest.pos.x) + (trs.pos.z-closest.pos.z)*(trs.pos.z-
  minDist = Mathf.Sqrt(curDistSq)  + 0.001f;

  closestIds[i] = closest.id;
 }
```

```
 //maybe could speed up by creating a list of points nearby and then sorting theese points by distance
}

//connecting
HashSet<LinkIds> connections = new HashSet<LinkIds>();
Dictionary<int,int> idToLinksCount = new Dictionary<int,int>();
foreach (var kvp in closestMap)
{
 int id1 = kvp.Key;
 int[] closestIds1 = kvp.Value;

 for (int num1=0; num1<closestIds1.Length; num1++)
 {
  int id2 = closestIds1[num1];
  if (id2 == 0) continue; // no more objects left during closest map

  int[] closestIds2 = closestMap[id2];
  int num2 = closestIds2.Find(id1);

  //if id1 not contains in closestIds2
  if (num2 < 0) continue;

  //if this link was not created earlier
  if (connections.Contains( new LinkIds(id1, id2) ) ||
   connections.Contains( new LinkIds(id2, id1) ) )
    continue;
```

```
  //if there is no common ids before num1 and num2
  bool hasCommonNodeCloser = false; //SNIPPET: the ideal case of using GOTO
  for (int i=0; i<num1; i++)
  {
   for (int j=0; j<num2; j++)
    if (closestIds1[i] == closestIds2[j]) { hasCommonNodeCloser = true; break; }
   if (hasCommonNodeCloser) break;
  }
  if (hasCommonNodeCloser) continue;

  //if the maximum number of connections reached
  idToLinksCount.TryGetValue(id1, out int linksCount1);
  idToLinksCount.TryGetValue(id2, out int linksCount2);
  if (linksCount1 >= maxLinks || linksCount2 >= maxLinks)
   continue;

  connections.Add( new LinkIds(id1, id2) );
  idToLinksCount.ForceAdd(id1, linksCount1 + 1);
  idToLinksCount.ForceAdd(id2, linksCount2 + 1);
 }
}

//converting connection links to positions
Dictionary<int,Vector3> idToPos = new Dictionary<int, Vector3>();
foreach (Transition trs in objs.All())
```

```
   idToPos.Add(trs.id, trs.pos);


  foreach (LinkIds ids in connections)
  {
   Vector3 pos1 = idToPos[ids.id1];
   Vector3 pos2 = idToPos[ids.id2];

   Line line = new Line(pos1, pos2);
   line.SetAllTangentTypes(Node.TangentType.linear);
   splineSys.AddLine(line);
  }

 }
}



[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Pathfinding", iconName=null, disengageable = true, h
public class Pathfinding200 : Generator, IMultiInlet, IOutlet<SplineSys>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Draft", "Inlet")]  public readonly Inlet<SplineSys> draftIn = new Inlet<SplineSys>();
 [Val("Height", "Inlet")] public readonly Inlet<MatrixWorld> heightIn = new Inlet<MatrixWorld>();
 public IEnumerable<IInlet<object>> Inlets () { yield return draftIn; yield return heightIn; }
```

```
[Val("Resolution")]   public int resolution = 32;
[Val("Distance Factor")]  public float distanceFactor = 1f;
[Val("Elevation Factor")]  public float elevationFactor = 1f;
[Val("Straighten Factor")]  public float straightenFactor = 1f;
[Val("Max Elevation")]  public float maxElevation = 0.1f;
[Val("Max Iterations")]  public int maxIterations = 1000000;
[Val("Weld Endpoints")]  public bool weldEndpoints = true;



public override void Generate (TileData data, StopToken stop)
{
 SplineSys src = data.ReadInletProduct(draftIn);
 MatrixWorld heights = data.ReadInletProduct(heightIn);
 if (src == null) return;
 if (heights == null) { data.StoreProduct(this, src); return; }

 if (stop!=null && stop.stop) return;
 MatrixWorld downsampledHeights = new MatrixWorld(new CoordRect(0,0,resolution,resolution), heights.
 MatrixOps.Resize(heights, downsampledHeights);

 if (stop!=null && stop.stop) return;
 SplineSys clamped = new SplineSys(src);
 clamped.Clamp(heights.worldPos, heights.worldSize);
 SplineSys dst = FindPaths(clamped, downsampledHeights);

 if (weldEndpoints)
```

```
  WeldEndpoints(dst);


 dst.Update();
 if (stop!=null && stop.stop) return;
 data.StoreProduct(this,  dst);
}



public SplineSys FindPaths (SplineSys src, MatrixWorld downsampledHeights)
{
 List<Line> dstLines = new List<Line>();

 Matrix weights = new Matrix(downsampledHeights.rect);
 Matrix2D<Coord> dirs = new Matrix2D<Coord>(downsampledHeights.rect);

 FixedListPathfinding pathfind = new FixedListPathfinding() {
  distanceFactor = distanceFactor,
  elevationFactor = elevationFactor,
  straightenFactor = straightenFactor,
  maxElevation = maxElevation };

 for (int l=0; l<src.lines.Length; l++)
 {
  Line line = src.lines[l];
  List<Vector3> newPath = null;
```

```
for (int s=0; s<line.segments.Length; s++)
{
 Vector3 fromWorld = line.segments[s].start.pos;
 Vector3 toWorld = line.segments[s].end.pos;

 //checking if this line lays within heights matrix, and clamping all of the line segments out of matrix
 Rect worldRect2D = new Rect(downsampledHeights.worldPos.x, downsampledHeights.worldPos.z, dow

 Vector2 fromWorld2D = fromWorld.V2();
 Vector2 toWorld2D = toWorld.V2();

 if (!worldRect2D.IntersectsLine(fromWorld2D, toWorld2D)) continue;
 worldRect2D.ClampLine(ref fromWorld2D, ref toWorld2D);

 fromWorld = fromWorld2D.V3();
 toWorld = toWorld2D.V3();

 //world to coordinates
 Coord fromCoord = downsampledHeights.WorldToPixel(fromWorld.x, fromWorld.z);
 Coord toCoord = downsampledHeights.WorldToPixel(toWorld.x, toWorld.z);

 fromCoord.ClampByRect(downsampledHeights.rect);
 toCoord.ClampByRect(downsampledHeights.rect);

 //pathfinding
 Coord[] pathCoord = pathfind.FindPathDijkstra(fromCoord, toCoord, downsampledHeights, weights, dirs
```

```
if (pathCoord == null) break;

//coords to world
Vector3[] pathWorld = new Vector3[pathCoord.Length];
for (int i=0; i<pathCoord.Length; i++)
 pathWorld[i] = downsampledHeights.PixelToWorld(pathCoord[i].x, pathCoord[i].z);

//slightly moving all nodes to make start and end match the src nodes
/*Vector3 startDelta = fromWorld - pathWorld[0];
Vector3 endDelta = toWorld - pathWorld[pathWorld.Length-1];
for (int i=0; i<pathCoord.Length; i++)
{
 float percent = 1f * i / (pathWorld.Length-1);
 pathWorld[i] += startDelta*(1-percent) + endDelta*percent;
}*/

//flooring nodes
//DebugGizmos.Clear("Path");
pathWorld[0].y = downsampledHeights.GetWorldInterpolatedValue(pathWorld[0].x, pathWorld[1].z) * do
for (int i=0; i<pathWorld.Length-1; i++)
{
 pathWorld[i+1].y = downsampledHeights.GetWorldInterpolatedValue(pathWorld[i+1].x, pathWorld[i+1].z
 //DebugGizmos.AddLine("Path", pathWorld[i], pathWorld[i+1]);
}

if (newPath == null) newPath = new List<Vector3>();
```

```
    newPath.AddRange(pathWorld);
   }

   if (newPath != null)
   {
    Line newLine = new Line();
    newLine.SetNodes(newPath.ToArray());

    dstLines.Add(newLine);
   }
  }

  SplineSys dst = new SplineSys();
  dst.lines = dstLines.ToArray();

  return dst;
 }

 public void WeldEndpoints (SplineSys src)
 {
  Vector3[] allEndpoints = new Vector3[src.lines.Length*2];
  for (int l=0; l<src.lines.Length; l++)
  {
   if (src.lines[l].segments.Length == 0)
    continue;
```

```
 allEndpoints[l*2] = src.lines[l].startPos;
 allEndpoints[l*2+1] = src.lines[l].endPos;
}



bool[] processed = new bool[allEndpoints.Length];
bool[] samePos = new bool[allEndpoints.Length];
for (int i=0; i<allEndpoints.Length; i++)
{
 if (processed[i])
  continue;

 //finding endpoint with same 2d pos, writing to samepos array
 samePos.Fill(false);
 Vector3 pos = allEndpoints[i];
 for (int j=0; j<allEndpoints.Length; j++)
 {
  if (processed[j])
   continue;

  float deltaX = allEndpoints[j].x - pos.x;
  float deltaZ = allEndpoints[j].z - pos.z;
  if (deltaX*deltaX + deltaZ*deltaZ < 0.1f)
   samePos[j] = true;
 }
```

```
 //getting avg height
 float sumHeight = 0;
 int sumNum = 0;
 for (int j=0; j<allEndpoints.Length; j++)
 {
  if (!samePos[j])
   continue;

  sumHeight += allEndpoints[j].y;
  sumNum ++;
 }
 float avgHeight = sumHeight / sumNum;

 //storing avg height in endpoints array
 for (int j=0; j<allEndpoints.Length; j++)
 {
  if (!samePos[j])
   continue;

  allEndpoints[j].y = avgHeight;
 }
}

//writing endpoints back to splines
for (int l=0; l<src.lines.Length; l++)
{
```

```
   if (src.lines[l].segments.Length == 0)
    continue;

    src.lines[l].startPos = allEndpoints[l*2];
    src.lines[l].endPos = allEndpoints[l*2+1];
  }
 }
}
```

```
[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Stroke", iconName="GeneratorIcons/Constant", disen
 colorType = typeof(SplineSys),
 helpLink ="https://gitlab.com/denispahunov/mapmagic/wikis/map_generators/constant")]
public class Stroke200 : Generator, IInlet<SplineSys>, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Width")]   public float width = 10;
 [Val("Hardness")] public float hardness = 0.0f;

 public override void Generate (TileData data, StopToken stop)
 {
  SplineSys splineSys = data.ReadInletProduct(this);
  if (splineSys == null || !enabled) return;
```

```
 //stroking
 if (stop!=null && stop.stop) return;
 MatrixWorld strokeMatrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.wo
 SplineMatrixOps.Stroke(splineSys, strokeMatrix, white:true, antialiased:true);

 //spreading
 if (stop!=null && stop.stop) return;
 MatrixWorld spreadMatrix = Spread(strokeMatrix, width);

 //hardness
 if (hardness > 0.0001f)
 {
  float h = 1f/(1f-hardness);
  if (h > 9999) h=9999; //infinity if hardness is 1

  spreadMatrix.Multiply(h);
  spreadMatrix.Clamp01();
 }

 if (stop!=null && stop.stop) return;
 data.StoreProduct(this,  spreadMatrix);
}
```

```
public static MatrixWorld Spread (MatrixWorld matrix, float range)
{
 MatrixWorld spreadMatrix;
 float pixelRange = range / matrix.PixelSize.x;

 if (pixelRange < 1) //if less than a pixel making line less noticable
 {
  spreadMatrix = matrix;
  spreadMatrix.Multiply(pixelRange);
 }

 else //spreading the usual way
 {
  spreadMatrix = new MatrixWorld(matrix);
  MatrixOps.SpreadLinear(matrix, spreadMatrix, subtract:1f/pixelRange, diagonals:true, quarters:true);

 }

 return spreadMatrix;
}

public static void SpreadOrMultiply (MatrixWorld src, MatrixWorld dst, float range)
{
 float pixelRange = range / src.PixelSize.x;
```

```
  if (pixelRange < 1) //if less than a pixel making line less noticable
   dst.Multiply(pixelRange);
  else
   MatrixOps.SpreadLinear(src, dst, subtract:1f/pixelRange);
 }
}


[System.Serializable]
[GeneratorMenu (
 menu="Spline/Standard",
 name ="Align",
 iconName="GeneratorIcons/Constant",
 colorType = typeof(SplineSys),
 disengageable = true,
 helpLink ="https://gitlab.com/denispahunov/mapmagic/wikis/map_generators/constant")]
public class Align200 : Generator, IMultiInlet, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Spline", "Inlet")] public readonly Inlet<SplineSys> splineIn = new Inlet<SplineSys>();
 [Val("Height", "Inlet")] public readonly Inlet<MatrixWorld> heightIn = new Inlet<MatrixWorld>();
 public IEnumerable<IInlet<object>> Inlets () { yield return splineIn; yield return heightIn; }

 [Val("Range")] public float range = 30;
 [Val("Flat")] public float flat = 0.25f;
```

```
[Val("Detail")] public float detail = 0f;



public override void Generate (TileData data, StopToken stop)
{
 SplineSys splineSys = data.ReadInletProduct(splineIn);
 MatrixWorld heightMatrix = data.ReadInletProduct(heightIn);
 if (splineSys == null) return;
 if (!enabled || heightMatrix == null) { data.StoreProduct(this, null); return; }

 if (stop!=null && stop.stop) return;
 MatrixWorld splineMatrix = Stamp(heightMatrix, splineSys, stop);

 if (stop!=null && stop.stop) return;
 data.StoreProduct(this,  splineMatrix);
}



private MatrixWorld Stamp (MatrixWorld srcHeights, SplineSys splineSys, StopToken stop)
{
 //contours matrix
 if (stop!=null && stop.stop) return null;
 MatrixWorld lineContours = new MatrixWorld(srcHeights.rect, srcHeights.worldPos, srcHeights.worldSize
 SplineMatrixOps.Stroke(splineSys, lineContours, white:true, antialiased:true);
```

```
//line heights matrix
if (stop!=null && stop.stop) return null;
MatrixWorld lineHeightsSrc = new MatrixWorld(srcHeights.rect, srcHeights.worldPos, srcHeights.worldSi
SplineMatrixOps.Stroke(splineSys, lineHeightsSrc, padOnePixel:true);
MatrixWorld lineHeights = new MatrixWorld(lineHeightsSrc); //TODO: use same src/dst matrix in padding
MatrixOps.PaddingMipped(lineHeightsSrc, lineContours, lineHeights);



//distances matrix
if (stop!=null && stop.stop) return null;
MatrixWorld lineDistances = new MatrixWorld(lineContours);

float pixelRange = range / srcHeights.PixelSize.x;

if (pixelRange < 1) //if less than a pixel making line less noticable
 lineDistances.Multiply(pixelRange);
else
 MatrixOps.SpreadLinear(lineContours, lineDistances, subtract:1f/pixelRange);



//saving detail matrix if detail is used (and then operating on lower-detail)
if (stop!=null && stop.stop) return null;
MatrixWorld detailMatrix = null;
if (detail > 0.00001f)
{
 int downsample = (int)(detail+1);
```

```
 float blur = (detail+1) - downsample;

 MatrixWorld originalHeights = srcHeights;
 srcHeights = new MatrixWorld(srcHeights); //further operating on blurred matrix
 MatrixOps.DownsampleBlur(srcHeights, downsample, blur);

 detailMatrix = new MatrixWorld(srcHeights); //taking blurred matrix
 detailMatrix.InvSubtract(originalHeights); //and subtracting src (non-blurred) from it
}



//blending line heights with terrain heights
if (stop!=null && stop.stop) return null;
for (int i=0; i<srcHeights.arr.Length; i++)  //TODO: replace with matrix mix
{
 float dist = lineDistances.arr[i];
 if (dist == 0) { lineHeights.arr[i] = srcHeights.arr[i]; continue; }
 if (1-dist < flat) continue;

 float percent = dist / (1-flat);
 percent = 3*percent*percent - 2*percent*percent*percent;

 lineHeights.arr[i] = lineHeights.arr[i]*percent + srcHeights.arr[i]*(1-percent);
}

//applying detail
```

```
  if (detailMatrix != null)
   lineHeights.Add(detailMatrix);

  return lineHeights;
 }
}

[System.Serializable]
[GeneratorMenu (
 menu="Spline/Standard",
 name ="Stamp",
 iconName="GeneratorIcons/Constant",
 colorType = typeof(SplineSys),
 disengageable = true,
 helpLink ="https://gitlab.com/denispahunov/mapmagic/wikis/map_generators/constant")]
public class Stamp200 : Generator, IMultiInlet, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Spline", "Inlet")] public readonly Inlet<SplineSys> splineIn = new Inlet<SplineSys>();
 [Val("Height", "Inlet")] public readonly Inlet<MatrixWorld> heightIn = new Inlet<MatrixWorld>();
 public IEnumerable<IInlet<object>> Inlets () { yield return splineIn; yield return heightIn; }

 public enum Algorithm { Flatten, Detail, Both };
 public Algorithm algorithm;
```

```
public float flatRange = 2;
public float blendRange = 16;
public float detailRange = 32;
public int detail = 1;
public float fallof = 1;



public override void Generate (TileData data, StopToken stop)
{
 SplineSys splineSys = data.ReadInletProduct(splineIn);
 MatrixWorld heightMatrix = data.ReadInletProduct(heightIn);
 if (splineSys == null ||  heightMatrix == null) return;
 if (!enabled) { data.StoreProduct(this, heightMatrix); return; }

 if (stop!=null && stop.stop) return;
 MatrixWorld splineMatrix = Stamp(heightMatrix, splineSys, stop);

 if (stop!=null && stop.stop) return;
 data.StoreProduct(this,  splineMatrix);
}



private MatrixWorld Stamp (MatrixWorld srcHeights, SplineSys splineSys, StopToken stop)
{
 MatrixWorld dstHeights = new MatrixWorld(srcHeights);
```

```
//transforming ranges to relative (0-1)
float maxRange = Mathf.Max(detailRange, blendRange);
float relDetailRange = 1 - detailRange/maxRange;
float relBlendRange = 1 - blendRange/maxRange;
float relFlatRange = 1 - flatRange/maxRange;

//contours matrix
if (stop!=null && stop.stop) return null;
MatrixWorld lineContours = new MatrixWorld(srcHeights.rect, srcHeights.worldPos, srcHeights.worldSize
SplineMatrixOps.Stroke(splineSys, lineContours, white:true, antialiased:true);


//line heights matrix
if (stop!=null && stop.stop) return null;
MatrixWorld lineHeightsSrc = new MatrixWorld(srcHeights.rect, srcHeights.worldPos, srcHeights.worldSi
SplineMatrixOps.Stroke(splineSys, lineHeightsSrc, padOnePixel:true);
MatrixWorld lineHeights = new MatrixWorld(lineHeightsSrc); //TODO: use same src/dst matrix in padding
MatrixOps.PaddingMipped(lineHeightsSrc, lineContours, lineHeights);


//distances matrix
if (stop!=null && stop.stop) return null;
MatrixWorld lineDistances = new MatrixWorld(lineContours);

float pixelRange = maxRange / srcHeights.PixelSize.x;
```

```
if (pixelRange < 1) //if less than a pixel making line less noticeable
 lineDistances.Multiply(pixelRange);
else
 MatrixOps.SpreadLinear(lineContours, lineDistances, subtract:1f/pixelRange);

//adding gamma to fallof - needed for Brush mainly
if (fallof < 0.999f)
{
 lineDistances.InvertOne();
 lineDistances.Pow(fallof);
 lineDistances.InvertOne();
}

//applying detail
if (stop!=null && stop.stop) return null;
if ((algorithm == Algorithm.Detail || algorithm == Algorithm.Both) && detail > 0.0001f)
{
 int downsample = (int)(detail+1);
 float blur = (detail+1) - downsample;
 MatrixOps.DownsampleBlur(dstHeights, detail, 1);
 //MatrixOps.GaussianBlur(dstHeights, detail);

 MatrixWorld detailMatrix = new MatrixWorld(dstHeights);  //taking blurred matrix
 detailMatrix.InvSubtract(srcHeights); //and subtracting non-blurred from it

 dstHeights.Mix(lineHeights, lineDistances, 0, relDetailRange, maskInvert:false, fallof:false, opacity:1);  //s
```

```
   dstHeights.Add(detailMatrix); //and returning details back
  }



  //applying fallof
  if (stop!=null && stop.stop) return null;
  if (algorithm == Algorithm.Flatten || algorithm == Algorithm.Both)
  {
   dstHeights.Mix(lineHeights, lineDistances, relBlendRange, relFlatRange, maskInvert:false, fallof:false, o
  }

  return dstHeights;
 }
}




[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Optimize", iconName=null, disengageable = true, help
public class Optimize200 : Generator, IInlet<SplineSys>, IOutlet<SplineSys>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on


 [Val("Split")] public int split = 3;
 [Val("Deviation")] public float deviation = 1;
```

```
 public override void Generate (TileData data, StopToken stop)
 {
  SplineSys src = data.ReadInletProduct(this);
  if (src == null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }

  if (stop!=null && stop.stop) return;
  SplineSys dst = new SplineSys(src);
  dst.Subdivide(split);
  dst.UpdateTangents();
  dst.Optimize(deviation);
  dst.Update();

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this,  dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Relax", iconName=null, disengageable = true, helpLin
public class Relax200 : Generator, IInlet<SplineSys>, IOutlet<SplineSys>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
```

```
 [Val("Spline", "Inlet")] public readonly Inlet<SplineSys> splineIn = new Inlet<SplineSys>();


 [Val("Blur")] public float blur = 1;
 [Val("Iterations")] public int iterations = 3;


 public override void Generate (TileData data, StopToken stop)
 {
  SplineSys src = data.ReadInletProduct(this);
  if (src == null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }


  if (stop!=null && stop.stop) return;
  SplineSys dst = new SplineSys(src);
  dst.Relax(blur, iterations);
  dst.Update();


  if (stop!=null && stop.stop) return;
  data.StoreProduct(this,  dst);
 }
}



[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Combine", iconName="GeneratorIcons/Blend", diseng
 helpLink = "https://gitlab.com/denispahunov/mapmagic/-/wikis/MatrixGenerators/Blend")]
public class Combine217 : Generator, IMultiInlet, IOutlet<SplineSys>
```

```
{
 public class Layer
 {
  public readonly Inlet<SplineSys> inlet = new Inlet<SplineSys>();
 }

 public Layer[] layers = new Layer[] { new Layer(), new Layer() };
 public Layer[] Layers => layers;
 public void SetLayers(object[] ls) => layers = Array.ConvertAll(ls, i=>(Layer)i);

 public IEnumerable<IInlet<object>> Inlets()
 {
  for (int i=0; i<layers.Length; i++)
   yield return layers[i].inlet;
 }

 public override void Generate (TileData data, StopToken stop)
 {
  if (stop!=null && stop.stop) return;
  if (!enabled) return;

  List<Line> lines = new List<Line>();

  if (stop!=null && stop.stop) return;
  for (int i = 0; i < layers.Length; i++)
  {
```

```
   Layer layer = layers[i];
   if (layer.inlet == null) continue;

   SplineSys otherSpline = data.ReadInletProduct(layer.inlet);
   if (otherSpline == null) continue;

   foreach (Line otherLine in otherSpline.lines)
   {
    Line copyLine = new Line(otherLine);
    lines.Add(copyLine);
   }
  }

  SplineSys spline = new SplineSys();
  spline.AddLines(lines.ToArray());

  data.StoreProduct(this, spline);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Weld Close", iconName=null, disengageable = true, h
public class WeldClose200 : Generator, IInlet<SplineSys>, IOutlet<SplineSys>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
```

```
 [Val("Spline", "Inlet")] public readonly Inlet<SplineSys> splineIn = new Inlet<SplineSys>();

 [Val("Threshold")] public float threshold = 25;

 public override void Generate (TileData data, StopToken stop)
 {
  SplineSys src = data.ReadInletProduct(this);
  if (src == null) return;
  if (!enabled) { data.StoreProduct(this, src); return; }

  if (stop!=null && stop.stop) return;
  SplineSys dst = new SplineSys(src);
  dst.WeldCloseLines(threshold);
  dst.Update();

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this,  dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Floor", iconName=null, disengageable = true, helpLink
public class Floor200 : Generator, IMultiInlet, IOutlet<SplineSys>
{
```

```
public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

[Val("Spline", "Inlet")] public readonly Inlet<SplineSys> splineIn = new Inlet<SplineSys>();
[Val("Spline", "Height")] public readonly Inlet<MatrixWorld> heightIn = new Inlet<MatrixWorld>();
public IEnumerable<IInlet<object>> Inlets () { yield return splineIn; yield return heightIn; }

public override void Generate (TileData data, StopToken stop)
{
 SplineSys src = data.ReadInletProduct(splineIn);
 MatrixWorld heights = data.ReadInletProduct(heightIn);
 if (src == null) return;
 if (!enabled || heights==null) { data.StoreProduct(this, src); return; }

 if (stop!=null && stop.stop) return;
 SplineSys dst = new SplineSys(src);

 FloorSplines(dst, heights);
 dst.Update();

 if (stop!=null && stop.stop) return;
 data.StoreProduct(this,  dst);


 DebugGizmos.Clear("Spline");
 foreach (Segment segment in dst.lines[0].segments)
  DebugGizmos.DrawLine("Spline", segment.start.pos, segment.end.pos, Color.white, additive:true);
```

```
}

public static void FloorSplines (SplineSys dst, MatrixWorld heights)
{
 foreach (Line line in dst.lines)
 {
  for (int s=0; s<line.segments.Length; s++)
   line.segments[s].start.pos.y = FloorPoint(line.segments[s].start.pos, heights);

  line.segments[line.segments.Length-1].end.pos.y = FloorPoint(line.segments[line.segments.Length-1].en
 }
}

public static float FloorPoint (Vector3 pos, MatrixWorld heights)
{
 if (pos.x <= heights.worldPos.x) pos.x = heights.worldPos.x+0.001f;
 if (pos.x >= heights.worldPos.x +heights.worldSize.x) pos.x = heights.worldPos.x +heights.worldSize.x-0.

 if (pos.z <= heights.worldPos.z) pos.z = heights.worldPos.z+0.001f;
 if (pos.z >= heights.worldPos.z +heights.worldSize.z) pos.z = heights.worldPos.z +heights.worldSize.z-0.

 float h = heights.GetWorldInterpolatedValue(pos.x, pos.z);
 return h * heights.worldSize.y;
}
}
```

```
[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Avoid", iconName=null, disengageable = true, helpLin
public class Avoid200 : Generator, IMultiInlet, IOutlet<SplineSys>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Spline", "Inlet")] public readonly Inlet<SplineSys> splineIn = new Inlet<SplineSys>();
 [Val("Spline", "Positions")] public readonly Inlet<TransitionsList> objsIn = new Inlet<TransitionsList>();
 public IEnumerable<IInlet<object>> Inlets () { yield return splineIn; yield return objsIn; }

 [Val("Distance")] public float distance = 10;
 [Val("Size Factor")] public int sizeFactor = 0;
 [Val("Iterations")] public int iterations = 10;

 public override void Generate (TileData data, StopToken stop)
 {
  SplineSys src = data.ReadInletProduct(splineIn);
  TransitionsList objs = data.ReadInletProduct(objsIn);
  if (src == null) return;
  if (!enabled || objs==null) { data.StoreProduct(this, src); return; }

  if (stop!=null && stop.stop) return;
  SplineSys dst = new SplineSys(src);

  Vector3[] points = new Vector3[objs.count];
```

```
  float[] ranges = new float[objs.count];

  for (int o=0; o<objs.count; o++)
  {
   points[o] = objs.arr[o].pos;
   ranges[o] = distance * (1-sizeFactor + objs.arr[o].scale.x*sizeFactor);
  }

  dst.SplitNearPoints(points, ranges, true, startEndProximityFactor:1f, maxIterations:iterations);
  dst.PushStartEnd(points, ranges, true, 0.5f);
  dst.PushStartEnd(points, ranges, true, 1.01f);

  dst.Update();

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this,  dst);
 }
}

[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Push", iconName=null, disengageable = true,
 colorType = typeof(SplineSys),
 helpLink ="https://gitlab.com/denispahunov/mapmagic/wikis/map_generators/constant")]
public class Push200 : Generator, IMultiInlet, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on
```

```
[Val("Spline", "Objects")] public readonly Inlet<TransitionsList> objsIn = new Inlet<TransitionsList>();
[Val("Spline", "Spline")] public readonly Inlet<SplineSys> splineIn = new Inlet<SplineSys>();
public IEnumerable<IInlet<object>> Inlets () { yield return objsIn; yield return splineIn; }

[Val("Distance")] public float distance = 10;
[Val("Size Factor")] public int sizeFactor = 0;
[Val("Iterations")] public int iterations = 10;

public override void Generate (TileData data, StopToken stop)
{
 SplineSys spline = data.ReadInletProduct(splineIn);
 TransitionsList objs = data.ReadInletProduct(objsIn);
 if (objs == null) return;
 if (!enabled || spline==null) { data.StoreProduct(this, objs); return; }

 if (stop!=null && stop.stop) return;

 Vector3[] points = new Vector3[objs.count];
 float[] ranges = new float[objs.count];

 for (int o=0; o<objs.count; o++)
 {
  points[o] = objs.arr[o].pos;
  ranges[o] = distance * (1-sizeFactor + objs.arr[o].scale.x*sizeFactor);
 }
```

```
  for (int i=0; i<iterations; i++)
  {
   float distFactor = Mathf.Sqrt((i+1f)/iterations);
   spline.PushPoints(points, ranges, horizontalOnly:true, distFactor:distFactor);
  }

  TransitionsList dst = new TransitionsList(objs);
  for (int o=0; o<objs.count; o++)
   dst.arr[o].pos = points[o];

  if (stop!=null && stop.stop) return;
  data.StoreProduct(this,  dst);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Isoline", iconName=null, disengageable = true,
 colorType = typeof(SplineSys),
 helpLink ="https://gitlab.com/denispahunov/mapmagic/wikis/map_generators/constant")]
public class Isoline200 : Generator, IInlet<MatrixWorld>, IOutlet<SplineSys>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 //[Val("Distance")] public float distance = 10;
```

```
 //[Val("Size Factor")] public int sizeFactor = 0;
 //[Val("Iterations")] public int iterations = 10;

 public override void Generate (TileData data, StopToken stop)
 {
  MatrixWorld matrix = data.ReadInletProduct(this);
  if (matrix == null || !enabled) return;

  if (stop!=null && stop.stop) return;
  //data.StoreProduct(this,  matrix);
 }
}


[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Silhouette", iconName=null, disengageable = true,
 colorType = typeof(SplineSys),
 helpLink ="https://gitlab.com/denispahunov/mapmagic/wikis/map_generators/constant")]
public class Silhouette200 : Generator, IInlet<SplineSys>, IOutlet<MatrixWorld>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Width")] public float width = 10;
 //[Val("Size Factor")] public int sizeFactor = 0;
 //[Val("Iterations")] public int iterations = 10;
```

```
public override void Generate (TileData data, StopToken stop)
{
 SplineSys splineSys = data.ReadInletProduct(this);
 if (splineSys == null || !enabled) return;

 //stroke and spread
 if (stop!=null && stop.stop) return;
 MatrixWorld strokeMatrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.wo
 SplineMatrixOps.Stroke (splineSys, strokeMatrix, white:true, intensity:0.5f, antialiased:true);
 MatrixWorld spreadMatrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.w
 MatrixOps.SpreadLinear(strokeMatrix, spreadMatrix, subtract: spreadMatrix.PixelSize.x/width / 2);

 //silhouette
 if (stop!=null && stop.stop) return;
 MatrixWorld silhouetteMatrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full
 SplineMatrixOps.Stroke (splineSys, silhouetteMatrix, white:true, intensity:0.5f, antialiased:false, padOneP
 SplineMatrixOps.Silhouette(splineSys, silhouetteMatrix, silhouetteMatrix);

 //combining
 if (stop!=null && stop.stop) return;
 SplineMatrixOps.CombineSilhouetteSpread(silhouetteMatrix, spreadMatrix, silhouetteMatrix);

 if (stop!=null && stop.stop) return;
 data.StoreProduct(this,  silhouetteMatrix);
}
```

```
[Obsolete] public void Generate_NoAA (TileData data, StopToken stop)
{
 SplineSys splineSys = data.ReadInletProduct(this);
 if (splineSys == null || !enabled) return;

 //stroke
 if (stop!=null && stop.stop) return;
 MatrixWorld strokeMatrix = new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.wo
 SplineMatrixOps.Stroke (splineSys, strokeMatrix, white:true, intensity:0.5f, antialiased:false, padOnePixe

 //spreading
 if (stop!=null && stop.stop) return;
 MatrixWorld spreadMatrix = new MatrixWorld(strokeMatrix);
 MatrixOps.SpreadLinear(strokeMatrix, spreadMatrix, subtract: spreadMatrix.PixelSize.x/width / 2);

 //silhouette
 if (stop!=null && stop.stop) return;
 MatrixWorld silhouetteMatrix = strokeMatrix; //just renaming it
 SplineMatrixOps.Silhouette(splineSys, silhouetteMatrix, silhouetteMatrix);

 //combining
 if (stop!=null && stop.stop) return;
 SplineMatrixOps.CombineSilhouetteSpread(silhouetteMatrix, spreadMatrix, silhouetteMatrix);

 if (stop!=null && stop.stop) return;
 data.StoreProduct(this,  silhouetteMatrix);
```

```
 }
}

[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Scatter", iconName=null, disengageable = true,
 colorType = typeof(SplineSys),
 helpLink ="https://gitlab.com/denispahunov/mapmagic/wikis/map_generators/constant")]
public class Scatter2112 : Generator, IInlet<SplineSys>, IOutlet<TransitionsList>
{
 public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

 [Val("Distance")] public float dist = 10;  /// Adds nodes so that the no distances between nodes is larger th
 [Val("Min Objs/Seg")] public int minRes = 4;  /// Adds nodes so that the no distances between nodes is lar
 [Val("Max Objs/Seg")] public int maxRes = 20;  /// Adds nodes so that the no distances between nodes is
 [Val("Rotate")] public bool rotate = true;

 public override void Generate (TileData data, StopToken stop)
 {
  SplineSys src = data.ReadInletProduct(this);
  if (src == null || !enabled) return;

  TransitionsList trns = new TransitionsList();

  float worldHeight = data.globals.height;

  foreach (Line line in src.lines)
```

```
  {
   line.UpdateLength();

   (Vector3[] points, Vector3[] derivatives) = line.GetAllPointsDerivatives(1f/dist, minRes, maxRes);

   for (int p=0; p<points.Length; p++)
   {
    Transition trs = new Transition(points[p].x, points[p].y/worldHeight, points[p].z);

    if (rotate)
     trs.rotation = Quaternion.LookRotation(derivatives[p], Vector3.up);

    trns.Add(trs);
   }
  }

  data.StoreProduct(this, trns);
 }
}

[System.Serializable]
[GeneratorMenu (menu="Spline/Standard", name ="Adjust", iconName=null, disengageable = true,
 colorType = typeof(SplineSys),
 helpLink ="https://gitlab.com/denispahunov/mapmagic/wikis/map_generators/constant")]
public class Adjust2113 : Generator, IMultiInlet, IOutlet<SplineSys>
{
```

```
public override (string, int) GetCodeFileLine () => GetCodeFileLineBase();  //to get here with right-click on

[Val("Input", "Inlet")]  public readonly Inlet<SplineSys> input = new Inlet<SplineSys>();

[Val("Intensity", "Inlet")] public readonly Inlet<MatrixWorld> intensityIn = new Inlet<MatrixWorld>();

public IEnumerable<IInlet<object>> Inlets () { yield return input; yield return intensityIn; }

public bool useRandom = false;

public int seed = 12345;

public enum Relativeness { absolute, relative };

public Relativeness relativeness = Relativeness.relative;

//public enum Randomness { equally, range };

//public Randomness randomness = Randomness.equally;

public Vector2 offsetFront = Vector2.zero;

public Vector2 offsetRight = Vector2.zero;

public Vector2 height = Vector2.zero;  //x is min, y is max

public Vector2 rotation = Vector2.zero;

public Vector2 scale = Vector2.one;



public override void Generate (TileData data, StopToken stop)

{

 SplineSys src = data.ReadInletProduct(input);

 if (src == null) return;
```

```
 if (!enabled) { data.StoreProduct(this, src); return; }

 SplineSys dst = new SplineSys(src);

 MatrixWorld intensityMatrix = data.ReadInletProduct(intensityIn);
 Noise rnd = useRandom ? new Noise(data.random, seed) : null;

 for (int l=0; l<dst.lines.Length; l++)
 {
  Line line = dst.lines[l];
  for (int n=0; n<line.segments.Length; n++)
  {
   Adjust(ref line.segments[n].start.pos, l*1000+n, intensityMatrix, rnd);
  }

  Adjust(ref line.segments[line.segments.Length-1].end.pos, l*10000, intensityMatrix, rnd);
 }

 dst.Update();

 data.StoreProduct(this, dst);
}


public void Adjust (ref Vector3 pos, int hash, MatrixWorld intensityMatrix, Noise rnd)
{
```

```
//generating random vals
float heightRnd, rotRnd, scaleRnd, frontRnd, rightRnd;
if (rnd != null)
{
 heightRnd = rnd.Random(hash, 0);
 heightRnd = height.x + heightRnd*(height.y-height.x);

 rotRnd = rnd.Random(hash, 1);
 rotRnd = rotation.x + rotRnd*(rotation.y-rotation.x);

 scaleRnd = rnd.Random(hash, 2);
 scaleRnd = scale.x + scaleRnd*(scale.y-scale.x);

 frontRnd = rnd.Random(hash, 3); //goes last for compatibility random
 frontRnd = scale.x + frontRnd*(offsetFront.y-offsetFront.x);

 rightRnd = rnd.Random(hash, 3);
 rightRnd = scale.x + frontRnd*(offsetRight.y-offsetRight.x);
}
else
{
 heightRnd = height.x;
 rotRnd = rotation.x;
 scaleRnd = scale.x;
 frontRnd = offsetFront.x;
 rightRnd = offsetRight.x;
```

```
 }

 //calculating intensity
 float intensity = 1;
 if (intensityMatrix != null)
 {
  if (!intensityMatrix.ContainsWorldValue(pos.x, pos.z)) intensity = 0;
  else intensity = intensityMatrix.GetWorldValue(pos.x, pos.z);
 }



 if (relativeness == Relativeness.relative)
 {
  //(Vector2D front, Vector2D right) = trn.FrontRight2D;
  //pos += (Vector3)front * frontRnd * intensity;
  //pos += (Vector3)right * rightRnd * intensity;
  pos.y += heightRnd * intensity; // / height; //not multiplying in output
 }
 else
 {
  //(Vector2D front, Vector2D right) = trn.FrontRight2D;
  //pos += (Vector3)front * frontRnd * intensity;
  //pos += (Vector3)right * rightRnd * intensity;
  pos.y = heightRnd * intensity; // / height;
 }
}
```

```
}
```

```
/*[System.Serializable]
[GeneratorMenu (menu="Spline", name ="Embankment", icon="GeneratorIcons/Constant", disengageable
public class EmbankmentGenerator : StandardGenerator, IOutlet<SplineSys>
{
 [Val("Spline", "Inlet")]  public readonly Inlet<SplineSys> splineIn = new Inlet<SplineSys>();
 [Val("Height", "Inlet")]  public readonly Inlet<MatrixWorld> heightIn = new Inlet<MatrixWorld>();

 [Val("Iterations")]  public int iterations = 8;
 [Val("Max Links")]  public int maxLinks = 4;

 public override IEnumerable<Inlet> Inlets () { yield return input; }

 public override void GenerateProduct (TileData data, StopToken stop)
 {
  PosTab objs = data.products[input);
  if (objs == null) return;

  data.StoreProduct(this,  spline);
 }
}*/
```

```
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using Den.Tools.Splines;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Terrains;



namespace MapMagic.Nodes.SplinesGenerators
{
 [System.Serializable]
 [GeneratorMenu(
  menu = "Spline",
  name = "Output",
  section=2,
  colorType = typeof(SplineSys),
  helpLink = "https://gitlab.com/denispahunov/mapmagic/wikis/output_generators/Height")]
 public sealed class  SplineOutput200 : OutputGenerator, IInlet<SplineSys>  //virtually standard generator (
 {
  public OutputLevel outputLevel = OutputLevel.Main;
```

```
public override OutputLevel OutputLevel { get{ return outputLevel; } }


public override void Generate (TileData data, StopToken stop)
{
 //loading source
 if (stop!=null && stop.stop) return;
 SplineSys src = data.ReadInletProduct(this);
 if (src == null) return;

 //adding to finalize
 if (stop!=null && stop.stop) return;
 if (enabled)
 {
  data.StoreOutput(this, typeof(SplineOutput200), this, src);
  data.MarkFinalize(Finalize, stop);
 }
 else
  data.RemoveFinalize(finalizeAction);
}


public static FinalizeAction finalizeAction = Finalize; //class identified for FinalizeData
public static void Finalize (TileData data, StopToken stop)
{
 //purging if no outputs
 int splinesCount = data.OutputsCount(typeof(SplineOutput200), inSubs:true);
```

```
if (splinesCount == 0)
{
 if (stop!=null && stop.stop) return;
 data.MarkApply(ApplyData.Empty);
 return;
}

//merging splines
SplineSys mergedSpline = null;
if (splinesCount > 1)
{
 mergedSpline = new SplineSys();

 //foreach (SplineSys spline in outputs)
 // mergedSpline.Add...
}
else
{
 foreach ((SplineOutput200 output, SplineSys product, MatrixWorld biomeMask)
  in data.Outputs<SplineOutput200,SplineSys,MatrixWorld>(typeof(SplineOutput200), inSubs:true))
   { mergedSpline = product; break; }
}

//pushing to apply
if (stop!=null && stop.stop) return;
ApplyData applyData = new ApplyData() {spline=mergedSpline};
```

```
 Graph.OnOutputFinalized?.Invoke(typeof(SplineOutput200), data, applyData, stop);
 data.MarkApply(applyData);
}



public class ApplyData : IApplyData
{
 public SplineSys spline;

 public void Apply(Terrain terrain)
 {
  //finding holder
  SplineObject splineObj = terrain.GetComponent<SplineObject>();
  if (splineObj == null) splineObj = terrain.transform.parent.GetComponentInChildren<SplineObject>();

  //or creating it
  if (splineObj == null)
  {
   GameObject go = new GameObject();
   go.transform.parent = terrain.transform.parent;
   go.transform.localPosition = new Vector3();
   go.name = "Spline";
   splineObj = go.AddComponent<SplineObject>();
  }

  splineObj.splineSys = spline;
```

```
 }

 public static ApplyData Empty
  {get{ return new ApplyData() { spline = null }; }}

 public int Resolution {get{ return 0; }}
}


public static void Purge(CoordRect rect, Terrain terrain)
{

}

public override void ClearApplied (TileData data, Terrain terrain)
{
 /*TerrainData terrainData = terrain.terrainData;
 Vector3 terrainSize = terrainData.size;

 terrainData.detailPrototypes = new DetailPrototype[0];
 terrainData.SetDetailResolution(32, 32);*/

 throw new System.NotImplementedException();
 }
}
```

}

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using MapMagic.Products;

namespace MapMagic.Nodes.SplinesGenerators
{
 [GeneratorMenu (menu = "Spline/Portals", name = "Enter", iconName = "GeneratorIcons/PortalIn", lookLik
 public class SplinePortalEnter : PortalEnter<Den.Tools.Splines.SplineSys> { }

 [GeneratorMenu (menu = "Spline/Portals", name ="Exit", iconName = "GeneratorIcons/PortalOut", lookLik
 public class SplinePortalExit : PortalExit<Den.Tools.Splines.SplineSys> { }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using System.Runtime.CompilerServices;

using Den.Tools;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes;
using MapMagic.Terrains;



namespace MapMagic.Locks
{
 [Serializable]
 public class Lock
 {
  public bool locked; //i.e. enabled

  public Vector3 worldPos;
  public float worldRadius = 100;
  public float worldTransition = 20;

  public bool rescaleDraft = true;
```

```
public bool relativeHeight = false;


public string guiName = "Location";
public bool guiExpanded = true;
public float guiHeight; //the height of the position handle


//[NonSerialized] Dictionary<TileData,LockDataSet> lockDatas = new Dictionary<TileData, LockDataSet>


static Dictionary<TileData, Dictionary<Lock,LockDataSet>> lockDatas = new Dictionary<TileData, Diction


#region Events


 #if UNITY_EDITOR
 [UnityEditor.InitializeOnLoadMethod]
 #endif
 [RuntimeInitializeOnLoadMethod]
 static void Subscribe ()
 {
  TerrainTile.OnBeforeTilePrepare += OnTilePrepare_ReadLocks;
  TerrainTile.OnBeforeTileGenerate += OnGenerateStarted_ResizeDrafts;
  Graph.OnOutputFinalized += OnOutputFinalized_WriteLocksInThread;
  //TerrainTile.OnTileApplied += OnTileApplied_WriteLocksInApply; //using apply in thread instead
  TerrainTile.OnAllComplete += OnAllComplete_FlushAllLocks;  //just in case some locks still left


  TerrainTile.OnBeforeResetTerrain += OnTerrainReset_ReadLocks;
  TerrainTile.OnAfterResetTerrain += OnTerrainReset_WriteLocks;
```

```
}



public static void OnTilePrepare_ReadLocks (TerrainTile tile, TileData tileData)
{
 //finding locks intersecting tile
 List<Lock> intersectingLocks = null;
 Lock[] allLocks = tile.mapMagic.locks;
 for (int i=0; i<allLocks.Length; i++)
 {
  if (!allLocks[i].locked) continue;
  if (!allLocks[i].IsIntersecting(tileData.area.active)) continue;
  if (tileData.isDraft && !allLocks[i].rescaleDraft) continue;

  if (intersectingLocks==null)
   intersectingLocks = new List<Lock>();
  intersectingLocks.Add(allLocks[i]);
 }

 if (intersectingLocks == null) //no locks on this tile
  return;

 //preparing lock-to-data dict
 Dictionary<Lock, LockDataSet> lockDatasDict;
 if (!lockDatas.TryGetValue(tileData, out lockDatasDict))
 {
```

```
 lockDatasDict = new Dictionary<Lock,LockDataSet>();
 lockDatas.Add(tileData, lockDatasDict);
}

//writing locks
Terrain terrain = tile.GetTerrain(tileData.isDraft);

int intersectingCount = intersectingLocks.Count;
for (int i=0; i<intersectingCount; i++)
{
 Lock lk = intersectingLocks[i];

 if (lockDatasDict.ContainsKey(lk)) continue;
 //do not read anything if already contains data ?

 LockDataSet lockData = new LockDataSet();
 lockData.Read(terrain, lk);
 lockDatasDict.Add(lk, lockData);
}
}


public static void OnGenerateStarted_ResizeDrafts (TerrainTile tile, TileData draftTileData, StopToken st
{
 if (!draftTileData.isDraft) return;
```

```
 TileData mainTileData = tile.main?.data;
 if (mainTileData == null) return;

 if (!lockDatas.TryGetValue(draftTileData, out Dictionary<Lock, LockDataSet> draftLockDatasDict)) return
 if (!lockDatas.TryGetValue(mainTileData, out Dictionary<Lock, LockDataSet> mainLockDatasDict)) retur

 foreach (var kvp in draftLockDatasDict)
 {
  Lock lk = kvp.Key;
  if (!lk.rescaleDraft) continue;

  LockDataSet draftLockData = kvp.Value;
  LockDataSet mainLockData;

  if (!mainLockDatasDict.TryGetValue(lk, out mainLockData)) continue;

  LockDataSet.Resize(mainLockData, draftLockData);
 }
}

public static void OnOutputFinalized_WriteLocksInThread (Type type, TileData tileData, IApplyData apply
{
 Dictionary<Lock, LockDataSet> lockDatasDict;
 if (!lockDatas.TryGetValue(tileData, out lockDatasDict))
  return;
```

```
 foreach (var kvp in lockDatasDict)
 {
  Lock lk = kvp.Key;
  LockDataSet lockData = kvp.Value;

  if (!lk.locked) continue;

  bool relativeHeight = lk.relativeHeight;
  if (lk.IsIntersecting(tileData.area.active)  &&  !lk.IsContained(tileData.area.active))
   relativeHeight = false;

  lockData.WriteInThread(applyData, relativeHeight);
 }
}


public static void OnTileApplied_WriteLocksInApply (TerrainTile tile, TileData tileData, StopToken stop)
{
 Dictionary<Lock, LockDataSet> lockDatasDict;
 if (!lockDatas.TryGetValue(tileData, out lockDatasDict)) return;

 Terrain terrain = tile.GetTerrain(tileData.isDraft);

 foreach (LockDataSet lockData in lockDatasDict.Values)
  lockData.WriteInApply(terrain, resizeTerrain:false);
```

```
 if (!tileData.isDraft)
  lockDatas.Remove(tileData);
  //leaving lock data for draft since it might be currently generating and nearly applyied (and there is no da
}


public static void OnAllComplete_FlushAllLocks (MapMagicObject mapMagic)
{
 lockDatas.Clear();
}

#endregion

#region Preserve Lock while reset

 //Non-threaded, executed in a single frame

 private static Dictionary<Lock, LockDataSet> mainDatasDict;
 private static Dictionary<Lock, LockDataSet> draftDatasDict;

 public static void OnTerrainReset_ReadLocks (TerrainTile tile)
 {
  //finding locks intersecting tile
  List<Lock> intersectingLocks = null;
  Lock[] allLocks = tile.mapMagic.locks;
```

```
for (int i=0; i<allLocks.Length; i++)
{
 if (!allLocks[i].locked) continue;
 if (!allLocks[i].IsIntersecting(tile.WorldRect)) continue;

 if (intersectingLocks==null)
  intersectingLocks = new List<Lock>();
 intersectingLocks.Add(allLocks[i]);
}

if (intersectingLocks == null) //no locks on this tile
 return;

//writing locks
if (tile.main != null) mainDatasDict = new Dictionary<Lock,LockDataSet>();
if (tile.draft != null) draftDatasDict = new Dictionary<Lock,LockDataSet>();

foreach (Lock lk in intersectingLocks)
{
 if (tile.main != null)
 {
  LockDataSet lockData = new LockDataSet();
  lockData.Read(tile.main.terrain, lk);
  mainDatasDict.Add(lk, lockData);
 }
```

```
  if (tile.draft != null)
  {
   LockDataSet lockData = new LockDataSet();
   lockData.Read(tile.draft.terrain, lk);
   draftDatasDict.Add(lk, lockData);
  }
 }
}

public static void OnTerrainReset_WriteLocks (TerrainTile tile)
{
 if (tile.main != null  &&  mainDatasDict != null)
 {
  foreach (LockDataSet lockData in mainDatasDict.Values)
   lockData.WriteInApply(tile.main.terrain, resizeTerrain:true);
 }

 if (tile.draft != null  &&  draftDatasDict != null)
 {
  foreach (LockDataSet lockData in draftDatasDict.Values)
   lockData.WriteInApply(tile.draft.terrain, resizeTerrain:true);
 }

 mainDatasDict = null;
 draftDatasDict = null;
}
```

```
#endregion

#region Legacy
/*public void ReadLock (TerrainTile tile, TileData tileData)
{
 if (!locked) return;
 if (!IsIntersecting(tileData.area)) return;

 lockDatas.TryGetValue(tileData, out LockDataSet lockData);

 //if (tileData.isDraft && lockData != null) continue; //do not reload draft data if draft is already generating
 //if (lockData != null) return; //do not reload any data if tile is already generating

 if (lockData == null)
 {
  lockData = new LockDataSet();
  lockDatas.Add(tileData, lockData);
 }

 Terrain terrain = tile.GetTerrain(tileData.isDraft);
 lockData.Read(terrain, worldPos, worldRadius, worldTransition);
}
```

```
public void ProcessLock (TerrainTile tile, TileData tileData, StopToken stop)
{
 if (!lockDatas.TryGetValue(tileData, out LockDataSet lockData)) return;
 if (!IsIntersecting(tileData.area) || !locked) return;

 float heightDelta = 0;
 if (relativeHeight && !(tileData.isDraft && rescaleDraft))
  heightDelta = lockData.GetHeightDelta(tileData.apply);

 //re-scaling draft data
 if (tileData.isDraft && rescaleDraft)
 {
  TileData mainTileData = tile.main.data;
  if (lockDatas.TryGetValue(mainTileData, out LockDataSet mainLockData))
  {
   LockDataSet.Resize(mainLockData, lockData);

   if (relativeHeight)
    heightDelta = mainLockData.GetHeightDelta(mainTileData.apply);
  }
 }

 lockData.Process(tileData.apply, heightDelta);
}
```

```
public void WriteLock (TerrainTile tile, TileData tileData, StopToken stop)
{
 if (!lockDatas.TryGetValue(tileData, out LockDataSet lockData)) return;
 if (!IsIntersecting(tileData.area) || !locked) return;

 Terrain terrain = tile.GetTerrain(tileData.isDraft);
 lockData.Write(terrain);

 //if (!stop.stop  &&  !stop.restart) //if non-stopped and draft not restarted
 // lockDatas.Remove(tileData); //flushing after writing
}*/
#endregion

#region Intersections

public bool IsIntersecting (Terrain terrain)
 /// Checks if this terrain contains this lock
 {
  float fullRadius = worldRadius + worldTransition;

  Vector3 terrainPos = terrain.transform.localPosition;
  TerrainData terrainData = terrain.terrainData;
  Vector3 terrainSize = terrainData.size;

  if (terrainPos.x > worldPos.x + fullRadius  ||  terrainPos.x + terrainSize.x < worldPos.x - fullRadius ||
   terrainPos.z > worldPos.z + fullRadius  ||  terrainPos.z + terrainSize.z < worldPos.z - fullRadius )
```

```
   return false;


 return true;
}



public bool IsIntersecting (Area.Dimensions dim)
/// Checks if locks placed within the area or intersects it
{
 float fullRadius = worldRadius + worldTransition;

 if (dim.worldPos.x > worldPos.x + fullRadius  ||  dim.worldPos.x + dim.worldSize.x < worldPos.x - fullRad
  dim.worldPos.z > worldPos.z + fullRadius  ||  dim.worldPos.z + dim.worldSize.z < worldPos.z - fullRadiu
   return false;

 return true;
}

public bool IsIntersecting (Rect rect)
/// Checks if locks placed within the area or intersects it
{
 Vector2 min = rect.min;
 Vector2 max = rect.max;

 float fullRadius = worldRadius + worldTransition;
```

```
 if (worldPos.x+fullRadius < min.x  ||  worldPos.x-fullRadius > max.x ||
  worldPos.z+fullRadius < min.y  ||  worldPos.z-fullRadius > max.y )
   return false;

 return true;
}


public bool IsContained (Area.Dimensions dim)
/// Checks if this lock is placed fully within area.dimensions
{
 float fullRadius = worldRadius + worldTransition;

 if (worldPos.x-fullRadius > dim.worldPos.x  &&  worldPos.x+fullRadius < dim.worldPos.x+dim.worldSize.
  worldPos.z-fullRadius > dim.worldPos.z  &&  worldPos.z+fullRadius < dim.worldPos.z+dim.worldSize.z)
   return true;

 return false;
}


public bool IsContained (Rect rect)
/// Checks if this lock is placed fully within area.dimensions
{
 Vector2 min = rect.min;
 Vector2 max = rect.max;
```

```
 float fullRadius = worldRadius + worldTransition;

 if (worldPos.x-fullRadius > min.x  &&  worldPos.x+fullRadius < max.x  &&
  worldPos.z-fullRadius > min.y  &&  worldPos.z+fullRadius < max.y)
   return true;

 return false;
}


public bool IsContainedInAll (IEnumerable<Rect> rects)
/// Checks if lock is on all the terrains, and not intersecting outer terrain group borders (however it might b
{
 CoordRect lockRect = new CoordRect( new Coord((int)(worldPos.x+0.5f), (int)(worldPos.z+0.5f)), (int)(w
 int numUnits = lockRect.size.x * lockRect.size.z;

 foreach (Rect rect in rects)
 {
  CoordRect areaRect = new CoordRect((int)(rect.position.x+0.5f), (int)(rect.position.y+0.5f), (int)(rect.size
  CoordRect intersection = CoordRect.Intersected(lockRect, areaRect);
  numUnits -= intersection.size.x*intersection.size.z;
 }

 return numUnits <= 0;
}
```

```
  public bool IsContainedInAny (IEnumerable<Rect> rects)
  /// Checks if lock is fully on one of terrains, and not intersecting seam of this or other terrains
  {
   bool containedInAny = false;

   foreach (Rect rect in rects)
   {
    bool contained = IsContained(rect);
    bool intersects = IsIntersecting(rect);

    if (intersects && !contained) return false;

    if (contained) containedInAny = true;
   }

   return containedInAny;
  }

 #endregion
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Terrains;
using MapMagic.Nodes;

namespace MapMagic.Locks
{
 //Could be unified with ITerrainData, but actually there is not a single reason to do so

 public interface ILockData
 {
  void Read (Terrain terrain, Lock lk);
  void WriteInThread (IApplyData applyData);
  void WriteInApply (Terrain terrain, bool resizeTerrain);
  void ApplyHeightDelta (Matrix src, Matrix dst);
  void ResizeFrom (ILockData lockData);

  //void Write (TypeDict apply, out float heightDelta, float relativeHeight); //writes heightdelta for height locks
  //void Write (TypeDict apply, float heightDelta); //reads height delta for others
  //void Process (); //do blending, circle extend, height delta calculation
```

```
}

public class LockDataSet
{
 private ILockData[] lockDatas;

 public LockDataSet ()
 {
  //default lock datas
  List<ILockData> lockDatasList = new List<ILockData> {
   new HeightData(),
   new TexturesData(),
   new GrassData() };
   //new TreesData(),
   //new ObjectsData() };
   //these lock datas are in Objects module

  //other lock datas (objects)
  Type[] allLockTypes = typeof(ILockData).Subtypes();

  for (int i=0; i<allLockTypes.Length; i++)
  {
   if (lockDatasList.FindIndex(ld => ld.GetType() == allLockTypes[i]) < 0)
    lockDatasList.Add(Activator.CreateInstance(allLockTypes[i]) as ILockData);
  }
```

```
 lockDatas = lockDatasList.ToArray();
}

private HeightData HeightData { get{ return (HeightData)lockDatas[0]; } }



public void Read (Terrain terrain, Lock lk)
{
 for (int i=0; i<lockDatas.Length; i++)
  lockDatas[i].Read(terrain, lk);
}

/*public void Process (Dictionary<Type,IApplyData> apply, float heightDelta)
{
 for (int i=0; i<lockDatas.Length; i++)
  lockDatas[i].Process(apply, heightDelta);
}*/

public static void Resize (LockDataSet src, LockDataSet dst)
{
 for (int i=0; i<dst.lockDatas.Length; i++)
  dst.lockDatas[i].ResizeFrom(src.lockDatas[i]);
}

public void WriteInThread (IApplyData applyData, bool relativeHeight)
{
```

```
 if (!relativeHeight)
 {
  for (int i=0; i<lockDatas.Length; i++)
   lockDatas[i].WriteInThread(applyData);
 }

 else
 {
  //if height - applying height and amending other lock reads with height delta
  //NB that height finalize always go first by design
  if (applyData is Nodes.MatrixGenerators.HeightOutput200.IApplyHeightData applyHeightData)
  {
   (Matrix heightSrc, Matrix heightDst) = HeightData.WriteWithHeightDelta(applyHeightData);

   for (int i=1; i<lockDatas.Length; i++)
    lockDatas[i].ApplyHeightDelta(heightSrc, heightDst);
  }

  //or write others
  else
  {
   for (int i=1; i<lockDatas.Length; i++)
    lockDatas[i].WriteInThread(applyData);
  }
 }
}
```

```
 public void WriteInApply (Terrain terrain, bool resizeTerrain)
 {
  for (int i=0; i<lockDatas.Length; i++)
   lockDatas[i].WriteInApply(terrain, resizeTerrain);
 }



 //public float GetHeightDelta (Dictionary<Type,IApplyData> apply)
 // { return HeightData.GetHeightDelta(apply); }
}



public struct CoordCircle
{
 public Coord center;
 public int radius;
 public int transition;
 public int fullRadius;
 public CoordRect rect; //not the same as center+radius since it could be clamped by terrain

 public CoordCircle (Terrain terrain, int resolution, Vector3 worldCenter, float worldRadius, float worldTrans
 {
  Vector3 terrainPos = terrain.transform.parent.localPosition; //taking tile local position, tot terrain one!
  TerrainData terrainData = terrain.terrainData;
  Vector3 terrainSize = terrainData.size;
```

```
    center = Coord.Round(
     (worldCenter.x-terrainPos.x)/terrainSize.x * resolution,
     (worldCenter.z-terrainPos.z)/terrainSize.z * resolution);
    radius = (int)(worldRadius/terrainSize.x * resolution);
    transition = (int)(worldTransition/terrainSize.x * resolution);
    fullRadius = radius+transition;

    rect = new CoordRect(center, radius+transition);
    rect = CoordRect.Intersected(rect, new CoordRect(0,0,resolution,resolution));
   }
  }
 }
```

```
ï»¿
using UnityEngine;
using UnityEditor;
using UnityEngine.Profiling;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;

using MapMagic.Core;
using MapMagic.Terrains;
using MapMagic.Core.GUI;

namespace MapMagic.Locks
{
 public static class LockDraw
 {
  static readonly Color lockColor = new Color(1,0.4f,0,1);
  static readonly Color transitionColor = new Color(0.6f, 0.1f, 0, 1); //alpha is 1
  static readonly Color illegalLockColor = new Color(1,0,0.2f,1);
  static readonly Color illegalTransitionColor = new Color(0.5f,0,0.1f,1);
  const int numCorners = 128;
  const float lineThickness = 5;

  static private PolyLine polyLine = new PolyLine(numCorners);
```

```
static private Vector3[] corners = new Vector3[numCorners];



public static void DrawSceneGUI (MapMagicObject mapMagic)
{
 //drawing locks
 if (Event.current.type == EventType.Repaint)
 {
  UnityEngine.Profiling.Profiler.BeginSample("Drawing Locks");

  int curNumCorners = numCorners;
  if (mapMagic.locks.Length > 3) curNumCorners = numCorners/2;
  if (mapMagic.locks.Length > 6) curNumCorners = numCorners/4;
  if (corners.Length > curNumCorners) corners = new Vector3[curNumCorners];

  //list of all terrains to draw brush
  //Terrain[] terrains = mapMagic.GetComponentsInChildren<Terrain>();
  List<Terrain> terrains = new List<Terrain>(mapMagic.tiles.AllActiveTerrains());
  List<Rect> terrainRects = new List<Rect>(mapMagic.tiles.AllWorldRects());

  //offseting MM-coordsys on mm pos
  //for (int i=0; i<terrainRects.Count; i++)
  // terrainRects[i] = new Rect(terrainRects[i].position + mapMagic.transform.position.V2(), terrainRects[i].s

  for (int l=0; l<mapMagic.locks.Length; l++)
  {
```

```
  Lock lockArea = mapMagic.locks[l];
  //if (!lockArea.guiExpanded) continue;

  Color color = lockColor;
  Color tcolor = transitionColor;
  if (!lockArea.IsContainedInAll(terrainRects))
   { color = illegalLockColor; tcolor = illegalTransitionColor; }

  lockArea.guiHeight = DrawCircle(
   polyLine,
   lockArea.worldPos,
   lockArea.worldRadius,
   color, corners, terrains, terrainRects,
   parent:mapMagic.transform);

  DrawCircle(
   polyLine,
   lockArea.worldPos,
   lockArea.worldRadius+lockArea.worldTransition,
   tcolor, corners, terrains, terrainRects,
   parent:mapMagic.transform);
 }

 UnityEngine.Profiling.Profiler.EndSample();
}
```

```
 //drawing locks handles
 UnityEngine.Profiling.Profiler.BeginSample("Drawing Lock Handles " + Event.current.type);

 for (int l=0; l<mapMagic.locks.Length; l++)
 {
  Lock lockArea = mapMagic.locks[l];

  if (!lockArea.locked)
  {
   UnityEditor.Undo.RecordObject(mapMagic, "MapMagic Lock Move");

   Vector3 lockCenter = lockArea.worldPos + mapMagic.transform.position;
   lockCenter.y = lockArea.guiHeight;
   lockCenter = Handles.PositionHandle(lockCenter, Quaternion.identity);
   lockArea.worldPos = new Vector3(lockCenter.x, 0, lockCenter.z) - mapMagic.transform.position;
  }
  //Handles.ArrowHandleCap(0, lk.worldPos, Quaternion.identity, 100, EventType.Repaint);
 }

 UnityEngine.Profiling.Profiler.EndSample();
}


public static void DrawInspectorGUI (MapMagicObject mapMagic)
{
```

```
 /*using (Cell.Line)
 {
  //Cell.current.margins = new Padding(-2,0);
  foreach (int num in LayersEditor.DrawLayersEnumerable(
   mapMagic.locks.Length,
   onAdd:n => AddLockLayer(mapMagic,n),
   onRemove:n => RemoveLockLayer(mapMagic,n),
   onMove:(n1,n2) => MoveLockLayer(mapMagic,n1,n2)) )
   {
    using (Cell.Line)
     DrawLock(mapMagic.locks[num]);
   }
 }*/

 using (Cell.Line)
 {
  LayersEditor.DrawLayers(
   mapMagic.locks.Length,
   onDraw:n => DrawLock(mapMagic.locks[n]),
   onAdd:n => AddLockLayer(mapMagic,n),
   onRemove:n => RemoveLockLayer(mapMagic,n),
   onMove:(n1,n2) => MoveLockLayer(mapMagic,n1,n2));
 }
}

private static void DrawLock (Lock lk)
```

```
{
 bool locked = lk.locked;
 Texture2D icon = locked ? UI.current.textures.GetTexture("DPUI/Icons/LockLocked") : UI.current.textures

 Cell.EmptyLinePx(4);

 using (Cell.LineStd)
 {
  using (Cell.RowPx(20)) Draw.Icon(icon);

  using (Cell.Row) Draw.EditableLabel(ref lk.guiName);

  using (Cell.RowPx(20))
   lk.guiExpanded = Draw.CheckButton(lk.guiExpanded,
    UI.current.textures.GetTexture("DPUI/Chevrons/Down"),
    UI.current.textures.GetTexture("DPUI/Chevrons/Left"),
    iconScale:0.5f,
    visible:false );
 }



 if (lk.guiExpanded)
 {
  Cell.EmptyLinePx(4);
```

```
using (Cell.Line)
{
 Cell.EmptyRowPx(20);

 using (Cell.Row)
 {
  using (Cell.Line)
  {
   Cell.current.disabled = locked;

   using (Cell.LineStd)
   {
    using (Cell.RowRel(0.3f)) Draw.Label("Position");
    using (Cell.RowRel(0.7f))
    {
     Cell.current.fieldWidth = 0.8f;
     using (Cell.Row) Draw.Field(ref lk.worldPos.x, "X");
     using (Cell.Row) Draw.Field(ref lk.worldPos.z, "Z");
    }
   }

   Cell.current.fieldWidth = 0.7f;
   using (Cell.LineStd) Draw.Field(ref lk.worldRadius, "Radius");
   using (Cell.LineStd) Draw.Field(ref lk.worldTransition, "Transition");

   if (Cell.current.valChanged)
```

```
  SceneView.lastActiveSceneView?.Repaint();
}

bool isContainedInAll = lk.IsContainedInAll(MapMagicInspector.current.mapMagic.tiles.AllWorldRects()
bool isContainedInAny = lk.IsContainedInAny(MapMagicInspector.current.mapMagic.tiles.AllWorldRect

using (Cell.LineStd) Draw.ToggleLeft(ref lk.rescaleDraft, "Sync Draft");
using (Cell.LineStd)
{
 Cell.current.disabled = !isContainedInAny;
 Draw.ToggleLeft(ref lk.relativeHeight, "Relative Height (beta)");
}

if (!isContainedInAny)
 using (Cell.LinePx(42))
  Draw.Helpbox("This lock is partly placed on terrain edge. The Relative Height feature is not available.

if (!isContainedInAll)
 using (Cell.LinePx(42))
  Draw.Helpbox("This lock is partly placed on non-pinned terrain. It could not be welded with the newly



using (Cell.LinePx(22))
{
 Draw.CheckButton(ref locked, "");
 if (Cell.current.valChanged)
```

```
      lk.locked = locked;


     Cell.EmptyRowRel(1);
     using (Cell.RowPx(14)) Draw.Icon(icon);
     using (Cell.RowPx(35))
     {
      Cell.EmptyLineRel(0.5f);
      using (Cell.LinePx(20)) Draw.Label("Lock", style:UI.current.styles.middleLabel);
      Cell.EmptyLineRel(0.5f);
     }
     Cell.EmptyRowRel(1);
    }
   }


   Cell.EmptyRowPx(4);


   //if (Draw.Button("Lock Current"))
   // lk.ReadTerrain();
  }
 }


 Cell.EmptyLinePx(4);
}


private static void AddLockLayer (MapMagicObject mapMagic, int num)
{
```

```
 Lock newLock = new Lock();
 newLock.guiName = "Location " + mapMagic.locks.Length;
 newLock.worldPos = new Vector3(mapMagic.tileSize.x/2, 0, mapMagic.tileSize.z/2); //placing at the cent


 ArrayTools.Insert(ref mapMagic.locks, mapMagic.locks.Length, newLock);
 //mapMagic.ClearAll(); //don't jenerate on adding lock (we might have a custom location prepared)
 //mapMagic.StartGenerate();
 //mapMagic.GenerateAll();
 SceneView.RepaintAll();
}
public static void RemoveLockLayer (MapMagicObject mapMagic, int num)
{
 ArrayTools.RemoveAt(ref mapMagic.locks, num);
 //mapMagic.ClearAll();
 //mapMagic.StartGenerate();
 //mapMagic.GenerateAll();
 SceneView.RepaintAll();
}
public static void MoveLockLayer (MapMagicObject mapMagic, int from, int to)
{
 ArrayTools.Move(mapMagic.locks, from, to);
 //mapMagic.ClearAll();
 //mapMagic.StartGenerate();
 //mapMagic.GenerateAll();
 SceneView.RepaintAll();
}
```

```
public static float DrawCircle (PolyLine line, Vector3 center, float radius, Color color, Vector3[] corners, Lis
/// It will return an average height BTW almost for free :)
{
 int numCorners = corners.Length;
 float step = 360f/(numCorners-1);

 Terrain prevTerrain = null;
 Rect prevRect = new Rect();

 for (int i=0; i<corners.Length; i++)
 {
  //corner initial position
  Vector3 corner = new Vector3( Mathf.Sin(step*i*Mathf.Deg2Rad), 0, Mathf.Cos(step*i*Mathf.Deg2Rad) )

  //checking if the corner lays within the same terrain first
  Terrain terrain = null;
  if (prevRect.Contains( new Vector2(corner.x, corner.z) ))
   terrain = prevTerrain;

  //finding proper terrain in all terrains in it's not in rect
  else
  {
   int rectsCount = terrainRects.Count;
   for (int r=0; r<rectsCount; r++)
```

```
    {
     if (terrainRects[r].Contains( new Vector2(corner.x, corner.z) ))
     {
      terrain = terrains[r];
      prevTerrain = terrains[r];
      prevRect = terrainRects[r];
      break;
     }
    }
   }

   //sampling height
   corners[i] = corner;
   if (terrain != null) corners[i].y = terrain.SampleHeight(corner);
  }

  line.DrawLine(corners, color, lineThickness, zMode:PolyLine.ZMode.Overlay, offset:0, parent:parent);
  //Handles.DrawAAPolyLine(lineThickness, corners);

  //adjusting center height
  float heightSum = 0;
  for (int i=0; i<corners.Length; i++)
   heightSum += corners[i].y;
  return heightSum/(corners.Length-1);
 }
}
```

}

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Terrains;
using MapMagic.Nodes;
using MapMagic.Nodes.MatrixGenerators;

namespace MapMagic.Locks
{
 public class GrassData : ILockData
 {
  public int[][,] lockLayers;
  public DetailPrototype[] lockPrototypes;
  public int patchResolution = 16;

  CoordCircle circle;
  int resolution; //to determine if it's changed and avoid writing


  public void Read (Terrain terrain, Lock lk)
  {
   TerrainData terrainData = terrain.terrainData;
```

```
 if (terrainData.detailPrototypes == null || terrainData.detailPrototypes.Length == 0)
  { lockLayers=null; lockPrototypes=null; return; }
  // Don't perform lock if no grass prototypes

 resolution = terrain.terrainData.detailResolution;
 circle = new CoordCircle(terrain, resolution, lk.worldPos, lk.worldRadius, lk.worldTransition);


 lockPrototypes = terrainData.detailPrototypes;
 lockLayers = new int[lockPrototypes.Length][,];
 for (int i=0; i<lockPrototypes.Length; i++)
 {
  int[,] layer = terrainData.GetDetailLayer(circle.rect.offset.x, circle.rect.offset.z, circle.rect.size.x, circle.rec
  lockLayers[i] = layer;
 }
 patchResolution = terrainData.detailResolutionPerPatch;
}


public void WriteInThread (IApplyData applyData)
{
 if (! (applyData is GrassOutput200.ApplyData applyGrassData) ) return;

 if (lockLayers == null || lockPrototypes == null) return; // Don't perform lock if nothing is stored
 if (Mathf.ClosestPowerOfTwo(applyGrassData.Resolution) != resolution) return; //Don't perform lock if res
```

```
 UnifyPrototypes(ref applyGrassData.detailPrototypes, ref applyGrassData.detailLayers, ref lockPrototype

 for (int c=0; c<lockPrototypes.Length; c++)
  Stamp(applyGrassData.detailLayers[c], lockLayers[c], circle.rect.offset, circle.center, circle.fullRadius);
}


public void WriteInApply (Terrain terrain, bool resizeTerrain=false)
{
 TerrainData terrainData = terrain.terrainData;

 if (lockLayers == null || lockPrototypes == null) return; // Don't perform lock if nothing is stored

 if (terrain.terrainData.detailResolution != resolution)
 {
  if (resizeTerrain)
   terrainData.SetDetailResolution(resolution, patchResolution);
  else
   return;
 }

 terrainData.detailPrototypes = lockPrototypes;
 for (int i=0; i<lockLayers.Length; i++)
  terrainData.SetDetailLayer(circle.rect.offset.x, circle.rect.offset.z, i, lockLayers[i]);

}
```

```
public void ApplyHeightDelta (Matrix src, Matrix dst) { }



public void ResizeFrom (ILockData otherData)
{
 GrassData src = (GrassData)otherData;

 if (src.lockLayers==null || lockLayers==null) return;
 int numLayers = Mathf.Min(src.lockLayers.Length, lockLayers.Length);
 if (numLayers == 0) return;

 for (int i=0; i<numLayers; i++)
 {
  int[,] srcLayer = src.lockLayers[i];
  int[,] dstLayer = lockLayers[i];

  int srcResX = dstLayer.GetLength(0);
  int srcResZ = dstLayer.GetLength(1);

  int dstResX = srcLayer.GetLength(0);
  int dstResZ = srcLayer.GetLength(1);

  float resFactorX = 1f*dstResX/srcResX;
  float resFactorZ = 1f*dstResZ/srcResZ;
```

```
  for (int dstX=0; dstX<dstResX; dstX++)
   for (int dstZ=0; dstZ<dstResZ; dstZ++)
   {
    int srcX = (int)(dstX/resFactorX);
    int srcZ = (int)(dstZ/resFactorZ);

    dstLayer[dstX,dstZ] = srcLayer[srcX,srcZ];
   }
 }
}



private static void UnifyPrototypes (ref DetailPrototype[] basePrototypes, ref int[][,] baseData,
          ref DetailPrototype[] addPrototypes, ref int[][,] addData)
/// Makes both datas prototypes arrays equal, and the layers arrays relevant to prototypes (empty arrays)
/// Safe per-channel blend could be performed after this operation
{
 //guard if prototypes have not been changed
 if (ArrayTools.MatchExactly(basePrototypes, addPrototypes)) return;

 //creating array of unified prototypes
 List<DetailPrototype> unifiedPrototypes = new List<DetailPrototype>();
 unifiedPrototypes.AddRange(basePrototypes); //do not change the base prototypes order
 for (int p=0; p<addPrototypes.Length; p++)
```

```
{
 if (!unifiedPrototypes.Contains(addPrototypes[p]))
  unifiedPrototypes.Add(addPrototypes[p]);
}

//lut to convert prototypes indexes
Dictionary<int,int> baseToUnifiedIndex = new Dictionary<int, int>();
Dictionary<int,int> addToUnifiedIndex = new Dictionary<int, int>();

for (int p=0; p<basePrototypes.Length; p++)
 baseToUnifiedIndex.Add(p, unifiedPrototypes.IndexOf(basePrototypes[p]));  //should be 1,2,3,4,5, but do

for (int p=0; p<addPrototypes.Length; p++)
 addToUnifiedIndex.Add(p, unifiedPrototypes.IndexOf(addPrototypes[p]));

//re-creating base data
{
 int[][,] newBaseData = new int[unifiedPrototypes.Count][,];

 int baseDataLayers = baseData.Length;
 for (int i=0; i<baseDataLayers; i++)
  newBaseData[ baseToUnifiedIndex[i] ] = baseData[i];

 baseData = newBaseData;
}
```

```
 //re-creating add data
 {
  int[][,] newAddData = new int[unifiedPrototypes.Count][,];

  int addDataLayers = addData.Length;
  for (int i=0; i<addDataLayers; i++)
   newAddData[ addToUnifiedIndex[i] ] = addData[i];

  addData = newAddData;
 }
 //saving prototypes
 basePrototypes = unifiedPrototypes.ToArray();
 addPrototypes = unifiedPrototypes.ToArray();
}


private static void Stamp (int[,] arr, int[,] stamp, Coord stampOffset, Coord center, int radius)
/// stamps one splat array onto the other using coords center, radius and transition
/// using channelCompliance to swap channels according their prototypes
{
 int stampResX = stamp.GetLength(1); //x and z are swapped
 int stampResZ = stamp.GetLength(0);

 for (int x=0; x<stampResX; x++)
  for (int z=0; z<stampResZ; z++)
  {
```

```
      int sx = stampOffset.x + x;
      int sz = stampOffset.z + z;

      float dist = Mathf.Sqrt((sx-center.x)*(sx-center.x) + (sz-center.z)*(sz-center.z));
      if (dist > radius) continue;

      arr[sz, sx] = stamp[z, x];

      //SOON: soft blending. Convert each channel to matrix, extend fields, stamp and blend instead.
    }
  }
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Terrains;
using MapMagic.Nodes;
using MapMagic.Nodes.MatrixGenerators;

namespace MapMagic.Locks
{
 public class HeightData : ILockData
 {
  public float[,] heightsArr;

  CoordCircle circle;
  int resolution; //to determine if it's changed and avoid writing. And to rescale terraindata on terrain reset


  public void Read (Terrain terrain, Lock lk)
  {
   TerrainData terrainData = terrain.terrainData;

   resolution = terrainData.heightmapResolution;
```

```
 circle = new CoordCircle(terrain, resolution, lk.worldPos, lk.worldRadius, lk.worldTransition);

 heightsArr = terrainData.GetHeights(circle.rect.offset.x, circle.rect.offset.z, circle.rect.size.x, circle.rect.siz
}

public void WriteInThread (IApplyData applyData)
{
 if (!(applyData is HeightOutput200.IApplyHeightData)) return;

 if (applyData.Resolution != resolution) return; //Don't perform lock if resolution changed

 Matrix terrainMatrix = new Matrix(circle.rect);
 ImportMatrix(terrainMatrix, applyData);

 Matrix lockMatrix = new Matrix(circle.rect);
 lockMatrix.ImportHeights(heightsArr);

 lockMatrix.ExtendCircular(circle.center, circle.radius-1, circle.transition+2, 50);
 terrainMatrix.BlendStamped(terrainMatrix, lockMatrix, circle.center.x, circle.center.z, circle.radius, circle.t
 //terrainMatrix.Fill(lockMatrix);

 ExportMatrix(terrainMatrix, applyData);
}
```

```
public (Matrix heightSrc, Matrix heightDst) WriteWithHeightDelta (HeightOutput200.IApplyHeightData app
{
 Matrix terrainMatrix = new Matrix(circle.rect); //changed matrix with new height
 ImportMatrix(terrainMatrix, applyData);

 Matrix lockMatrix = new Matrix(circle.rect); //un-changed lock matrix
 lockMatrix.ImportHeights(heightsArr);

 Matrix stampMatrix = new Matrix(lockMatrix);
 float heightDelta = GetHeightDelta(lockMatrix, terrainMatrix);
 stampMatrix.Add(heightDelta);

 stampMatrix.ExtendCircular(circle.center, circle.radius-1, circle.transition+2, 50);
 terrainMatrix.BlendStamped(terrainMatrix, stampMatrix, circle.center.x, circle.center.z, circle.radius, circle

 ExportMatrix(terrainMatrix, applyData);

 return (lockMatrix, terrainMatrix);
}


public void WriteInApply (Terrain terrain, bool resizeTerrain=false)
{
 if (heightsArr == null) return; // Don't perform lock if nothing is stored

 TerrainData terrainData = terrain.terrainData;
```

```
 if (terrain.terrainData.heightmapResolution != resolution)
 {
  if (resizeTerrain)
  {
   Vector3 prevSize = terrainData.size;
   terrainData.heightmapResolution = resolution;
   terrainData.size = prevSize;
  }

  else
   return;
 }

 terrainData.SetHeights(circle.rect.offset.x, circle.rect.offset.z, heightsArr);
}

public void ApplyHeightDelta (Matrix src, Matrix dst) { }


public void ResizeFrom (ILockData otherData)
{
 HeightData other = (HeightData)otherData;

 Matrix otherMatrix = new Matrix(other.circle.rect);
 otherMatrix.ImportHeights(other.heightsArr);
```

```
 Matrix matrix = new Matrix(circle.rect);
 MatrixOps.Resize(otherMatrix, matrix);

 matrix.ExportHeights(heightsArr);
}



/*public float GetHeightDelta (Dictionary<Type,IApplyData> applyDatas)
{
 float lockedAvg = GetAvgInCircle(heightsArr, circle.center-circle.rect.offset, circle.radius);
 Vector2 lockMinMax = GetMinMaxInRadius(heightsArr, circle.center-circle.rect.offset, circle.radius);

 Matrix terrainMatrix = GetApplyMatrix(applyDatas);
 if (terrainMatrix == null) return 0;
 float genAvg = GetAvgInCircle(terrainMatrix, circle.center, circle.radius);
 //TODO: read directly without loading matrix

 float heightDelta = genAvg - lockedAvg;
 if (lockMinMax.x + heightDelta < 0) heightDelta = 0 - lockMinMax.x; //lockMin
 if (lockMinMax.y + heightDelta > 1) heightDelta = 1 - lockMinMax.y; //lockMax

 return heightDelta;
}*/
```

```
private float GetHeightDelta (Matrix lockMatrix, Matrix genMatrix)
{
 float lockAvg = GetAvgInCircle(lockMatrix, circle.center, circle.radius);
 Vector2 lockMinMax = GetMinMaxInRadius(lockMatrix, circle.center, circle.radius);

 float genAvg = GetAvgInCircle(genMatrix, circle.center, circle.radius);

 float heightDelta = genAvg - lockAvg;
 if (lockMinMax.x + heightDelta < 0) heightDelta = 0 - lockMinMax.x; //lockMin
 if (lockMinMax.y + heightDelta > 1) heightDelta = 1 - lockMinMax.y; //lockMax

 return heightDelta;
}


private static float GetAvgInCircle (Matrix matrix, Coord center, int radius)
/// Gets average value of the circumference with given radius (not the internal area)
{
 Coord min = center-radius;
 Coord max = center+radius;

 float avgVal = 0;
 int sum = 0;

 int numSamples = (int)(Mathf.PI * radius * 2);
 float radStep = Mathf.PI*2 / numSamples;
```

```
 for (int i=0; i<numSamples; i++)
 {
  float radAngle = radStep * i;
  float dirX = Mathf.Sin(radAngle);
  float dirZ = Mathf.Cos(radAngle);

  int sX = center.x + (int)(dirX*(radius-2)); //to make sure evaluating in locked area
  int sZ = center.z + (int)(dirZ*(radius-2));

  if (sX < min.x || sX >= max.x || sZ < min.z || sZ >= max.z) continue;

  int pos = (sZ-matrix.rect.offset.z)*matrix.rect.size.x + sX - matrix.rect.offset.x;
  float val = matrix.arr[pos];

  avgVal += val;
  sum ++;
 }

 if (sum >= 0) avgVal /= sum;
 return avgVal;
}


private static float GetAvgInCircle (float[,] heightsArr, Coord center, int radius)
/// Gets average value of the circumference with given radius (not the internal area)
{
```

```
Coord min = center-radius;
Coord max = center+radius;

float avgVal = 0;
int sum = 0;

int numSamples = (int)(Mathf.PI * radius * 2);
float radStep = Mathf.PI*2 / numSamples;
for (int i=0; i<numSamples; i++)
{
 float radAngle = radStep * i;
 float dirX = Mathf.Sin(radAngle);
 float dirZ = Mathf.Cos(radAngle);

 int sX = center.x + (int)(dirX*(radius-2)); //to make sure evaluating in locked area
 int sZ = center.z + (int)(dirZ*(radius-2));

 if (sX < min.x || sX >= max.x || sZ < min.z || sZ >= max.z) continue;

 float val = heightsArr[sX,sZ];

 avgVal += val;
 sum ++;
}

if (sum >= 0) avgVal /= sum;
```

```
 return avgVal;
}



private static Vector2 GetMinMaxInRadius (Matrix matrix, Coord center, int radius)
/// Gets minimum and maximum within radius (including internal area)
{
 Coord min = center-radius;
 Coord max = center+radius;

 float minVal = 200000000;
 float maxVal = -200000000;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   float dist = Mathf.Sqrt((x-center.x)*(x-center.x) + (z-center.z)*(z-center.z));
   if (dist > radius-1) continue;

   int pos = (z-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;
   float val = matrix.arr[pos];

   //float val = data.heights2D[z-data.offset.z, x-data.offset.x]; //x and z are swapped

   if (val < minVal) minVal = val;
   if (val > maxVal) maxVal = val;
```

```
  }

 return new Vector2(minVal, maxVal);
}

private static Vector2 GetMinMaxInRadius (float[,] heightsArr, Coord center, int radius)
/// Gets minimum and maximum within radius (including internal area)
{
 Coord min = center-radius;
 Coord max = center+radius;

 float minVal = 200000000;
 float maxVal = -200000000;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   float dist = Mathf.Sqrt((x-center.x)*(x-center.x) + (z-center.z)*(z-center.z));
   if (dist > radius-1) continue;

   float val = heightsArr[x,z];

   //float val = data.heights2D[z-data.offset.z, x-data.offset.x]; //x and z are swapped

   if (val < minVal) minVal = val;
```

```
   if (val > maxVal) maxVal = val;
  }

 return new Vector2(minVal, maxVal);
}



private static void ImportMatrix (Matrix terrainMatrix, IApplyData applyData)
{
 if (applyData is HeightOutput200.ApplySetData heightSetData)
  terrainMatrix.ImportHeights(heightSetData.heights2D, new Coord(0,0));

 else if (applyData is HeightOutput200.ApplySplitData heightSplitData)
  terrainMatrix.ImportHeightStrips(heightSplitData.heights2DSplits, new Coord(0,0));

 #if UNITY_2019_1_OR_NEWER
 else if (applyData is HeightOutput200.ApplyTexData heightTexData)
  terrainMatrix.ImportRawFloat(heightTexData.texBytes, new Coord(0,0), new Coord(heightTexData.res,h
 #endif
}

private static void ExportMatrix (Matrix terrainMatrix, IApplyData applyData)
{
 if (applyData is HeightOutput200.ApplySetData heightSetData)
  terrainMatrix.ExportHeights(heightSetData.heights2D, new Coord(0,0));
```

```
    else if (applyData is HeightOutput200.ApplySplitData heightSplitData)
     terrainMatrix.ExportHeightStrips(heightSplitData.heights2DSplits, new Coord(0,0));

    #if UNITY_2019_1_OR_NEWER
    else if (applyData is HeightOutput200.ApplyTexData heightTexData)
     terrainMatrix.ExportRawFloat(heightTexData.texBytes, new Coord(0,0), new Coord(heightTexData.res,h
    #endif
   }
  }
 }
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Terrains;
using MapMagic.Nodes;
using MapMagic.Nodes.MatrixGenerators;

namespace MapMagic.Locks
{
 public class TexturesData : ILockData
 {
  private float[,,] lockSplats;
  private TerrainLayer[] lockPrototypes;

  CoordCircle circle;
  int resolution; //to determine if it's changed and avoid writing


  public void Read (Terrain terrain, Lock lk)
  {
   TerrainData terrainData = terrain.terrainData;
   if (terrainData.terrainLayers == null || terrainData.terrainLayers.Length == 0)
```

```
  { lockSplats=null; lockPrototypes=null; return; }
  // Don't perform lock

 resolution = terrain.terrainData.alphamapResolution;
 circle = new CoordCircle(terrain, resolution, lk.worldPos, lk.worldRadius, lk.worldTransition);

 lockSplats = terrainData.GetAlphamaps(circle.rect.offset.x, circle.rect.offset.z, circle.rect.size.x, circle.rec
 lockPrototypes = terrainData.terrainLayers;
}



public void WriteInThread (IApplyData applyData)
{
 if (! (applyData is TexturesOutput200.ApplyData applyTexData) ) return;

 if (lockSplats == null || lockPrototypes == null) return; // Don't perform lock if nothing is stored
 if (applyTexData.Resolution != resolution) return; //Don't perform lock if resolution changed

 UnifyPrototypes(ref applyTexData.prototypes, ref applyTexData.splats, ref lockPrototypes, ref lockSplats)

 Matrix lockMatrix = new Matrix(circle.rect);
 Matrix genMatrix = new Matrix(circle.rect);
 for (int c=0; c<lockPrototypes.Length; c++)
 {
  lockMatrix.Fill(0);
  lockMatrix.ImportSplats(lockSplats, circle.rect.offset, c);
```

```
  genMatrix.ImportSplats(applyTexData.splats, new Coord(0,0), c);

  lockMatrix.ExtendCircular(circle.center, circle.radius-1, circle.transition+2, 0);
  lockMatrix.Clamp01();

  genMatrix.BlendStamped(genMatrix, lockMatrix, circle.center.x, circle.center.z, circle.radius, circle.transi

  genMatrix.ExportSplats(applyTexData.splats, new Coord(0,0), c);
 }
}



public void WriteInApply (Terrain terrain, bool resizeTerrain=false)
{
 if (lockSplats == null || lockPrototypes == null) return; // Don't perform lock if nothing is stored

 TerrainData terrainData = terrain.terrainData;

 if (terrain.terrainData.alphamapResolution != resolution)
 {
  if (resizeTerrain)
   terrainData.alphamapResolution = resolution;
  else
   return;
 }
```

```
 terrainData.terrainLayers = lockPrototypes;
 terrainData.SetAlphamaps(circle.rect.offset.x, circle.rect.offset.z, lockSplats);
}



public void ApplyHeightDelta (Matrix src, Matrix dst) { }



public void ResizeFrom (ILockData otherData)
{
 TexturesData src = (TexturesData)otherData;

 if (src.lockSplats==null || lockSplats==null) return;
 int numLayers = Mathf.Min(src.lockSplats.GetLength(2), lockSplats.GetLength(2));
 if (numLayers==0) return;

 Matrix srcMatrix = new Matrix(src.circle.rect);
 Matrix dstMatrix = new Matrix(circle.rect);

 for (int i=0; i<numLayers; i++)
 {
  srcMatrix.ImportSplats(src.lockSplats, i);
  MatrixOps.Resize(srcMatrix, dstMatrix);
  dstMatrix.ExportSplats(lockSplats, i);
 }
}
```

```
private static void UnifyPrototypes (ref TerrainLayer[] basePrototypes, ref float[,,] baseData,
        ref TerrainLayer[] addPrototypes, ref float[,,] addData)
/// Makes both datas prototypes arrays equal, and the layers arrays relevant to prototypes (empty arrays)
/// Safe per-channel blend could be performed after this operation
{
 //guard if prototypes have not been changed
 if (ArrayTools.MatchExactly(basePrototypes, addPrototypes)) return;

 //creating array of unified prototypes
 List<TerrainLayer> unifiedPrototypes = new List<TerrainLayer>();
 unifiedPrototypes.AddRange(basePrototypes); //do not change the base prototypes order
 for (int p=0; p<addPrototypes.Length; p++)
 {
  if (!unifiedPrototypes.Contains(addPrototypes[p]))
   unifiedPrototypes.Add(addPrototypes[p]);
 }

 //lut to convert prototypes indexes
 Dictionary<int,int> baseToUnifiedIndex = new Dictionary<int, int>();
 Dictionary<int,int> addToUnifiedIndex = new Dictionary<int, int>();

 for (int p=0; p<basePrototypes.Length; p++)
  baseToUnifiedIndex.Add(p, unifiedPrototypes.IndexOf(basePrototypes[p]));  //should be 1,2,3,4,5, but do
```

```
for (int p=0; p<addPrototypes.Length; p++)
 addToUnifiedIndex.Add(p, unifiedPrototypes.IndexOf(addPrototypes[p]));

//re-creating base data
{
 float[,,] newBaseData = new float[baseData.GetLength(0), baseData.GetLength(1), unifiedPrototypes.Co

 int baseDataLayers = baseData.GetLength(2);
 for (int i=0; i<baseDataLayers; i++)
  ArrayTools.CopyLayer(baseData, newBaseData, i, baseToUnifiedIndex[i]);

 baseData = newBaseData;
}

//re-creating add data
{
 float[,,] newAddData = new float[addData.GetLength(0), addData.GetLength(1), unifiedPrototypes.Count

 int addDataLayers = addData.GetLength(2);
 for (int i=0; i<addDataLayers; i++)
  ArrayTools.CopyLayer(addData, newAddData, i, addToUnifiedIndex[i]);

 addData = newAddData;
}

//saving prototypes
```

```
 basePrototypes = unifiedPrototypes.ToArray();
 addPrototypes = unifiedPrototypes.ToArray();
}



private static bool ComparePrototypes (TerrainLayer p1, TerrainLayer p2)
{
 return  p1.diffuseTexture==p2.diffuseTexture &&
   p1.normalMapTexture==p2.normalMapTexture &&
   p1.tileSize==p2.tileSize &&
   p1.tileOffset==p2.tileOffset &&
   p1.smoothness==p2.smoothness &&
   p1.metallic==p2.metallic; //for test
}



[Obsolete] private static void Stamp (float[,,] arr, int arrChannel, float[,,] stamp, Coord stampOffset, int sta
/// stamps one splat array onto the other using coords center, radius and transition
/// works much faster than matrices
{
 int stampResX = stamp.GetLength(1); //x and z are swapped
 int stampResZ = stamp.GetLength(0);

 for (int x=0; x<stampResX; x++)
  for (int z=0; z<stampResZ; z++)
  {
```

```
      int sx = stampOffset.x + x;
      int sz = stampOffset.z + z;

      float dist = Mathf.Sqrt((sx-center.x)*(sx-center.x) + (sz-center.z)*(sz-center.z));
      if (dist > radius) continue;

      arr[sz, sx, arrChannel] = stamp[z, x, stampChannel];
    }
  }
 }

}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices; //Normalize gen
//using Den.Tools.Segs;
using Den.Tools.Splines;

using MapMagic.Core;
using MapMagic.Products;

namespace MapMagic.Nodes
{
 public interface IUnit
 /// Either Generator or significant (processed by graph Generate) Layer
 {
  Generator Gen { get; }
  void SetGen (Generator gen);

  ulong Id { get; set; }

  IUnit ShallowCopy();
 }
```

```
public interface IInlet<out T> : IUnit where T:class
 /// The one that linked with the outlet via graph dictionary
/// Could be generator or layer itself or a special inlet obj
{
 ulong LinkedOutletId { get; set; } //is set each on clear or generate from graph luts. Just to quickly get inle
 ulong LinkedGenId { get; set; } //the same, but gets Outlet.Gen (for ready)
}

public interface IOutlet<out T> : IUnit where T:class { }
/// The one that generates and stores product
/// Could be generator or layer itself or a special outlet obj

public interface IMultiInlet
/// Generator that stores multiple inlets (either layered or standard)
{
 IEnumerable<IInlet<object>> Inlets ();
}

public interface IMultiOutlet
/// Generator that stores multiple outlets
{
 IEnumerable<IOutlet<object>> Outlets ();
}

[Serializable]
```

```
public class Inlet<T> : IInlet<T> where T: class
/// The one that is assigned in non-layered multiinlet generators
{
 [SerializeField] private Generator gen;
 public Generator Gen { get{return gen;} private set{gen=value;} } //auto-property is not serialized
 public void SetGen (Generator gen) => Gen=gen;

 public ulong id; //don't generate on creating (will cause error on serialization. And no need to generate it o
 public ulong Id { get{if (id==0) id=Den.Tools.Id.Generate(); return id;} set{id=value;} }
 public ulong LinkedOutletId { get; set; }  //Assigned every before each clear or generate
 public ulong LinkedGenId { get; set; }

 public IUnit ShallowCopy() => (Inlet<T>)this.MemberwiseClone();
}

[Serializable]
public class Outlet<T> : IOutlet<T> where T: class
/// The one that is assigned in non-layered multioutlet generators
{
 [SerializeField] private Generator gen;
 public Generator Gen { get{return gen;} private set{gen=value;} }
 public void SetGen (Generator gen) => Gen=gen;

 public ulong id; //don't generate on creating (will cause error on serialization)
 public ulong Id { get{if (id==0) id=Den.Tools.Id.Generate(); return id;} set{id=value;} }
```

```
 public IUnit ShallowCopy() => (Outlet<T>)this.MemberwiseClone();
}

public interface IPrepare
/// Node has something to make in main thread before generate start in Prepare fn
{
 void Prepare (TileData data, Terrain terrain);
 ulong Id { get; set; }
}



public interface ISceneGizmo
/// Displays some gizmo in a scene view
{
 void DrawGizmo();
 bool hideDefaultToolGizmo {get;set;}
}



[Flags] public enum OutputLevel { Draft=1, Main=2, Both=3 }  //Both is for gui purpose only

public abstract class OutputGenerator : Generator
/// Final output node (height, textures, objects, etc)
{
 //just to mention: static Finalize (TileData data, StopToken stop);
 //Action<TileData,StopToken> FinalizeAction { get; }
```

```
 public abstract void ClearApplied (TileData data, Terrain terrain);
 public abstract OutputLevel OutputLevel {get;}
}

/*public interface IOutput
/// Either output layer or output generator itself
/// TODO: merge with output generator?
{
 Generator Gen { get; }
 void SetGen (Generator gen);
 ulong Id { get; set; }
}
*/

public interface IApplyData
{
 void Apply (Terrain terrain);
 int Resolution {get;}
}


public interface IApplyDataRoutine : IApplyData
{
 IEnumerator ApplyRoutine (Terrain terrain);
}
```

```
public interface ICustomComplexity
/// To implement both Complexity and Progress properties
{
 float Complexity { get; } //default is 1
 float Progress (TileData data);  //can be different from Complexity if the generator is partly done. Max is C
}



public interface IBiome : IUnit
/// Passes the current graph commands to sub graph(s)
{
 Graph SubGraph { get; }
 TileData SubData (TileData parent);  //clusters have non-hierarchial sub-data
 Expose.Override Override { get; set; }
}

public interface ICustomDependence
/// Makes PriorGens generated before generating this one
/// Used in Portals, mainly for checking link validity
{
 IEnumerable<Generator> PriorGens ();
}

public interface IRelevant { }
/// Should be generated when generating graph
```

```
public interface IMultiLayer
/// If generator contains layers that should be processed with graph Generate function
/// In short - if layer contains id generator should be multi-layered
/// Theoretically it should replace IMultiInlet/IMultiOutlet/IMultiBiome
{
//  IEnumerable<IUnit> Layers ();
 IList<IUnit> Layers { get; set; }
 bool Inversed { get; } //for gui purpose, to draw mini
 bool HideFirst { get; }
}

public interface ICustomClear
/// Performs additional actions before clearing (like clearing sub-datas)
{
 void OnClearing (Graph graph, TileData data, ref bool isReady, bool totalRebuild=false);  //calls when clea
 //Not that regular change check if still performed on node
 //isReady: this call happens before marking ready in data, so isReady = default ready check (inlets, portal
 //Setting isReady to true will NOT mark this generator as ready on clear (while setting to false will)
 //totalRebuild: some nodes like Cluster require knowing whether user pressed "Rebuild" to do some flush

 //void ClearDirectly (TileData data);  //Called by graph if gen field was changed. Will call data.ClearReady
 //void ClearRecursive (TileData data);  //Called by graph on clearing recursive (no matter ready or not). In
 //void ClearAny (Generator gen, TileData data); //Called at top level graph each time any node changes. I
}
```

```
public sealed class GeneratorMenuAttribute : Attribute
{
 public string menu;
 public int section;
 public string name;
 public string menuName; //if not defined using real name
 public string iconName;
 public bool disengageable;
 public bool disabled;
 public int priority;
 public string helpLink;
 public bool lookLikePortal = false; //brush reads/writes are portals, but should look like gens
 public bool drawInlets = true;
 public bool drawOutlet = true;
 public bool drawButtons = true;
 public bool advancedOptions = false;
 public Type colorType = null; ///> to display the node in the color of given outlet type
 public Type updateType;  ///> The class legacy generator updates to when clicking Update
 public string codeFile;
 public int codeLine;

 //these are assigned on load attribute in gen and should be null by default
 public string nameUpper;
 public float nameWidth; //the size of the nameUpper in pixels
 public Texture2D icon;
```

```
 public Type type;
 public Color color;
}


[System.Serializable]
public class Layer : IUnit
/// Standard values to avoid duplicating them in each of generators layers
{
 public Generator Gen { get; private set; }
 public void SetGen (Generator gen) => Gen=gen;

 public ulong id; //properties not serialized
 public ulong Id { get{return id;} set{id=value;} }
 public ulong LinkedOutletId { get; set; }  //if it's inlet. Assigned every before each clear or generate
 public ulong LinkedGenId { get; set; }

 public IUnit ShallowCopy() => (Layer)this.MemberwiseClone();
}


[System.Serializable]
public abstract class Generator : IUnit
{
 public bool enabled = true;
```

```
public ulong id; //properties not serialized  //0 is empty id - reassigning it automatically

public ulong Id { get{return id;} set{id=value;} }

public ulong LinkedOutletId { get; set; }  //if it's inlet. Assigned every before each clear or generate

public ulong LinkedGenId { get; set; }


public ulong version; //increment with GUI each time any parameter change to compare with data's last ge


#if MM_DEBUG

public double draftTime;

public double mainTime;

#endif


public Vector2 guiPosition;

public Vector2 guiSize;  //to add this node to group


public bool guiPreview; //is preview for this generator opened

public bool guiAdvanced;

public bool guiDebug;


//just to avoid implementing it in each generator

public Generator Gen { get{ return this; } }

public void SetGen (Generator gen) { }



public static Generator Create (Type type)

///Factory instead of constructor since could not create instance of abstract class
```

```
{
 if (type.IsGenericTypeDefinition) type = type.MakeGenericType(typeof(Den.Tools.Matrices.MatrixWorld))

 Generator gen = (Generator)Activator.CreateInstance(type);
 gen.id = Den.Tools.Id.Generate();

 if (gen is IMultiLayer lgen)
  foreach (IUnit layer in lgen.Layers)
   { layer.SetGen(gen); layer.Id = Den.Tools.Id.Generate(); }

 if (gen is IMultiInlet igen)
  foreach (IUnit layer in igen.Inlets())
   { layer.SetGen(gen); layer.Id = Den.Tools.Id.Generate(); }

 if (gen is IMultiOutlet ogen)
  foreach (IUnit layer in ogen.Outlets())
   { layer.SetGen(gen); layer.Id = Den.Tools.Id.Generate(); }

 return gen;
}

public IUnit ShallowCopy() => (Generator)this.MemberwiseClone();  //1000 noise generators in 0.1 ms

public abstract void Generate (TileData data, StopToken stop);
/// The stuff generator does to read inputs (already prepared), generate, and write product(s). Does not af
```

```
#region Generic Type


 public static Type GetGenericType (Type type)
 /// Finds out if it's map, objects or splines node/inlet/outlet.
 /// Returns T if type is T, IOutlet<T>, Inlet<T>, or inherited from any of those (where T is MatrixWorls, Tra
 {
  Type[] interfaces = type.GetInterfaces();
  foreach (Type itype in interfaces)
  {
   if (!itype.IsGenericType) continue;
   //if (!typeof(IOutlet).IsAssignableFrom(itype) && !typeof(Inlet).IsAssignableFrom(itype)) continue;
   return itype.GenericTypeArguments[0];
  }

  return null;
 }

 //shotcuts to avoid evaluating by type
 public static Type GetGenericType<T> (IOutlet<T> outlet) where T: class  =>  typeof(T);
 public static Type GetGenericType<T> (IInlet<T> inlet) where T: class  =>  typeof(T);
 public static Type GetGenericType (Generator gen)
 {
  if (gen is IOutlet<object> outlet) return GetGenericType(outlet);
  if (gen is IInlet<object> inlet) return GetGenericType(inlet);
  return null;
```

```
 }
 public static Type GetGenericType (IOutlet<object> outlet)
 {
  if (outlet is IOutlet<MatrixWorld>) return typeof(MatrixWorld);
  else if (outlet is IOutlet<TransitionsList>) return typeof(TransitionsList);
  else if (outlet is IOutlet<SplineSys>) return typeof(SplineSys);
  else return GetGenericType(outlet.GetType());
 }
 public static Type GetGenericType (IInlet<object> inlet)
 {
  if (inlet is IInlet<MatrixWorld>) return typeof(MatrixWorld);
  else if (inlet is IInlet<TransitionsList>) return typeof(TransitionsList);
  else if (inlet is IInlet<SplineSys>) return typeof(SplineSys);
  else return GetGenericType(inlet.GetType());
 }

#endregion



#region Serialization/Placeholders

 public Type AlternativeSerializationType
 {get{
  //if (this is IInlet<object>
  return typeof(Placeholders.InletOutletPlaceholder);
 }}
```

```
  #endregion

  public virtual (string, int) GetCodeFileLine () => GetCodeFileLineBase();

  public (string, int) GetCodeFileLineBase (
   [System.Runtime.CompilerServices.CallerFilePath] string sourceFilePath = "",
   [System.Runtime.CompilerServices.CallerLineNumber] int sourceLineNumber = 0)
  {
   return (sourceFilePath, sourceLineNumber);
   //var sf = new System.Diagnostics.StackTrace(1).GetFrame(0);
   //return (sf.GetFileName(), sf.GetFileLineNumber());
  }
 }
}
```

```
ï»¿using System;
using System.Reflection;

using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.Profiling;

using Den.Tools;
using MapMagic.Products;
using MapMagic.Core; //version number
using Den.Tools.Matrices; //get generic type
using MapMagic.Expose;

namespace MapMagic.Nodes
{
 [System.Serializable]
 [HelpURL("https://gitlab.com/denispahunov/mapmagic/wikis/home")]
 [CreateAssetMenu(menuName = "MapMagic/Empty Graph", fileName = "Graph.asset", order = 101)]
 public class Graph : ScriptableObject , ISerializationCallbackReceiver
 {
  [NonSerialized] public Generator[] generators = new Generator[0];  //it's all serialized via callback
  [NonSerialized] public Dictionary<IInlet<object>, IOutlet<object>> links = new Dictionary<IInlet<object>, I
  [NonSerialized] public Group[] groups = new Group[0];
  [NonSerialized] public Noise random = new Noise(12345, 32768); //noise created on serialization as well,
```

```
    [NonSerialized] public Exposed exposed = new Exposed();
    [NonSerialized] public Override defaults = new Override();

    public int serializedVersion = 0; //mapmagic version graph was last serialized to update it
//  public int instanceId; //cached instanceId during serialization

    public static Action<Generator, TileData> OnBeforeNodeCleared;
    public static Action<Generator, TileData> OnAfterNodeGenerated;
    public static Action<Type, TileData, IApplyData, StopToken> OnOutputFinalized; //TODO: rename onAfte

    public bool guiShowDependent;
    public bool guiShowShared;
    public bool guiShowExposed;
    public bool guiShowDebug;
    public Vector2 guiMiniPos = new Vector2(20,20);
    public Vector2 guiMiniAnchor = new Vector2(0,0);

    #if MM_DEBUG
    public bool debugGenerate = true;
    public bool debugGenInfo = false;
    public bool debugGraphBackground = true;
    public float debugGraphBackColor = 0.5f;
    public bool debugGraphFps = true;
    public bool drawInSceneView = false;
    #endif
```

```
//  public void OnEnable () => instanceId=GetInstanceID(); //GetInstanceID not allowed during serialization

 public static Graph Create (Graph src=null, bool inThread=false)
 {
  Graph graph;

  if (inThread)
   graph = (Graph)System.Runtime.Serialization.FormatterServices.GetUninitializedObject(typeof(Graph));
  else
   graph = ScriptableObject.CreateInstance<Graph>();

  //copy source graph
  if (src==null) graph.generators = new Generator[0];
  else
  {
   graph.generators = (Generator[])Serializer.DeepCopy(src.generators);
   graph.random = src.random;
  }

  return graph;
 }


 #region Node Operations

  public void Add (Generator gen)
```

```
{
 if (ArrayTools.Contains(generators,gen))
   throw new Exception("Could not add generator " + gen + " since it is already in graph");

 //gen.id = NewGenId; //id is assigned on create

 ArrayTools.Add(ref generators, gen);
 //cachedGuidLut = null;
}

public void Add (Group grp)
{
 if (ArrayTools.Contains(groups, grp))
   throw new Exception("Could not add group " + grp + " since it is already in graph");

 ArrayTools.Add(ref groups, grp);
 //cachedGuidLut = null;
}

public void Remove (Generator gen)
{
 if (!ArrayTools.Contains(generators,gen))
  throw new Exception("Could not remove generator " + gen + " since it is not in graph");
```

```
 UnlinkGenerator(gen);
 exposed.RemoveUnused(this); //will also remove exposed layers

 ArrayTools.Remove(ref generators, gen);
 //cachedGuidLut = null;
}


public void Remove (Group grp)
{
 if (!ArrayTools.Contains(groups, grp))
  throw new Exception("Could not remove group " + grp + " since it is not in graph");

 ArrayTools.Remove(ref groups, grp);
 //cachedGuidLut = null;
}


public Generator[] Import (Graph other, bool createOverride=false)
/// Returns the array of imported generators (they are copy of the ones in souce graph)
{
 Graph copied = ScriptableObject.CreateInstance<Graph>();
 DeepCopy(other, copied);

 //gens
 ArrayTools.AddRange(ref generators, copied.generators);
```

```
 //links
 foreach (var kvp in copied.links)
  links.Add(kvp.Key, kvp.Value);

 //groups
 ArrayTools.AddRange(ref groups, copied.groups);

 //ids
 Dictionary<ulong,ulong> replacedIds = CheckFixIds( new HashSet<IUnit>(copied.generators) );

 //expose
 copied.exposed.ReplaceIds(replacedIds);
 exposed.AddRange(copied.exposed);

 return copied.generators;
}


public Graph Export (HashSet<Generator> gensHash)
{
 Graph exported = ScriptableObject.CreateInstance<Graph>();

 //gens
 exported.generators = gensHash.ToArray();
```

```
 //links
 foreach (var kvp in links)
 {
  if (gensHash.Contains(kvp.Key.Gen) && gensHash.Contains(kvp.Value.Gen))
   exported.links.Add(kvp.Key, kvp.Value);
 }

 //expose
 foreach (Generator gen in exported.generators)
 {
  Exposed.Entry[] genEntries = exposed[gen.Id];
  if (genEntries != null)
   exported.exposed.AddRange(genEntries);
 }

 //copy to duplicate generators and links
 Graph copied = ScriptableObject.CreateInstance<Graph>();
 DeepCopy(exported, copied);

 return copied;
}



public Generator[] Duplicate (HashSet<Generator> gens)
/// Returns the list of duplicated generators
{
```

```
 Graph exported = Export(gens);
 Generator[] expGens = exported.generators;
 Import(exported);

 return expGens;
}



public static void Reposition (IList<Generator> gens, Vector2 newCenter)
/// Moves array of generators center to new position
{
 Vector2 currCenter = Vector2.zero;
 foreach (Generator gen in gens)
  currCenter += new Vector2(gen.guiPosition.x + gen.guiSize.x/2, gen.guiPosition.y);

 currCenter /= gens.Count;

 Vector2 delta = newCenter - currCenter;

 foreach (Generator gen in gens)
  gen.guiPosition += delta;
}



public Dictionary<ulong,ulong> CheckFixIds (HashSet<IUnit> susGens=null)
// Ensures that ids of generators and layers are never match
```

```
// Will try to change ids in susGens if any (leaving othrs intact)
// Returns approx dict of what ids were replaced with what. Duplicated ids and 0 are not included in list
{
 Dictionary<ulong,IUnit> allIds = new Dictionary<ulong,IUnit>(); //not hash set but dict to skip generator if
 Dictionary<ulong,ulong> oldNewIds = new Dictionary<ulong,ulong>();
 List<ulong> zeroNewIds = new List<ulong>();

 CheckFixIdsRecursively(susGens, allIds, oldNewIds, zeroNewIds);

 if (oldNewIds.Count != 0  )
  Debug.Log($"Changed generators ids on serialize: {oldNewIds.Count + zeroNewIds.Count}");

 return oldNewIds;
}

private void CheckFixIdsRecursively (HashSet<IUnit> susGens, Dictionary<ulong,IUnit> allIds, Dictionary
//populates a dict of renamed ids
//those ids that were renamed from 0 are added to zeroNewIds
{
 if (generators == null)
  OnAfterDeserialize(); //in some cases got to de-serialize sub-graphs

 //not-suspicious generators first
 foreach (IUnit unit in AllUnits())
 {
  if (susGens != null  &&  susGens.Contains(unit))
```

```
  continue;

 if (allIds.TryGetValue(unit.Id, out IUnit contUnit)  || unit.Id == 0)
 {
  if (contUnit == unit) //is itself
   continue;

  ulong newId = Id.Generate();
  while (allIds.ContainsKey(newId))
   newId = Id.Generate();

  if (unit.Id != 0)
   oldNewIds.Add(unit.Id, newId);
  else
   zeroNewIds.Add(newId);

  unit.Id = newId;
 }

 allIds.Add(unit.Id, unit);
}

//sub-graph next
foreach (IBiome biome in UnitsOfType<IBiome>())
{
 if (biome.SubGraph != null)
```

```
  biome.SubGraph.CheckFixIdsRecursively(susGens, allIds, oldNewIds, zeroNewIds);
}

//suspicious generator last (after all other generators are marked and won't change)
if (susGens != null)
 foreach (IUnit unit in AllUnits())
 {
  if (!susGens.Contains(unit))
   continue;

  if (allIds.TryGetValue(unit.Id, out IUnit contUnit) || unit.Id == 0)
  {
   if (contUnit == unit) //is itself
    continue;

   ulong newId = Id.Generate();
   while (allIds.ContainsKey(newId))
    newId = Id.Generate();

   if (unit.Id != 0)
    oldNewIds.Add(unit.Id, newId);
   else
    zeroNewIds.Add(newId);

   unit.Id = newId;
  }
```

```
    allIds.Add(unit.Id, unit);
   }
 }

#endregion

#region Linking

 public void Link (IOutlet<object> outlet, IInlet<object> inlet)
 {
  //unlinking
  if (outlet == null  &&  links.ContainsKey(inlet))
   links.Remove(inlet);

  //linking
  else //if (CheckLinkValidity(outlet, inlet))
  {
   if (links.ContainsKey(inlet)) links[inlet] = outlet;
   else links.Add(inlet, outlet);
  }

  inlet.Gen.version++;
 }
```

```
public void Link (IInlet<object> inlet, IOutlet<object> outlet)
/// The same in case this order is more convenient
 { Link(outlet, inlet); }


public bool CheckLinkValidity (IOutlet<object> outlet, IInlet<object> inlet)
{
 if (Generator.GetGenericType(outlet) != Generator.GetGenericType(inlet))
  return false;

 if (AreDependent(inlet.Gen, outlet.Gen)) //in this order
  return false;

 return true;
}

public bool AreDependent (Generator prevGen, Generator nextGen)
/// Performance: 1000 iterations on TutorialObjects graph take 20-40 ms
{
 if (prevGen == nextGen)
  return true;

 if (nextGen is IInlet<object> nextInlet  &&
  links.TryGetValue(nextInlet, out IOutlet<object> nextInletLink)  &&
  AreDependent(prevGen, nextInletLink.Gen) )
   return true;
```

```
 if (nextGen is IMultiInlet nextMulIn)
 {
  foreach (IInlet<object> nextIn in nextMulIn.Inlets())
  {
   if (links.TryGetValue(nextIn, out IOutlet<object> nextInLink)  &&
    AreDependent(prevGen, nextInLink.Gen) )
     return true;
  }
 }

 if (nextGen is ICustomDependence cusDepGen)
 {
  foreach (Generator priorGen in cusDepGen.PriorGens())
  {
   if (AreDependent(prevGen, priorGen))
    return true;
  }
 }

 return false;
}

public bool AreDependent (Generator prevGen, IInlet<object> nextInlet)
{
 if (links.TryGetValue(nextInlet, out IOutlet<object> parentOut))
```

```
 {
  Generator parentGen = parentOut.Gen;
  return AreDependent(prevGen, parentGen);
 }
 else
  return false;
}



public bool IsLinked (IInlet<object> inlet) => links.ContainsKey(inlet);
/// Is this inlet linked to anything



public IOutlet<object> GetLink (IInlet<object> inlet) => links.TryGetValue(inlet, out IOutlet<object> outlet)
/// Simply gets inlet's link



public void UnlinkInlet (IInlet<object> inlet)
{
 if (links.ContainsKey(inlet))
  links.Remove(inlet);

 inlet.Gen.version++;
}
```

```
public void UnlinkOutlet (IOutlet<object> outlet)
/// Removes any links to this outlet
{
 List<IInlet<object>> linkedInlets = new List<IInlet<object>>();

 foreach (IInlet<object> inlet in linkedInlets)
 {
  links.Remove(inlet);
  inlet.Gen.version++;
 }
}

public void UnlinkGenerator (Generator gen)
/// Removes all links from and to this generator
{
 List<IInlet<object>> genLinks = new List<IInlet<object>>(); //both that connected to this gen inlets and o

 foreach (var kvp in links)
 {
  IInlet<object> inlet = kvp.Key;
  IOutlet<object> outlet = kvp.Value;

  //unlinking this from others (not needed on remove, but Unlink could be called not only on remove)
  if (inlet.Gen == gen)
  {
```

```
   genLinks.Add(inlet);
   outlet.Gen.version++;
  }

  //unlinking others from this
  if (outlet.Gen == gen)
  {
   genLinks.Add(inlet);
   inlet.Gen.version++;
  }
 }

 foreach (IInlet<object> inlet in genLinks)
  links.Remove(inlet);

 gen.version++;
}



private List<IInlet<object>> LinkedInlets (IOutlet<object> outlet)
/// Isn't fast, so using for internal purpose only. For other cases use cachedBackwardsLinks
{
 List<IInlet<object>> linkedInlets = new List<IInlet<object>>();

 foreach (var kvp in links)
  if (kvp.Value == outlet)
```

```
   linkedInlets.Add(kvp.Key);


 return linkedInlets;
}



public void ThroughLink (Generator gen)
/// Connects previous gen outlet with next gen inlet maintaining link before removing this gen
/// This will not unlink generator completely - other inlets may remain
{
 //choosing the proper inlets and outlets for re-link
 IInlet<object> inlet = null;
 IOutlet<object> outlet = null;

 if (gen is IInlet<object>  &&  gen is IOutlet<object>)
  { inlet = (IInlet<object>)gen; outlet = (IOutlet<object>)gen; }

 if (gen is IMultiInlet multInGen  &&  gen is IOutlet<object> outletGen)
 {
  Type genericType = Generator.GetGenericType(gen);
  foreach (IInlet<object> genInlet in multInGen.Inlets())
  {
   if (!IsLinked(genInlet)) continue;
   if (Generator.GetGenericType(genInlet) == genericType) inlet = genInlet; //the first inlet of gen type
  }
 }
```

```
 if (gen is IInlet<object> inletGen  &&  gen is IMultiOutlet multOutGen)
 {
  Type genericType = Generator.GetGenericType(gen);
  foreach (IOutlet<object> genOutlet in multOutGen.Outlets())
  {
   if (Generator.GetGenericType(genOutlet) == genericType) outlet = genOutlet; //the first outlet of gen ty
  }
 }

 if (inlet == null || outlet == null) return;

 // re-linking
 List<IInlet<object>> linkedInlets = LinkedInlets(outlet); //other generator's inlet connected to this gen
 if (linkedInlets.Count == 0)
  return;

 IOutlet<object> linkedOutlet;
 if (!links.TryGetValue(inlet, out linkedOutlet))
  return;

 foreach (IInlet<object> linkedInlet in linkedInlets)
  Link(linkedOutlet, linkedInlet);
}
```

```
 public void AutoLink (Generator gen, IOutlet<object> outlet)
 /// Links with first gen's inlet of the same type as outlet
 {
  Type outletType = Generator.GetGenericType(outlet);

  if (gen is IInlet<object> inletGen)
  {
   if (Generator.GetGenericType(inletGen) == outletType)
    Link(outlet, inletGen);
  }

  else if (gen is IMultiInlet multInGen)
  {
   foreach (IInlet<object> inlet in multInGen.Inlets())
    if (Generator.GetGenericType(inlet) == outletType)
     { Link(outlet,inlet); break; }
  }
 }

#endregion


#region Iterating Nodes

 // Not iteration nodes in subGraphs
 // Using recursive fn calls instead (with Graph in SubGraphs)
```

```
public IEnumerable<Generator> GetGenerators (Predicate<Generator> predicate)
/// Iterates in all generators that match predicate condition
{
 int i = -1;
 for (int g=0; g<generators.Length; g++)
 {
  i = Array.FindIndex(generators, i+1, predicate);
  if (i>=0) yield return generators[i];
  else break;
 }
}


public Generator GetGenerator (Predicate<Generator> predicate)
/// Finds first generator that matches condition
/// Returns null if nothing found (no need to use TryGet)
{
 int i = Array.FindIndex(generators, predicate);
 if (i>=0) return generators[i];
 else return null;
}

public Generator GetGeneratorById (ulong id)
/// Finds first generator with given id
{
```

```
 for (int g=0; g<generators.Length; g++)
 {
  if (generators[g].id == id)
   return generators[g];
 }
 return null;
}

public IEnumerable<T> GeneratorsOfType<T> ()
/// Iterates all generators of given type
{
 for (int g=0; g<generators.Length; g++)
 {
  if (generators[g] is T tGen)
   yield return tGen;
 }
}


public int GeneratorsCount (Predicate<Generator> predicate)
/// Finds the number of generators that match given condition
{
 int count = 0;

 int i = -1;
 for (int g=0; g<generators.Length; g++)
```

```
 {
  i = Array.FindIndex(generators, i+1, predicate);
  if (i>=0) count++;
 }

 return count;
}


public bool ContainsGenerator (Generator gen) => GetGenerator(g => g==gen) != null;

public bool ContainsGeneratorOfType<T> () => GetGenerator(g => g is T) != null;


public int GeneratorsCount<T> () where T: class
{
 bool findByType (Generator g) => g is T;
 return GeneratorsCount(findByType);
}


public IEnumerable<IUnit> AllUnits ()
/// Iterates all generators and layers in graph. May iterate twice if layer is input and output
{
 foreach (Generator gen in generators)
 {
```

```
  yield return gen;

  if (gen is IMultiLayer multGen)
   foreach (IUnit layer in multGen.Layers)
    yield return layer;

  if (gen is IMultiInlet inlGen)
   foreach (IInlet<object> layer in inlGen.Inlets())
    yield return layer;

  if (gen is IMultiOutlet outGen)
   foreach (IOutlet<object> layer in outGen.Outlets())
    yield return layer;
 }
}


public IEnumerable<T> UnitsOfType<T> ()
/// Iterates all generators and layers in graph. May iterate twice if layer is input and output
{
 foreach (Generator gen in generators)
 {
  if (gen is T tgen)
   yield return tgen;

  if (gen is IMultiLayer multGen)
```

```
   foreach (IUnit layer in multGen.Layers)
    if (layer is T tlayer)
     yield return tlayer;

  if (gen is IMultiInlet inlGen)
   foreach (IInlet<object> layer in inlGen.Inlets())
    if (layer is T tlayer)
     yield return tlayer;

  if (gen is IMultiOutlet outGen)
   foreach (IOutlet<object> layer in outGen.Outlets())
    if (layer is T tlayer)
     yield return tlayer;
 }
}



public ulong IdsVersions ()
/// Practically unique identifier that describes this graph and it's current state.
/// Used to update functions and clusters only when graph changed
/// Summary of all generator ids + versions + seed.
{
 ulong ids = 0;
 ulong versions = 0;
 foreach (Generator gen in generators)
 {
```

```
  ids += gen.id;
  versions += gen.version;

  if (gen is IMultiLayer multGen)
   foreach (IUnit layer in multGen.Layers)
    ids += layer.Id;
 }

 return (ulong)(random.Seed<<24) + ids + versions;
}



public IEnumerable<Graph> SubGraphs (bool recursively=false)
/// Enumerates in all child graphs recursively
{
 foreach (IBiome biome in UnitsOfType<IBiome>())
 {
  Graph subGraph = biome.SubGraph;
  if (subGraph == null) continue;

  yield return biome.SubGraph;

  if (recursively)
   foreach (Graph subSubGraph in subGraph.SubGraphs(recursively:true))
    yield return subSubGraph;
 }
```

```
}



public bool ContainsSubGraph (Graph subGraph, bool recursively=false)
{
 foreach (IBiome biome in UnitsOfType<IBiome>())
 {
  Graph biomeSubGraph = biome.SubGraph;
  if (biomeSubGraph == null) continue;
  if (biomeSubGraph == subGraph) return true;
  if (recursively && biomeSubGraph.ContainsSubGraph(subGraph, recursively:true)) return true;
 }

 return false;
}



public IEnumerable<Generator> RelevantGenerators (bool isDraft)
/// All nodes that end chains, have previes, etc - all that should be generated
{
 for (int g=0; g<generators.Length; g++)
  if (IsRelevant(generators[g], isDraft))
   yield return generators[g];
}
```

```
public bool IsRelevant (Generator gen, bool isDraft)
{
 if (gen is OutputGenerator outGen)
 {
  if (isDraft  &&  outGen.OutputLevel.HasFlag(OutputLevel.Draft)) return true;
  if (!isDraft  && outGen.OutputLevel.HasFlag(OutputLevel.Main)) return true;
 }

 else if (gen is IRelevant)
  return true;

 else if (gen is IBiome biomeGen && biomeGen.SubGraph!=null)
  return true;

 else if (gen is IMultiLayer multiLayerGen)
 {
  foreach (IUnit layer in multiLayerGen.Layers)
   if (layer is IBiome)
    return true;
 }

 else if (gen.guiPreview)
  return true;

 return false;
}
```

```
#endregion



#region Generate

 //And all the stuff that takes data into account

 public bool ClearChanged (TileData data, bool totalRebuild=false)
 /// Removes ready state if any of prev gens is not ready
 /// Clears all relevants if prior generator is clear
 {
  RefreshInputHashIds();

  //clearing nodes that are not in graph
  data.ClearStray(this);

  Dictionary<Generator,bool> processed = new Dictionary<Generator,bool>();
  //using processed instead of ready since non-ready generators should be cleared too - they might have l

  //clearing graph nodes
  bool allReady = true;
  foreach (Generator relGen in RelevantGenerators(data.isDraft))
   allReady = allReady & ClearChangedRecursive(relGen, data, processed, totalRebuild);

  return allReady;
```

```
}



private bool ClearChangedRecursive (Generator gen, TileData data, Dictionary<Generator,bool> process
/// Removes ready state if any of prev gens is not ready, per-gen
/// Will iterate at least once all the nodes, even if they were not changed
{
 if (processed.TryGetValue(gen, out bool processedReady))
  return processedReady;

 bool ready = data.IsReady(gen);
 //don't return if not ready, got to check inlet chains and biomes subs

 if (gen is IInlet<object> inletGen)
 {
  if (links.TryGetValue(inletGen, out IOutlet<object> precedingOutlet))
  {
   Generator precedingGen = precedingOutlet.Gen;

   bool inletReady; //loading from lut or clearing recursive
   if (!processed.TryGetValue(gen, out inletReady))
    inletReady = ClearChangedRecursive(precedingGen, data, processed, totalRebuild);

   ready = ready && inletReady;
  }
 }
```

```
if (gen is IMultiInlet multInGen)
 foreach (IInlet<object> inlet in multInGen.Inlets())
 {
  //the same as inletGen
  if (links.TryGetValue(inlet, out IOutlet<object> precedingOutlet))
  {
   Generator precedingGen = precedingOutlet.Gen;

   bool inletReady; //loading from lut or clearing recursive
   if (!processed.TryGetValue(gen, out inletReady))
    inletReady = ClearChangedRecursive(precedingGen, data, processed, totalRebuild);

   ready = ready && inletReady;
   //no break, need to check-clear other layers chains
  }
 }

if (gen is ICustomDependence customDepGen)
 foreach (Generator priorGen in customDepGen.PriorGens())
 {
  ClearChangedRecursive(priorGen, data, processed, totalRebuild);

  if (!data.IsReady(priorGen))
   { ready = false; break; }  //was ready=true. Mistake?
 }
```

```
 if (gen is ICustomClear cgen)
 {
  cgen.OnClearing(this, data, ref ready, totalRebuild);
  //cgen.ClearRecursive(data);
  //ready = data.IsReady(gen);
 }

 if (!ready) data.ClearReady(gen);
 processed.Add(gen, ready);
 return ready;
}



public void Prepare (TileData data, Terrain terrain, Override ovd=null)
/// Executed in main thread to perform terrain reads or something
/// Not recursive just for performance reasons. Prepares only on-ready generators
{
 if (ovd==null)
  ovd = defaults;

 foreach (Generator gen in generators)
 {
  if (!(gen is IPrepare prepGen)) continue;
  if (data.IsReady(gen)) continue;
```

```
  //applying override (duplicating gen if needed)
  Generator cGen = gen;
  if (Assigner.IsExposed(gen, exposed))
   cGen = (Generator)Assigner.CopyAndAssign(gen, exposed, ovd);

  //preparing
  ((IPrepare)cGen).Prepare(data, terrain);
 }
}

public void PrepareRecursive (Generator gen, TileData data, Terrain terrain, Override ovd, HashSet<Ge
{
 if (processed.Contains(gen)) //already prepared
  return;

 if (gen is IInlet<object> inletGen)
 {
  if (links.TryGetValue(inletGen, out IOutlet<object> outlet))
   PrepareRecursive(outlet.Gen, data, terrain, ovd, processed);
 }

 if (gen is IMultiInlet multInGen)
 {
  foreach (IInlet<object> inlet in multInGen.Inlets())
   if (links.TryGetValue(inlet, out IOutlet<object> outlet))
```

```
    PrepareRecursive(outlet.Gen, data, terrain, ovd, processed);
  }

  if (gen is ICustomDependence customDepGen)
  {
   foreach (Generator priorGen in customDepGen.PriorGens())
    PrepareRecursive(priorGen, data, terrain, ovd, processed);
  }

  //not excluding disabled: they should generate the chain before them (including outputs like Textures)
 }



 public void Generate (TileData data, StopToken stop=null, Override ovd=null)
 {
  #if MM_DEBUG
  if (data.area != null) //can be null on clearing
   Log.Add("Generate (draft:" + data.isDraft + ")", $"{data.area.Coord.x}, {data.area.Coord.z}");
  else
   Log.Add("Generate (draft:" + data.isDraft + ")", "area null");
  #endif

  //refreshing link ids lut (only for top level graph)
  RefreshInputHashIds();

  if (ovd==null)
```

```
 ovd = defaults;

#if MM_DEBUG
if (!debugGenerate) return;
#endif

//main generate pass - all changed gens recursively
foreach (Generator relGen in RelevantGenerators(data.isDraft))
{
 if (stop!=null && stop.stop) return;
 GenerateRecursive(relGen, data, ovd,  stop:stop); //will not generate if it has not changed
}
}


public void GenerateRecursive (Generator gen, TileData data, Override ovd, StopToken stop=null)
{
if (stop!=null && stop.stop) return;
if (data.IsReady(gen)) return;
ulong startedVersion = gen.version;

//generating inlets recursively
if (gen is IInlet<object> inletGen)
{
 if (links.TryGetValue(inletGen, out IOutlet<object> outlet))
  GenerateRecursive(outlet.Gen, data, ovd, stop:stop);
```

```
}

if (gen is IMultiInlet multInGen)
{
 foreach (IInlet<object> inlet in multInGen.Inlets())
  if (links.TryGetValue(inlet, out IOutlet<object> outlet))
   GenerateRecursive(outlet.Gen, data, ovd, stop:stop);
}

if (gen is ICustomDependence customDepGen)
{
 foreach (Generator priorGen in customDepGen.PriorGens())
  GenerateRecursive(priorGen, data, ovd:ovd, stop:stop);
}

//checking for generating twice
if (stop!=null && stop.stop) return;
if (data.IsReady(gen))
 throw new Exception($"Generating twice {gen}, id: {Id.ToString(gen.id)}, draft: {data.isDraft}, stop: {(stop

//before-generated event
//if (gen is ICustomGenerate customGen)
// customGen.OnBeforeGenerated(this, data, stop);

//checking if all layers Gen and Id assigned (not necessary, remove)
if (gen is IMultiLayer layerGen)
```

```
 foreach (IUnit layer in layerGen.Layers)
 {
  if (layer.Gen == null) layer.SetGen(gen);
  if (layer.Id == 0) layer.Id = Id.Generate();
 }

//main generate fn
long startTime = System.Diagnostics.Stopwatch.GetTimestamp();
if (stop!=null && stop.stop) return;

//applying override (duplicating gen if needed)
Generator cGen = gen;
if (Assigner.IsExposed(gen, exposed))
 cGen = (Generator)Assigner.CopyAndAssign(gen, exposed, ovd);

//generating
cGen.Generate(data, stop);

//debug data
#if MM_DEBUG
long deltaTime = System.Diagnostics.Stopwatch.GetTimestamp() - startTime;
if (data.isDraft) gen.draftTime = 1000.0 * deltaTime / System.Diagnostics.Stopwatch.Frequency;
else gen.mainTime = 1000.0 * deltaTime / System.Diagnostics.Stopwatch.Frequency;
#endif

//marking ready
```

```
 if (stop!=null && stop.stop) return;
 //if (gen.version==startedVersion || data.isDraft)
 //if it's still relevant (or draft - drafts allowed to be partly wrong) - theoretically it should be so, but MM wo
 data.MarkReady(gen);
 OnAfterNodeGenerated?.Invoke(gen, data);
}



public void Finalize (TileData data, StopToken stop=null)
{
 if (stop!=null && stop.stop) { data.ClearFinalize(); return; } //no need to leave finalize for further generate

 while (data.FinalizeMarksCount > 0)
 {
  FinalizeAction action = data.DequeueFinalize();
  //data returns finalize actions with priorities (height first)

  //if (action == MatrixGenerators.HeightOutput200.finalizeAction)
  //{
  // data.MarkFinalize(ObjectsGenerators.ObjectsOutput.finalizeAction, stop);
  // data.MarkFinalize(ObjectsGenerators.TreesOutput.finalizeAction, stop);
  //}
  //can't use ObjectsGenerators since they are in the other module. Using OnOutputFinalized event instea

  if (stop!=null && stop.stop) { data.ClearFinalize(); return; }
  action(data, stop);
```

```
 }
}



[Obsolete] private IEnumerator Apply (TileData data, Terrain terrain, StopToken stop=null)
/// Actually not a graph function. Here for the template. Not used.
{
 if (stop!=null && stop.stop) yield break;
 while (data.ApplyMarksCount != 0)
 {
  IApplyData apply = data.DequeueApply(); //this will remove apply from the list

  //applying routine
  if (apply is IApplyDataRoutine)
  {
   IEnumerator e = ((IApplyDataRoutine)apply).ApplyRoutine(terrain);
   while (e.MoveNext())
   {
    if (stop!=null && stop.stop) yield break;
    yield return null;
   }
  }

  //applying at once
  else
  {
```

```
   apply.Apply(terrain);
   yield return null;
  }
 }


 #if UNITY_EDITOR
 if (data.isPreview)
  UnityEditor.EditorWindow.GetWindow<UnityEditor.EditorWindow>("MapMagic Graph");
 #endif

 //OnGenerateComplete.Raise(data);
}


public void Purge (Type type, TileData data, Terrain terrain)
/// Purges the results of all output generators of type
{
 for (int g=0; g<generators.Length; g++)
 {
  if (!(generators[g] is OutputGenerator outGen)) continue;

  Type genType = outGen.GetType();
  if (genType==type  ||  type.IsAssignableFrom(genType))
   outGen.ClearApplied(data, terrain);
 }
```

```
 foreach (Graph subGraph in SubGraphs())
  subGraph.Purge(type, data, terrain);
}



private void RefreshInputHashIds ()
/// For each inlet in graph writes the id of linked outlet
{
 //foreach (IUnit unit in AllUnits()) if (unit is IInlet<object> inlet)  inlet.LinkedOutletId = 0;
 //do not set LinkedOutletId beforehand - it might be generating, and some generator will read this id. Cau

 foreach (var kvp in links)
 {
  IInlet<object> inlet = kvp.Key;
  IOutlet<object> outlet = kvp.Value;
  inlet.LinkedOutletId = outlet.Id;
  inlet.LinkedGenId = outlet.Gen.id;
 }

 foreach (IUnit unit in AllUnits())
  if (unit is IInlet<object> inlet  &&  !links.ContainsKey(inlet))
   inlet.LinkedOutletId = 0;
}
```

```
 public static IEnumerable<(Graph,TileData)> AllGraphsDatas (Graph rootGraph, TileData rootData, bool
{
 if (includeSelf)
  yield return (rootGraph, rootData);

 foreach (IBiome biome in rootGraph.UnitsOfType<IBiome>()) //top level first
 {
  Graph subGraph = biome.SubGraph;
  if (subGraph == null)
   continue;

  TileData subData = biome.SubData(rootData);
  if (subData == null)
   continue;

  foreach ((Graph,TileData) subSub in AllGraphsDatas(subGraph, subData))
   yield return subSub;
 }
}

#endregion


#region Complexity/Progress

 public float GetGenerateComplexity ()
```

```
/// Gets the total complexity of the graph (including biomes) to evaluate the generate progress
{
 float complexity = 0;

 for (int g=0; g<generators.Length; g++)
 {
  if (generators[g] is ICustomComplexity)
   complexity += ((ICustomComplexity)generators[g]).Complexity;

  else
   complexity ++;
 }

 return complexity;
}


public float GetGenerateProgress (TileData data)
/// The summary complexity of the nodes Complete (ready) in data (shows only the graph nodes)
/// No need to combine with GetComplexity since these methods are called separately
{
 float complete = 0;

 //generate
 for (int g=0; g<generators.Length; g++)
 {
```

```
  if (generators[g] is ICustomComplexity)
   complete += ((ICustomComplexity)generators[g]).Progress(data);

  else
  {
   if (data.IsReady(generators[g]))
    complete ++;
  }
 }

 return complete;
}



public float GetApplyComplexity ()
/// Gets the total complexity of the graph (including biomes) to evaluate the generate progress
{
 HashSet<Type> allApplyTypes = GetAllOutputTypes();
 return allApplyTypes.Count;
}



private HashSet<Type> GetAllOutputTypes (HashSet<Type> outputTypes=null)
/// Looks in subGraphs recursively
{
 if (outputTypes == null)
```

```
   outputTypes = new HashSet<Type>();

  for (int g=0; g<generators.Length; g++)
   if (generators[g] is OutputGenerator)
   {
    Type type = generators[g].GetType();
    if (!outputTypes.Contains(type))
     outputTypes.Add(type);
   }

  foreach (Graph subGraph in SubGraphs())
   subGraph.GetAllOutputTypes(outputTypes);

  return outputTypes;
 }



 public float GetApplyProgress (TileData data)
 {
  return data.ApplyMarksCount;
 }

#endregion



#region Serialization
```

```
  [SerializeField] private GraphSerializer200Beta serializer200beta = null;
  //[SerializeField] private GraphSerializer199 serializer199 = null;
  //public Serializer.Object[] serializedNodes = new Serializer.Object[0];



  public void OnBeforeSerialize ()
  {
   if (generators == null)
    OnAfterDeserialize(); //trying to load graph once more

   if (generators == null)
    throw new Exception("Could not save graph data, node array is null. Check if graph was loaded succes

   if (serializer200beta == null)
    serializer200beta = new GraphSerializer200Beta();
   serializer200beta.Serialize(this);
  }

  public void OnAfterDeserialize ()
  {
//    try
   {
    if (serializer200beta != null) serializer200beta.Deserialize(this);
```

```
      //if (serializer199 != null) serializer199.Deserialize(this);
      //if (serializedNodes!=null && serializedNodes.Length!=0) generators = (Generator[])Serializer.Deserializ
    }
/*
    catch (Exception e)
    {
     generators=null;
     //Den.Tools.Tasks.CoroutineManager.Enqueue(()=>Debug.LogError("Could not load graph data: " + na
     throw new Exception("Could not load graph data:\n" + e);
    }
*/
   }

   public static void DeepCopy (Graph src, Graph dst)
   {
    dst.name = src.name;
    if (dst.serializer200beta==null) dst.serializer200beta = new GraphSerializer200Beta();
    dst.serializer200beta.Serialize(src); //using dst's serializer (no need to create new one)
    dst.serializer200beta.Deserialize(dst);
   }


  #endregion


  #region Debug
```

```
public static Graph debugGraph; //assign via watch


public string DebugUnitName (ulong id, bool useGraphName=true, string prevGraphNames=null)
/// Gets the unit name, number, graph, layer, etc by it's.
/// Looks in subgraphs too
/// Disable use graph name when running from thread
{
 foreach (IUnit unit in AllUnits())
  if (unit.Id == id)
  {
   string unitName = DebugUnitName(unit);


   if (useGraphName) unitName += $" graph:{name}";
   if (prevGraphNames != null) unitName += $" prev:{prevGraphNames}";


   return unitName;
  }


 int biomeNum = 0;
 foreach (IUnit unit in AllUnits())
  if (unit is IBiome biome)
  {
   string subPrefix;
   if (useGraphName) subPrefix =  $"{prevGraphNames}.{name} (b:{biomeNum})";
```

```
   else subPrefix = $"(b:{biomeNum})";

   string subResult = biome.SubGraph.DebugUnitName(id, useGraphName, subPrefix);
   if (subResult != null)
    return subResult;

   biomeNum++;
  }

 return null;
}


public string[] DebugAllUnits (int subLevel=0)
{
 List<string> strs = new List<string>();

 foreach (IUnit unit in AllUnits())
  strs.Add( DebugUnitName(unit) + " sub:" + subLevel );

 foreach (IUnit unit in AllUnits())
  if (unit is IBiome biome)
   strs.AddRange( biome.SubGraph.DebugAllUnits(subLevel+1) );

 return strs.ToArray();
}
```

```
public string DebugUnitName (IUnit unit)
/// Just returns unit information
{
 string unitName = $"{unit.GetType().Namespace}.{unit.GetType().Name}";
 string genName = $"{unit.Gen.GetType().Namespace}.{unit.Gen.GetType().Name}";

 int genNum = -1;
 for (int g=0; g<generators.Length; g++)
  if (generators[g] == unit.Gen)
   { genNum=g; break; }

 int layerNum = -1; int layerCounter = 0;
 if (unit.Gen != unit  &&  unit.Gen is IMultiLayer multLayerGen)
  foreach (IUnit layer in multLayerGen.Layers)
  {
   if (layer == unit)
    { layerNum = layerCounter; break; }
   layerCounter++;
  }

 int inletNum = -1; int inletCounter = 0;
 if (unit.Gen != unit  &&  unit.Gen is IMultiInlet multInletGen)
  foreach (IUnit inlet in multInletGen.Inlets())
  {
```

```
      if (inlet == unit)
       { inletNum = inletCounter; break; }
      inletCounter++;
     }

    int outletNum = -1; int outletCounter = 0;
    if (unit.Gen != unit  &&  unit.Gen is IMultiOutlet multOutletGen)
     foreach (IUnit outlet in multOutletGen.Outlets())
     {
      if (outlet == unit)
       { outletNum = outletCounter; break; }
      outletCounter++;
     }

    return $"unit:{unitName}, gen:{genName}, id:{Id.ToString(unit.Id)}, g:{genNum}, l:{layerNum}, i:{inletNum
   }

  #endregion
 }
}
```

```
ï»¿using System;
using System.Reflection;

using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.Profiling;

using Den.Tools;
using MapMagic.Products;
using MapMagic.Core; //version number
using MapMagic.Expose;

namespace MapMagic.Nodes
{
 public interface IGraphSerializer
 {
  void Serialize (Graph graph);
  void Deserialize (Graph graph);
 }

 [Serializable]
 public class GraphSerializer200Beta : IGraphSerializer
 {
  public Serializer.Object[] serializedData = new Serializer.Object[0];
```

```
private class SerializedDataProt
{
 public SemVer ver;
 public Generator[] generators;
 //public Dictionary<IInlet<object>, IOutlet<object>> links;
 public IInlet<object>[] linkInlets;
 public IOutlet<object>[] linkOutlets;
 public Group[] groups;
 public int seed = 12345;

 public Exposed exposed = new Exposed();
 public Override defaults = new Override();
}

public void Serialize (Graph graph)
{
 SerializedDataProt prot = new SerializedDataProt() {
  ver = MapMagicObject.version,
  generators = graph.generators,
  linkInlets = graph.links.Keys.ToArray(),
  linkOutlets = graph.links.Values.ToArray(),
  groups = graph.groups,
  seed = graph.random.Seed,
  exposed = graph.exposed,
  defaults = graph.defaults};
```

```
  serializedData = Serializer.Serialize(prot, onAfterSerialize:AfterSerialize);
}

public void Deserialize (Graph graph)
{
 if (serializedData != null)
 {
  object obj = Serializer.Deserialize(serializedData, onBeforeDeserialize:BeforeDeserialize);

  SerializedDataProt prot = (SerializedDataProt)Serializer.Deserialize(serializedData, onBeforeDeserialize
  CheckNullGenerators(prot.generators);
  CheckNullLinks(ref prot.linkInlets, ref prot.linkOutlets);

  graph.generators = prot.generators;
  graph.links = new Dictionary<IInlet<object>, IOutlet<object>>();
  graph.links.AddRange(prot.linkInlets, prot.linkOutlets);
  graph.groups = prot.groups;

  graph.exposed = prot.exposed;
  if (graph.exposed == null) graph.exposed = new Exposed();

  graph.defaults = prot.defaults;
  if (graph.defaults == null) graph.defaults = new Override();

  graph.random = new Noise(prot.seed, 32768);
```

```
  //if (graph.serializedVersion < 210)
    graph.CheckFixIds();

   CheckInletsGensAssigned(graph);
 }
}



public static void CheckInletsGensAssigned (Graph graph)
{
 foreach (Generator gen in graph.generators)
 {
  if (gen is IMultiInlet multiInlet)
   foreach (IInlet<object> inlet in multiInlet.Inlets())
    if (inlet.Gen==null)
    {
     Debug.Log ($"Generator {gen} inlet Gen is not assigned. Fixing.");
     inlet.SetGen(gen);
    }

  if (gen is IMultiOutlet multiOutlet)
   foreach (IOutlet<object> outlet in multiOutlet.Outlets())
    if (outlet.Gen==null)
    {
     Debug.Log ($"Generator {gen} outlet Gen is not assigned. Fixing.");
     outlet.SetGen(gen);
```

```
    }
 }
}


private static void CheckNullGenerators (Generator[] gens)
{
 for (int g=0; g<gens.Length; g++)
 {
  if (gens[g] == null)
   gens[g] = new Placeholders.Placeholder();
 }
}


private static void CheckNullLinks (ref IInlet<object>[] inlets, ref IOutlet<object>[] outlets)
{
 if (!inlets.ContainsNull() && !outlets.ContainsNull())
  return;

 List<IInlet<object>> newInlets = new List<IInlet<object>>();
 List<IOutlet<object>> newOutlets = new List<IOutlet<object>>();

 int count = inlets.Length;
 for (int i=0; i<count; i++)
 {
```

```
  if (inlets[i] != null  &&  outlets[i] != null)
  {
   newInlets.Add(inlets[i]);
   newOutlets.Add(outlets[i]);
  }
 }

 inlets = newInlets.ToArray();
 outlets = newOutlets.ToArray();
}



private static void AfterSerialize (object obj, Serializer.Object serObj)
{
 if (obj is Generator)
  serObj.altType = typeof(Placeholders.InletOutletPlaceholder).AssemblyQualifiedName;

 //if (obj is Placeholders.Placeholder placeholder)
 //{
 // serObj.type = placeholder.origType;
 //}
 //actually done in placeholder itself

 #if !UNITY_2019_2_OR_NEWER
 //inlets/outlets generics
 if (serObj.type.Contains("Den.Tools.Matrices.MatrixWorld, Assembly-CSharp, Version=0.0.0.0, Culture=r
```

```
 serObj.type = serObj.type.Replace("Den.Tools.Matrices.MatrixWorld, Assembly-CSharp, Version=0.0.0.
if (serObj.type.Contains("Den.Tools.TransitionsList, Assembly-CSharp, Version=0.0.0.0, Culture=neutral,
 serObj.type = serObj.type.Replace("Den.Tools.TransitionsList, Assembly-CSharp, Version=0.0.0.0, Cult
if (serObj.type.Contains("Den.Tools.Splines.SplineSys, Assembly-CSharp, Version=0.0.0.0, Culture=neu
 serObj.type = serObj.type.Replace("Den.Tools.Splines.SplineSys, Assembly-CSharp, Version=0.0.0.0, C

/*if (serObj.type == "MapMagic.Nodes.Inlet`1[[Den.Tools.Matrices.MatrixWorld, Assembly-CSharp, Versi
 serObj.type =  "MapMagic.Nodes.Inlet`1[[Den.Tools.Matrices.MatrixWorld, Tools, Version=0.0.0.0]], Map
if (serObj.type == "MapMagic.Nodes.Inlet`1[[Den.Tools.TransitionsList, Assembly-CSharp, Version=0.0.0
 serObj.type =  "MapMagic.Nodes.Inlet`1[[Den.Tools.TransitionsList, Tools, Version=0.0.0.0]], MapMagic
if (serObj.type == "MapMagic.Nodes.Inlet`1[[Den.Tools.Splines.SplineSys, Assembly-CSharp, Version=0
 serObj.type =  "MapMagic.Nodes.Inlet`1[[Den.Tools.Splines.SplineSys, Tools, Version=0.0.0.0]], MapMa
if (serObj.type == "MapMagic.Nodes.Outlet`1[[Den.Tools.Matrices.MatrixWorld, Assembly-CSharp, Versi
 serObj.type =  "MapMagic.Nodes.Outlet`1[[Den.Tools.Matrices.MatrixWorld, Tools, Version=0.0.0.0]], Ma
if (serObj.type == "MapMagic.Nodes.Outlet`1[[Den.Tools.TransitionsList, Assembly-CSharp, Version=0.0
 serObj.type =  "MapMagic.Nodes.Outlet`1[[Den.Tools.TransitionsList, Tools, Version=0.0.0.0]], MapMag
if (serObj.type == "MapMagic.Nodes.Outlet`1[[Den.Tools.Splines.SplineSys, Assembly-CSharp, Version=
 serObj.type =  "MapMagic.Nodes.Outlet`1[[Den.Tools.Splines.SplineSys, Tools, Version=0.0.0.0]], MapM
if (serObj.type == "MapMagic.Nodes.Inlet`1[[Den.Tools.TransitionsList, Assembly-CSharp, Version=0.0.0
 serObj.type =  "MapMagic.Nodes.Inlet`1[[Den.Tools.TransitionsList, Tools, Version=0.0.0.0, Culture=neu

serObj.type = ModifyTypeName(serObj.type);
if (serObj.altType != null && serObj.altType.Length != 0)
 serObj.altType = ModifyTypeName(serObj.altType);

if (serObj.special != null  &&  serObj.special.Contains(", Assembly-CSharp, Version=0.0.0.0, Culture=neu
```

```
    serObj.special = serObj.special.Replace(", Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicK

   #endif
  }

  private static string ModifyTypeName (string typeName)
  {
   if (typeName.StartsWith("MapMagic."))
    typeName = typeName.Replace(", Assembly-CSharp, ", ", MapMagic, ");
   if (typeName.StartsWith("Den.Tools."))
    typeName = typeName.Replace(", Assembly-CSharp, ", ", Tools, ");
   return typeName;
  }



  private static void BeforeDeserialize (Serializer.Object serObj)
  {
   if (serObj.type == "MapMagic.Nodes.Exposed, MapMagic, Version=0.0.0.0, Culture=neutral, PublicKeyT
    serObj.type = null; //resetting obj to null

//if (serObj.type!=null && serObj.type.Contains("MapMagic.Expose.Override")) //temporary
// serObj.type = null;

   if (serObj.type == null)  //for some reason type null after trying to reset expose
    return;
```

```
foreach (var kvp in classRenames)
{
 if (serObj.type.Contains(kvp.Key))
  serObj.type = serObj.type.Replace(kvp.Key, kvp.Value);
}

#if !UNITY_2019_2_OR_NEWER
if (serObj.type.Contains(", MapMagic, "))
 serObj.type = serObj.type.Replace(", MapMagic, ", ", Assembly-CSharp, ");
if (serObj.type.Contains(", Tools, "))
 serObj.type = serObj.type.Replace(", Tools, ", ", Assembly-CSharp, ");
if (serObj.altType != null)
{
 if (serObj.altType.Contains(", MapMagic, "))
  serObj.altType = serObj.altType.Replace(", MapMagic, ", ", Assembly-CSharp, ");
 if (serObj.altType.Contains(", Tools, "))
  serObj.altType = serObj.altType.Replace(", Tools, ", ", Assembly-CSharp, ");
}

if (serObj.special != null)
{
 //"contrast, MapMagic.Nodes.MatrixGenerators.Contrast200, MapMagic, Version=0.0.0.0, Culture=neutr
 if (serObj.special.Contains(", MapMagic, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null"))
  serObj.special = serObj.special.Replace(", MapMagic, Version=0.0.0.0, Culture=neutral, PublicKeyToke
}
```

```
    #endif
  }

  private static readonly Dictionary<string,string> classRenames = new Dictionary<string, string>() {
    {"Plugins.", "Den.Tools." },
    {"MapMagic.Nodes.ObjectsGenerators.PositioningSettings", "MapMagic.Nodes.PositioningSettings" },
    {", Tools, Version", ", Den.Tools, Version"}, //Tools assembly rename
    {"MapMagic.Nodes.ObjectsGenerators.BiomeBlend", "MapMagic.Nodes.BiomeBlend" },
    {"MapMagic.Nodes.MatrixGenerators.MicroSplatOutput200, Assembly-CSharp,", "MapMagic.Nodes.Matr
    {"MapMagic.Nodes.MatrixGenerators.MicroSplatOutput200, Assembly-CSharp-firstpass,", "MapMagic.No
    };
  }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public static class GraphUpdater
{

    /*#region Updating older versions
     if (serializedVersion < 10) Debug.LogError("MapMagic: trying to load unknow version scene (v." + serial
      "This may cause errors or drastic drop in performance. " +
      "Delete this MapMagic object and create the new one from scratch when possible.");

     //loading old serializer
     if (classes.Length==0 && serializer.entities.Count!=0)
     {
      serializer.ClearLinks();
      all = (Generator[])serializer.Retrieve(listNum);
      serializer.ClearLinks();

      OnBeforeSerialize();
      serializer = null; //will not make it null, just 0-length
      OnAfterDeserialize ();
     }

     if (serializedVersion < 200)
     {
```

```
//      for (int i=0; i<classes.Length; i++)
//       classes[i] = Update150to200(classes[i]);


      string tmp = "";
      for (int i=0; i<classes.Length; i++)
       tmp += classes[i] + "\n";
      Debug.Log(tmp);
     }
    #endregion*/




    /*List<string> classesList = new List<string>();  classesList.AddRange(classes);
    List<UnityEngine.Object> objectsList = new List<UnityEngine.Object>();  objectsList.AddRange(objects)
    List<float> floatsList = new List<float>();  floatsList.AddRange(floats);

    references.Clear();
    Generator[] newAll = null;
    //try {
    newAll = (Generator[])CustomSerialization.ReadClass(0, classesList, objectsList, floatsList, references);
    //catch (Exception e) { Debug.LogError ("Error loading graph: " + e); }
    if (newAll != null) all = newAll;
    references.Clear();

    //post-processing generators
```

```
 for (int i=0; i<all.Length; i++)
 {
  Generator gen = all[i];

  //adding special arrays
  if (gen is OutputGenerator) ArrayTools.Add(ref outputs, gen as OutputGenerator);
  if (gen is Group) ArrayTools.Add(ref groups, gen as Group);
  if (gen is Biome) ArrayTools.Add(ref biomes, gen as Biome);
 }
}



/*public string Update150to200 (string src)
{
 string matrixClassName = "Plugins.MatrixWorld";
 string objectClassName = "Plugins.PosTab";
 //TODO: should be MapMagic

 int headStop = src.IndexOf(">") + 1;
 string header = src.Remove(headStop,src.Length-headStop);
 Debug.Log(header + ": Updating v1.5 graph data to v2.0. Check graph consistency after update.");

 //inlet classes
 if (src.Contains("<MapMagic.Generator+Input>"))
 {
  if (src.Contains("<type type=MapMagic.Generator+InoutType value=0/>")) //type == Type.Map
```

```
 {
  src = src.Replace("<MapMagic.Generator+Input>", "<MapMagic.Generator+Inlet`1[" + matrixClassNam
  src = src.Replace("</MapMagic.Generator+Input>", "</MapMagic.Generator+Inlet`1[" + matrixClassNa
 }

 if (src.Contains("<type type=MapMagic.Generator+InoutType value=1/>")) //type == Type.Objects
 {
  src = src.Replace("<MapMagic.Generator+Input>", "<MapMagic.Generator+Inlet`1[" + objectClassNam
  src = src.Replace("</MapMagic.Generator+Input>", "</MapMagic.Generator+Inlet`1[" + objectClassNa
 }
}

//outlet classes
if (src.Contains("<MapMagic.Generator+Output>"))
{
 if (src.Contains("<type type=MapMagic.Generator+InoutType value=0/>")) //type == Type.Map
 {
  src = src.Replace("<MapMagic.Generator+Output>", "<MapMagic.Generator+Outlet`1[" + matrixClassN
  src = src.Replace("</MapMagic.Generator+Output>", "</MapMagic.Generator+Outlet`1[" + matrixClass
 }

 if (src.Contains("<type type=MapMagic.Generator+InoutType value=1/>")) //type == Type.Objects
 {
  src = src.Replace("<MapMagic.Generator+Output>", "<MapMagic.Generator+Outlet`1[" + objectClassN
  src = src.Replace("</MapMagic.Generator+Output>", "</MapMagic.Generator+Outlet`1[" + objectClass
 }
```

```
}

//special case for combine generators array
if (src.StartsWith("<MapMagic.CombineGenerator>") && src.Contains("<<inputs type=MapMagic.Genera
 src = src.Replace("<inputs type=MapMagic.Generator+Input[]", "<inputs type=MapMagic.Generator+Inl
if (src.StartsWith("<MapMagic.Generator+Input[]"))
{
 while (src.Contains("type=MapMagic.Generator+Input"))
  src = src.Replace("type=MapMagic.Generator+Input", "type=MapMagic.Generator+Inlet`1[" + objectCla
 src = src.Replace("MapMagic.Generator+Input[]", "MapMagic.Generator+Inlet`1[" + objectClassName +
}

//object inlet/outlet values
string[] allPossibleObjectInlets = new string[] { //first generator name, then value name
 "ObjectOutput+Layer", "input",
 "TreesOutput+Layer", "input",
 "AdjustGenerator",  "input",
 "CleanUpGenerator",  "input",
 "SplitGenerator",  "input",
 "SubtractGenerator", "minuendIn",
 "SubtractGenerator", "subtrahendIn",
 "RarefyGenerator",  "input",
 "CombineGenerator",  "inputs",
 "PropagateGenerator", "input",
 "StampGenerator",  "positionsIn",
 "BlobGenerator",  "objectsIn",
```

```
 "FlattenGenerator",  "objectsIn",
 "ForestGenerator",  "seedlingsIn",
 "ForestGenerator",  "otherTreesIn",
 "SlideGenerator",  "input" };

string[] allPossibleObjectOutlets = new string[] { //first generator name, then value name
 "ScatterGenerator",  "output",
 "AdjustGenerator",  "output",
 "CleanUpGenerator",  "output",
 "SplitGenerator",  "output",
 "SubtractGenerator", "minuendOut",
 "RarefyGenerator",  "output",
 "CombineGenerator",  "output",
 "PropagateGenerator", "output",
 "ForestGenerator",  "treesOut",
 "SlideGenerator",  "output" };

for (int i=0; i<allPossibleObjectInlets.Length; i++)
{
 while (src.StartsWith("<MapMagic." + allPossibleObjectInlets[i]) && src.Contains("<" + allPossibleObject
  src = src.Replace("<" + allPossibleObjectInlets[i+1] + " type=MapMagic.Generator+Input", "<" + allPoss
}

for (int i=0; i<allPossibleObjectOutlets.Length; i++)
{
 while (src.StartsWith("<MapMagic." + allPossibleObjectOutlets[i]) && src.Contains("<" + allPossibleObje
```

```
  src = src.Replace("<" + allPossibleObjectOutlets[i+1] + " type=MapMagic.Generator+Output", "<" + allP
}

//portals
if (src.StartsWith("<MapMagic.Portal>"))
{
 //src.Replace("<input type=MapMagic.Generator+Input", "<input type=MapMagic.Generator+Inlet");
 //src.Replace("<output type=MapMagic.Generator+Output", "<output type=MapMagic.Generator+Outlet



 if (src.Contains("<type type=MapMagic.Generator+InoutType value=0/>")) //map type
 {
  if (src.Contains("<form type=MapMagic.Portal+PortalForm value=0/>"))
  {
   src = src.Replace ("<MapMagic.Portal>", "<MapMagic.InletPortal`1[" + matrixClassName + "]>");
   src = src.Replace ("</MapMagic.Portal>", "</MapMagic.InletPortal`1[" + matrixClassName + "]>");
  }
  if (src.Contains("<form type=MapMagic.Portal+PortalForm value=1/>"))
  {
   src = src.Replace ("<MapMagic.Portal>", "<MapMagic.OutletPortal`1[" + matrixClassName + "]>");
   src = src.Replace ("</MapMagic.Portal>", "</MapMagic.OutletPortal`1[" + matrixClassName + "]>");
  }
  src = src.Replace ("<input type=MapMagic.Generator+Input", "<input type=MapMagic.Generator+Inlet`
  src = src.Replace ("<output type=MapMagic.Generator+Output", "<output type=MapMagic.Generator+O
 }
 if (src.Contains("<type type=MapMagic.Generator+InoutType value=1/>")) //object type
```

```
 {
  if (src.Contains("<form type=MapMagic.Portal+PortalForm value=0/>"))
  {
   src = src.Replace ("<MapMagic.Portal>", "<MapMagic.InletPortal`1[" + objectClassName + "]>");
   src = src.Replace ("</MapMagic.Portal>", "</MapMagic.InletPortal`1[" + objectClassName + "]>");
  }
  if (src.Contains("<form type=MapMagic.Portal+PortalForm value=1/>"))
  {
   src = src.Replace ("<MapMagic.Portal>", "<MapMagic.OutletPortal`1[" + objectClassName + "]>");
   src = src.Replace ("</MapMagic.Portal>", "</MapMagic.OutletPortal`1[" + objectClassName + "]>");
  }
  src = src.Replace ("<input type=MapMagic.Generator+Input", "<input type=MapMagic.Generator+Inlet`
  src = src.Replace ("<output type=MapMagic.Generator+Output", "<output type=MapMagic.Generator+C
 }
}

//other inlet/outlet values
while (src.Contains("type=MapMagic.Generator+Input"))
{
 src = src.Replace("type=MapMagic.Generator+Input", "type=MapMagic.Generator+Inlet`1[" + matrixCla
}

while (src.Contains("type=MapMagic.Generator+Output"))
{
 src = src.Replace("type=MapMagic.Generator+Output", "type=MapMagic.Generator+Outlet`1[" + matrix
}
```

```
    //replacing matrix link in objects input
    if (src.StartsWith("<MapMagic.Generator+Inlet`1[" + objectClassName + "]>"))
     src.Replace("<link type=MapMagic.Generator+Outlet`1[" + matrixClassName + "]", "<link type=MapMag

    return src;
   }*/
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;
using MapMagic.Products;

namespace MapMagic.Nodes
{

 [System.Serializable]
 public class Group
 {
  public string name = "Group";
  public string comment = "Drag in generators to group them";
  public Color color = new Color(0.625f, 0.625f, 0.625f, 1);

  public Vector2 guiPos;
  public Vector2 guiSize = new Vector2(100,100);
 }
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices; //Normalize gen
//using Den.Tools.Segs;

using MapMagic.Core;
using MapMagic.Products;

namespace MapMagic.Nodes
{
 /*[System.Serializable]
 public abstract class ILayersGenerator : Generator, IMultiNode
 /// The node with the layers. Each layer has 0 or more inputs and one output.
 {
  public abstract class Layer : INode, IOutlet<object>
  {
   public Generator parent;
   public Generator Gen { get{ return parent;} set{parent=value;} }
  }
```

```
 public Layer[] layers = new Layer[0];

 public abstract Layer CreateLayer ();

 public virtual void AddLayer (int num) { ArrayTools.Insert(ref layers, layers.Length, CreateLayer()); }
 public virtual void RemoveLayer (int num) { ArrayTools.RemoveAt(ref layers, num); }
 public virtual void MoveLayer (int from, int to) { ArrayTools.Move(layers, from, to); }

 public override IEnumerable<IInlet<object>> Inlets()
 {
  for (int i=0; i<layers.Length; i++)
   foreach (IInlet<object> inlet in layers[i].Inlets())
    yield return inlet;
 }

 public IEnumerable<INode> InternalNodes()
 {
  for (int i=0; i<layers.Length; i++)
   yield return layers[i];
 }
}*/


/*public abstract class NormalizeGenerator : Generator, IMultiInlet, IMultiOutlet
/// A generator with normalized layers. Each layer has 1 input and 1 output, matrix only.
/// Here only because it's used quite often
```

```
{
 public interface INormalizableLayer : IInlet<MatrixWorld>, IOutlet<MatrixWorld>
 {
  float Opacity { get; }
 }

 public class NormalizeLayer : INormalizableLayer, IInlet<MatrixWorld>, IOutlet<MatrixWorld>
 {
  public Generator Gen {get; set; }
  public float Opacity { get; set; }
 }

 public NormalizeLayer[] layers = new NormalizeLayer[0];

 //inversed oder
 public void AddLayer (int num)  { ArrayTools.Insert(ref layers, layers.Length, new NormalizeLayer() {Gen=
 public void RemoveLayer (int num)  { ArrayTools.RemoveAt(ref layers, layers.Length-1 - num); }
 public void MoveLayer (int from, int to)  { ArrayTools.Move(layers, layers.Length-1 - from, layers.Length-1

 public IEnumerable<IInlet<object>> Inlets()
 {
  for (int i=0; i<layers.Length; i++)
   yield return layers[i];
 }

 public IEnumerable<IOutlet<object>> Outlets()
```

```
{
 for (int i=0; i<layers.Length; i++)
  yield return layers[i];
}

public override void Generate (TileData data, StopToken stop)
{
 if (layers.Length == 0) return;
 NormalizeLayers(layers, data, stop);
}

public static void NormalizeLayers (INormalizableLayer[] layers, TileData data, StopToken stop)
{
 //reading products
 MatrixWorld[] matrices = new MatrixWorld[layers.Length];
 float[] opacities = new float[layers.Length];

 if (stop!=null && stop.stop) return;
 for (int i=0; i<layers.Length; i++)
 {
  if (stop!=null && stop.stop) return;
  NormalizeLayer layer = (NormalizeLayer)layers[i];

  MatrixWorld srcMatrix = data.ReadInletProduct(layer);
  if (srcMatrix != null) matrices[i] = new MatrixWorld(srcMatrix);
```

```
    opacities[i] = layer.Opacity;
   }

   //normalizing
   if (stop!=null && stop.stop) return;
   matrices.FillNulls(() => new MatrixWorld(data.area.full.rect, data.area.full.worldPos, data.area.full.worldS
   matrices[0].Fill(1);
   Matrix.BlendLayers(matrices, opacities);

   //saving products
   if (stop!=null && stop.stop) return;
   for (int i=0; i<layers.Length; i++)
    data.products[layers[i]] = matrices[i];
  }
 }*/

}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;

using MapMagic.Core;
using MapMagic.Products;

namespace MapMagic.Nodes
{
 public static class Placeholders
 {
  [Serializable]
  public class SerObject
  /// nearly copy of Serializer.Object, but has some fields missed intentionally to allow serialization
  {
   public string type = null;
   public string[] fields = null;
   public Serializer.Value[] values = null;

   public static explicit operator Serializer.Object (SerObject src)
   {
    Serializer.Object dst = new Serializer.Object();
```

```
  dst.type = src.type;
  dst.fields = src.fields;
  dst.values = src.values;
  return dst;
 }

 public static explicit operator SerObject (Serializer.Object src)
 {
  SerObject dst = new SerObject();
  dst.type = src.type;
  dst.fields = src.fields;
  dst.values = src.values;
  return dst;
 }
}

public abstract class GenericPlaceholder : Generator,  Serializer.ICustomSerialization
{
 public string origType;

 [NonSerialized] public string[] origFields = new string[0];
 [NonSerialized] public Serializer.Value[] origValues = new Serializer.Value[0];

 //TODO: does not serialize UnityObject references
 //TODO: does not serialize layers or multi-inlet links
```

```
public override void Generate (TileData data, StopToken stop) { }



public void PreprocessBeforeDeserialize (Serializer.Object serObj, Serializer.Object[] allSerialized, objec
/// Loading placeholder
/// Reading serObj and converting it to placeholder values
{
 origType = serObj.type;

 List<string> fieldsList = new List<string>();
 List<Serializer.Value> valuesList = new List<Serializer.Value>();

 for (int v=0; v<serObj.values.Length; v++)
 {
  if (serObj.values[v].t != 255) //if not reference
  {
   fieldsList.Add(serObj.fields[v]);
   valuesList.Add(serObj.values[v]);
  }

  /*else if (serObj.values[v] >= 0) //if not null
  {
    Serializer.Object fieldObj = allSerialized[serObj.values[v]];
    Type fieldType = Type.GetType(fieldObj.type);
    if (fieldType.IsGenericType &&
     IsInletType(fieldType) )
```

```
      Debug.Log(fieldType);
    }*/
   }

   origFields = fieldsList.ToArray();
   origValues = valuesList.ToArray();
  }


  public void PostprocessAfterSerialize (Serializer.Object serObj, Dictionary<object,Serializer.Object> allSe
  /// Storing original instead of placeholder
  /// Writing serObj
  {
   serObj.type = origType;

   ArrayTools.Append(ref serObj.fields, origFields);
   ArrayTools.Append(ref serObj.values, origValues);
  }
 }

public static bool IsInletType (Type type) =>
 type.GetInterfaces().Find(i => i.GetGenericTypeDefinition() == typeof(IInlet<>))  >=  0;

[GeneratorMenu (name = "Unknown", iconName="GeneratorIcons/Generator")]
public class InletOutletPlaceholder : GenericPlaceholder, IInlet<object>, IOutlet<object> { }
```

```
  [GeneratorMenu (name = "Unknown", iconName="GeneratorIcons/Generator")]
  public class InletPlaceholder : GenericPlaceholder, IInlet<object> { }

  [GeneratorMenu (name = "Unknown", iconName="GeneratorIcons/Generator")]
  public class OutletPlaceholder : GenericPlaceholder, IOutlet<object> { }

  [GeneratorMenu (name = "Unknown", iconName="GeneratorIcons/Generator")]
  public class Placeholder : GenericPlaceholder { }


  /*[GeneratorMenu (name = "Unknown", iconName="GeneratorIcons/Generator")]
  public class MultiInletOutletPlaceholder : GenericPlaceholder, IMultiInlet, IMultiOutlet
  {

  }*/
 }
}
```

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using MapMagic.Products;

namespace MapMagic.Nodes
{
 [GeneratorMenu (menu = "Map/Portals", name = "Enter", iconName = "GeneratorIcons/PortalIn", lookLike
 [GeneratorMenu (menu = "Map/Portals", name ="Exit", iconName = "GeneratorIcons/PortalOut", lookLikeP
 //[GeneratorMenu (menu = "Objects/Portals", name = "Enter", iconName = "GeneratorIcons/PortalIn", look
 //[GeneratorMenu (menu = "Objects/Portals", name ="Exit", iconName = "GeneratorIcons/PortalOut", look
 //[GeneratorMenu (menu = "Spline/Portals", name = "Enter", iconName = "GeneratorIcons/PortalIn", lookL
 //[GeneratorMenu (menu = "Spline/Portals", name ="Exit", iconName = "GeneratorIcons/PortalOut", lookLi


 public interface IPortalEnter<out T> : IInlet<T> where T:class
 {
  string Name {get; set; }
 }


 public interface IPortalExit<out T> :  IOutlet<T> where T:class
```

```
{
 IPortalEnter<T> Enter { get; }
 void AssignEnter (IPortalEnter<object> enter, Graph graph);
}



public interface IFnPortal<out T>  { string Name { get; set; } }


public interface IFnEnter<out T> : IFnPortal<T>, IOutlet<T>  where T: class { }  //to use objects of type IFn
public interface IFnExit<out T> : IFnPortal<T>, IInlet<T>, IRelevant where T: class { } //fnExit is always ge
//interfaces required in draw editor, so they are stored in portals.cs, not module



[System.Serializable]
[GeneratorMenu(name = "Generic Portal Enter")]
public class PortalEnter<T> : Generator, IInlet<T>, IPortalEnter<T>  where T: class, ICloneable
{
 public string name = "Portal";
 public string Name { get{ return name; } set{ name = value; } }

 public override void Generate (TileData data, StopToken stop) { }
}



[Serializable]
[GeneratorMenu (name ="Generic Portal Exit")]
```

```
public class PortalExit<T> : Generator, IOutlet<T>, IPortalExit<T>, ICustomDependence   where T: class,
{
 public PortalEnter<T> enter; //TODO: don't serialize, keep name only for serialization simplicity
 public IPortalEnter<T> Enter => enter;

 public void AssignEnter (IPortalEnter<object> ienter, Graph graph)
 {
  if (!(ienter is PortalEnter<T> enter)) return;
  //TODO: other validity check
  this.enter = enter;
 }

 public override void Generate (TileData data, StopToken stop)
 {
  if (enter != null   &&  !stop.stop)
  {
   data.StoreProduct(this, data.ReadInletProduct(enter));
   //TODO: clone?
  }
 }

 public IEnumerable<Generator> PriorGens ()
 {
  if (enter != null)
   yield return enter;
 }
```

```
 }

}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
using MapMagic.Products;
using MapMagic.Terrains;

namespace MapMagic.Nodes
{
 public enum BiomeBlend { Sharp, Random, Scale, Pure }

 [System.Serializable]
 public class PositioningSettings
 /// Contains the classical PRS settings for placing both trees and objects (for brush too)
 /// Not in objects but in Nodes since it's used by Brush core
 {
  //height
  public bool objHeight = true;
  public bool relativeHeight = true;
  public bool guiHeight;
```

```
//rotation
public bool useRotation = true; //in base since tree could also be rotated. Not the imposter ones, but anyw
public bool takeTerrainNormal = false;
public bool rotateYonly = false;
public bool regardPrefabRotation = false;
public bool guiRotation;

//scale
public bool useScale = true;
public bool scaleYonly = false;
public bool regardPrefabScale = false;
public bool guiScale;



public void MoveRotateScale (ref Transition trs, TileData data)
/// Floors object, and erases (yep) trs roation/scale values according to layer settings
{
 if (!objHeight) trs.pos.y = 0;

 //flooring
 float terrainHeight = 0;
 if (relativeHeight && data.heights != null) //if checbox enabled and heights exist (at least one height gene
  terrainHeight = data.heights.GetWorldInterpolatedValue(trs.pos.x, trs.pos.z, roundToShort:true);
 if (terrainHeight > 1) terrainHeight = 1;
 terrainHeight *= data.globals.height;  //all coords should be in world units
 trs.pos.y += terrainHeight;
```

```
 if (!useScale) trs.scale = new Vector3(1,1,1);
 else if (scaleYonly) trs.scale = new Vector3(1, trs.scale.y, 1);

 if (!useRotation) trs.rotation = Quaternion.identity;
 else if (takeTerrainNormal)
 {
  Vector3 terrainNormal = GetTerrainNormal(trs.pos.x, trs.pos.z, data.heights, data.globals.height, data.ar
  Vector3 terrainTangent = Vector3.Cross(trs.rotation*new Vector3(0,0,1), terrainNormal);
  trs.rotation = Quaternion.LookRotation(terrainTangent, terrainNormal);
 }
 else if (rotateYonly) trs.rotation = Quaternion.Euler(0,trs.Yaw,0);
}


public static Vector3 GetTerrainNormal (float fx, float fz, MatrixWorld heightmap, float heightFactor, float p
{
 Coord coord = heightmap.WorldToPixel(fx, fz);
 int pos = heightmap.rect.GetPos(coord);

 float curHeight = heightmap.arr[pos];

 float prevXHeight = curHeight;
 if (coord.x>=heightmap.rect.offset.x+1) prevXHeight = heightmap.arr[pos-1];

 float nextXHeight = curHeight;
```

```
 if (coord.x<=heightmap.rect.offset.x+heightmap.rect.size.x-1) nextXHeight = heightmap.arr[pos+1];

 float prevZHeight = curHeight;
 if (coord.z>=heightmap.rect.offset.z+1) prevZHeight = heightmap.arr[pos-heightmap.rect.size.x];

 float nextZHeight = curHeight;
 if (coord.z<=heightmap.rect.offset.z+heightmap.rect.size.z-1) nextZHeight = heightmap.arr[pos+heightma

 return new Vector3((prevXHeight-nextXHeight)*heightFactor, pixelSize*2, (prevZHeight-nextZHeight)*hei
}


public static bool SkipOnBiome (ref Transition trs, BiomeBlend biomeBlend, MatrixWorld biomeMask, Noi
/// True if object should not be spawned because of biome mask
/// ref since it can change scale
{
 float biomeFactor = biomeMask!=null ?  biomeMask.GetWorldInterpolatedValue(trs.pos.x, trs.pos.z) : 1;
 if (biomeFactor < 0.00001f) return true;

 bool skip;
 switch (biomeBlend)
 {
  case BiomeBlend.Sharp:
   skip = biomeFactor < 0.5f;
   break;
  case BiomeBlend.Random:
```

```
      float rnd = random.Random((int)trs.pos.x, (int)trs.pos.y); //TODO: use id?
      if (biomeFactor > 0.5f) rnd = 1-rnd;
      skip = biomeFactor < rnd;
      break;
     case BiomeBlend.Scale:
      trs.scale *= biomeFactor;
      skip = biomeFactor < 0.0001f;
      break;
     case BiomeBlend.Pure:
      skip = biomeFactor < 0.9999f;
      break;
     default: skip = false; break;
    }

    return skip;
   }
  }
}
```

```
ï»¿using System;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine;

namespace MapMagic.Nodes
{
 [Serializable]
 public class SharedValuesHolder : ISerializationCallbackReceiver
 {
  [NonSerialized] private Dictionary<(Type type, string name), object> sharedValues = new Dictionary<(Typ



  public SharedValuesHolder () { }
  public SharedValuesHolder (SharedValuesHolder src)
   { sharedValues = new Dictionary<(Type,string),object>(sharedValues); }



  public object GetValue (Type holderType, string name)
  {
   if (sharedValues.TryGetValue((holderType,name), out object val))
    return val;
   else
    //throw new Exception("Shared value " + name + " has not been added to holder");
    return null;
  }
```

```
public T GetValue<T> (Type holderType, string name)
{
 if (sharedValues.TryGetValue((holderType,name), out object val))
  return (T)val;
 else
  throw new Exception("Shared value " + name + " has not been added to holder");
}


public void SetValue (Type type, string name, object val)
{
 (Type type, string name) typeName = (type, name);
 if (sharedValues.ContainsKey(typeName))
  sharedValues[typeName] = val;
 else
  sharedValues.Add(typeName, val);
}


public void SetDefaults (object obj)
{
 Type type = obj.GetType();
 foreach (SharedValueDefaultAttribute defValAtt in type.GetCustomAttributes<SharedValueDefaultAttribu
 {
```

```
  (Type type, string name) typeName = (type, defValAtt.name);

  //adding if not yet added
  if (!sharedValues.ContainsKey(typeName))
   sharedValues.Add(typeName, defValAtt.val);

  //re-writing if val type changed
  else if (defValAtt.val.GetType() != sharedValues[typeName].GetType())
   sharedValues[typeName] = defValAtt.val;
 }
}



public (Type,string)[] GetTypeNames ()
{
 (Type, string)[] typeNames = new (Type,string)[sharedValues.Count];
 sharedValues.Keys.CopyTo(typeNames, 0);
 return typeNames;
}



Type[] serializedTypes = new Type[0];
string[] serializedNames = new string[0];
object[] serializedVals = new object[0];

public void OnBeforeSerialize ()
```

```
 {
  if (serializedTypes.Length != sharedValues.Count)
  {
   serializedTypes = new Type[sharedValues.Count];
   serializedNames = new string[sharedValues.Count];
   serializedVals = new object[sharedValues.Count];
  }

  int counter = 0;
  foreach (var kvp in sharedValues)
  {
   var typeName = kvp.Key;
   serializedTypes[counter] = typeName.type;
   serializedNames[counter] = typeName.name;
   serializedVals[counter] = kvp.Value;
   counter++;
  }
 }

 public void OnAfterDeserialize ()
 {
  sharedValues.Clear();
  for (int i=0; i<serializedTypes.Length; i++)
   sharedValues.Add((serializedTypes[i], serializedNames[i]), serializedVals[i]);
 }
}
```

```
 [AttributeUsage(AttributeTargets.Class, AllowMultiple = true)]
 public sealed class SharedValueDefaultAttribute : Attribute
 {
  public string name;
  public object val;

  public SharedValueDefaultAttribute(string name, object val) { this.name = name; this.val = val; }
 }

}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Products;

namespace MapMagic.Nodes.GUI
{

 public static class CurveDraw
 {
  private const int pointSize = 5;
  private const int moveRange = 10;
  private const int addRange = 5;
  private const int addRectDensity = 15; //each 15 pixels
  private const int maxSubSegments = 25;

  public const int uiSize = 146;
  public const int uiVertMargins = 6;
  public const float nodesDraggedZoom = 0.7f;

  private enum PointRemoveState { None, MovedOut, Removed }
```

```
  private static Curve draggedCurve; //curve currently dragging, a backup without a node removed. Presum



  public static void DrawCurve (Curve curve, float[] histogram)
  {
   if (!UI.current.layout && !(UI.current.optimizeElements && !UI.current.IsInWindow()))
   {
    if (histogram != null)
    {
     Material histogramMat = UI.current.textures.GetMaterial("Hidden/DPLayout/Histogram");
     histogramMat.SetFloatArray("_Histogram", histogram);
     histogramMat.SetVector("_Backcolor", new Vector4(0,0,0,0));
     histogramMat.SetVector("_Forecolor", new Vector4(0,0,0,0.25f));
//     Draw.Texture(null, histogramMat);
    }

    Draw.Grid(new Color(0,0,0,0.4f)); //background grid
   }

   if (UI.current.scrollZoom == null || UI.current.scrollZoom.zoom > 0.75f)
    CurveDraw.DragCurve(curve);
   CurveDraw.DisplayCurve(curve);
  }
```

```
public static void DragCurve (Curve curve)
/// Backup curve to return the removed nodes when dragging cursor returned to curve field
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 //moving
 bool isDragging = false;
 bool isReleased = false;
 int dragNum = -1;
 Curve.Node[] originalPoints = null;

 for (int i=0; i<curve.points.Length; i++)
 {
  bool newDragging = false; bool newReleased = false;
  Vector2 newPos = DragPoint(curve, i, ref newDragging, ref newReleased);

  if (newDragging)
  {
   originalPoints = new Curve.Node[curve.points.Length]; //curve.GetPositions();
   for (int p=0; p<originalPoints.Length; p++)
    originalPoints[p] = new Curve.Node(curve.points[p]);

   curve.points[i].pos = newPos;
  }
```

```
 if (newDragging || newReleased)
  dragNum = i;

 isDragging = isDragging || newDragging;
 isReleased = isReleased || newReleased;
}

//adding
if (ClickedNearCurve(curve))
{
 Vector2 addPos = ToCurve(UI.current.mousePos);
 int addedNum = AddPoint(curve, addPos);

 //starting drag
 DragPoint (curve, addedNum, ref isDragging, ref isReleased);  //just to start drag
}

//removing
if (isDragging)
{
 //calc if node should be removed
 bool isRemoved = false;
 if (dragNum!=0  &&  dragNum!=curve.points.Length-1) //ignoring first and last
 {
  Vector2 pos = ToCell(curve.points[dragNum].pos);
  if (!Cell.current.InternalRect.Extended(10).Contains(pos))
```

```
   isRemoved = true;
  }

  //removing
  if (isRemoved)
  {
   UI.current.MarkChanged(completeUndo:true);
   ArrayTools.RemoveAt(ref curve.points, dragNum);
  }

  //clamping if cursor is too close to the field to remove
  else
   ClampPoint(curve, dragNum);
 }

 //if returned dragging to field
 else if (DragDrop.obj != null && DragDrop.obj.GetType() == typeof((Curve,Curve.Node,int)) )
 {
  (Curve curve, Curve.Node node, int num) dragObj = ((Curve,Curve.Node,int))DragDrop.obj;
  if (dragObj.curve == curve && !curve.points.Contains(dragObj.node))
  {
   DragDrop.TryDrag(dragObj, UI.current.mousePos);
   //to make it repaint

   if (Cell.current.InternalRect.Extended(10).Contains(UI.current.mousePos))
   {
```

```
    ArrayTools.Insert(ref curve.points, dragObj.num, dragObj.node);
    dragObj.node.pos = ToCurve(UI.current.mousePos);
    ClampPoint(dragObj.curve, dragObj.num); //this will place it between prev and next points
   }

   DragDrop.TryRelease(dragObj, UI.current.mousePos);
   //otherwise it will not be released forever
  }
 }

 if (Cell.current.valChanged)
  curve.Refresh();
}


public static void DisplayCurve (Curve curve)
/// Just draws curve without dragging nodes
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 //drawing curve nodes
 if (UI.current.scrollZoom == null  ||  UI.current.scrollZoom.zoom > nodesDraggedZoom)
  for (int i=0; i<curve.points.Length; i++)
 {
  Vector2 cellPos = ToCell(curve.points[i].pos);
```

```
 Draw.Icon(UI.current.textures.GetTexture("DPUI/Curve/Key"), cellPos);
}

//dispPos.x = (int)(dispPos.x + 1.5f); //1-pixel offset to place nice
//dispPos.y = (int)(dispPos.y + 0.5f);

//Rect pointRect = new Rect(dispPos.x-(scrolledSize-1)/2, dispPos.y-(scrolledSize-1)/2, scrolledSize,scro
//Texture2D pointTex = UI.current.textures.GetTexture("DPUI/Curve/Key");
//UnityEngine.GUI.DrawTexture(pointRect, pointTex);

//drawing curve itself
Material curveMat = UI.current.textures.GetMaterial("Hidden/DPLayout/Curve");
curveMat.SetVector("_Forecolor", new Vector4(0,0,0,1));
curveMat.SetVector("_CurveRect", Cell.current.GetRect(UI.current.scrollZoom).ToV4());
curveMat.SetFloatArray("_Curve", curve.lut);
Draw.Texture(null, curveMat);
}


[Obsolete] public static void DisplayCurve_AAPolyLine (Curve curve)
/// Older way to display a curve without shader
{
if (UI.current.layout) return;
if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

UnityEditor.Handles.color = Color.black;
```

```
 if (UI.current.scrollZoom == null  ||  UI.current.scrollZoom.zoom > nodesDraggedZoom)
 for (int i=0; i<curve.points.Length; i++)
  DrawPoint(curve.points[i]);

 int numSubSegments = UI.current.scrollZoom != null ?
  (int)(maxSubSegments*UI.current.scrollZoom.zoom) :
  maxSubSegments;
 if (numSubSegments < 3) numSubSegments = 3;
 Vector3[] posArray = new Vector3[numSubSegments]; //positions array re-use
 for (int i=0; i<curve.points.Length-1; i++)
  DrawSegment(curve.points[i], curve.points[i+1], posArray, Cell.current);

 //first and last segments
 if (curve.points[0].pos.x > 0.0001f)
  DrawHorizontalSegment(new Vector2(0,curve.points[0].pos.y), curve.points[0].pos, posArray);

 int lastNum = curve.points.Length-1;
  DrawHorizontalSegment(curve.points[curve.points.Length-1].pos, new Vector2(1,curve.points[ curve.poi

}

private static void DrawPoint (Curve.Node node)
```

```
{
 float zoom = UI.current.scrollZoom != null ? UI.current.scrollZoom.zoom : 1;
 float scrolledSize = pointSize*UI.current.scrollZoom.zoom / 2f  +  pointSize / 2f;

 Vector2 cellPos = ToCell(node.pos);
 Vector2 dispPos = UI.current.scrollZoom != null ?
  UI.current.scrollZoom.ToScreen(cellPos) :
  cellPos;

 //dispPos.x = (int)(dispPos.x + 1.5f); //1-pixel offset to place nice
 //dispPos.y = (int)(dispPos.y + 0.5f);

 Rect pointRect = new Rect(dispPos.x-(scrolledSize-1)/2, dispPos.y-(scrolledSize-1)/2, scrolledSize,scroll
 Texture2D pointTex = UI.current.textures.GetTexture("DPUI/Curve/Key");
 UnityEngine.GUI.DrawTexture(pointRect, pointTex);
}


private static void DrawSegment (Curve.Node prevNode, Curve.Node nextNode, Vector3[] posArray, Cell
{
 if (prevNode.pos.x > nextNode.pos.x)
  { Debug.LogError("Wrong curve nodes order"); return; }  //throw new Exception("Wrong curve nodes ord

 float range = nextNode.pos.x - prevNode.pos.x;

 int pointsNum = (int)(posArray.Length*range) + 5;
```

```
if (pointsNum > posArray.Length) pointsNum = posArray.Length; //in case spline has the only segment


float step = 1f / (pointsNum-1);


//making time/value array
for (int i=0; i<pointsNum; i++)
{
 float val = Curve.EvaluatePrecise(prevNode, nextNode, i*step);
 posArray[i] = new Vector3(prevNode.pos.x + i*step*range, val, 0);
}


//clamping time/values
for (int i=1; i<pointsNum-1; i++)
 posArray[i] = Clamp01Line(posArray[i-1], posArray[i], posArray[i+1]);
Clamp01Line(posArray[0], posArray[1], posArray[1]);
Clamp01Line(posArray[pointsNum-2], posArray[pointsNum-2], posArray[pointsNum-1]);



//transforming time/values to point coordinates
for (int i=0; i<pointsNum; i++)
{
 Vector2 pos = ToCell(posArray[i]);
 if (UI.current.scrollZoom != null) pos = UI.current.scrollZoom.ToScreen(pos);
 posArray[i] = pos  +  new Vector2(0.5f, 0.5f); //adding 0.5 to make the line center-sized at pixel
}
```

```
 UnityEditor.Handles.color = Color.black;
 UnityEditor.Handles.DrawAAPolyLine(2, pointsNum, posArray);
}


private static void DrawHorizontalSegment (Vector2 start, Vector2 end, Vector3[] posArray)
{
 posArray[0] = UI.current.scrollZoom != null ?
  UI.current.scrollZoom.ToScreen( ToCell(start) ) :
  ToCell(start);
 posArray[1] = UI.current.scrollZoom != null ?
  UI.current.scrollZoom.ToScreen( ToCell(end) ) :
  ToCell(end);

 //rounding to in-between pixel
 posArray[0].x = (int)(posArray[0].x)+1f;  posArray[0].y = (int)(posArray[0].y)+1f;
 posArray[1].x = (int)(posArray[1].x)+1f;  posArray[1].y = (int)(posArray[1].y)+1f;

 UnityEditor.Handles.color = Color.black;
 UnityEditor.Handles.DrawAAPolyLine(2, 2, posArray);
}


private static Vector2 DragPoint (Curve curve, int num, ref bool isDragging, ref bool isReleased)
/// Will not move the point, just returns it's new dragged position
{
```

```
Curve.Node node = curve.points[num];
Vector2 pos = ToCell(node.pos);
Vector2 newPos = pos;
Rect rect = new Rect(pos.x-moveRange, pos.y-moveRange, 1+moveRange*2, 1+moveRange*2);

//cursor
Rect dispRect = UI.current.scrollZoom != null ?
 UI.current.scrollZoom.ToScreen(rect.position, rect.size) :
 new Rect(rect.position, rect.size);
UnityEditor.EditorGUIUtility.AddCursorRect(dispRect, UnityEditor.MouseCursor.MoveArrow);

(Curve,Curve.Node,int) dragObj = (curve,node,num);

if (DragDrop.TryDrag(dragObj, UI.current.mousePos))
{
 newPos = DragDrop.initialMousePos + DragDrop.totalDelta;
 newPos = ToCurve(newPos);

 if (newPos.x > pos.x+0.001f || newPos.x < pos.x-0.001f || newPos.y > pos.y+0.001f || newPos.y < pos.y-
  UI.current.MarkChanged();

 isDragging = true;
}

if (DragDrop.TryRelease(dragObj, UI.current.mousePos))
 isReleased = true;
```

```
 if (DragDrop.TryStart(dragObj, UI.current.mousePos, rect))
  DragDrop.group = "DragCurve";

 return newPos;
}


private static bool ClampPoint (Curve curve, int num)
/// return true if point was clamped (moved)
{
 bool clamped = false;

 //between min and max
 if (curve.points[num].pos.x < 0) { curve.points[num].pos.x = 0; clamped=true; }
 if (curve.points[num].pos.x > 1) { curve.points[num].pos.x = 1; clamped=true; }
 if (curve.points[num].pos.y < 0) { curve.points[num].pos.y = 0; clamped=true; }
 if (curve.points[num].pos.y > 1) { curve.points[num].pos.y = 1; clamped=true; }

 //between prev and next
 if (num != 0)
 {
  float prev = curve.points[num-1].pos.x;
  if (curve.points[num].pos.x < prev+0.001f) { curve.points[num].pos.x = prev+0.001f;  clamped=true; }
 }
```

```
 if (num != curve.points.Length-1)
 {
  float next = curve.points[num+1].pos.x;
  if (curve.points[num].pos.x > next-0.001f) { curve.points[num].pos.x = next-0.001f; clamped=true; }
 }

 return clamped;
}



private static bool ClickedNearCurve (Curve curve)
{
 //TODO: to optimize A LOT. Calculating AddCursorRect each frame are damn slow!
 //however it's called only in cursor is in curve field

 if (UI.current.layout) return false;
 //if (Event.current.type != EventType.MouseDown) return false;
 //if (!Cell.current.Rect.Extended(10).Contains(UI.mousePos)) return false;

 for (int s=0; s<curve.points.Length-1; s++)
 {
  Curve.Node prevNode = curve.points[s];
  Curve.Node nextNode = curve.points[s+1];
  float range = nextNode.pos.x - prevNode.pos.x;
  float start = prevNode.pos.x;
```

```
int numAddRects = (int)(range * Cell.current.finalSize.x / addRectDensity) + 2;

for (int i=0; i<numAddRects-1; i++)
{
 float time = 1f*i / (numAddRects-1);
 float val = Den.Tools.Curve.EvaluatePrecise(prevNode,nextNode,time);
 Vector2 prev = ToCell( new Vector2(start + time*range,val) );

 time = 1f*(i+1) / (numAddRects-1);
 val = Den.Tools.Curve.EvaluatePrecise(prevNode,nextNode,time);
 Vector2 next = ToCell( new Vector2(start + time*range,val) );

 Rect rect = new Rect(
  prev.x,
  Mathf.Min(next.y, prev.y),
  next.x-prev.x,
  Mathf.Abs(next.y-prev.y) );

 if (rect.y < Cell.current.worldPosition.y) rect.y = Cell.current.worldPosition.y;
 if (rect.max.y > Cell.current.worldPosition.y+Cell.current.finalSize.y) rect.max = new Vector2(rect.max.x
 if (rect.height < 0) { rect.y += rect.height; rect.height = 0; }

 rect = rect.Extended(addRange);

 #if UNITY_EDITOR
 Rect dispRect = UI.current.scrollZoom != null ?
```

```
    UI.current.scrollZoom.ToScreen(rect.position, rect.size) :
    new Rect(rect.position, rect.size);
   UnityEditor.EditorGUIUtility.AddCursorRect(dispRect, UnityEditor.MouseCursor.ArrowPlus);
   //UnityEditor.EditorGUI.DrawRect(dispRect,Color.red);
   #endif

   if (Event.current.type==EventType.MouseDown  &&  rect.Contains(UI.current.mousePos)  &&  Event.cu
   {
    //excluding points that should be dragged instead
    for (int n=0; n<curve.points.Length; n++)
    {
     Vector2 nPos = ToCell(curve.points[n].pos);
     Rect nRect = new Rect(nPos.x-moveRange, nPos.y-moveRange, 1+moveRange*2, 1+moveRange*2)
     if (nRect.Contains(UI.current.mousePos)) return false;
    }

    return true;
   }
  }
 }

 return false;
}


private static int AddPoint (Curve curve, Vector2 addPos)
```

```
/// Returns the number of added point. AddPos is in curve-relative coordinates
{
 //finding segemnt
 int addSegment = 0;
 if (addPos.x < curve.points[0].pos.x) addSegment = 0;
 else if (addPos.x > curve.points[curve.points.Length-1].pos.x) addSegment = curve.points.Length;
 else
  for (int p=0; p<curve.points.Length-1; p++)
  {
   if (addPos.x > curve.points[p].pos.x && addPos.x < curve.points[p+1].pos.x)
    addSegment = p+1;
  }

 UI.current.MarkChanged();

 ArrayTools.Insert(ref curve.points, addSegment, new Curve.Node(addPos));

 return addSegment;
}


private static Vector2 Clamp01Line (Vector3 prev, Vector3 current, Vector3 next)
{
 if (current.y > 1)
 {
  if (prev.y < 1)
```

```
 {
  float clampPercent = (current.y-1) / (current.y-prev.y);
  current.x = prev.x + (current.x-prev.x)*(1-clampPercent);
 }

 if (next.y < 1)
 {
  float clampPercent = (current.y-1) / (current.y-next.y);
  current.x = next.x + (current.x-next.x)*(1-clampPercent);
 }

 current.y = 1;
}

if (current.y < 0)
{
 if (prev.y > 0)
 {
  float clampPercent = current.y / (current.y-prev.y);
  current.x = prev.x + (current.x-prev.x)*(1-clampPercent);
 }

 if (next.y > 0)
 {
  float clampPercent = current.y / (current.y-next.y);
  current.x = next.x + (current.x-next.x)*(1-clampPercent);
```

```
 }

 current.y = 0;
}

return current;
}

private static void DrawSegmentBeizer (Curve curve, int num, Vector3[] posArray, Cell cell)
///For debug purpose
{
 /*Vector2 prevPointPercent = curve.points[num-1].pos;
 Vector2 prevPointCenter = cell.finalOffset + new Vector2(prevPointPercent.x*cell.finalSize.x, (1-prevPoin
 prevPointCenter = scrollZoom.ToDisplay(prevPointCenter);

 Vector2 prevTangentPercent = curve.points[num-1].outTangent + prevPointPercent;
 Vector2 prevTangentCenter = cell.finalOffset + new Vector2(prevTangentPercent.x*cell.finalSize.x, (1-pre
 prevTangentCenter = scrollZoom.ToDisplay(prevTangentCenter);

 Vector2 nextPointPercent = curve.points[num].pos;
 Vector2 nextPointCenter = cell.finalOffset + new Vector2(nextPointPercent.x*cell.finalSize.x, (1-nextPoint
 nextPointCenter = scrollZoom.ToDisplay(nextPointCenter);

 Vector2 nextTangentPercent = curve.points[num].inTangent + nextPointPercent;
 Vector2 nextTangentCenter = cell.finalOffset + new Vector2(nextTangentPercent.x*cell.finalSize.x, (1-ne
```

```
nextTangentCenter = scrollZoom.ToDisplay(nextTangentCenter);

UnityEditor.Handles.DrawBezier(
 prevPointCenter,
 nextPointCenter,
 prevTangentCenter,
 nextTangentCenter,
 Color.red,
 Cached.GetTexture("DPUI/SplineTex", theme:stylesTheme),
 1);




for (int i=0; i<20; i++)
{
 Curve.Node start = curve.points[num-1];
 Curve.Node end = curve.points[num];

 float p = i/20f;
 float ip = 1f-p;
 Vector2 pos = ip*ip*ip*start.pos + 3*p*ip*ip*(start.pos+start.outTangent) + 3*p*p*ip*(end.pos+end.inTang

 pos = cell.finalOffset + new Vector2(pos.x*cell.finalSize.x, (1-pos.y)*cell.finalSize.y);
 pos = ui.scrollZoom.ToDisplay(pos);

 posArray[i] = pos;
```

```
 }

 UnityEditor.Handles.color = Color.yellow;
 UnityEditor.Handles.DrawAAPolyLine(1, 20, posArray);*/
}

private static Rect NodeRect (Curve.Node node)
{
 Vector2 pos = ToCell(node.pos);
 return new Rect(pos.x-moveRange, pos.y-moveRange, 1+moveRange*2, 1+moveRange*2);
}


private static Vector2 ToCell (Vector2 pos)
{
 Cell cell = Cell.current;

 pos.x = cell.worldPosition.x + pos.x*(cell.finalSize.x-1);
 pos.y = cell.worldPosition.y + (1-pos.y)*(cell.finalSize.y-1);
 return pos;
}


private static Vector2 ToCurve (Vector2 pos)
{
 Cell cell = Cell.current;
```

```
    pos.x = (pos.x - cell.worldPosition.x) / (cell.finalSize.x-1);
    pos.y = 1 - (pos.y - cell.worldPosition.y)/(cell.finalSize.y-1);
    return pos;
   }
  }
 }
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;
//using Den.Tools.Segs;
using Den.Tools.Splines;
using MapMagic.Core;  //used once to get tile size
using MapMagic.Products;
using MapMagic.Expose.GUI;
using MapMagic.Expose;

namespace MapMagic.Nodes.GUI
{
 public static partial class GeneratorDraw
 {
  public const int nodeWidth = 150;
  public const int miniWidth = 140;
  public const int headerHeight = 24;
  public const int inletOutletDragArea = 20;
  public static readonly RectOffset shadowBorders = new RectOffset(38,38,38,38);
```

```
public static readonly RectOffset shadowOverflow = new RectOffset(32, 32, 32, 32);
public static readonly RectOffset frameBorders = new RectOffset(1, 1, 1, 1);
public static readonly RectOffset selectionBorders = new RectOffset(2, 2, 2, 2);
public const int portalHeight = 26;

private const float miniInletsScale = 0.875f / GraphWindow.miniZoom;
private const float miniInletLineHeight = 12 / GraphWindow.miniZoom;
private static GUIStyle miniNameStyle;
private static GUIStyle miniInletStyle;


public static void DrawGeneratorOrPortal (Generator gen, Graph graph, bool isMini=false, bool selected=
{
 Type genType = gen.GetType();
 GeneratorMenuAttribute menuAtt = GeneratorDraw.GetMenuAttribute(genType);

 if (menuAtt.lookLikePortal)
  GeneratorDraw.DrawPortal(gen, graph, selected:selected, isMini:isMini);

 else
 {
  try
  {
   if (!isMini) GeneratorDraw.DrawGenerator(gen, graph, selected:selected, menuAtt:menuAtt);
   else GeneratorDraw.DrawMini(gen, graph, selected:selected, menuAtt:menuAtt);
  }
```

```
  catch (ExitGUIException)
   { } //ignoring
  catch (Exception e)
   { throw e; }
                //{ Debug.LogError($"Draw Generator {genType} failed: \n" + e); }
 }
}



public static void DrawGenerator (Generator gen, Graph graph, bool selected=false, bool activeLinks=true
{
 if (UI.current.layout)
  UI.current.cellObjs.ForceAdd(gen, Cell.current, "Generator");

 if (menuAtt == null)
 {
  Type genType = gen.GetType();
  menuAtt = GetMenuAttribute(genType);
 }

 //background
 if (!UI.current.layout)
 {
  //shadow
  GUIStyle shadowStyle = UI.current.textures.GetElementStyle(selected ? "MapMagic/Node/SelectionSha
   borders:shadowBorders,
```

```
     overflow:shadowOverflow);
    #if !MM_DEBUG
     Draw.Element(shadowStyle);
    #else
     if (graph.debugGraphBackground)
      Draw.Element(shadowStyle);
    #endif

    //frame
    GUIStyle frameStyle = UI.current.textures.GetElementStyle(selected ? "MapMagic/Node/SelectionFrame
     borders:selected ? selectionBorders : frameBorders,
     overflow:selected ? selectionBorders : frameBorders);
    Draw.Element(frameStyle);

    //gray field color (background to all node, including the header)
    GUIStyle fieldBackStyle = UI.current.textures.GetElementStyle("MapMagic/Node/Background");
    Draw.Element(fieldBackStyle);
   }

   //header
//   using (Timer.Start("Header"))
   using (Cell.LinePx(24)) //(Cell.LinePx(0)))
   {
    DrawHeader(gen, menuAtt, activeLinks);
   }
```

```
   //field
   bool fieldDrawn = false;
//   using (Timer.Start("Field"))
   using (Cell.LinePx(0))
   {
    Cell.current.fieldWidth = 0.44f;

    using (Cell.LinePx(0))
    {
     Cell.EmptyRowPx(1);

     using (Cell.Row)
     {
      if (!menuAtt.advancedOptions)
       fieldDrawn = CellExpose.ExposableClass(gen, gen.id);

      else
      {
       fieldDrawn = true;

       using (Cell.LinePx(0))
        CellExpose.ExposableClass(gen, gen.id);

       Cell.EmptyLinePx(2);
       using (Cell.LinePx(0))
```

```
    using (new Draw.FoldoutGroup(ref gen.guiAdvanced, "Advanced", padding:0))
     if (gen.guiAdvanced)
      CellExpose.ExposableClass(gen, gen.id, category:"Advanced");
   }
  }

  Cell.EmptyRowPx(1);
 }

 using (Cell.LinePx(0))
  fieldDrawn = Draw.Editor(gen) || fieldDrawn;

 if (Cell.current.valChanged)
 {
  gen.version ++;
  GraphWindow.current?.RefreshMapMagic();
 }
}



//debug
#if MM_DEBUG
//if (graph.debugGenInfo)
{
 DrawDebug(gen);
 fieldDrawn = true;
```

```
    }
    #endif

    //preview
    if (gen is IOutlet<object> &&  gen.guiPreview)
    {
     using (Cell.LinePx(nodeWidth))
      Previews.PreviewDraw.DrawPreview((IOutlet<object>)gen);
    }



    //footer
    float footerHeight = (gen is IOutlet<object>) ? 10 : 4;
    using (Cell.LinePx(footerHeight))
//    using (Timer.Start("Footer"))
    {
     Color color = menuAtt.color;
     if (!gen.enabled) color = DisableGeneratorColor(color);
     Texture2D footerTex = UI.current.textures.GetColorizedTexture(fieldDrawn ? "MapMagic/Node/Footer" :
     Draw.Texture(footerTex);

     if (gen is IOutlet<object>)
     {
      Cell.EmptyRow();

      Texture2D chevronIcon = UI.current.textures.GetTexture(gen.guiPreview ? "DPUI/Chevrons/TickUp" : "
```

```
   using (Cell.RowPx(100)) Draw.CheckButton(ref gen.guiPreview, chevronIcon, visible:false);

   Cell.EmptyRow();
  }
 }


 if (!UI.current.layout)
 {
  gen.guiSize.y = Cell.current.finalSize.y;
  gen.guiSize.x = nodeWidth;
 }
}


public static void DragGenerator (Generator gen, HashSet<Generator> otherSelected=null)
/// Returns the position of generator while dragging
{
 if (UI.current.layout) return;

 Cell cell = UI.current.cellObjs.GetCell(gen, "Generator");

 if (DragDrop.TryDrag(cell, UI.current.mousePos))
 {
  gen.guiPosition = MoveGenerator(cell, DragDrop.initialRect.position + DragDrop.totalDelta);
```

```
  //dragging other selected
  if (otherSelected!=null  &&  otherSelected.Contains(gen)) //if clicked one of selected
  {
   foreach (Generator ogen in otherSelected)
    if (ogen != gen) //to avoid moving twice
    {
     Cell ogenCell = UI.current.cellObjs.GetCell(ogen, "Generator");
     ogen.guiPosition = MoveGenerator(ogenCell, ogen.guiPosition + DragDrop.currentDelta);
    }
  }
 }

 DragDrop.TryRelease(cell);

 DragDrop.TryStart(cell, UI.current.mousePos, cell.InternalRect);
}

public static Vector2 MoveGenerator (Cell cell, Vector2 moveTo)
/// Internal move function for DragGenerator
/// Returns where was generator actually moved
{
 GraphWindow.current.graphUI.undo.Record();

 if (Event.current.control)
 {
  moveTo.x = (int)(float)(moveTo.x / 32 + 0.5f) * 32;
```

```
  moveTo.y = (int)(float)(moveTo.y / 32 - 0.5f) * 32;
 }

 float dpiFactor = UI.current.DpiScaleFactor;
 float zoom = UI.current.scrollZoom != null ? UI.current.scrollZoom.zoom : 1;

 Vector2 roundVal = new Vector2(  //0.5002 prevents cells un-align for the reason I don't remember
  moveTo.x > 0  ?  0.5002f  :  -0.5002f,
  moveTo.y > 0  ?  0.5002f  :  -0.5002f );

 moveTo.x = (int)(float)(moveTo.x*dpiFactor*zoom + roundVal.x) / (dpiFactor*zoom);
 moveTo.y = (int)(float)(moveTo.y*dpiFactor*zoom + roundVal.y) / (dpiFactor*zoom);

 //gen.guiPosition = moveTo;

 cell.worldPosition = moveTo;
 cell.CalculateSubRects(); //re-layout cell

 return moveTo;
}


public static void DrawHeader  (Generator gen, GeneratorMenuAttribute menuAtt, bool activeLinks=true)
{
 Color color = menuAtt.color;
 if (!gen.enabled) color = DisableGeneratorColor(color);
```

```
Texture2D headerTex = UI.current.textures.GetColorizedTexture("MapMagic/Node/Header", color);
//GUIStyle headBackStyle = UI.current.textures.GetElementStyle(headerTex);//, borders:new RectOffset(
Draw.Texture(headerTex);

//label & inline inlets/outlets
using (Cell.LinePx(24))
{
 if (gen is IInlet<object> inletGen)
 {
  using (Cell.RowPx(0))
   DrawInlet(inletGen, gen, addCellObj:activeLinks);

  Cell.EmptyRowPx(10);
  using (Cell.Row) Draw.Label(menuAtt.nameUpper, style:UI.current.styles.bigLabel);

  if (menuAtt.nameWidth < nodeWidth-24 -10 -(gen is IOutlet<object> ? 10 : 0))
   using (Cell.RowPx(24)) Draw.Icon(menuAtt.icon, scale:0.5f);
 }

 else
 {
  using (Cell.Row) Draw.Label(menuAtt.nameUpper, style:UI.current.styles.bigLabel);

  if (menuAtt.nameWidth < nodeWidth-24)
   using (Cell.RowPx(24)) Draw.Icon(menuAtt.icon, scale:0.5f);
 }
```

```
 if (gen is IOutlet<object> outletGen)
 {
  Cell.EmptyRowPx(10);
  using (Cell.RowPx(0))
   DrawOutlet(outletGen, addCellObj:activeLinks);
 }
}

//multi inlets/outlets
if (gen is IMultiInlet  ||  gen is IMultiOutlet)
 using (Cell.LinePx(0))
 {
  using (Cell.Row)
  {
   if (gen is IMultiInlet multiInGen)
   {
    ValAttribute[] inletVals = GetInletVals(menuAtt.type);

    foreach (ValAttribute val in inletVals)
     using (Cell.LineStd)
     {
      using (Cell.RowPx(0))
      {
       IInlet<object> inlet = (IInlet<object>)val.field.GetValue(gen);
       DrawInlet(inlet, gen, addCellObj:activeLinks);
```

```
    }
    Cell.EmptyRowPx(10);
    using (Cell.Row) Draw.Label(val.name);
   }
 }
}

using (Cell.Row)
{
 if (gen is IMultiOutlet multiOutGen)
 {
  ValAttribute[] outletVals = GetOutletVals(menuAtt.type);

  foreach (ValAttribute val in outletVals)
   using (Cell.LineStd)
   {
    using (Cell.Row) Draw.Label(val.name, UI.current.styles.rightLabel);
    Cell.EmptyRowPx(12);

    using (Cell.RowPx(0))
    {
     IOutlet<object> outlet = (IOutlet<object>)val.field.GetValue(gen);
     DrawOutlet(outlet, addCellObj:activeLinks);
    }
   }
 }
```

```
   }
  }

 //custom header editor
 using (Cell.LinePx(0))
  Draw.Editor(gen, cat:"Header");
}



public static void DrawLayersAddRemove<T>  (Generator gen, ref T[] layers, bool inversed=false, bool ur
 layers = DrawLayersAddRemove (gen, layers, inversed);

public static T[] DrawLayersAddRemove<T> (Generator gen, T[] layers, bool inversed=false, bool unlinkB
{
 float backCol = StylesCache.isPro ? 0.25f : 0.66f;
 Draw.Rect( new Color(backCol, backCol, backCol) );

 Cell layersCell = Cell.Parent;

 using (Cell.LinePx(20))
  LayersEditor.DrawAddRemove(layersCell, "Layers",
   onAdd: n => AddLayer<T>(gen, ref layers, inversed, n),
   onRemove: n => RemoveLayer<T>(gen, ref layers, inversed, n, unlinkBackground),
   onMove: (f,t) => MoveLayer<T>(gen, ref layers, inversed, f, t, unlinkBackground) );
```

```
 return layers;
}

public static void DrawLayersThemselves<T> (Generator gen, T[] layers, bool inversed=false, Action<Ge
{
 float backCol = StylesCache.isPro ? 0.25f : 0.66f;
 Draw.Rect( new Color(backCol, backCol, backCol) );

 Cell layersCell = Cell.Parent;

 using (Cell.LinePx(0))
  using (Cell.Padded(-1,-1,0,0))
   LayersEditor.DrawLayersThemselves(layersCell, layers.Length,
    onDraw: n => DrawLayer<T>(gen, ref layers, inversed, n, layerEditor),
    roundBottom:false);

 //if (Cell.current.valChanged  &&  GraphWindow.current.mapMagic!=null)
 // GraphWindow.RefreshMapMagic(gen);
 //Already called from DrawGenerator (field)
}

private static void DrawLayer<T> (Generator gen, ref T[] layers, bool inversed, int num, Action<Generator
{
 if (inversed) num = layers.Length-1 - num; //inversed num

 UI.current.cellObjs.ForceAdd(layers[num], Cell.current, "Layer");
```

```
 Draw.Class(layers[num]);
 Draw.Editor(layers[num], new object[] {num, gen} );
 layerEditor?.Invoke(gen, num);
}

private static void AddLayer<T> (Generator gen, ref T[] layers, bool inversed, int num) where T : class, ne
{
 num = inversed ? layers.Length : 0;

 T layer = new T();
 //(T)Activator.CreateInstance(layerType); //object layer = new T(); // layGen.CreateLayer();

 if (layer is IUnit unitLayer) { unitLayer.SetGen(gen); unitLayer.Id = Id.Generate(); }
 //if (layer is IInlet<object> inletLayer) { inletLayer.SetGen(gen); inletLayer.Id = gen.NewLayerId; }
 //if (layer is IOutlet<object> outletLayer) { outletLayer.SetGen(gen); outletLayer.Id = gen.NewLayerId; }
 //inlets and outlets are already IUnit

 ArrayTools.Insert(ref layers, num, layer);
}

private static void RemoveLayer<T> (Generator gen, ref T[] layers, bool inversed, int num, bool unlinkBac
///IDEA: To graph or generator since it uses not much of gui stuff (but usesGraphWindow.current.graph)
{
 if (inversed) num = layers.Length-1 - num;
```

```
 if (layers[num] is IOutlet<object> outlet) GraphWindow.current.graph.UnlinkOutlet(outlet);
 if (layers[num] is IInlet<object> inlet) GraphWindow.current.graph.UnlinkInlet(inlet);

 if (layers[num] is IUnit unitLayer)
  GraphWindow.current.graph.exposed.Remove(unitLayer.Id);

 ArrayTools.RemoveAt(ref layers, num);

 //unlinking background inlet (we could remove tthe background layer and now here's the new one)
 if (unlinkBackground  &&  layers[0] is IInlet<object> nInlet)
  GraphWindow.current.graph.UnlinkInlet(nInlet);
}

private static void MoveLayer<T> (Generator gen, ref T[] layers, bool inversed, int from, int to, bool unlink
{
 if (inversed)
 {
  from = layers.Length-1 - from;
  to = layers.Length-1 - to;
 }

 ArrayTools.Move(layers, from, to);

 //if (layGen.Expanded == from) layGen.Expanded = to;
 //if (layGen.Expanded == to) layGen.Expanded = from;
```

```
 //unlinking background inlet
 if (unlinkBackground  &&  layers[0] is IInlet<object> nInlet)
  GraphWindow.current.graph.UnlinkInlet(nInlet);
}



#if MM_DEBUG
private static void DrawDebug (Generator gen)
{
 using (Cell.LinePx(0))
  using (new Draw.FoldoutGroup(ref gen.guiDebug, "Debug"))
   if (gen.guiDebug)
   {
    using (Cell.LineStd) Draw.Label("Id: " + Id.ToString(gen.id));
    //using (Cell.LineStd) Draw.Label(gen.id.ToString());

    Cell.EmptyLinePx(5);
    using (Cell.LineStd) Draw.DualLabel("Version GUI:", gen.version.ToString());

    TileData lastData = null;
    if (GraphWindow.current.mapMagic is MapMagicObject mapMagicObject) //trying to use preview data f
     lastData = mapMagicObject?.PreviewData;

    if (lastData != null)
    {
     Graph rootGraph = GraphWindow.current.mapMagic.Graph;
```

```
 foreach ((Graph subGraph, TileData subData) in Graph.AllGraphsDatas(rootGraph,lastData,includeSe
 {
  if (!subGraph.ContainsGenerator(gen))
   continue;

  ulong? dataVersion = subData.Version(gen);
  using (Cell.LineStd) Draw.DualLabel("Version Data:", dataVersion!=null ? dataVersion.Value.ToString
  using (Cell.LineStd) Draw.Toggle(subData.IsReady(gen), "Ready");

  Cell.EmptyLinePx(5);
  if (gen is ICustomComplexity)
   using (Cell.LineStd) Draw.DualLabel("Progress:", subData.GetProgress((ICustomComplexity)gen).To
  if (gen is IOutlet<object> outlet)
  {
   object product = subData.ReadOutletProduct(outlet);
   string hashString = "null";
   if (product != null)
   {
    int hashCode = System.Runtime.CompilerServices.RuntimeHelpers.GetHashCode(product);
    hashString = Convert.ToBase64String( BitConverter.GetBytes(hashCode) );
   }
   using (Cell.LineStd) Draw.DualLabel("Product:", hashString);
  }
 }
}
```

```
    else
     using (Cell.LineStd) Draw.Label("No preview data");



    using (Cell.LineStd) Draw.DualLabel("Time Draft:", gen.draftTime.ToString("0.00") + "ms");
    using (Cell.LineStd) Draw.DualLabel("Time Main:", gen.mainTime.ToString("0.00") + "ms");
   }
}

#endif



public static void SelectGenerators (HashSet<Generator> selection, bool shiftForSingleSelect=false)
/// Using cell lut as a list of all generators
/// shiftForSingleSelect selects sinhgle gens only when shift is pressed. Otherwise selected by click
{
 //one by one
 bool genClicked = false;
 if ((Event.current.shift || !shiftForSingleSelect)  &&  Event.current.type == EventType.MouseDown  &&  Ev
 {
  //returning if clicked to already selected generator (to move several generators it in mini)
  if (!shiftForSingleSelect)
  {
   foreach (Cell cell in UI.current.cellObjs.GetAllCells("Generator"))
    if (cell.Contains(UI.current.mousePos))
    {
```

```
    Generator gen = UI.current.cellObjs.GetObject<Generator>(cell, "Generator");
    if (selection.Contains(gen))
     return;
   }
  }

 //selecting
 foreach (Cell cell in UI.current.cellObjs.GetAllCells("Generator"))
  if (cell.Contains(UI.current.mousePos))
  {
   Generator gen = UI.current.cellObjs.GetObject<Generator>(cell, "Generator");

   //if (selection.Contains(gen)) selection.Remove(gen);
   //else selection.Add(gen);
   selection.Clear();
   selection.Add(gen);

   genClicked = true;
   break;
  }
}

//selection frame
if (!genClicked)
{
 if (DragDrop.TryDrag("NodeSelectionFrame"))
```

```
{
 GUIStyle frameStyle = UI.current.textures.GetElementStyle("DPUI/Backgrounds/SelectionFrame");

 Rect rect = new Rect(DragDrop.initialMousePos, UI.current.mousePos-DragDrop.initialMousePos);
 rect = rect.TurnNonNegative();

 Draw.Element(rect, frameStyle);
}

if (DragDrop.TryRelease("NodeSelectionFrame")  &&  !UI.current.layout)
{
 selection.Clear();

 Rect frameRect = new Rect(DragDrop.initialMousePos, UI.current.mousePos-DragDrop.initialMousePo
 frameRect = frameRect.TurnNonNegative();

 foreach (Cell cell in UI.current.cellObjs.GetAllCells("Generator")) //(var kvp in genCellLut.d1)
 {
  Generator gen = UI.current.cellObjs.GetObject<Generator>(cell, "Generator");

  Rect genRect = new Rect(gen.guiPosition, cell.finalSize);
  if (frameRect.Contains(genRect))
   { if (!selection.Contains(gen)) selection.Add(gen); }
  //else
  // { if (selection.Contains(gen)) selection.Remove(gen); }
  //clearing selection anyways
```

```
    }

    genClicked = true;
   }

   if (Event.current.shift  &&  Event.current.type == EventType.MouseDown   &&  Event.current.button == 0
    DragDrop.ForceStart("NodeSelectionFrame", UI.current.mousePos, new Rect(0,0,0,0));
  }

  if (genClicked)
   UI.RemoveFocusOnControl();
 }

 public static void DeselectGenerators (HashSet<Generator> selection)
 {
  //if non-shift clicked any other place rather than selected node - deselecting
  if (!UI.current.layout  &&
   selection.Count != 0  &&
   Event.current.type == EventType.MouseDown  &&
   Event.current.button == 0  &&
   !Event.current.shift)
   {
    //deselecting if dragging anything else but generator
    if (DragDrop.obj != null  &&  DragDrop.obj is Cell cell  &&  UI.current.cellObjs.ContainsCell(cell, "Genera
     return;
```

```
    selection.Clear();
    UI.RemoveFocusOnControl();
    UI.current.editorWindow?.Repaint();
  }
}
```

```
public static void DrawInlet (IInlet<object> inlet, Generator gen, float scale=1, bool addCellObj=true)
/// Drawing in a 0-pixel of a cell
{
 Cell inletCell;

 Graph graph = GraphWindow.current.graph;

 Color color = GetLinkColor(inlet);

 //drawing
 using (inletCell = Cell.Center(0, 0))
 {
  if (UI.current.layout  &&  addCellObj) //adding on layout to draw links before generators
   UI.current.cellObjs.ForceAdd(inlet, Cell.current, "Inlet");

  if (!UI.current.layout)
  {
   IOutlet<object> outlet = graph.GetLink(inlet);
```

```
  Texture2D icon = UI.current.textures.GetColorizedTexture("MapMagic/InletOutlet", color);
  Draw.Icon(icon, scale:0.5f*scale);
 }
}

//dragging
if (!UI.current.layout)
{
 IOutlet<object> outlet = null;

 if (DragDrop.TryDrag(inletCell, UI.current.mousePos))
 {
  Cell outletCell = null;

  foreach (Cell cell in UI.current.cellObjs.GetAllCells("Outlet"))
   if (cell.Contains(UI.current.mousePos, (int)(inletOutletDragArea/2 * scale))) //outlet cell size is 0, using
   {
    outletCell = cell;
    outlet = UI.current.cellObjs.GetObject<IOutlet<object>>(cell, "Outlet");
   }


  color.a = 0.5f;

  if (outlet != null)
  {
```

```
  if (!graph.CheckLinkValidity(outlet, inlet)) color = Color.red;
  DrawLink(StartCellLinkpos(outletCell), EndCellLinkpos(inletCell), color);
 }
 else
  DrawLink(UI.current.mousePos, EndCellLinkpos(inletCell), color);
}

if (DragDrop.TryRelease(inletCell, UI.current.mousePos))
{
 GraphWindow.RecordCompleteUndo();

 if (outlet == null)
 {
  graph.UnlinkInlet(inlet);
  GraphEditorActions.RemoveLink(graph, inlet);
 }
 else if (graph.CheckLinkValidity(outlet,inlet))
  graph.Link(inlet, outlet);

 GraphWindow.current?.RefreshMapMagic();
}

Rect inletRect = new Rect(
 inletCell.worldPosition.x - inletOutletDragArea/2*scale,
 inletCell.worldPosition.y - inletOutletDragArea/2*scale,
 inletCell.finalSize.x + inletOutletDragArea*scale,
```

```
   inletCell.finalSize.x + inletOutletDragArea*scale);
  DragDrop.TryStart(inletCell, UI.current.mousePos, inletRect);
 }
}



public static void DrawOutlet (IOutlet<object> outlet, float scale=1, bool addCellObj=true)
/// Requires 0-width cell, draws outlet in the center
{
 //if (UI.current.layout) return;
 //no optimize!

 Graph graph = GraphWindow.current.graph;

 Color color = GetLinkColor(outlet);

 //drawing
 Cell outletCell;
 using (outletCell = Cell.Center(0,0))
 {
  if (UI.current.layout && addCellObj)
   UI.current.cellObjs.ForceAdd(outlet, Cell.current, "Outlet");

  if (!UI.current.layout)
  {
   Texture2D icon = UI.current.textures.GetColorizedTexture("MapMagic/InletOutlet", color);
```

```
  Draw.Icon(icon, scale:0.5f*scale);
 }
}

//dragging
if (!UI.current.layout)
{
 IInlet<object> inlet = null;

 if (DragDrop.TryDrag(outletCell, UI.current.mousePos))
 {
  Cell inletCell = null;

  foreach (Cell cell in UI.current.cellObjs.GetAllCells("Inlet"))
   if (cell.Contains(UI.current.mousePos, (int)(inletOutletDragArea/2 * scale))) //inlet cell size is 0, using 1
   {
    inlet = UI.current.cellObjs.GetObject<IInlet<object>>(cell, "Inlet");
    inletCell = cell;
   }

  color.a = 0.5f;

  if (inlet != null)
  {
   if (!graph.CheckLinkValidity(outlet,inlet)) color = Color.red;
   DrawLink(StartCellLinkpos(outletCell), EndCellLinkpos(inletCell), color);
```

```
    }
    else DrawLink(StartCellLinkpos(outletCell), UI.current.mousePos, color);
   }

   if (DragDrop.TryRelease(outletCell, UI.current.mousePos))
   {
    GraphWindow.RecordCompleteUndo();

    if (inlet != null  &&  graph.CheckLinkValidity(outlet,inlet))
     graph.Link(inlet, outlet);

    GraphWindow.current?.RefreshMapMagic();
   }

   Rect outletRect = new Rect(
    outletCell.worldPosition.x - inletOutletDragArea/2*scale,
    outletCell.worldPosition.y - inletOutletDragArea/2*scale,
    outletCell.finalSize.x + inletOutletDragArea*scale,
    outletCell.finalSize.x + inletOutletDragArea*scale);
   DragDrop.TryStart(outletCell, UI.current.mousePos, outletRect);
  }
 }
```

```
public static void DrawPortal (Generator gen, Graph graph, bool selected=false, bool isMini=false, string t
{
 IPortalEnter<object> portalEnter = gen as IPortalEnter<object>;
 IPortalExit<object> portalExit = gen as IPortalExit<object>;
 IFnEnter<object> functionInput = gen as IFnEnter<object>;
 IFnExit<object> functionOutput = gen as IFnExit<object>;

 if (UI.current.layout)
  UI.current.cellObjs.ForceAdd(gen, Cell.current, "Generator");

 //background
 if (!UI.current.layout)
 {
  //shadow
  #if !MM_DOC
  GUIStyle shadowStyle = UI.current.textures.GetElementStyle(selected ? "MapMagic/Node/SelectionSha
   borders:shadowBorders,
   overflow:shadowOverflow);
  Draw.Element(shadowStyle);
  #endif

  //frame
  GUIStyle frameStyle = UI.current.textures.GetElementStyle(selected ? "MapMagic/Node/SelectionFrame
   borders:selected ? selectionBorders : frameBorders,
   overflow:selected ? selectionBorders : frameBorders);
  Draw.Element(frameStyle);
```

```
 //gray field color (background to all node, including the header)
 GUIStyle fieldBackStyle = UI.current.textures.GetElementStyle("MapMagic/Node/Background");
 Draw.Element(fieldBackStyle);
}

//header (and field)
Color color = GeneratorDraw.GetGeneratorColor(gen);
//Texture2D headerTex = UI.current.textures.GetColorizedTexture("MapMagic/Node/Header", color);
//GUIStyle headBackStyle = UI.current.textures.GetElementStyle(headerTex, borders:new RectOffset(40
//Draw.Element(headBackStyle);
Texture2D headerTex = UI.current.textures.GetColorizedTexture("MapMagic/Node/Header", color);
Draw.Texture(headerTex);

using (Cell.LinePx(portalHeight))
{
 float inletScale = isMini ? miniInletsScale : 1;
 float nodeHeight = 24; if (isMini) nodeHeight /= GraphWindow.miniZoom;
 float iconScale = 0.5f; if (isMini) iconScale /= GraphWindow.miniZoom;
 float iconSize = 20; if (isMini) iconSize /= GraphWindow.miniZoom;
 float offsetSize = 8; if (isMini) offsetSize /= GraphWindow.miniZoom;

 Cell.EmptyLineRel(1);
 using (Cell.LinePx(nodeHeight))
 {
   if (portalEnter != null)
```

```
{
 //inlet
 using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(portalEnter, gen, scale:inletScale);

 //icon
 Cell.EmptyRowPx(offsetSize);
 Texture2D genIcon = UI.current.textures.GetTexture("GeneratorIcons/PortalIn");
 using (Cell.RowPx(iconSize)) Draw.Icon(genIcon, scale:iconScale);

 //label
 if (!isMini)
  using (Cell.Row)
 {
  Color prevSelectionColor = UnityEngine.GUI.skin.settings.selectionColor;
  UnityEngine.GUI.skin.settings.selectionColor = new Color(0, 0.3555f, 0.78125f);
  portalEnter.Name = Draw.EditableLabel(portalEnter.Name, style:UI.current.styles.bigLabel);
  UnityEngine.GUI.skin.settings.selectionColor = prevSelectionColor;
 }

 Cell.EmptyRowPx(5);
}

else if (portalExit != null)
{
 //icon
 Cell.EmptyRowPx(offsetSize/2);
```

```
 Texture2D genIcon = UI.current.textures.GetTexture("GeneratorIcons/PortalOut");
 using (Cell.RowPx(iconSize)) Draw.Icon(genIcon, scale:iconScale);
 //UI.Empty(Size.RowPixels(5));

 //label
 using (Cell.Row) //cell or empty cell if mini
  if (!isMini)

 {
  string label = "(Empty)";
  if (portalExit.Enter != null)
   label = portalExit.Enter.Name;
  Draw.Label(label, style:UI.current.styles.bigLabel);
 }

 Texture2D chevronDown = UI.current.textures.GetTexture("DPUI/Chevrons/Down");
 using (Cell.RowPx(iconSize))
  if (Draw.Button(null, icon:chevronDown, iconScale:iconScale, visible:false))
   PortalSelectorPopup.DrawPortalSelector(graph, portalExit);

 Cell.EmptyRowPx(10);

 using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet((IOutlet<object>)gen, scale:inletScale);
}

else if (functionInput != null)
```

```
{
 //icon
 Cell.EmptyRowPx(4);
 Texture2D genIcon = UI.current.textures.GetTexture("GeneratorIcons/FunctionIn");
 using (Cell.RowPx(iconSize)) Draw.Icon(genIcon, scale:iconScale);
 //UI.Empty(Size.RowPixels(5));

 //label
 using (Cell.Row)
  if (!isMini)
 {
  Color prevSelectionColor = UnityEngine.GUI.skin.settings.selectionColor;
  UnityEngine.GUI.skin.settings.selectionColor = new Color(0, 0.3555f, 0.78125f);
  functionInput.Name = Draw.EditableLabel(functionInput.Name, style:UI.current.styles.bigLabel);
  UnityEngine.GUI.skin.settings.selectionColor = prevSelectionColor;
 }

 Cell.EmptyRowPx(10);

 using (Cell.RowPx(0)) GeneratorDraw.DrawOutlet((IOutlet<object>)gen, scale:inletScale);
}

else if (functionOutput != null)
{
 //inlet
 using (Cell.RowPx(0)) GeneratorDraw.DrawInlet(functionOutput, gen, scale:inletScale);
```

```
    //icon
    Cell.EmptyRowPx(offsetSize);
    Texture2D genIcon = UI.current.textures.GetTexture("GeneratorIcons/FunctionOut");
    using (Cell.RowPx(iconSize)) Draw.Icon(genIcon, scale:iconScale);
    //UI.Empty(Size.RowPixels(5));

    //label
    using (Cell.Row)
     if (!isMini)
    {
     Color prevSelectionColor = UnityEngine.GUI.skin.settings.selectionColor;
     UnityEngine.GUI.skin.settings.selectionColor = new Color(0, 0.3555f, 0.78125f);
     functionOutput.Name = Draw.EditableLabel(functionOutput.Name, style:UI.current.styles.bigLabel);
     UnityEngine.GUI.skin.settings.selectionColor = prevSelectionColor;
    }

    Cell.EmptyRowPx(5);
   }
 }
 Cell.EmptyLineRel(1);
}

#if MM_DEBUG
//if (graph.debugGenInfo)
 DrawDebug(gen);
```

```
 #endif

 if (!UI.current.layout)
 {
  gen.guiSize.y = Cell.current.finalSize.y;
  gen.guiSize.x = nodeWidth;
 }
}

public static T DrawGlobalVar<T> (T val, string label)
{
 using (Cell.RowRel(1-Cell.current.fieldWidth))
 {
  Cell.EmptyRowPx(3);
  using (Cell.RowPx(9)) Draw.Icon(UI.current.textures.GetTexture("DPUI/Icons/Linked"));
  using (Cell.Row) Draw.Label(label);

  if (val != null  &&  val is float)
   val = (T)(object)Draw.DragValue((float)(object)val);
  if (val != null  &&  val is int)
   val = (T)(object)Draw.DragValue((int)(object)val);
 }

 using (Cell.RowRel(Cell.current.fieldWidth))
  val = (T)Draw.UniversalField(val, typeof(T));
```

```
 return val;
}
public static void DrawGlobalVar<T> (ref T val, string label) => val = DrawGlobalVar<T>(val, label);



public static void SubGraph (IBiome biome, ref Graph subGraph, bool refreshOnGraphChange=true)
{
 using (Cell.Row)
 {
  Graph prevSubGraph = subGraph;
  Draw.ObjectField(ref subGraph);

  if (Cell.current.valChanged)
  {
   if (SubGraphDependenceError(biome, subGraph))
   {
    Debug.Log($"Graph {subGraph} contains this biome (directly or in sub-graphs). Could not be assigned
    subGraph = prevSubGraph;
    return;
   }

   //if (biome is Biomes.BaseFunctionGenerator fn)
   // if (biome.SubGraph != null)
   //  fn.ovd = new Expose.Override(subGraph.defaults);
   //requires an access to module. Moved to fn editor
```

```
   if (refreshOnGraphChange)
    GraphWindow.current?.RefreshMapMagic();
  }
 }

 Texture2D openIcon = UI.current.textures.GetTexture("DPUI/Icons/FolderOpen");
  using (Cell.RowPx(22))
   if (Draw.Button(icon:openIcon, iconScale:0.5f, visible:false) && subGraph!=null)
   {
    Graph subGraphClosure = subGraph;
    UI.current.DrawAfter( ()=> GraphWindow.current.OpenBiome(subGraphClosure) );
   }
}

private static bool SubGraphDependenceError (IBiome biome, Graph subGraph)
/// True if subGraph contains biome (even in sub-sub grpahs)
{
 if (subGraph == null)
  return false;

 Generator gen = ((IUnit)biome).Gen;

 if (subGraph.ContainsGenerator(gen))
  return true;

 foreach (Graph subSubGraph in subGraph.SubGraphs(recursively:true))
```

```
  if (subSubGraph.ContainsGenerator(gen))
   return true;

 return false;
}



public static void DrawMini (Generator gen, Graph graph, bool selected=false, GeneratorMenuAttribute m
{
 if (UI.current.layout)
  UI.current.cellObjs.ForceAdd(gen, Cell.current, "Generator");

 if (menuAtt == null)
 {
  Type genType = gen.GetType();
  menuAtt = GetMenuAttribute(genType);
 }

 //background
 if (!UI.current.layout)
 {
  //shadow
  GUIStyle shadowStyle = UI.current.textures.GetElementStyle(selected ? "MapMagic/Node/SelectionSha
   borders:shadowBorders,
   overflow:shadowOverflow);
  #if !MM_DEBUG
```

```
 Draw.Element(shadowStyle);
#else
 if (graph.debugGraphBackground)
  Draw.Element(shadowStyle);
#endif

//frame
GUIStyle frameStyle = UI.current.textures.GetElementStyle(selected ? "MapMagic/Node/SelectionFrame
 borders:selected ? selectionBorders : frameBorders,
 overflow:selected ? selectionBorders : frameBorders);
Draw.Element(frameStyle);

//gray field color (borders in this case)
GUIStyle fieldBackStyle = UI.current.textures.GetElementStyle("MapMagic/Node/Background");
Draw.Element(fieldBackStyle);

//header color
Color color = menuAtt.color;
if (!gen.enabled) color = DisableGeneratorColor(color);
Texture2D headerTex = UI.current.textures.GetColorizedTexture("MapMagic/Node/Header", color);
//GUIStyle headBackStyle = UI.current.textures.GetElementStyle(headerTex);//, borders:new RectOffset
Draw.Texture(headerTex);
}

Cell.EmptyLinePx(4/GraphWindow.miniZoom);
```

```
//icon/single inlets/single outlets
using (Cell.LinePx(0))
{
 if (gen is IInlet<object> inletGen)
 {
  using (Cell.RowPx(0))
   DrawInlet(inletGen, gen, miniInletsScale);
 }

 using (Cell.Row)
  using (Cell.LinePx(16f/GraphWindow.miniZoom)) Draw.Icon(menuAtt.icon, scale:0.5f/GraphWindow.mi

 if (gen is IOutlet<object> outletGen)
 {
  using (Cell.RowPx(0))
   DrawOutlet(outletGen, miniInletsScale);
 }
}

//name
using (Cell.LinePx(16/GraphWindow.miniZoom))
{
 if (miniNameStyle == null)
 {
  miniNameStyle = new GUIStyle(UI.current.styles.bigLabel);
  miniNameStyle.fontSize = (int)(miniNameStyle.fontSize * 0.6f / GraphWindow.miniZoom);
```

```
  miniNameStyle.alignment = TextAnchor.LowerCenter;
 }

 Draw.Label(menuAtt.nameUpper, style:miniNameStyle);
}

//layers
if (miniInletStyle == null)
{
 miniInletStyle = new GUIStyle(UI.current.styles.label);
 miniInletStyle.fontSize = (int)(miniInletStyle.fontSize*0.8f / GraphWindow.miniZoom);
}

if (gen is IMultiLayer mlayGen)
{
 IList<IUnit> layers = mlayGen.Layers;
 int layersCount = layers.Count;

 //draing inlets that are not layers for Whittaker
 if (gen is IMultiInlet multiInGen)
  foreach (IInlet<object> inlet in multiInGen.Inlets())
   if (!(inlet is IOutlet<object>))
    using (Cell.LinePx(miniInletLineHeight))
     using (Cell.RowPx(0))
      using (Cell.LinePx(miniInletLineHeight)) DrawInlet(inlet, gen, miniInletsScale);
```

```
//drawing layers
for (int i=0; i<layersCount; i++)
{
 int ii = !mlayGen.Inversed ?  i  :  layersCount-1 - i;
 IUnit layer = layers[ii];

 if (layer is IInlet<object>  ||  layer is IOutlet<object>)
  using (Cell.LinePx(miniInletLineHeight))
  {
   if (layer is IInlet<object> inlet)
    if (!(ii==0 && mlayGen.HideFirst))
     using (Cell.RowPx(0)) DrawInlet(inlet, gen, miniInletsScale);

   Cell.EmptyRowPx(14);

   string name = null;
   if (layer is IBiome biome  &&  biome.SubGraph != null) name = biome.SubGraph.name;
   if (layer is MatrixGenerators.BaseTextureLayer texLayer) name = texLayer.name;

   using (Cell.Row)
   {
    if (name != null)
     Draw.Label(name, miniInletStyle);
   }

   Cell.EmptyRowPx(14);
```

```
    if (layer is IOutlet<object> outlet)
     using (Cell.RowPx(0)) DrawOutlet(outlet, miniInletsScale);
   }
  }
 }

//multi inlets/outlets
else if (gen is IMultiInlet  ||  gen is IMultiOutlet)
 using (Cell.LinePx(0))
 {
  //inlets
  using (Cell.RowPx(0))
   if (gen is IMultiInlet multiInGen)
   {
    foreach (IInlet<object> inlet in multiInGen.Inlets())
     using (Cell.LinePx(miniInletLineHeight)) DrawInlet(inlet, gen, miniInletsScale);
   }

  Cell.EmptyRowPx(14);

  //names
  using (Cell.Row)
   if (gen is IMultiInlet multiInGen)
   {
    ValAttribute[] inletVals = GetInletVals(menuAtt.type);
```

```
      foreach (ValAttribute val in inletVals)
        using (Cell.LinePx(miniInletLineHeight)) Draw.Label(val.name, miniInletStyle);
      }

    Cell.EmptyRowPx(16);

    //outlets
    using (Cell.RowPx(0))
     if (gen is IMultiOutlet multiOutGen)
     {
      foreach (IOutlet<object> outlet in multiOutGen.Outlets())
        using (Cell.LinePx(miniInletLineHeight)) DrawOutlet(outlet, miniInletsScale);
     }
   }

  //custom header editor
  using (Cell.LinePx(0))
   Draw.Editor(gen, cat:"Header");
}




#region Links
```

```
public static void DrawLink (Vector2 startPos, Vector2 endPos, Color color, bool dotted=false, int density=
{
 if (UI.current.layout) return;
 //if (UI.current.optimizeElements &&
 // (end.x<Cell.current.worldPosition.x || end.y<Cell.current.worldPosition.y))// ||
  //start.x>Cell.current.worldPosition.x+Cell.current.finalSize.x || start.y>Cell.current.worldPosition.y+Cell.
 //  return;



 float distance = (endPos-startPos).magnitude;

 Vector2 startTan = new Vector2(startPos.x + distance/4, startPos.y);
 Vector2 endTan = new Vector2(endPos.x-distance/4, endPos.y);



 //Mathf.Min(startPos, endPos, startTangent, endTangent)
 float minX = startPos.x<endPos.x ? startPos.x : endPos.x;  minX=minX<startTan.x ? minX : startTan.x;
 float maxX = startPos.x>endPos.x ? startPos.x : endPos.x;  maxX=maxX>startTan.x ? maxX : startTan.x
 float minY = startPos.y<endPos.y ? startPos.y : endPos.y;  minY=minY<startTan.y ? minY : startTan.y;
 float maxY = startPos.y>endPos.y ? startPos.y : endPos.y;  maxY=maxY>startTan.y ? maxY : startTan.y

 if (UI.current.optimizeElements && !UI.current.IsInWindow(minX, maxX, minY, maxY)) return;

 startPos = UI.current.scrollZoom.ToScreen(startPos);
```

```
endPos = UI.current.scrollZoom.ToScreen(endPos);
startTan = UI.current.scrollZoom.ToScreen(startTan);
endTan = UI.current.scrollZoom.ToScreen(endTan);

Texture2D splineTex = UI.current.textures.GetTexture("DPUI/SplineTex");

float curWidth = width*UI.current.scrollZoom.zoom+2f;

//20 skin
if (!dotted)
 #if !MM_DOC
 UnityEditor.Handles.DrawBezier(startPos, endPos, startTan, endTan, color, splineTex, curWidth);
 #else
 UnityEditor.Handles.DrawBezier(startPos, endPos, startTan, endTan, color, UI.current.textures.GetText
 #endif

//manual spline
else
{
 Vector3[] splinePoints = new Vector3[2];

 int steps = (int)(distance / 10);

 for (int i=0; i<steps; i+=2)
 {
  float p = 1f*i/steps;
```

```
    float ip = 1f-p;
    splinePoints[0] = ip*ip*ip*startPos + 3*p*ip*ip*startTan + 3*p*p*ip*endTan + p*p*p*endPos;

    p = 1f*(i+1)/steps;
    ip = 1f-p;
    splinePoints[1] = ip*ip*ip*startPos + 3*p*ip*ip*startTan + 3*p*p*ip*endTan + p*p*p*endPos;

    UnityEditor.Handles.color = color;
    UnityEditor.Handles.DrawAAPolyLine(splineTex, curWidth, splinePoints);
   }
  }
 }

public static Vector2 StartCellLinkpos (Cell cell) { return cell.InternalCenter+new Vector2(5,0); }
public static Vector2 EndCellLinkpos (Cell cell) { return cell.InternalCenter+new Vector2(-5,0); }

public static float DistToLink (Vector2 pos, IOutlet<object> outlet, IInlet<object> inlet)
{
 Cell outletCell = UI.current.cellObjs.GetCell(outlet, "Outlet");
 Cell inletCell = UI.current.cellObjs.GetCell(inlet, "Inlet");
 if (outletCell==null || inletCell==null)
  return float.MaxValue;

 Vector2 linkStart = StartCellLinkpos(outletCell);
 Vector2 linkEnd = EndCellLinkpos(inletCell);
```

```
 return DistToLink(pos, linkStart, linkEnd);
}



public static float DistToLink (Vector2 pos, Vector2 startPos, Vector2 endPos)
{
 float distance = (endPos-startPos).magnitude;

 Vector2 startTan = new Vector2(startPos.x + distance/4, startPos.y);
 Vector2 endTan = new Vector2(endPos.x-distance/4, endPos.y);

 return UnityEditor.HandleUtility.DistancePointBezier(pos, startPos, endPos, startTan, endTan);
}

/*public static void MinDistToLink (Vector2 pos, out float minDist, out IInlet<object> minInlet)
{
 minDist = int.MaxValue;
 minInlet = null;
 foreach (var kvp in linkLinesLut.d1)
 {
  (Vector2,Vector2) link = kvp.Key;
  float dist = DistToLink(pos, link.Item1, link.Item2);
  if (dist < minDist)
  {
   minDist = dist;
   minInlet = kvp.Value;
```

```
  }
 }
}*/




#endregion



#region Helpers

 private static readonly Dictionary<Type,GeneratorMenuAttribute> cachedAttributes = new Dictionary<Typ

 public static GeneratorMenuAttribute GetMenuAttribute (Type genType)
 {
  if (cachedAttributes.TryGetValue(genType, out GeneratorMenuAttribute att)) return att;

  Attribute[] atts = Attribute.GetCustomAttributes(genType);

  GeneratorMenuAttribute menuAtt = null;
  for (int i=0; i<atts.Length; i++)
   if (atts[i] is GeneratorMenuAttribute) menuAtt = (GeneratorMenuAttribute)atts[i];

  if (menuAtt != null)
  {
```

```
  menuAtt.nameUpper = menuAtt.name.ToUpper();
  menuAtt.nameWidth = UI.current.styles.bigLabel.CalcSize( new GUIContent(menuAtt.nameUpper) ).x;
  menuAtt.icon = UI.current.textures.GetTexture(menuAtt.iconName ?? "GeneratorIcons/Generator");
  menuAtt.type = genType;
  menuAtt.color = menuAtt.colorType != null ?
   GetGeneratorColor( menuAtt.colorType) :
   GetGeneratorColor( Generator.GetGenericType(genType) );
 }
 //else
 // throw new Exception("Could not find attribute for " + genType.ToString());
 // there are still some generators without attribute. Like "MyGenerator"

 cachedAttributes.Add(genType, menuAtt);
 return menuAtt;
}



private static readonly Dictionary<Type,ValAttribute[]> cachedInletVals = new Dictionary<Type, ValAttribu

public static ValAttribute[] GetInletVals (Type genType)
{
 if (cachedInletVals.TryGetValue(genType, out ValAttribute[] inletValsArr)) return inletValsArr;

 else
 {
  List<ValAttribute> inletVals = new List<ValAttribute>();
```

```
  ValAttribute[] allAttributes = ValAttribute.GetAttributes(genType);
  for (int v=0; v<allAttributes.Length; v++)
   if (allAttributes[v].type != null  &&  (typeof(IInlet<object>)).IsAssignableFrom(allAttributes[v].type)) //if su
    inletVals.Add(allAttributes[v]);

  inletValsArr = inletVals.ToArray();
  cachedInletVals.Add(genType, inletValsArr);
  return inletValsArr;
 }
}


private static readonly Dictionary<Type,ValAttribute[]> cachedOutletVals = new Dictionary<Type, ValAttri

public static ValAttribute[] GetOutletVals (Type genType)
{
 if (cachedOutletVals.TryGetValue(genType, out ValAttribute[] outletValsArr)) return outletValsArr;

 else
 {
  List<ValAttribute> outletVals = new List<ValAttribute>();

  ValAttribute[] allAttributes = ValAttribute.GetAttributes(genType);
  for (int v=0; v<allAttributes.Length; v++)
   if (allAttributes[v].type != null  &&  (typeof(IOutlet<object>)).IsAssignableFrom(allAttributes[v].type)) //if s
```

```
    outletVals.Add(allAttributes[v]);


  outletValsArr = outletVals.ToArray();
  cachedOutletVals.Add(genType, outletValsArr);
  return outletValsArr;
 }
}



public static Color GetGeneratorColor (Generator gen) => GetGeneratorColor(Generator.GetGenericTyp
public static Color GetGeneratorColor (Type genericType)
{
 if (StylesCache.isPro) return GetGeneratorColorPro(genericType);

 Color color;

 if (genericType == typeof(MatrixWorld)) color = new Color(0.4f, 0.666f, 1f, 1f);
 else if (genericType == typeof(TransitionsList)) color = new Color(0.444f, 0.871f, 0.382f, 1f);
 else if (genericType == typeof(SplineSys)) color = new Color(1f, 0.6f, 0f, 1f);
 else if (genericType == typeof(Den.Tools.Splines.SplineSys)) color = new Color(1f, 0.6f, 0f, 1f);
 else if (genericType == typeof(IBiome)) color = new Color(0.45f, 0.55f, 0.56f, 1f);
 else if (genericType == typeof(MatrixSet)) color = new Color(0.3f, 0.444f, 0.75f, 1f);
 else color = new Color(0.65f, 0.65f, 0.65f, 1);

 return color;
}
```

```
private static Color GetGeneratorColorPro (Type genericType)
{
 Color color;

 if (genericType == typeof(MatrixWorld)) color = new Color(0, 0.325f, 0.75f, 0.7f);
 else if (genericType == typeof(TransitionsList)) color = new Color(0.13f, 0.65f, 0, 0.65f);
 else if (genericType == typeof(SplineSys)) color = new Color(0.65f, 0.325f, 0f, 0.7f);
 else if (genericType == typeof(Den.Tools.Splines.SplineSys)) color = new Color(0.65f, 0.325f, 0f, 0.7f);
 else if (genericType == typeof(IBiome)) color = new Color(0.45f, 0.55f, 0.56f, 1f);
 else if (genericType == typeof(MatrixSet)) color = new Color(0, 0.2f, 0.5f, 0.7f);
 else color = new Color(0.65f, 0.65f, 0.65f, 1);

 return color;
}

private static Color DisableGeneratorColor (Color color)
//returns the disabled color from original generator's color
{
 return Color.Lerp(color, new Color(0.8f, 0.8f, 0.8f, 1), 0.75f);
}

public static Color GetLinkColor (IInlet<object> inlet) => GetLinkColor(Generator.GetGenericType(inlet));
public static Color GetLinkColor (IOutlet<object> outlet) => GetLinkColor(Generator.GetGenericType(out
public static Color GetLinkColor (Type genericType)
{
```

```
bool isPro = StylesCache.isPro;

if (genericType == typeof(MatrixWorld))
{
 if (isPro) return new Color(0, 0.5f, 1f, 1f);
 else return new Color(0, 0.333f, 0.666f, 1f);
}

if (genericType == typeof(TransitionsList))
{
 if (isPro) return new Color(0.2f, 1f, 0, 1f);
 else return new Color(0, 0.465f, 0, 1f);
}

if (genericType == typeof(SplineSys))
{
 if (isPro) return new Color(1f, 0.5f, 0f, 1f);
 else return new Color(0.666f, 0.333f, 0, 1f);
}

if (genericType == typeof(Den.Tools.Splines.SplineSys))
{
 if (isPro) return new Color(1f, 0.5f, 0f, 1f);
 else return new Color(0.666f, 0.333f, 0, 1f);
}
```

```
 if (genericType == typeof(MatrixSet))
 {
  if (isPro) return new Color(0, 0.25f, 0.5f, 1f);
  else return new Color(0, 0.16f, 0.333f, 1f);
 }

 return Color.gray;
}

#endregion



#region AutoPositioning

 public static bool FindPlace (ref Vector2 original, Vector2 range, Graph graph, int step=10, int margins=5
 /// Tries to find the position that is not intersecting with other nodes. Return true if found (and changes ref
 {
  //creating a list of generators within range
  List<Rect> gensInRange = new List<Rect>();
  Rect rangeRect = new Rect(
   original.x-range.x-nodeWidth/2-margins,
   original.y-range.y-nodeWidth/4-margins,
   range.x*2+nodeWidth+margins*2,
   range.y*2+nodeWidth/2+margins*2);
  for (int g=0; g<graph.generators.Length; g++)
  {
```

```
 if (!rangeRect.Intersects(graph.generators[g].guiPosition, graph.generators[g].guiSize)) continue;
 gensInRange.Add(new Rect(graph.generators[g].guiPosition, graph.generators[g].guiSize));
}
int gensInRangeCount = gensInRange.Count;

Vector2 defSize = new Vector2(nodeWidth + margins*2, nodeWidth/2 + margins*2);

//finding number of steps
int stepsX = (int)(range.x/step) + 1;
int stepsY = (int)(range.y/step) + 1;
int stepsMax = Mathf.Max(stepsX, stepsY);

//find the closest possible place (starting from center)
Coord center = new Coord(0,0);
foreach (Coord coord in center.DistanceArea(stepsMax))
{
 int x = coord.x;
 int y = coord.z;

 if (x>stepsX || x<-stepsX || y>stepsY || y<-stepsY) continue;

 Vector2 newPos = new Vector2(original.x + x*step - margins, original.y + y*step - margins);
 //Vector2 pos = new Vector2(original.x + x*step, original.y + y*step);

 bool intersects = false;
 for (int g=0; g<gensInRangeCount; g++)
```

```
  {
   if (gensInRange[g].Intersects(newPos, defSize))
    { intersects = true; break; }
  }

  if (!intersects)
  {
   //using (Cell.Custom( new Rect(pos, defSize)))
   // Draw.Rect(Color.red);

   original = newPos + new Vector2(margins,margins);
   return true;
  }
 }

 return false;
}

public static void RelaxIteration (Generator gen, List<Generator> nearGens, List<IOutlet<object>> inletG
{

}

internal static void TestRelax (Graph graph)
```

```
  {
   Generator gen = null;
   for (int g=0; g<graph.generators.Length; g++)
    if ( new Rect(graph.generators[g].guiPosition, graph.generators[g].guiSize).Contains(UI.current.mouseP
     { gen = graph.generators[g]; break; }

   if (gen != null)
   {
    using (Cell.Custom(  new Rect(gen.guiPosition, gen.guiSize) ))
     Draw.Rect(Color.red);
   }
  }


 #endregion
}
}
```

```
ï»¿using System;
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;

using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Products;
using MapMagic.Previews;

namespace MapMagic.Nodes.GUI
{
 public static class GraphEditorActions
 /// Editor implementation (with undo, messages, mouse pos, stuff) of the graph actions
 {
  public static void RemoveGenerators (this Graph graph, HashSet<Generator> selected, Generator clicked
  {
   if (selected==null  ||  selected.Count==0  ||  !selected.Contains(clicked))
   //if clicked not on selected - removing it instead
    selected = new HashSet<Generator>() { clicked };
```

```
 RemoveGenerators(graph, selected);
}


public static void RemoveGenerators (this Graph graph, HashSet<Generator> selected)
{
 CheckAndClearApplied(selected);

 GraphWindow.RecordCompleteUndo();

 foreach (Generator sgen in selected)
 {
  if (graph.ContainsGenerator(sgen))
   graph.Remove(sgen);
 }

 GraphWindow.current.Focus();
 GraphWindow.current.Repaint();

 GraphWindow.current.RefreshMapMagic();
}


public static void EnableDisableGenerators (this Graph graph, HashSet<Generator> selected, Generator
{
```

```
 if (selected==null  ||  selected.Count==0  ||  !selected.Contains(clicked))
 //if clicked not on selected - disabling it instead
  selected = new HashSet<Generator>() { clicked };

 EnableDisableGenerators(graph, selected);
}


public static void EnableDisableGenerators (this Graph graph, HashSet<Generator> selected)
/// Finds an active state and enables/disables nodes
/// If all of the selected generators are disabled - enables them. If any is enabled - disables
{
 bool enable = true; //do wee need to enable or disable gens?
 foreach (Generator sgen in selected)
  if (sgen.enabled)
   { enable = false; break; }

 if (!enable)
  CheckAndClearApplied(selected);

 foreach (Generator sgen in selected)
 {
  sgen.enabled = !sgen.enabled;
  if (sgen.enabled) sgen.version++;
 }
```

```
 GraphWindow.current?.RefreshMapMagic();
 GraphWindow.current.Focus();
 GraphWindow.current.Repaint();
}

public static void RemoveLink(this Graph graph, IInlet<object> inlet)
{
 HashSet<Generator> inletGens = new HashSet<Generator>();
 inletGens.Add(inlet.Gen);
 CheckAndClearApplied(inletGens);
}



private static void CheckAndClearApplied (HashSet<Generator> gens)
/// Checks whether gens contain an output, displays message window to clear it, and clears if selected Ye
{
 bool hasOutput = false;
 foreach (Generator gen in gens)
  if (gen is OutputGenerator)
   { hasOutput=true; break; }

 if (hasOutput  &&
  GraphWindow.current.mapMagic != null  &&
  GraphWindow.current.mapMagic is MapMagicObject mapMagicObject)
  //EditorUtility.DisplayDialog("Clear Applied", "Would you like to clear generated and applied results on te
 {
```

```
  mapMagicObject.ResetTerrains();
 }
}


public static void CreateGenerator (this Graph graph, Type type, Vector2 mousePos)
{
 GraphWindow.RecordCompleteUndo();

 Generator gen = Generator.Create(type);
 gen.guiPosition = new Vector2(mousePos.x-GeneratorDraw.nodeWidth/2, mousePos.y-GeneratorDraw.h
 graph.Add(gen);

 GraphWindow.current.Focus(); //to return focus from right-click menu
 GraphWindow.current.Repaint();

 //GraphWindow.RefreshMapMagic();
 //no need to refresh since added node isn't linked anywhere
}


public static void DuplicateGenerator (this Graph graph, Generator gen, ref HashSet<Generator> selecte
{
 if (selected==null || selected.Count==0)
  DuplicateGenerator(graph, gen);
```

```
 else if (selected.Contains(gen)) //removing selected only if gen is among them
  DuplicateGenerators(graph, ref selected);
}

public static void DuplicateGenerator (this Graph graph, Generator gen)
{
 HashSet<Generator> gens = new HashSet<Generator>();
 gens.Add(gen);
 DuplicateGenerators(graph, ref gens);
}

public static void DuplicateGenerators (this Graph graph, ref HashSet<Generator> gens)
/// Changes the selected hashset
{
 GraphWindow.RecordCompleteUndo();

 if (gens.Count == 0) return;

 Generator[] duplicatedGens = graph.Duplicate(gens);

 //placing under source generators
 Rect selectionRect = new Rect(duplicatedGens[0].guiPosition, Vector2.zero);
 foreach (Generator gen in duplicatedGens)
  selectionRect = selectionRect.Encapsulate( new Rect(gen.guiPosition, gen.guiSize) );
 foreach (Generator gen in duplicatedGens)
  gen.guiPosition.y += selectionRect.size.y + 20;
```

```
    gens.Clear();
    gens.AddRange(duplicatedGens);

    GraphWindow.current.Focus(); //to return focus from right-click menu
    GraphWindow.current.Repaint();

    GraphWindow.current.RefreshMapMagic();
   }
  }
 }
```

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Core.GUI;
using MapMagic.Expose.GUI;

namespace MapMagic.Nodes.GUI
{
 [CustomEditor(typeof(Graph))]
 //[InitializeOnLoad]
 public class GraphInspector : Editor
 {
  Graph graph; //aka target
  UI ui = new UI();

  /*public void OnEnable ()
  {
   SceneView.onSceneGUIDelegate -= DragGraphToScene;
   SceneView.onSceneGUIDelegate += DragGraphToScene;
```

```
}*/

public static HashSet<string> allGraphsGuids;

[RuntimeInitializeOnLoadMethod, UnityEditor.InitializeOnLoadMethod]
static void Subscribe()
{
 #if UNITY_2019_1_OR_NEWER
 SceneView.duringSceneGui -= DragGraphToScene;
 SceneView.duringSceneGui += DragGraphToScene;
 #else
 SceneView.onSceneGUIDelegate -= DragGraphToScene;
 SceneView.onSceneGUIDelegate += DragGraphToScene;
 #endif

 allGraphsGuids = new HashSet<string>(AssetDatabase.FindAssets("t:Graph"));   //compiling all graph gu
 EditorHacks.SubscribeToListIconDrawCallback(DrawListIcon);
 EditorHacks.SubscribeToTreeIconDrawCallback(DrawTreeIcon);

 //SceneView.duringSceneGui -= DrawInSceneView;
 //#if MM_DEBUG
 //SceneView.duringSceneGui += DrawInSceneView;
 //#endif
}

static void DragGraphToScene (SceneView sceneView)
```

```
{
 UnityEngine.Object[] draggedObjs = DragAndDrop.objectReferences;
 if (draggedObjs == null || draggedObjs.Length != 1 || !(draggedObjs[0] is Graph)) return;

 Graph graph = (Graph)draggedObjs[0];

 if (graph != null)
 //if (Event.current.type == EventType.DragUpdated || Event.current.type == EventType.DragPerform)
 //in Unity 2021.2 won't produce events if cursor is Rejected
 {
  DragAndDrop.visualMode = DragAndDropVisualMode.Copy; // show a drag-add icon on the mouse curs
   // won't perform a drag if cursor is Rejected
 }

 if (Event.current.type == EventType.DragPerform)
 {
  DragAndDrop.AcceptDrag();
  MapMagicInspector.CreateMapMagic(graph);
 }
}

/*private static Graph sceneViewGraph = null;
private static GraphInspector sceneViewGraphInspector = null;
private static void DrawInSceneView (SceneView sceneView)
{
 if (sceneViewGraphInspector != null  &&  sceneViewGraph != null  &&  sceneViewGraph.drawInSceneVie
```

```
  sceneViewGraphInspector.OnInspectorGUI();
}*/



public override void  OnInspectorGUI ()
{
 graph = (Graph)target;

 //assign to draw in scene view
 //#if MM_DEBUG
 //if (graph.drawInSceneView)
 // { sceneViewGraph = graph; sceneViewGraphInspector = this; }
 //#endif

 //undo
 if (ui.undo == null) ui.undo = new Den.Tools.GUI.Undo();
 ui.undo.undoObject = graph;
 ui.undo.undoName = "MapMagic Graph Settings";

 //drawing
 ui.Draw(DrawGUI, inInspector:true);
}

public void DrawGUI ()
{
  using (Cell.LinePx(32))
```

```
 Draw.Label("WARNING: Keeping this asset selected in \nInspector can slow down editor GUI performa
Cell.EmptyLinePx(5);

using (Cell.LinePx(24)) if ( Draw.Button("Open Editor"))
 GraphWindow.Show(graph);
using (Cell.LinePx(20)) if ( Draw.Button("Open in New Tab"))
 GraphWindow.ShowInNewTab(graph);

//seed
Cell.EmptyLinePx(5);
using (Cell.LineStd)
{
 int newSeed = Draw.Field(graph.random.Seed, "Seed"); //
 if (newSeed != graph.random.Seed)
 {
  graph.random.Seed = newSeed;
  //Graph.OnChange.Raise(graph);
 }
}

using (Cell.LineStd) Draw.DualLabel("Nodes", graph.generators.Length.ToString());
using (Cell.LineStd) Draw.DualLabel("MapMagic ver", graph.serializedVersion.ToString());
Cell.EmptyLinePx(5);



//global values
```

```
/*using (Cell.LineStd)
 using (new Draw.FoldoutGroup (ref showShared, "Global Values"))
  if (showShared)
 {
  List<string> changedNames = new List<string>();
  List<object> changedVals = new List<object>();

  (Type type, string name)[] typeNames = graph.sharedVals.GetTypeNames();
  for (int i=0; i<typeNames.Length; i++)
   using (Cell.LineStd) GeneratorDraw.DrawGlobalVar(typeNames[i].type, typeNames[i].name);

  if (Cell.current.valChanged)
  {
   GraphWindow.current.mapMagic.ClearAllNodes();
   GraphWindow.current.mapMagic.StartGenerate();
  }
 }*/

//exposed values
using (Cell.LineStd)
 using (new Draw.FoldoutGroup (ref graph.guiShowExposed, "Overridden Variables"))
  if (graph.guiShowExposed)
  {
   using (Cell.LinePx(0))
    OverrideInspector.DrawLayeredOverride(graph);
  }
```

```
  //dependent graphs
  using (Cell.LineStd)
   using (new Draw.FoldoutGroup (ref graph.guiShowDependent, "Dependent Graphs"))
    if (graph.guiShowDependent)
    {
     using (Cell.LinePx(0))
      DrawDependentGraphs(graph);
    }

  //debug
  #if MM_DEBUG
  using (Cell.LineStd)
   using (new Draw.FoldoutGroup (ref graph.guiShowDebug, "Debug"))
    if (graph.guiShowDebug)
    {
     using (Cell.LineStd) Draw.Toggle(ref graph.debugGenerate, "Generate");
     using (Cell.LineStd) Draw.Toggle(ref graph.debugGenInfo, "Info");
     using (Cell.LineStd) Draw.Toggle(ref graph.debugGraphBackground, "Background");
     using (Cell.LineStd) Draw.Field(ref graph.debugGraphBackColor, "Back Color");
     using (Cell.LineStd) Draw.Toggle(ref graph.debugGraphFps, "Graph FPS");
     using (Cell.LineStd) Draw.Toggle(ref graph.drawInSceneView, "Draw In Scene View");
    }
  #endif
}
```

```
private void DrawDependentGraphs (Graph graph)
/// Draws subgraphs recursively
{
 foreach (Graph subGraph in graph.SubGraphs())
 {
  using (Cell.LineStd)
  {
   using (Cell.Row) Draw.Label(subGraph.name);
   using (Cell.RowPx(100)) Draw.ObjectField(subGraph);
  }

  using (Cell.LinePx(0))
  {
   Cell.EmptyRowPx(10);
   DrawDependentGraphs(subGraph);
  }
 }
}



public static void DrawListIcon (Rect iconRect, string guid, bool isListMode)
{
 if (!allGraphsGuids.Contains(guid)) return;
 Texture2D icon = TexturesCache.LoadTextureAtPath("MapMagic/Icons/AssetBig");
 UnityEngine.GUI.DrawTexture(iconRect, icon, ScaleMode.ScaleToFit);
```

```
  }



  public static void DrawTreeIcon (Rect iconRect, string guid)
  {
   if (!allGraphsGuids.Contains(guid)) return;


   if (!BuildPipeline.isBuildingPlayer) //otherwise will log an error during build that cannot find AssetSmall ico
   {
    Texture2D icon = TexturesCache.LoadTextureAtPath("MapMagic/Icons/AssetSmall");
    UnityEngine.GUI.DrawTexture(iconRect, icon, ScaleMode.ScaleToFit);
   }
  }
 }
}
```

```
ï»¿using System;
using System.Reflection;
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Core.GUI;
using MapMagic.Expose.GUI;

namespace MapMagic.Nodes.GUI
{
 public static class GraphTemplates
 {
   //empty graph is created viaCreateAssetMenuAttribute,
   //but unfortunately there's only one attribute per class

   [MenuItem("Assets/Create/MapMagic/Simple Graph", priority = 102)]
   static void MenuCreateMapMagicGraph(MenuCommand menuCommand)
   {
    ProjectWindowUtil.StartNameEditingIfProjectWindowExists(0,
     ScriptableObject.CreateInstance<TmpCallbackReciever>(),
     "MapMagic Graph.asset",
```

```
  TexturesCache.LoadTextureAtPath("MapMagic/Icons/AssetBig"),
  null);
}

class TmpCallbackReciever : UnityEditor.ProjectWindowCallback.EndNameEditAction
{
 public override void Action(int instanceId, string pathName, string resourceFile)
 {
  Graph graph = CreateTemplate();
  graph.name = System.IO.Path.GetFileName(pathName);
  AssetDatabase.CreateAsset(graph, pathName);

  ProjectWindowUtil.ShowCreatedAsset(graph);

  GraphInspector.allGraphsGuids = new HashSet<string>(AssetDatabase.FindAssets("t:Graph"));
 }
}

public static Graph CreateTemplate ()
{
 Graph graph = GraphInspector.CreateInstance<Graph>();

 MatrixGenerators.Noise200 noise = (MatrixGenerators.Noise200)Generator.Create(typeof(MatrixGenera
 graph.Add(noise);
 noise.guiPosition = new Vector2(-270,-100);
```

```
 MatrixGenerators.Erosion200 erosion = (MatrixGenerators.Erosion200)Generator.Create(typeof(MatrixG
 graph.Add(erosion);
 erosion.guiPosition = new Vector2(-70,-100);
 graph.Link(erosion, noise);

 MatrixGenerators.HeightOutput200 output = (MatrixGenerators.HeightOutput200)Generator.Create(type
 graph.Add(output);
 output.guiPosition = new Vector2(130, -100);
 graph.Link(output, erosion);

 return graph;
}



#if MM_DEBUG
[MenuItem("Assets/Create/MapMagic/PerfTest Graph", priority = 102)]
static void MenuCreatePerfTestGraph(MenuCommand menuCommand)
{
 ProjectWindowUtil.StartNameEditingIfProjectWindowExists(0,
  ScriptableObject.CreateInstance<TmpCallbackRecieverBig>(),
  "PefrfTest Graph.asset",
  TexturesCache.LoadTextureAtPath("MapMagic/Icons/AssetBig"),
  null);
}

class TmpCallbackRecieverBig : UnityEditor.ProjectWindowCallback.EndNameEditAction
```

```
{
 public override void Action(int instanceId, string pathName, string resourceFile)
 {
  Graph graph = CreateBig();
  graph.name = System.IO.Path.GetFileName(pathName);
  AssetDatabase.CreateAsset(graph, pathName);

  ProjectWindowUtil.ShowCreatedAsset(graph);

  GraphInspector.allGraphsGuids = new HashSet<string>(AssetDatabase.FindAssets("t:Graph"));
 }
}

public static Graph CreateBig ()
{
 Graph graph = GraphInspector.CreateInstance<Graph>();

 /*for (int j=0; j<10; j++)
 {
  MatrixGenerators.Noise200 noise = (MatrixGenerators.Noise200)Generator.Create(typeof(MatrixGener
  graph.Add(noise);
  noise.guiPosition = new Vector2(-270,-100 + j*200);

  MatrixGenerators.Terrace200 terrace = null;
  for (int i=0; i<98; i++)
  {
```

```
      MatrixGenerators.Terrace200 newTerrace = (MatrixGenerators.Terrace200)Generator.Create(typeof(M
      graph.Add(newTerrace);
      newTerrace.guiPosition = new Vector2(-70 + 200*i,-100 + j*200);
      if (i==0) graph.Link(newTerrace, noise);
      else graph.Link(newTerrace, (IOutlet<object>)terrace);
      terrace = newTerrace;
     }

     MatrixGenerators.HeightOutput200 output = (MatrixGenerators.HeightOutput200)Generator.Create(type
     graph.Add(output);
     output.guiPosition = new Vector2(130 + 200*98, -100 + j*200);
     graph.Link(output, terrace);
    }*/

    return graph;
   }
   #endif
 }
}
```

```
ï»¿using System;
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;

using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Products;
using MapMagic.Previews;

namespace MapMagic.Nodes.GUI
{
 //[EditoWindowTitle(title = "MapMagic Graph")]  //it's internal Unity stuff
 public class GraphWindow : EditorWindow
 {
  public static GraphWindow current;  //assigned each gui draw (and nulled after)

  //public List<Graph> graphs = new List<Graph>();
  //public Graph CurrentGraph { get{ if (graphs.Count==0) return null; else return graphs[graphs.Count-1]; }}
  //public Graph RootGraph { get{ if (graphs.Count==0) return null; else return graphs[0]; }}
```

```
public Graph graph;
public List<Graph> parentGraphs;   //the ones on pressing "up level" button
        //we can have the same function in two biomes. Where should we exit on pressing "up level"?
        //automatically created on opening window, though

private bool drawAddRemoveButton = true;  //turning off Add/Remove on opening popup with it, and re-en

public static Dictionary<Graph,Vector3> graphsScrollZooms = new Dictionary<Graph, Vector3>();
//to remember the stored graphs scroll/zoom to switch between graphs
//public for snapshots

public IMapMagic mapMagic;

private static UnityEngine.SceneManagement.Scene prevSceneLoaded; //to search for mapmagic only w
private static int prevRootObjsCount;
private static UnityEngine.Object prevObjSelected;

public bool MapMagicRelevant =>
 mapMagic != null  &&
 mapMagic.Graph != null  &&
 (mapMagic.Graph == current.graph  ||  mapMagic.Graph.ContainsSubGraph(current.graph, recursively:tr

public void UpdateRelatedMapMagic () => mapMagic = FindRelatedMapMagic(graph); //mostly for public

public static IMapMagic FindRelatedMapMagic (Graph graph)
```

```
{
 if (Selection.activeObject is IMapMagic imm)
  if (imm.ContainsGraph(graph)) return imm;
 //doesn't work with MM object, but leaving here just in case

 //looking in selection
 if (Selection.activeObject is GameObject selectedGameObj)
 {
  MapMagicObject mmo = selectedGameObj.GetComponent<MapMagicObject>();
  if (mmo != null && mmo.ContainsGraph(graph)) return mmo;
  //we can't assign ClusterAsset same way! Add code for it
 }

 //looking in all objects (only on scene reload or selection change)
 UnityEngine.SceneManagement.Scene scene = UnityEngine.SceneManagement.SceneManager.GetAct
 if (scene != prevSceneLoaded || scene.rootCount != prevRootObjsCount || Selection.activeObject != prev
 {
  MapMagicObject[] allMM = GameObject.FindObjectsOfType<MapMagicObject>();
  for (int m=0; m<allMM.Length; m++)
   if (allMM[m].ContainsGraph(graph)) return allMM[m];

  prevSceneLoaded = scene;
  prevRootObjsCount = scene.rootCount;
  prevObjSelected = Selection.activeObject;

  #if MM_DEBUG
```

```
  Debug.Log("Finding MapMagic");
  #endif
 }

 return null;
}

public static IMapMagic RelatedMapMagic
{get{
 if (current == null  ||  current.mapMagic == null  ||  current.mapMagic.Graph == null) return null;
 if (current.mapMagic.Graph != current.graph  &&  !current.mapMagic.Graph.ContainsSubGraph(current.g
 return current.mapMagic;
}}

public void RefreshMapMagic ()
{
 //if (MapMagicRelevant)  //with new clear system relevancy check is not necessary
  mapMagic?.Refresh(false);

 OnGraphChanged?.Invoke(graph);
}

public static Action<Graph> OnGraphChanged;



public static void RecordCompleteUndo ()
```

```
{
 current.graphUI.undo.Record(completeUndo:true);
}
//the usual undo is recorded on valChange via gui

const int toolbarSize = 20;

public UI graphUI = UI.ScrolledUI(maxZoom:1, minZoom:0.375f);  //public for snapshot
UI toolbarUI = new UI();
UI dragUI = new UI();
UI miniSelectedUI = new UI();

public bool IsMini => graphUI.scrollZoom.zoom < 0.4f; //minimum full-version zoom is 0.4375. Next step (
public const float miniZoom = 0.375f;

bool wasGenerating = false; //to update final frame when generate is finished

private static Vector2 addDragTo = new Vector2(Screen.width-50,20);
private static Vector2 AddDragDefault {get{ return new Vector2(Screen.width-50,20); }}
private const int addDragSize = 34;
private const int addDragOffset = 20; //the offset from screen corner
private static readonly object addDragId = new object();

private Vector2 addButtonDragOffset;

public HashSet<Generator> selected = new HashSet<Generator>();
```

```
private long lastFrameTime;



public void OnEnable ()
{
 //redrawing previews
 //Preview.OnRefreshed += p => Repaint();

 #if UNITY_2019_1_OR_NEWER
 SceneView.duringSceneGui -= OnSceneGUI;
 SceneView.duringSceneGui += OnSceneGUI;
 #else
 SceneView.onSceneGUIDelegate -= OnSceneGUI;
 SceneView.onSceneGUIDelegate += OnSceneGUI;
 #endif

 ScrollZoomOnOpen(); //focusing after script re-compile

 selected.Clear(); //removing selection from previous graph
}

public void OnDisable ()
{
 #if UNITY_2019_1_OR_NEWER
 SceneView.duringSceneGui -= OnSceneGUI;
```

```
 #else
 SceneView.onSceneGUIDelegate -= OnSceneGUI;
 #endif

  UnityEditor.Tools.hidden = false; //in case gizmo node is turned on
}

public void OnInspectorUpdate ()
{
 current = this;

 //updating gauge
 if (mapMagic == null) return;
 bool isGenerating = mapMagic.IsGenerating()  &&  mapMagic.ContainsGraph(graph);
 if (wasGenerating) { Repaint(); wasGenerating=false; } //1 frame delay after generate is finished
 if (isGenerating) { Repaint(); wasGenerating=true; }
}


private void OnGUI()
{
 current = this;

 mapMagic = FindRelatedMapMagic(graph);

 if (graph==null || graph.generators==null) return;
```

```
if (mapMagic==null) Den.Tools.Tasks.CoroutineManager.Update(); //updating coroutine if no mm assigne


//fps timer
#if MM_DEBUG
long frameStart = System.Diagnostics.Stopwatch.GetTimestamp();
#endif


//undo
if (graphUI.undo == null)
{
 graphUI.undo = new Den.Tools.GUI.Undo() { undoObject = graph , undoName = "MapMagic Graph Cha
 graphUI.undo.undoAction = GraphWindow.current.RefreshMapMagic;
}
graphUI.undo.undoObject = graph;



//mini with selection
if (selected.Count == 1  &&  IsMini)
{
 Generator selectedGen = selected.Any();

 Rect selectedRect = new Rect(PlaceByAnchor(graph.guiMiniAnchor,graph.guiMiniPos,selectedGen.guiS
 selectedRect.position = selectedRect.position+new Vector2(0,toolbarSize);

 //skipping drawing graph if clicked somewhere in generator
```

```
 bool clickedToGen = Event.current.isMouse  &&  selectedRect.Contains(Event.current.mousePosition);

 using (new UnityEngine.GUI.ClipScope(new Rect(0, toolbarSize, Screen.width, Screen.height-toolbarSiz
 {
  if (clickedToGen) EditorGUI.BeginDisabledGroup(true); //disabling other controls when clicking selected
  graphUI.Draw(DrawGraph, inInspector:false);
  if (clickedToGen) EditorGUI.EndDisabledGroup();

  miniSelectedUI.Draw(DrawMiniSelected, inInspector:false);
 }

 //using (new UnityEngine.GUI.ClipScope(new Rect(20, 20+toolbarSize, selectedGen.guiSize.x, selected
 // miniSelectedUI.Draw(DrawMiniSelected, inInspector:false);
}

//standard graph/mini
else
 using (new UnityEngine.GUI.ClipScope(new Rect(0, toolbarSize, Screen.width, Screen.height-toolbarSiz
  graphUI.Draw(DrawGraph, inInspector:false, customRect:new Rect(0, toolbarSize, Screen.width, Scree

//toolbar
using (new UnityEngine.GUI.ClipScope(new Rect(0,0, Screen.width, toolbarSize)))
 toolbarUI.Draw(DrawToolbar, inInspector:false);

//storing graph pivot to focus it on load
Vector3 scrollZoom = graphUI.scrollZoom.GetWindowCenter(position.size);
```

```
scrollZoom.z = graphUI.scrollZoom.zoom;
if (graphsScrollZooms.ContainsKey(graph)) graphsScrollZooms[graph] = scrollZoom;
else graphsScrollZooms.Add(graph, scrollZoom);


//preventing switching to main while dragging field (for MMobject only)
if (mapMagic != null  &&  mapMagic is MapMagicObject mapMagicObject)
{
 bool newForceDrafts = DragDrop.obj!=null && (DragDrop.group=="DragField" || DragDrop.group=="Drag
 if (!newForceDrafts  &&  mapMagicObject.guiDraggingField)
 {
  mapMagicObject.guiDraggingField = newForceDrafts;
  mapMagicObject.SwitchLods();
 }
 mapMagicObject.guiDraggingField = newForceDrafts;
}


//showing fps
#if MM_DEBUG
if (graph.debugGraphFps)
{
 long frameEnd = System.Diagnostics.Stopwatch.GetTimestamp();
 float timeDelta = 1f * (frameEnd-frameStart) / System.Diagnostics.Stopwatch.Frequency;
 float fps = 1f / timeDelta;
 EditorGUI.LabelField(new Rect(10, toolbarSize+10, 70, 18), "FPS:" + fps.ToString("0.0"));
}
#endif
```

```
 //moving scene view
 #if MM_DEBUG
 if (graph.drawInSceneView)
  MoveSceneView();
 #endif
}



private void DrawGraph ()
{
 bool isMini = IsMini;

 //background
 float gridColor = !StylesCache.isPro ? 0.45f : 0.12f;
 float gridBackgroundColor = !StylesCache.isPro ? 0.5f : 0.15f;

 #if MM_DEBUG
  if (!graph.debugGraphBackground)
  {
   gridColor = graph.debugGraphBackColor;
   gridBackgroundColor = graph.debugGraphBackColor;
  }
 #endif

 Draw.StaticGrid(
```

```
 displayRect: new Rect(0, 0, Screen.width, Screen.height-toolbarSize),
 cellSize:32,
 color:new Color(gridColor,gridColor,gridColor),
 background:new Color(gridBackgroundColor,gridBackgroundColor,gridBackgroundColor),
 fadeWithZoom:true);

#if MM_DEBUG
 if (graph.drawInSceneView)
 {
  using (Cell.Full)
   DrawSceneView();
 }
#endif

//drawing groups
foreach (Group group in graph.groups)
 using (Cell.Custom(group.guiPos.x, group.guiPos.y, group.guiSize.x, group.guiSize.y))
 {
  GroupDraw.DragGroup(group, graph.generators);
  GroupDraw.DrawGroup(group, isMini:isMini);
 }


//dragging nodes
foreach (Generator gen in graph.generators)
 GeneratorDraw.DragGenerator(gen, selected);
```

```
//drawing links
//using (Timer.Start("Links"))
if (!UI.current.layout)
{
 List<(IInlet<object> inlet, IOutlet<object> outlet)> linksToRemove = null;
 foreach (var kvp in graph.links)
 {
  IInlet<object> inlet = kvp.Key;
  IOutlet<object> outlet = kvp.Value;

  Cell outletCell = UI.current.cellObjs.GetCell(outlet, "Outlet");
  Cell inletCell = UI.current.cellObjs.GetCell(inlet, "Inlet");

  if (outletCell == null || inletCell == null)
  {
   Debug.LogError("Could not find a cell for inlet/outlet. Removing link");
   if (linksToRemove == null) linksToRemove = new List<(IInlet<object> inlet, IOutlet<object> outlet)>();
   linksToRemove.Add((inlet,outlet));
   continue;
  }

  GeneratorDraw.DrawLink(
   GeneratorDraw.StartCellLinkpos(outletCell),
   GeneratorDraw.EndCellLinkpos(inletCell),
```

```
   GeneratorDraw.GetLinkColor(inlet),
   width:!isMini ? 4f : 6f );
 }

 if (linksToRemove != null)
  foreach ((IInlet<object> inlet, IOutlet<object> outlet) in linksToRemove)
  {
   graph.UnlinkInlet(inlet);
   graph.UnlinkOutlet(outlet);
  }
}

//removing null generators (for test purpose)
for (int n=graph.generators.Length-1; n>=0; n--)
{
 if (graph.generators[n] == null)
  ArrayTools.RemoveAt(ref graph.generators, n);
}

//drawing generators
//using (Timer.Start("Generators"))
float nodeWidth = !isMini ? GeneratorDraw.nodeWidth : GeneratorDraw.miniWidth;
foreach (Generator gen in graph.generators)
 using (Cell.Custom(gen.guiPosition.x, gen.guiPosition.y, nodeWidth, 0))
  GeneratorDraw.DrawGeneratorOrPortal(gen, graph, isMini:isMini, selected.Contains(gen));
```

```
//de-selecting nodes (after dragging and drawing since using drag obj)
if (!UI.current.layout)
{
 GeneratorDraw.SelectGenerators(selected, shiftForSingleSelect:!isMini);
 GeneratorDraw.DeselectGenerators(selected); //and deselected always without shift
}

//add/remove button
//using (Timer.Start("AddRemove"))
using (Cell.Full)
 DragDrawAddRemove();

//right click menu (should have access to cellObjs)
if (!UI.current.layout  &&  Event.current.type == EventType.MouseDown  &&  Event.current.button == 1)
 RightClick.DrawRightClickItems(graphUI, graphUI.mousePos, graph);

//create menu on space
if (!UI.current.layout  &&  Event.current.type == EventType.KeyDown  &&  Event.current.keyCode == Key
 CreateRightClick.DrawCreateItems(graphUI.mousePos, graph);

//delete selected generators
if (selected!=null  &&  selected.Count!=0  &&  Event.current.type==EventType.KeyDown  &&  Event.curre
 GraphEditorActions.RemoveGenerators(graph, selected);
}
```

```
private void DrawMiniSelected ()
{
 if (selected.Count != 1)
  return;

 Generator gen = selected.Any();

 Vector2 cellSize = new Vector2(gen.guiSize.x, 0);
 Vector2 cellPos = PlaceByAnchor(graph.guiMiniAnchor, graph.guiMiniPos, gen.guiSize);

 using (Cell.Custom(cellPos, cellSize))
 {
  //dragging
  if (!UI.current.layout)
  {
   if (DragDrop.TryDrag(Cell.current, UI.current.mousePos))
   {
    Vector2 newPosition = GeneratorDraw.MoveGenerator(Cell.current, DragDrop.initialRect.position + Dra
    (graph.guiMiniAnchor, graph.guiMiniPos) = GetAnchorPos(newPosition, gen.guiSize);
   }
   DragDrop.TryRelease(Cell.current);
   DragDrop.TryStart(Cell.current, UI.current.mousePos, Cell.current.InternalRect);
  }

  //shadow
  //GUIStyle shadowStyle = UI.current.textures.GetElementStyle("MapMagic/Node/ShadowMini",
```

```
  // borders:GeneratorDraw.shadowBorders,
  // overflow:GeneratorDraw.shadowOverflow);
  //Draw.Element(shadowStyle);

  //drawing
  try { GeneratorDraw.DrawGenerator(gen, graph, selected:false, activeLinks:true); }
  catch (ExitGUIException)
   { } //ignoring
  catch (Exception e)
   { Debug.LogError("Draw Graph Window failed: " + e); }

  //right click menu (should have access to cellObjs)
  if (!UI.current.layout  &&  Event.current.type == EventType.MouseDown  &&  Event.current.button == 1)
   RightClick.DrawRightClickItems(miniSelectedUI, miniSelectedUI.mousePos, graph);
 }
}


private void DrawToolbar ()
{
 //using (Timer.Start("DrawToolbar"))

 using (Cell.LinePx(toolbarSize))
 {
  Draw.Button();
```

```
//Graph graph = CurrentGraph;
//Graph rootGraph = mapMagic.graph;

//if (mapMagic != null  &&  mapMagic.graph!=graph  &&  mapMagic.graph!=rootGraph) mapMagic = null;

UI.current.styles.Resize(0.9f);  //shrinking all font sizes

Draw.Element(UI.current.styles.toolbar);



//undefined graph
if (graph==null)
{
 using (Cell.RowPx(200)) Draw.Label("No graph selected to display. Select:");
 using (Cell.RowPx(100)) Draw.ObjectField(ref graph);
 return;
}

//if graph loaded corrupted
if (graph.generators==null)
{
 using (Cell.RowPx(300)) Draw.Label("Graph is null. Check the console for the error on load.");

 using (Cell.RowPx(100))
  if (Draw.Button("Reload", style:UI.current.styles.toolbarButton)) graph.OnAfterDeserialize();
```

```
 using (Cell.RowPx(100))
 {
  if (Draw.Button("Reset", style:UI.current.styles.toolbarButton)) graph.generators = new Generator[0];
 }

 Cell.EmptyRowRel(1);

 return;
}

//root graph
Graph rootGraph = null;
if (parentGraphs != null  &&  parentGraphs.Count != 0)
 rootGraph = parentGraphs[0];
 //this has nothing to do with currently assigned mm graph - we can view subGraphs with no mm in scen

if (rootGraph != null)
{
 Vector2 rootBtnSize = UnityEngine.GUI.skin.label.CalcSize( new GUIContent(rootGraph.name) );
 using (Cell.RowPx(rootBtnSize.x))
 {
  //Draw.Button(graph.name, style:UI.current.styles.toolbarButton, cell:rootBtnCell);
  Draw.Label(rootGraph.name);
   if (Draw.Button("", visible:false))
    EditorGUIUtility.PingObject(rootGraph);
 }
```

```
 using (Cell.RowPx(20)) Draw.Label(">>");
}

//this graph
Vector2 graphBtnSize = UnityEngine.GUI.skin.label.CalcSize( new GUIContent(graph.name) );
using (Cell.RowPx(graphBtnSize.x))
{
 Draw.Label(graph.name);
 if (Draw.Button("", visible:false))
  EditorGUIUtility.PingObject(graph);
}

//up-level and tree
using (Cell.RowPx(20))
{
 if (Draw.Button(null, icon:UI.current.textures.GetTexture("DPUI/Icons/FolderTree"), iconScale:0.5f, visib
  GraphTreePopup.DrawGraphTree(rootGraph!=null ? rootGraph : graph);
}

using (Cell.RowPx(20))
{
 if (parentGraphs != null  &&  parentGraphs.Count != 0  &&
  Draw.Button(null, icon:UI.current.textures.GetTexture("DPUI/Icons/FolderUp"), iconScale:0.5f, visible:fa
 {
  graph = parentGraphs[parentGraphs.Count-1];
```

```
  parentGraphs.RemoveAt(parentGraphs.Count-1);
  ScrollZoomOnOpen();
  Repaint();
 }
}

Cell.EmptyRowRel(1); //switching to right corner

//seed
Cell.EmptyRowPx(5);
using (Cell.RowPx(1)) Draw.ToolbarSeparator();

using (Cell.RowPx(90))
// using (Cell.LinePx(toolbarSize-1))  //-1 just to place it nicely
{
 #if UNITY_2019_1_OR_NEWER
 int newSeed;
 using (Cell.RowRel(0.4f)) Draw.Label("Seed:");
 using (Cell.RowRel(0.6f))
  using (Cell.Padded(1))
   newSeed = (int)Draw.Field(graph.random.Seed, style:UI.current.styles.toolbarField);
 #else
 Cell.current.fieldWidth = 0.6f;
 int newSeed = Draw.Field(graph.random.Seed, "Seed:");
 #endif
 if (newSeed != graph.random.Seed)
```

```
 {
  GraphWindow.RecordCompleteUndo();
  graph.random.Seed = newSeed;
  GraphWindow.current?.RefreshMapMagic();
 }
}

Cell.EmptyRowPx(2);



//gauge
using (Cell.RowPx(1)) Draw.ToolbarSeparator();

using (Cell.RowPx(200))
 using (Cell.LinePx(toolbarSize-1)) //-1 to leave underscore under gauge
{
 if (mapMagic != null)
 {
  bool isGenerating = mapMagic.IsGenerating();

  //background gauge
  if (isGenerating)
  {
   float progress = mapMagic.GetProgress();

   if (progress < 1 && progress != 0)
```

```
 {
  Texture2D backgroundTex = UI.current.textures.GetTexture("DPUI/ProgressBar/BackgroundBorderle
  mapMagic.GetProgress();
  Draw.Texture(backgroundTex);

  Texture2D fillTex = UI.current.textures.GetBlankTexture(StylesCache.isPro ? Color.grey : Color.white
  Color color = StylesCache.isPro ? new Color(0.24f, 0.37f, 0.58f) : new Color(0.44f, 0.574f, 0.773f);
  Draw.ProgressBarGauge(progress, fillTex, color);
 }

 //Repaint(); //doing it in OnInspectorUpdate
}

//refresh buttons
using (Cell.RowPx(20))
 if (Draw.Button(null, icon:UI.current.textures.GetTexture("DPUI/Icons/RefreshAll"), iconScale:0.5f, visib
 {
  //graphUI.undo.Record(completeUndo:true); //won't record changed terrain data
  if (mapMagic is MapMagicObject mapMagicObject)
  {
   foreach (Terrain terrain in mapMagicObject.tiles.AllActiveTerrains())
    UnityEditor.Undo.RegisterFullObjectHierarchyUndo(terrain.terrainData, "RefreshAll");
   EditorUtility.SetDirty(mapMagicObject);
  }

  current.mapMagic.Refresh(clearAll:true);
```

```
    }

   using (Cell.RowPx(20))
    if (Draw.Button(null, icon:UI.current.textures.GetTexture("DPUI/Icons/Refresh"), iconScale:0.5f, visible
    {
     current.mapMagic.Refresh(clearAll:false);
    }

   //ready mark
   if (!isGenerating)
   {
    Cell.EmptyRow();
    using (Cell.RowPx(40)) Draw.Label("Ready");
   }
  }

  else
   Draw.Label("Not Assigned to MapMagic Object");
 }

 using (Cell.RowPx(1)) Draw.ToolbarSeparator();

 //focus
 using (Cell.RowPx(20))
  if (Draw.Button(null, icon:UI.current.textures.GetTexture("DPUI/Icons/FocusSmall"), iconScale:0.5f, visib
  {
```

```
    graphUI.scrollZoom.FocusWindowOn(GetNodesCenter(graph), position.size);
   }

  using (Cell.RowPx(20))
  {
   if (graphUI.scrollZoom.zoom < 0.999f)
   {
    if (Draw.Button(null, icon:UI.current.textures.GetTexture("DPUI/Icons/ZoomSmallPlus"), iconScale:0.5f,
     graphUI.scrollZoom.Zoom(1f, position.size/2);
   }
   else
   {
    if (Draw.Button(null, icon:UI.current.textures.GetTexture("DPUI/Icons/ZoomSmallMinus"), iconScale:0.5
     graphUI.scrollZoom.Zoom(miniZoom, position.size/2);
   }
  }
 }
}


private void DrawSceneView ()
{
 Rect windowRect = UI.current.editorWindow.position;
 SceneView sceneView = SceneView.lastActiveSceneView;

 //drawing
```

```
 if (sceneTex == null  ||  sceneTex.width != (int)windowRect.width  ||  sceneTex.height != (int)windowRect.
  sceneTex = new RenderTexture((int)windowRect.width, (int)windowRect.height, 24, RenderTextureForm
 RenderTexture backTex = sceneView.camera.targetTexture;
 sceneView.camera.targetTexture = sceneTex;
 sceneView.camera.Render();
 sceneView.camera.targetTexture = backTex;

 using (Cell.Custom(
  0,
  0,
  UI.current.editorWindow.position.width,
  UI.current.editorWindow.position.height))
 {
  Cell.current.MakeStatic();
  //Draw.Icon(sceneTex);
  Draw.Texture(sceneTex);
 }

 //moving/rotating




}
private static RenderTexture sceneTex = null;

private void MoveSceneView ()
```

```
{
 SceneView sceneView = SceneView.lastActiveSceneView;

 if (Event.current.alt  &&  Event.current.button == 2)
 {
  Ray rayZero = sceneView.camera.ViewportPointToRay(Vector2.zero);
  Vector3 pointZero = rayZero.origin + rayZero.direction*sceneView.cameraDistance;

  Ray rayX = sceneView.camera.ViewportPointToRay(new Vector2(1,0));
  Vector3 pointX = rayX.origin + rayX.direction*sceneView.cameraDistance;

  Ray rayY = sceneView.camera.ViewportPointToRay(new Vector2(0,1));
  Vector3 pointY = rayY.origin + rayY.direction*sceneView.cameraDistance;

  Vector3 axisX = pointZero-pointX;
  Vector3 axisY = pointZero-pointY;

  Vector2 relativeDelta = Event.current.delta / new Vector2(Screen.width, Screen.height);
  relativeDelta.y = -relativeDelta.y;
  sceneView.pivot += axisX*relativeDelta.x + axisY*relativeDelta.y;

  //Debug.DrawLine(pointZero, pointX, Color.red);
  //unfortunately ViewportPointToRay uses scene view - and will return improper values if aspect is differe

  Repaint();
 }
```

```
 if (Event.current.alt  &&  Event.current.button == 0  &&  !Event.current.isScrollWheel)
 {
  Vector3 rotation = sceneView.rotation.eulerAngles;
  rotation.y += Event.current.delta.x / 10;
  rotation.x += Event.current.delta.y / 10;
  sceneView.rotation = Quaternion.Euler(rotation);

  Repaint();
 }

 if (Event.current.alt  &&  Event.current.isScrollWheel) //undocumented!
 {
  //Vector3 camVec = sceneView.camera.transform.position - sceneView.pivot;
  //float camVecLength = camVec.magnitude;
  //camVecLength *= 1 + Event.current.delta.y*0.02f;

  float size = sceneView.size;
  size *= 1 + Event.current.delta.y*0.02f;

  sceneView.LookAtDirect(sceneView.pivot, sceneView.rotation, size);

  Repaint();
 }
}
```

```
private static Vector2 GetNodesCenter (Graph graph)
{
 //Graph graph = CurrentGraph;
 if (graph.generators.Length==0) return new Vector2(0,0);

 Vector2 min = graph.generators[0].guiPosition;
 Vector2 max = min + graph.generators[0].guiSize;

 for (int g=1; g<graph.generators.Length; g++)
 {
  Vector2 pos = graph.generators[g].guiPosition;
  min = Vector2.Min(pos, min);
  max = Vector2.Max(pos + graph.generators[g].guiSize, max);
 }

 return (min + max)/2;
}


private static (Vector2,Vector2) GetAnchorPos (Vector2 genPos, Vector2 genSize)
{
 Vector2 genCenter = genPos + genSize/2;
 Vector2 anchor =  new Vector2(
  genCenter.x > Screen.width/2 ? 1 : 0,
  genCenter.y > Screen.height/2 ? 1 : 0 );
```

```
 Vector2 genCorner = genPos + genSize*anchor;
 Vector2 screenCorner = new Vector2(Screen.width, Screen.height)*anchor;
 Vector2 sign = -(anchor*2 - Vector2.one);

 Vector2 pos = (screenCorner + genCorner*sign)*sign;

 Vector2 absPos = pos*sign;
 if (absPos.x < 0) pos.x = 0;
 if (absPos.y < 0) pos.y = 0;

 return (anchor, pos);
}


private static Vector2 PlaceByAnchor  (Vector2 anchor, Vector2 pos, Vector2 size)
{
 Vector2 screenCorner = new Vector2(Screen.width, Screen.height)*anchor;
 Vector2 genCorner = size*anchor;

 return screenCorner - genCorner + pos;
}


public void OnSceneGUI (SceneView sceneview)
{
```

```
 if (graph==null || graph.generators==null) return; //if graph loaded corrupted

 bool hideDefaultToolGizmo = false; //if any of the nodes has it's gizmo enabled (to hide the default tool)

 for (int n=0; n<graph.generators.Length; n++)
  if (graph.generators[n] is ISceneGizmo)
  {
   ISceneGizmo gizmoNode = (ISceneGizmo)graph.generators[n];
   gizmoNode.DrawGizmo();
   if (gizmoNode.hideDefaultToolGizmo) hideDefaultToolGizmo = true;
  }

 if (hideDefaultToolGizmo) UnityEditor.Tools.hidden = true;
 else UnityEditor.Tools.hidden = false;
}


private void DragDrawAddRemove ()
{
 int origButtonSize = 34; int origButtonOffset = 20;

 Vector2 buttonPos = new Vector2(
  UI.current.editorWindow.position.width - (origButtonSize + origButtonOffset)*UI.current.DpiScaleFactor,
  20*UI.current.DpiScaleFactor);
 Vector2 buttonSize = new Vector2(origButtonSize,origButtonSize) * UI.current.DpiScaleFactor;
```

```
using (Cell.Custom(buttonPos,buttonSize))
//later button pos could be overriden if dragging it
{
 Cell.current.MakeStatic();



 //if dragging generator
 if (DragDrop.IsDragging()  &&  !DragDrop.IsStarted()  &&  DragDrop.obj is Cell  &&  UI.current.cellObjs.T


 {
  if (Cell.current.Contains(UI.current.mousePos))
   Draw.Texture(UI.current.textures.GetTexture("MapMagic/Icons/NodeRemoveActive"));
  else
   Draw.Texture(UI.current.textures.GetTexture("MapMagic/Icons/NodeRemove"));
 }



 //if released generator on remove icon
 else if (DragDrop.IsReleased()  &&
  DragDrop.releasedObj is Cell  &&
  UI.current.cellObjs.TryGetObject((Cell)DragDrop.releasedObj, "Generator", out Generator releasedGen
  Cell.current.Contains(UI.current.mousePos))
 {
  GraphEditorActions.RemoveGenerators(graph, selected, releasedGen);
  GraphWindow.current?.RefreshMapMagic();
 }
```

```
//if not dragging generator
else
{
 if (focusedWindow==this) drawAddRemoveButton = true;   //re-enabling when window is focused again
 bool drawFrame = false;
 Color frameColor = new Color();

 //dragging button
 if (DragDrop.TryDrag(addDragId, UI.current.mousePos))
 {
  Cell.current.pixelOffset += DragDrop.totalDelta; //offsetting cell position with the mouse

  Draw.Texture(UI.current.textures.GetTexture("MapMagic/Icons/NodeAdd"));

  //if dragging near link, output or node
  Vector2 mousePos = graphUI.mousePos;
  //Vector2 mousePos = graphUI.scrollZoom.ToInternal(addDragTo + new Vector2(addDragSize/2,addD

  object clickedObj = RightClick.ClickedOn(graphUI, mousePos);

  if (clickedObj != null  &&  !(clickedObj is Group))
  {
   drawFrame = true;
   frameColor = GeneratorDraw.GetLinkColor(Generator.GetGenericType(clickedObj.GetType()));
```

```
 }
}

//releasing button
if (DragDrop.TryRelease(addDragId))
{
 drawAddRemoveButton = false;

 Vector2 mousePos = graphUI.mousePos;
 //Vector2 mousePos = graphUI.scrollZoom.ToInternal(addDragTo + new Vector2(addDragSize/2,addD

 RightClick.ClickedNear (graphUI, mousePos,
  out Group clickedGroup, out Generator clickedGen, out IInlet<object> clickedLink, out IInlet<object> cl

 if (clickedOutlet != null)
  CreateRightClick.DrawAppendItems(mousePos, graph, clickedOutlet);

 else if (clickedLink != null)
  CreateRightClick.DrawInsertItems(mousePos, graph, clickedLink);

 else
  CreateRightClick.DrawCreateItems(mousePos, graph);
}

//starting button drag
DragDrop.TryStart(addDragId, UI.current.mousePos, Cell.current.InternalRect);
```

```
    //drawing button
    #if !MM_DOC
    if (drawAddRemoveButton) //don't show this button if right-click items are shown
     Draw.Texture(UI.current.textures.GetTexture("MapMagic/Icons/NodeAdd")); //using Texture since Icon
    #endif

    if (drawFrame)
    {
     Texture2D frameTex = UI.current.textures.GetColorizedTexture("MapMagic/Icons/NodeAddRemoveFra
     Draw.Texture(frameTex);
    }
   }
  }
 }

#region Showing Window

 public static GraphWindow ShowInNewTab (Graph graph)
 {
  GraphWindow window = CreateInstance<GraphWindow>();

  window.OpenRoot(graph);

  ShowWindow(window, inTab:true);
  return window;
```

```
}

public static GraphWindow Show (Graph graph)
{
 GraphWindow window = null;
 GraphWindow[] allWindows = Resources.FindObjectsOfTypeAll<GraphWindow>();

 //if opened as biome via focused graph window - opening as biome
 if (focusedWindow is GraphWindow focWin  &&  focWin.graph.ContainsSubGraph(graph))
 {
  focWin.OpenBiome(graph);
  return focWin;
 }

 //if opened only one window - using it (and trying to load mm biomes)
 if (window == null)
 {
  if (allWindows.Length == 1)
  {
   window = allWindows[0];
   if (!window.TryOpenMapMagicBiome(graph))
    window.OpenRoot(graph);
  }
 }

 //if window with this graph currently opened - just focusing it
```

```
if (window == null)
{
 for (int w=0; w<allWindows.Length; w++)
  if (allWindows[w].graph == graph)
   window = allWindows[w];
}

//if the window with parent graph currently opened
if (window == null)
{
 for (int w=0; w<allWindows.Length; w++)
  if (allWindows[w].graph.ContainsSubGraph(graph))
  {
   window = allWindows[w];
   window.OpenBiome(graph);
  }
}

//if no window found after all - creating new tab (and trying to load mm biomes)
if (window == null)
{
 window = CreateInstance<GraphWindow>();
 if (!window.TryOpenMapMagicBiome(graph))
  window.OpenRoot(graph);
}
```

```
 ShowWindow(window, inTab:false);
 return window;
}


public void OpenBiome (Graph graph)
/// In this case we know for sure what window should be opened. No internal checks
{
 if (parentGraphs == null) parentGraphs = new List<Graph>();
 parentGraphs.Add(this.graph);
 this.graph = graph;
 DragDrop.obj = null; //resetting dragDrop or it will move any generator to position of this one on open
 ScrollZoomOnOpen();
}


public void OpenBiome (Graph graph, Graph root)
/// Opens graph as sub-sub-sub biome to root
{
 parentGraphs = GetStepsToSubGraph(root, graph);
 this.graph = graph;
 DragDrop.obj = null;
 ScrollZoomOnOpen();
}
```

```
private bool TryOpenMapMagicBiome (Graph graph)
/// Finds MapMagic object in scene and opens graph as mm biome with mm graph as a root
/// Return false if it's wrong mm (or no mm at all)
{
 mapMagic = FindRelatedMapMagic(graph);
 if (mapMagic == null) return false;

 parentGraphs = GetStepsToSubGraph(mapMagic.Graph, graph);
 this.graph = graph;

 DragDrop.obj = null;
 ScrollZoomOnOpen();

 return true;
}


private void OpenRoot (Graph graph)
{
 this.graph = graph;
 parentGraphs = null;

 DragDrop.obj = null; //resetting dragDrop or it will move any generator to position of this one on open
 ScrollZoomOnOpen();
}
```

```
private static void ShowWindow (GraphWindow window, bool inTab=false)
/// Opens the graph window. But it should be created and graph assigned first.
{
 Texture2D icon = TexturesCache.LoadTextureAtPath("MapMagic/Icons/Window");
 window.titleContent = new GUIContent("MapMagic Graph", icon);

 if (inTab) window.ShowTab();
 else window.Show();
 window.Focus();
 window.Repaint();

 DragDrop.obj = null;
 window.ScrollZoomOnOpen(); //focusing after window has shown (since it needs window size)
}


private static GraphWindow FindReusableWindow (Graph graph)
/// Finds the most appropriate window among all of all currently opened
{
 GraphWindow[] allWindows = Resources.FindObjectsOfTypeAll<GraphWindow>();

 //if opened only one window - using it
 if (allWindows.Length == 1)
  return allWindows[0];
```

```
 //if opening from currently active window
 if (focusedWindow is GraphWindow focWin)
  if (focWin.graph.ContainsSubGraph(graph))
   return focWin;

 //if window with this graph currently opened
 for (int w=0; w<allWindows.Length; w++)
  if (allWindows[w].graph == graph)
   return allWindows[w];

 //if the window with parent graph currently opened
 for (int w=0; w<allWindows.Length; w++)
  if (allWindows[w].graph.ContainsSubGraph(graph))
   return allWindows[w];

 return null;
}


private void ScrollZoomOnOpen ()
///Finds a graph scroll and zoom from graphsScrollZooms and focuses on them. To switch between graph
///should be called each time new graph assigned
{
 if (graph == null) return;

 if (graphsScrollZooms.TryGetValue(graph, out Vector3 scrollZoom))
```

```
 {
  graphUI.scrollZoom.FocusWindowOn(new Vector2(scrollZoom.x, scrollZoom.y), position.size);
  graphUI.scrollZoom.zoom = scrollZoom.z;
 }

 else
  graphUI.scrollZoom.FocusWindowOn(GetNodesCenter(graph), position.size);
}



public static List<Graph> GetStepsToSubGraph (Graph rootGraph, Graph subGraph)
/// returns List(this > biome > innerBiome)
/// doesn't include the subgraph itself
/// doesn't perform check if subGraph is contained within graph at all
{
 List<Graph> steps = new List<Graph>();
 ContainsSubGraphSteps(rootGraph, subGraph, steps);
 steps.Reverse();
 return steps;
}



private static bool ContainsSubGraphSteps (Graph thisGraph, Graph subGraph, List<Graph> steps)
/// Same as ContainsSubGraph, but using track list for GetStepsToSubGraph
{
 if (thisGraph == subGraph)
```

```
  return true;

 foreach (Graph biomeSubGraph in thisGraph.SubGraphs())
  if (ContainsSubGraphSteps(biomeSubGraph, subGraph, steps))
  {
   steps.Add(thisGraph);
   return true;
  }

 return false;
}



[MenuItem ("Window/MapMagic/Editor")]
public static void ShowEditor ()
{
 MapMagicObject mm = FindObjectOfType<MapMagicObject>();
 Graph gens = mm!=null? mm.graph : null;
 GraphWindow.Show(mm?.graph);
}

[UnityEditor.Callbacks.OnOpenAsset(0)]
public static bool ShowEditor (int instanceID, int line)
{
 UnityEngine.Object obj = EditorUtility.InstanceIDToObject(instanceID);
 if (obj is Nodes.Graph graph)
```

```
    {
     if (graph.generators == null)
      graph.OnAfterDeserialize();
     if (graph.generators == null)
      throw new Exception("Error loading graph");

     if (UI.current != null) UI.current.DrawAfter( new Action( ()=>GraphWindow.Show(graph) ) ); //if opened v
     else Show(graph);
     return true;
    }
    if (obj is MapMagicObject) { GraphWindow.Show(((MapMagicObject)obj).graph); return true; }
    return false;
   }

  #endregion
 }

}//namespace
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;



namespace MapMagic.Nodes.GUI
{
 public static class GroupDraw
 {
  private static Generator[] draggedGroupNodes;
  private static Vector2[] initialGroupNodesPos;
  private static GUIStyle miniNameStyle;

  public static void DragGroup (Group group, Generator[] allGens=null)
  {
   //dragging
   if (!UI.current.layout)
   {
    if (DragDrop.TryDrag(group, UI.current.mousePos))
    {
```

```
 for (int i=0; i<draggedGroupNodes.Length; i++)
 {
  Generator gen = draggedGroupNodes[i];

  gen.guiPosition = initialGroupNodesPos[i] + DragDrop.totalDelta;

  //moving generators cells
  if (UI.current.cellObjs.TryGetCell(gen, "Generator", out Cell genCell))
  {
   genCell.worldPosition = gen.guiPosition;
   genCell.CalculateSubRects();
  }
 }

 group.guiPos = DragDrop.initialRect.position + DragDrop.totalDelta;
 Cell.current.worldPosition = group.guiPos;
 Cell.current.CalculateSubRects();
}

if (DragDrop.TryRelease(group, UI.current.mousePos))
{
 draggedGroupNodes = null;
 initialGroupNodesPos = null;

 #if UNITY_EDITOR //this should be an editor script, but it doesnt mentioned anywhere
 UnityEditor.EditorUtility.SetDirty(GraphWindow.current.graph);
```

```
   #endif
  }

  if (DragDrop.TryStart(group, UI.current.mousePos, Cell.current.InternalRect))
  {
   draggedGroupNodes = GetContainedGenerators(group, allGens);

   initialGroupNodesPos = new Vector2[draggedGroupNodes.Length];
   for (int i=0; i<draggedGroupNodes.Length; i++)
    initialGroupNodesPos[i] = draggedGroupNodes[i].guiPosition;
  }

  Rect cellRect = Cell.current.InternalRect;
  if (DragDrop.ResizeRect(group, UI.current.mousePos, ref cellRect, minSize:new Vector2(100,100)))
  {
   group.guiPos = cellRect.position;
   group.guiSize = cellRect.size;

   Cell.current.InternalRect = cellRect;
   Cell.current.CalculateSubRects();
  }
 }
}

public static void DrawGroup (Group group, bool isMini=false)
{
```

```
float miniFactor = !isMini ? 1 : 0.5f/GraphWindow.miniZoom; //group controls are slightly smaller than in f

//CellObject.SetObject(Cell.current, group);
if (UI.current.layout)
 UI.current.cellObjs.ForceAdd(group, Cell.current, "Group");

Texture2D tex = UI.current.textures.GetColorizedTexture("MapMagic/Group", group.color);
GUIStyle style = UI.current.textures.GetElementStyle(tex);
Draw.Element(style);

Cell.EmptyRowPx(5*miniFactor);
using (Cell.Row)
{
 if (isMini && miniNameStyle == null)
 {
  miniNameStyle = new GUIStyle(UI.current.styles.bigLabel);
  miniNameStyle.fontSize = (int)(miniNameStyle.fontSize/GraphWindow.miniZoom*0.6f);
 }

 Cell.EmptyLinePx(5*miniFactor);
 GUIStyle labelStyle = !isMini ? UI.current.styles.bigLabel : miniNameStyle;
 using (Cell.LinePx(24*miniFactor)) Draw.EditableLabelRight(ref group.name, style:labelStyle);
 //using (Cell.LineStd) Draw.EditableLabelRight(ref group.comment);
 Cell.EmptyLinePx(3*miniFactor);
 using (Cell.LineStd)
 {
```

```
   using (Cell.RowPx(0))
   {
    if (!isMini)
    {
     using (Cell.RowPx(20)) Draw.Icon(UI.current.textures.GetTexture("DPUI/Chevrons/Down"), scale:0.5f*
     using (Cell.RowPx(35)) Draw.Label("Color");
    }

    if (Draw.Button("", visible:false)) GroupRightClick.DrawGroupColorSelector(group);
   }
  }
 }
}



public static Generator[] GetContainedGenerators (Group group, Generator[] all)
/// Removes dragged-off gens and adds new ones
{
 List<Generator> generators = new List<Generator>();
 Rect rect = new Rect(group.guiPos, group.guiSize);
 for (int g=0; g<all.Length; g++)
 {
  if (!rect.Contains(all[g].guiPosition, all[g].guiSize)) continue;
  generators.Add(all[g]);
 }
```

```
   return generators.ToArray();
  }



  public static Generator[] GetContainedGenerators (Group group, Graph graph)
   { return GetContainedGenerators(group, graph.generators); }



  public static void RemoveGroupContents (Group group, Graph graph)
  /// Called from editor. Removes the enlisted generators on group remove
  {
   GraphWindow.RecordCompleteUndo();

   Generator[] containedGens = GetContainedGenerators(group, graph.generators);

   for (int i=0; i<containedGens.Length; i++)
    graph.Remove(containedGens[i]);
  }
 }
}
```

```
ï»¿using System;
using UnityEngine;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Nodes;

namespace MapMagic.Nodes.GUI
{
 public static class LevelsDraw
 {
  public const float diamondHeight = 15;
  public const int uiWidth = 140;
  public const int uiCurveHeight = 70;

  public static Vector3[] posArray = new Vector3[20];


  public static void DrawLevels (
   ref float inMin, ref float inMax,
   ref float gamma, //1-2
   ref float outMin, ref float outMax,
   float[] histogram = null)
  {
   //curve
   using (Cell.Line)
```

```
{
 using (Cell.RowPx(diamondHeight))
 {
  Texture2D minTex = UI.current.textures.GetTexture("DPUI/Levels/SliderVerBlack");
  outMin = VerDiamond(outMin, minTex);
 }

 //main field
 using (Cell.Row)
  if (!UI.current.layout && !(UI.current.optimizeElements && !UI.current.IsInWindow()))
 {
  Draw.Grid(new Color(0,0,0,0.5f));

  if (histogram != null)
  {
   Material histogramMat = UI.current.textures.GetMaterial("Hidden/DPLayout/Histogram");
   histogramMat.SetFloatArray("_Histogram", histogram);
   histogramMat.SetVector("_Backcolor", new Vector4(0,0,0,0));
   histogramMat.SetVector("_Forecolor", new Vector4(0,0,0,0.25f));
   Draw.Texture(null, histogramMat);
  }

  DrawCurve(inMin, inMax, gamma, outMin, outMax, posArray);
  //CheckCurve(min, max, gamma);
 }
```

```
  //right slider
  using (Cell.RowPx(diamondHeight))
  {
   Texture2D maxTex = UI.current.textures.GetTexture("DPUI/Levels/SliderVerWhite");
   outMax = VerDiamond(outMax, maxTex);
  }
 }

 //bottom sliders
 using (Cell.LinePx(diamondHeight))
 {
  Cell.EmptyRowPx(diamondHeight); //reserved for left slider

  using (Cell.Row)
   BottomDiamonds(ref inMin, ref inMax, ref gamma);

  Cell.EmptyRowPx(diamondHeight); //reserved for right slider
 }
}


public static void BottomDiamonds (ref float min, ref float max, ref float gamma)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;
```

```
float mid = min*(1-gamma/2f) + max*(gamma/2f); //in percent, 0-1
float origMid = mid; //to track change and convert mid to gamma on change

//preparing textures
Texture2D minTex = UI.current.textures.GetTexture("DPUI/Levels/SliderBlack");
Texture2D midTex = UI.current.textures.GetTexture("DPUI/Levels/SliderGray");
Texture2D maxTex = UI.current.textures.GetTexture("DPUI/Levels/SliderWhite");

min = HorDiamond(min, minTex, clickStart:0, clickEnd:(min+mid)/2, min:0, max:max);
mid = HorDiamond(mid, midTex, clickStart:(min+mid)/2, clickEnd:(mid+max)/2, min:min, max:max, def:(m
max = HorDiamond(max, maxTex, clickStart:(mid+max)/2, clickEnd:1, min:min, max:1);

//converting mid to gamma
if (mid>origMid+0.00001f || mid<origMid-0.00001f)
{
 UI.current.MarkChanged();

 if (max-min!=0) gamma = (mid-min)/(max-min)*2;
 else gamma = 1;

 gamma = ((int)(gamma*1000))/1000f;
}

//cursor
Rect cellRect = Cell.current.InternalRect;
cellRect.x -= minTex.width/2; cellRect.width += minTex.width;
```

```
 Rect dispRect = UI.current.scrollZoom.ToScreen(cellRect);
 UnityEditor.EditorGUIUtility.AddCursorRect(dispRect, UnityEditor.MouseCursor.MoveArrow);
}


public static void RightDiamonds (ref float min, ref float max)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Texture2D minTex = UI.current.textures.GetTexture("DPUI/Levels/SliderVerBlack");
 Texture2D maxTex = UI.current.textures.GetTexture("DPUI/Levels/SliderVerWhite");

 min = VerDiamond(min, minTex, clickStart:0, clickEnd:(min+max)/2, min:0, max:max);
 max = VerDiamond(max, maxTex, clickStart:(min+max)/2, clickEnd:1, min:min, max:1);

 //cursor
 Rect cellRect = Cell.current.InternalRect;
 cellRect.y -= minTex.height/2; cellRect.height += minTex.height;
 Rect dispRect = UI.current.scrollZoom.ToScreen(cellRect);
 UnityEditor.EditorGUIUtility.AddCursorRect(dispRect, UnityEditor.MouseCursor.MoveArrow);
}


private static float HorDiamond (
 float val,
```

```
 Texture2D texture,
 float clickStart, float clickEnd, //clicking between this values will drag the diamond
 float min=0, float max=1,
 float def=0, bool removable=false, //default when diamond is removed
 bool trackChange = true)
{
 //if (UI.current.layout) return val;
 //if (UI.current.optimizeElements && !UI.current.IsInWindow()) return val;
 //in BottomDiamonds

 Rect cellRect = Cell.current.InternalRect;
 Vector2 cellPos = cellRect.position; //somehow differes from Cell.current.worldPosition
 Vector2 cellSize = cellRect.size;

 Rect clickRect = new Rect (0, cellRect.y, 0, cellRect.height);
 clickRect.xMin = cellPos.x + cellSize.x*clickStart;
 clickRect.xMax = cellPos.x + cellSize.x*clickEnd;

 //appending click rects for Min and Max diamonds. Hacky, can live without it
 if (min < 0.0001f) clickRect.xMin -= texture.width/2;
 if (max > 0.999f) clickRect.xMax += texture.width/2;

 (Cell cell, Texture2D tex) dragObj = (Cell.current, texture); //using a temporary object to drag

 if (DragDrop.TryDrag(dragObj, UI.current.mousePos))
 {
```

```
 DragDrop.group = "DragLevels";

 float newVal = (UI.current.mousePos.x - cellPos.x) / cellSize.x;

 //removing
 if (removable && !cellRect.Extended(20).Contains(UI.current.mousePos))
  newVal = def;

 //clamping
 float pixelSize = 1f / cellSize.x;
 if (newVal < min+pixelSize) newVal = min+pixelSize;
 if (newVal > max-pixelSize) newVal = max-pixelSize;

 newVal = ((int)(newVal*1000))/1000f;

 if (trackChange && (newVal>val+0.00001f || newVal<val-0.00001f))
 {
  UI.current.MarkChanged();
  val = newVal;
 }
}

DragDrop.TryRelease(dragObj, UI.current.mousePos);

DragDrop.TryStart(dragObj, UI.current.mousePos, clickRect);
```

```
 //icon
 Vector2 iconPos = new Vector2(
  cellPos.x + val*cellSize.x,
  //cellPos.y + texture.height/2);
  cellPos.y + cellSize.y/2);
 Draw.Icon(texture, iconPos);

 //cursor
 cellRect.x -= texture.width/2; cellRect.width += texture.width;
 Rect dispRect = UI.current.scrollZoom.ToScreen(cellRect);
 UnityEditor.EditorGUIUtility.AddCursorRect(dispRect, UnityEditor.MouseCursor.MoveArrow);

 //cursor (in BottomDiamonds)
 //cellRect.y -= texture.height/2; cellRect.height += texture.height;
 //Rect dispRect = UI.current.scrollZoom.ToScreen(cellRect);
 //UnityEditor.EditorGUIUtility.AddCursorRect(dispRect, UnityEditor.MouseCursor.MoveArrow);

 return val;
}


private static float VerDiamond (
 float val,
 Texture2D texture,
 float clickStart=0, float clickEnd=1, //clicking between this values will drag the diamond. Clicking outside -
 float min=0, float max=1,  //values that could be passed over while dragging. Not used
```

```
 float def=0, bool removable=false) //default when diamond is removed
{
 if (UI.current.layout) return val;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return val;

 Rect cellRect = Cell.current.InternalRect;
 Vector2 cellPos = cellRect.position; //somehow differes from Cell.current.worldPosition
 Vector2 cellSize = cellRect.size;

 Rect clickRect = new Rect (cellRect.x, 0, cellRect.width, 0);
 clickRect.yMin = cellPos.y + cellSize.y*(1-clickEnd);
 clickRect.yMax = cellPos.y + cellSize.y*(1-clickStart);

 //appending click rects for Min and Max diamonds. Hacky, can live without it
 if (min < 0.999f) clickRect.yMin -= texture.height/2;
 if (max > 0.0001f) clickRect.yMax += texture.height/2;

 (Cell cell, Texture2D tex) dragObj = (Cell.current, texture); //using a temporary object to drag

 if (DragDrop.TryDrag(dragObj, UI.current.mousePos))
 {
  float newVal = 1 - (UI.current.mousePos.y - cellPos.y) / cellSize.y;

  //removing
  if (removable && !cellRect.Extended(20).Contains(UI.current.mousePos))
   newVal = def;
```

```
 //clamping
 float pixelSize = 1f / cellSize.y;
 if (newVal < min+pixelSize) newVal = min+pixelSize;
 if (newVal > max-pixelSize) newVal = max-pixelSize;

 newVal = ((int)(newVal*1000))/1000f;

 if (newVal>val+0.00001f || newVal<val-0.00001f)
 {
  UI.current.MarkChanged();
  val = newVal;
 }
}

DragDrop.TryRelease(dragObj, UI.current.mousePos);

DragDrop.TryStart(dragObj, UI.current.mousePos, clickRect);

//icon
Vector2 iconPos = new Vector2(
 //cellPos.x + texture.width/2,
 cellPos.x + cellSize.x/2,
 cellPos.y + (1-val)*cellSize.y);
Draw.Icon(texture, iconPos);
```

```
 //cursor
 cellRect.y -= texture.height/2; cellRect.height += texture.height;
 Rect dispRect = UI.current.scrollZoom.ToScreen(cellRect);
 UnityEditor.EditorGUIUtility.AddCursorRect(dispRect, UnityEditor.MouseCursor.MoveArrow);

 return val;
}

private static void Diamond (
 ref float val, //in percent, 0-1
 Texture2D texture,
 float min=0,
 float max=1,
 float def=0, //default when diamond is removed
 bool removable=false)
{
 if (UI.current.layout) return;

 Rect rect = new Rect(
  Cell.current.worldPosition.x + Cell.current.finalSize.x*val - texture.width/2,
  Cell.current.worldPosition.y + Cell.current.finalSize.y/2 - texture.height/2,
  texture.width,
  texture.height);

 //drag
```

```
DiamondDragObj dragObj = new DiamondDragObj(Cell.current, texture); //using a temporary object to dr



if (DragDrop.TryDrag(dragObj, UI.current.mousePos))
{
 float newVal = (DragDrop.initialRect.center.x + DragDrop.totalDelta.x - Cell.current.worldPosition.x) / Ce

 //removing
 if (removable)
 {
  Rect cellRect = Cell.current.InternalRect.Extended(20);
  if (!cellRect.Contains(UI.current.mousePos))
   newVal = def;
 }

 //clamping
 if (newVal < min) newVal = min;
 if (newVal > max) newVal = max;

 if (newVal != val)
 {
  UI.current.MarkChanged();
  val = newVal;
 }

 //refreshing diamond pos if dragging
```

```
  rect.x = Cell.current.worldPosition.x + Cell.current.finalSize.x*val - texture.width/2;
 }

 DragDrop.TryRelease(dragObj, UI.current.mousePos);

 DragDrop.TryStart(dragObj, UI.current.mousePos, rect);


 //icon
 Draw.Icon(texture, center:rect.center);
}


private static void VerticalDiamond (
 ref float val, //in percent, 0-1
 Texture2D texture,
 float min=0,
 float max=1,
 float def=0, //default when diamond is removed
 bool removable=false)
{
 if (UI.current.layout) return;

 Rect rect = new Rect(
  Cell.current.worldPosition.x + Cell.current.finalSize.x/2 - texture.width/2,
  Cell.current.worldPosition.y + Cell.current.finalSize.y*(1-val) - texture.height/2,
```

```
 texture.width,
 texture.height);

//drag
//DiamondDragObj dragObj = new DiamondDragObj(Cell.current, texture); //using a temporary object to c
Cell dragObj = Cell.current; //only one slider per cell

if (DragDrop.TryDrag(dragObj, UI.current.mousePos))
{
 float newVal = (DragDrop.initialRect.center.y + DragDrop.totalDelta.y - Cell.current.worldPosition.y) / Ce
 newVal = 1 - newVal;

 //removing
 if (removable)
 {
  Rect cellRect = Cell.current.InternalRect.Extended(20);
  if (!cellRect.Contains(UI.current.mousePos))
   newVal = def;
 }

 //clamping
 if (newVal < min) newVal = min;
 if (newVal > max) newVal = max;

 if (newVal != val)
 {
```

```
   UI.current.MarkChanged();
   val = newVal;
  }

  //refreshing diamond pos if dragging
  rect.y = Cell.current.worldPosition.y + Cell.current.finalSize.y*(1-val) - texture.height/2;
 }

 DragDrop.TryRelease(dragObj, UI.current.mousePos);

 DragDrop.TryStart(dragObj, UI.current.mousePos, rect);


 //icon
 Draw.Icon(texture, center:rect.center);
}


struct DiamondDragObj
{
 public Cell cell;
 public Texture2D texture;
 public DiamondDragObj (Cell cell, Texture2D texture) { this.cell=cell; this.texture=texture; }
}
```

```
private static void DrawCurve (
 float inMin, float inMax,  //in percent, 0-1
 float gamma, //0-2, 1 is empty
 float outMin, float outMax,
 Vector3[] posArray = null)
{
 if (posArray == null) posArray = new Vector3[5];

 for (int i=1; i<posArray.Length-1; i++)
 {
  float p = 1f * (i-1) / (posArray.Length-3);
  p = 3*p*p - 2*p*p*p;

  float x = inMin + p*(inMax-inMin);
  float y = LevelsFn(x, inMin, inMax, gamma, outMin, outMax);

  x = Cell.current.worldPosition.x + x*(Cell.current.finalSize.x-1);
  y = Cell.current.worldPosition.y + (1-y)*(Cell.current.finalSize.y-1) + 1;

  posArray[i] = UI.current.scrollZoom.ToScreen( new Vector2(x,y) );
 }

 posArray[0] = UI.current.scrollZoom.ToScreen( new Vector2(Cell.current.worldPosition.x, Cell.current.wo
 posArray[posArray.Length-1] = UI.current.scrollZoom.ToScreen( new Vector2(Cell.current.worldPosition.

 UnityEditor.Handles.color = Color.black;
```

```
 UnityEditor.Handles.DrawAAPolyLine(2, posArray);
}

private static void CheckCurve (
 float inMin, float inMax, float gamma, float outMin, float outMax,
 Cell cell=null)
{
 Vector3[] posArray = new Vector3[100];
 for (int i=0; i<posArray.Length; i++)
 {
  float x = 1f*i / posArray.Length;
  float y = LevelsFn(x, inMin, inMax, gamma, outMin, outMax);

  x = Cell.current.worldPosition.x + x*(Cell.current.finalSize.x-1);
  y = Cell.current.worldPosition.y + (1-y)*(Cell.current.finalSize.y-1) + 1;

  posArray[i] = UI.current.scrollZoom.ToScreen( new Vector2(x,y) );
 }

 UnityEditor.Handles.color = Color.red;
 UnityEditor.Handles.DrawAAPolyLine(2, posArray);
}

private static float LevelsFn (float val, float inMin, float inMax, float gamma, float outMin, float outMax)
/// Copy of Matrix.Levels (TODO: use shader, matrix and it's array, and call matrix.Levels)
{
```

```
//preliminary clamping
if (val < inMin) return outMin;
if (val > inMax) return outMax;

//input
float inDelta = inMax - inMin;
if (inDelta != 0)
 val = (val-inMin) / inDelta;
else
 val = inMin;

//gamma
if (gamma>1.00001f || gamma<0.9999f)  // gamma != 1
{
 if (gamma<1) val = Mathf.Pow(val, gamma);
 else val = Mathf.Pow(val, 1/(2-gamma));
}

//output
float outDelta = outMax - outMin;
if (outDelta != 0)
 val = outMin + val * outDelta;
else
 val = outMin;

return val;
```

```
  }
 }
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.GUI;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes.GUI;
//using MapMagic.Nodes.ObjectsGenerators; //not in objects anymore


/// Some object editor fns are brought to a separate file since they are needed by Brush module outputs

namespace MapMagic.Nodes.GUI
{
 public static partial class ObjectsEditors
 {
  public static void DrawObjectPrefabs (ref GameObject[] prefabs, bool multiPrefab=true, bool treeIcon=fals
  {
   string iconName = treeIcon ?  "DPUI/Icons/TreeDisabled" : "DPUI/Icons/ObjectDisabled";

   if (multiPrefab)
```

```
 using (Cell.LineStd)
 {
  GameObject[] prefabsCopy = prefabs; //TODO: Action not taking ref. The layer should have onDraw fun
  LayersEditor.DrawLayers(ref prefabs, onDraw: n => DrawObjectPrefabLayer(prefabsCopy,n,iconName)
 }
else
{
 if (prefabs.Length != 1) Array.Resize(ref prefabs, 1);

 Cell.EmptyLinePx(4);
 using (Cell.LineStd)
 {
  using (Cell.RowPx(24)) Draw.Icon( UI.current.textures.GetTexture(iconName) );
  Cell.EmptyRowPx(4);
  using (Cell.Row) Draw.Field(ref prefabs[0]);
  Cell.EmptyRowPx(4);
 }
}

Cell.EmptyLinePx(2);
}

private static void DrawObjectPrefabLayer (GameObject[] prefabs, int n, string iconName)
{
 if (n>=prefabs.Length) return; //on layer remove
 Cell.EmptyLinePx(4);
```

```
   using (Cell.LineStd)
   {
    using (Cell.RowPx(24)) Draw.Icon( UI.current.textures.GetTexture(iconName) );
    using (Cell.Row) prefabs[n] = Draw.ObjectField(prefabs[n]);
    Cell.EmptyRowPx(4);
   }
   Cell.EmptyLinePx(4);
  }

  public static void DrawPositioningSettings (PositioningSettings posSettings, bool billboardRotWaring=fals
  {
   //height
   Cell.EmptyLinePx(1);
   using (Cell.LinePx(0))
    using (new Draw.FoldoutGroup(ref posSettings.guiHeight, "Height"))
     if (posSettings.guiHeight)
     {
      using (Cell.LineStd) Draw.ToggleLeft(ref posSettings.objHeight, "Object Height");
//      if (showRelativeHeight)
       using (Cell.LineStd) Draw.ToggleLeft(ref posSettings.relativeHeight, "Relative Height");
     }

   //rotation
   Cell.EmptyLinePx(1);
   using (Cell.LinePx(0))
    using (new Draw.FoldoutGroup(ref posSettings.guiRotation, "Rotation"))
```

```
  if (posSettings.guiRotation)
  {
   using (Cell.LineStd) Draw.ToggleLeft(ref posSettings.useRotation, "Use Rotation");
   using (Cell.LineStd) Draw.ToggleLeft(ref posSettings.takeTerrainNormal, "Terrain Normal");
   using (Cell.LineStd)
   {
    Cell.current.disabled = posSettings.takeTerrainNormal;
    Draw.ToggleLeft(ref posSettings.rotateYonly, "Rotate Y Only"); //
   }
   using (Cell.LineStd) Draw.ToggleLeft(ref posSettings.regardPrefabRotation, "Use Prefab Rot.");

   if (billboardRotWaring)
   {
    using (Cell.LinePx(40))
     Draw.Helpbox("Note that Unity billboard trees could not be rotated");
    Cell.EmptyLinePx(2);
   }
  }

//scale
Cell.EmptyLinePx(1);
using (Cell.LinePx(0))
 using (new Draw.FoldoutGroup(ref posSettings.guiScale, "Scale"))
  if (posSettings.guiScale)
  {
   using (Cell.LineStd) Draw.ToggleLeft(ref posSettings.useScale, "Use Scale");
```

```
      using (Cell.LineStd) Draw.ToggleLeft(ref posSettings.scaleYonly, "Scale Y Only");
      using (Cell.LineStd) Draw.ToggleLeft(ref posSettings.regardPrefabScale, "Use Prefab Scale");
      //using (Cell.LineStd)
      //{
      // Cell.EmptyRowPx(18);
      // using (Cell.Row) Draw.Label("Scale");
      //}
     }
  }
 }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;
using UnityEditor;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.GUI.Popup;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Nodes.GUI;

namespace MapMagic.Nodes.GUI
{
 public static class CreateRightClick
 {
  public static HashSet<Type> generatorTypes = new HashSet<Type>();
  //public so addons could enlist themselves on initialize


  public static void DrawCreateItems (Vector2 mousePos, Graph graph)
   { DrawItem( CreateItems(mousePos, graph) ); }
```

```
public static void DrawInsertItems (Vector2 mousePos, Graph graph, IInlet<object> clickedLink)
 { DrawItem( InsertItems(mousePos, graph, clickedLink) ); }


public static void DrawAppendItems (Vector2 mousePos, Graph graph, IOutlet<object> clickedOutlet)
 { DrawItem( AppendItems(mousePos, graph, clickedOutlet) ); }


private static void DrawItem (Item item)
{
 #if MM_EXP || UNITY_2020_1_OR_NEWER || UNITY_EDITOR_LINUX
 SingleWindow menu = new SingleWindow(item);
 #else
 PopupMenu menu = new PopupMenu() {items=item.subItems, minWidth=150};
 #endif

 menu.Show(Event.current.mousePosition);
}


public static Item CreateItems (Vector2 mousePos, Graph graph, int priority=5)
{
 Item create = new Item("Add (Create)");
 create.onDraw = RightClick.DrawItem;
 create.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Create");
```

```
create.color = RightClick.defaultColor;
create.subItems = new List<Item>();
create.priority = priority;

//automatically adding generators from this assembly
Type[] types = typeof(Generator).Subtypes();
for (int t=0; t<types.Length; t++)
 if (!generatorTypes.Contains(types[t])) generatorTypes.Add(types[t]);

//adding outer-assembly types
//via their initialize

//creating unsorted create items
foreach (Type type in generatorTypes)
{
 GeneratorMenuAttribute attribute = GeneratorDraw.GetMenuAttribute(type);
 if (attribute == null) continue;

 string texPath = attribute.iconName ?? "MapMagic/Popup/Standard";
 string texName = texPath;
 //if (StylesCache.isPro) texName += "_icon";

 Item item = new Item( ) {
  name = (attribute.menuName!=null && attribute.menuName.Length!=0) ? attribute.menuName : attribut
  onDraw = RightClick.DrawItem,
  icon = RightClick.texturesCache.GetTexture(texPath, texName),
```

```
 color = GeneratorDraw.GetGeneratorColor(attribute.colorType ?? Generator.GetGenericType(type)),
 onClick = ()=> GraphEditorActions.CreateGenerator(graph, type, mousePos),
 priority = attribute.priority };

//moving into the right section using priority
//int sectionPriority = 10000 - attribute.section*1000;
//item.priority += sectionPriority;



//placing items in categories
string catName = attribute.menu;
if (catName == null) continue; //if no 'menu' defined this generator could not be created
string[] catNameSplit = catName.Split('/');

Item currCat = create;
if (catName != "")  //if empty menu is defined using root category
 for (int i=0; i<catNameSplit.Length; i++)
 {
  //trying to find category
  bool catFound = false;
  if (currCat.subItems != null)
   foreach (Item sub in currCat.subItems)
   {
    if (sub.onClick == null  &&  sub.name == catNameSplit[i])
    {
     currCat = sub;
```

```
      catFound = true;
      break;
     }
    }

   //creating if not found
   if (!catFound)
   {
    Item newCat = new Item(catNameSplit[i]);
    if (currCat.subItems == null) currCat.subItems = new List<Item>();
    currCat.subItems.Add(newCat);
    currCat = newCat;

    newCat.color = item.color;
   }
  }

 if (currCat.subItems == null) currCat.subItems = new List<Item>();
 currCat.subItems.Add(item);
}

//default sorting order
foreach (Item item in create.All(true))
{
 if (item.name == "Map" && item.onClick==null)
 {
```

```
 item.priority = 10004;
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Map");
 item.color = GeneratorDraw.GetGeneratorColor(typeof(Den.Tools.Matrices.MatrixWorld));
}
if (item.name == "Objects" && item.onClick==null)
{
 item.priority = 10003;
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Objects");
 item.color = GeneratorDraw.GetGeneratorColor(typeof(TransitionsList));
}
if (item.name == "Spline" && item.onClick==null)
{
 item.priority = 10002;
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Spline");
 item.color = GeneratorDraw.GetGeneratorColor(typeof(Den.Tools.Splines.SplineSys));
}
if (item.name == "Biomes")
{
 item.priority = 9999;
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Biomes");
 item.color = GeneratorDraw.GetGeneratorColor(typeof(IBiome));
}
if (item.name == "Functions")
{
 item.priority = 9999;
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Function");
```

```
  item.color = GeneratorDraw.GetGeneratorColor(typeof(IBiome));
 }

 if (item.name == "Enter"  && item.onClick==null) { item.icon = RightClick.texturesCache.GetTexture("Ge
 if (item.name == "Exit"  && item.onClick==null) { item.icon = RightClick.texturesCache.GetTexture("Gene

 if (item.name == "Initial") { item.priority = 10009; item.icon = RightClick.texturesCache.GetTexture("MapI
 if (item.name == "Modifiers") { item.priority = 10008; item.icon = RightClick.texturesCache.GetTexture("M
 if (item.name == "Standard") { item.priority = 10009; item.icon = RightClick.texturesCache.GetTexture("M
 if (item.name == "Output") { item.priority = 10007; item.icon = RightClick.texturesCache.GetTexture("Map
 if (item.name == "Outputs") { item.priority = 10006; item.icon = RightClick.texturesCache.GetTexture("Ma
 if (item.name == "Input") { item.priority = 10005; item.icon = RightClick.texturesCache.GetTexture("MapM
 if (item.name == "Inputs") { item.priority = 10004; item.icon = RightClick.texturesCache.GetTexture("Map
 if (item.name == "Portals") { item.priority = 10003; item.icon = RightClick.texturesCache.GetTexture("Ma
 if (item.name == "Function") { item.priority = 10002; item.icon = RightClick.texturesCache.GetTexture("M

 if (item.name == "Height") item.priority = 10003;
 if (item.name == "Textures") item.priority = 10002;
 if (item.name == "Grass") item.priority = 10001;

 if (item.onDraw == null) item.onDraw = RightClick.DrawItem;
}

//adding separator between standard and special categories
if (create.subItems.FindIndex(i=>i.name=="Biomes") >= 0) //add separator if biomes item present
{
```

```
  Item separator = Item.Separator(priority:10001);
  separator.onDraw = RightClick.DrawSeparator;
  separator.color =  RightClick.defaultColor;
  create.subItems.Add(separator);
 }

 return create;
}



public static Item InsertItems (Vector2 mousePos, Graph graph, IInlet<object> inlet, int priority=4)
{
 IOutlet<object> outlet = graph.GetLink(inlet);
 Item insertItems;

 if (inlet != null  &&  outlet != null)
 {
  Type genericLinkType = Generator.GetGenericType(inlet);

  Item createItems = CreateItems(mousePos, graph);
  Item catItems = createItems.Find( GetCategoryByType(genericLinkType) );
  insertItems = catItems.Find("Modifiers");

  //adding link to all create actions
  foreach(Item item in insertItems.All(true))
   if (item.onClick != null)
```

```
  {
   Action baseOnClick = item.onClick;
   item.onClick = () =>
   {
    baseOnClick();

    Generator createdGen = graph.generators[graph.generators.Length-1]; //the last on is the one that's ju
    if (createdGen!=null  &&  Generator.GetGenericType((Generator)createdGen) == genericLinkType)
    {
     //inlet
     graph.AutoLink(createdGen, outlet);

     //outlet
     if (createdGen is IOutlet<object> createdOutlet)
      graph.Link(createdOutlet, inlet);
    }

    GraphWindow.current?.RefreshMapMagic();
   };
  }
}
else
{
 insertItems = new Item("Add");
 insertItems.onDraw = RightClick.DrawItem;
 insertItems.disabled = true;
```

```
 }

 insertItems.name = "Add (Insert)";
 insertItems.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Create");
 insertItems.color =  RightClick.defaultColor;
 insertItems.priority = priority;
 return insertItems;
}


public static Item AppendItems (Vector2 mousePos, Graph graph, IOutlet<object> clickedOutlet, int priorit
/// Item set appeared on node or outlet click
{
 Item addItems = null;

 if (clickedOutlet != null)
 {
  Type genericLinkType = Generator.GetGenericType(clickedOutlet);
  if (genericLinkType==null)
   throw new Exception("Could not find category " + clickedOutlet.GetType().ToString());

  Item createItems = CreateItems(mousePos, graph);
  addItems = createItems.Find( GetCategoryByType(genericLinkType) );
 }

 if (addItems != null  &&  addItems.subItems != null)
```

```
{
 Item initial = addItems.Find("Initial");
 if (initial != null) initial.disabled = true;

 //adding link to all create actions
 foreach(Item item in addItems.All(true))
  if (item.onClick != null)
  {
   Action baseOnClick = item.onClick;
   item.onClick = () =>
   {
    baseOnClick();

    Generator createdGen = graph.generators[graph.generators.Length-1]; //the last on is the one that's ju

    Vector2 pos = clickedOutlet.Gen.guiPosition + new Vector2(200, 0);
    GeneratorDraw.FindPlace(ref pos, new Vector2(100,200), GraphWindow.current.graph);
    createdGen.guiPosition = pos;

    graph.AutoLink(createdGen, clickedOutlet);

    GraphWindow.current?.RefreshMapMagic();
   };
  }
}
else
```

```
  {
   addItems = new Item("Add");
   addItems.onDraw = RightClick.DrawItem;
   addItems.disabled = true;
  }

  addItems.name = "Add (Append)";
  addItems.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Create");
  addItems.color =  RightClick.defaultColor;
  addItems.priority = priority;

  return addItems;
 }



 private static string GetCategoryByType (Type genericType)
 {
  if (genericType == typeof(Den.Tools.Matrices.MatrixWorld)) return "Map";
  else if (genericType == typeof(TransitionsList)) return "Objects";
  //else if (genericType == typeof(Den.Tools.Segs.SplineSys)) return "Spline";
  else if (genericType == typeof(Den.Tools.Splines.SplineSys)) return "Spline";
  return null;
 }
}
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;
using UnityEditor;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.GUI.Popup;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Nodes.GUI;

namespace MapMagic.Nodes.GUI
{
 public static class GeneratorRightClick
 {
  public static Graph copiedGenerators;

  public static Item GeneratorItems (Vector2 mousePos, Generator gen, Graph graph, int priority=3)
  {
   Item genItems = new Item("Generator");
   genItems.onDraw = RightClick.DrawItem;
   genItems.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Generator");
```

```
genItems.color = RightClick.defaultColor;
genItems.subItems = new List<Item>();
genItems.priority = priority;

genItems.disabled = gen==null  &&  copiedGenerators==null;

{ //enable/disable
 string caption = (gen==null||gen.enabled) ? "Disable" : "Enable";
 Item item = new Item(caption, onDraw:RightClick.DrawItem, priority:11);
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Eye");
 item.color = RightClick.defaultColor;
 item.disabled = gen==null;
 item.onClick = ()=>
  GraphEditorActions.EnableDisableGenerators(graph, GraphWindow.current.selected, gen);
 genItems.subItems.Add(item);
}

//genItems.subItems.Add( new Item("Export", onDraw:RightClick.DrawItem, priority:10) { icon = RightClic
//genItems.subItems.Add( new Item("Import", onDraw:RightClick.DrawItem, priority:9) { icon = RightClick.

{ //duplicate
 Item item = new Item("Duplicate", onDraw:RightClick.DrawItem, priority:8);
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Duplicate");
 item.color = RightClick.defaultColor;
 item.disabled = gen==null;
 item.onClick = ()=>
```

```
  GraphEditorActions.DuplicateGenerator(graph, gen, ref GraphWindow.current.selected);
 genItems.subItems.Add(item);
}



{ //copy
 Item item = new Item("Copy", onDraw:RightClick.DrawItem, priority:8);
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Export");
 item.color = RightClick.defaultColor;
 item.disabled = !(gen!=null || (GraphWindow.current.selected!=null && GraphWindow.current.selected.C
 item.onClick = ()=>
 {
  HashSet<Generator> gens;
  if (GraphWindow.current.selected!=null && GraphWindow.current.selected.Count!=0)
   gens = GraphWindow.current.selected;
  else
   { gens = new HashSet<Generator>(); gens.Add(gen); }
  copiedGenerators = graph.Export(gens);
 };
 item.closeOnClick = true;
 genItems.subItems.Add(item);
}



{ //paste
 Item item = new Item("Paste", onDraw:RightClick.DrawItem, priority:7);
```

```
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Export");
 item.color = RightClick.defaultColor;
 item.disabled = copiedGenerators==null;
 item.onClick = ()=>
 {
  Generator[] imported = graph.Import(copiedGenerators);
  Graph.Reposition(imported, mousePos);
 };
 item.closeOnClick = true;
 genItems.subItems.Add(item);
}


{ //update
 Item item = new Item("Update", onDraw:RightClick.DrawItem, priority:7);
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Update");
 item.color = RightClick.defaultColor;
 item.closeOnClick = true;
 item.disabled = gen==null;
}


{ //reset
 Item item = new Item("Reset", onDraw:RightClick.DrawItem, priority:4);
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Reset");
 item.color = RightClick.defaultColor;
```

```
 item.closeOnClick = true;
 item.disabled = gen==null;
}


{ //remove
 Item item = new Item("Remove", onDraw:RightClick.DrawItem, priority:5);
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Remove");
 item.color = RightClick.defaultColor;
 item.disabled = gen==null;
 item.onClick = ()=>
  GraphEditorActions.RemoveGenerators(graph, GraphWindow.current.selected, gen);
 item.closeOnClick = true;
 genItems.subItems.Add(item);
}


{ //unlink
 Item item = new Item("Unlink", onDraw:RightClick.DrawItem, priority:6);
 item.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Unlink");
 item.color = RightClick.defaultColor;
 item.disabled = gen==null;
 item.onClick = ()=>
 {
  graph.UnlinkGenerator(gen);
  //undo
```

```
 };
 item.closeOnClick = true;
 genItems.subItems.Add(item);
}

if (gen!=null)
{ //help
 Item item = new Item($"Help", onDraw:RightClick.DrawItem, priority:7);
 item.color = RightClick.defaultColor;
 item.onClick = ()=>
 {
  GeneratorMenuAttribute att = GeneratorDraw.GetMenuAttribute(gen.GetType());
  if (att.helpLink != null)
   Application.OpenURL(att.helpLink);
 };
 item.closeOnClick = true;
 genItems.subItems.Add(item);
}

if (gen!=null)
{ //code
 Item item = new Item($"GoTo Code", onDraw:RightClick.DrawItem, priority:8);
 item.color = RightClick.defaultColor;
 item.onClick = ()=>
 {
  (string path, int line) = gen.GetCodeFileLine();
```

```
  bool found = false;
  if (path != null  &&  !path.Contains("Generator.cs"))
  {
   string file = System.IO.Path.GetFileNameWithoutExtension(path);
   string[] assets = AssetDatabase.FindAssets(file);
   if (assets.Length != 0)
   {
    AssetDatabase.OpenAsset(AssetDatabase.LoadAssetAtPath(AssetDatabase.GUIDToAssetPath(asse
    found = true;
   }
  }

  if (!found)
   EditorUtility.DisplayDialog("Error", "Could not find generator file: " + path, "OK");
 };
 item.closeOnClick = true;
 genItems.subItems.Add(item);
}

if (gen!=null)
{ //id
 Item item = new Item($"Id: {gen.id}", onDraw:RightClick.DrawItem, priority:3);
 item.color = RightClick.defaultColor;
 item.onClick = ()=> EditorGUIUtility.systemCopyBuffer = gen.id.ToString();
 item.closeOnClick = true;
```

```
    genItems.subItems.Add(item);
   }

   #if MM_DEBUG
   if (gen!=null)
   { //position
    Item item = new Item($"Pos: {gen.guiPosition}", onDraw:RightClick.DrawItem, priority:2);
    item.color = RightClick.defaultColor;
    item.onClick = ()=> EditorGUIUtility.systemCopyBuffer = gen.guiPosition.ToString();
    item.closeOnClick = true;
    genItems.subItems.Add(item);
   }

   #endif

   return genItems;
  }
 }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;
using UnityEditor;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.GUI.Popup;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Nodes.GUI;

namespace MapMagic.Nodes.GUI
{
 public static class GraphPopup
 {
   public static Item GraphItems (Graph graph, int priority=1)
   {
    Item graphItems = new Item("Graph");
    graphItems.onDraw = RightClick.DrawItem;
    graphItems.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Graph");
    graphItems.color = RightClick.defaultColor;
    graphItems.subItems = new List<Item>();
```

```
graphItems.priority = priority;



Item importItem = new Item("Import", onDraw:RightClick.DrawItem, priority:9) { icon = RightClick.texture
importItem.onClick = ()=>
{
 Graph imported = ScriptableAssetExtensions.LoadAsset<Graph>("Load Graph", filters: new string[]{"As
 if (imported != null)
  graph.Import(imported);
};
graphItems.subItems.Add(importItem);



Item exportItem = new Item("Export", onDraw:RightClick.DrawItem, priority:9) { icon = RightClick.texture
exportItem.disabled = GraphWindow.current.selected==null || GraphWindow.current.selected.Count==0
exportItem.onClick = ()=>
{
 Graph exported = graph.Export(GraphWindow.current.selected);
 ScriptableAssetExtensions.SaveAsset(exported, caption:"Save Graph", type:"asset");
};
graphItems.subItems.Add(exportItem);



graphItems.subItems.Add( new Item("Update All", onDraw:RightClick.DrawItem, priority:1) { icon = Right

//graphItems.subItems.Add( new Item("Background", onClick:()=>GraphWindow.current.noBackground=
```

```
    //graphItems.subItems.Add( new Item("Debug", onClick:()=>GraphWindow.current.drawGenDebug=!Gra


    return graphItems;

   }

 }

}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;
using UnityEditor;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.GUI.Popup;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Nodes.GUI;

namespace MapMagic.Nodes.GUI
{
 public static class GraphTreePopup
 {
  public static void DrawGraphTree (Graph rootGraph)
  {
   List<Item> items = new List<Item>();
   FillSubGraphItems(rootGraph, rootGraph, "", items);

   PopupMenu menu = new PopupMenu() { items=items, sortItems=false };  //items=new List<Item>() {item}
   menu.Show(Event.current.mousePosition);
```

```
  }



  private static void FillSubGraphItems (Graph graph, Graph root, string prefix, List<Item> items)
  {
   Item item = new Item(prefix + graph.name);
   items.Add(item);
   item.onClick = () => GraphWindow.current.OpenBiome(graph, root);

   foreach (Graph subGraph in graph.SubGraphs())
    FillSubGraphItems(subGraph, root, $"{prefix}   ", items);
  }
 }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;
using UnityEditor;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.GUI.Popup;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Nodes.GUI;

namespace MapMagic.Nodes.GUI
{
 public static class GroupRightClick
 {
   private static GUIStyle itemTextStyle;

   public static Item GroupItems (Vector2 mousePos, Group grp, Graph graph, int priority=3)
   {
    Item genItems = new Item("Group");
    genItems.onDraw = RightClick.DrawItem;
    genItems.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Generator");
```

```
genItems.color = RightClick.defaultColor;
genItems.subItems = new List<Item>();
genItems.priority = priority;

//genItems.disabled = grp == null;

genItems.subItems.Add( new Item("Create", onDraw:RightClick.DrawItem, priority:12) {
 icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/GroupAdd"),
 color =  RightClick.defaultColor,
 onClick = ()=> CreateGroup(mousePos, graph)} );

genItems.subItems.Add( new Item("Group Selected", onDraw:RightClick.DrawItem, priority:12) {
 icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/GroupSelected"),
 color =  RightClick.defaultColor,
 disabled = GraphWindow.current.selected==null || GraphWindow.current.selected.Count==0,
 onClick = ()=> GroupSelected(mousePos, graph)} );

//genItems.subItems.Add( new Item("Export", onDraw:DrawItem, priority:10) { icon = texturesCache.GetT
//genItems.subItems.Add( new Item("Import", onDraw:DrawItem, priority:9) { icon = texturesCache.GetTe
//genItems.subItems.Add( new Item("Duplicate", onDraw:DrawItem, priority:8) { icon = texturesCache.Ge
//genItems.subItems.Add( new Item("Update", onDraw:DrawItem, priority:7) { icon = texturesCache.GetT

genItems.subItems.Add( new Item("Ungroup", onDraw:RightClick.DrawItem, priority:5) {
 icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Ungroup"),
 color =  RightClick.defaultColor,
 onClick = ()=> RemoveGroup(grp,graph,withContent:false) });
```

```
 genItems.subItems.Add( new Item("Remove", onDraw:RightClick.DrawItem, priority:4) {
  icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Remove"),
  color =  RightClick.defaultColor,
  onClick = ()=> RemoveGroup(grp,graph,withContent:true) });

 return genItems;
}



public static Group CreateGroup (Vector2 mousePos, Graph graph)
{
  GraphWindow.RecordCompleteUndo();

  Group grp = new Group();
  grp.guiPos = mousePos;
  graph.Add(grp);

  GraphWindow.current.Focus();
  GraphWindow.current.Repaint();
  //GraphWindow.RefreshMapMagic(); //not necessary

  return grp;
}
```

```
public static void GroupSelected (Vector2 mousePos, Graph graph)
{
 Group ngrp = CreateGroup(mousePos, graph);
 PullGroupToSelected(ngrp);
}


public static void PullGroupToSelected (Group grp)
{
 HashSet<Generator> selected = GraphWindow.current.selected;
 if (selected != null  &&  selected.Count != 0)
 {
  Rect selectedRect = new Rect(selected.Any().guiPosition, selected.Any().guiSize);
  foreach (Generator gen in selected)
   selectedRect = selectedRect.Encapsulate( new Rect(gen.guiPosition, gen.guiSize) );

  selectedRect = selectedRect.Extended(20,20,60,20);

  grp.guiPos = selectedRect.position;
  grp.guiSize = selectedRect.size;
 }
}


public static void RemoveGroup (Group grp, Graph graph, bool withContent=false)
{
```

```
 GraphWindow.RecordCompleteUndo();

 if (withContent)
  GroupDraw.RemoveGroupContents(grp, graph);

 graph.Remove(grp);

 GraphWindow.current.Focus();
 GraphWindow.current.Repaint();

 if (withContent)
  GraphWindow.current?.RefreshMapMagic();
}

public static void DrawGroupColorSelector (Group group)
{
 Item menuItem = new Item("Colors");
 menuItem.subItems = new List<Item>
 {
  GetGroupColorSelectorItem(group, "Neutral", new Color(0.625f, 0.625f, 0.625f, 1), 210),
  GetGroupColorSelectorItem(group, "Rose Quartz", new Color(246f/256f, 202f/256f, 201f/256f, 1), 200),
  GetGroupColorSelectorItem(group, "Dahlia", new Color(235f/256f, 149f/256f, 135f/256f, 1), 190),
  GetGroupColorSelectorItem(group, "Flame", new Color(243f/256f, 85f/256f, 76f/256f, 1), 180),
  GetGroupColorSelectorItem(group, "Marsala", new Color(150f/256f, 79f/256f, 77f/256f, 1), 170),
  GetGroupColorSelectorItem(group, "Hazelnut", new Color(225f/256f, 174f/256f, 155f/256f, 1), 160),
```

```
  GetGroupColorSelectorItem(group, "Butterum", new Color(196f/256f, 142f/256f, 104f/256f, 1), 150),
  GetGroupColorSelectorItem(group, "Primrose", new Color(255f/256f, 204f/256f, 115f/256f, 1), 140),
  GetGroupColorSelectorItem(group, "Amber", new Color(255f/256f, 182f/256f, 72f/256f, 1), 130),
  GetGroupColorSelectorItem(group, "Cream Gold", new Color(221f/256f, 191f/256f, 94f/256f, 1), 120),
  GetGroupColorSelectorItem(group, "Gold Lime", new Color(155f/256f, 151f/256f, 64f/256f, 1), 110),
  GetGroupColorSelectorItem(group, "Lint", new Color(182f/256f, 186f/256f, 153f/256f, 1), 100),
  GetGroupColorSelectorItem(group, "Greenery", new Color(118f/256f, 177f/256f, 97f/256f, 1), 90),
  GetGroupColorSelectorItem(group, "Green", new Color(84f/256f, 194f/256f, 71f/256f, 1), 80),
  GetGroupColorSelectorItem(group, "Kale", new Color(89f/256f, 118f/256f, 87f/256f, 1), 70),
  GetGroupColorSelectorItem(group, "Beryl", new Color(96f/256f, 144f/256f, 135f/256f, 1), 60),
  GetGroupColorSelectorItem(group, "Arctic", new Color(100f/256f, 133f/256f, 137f/256f, 1), 50),
  GetGroupColorSelectorItem(group, "Niagara", new Color(51f/256f, 142f/256f, 163f/256f, 1), 40),
  GetGroupColorSelectorItem(group, "Island", new Color(118f/256f, 206f/256f, 216f/256f, 1), 30),
  GetGroupColorSelectorItem(group, "Carolina", new Color(139f/256f, 184f/256f, 232f/256f, 1), 20),
  GetGroupColorSelectorItem(group, "Navy", new Color(64f/256f, 63f/256f, 111f/256f, 1), 10)
 };
 PopupMenu menu = new PopupMenu() {items=menuItem.subItems, minWidth=150};
 menu.Show(Event.current.mousePosition);
}

private static void DrawGroupColorSelectorItem (Item item, Rect rect)
{
 Rect iconRect = new Rect(rect.x, rect.y, 18,18);
 Rect labelRect = new Rect(rect.x+iconRect.width+3, rect.y, rect.width-iconRect.width-3, rect.height);

 if (itemTextStyle == null)
```

```
 {
  itemTextStyle = new GUIStyle(UnityEditor.EditorStyles.label);
  itemTextStyle.normal.textColor = itemTextStyle.focused.textColor = itemTextStyle.active.textColor = Co
 }

 EditorGUI.DrawRect(iconRect.Extended(-2), new Color(item.color.r*0.7f, item.color.g*0.7f, item.color.b*0
 EditorGUI.DrawRect(iconRect.Extended(-3), item.color);

 UnityEditor.EditorGUI.LabelField(labelRect, item.name, itemTextStyle);
}

private static Item GetGroupColorSelectorItem (Group group, string name, Color color, int priority)
{
 TexturesCache texturesCache = UI.current.textures;
 return new Item()
 {
  onDraw = DrawGroupColorSelectorItem,
  color = color,
  name = name,
  onClick = () => group.color = color,
  priority = priority
 };
}

private static void FocusRepaintRefreshWindow ()
```

```
  {
   if (GraphWindow.current==null)
    return;

   GraphWindow.current.Focus();
   GraphWindow.current.Repaint();

   GraphWindow.current.RefreshMapMagic();
  }

 }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;
using UnityEditor;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.GUI.Popup;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Nodes.GUI;

namespace MapMagic.Nodes.GUI
{
 public static class PortalSelectorPopup
 {
  private static GUIStyle itemTextStyle;

  public static void DrawPortalSelector (Graph graph, IPortalExit<object> portalExit)
  {
   //if (MapMagic.instance.guiGens == null) MapMagic.instance.guiGens = MapMagic.instance.gens;
   //GeneratorsAsset gens = MapMagic.instance.guiGens;
   //if (MapMagic.instance.guiGens != null) gens = MapMagic.instance.guiGens;
```

```
Type exitType = portalExit.GetType().BaseType.GetGenericArguments()[0];


if (itemTextStyle == null)
{
 itemTextStyle = new GUIStyle(UnityEditor.EditorStyles.label);
 itemTextStyle.normal.textColor = itemTextStyle.focused.textColor = itemTextStyle.active.textColor = Col
}


List<Item> enterPortalsItems = new List<Item>();


for (int g=0; g<graph.generators.Length; g++)
{
 IPortalEnter<object> portalEnter = graph.generators[g] as IPortalEnter<object>;
 if (portalEnter == null) continue;
 if (portalEnter.GetType().BaseType.GetGenericArguments()[0] != exitType) continue;
 //TODO: generic portals


 Item item = new Item( portalEnter.Name,
  onDraw: (i, r) => EditorGUI.LabelField(r, i.name, itemTextStyle),
  onClick: ()=>
  {
   if (graph.AreDependent((Generator)portalExit, (Generator)portalEnter))
    { EditorUtility.DisplayDialog("MapMagic", "Linking portals this way will create a dependency loop.", "Ca


   portalExit.AssignEnter(portalEnter, graph);
```

```
        GraphWindow.current?.RefreshMapMagic();
      } );


     enterPortalsItems.Add(item);
    }


    PopupMenu menu = new PopupMenu() { items=enterPortalsItems };
    menu.Show(Event.current.mousePosition);


   }


  }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;
using UnityEditor;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.GUI.Popup;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Nodes.GUI;

namespace MapMagic.Nodes.GUI
{
 public static class RightClick
 {
  public static readonly Color defaultColor = new Color(0.5f, 0.5f, 0.5f, 0);

  private static GUIStyle itemTextStyle;
  public static TexturesCache texturesCache = new TexturesCache(); //to store icons


  public static void DrawRightClickItems (UI ui, Vector2 mousePos, Graph graph)
```

```
{
 Item item = RightClickItems(ui, mousePos, graph);

 //#if MM_EXP || UNITY_2020_1_OR_NEWER || UNITY_EDITOR_LINUX
 SingleWindow menu = new SingleWindow(item);
 //#else
 //PopupMenu menu = new PopupMenu() {items=item.subItems, minWidth=150};
 //#endif

 menu.Show(Event.current.mousePosition);
}


public static Item RightClickItems (UI ui, Vector2 mousePos, Graph graph)
{
 ClickedNear (ui, mousePos,
  out Group clickedGroup,
  out Generator clickedGen,
  out IInlet<object> clickedLink,
  out IInlet<object> clickedInlet,
  out IOutlet<object> clickedOutlet,
  out RightClickExpose clickedExpose);

 Item menu = new Item("Menu");
 menu.subItems = new List<Item>();
```

```
 if (clickedOutlet != null)
  menu.subItems.Add( CreateRightClick.AppendItems(mousePos, graph, clickedOutlet, priority:5) );

 else if (clickedLink != null)
  menu.subItems.Add( CreateRightClick.InsertItems(mousePos, graph, clickedLink, priority:5) );

 else
  menu.subItems.Add( CreateRightClick.CreateItems(mousePos, graph, priority:5) );

 menu.subItems.Add( GeneratorRightClick.GeneratorItems(mousePos, clickedGen, graph, priority:4) );
 menu.subItems.Add( GroupRightClick.GroupItems(mousePos, clickedGroup, graph, priority:3) );
 menu.subItems.Add( ValueRightClick.ValueItems(clickedExpose, clickedGen, graph, priority:2) );
 menu.subItems.Add( GraphPopup.GraphItems(graph, priority:1) );

 return menu;
}


public static bool ClickedNear (UI ui, Vector2 mousePos,
 out Group clickedGroup,
 out Generator clickedGen,
 out IInlet<object> clickedLink,
 out IInlet<object> clickedInlet,
 out IOutlet<object> clickedOutlet,
 out RightClickExpose clickedExpose)
/// Returns the top clicked object (or null) in clickedGroup-to-clickedField priority
```

```
{
 clickedGroup = null;
 clickedGen = null;
 clickedLink = null;
 clickedInlet = null;
 clickedOutlet = null;
 clickedExpose = null;

 List<Cell> cellsUnderCursor = new List<Cell>();
 ui.rootCell.FillCellsUnderCursor(cellsUnderCursor, mousePos);

 //checking cells
 for (int i=0; i<cellsUnderCursor.Count; i++)
 {
  Cell cell = cellsUnderCursor[i];

  //GeneratorDraw.genCellLut.TryGetValue(cell, out clickedGen);  //TryGet will overwrite to null if not found

  if (ui.cellObjs.TryGetObject(cell, "Generator", out Generator gen)) clickedGen = gen;
  if (ui.cellObjs.TryGetObject(cell, "Group", out Group group)) clickedGroup = group;
  if (ui.cellObjs.TryGetObject(cell, "Inlet", out IInlet<object> inlet)) clickedInlet = inlet;
  if (ui.cellObjs.TryGetObject(cell, "Outlet", out IOutlet<object> outlet)) clickedOutlet = outlet;
  if (ui.cellObjs.TryGetObject(cell, "Expose", out RightClickExpose field)) clickedExpose = field;

  if (clickedGen != null  &&  clickedOutlet == null  &&  clickedGen is IOutlet<object> o)
   clickedOutlet = o;
```

```
   //assigning outlet if clicked on single-outlet gen
  }

  //checking links
  float minDist = 10; //10 pixels is max dist to link
  if (UI.current.scrollZoom != null) minDist /= UI.current.scrollZoom.zoom;
  foreach (var kvp in GraphWindow.current.graph.links)
  {
   float dist = GeneratorDraw.DistToLink(mousePos, kvp.Value, kvp.Key);
   if (dist < minDist)
    {minDist = dist; clickedLink=kvp.Key;}
  }

  return clickedGroup != null ||  clickedGen != null || clickedLink != null || clickedInlet != null || clickedOutlet !
 }



 public static object ClickedOn (UI ui, Vector2 mousePos)
 /// Returns the top clicked object (or null) in clickedGroup-to-clickedField priority
 {
  ClickedNear (ui, mousePos,
   out Group clickedGroup, out Generator clickedGen, out IInlet<object> clickedLink, out IInlet<object> clic

  if (clickedExpose != null) return clickedExpose;
  if (clickedOutlet != null) return clickedOutlet;
  if (clickedInlet != null) return clickedInlet;
```

```
 if (clickedLink != null) return clickedLink;
 if (clickedGen != null) return clickedGen;
 if (clickedGroup != null) return clickedGroup;

 return null;
}


public static void DrawItem (Item item, Rect rect)
{
 Rect leftRect = new Rect(rect.x, rect.y, 28, rect.height);
 leftRect.x -= 1; leftRect.height += 2;
 item.color.a = 0.25f;
 EditorGUI.DrawRect(leftRect, item.color);

 Rect labelRect = new Rect(rect.x+leftRect.width+3, rect.y, rect.width-leftRect.width-3, rect.height);

 if (itemTextStyle == null)
 {
  itemTextStyle = new GUIStyle(UnityEditor.EditorStyles.label);
  itemTextStyle.normal.textColor = itemTextStyle.focused.textColor = itemTextStyle.active.textColor = Col
 }

 EditorGUI.LabelField(labelRect, item.name, itemTextStyle);

 if (item.icon!=null)
```

```
  {
   Rect iconRect = new Rect(leftRect.center.x-6, leftRect.center.y-6, 12,12);
   iconRect.y -= 2;
   UnityEngine.GUI.DrawTexture(iconRect, item.icon);
  }
 }



 public static void DrawSeparator (Item item, Rect rect)
 {
  Rect leftRect = new Rect(rect.x, rect.y, 28, rect.height);
  leftRect.x -= 1; leftRect.height += 2;
  item.color.a = 0.125f;
  EditorGUI.DrawRect(leftRect, item.color);

  Rect labelRect = new Rect(rect.x+leftRect.width+3, rect.y, rect.width-leftRect.width-3, rect.height);
  Rect separatorRect = new Rect(labelRect.x, labelRect.y+2, labelRect.width-6, 1);
  EditorGUI.DrawRect(separatorRect, new Color(0.3f, 0.3f, 0.3f, 1));
 }
}
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;
using UnityEditor;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.GUI.Popup;
using MapMagic.Core;
using MapMagic.Nodes;
using MapMagic.Nodes.GUI;
using MapMagic.Expose.GUI;

namespace MapMagic.Nodes.GUI
{
 public class RightClickExpose
 {
  //public FieldInfo field;
  public string fieldName;
  public Type fieldType;
  public ulong id; //generator or layer id
  public int channel; //for Vector fields
  public int arrIndex;
```

```
 public RightClickExpose (ulong id, string fieldName, Type fieldType, int channel, int arrIndex)
  { this.fieldName=fieldName; this.fieldType=fieldType; this.id=id; this.channel=channel; this.arrIndex=arrIn
}

public static class ValueRightClick
{
 public static Item ValueItems (RightClickExpose expose, Generator gen, Graph graph, int priority=2)
 /// chNum is a channel for vector fields
 {
  Item valItems = new Item("Value");
  valItems.onDraw = RightClick.DrawItem;
  valItems.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Value");
  valItems.color =  RightClick.defaultColor;
  valItems.subItems = new List<Item>();
  valItems.priority = priority;

  valItems.disabled = expose==null || gen==null;

  Item exposeItem = new Item("Expose", onDraw:RightClick.DrawItem, priority:6);
  exposeItem.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/Expose");
  if (expose != null) exposeItem.onClick =
   () => { ExposeWindow.ShowWindow(graph, expose.id, expose.fieldName, expose.fieldType, expose.cha
  valItems.subItems.Add(exposeItem);

  Item unExposeItem = new Item("UnExpose", onDraw:RightClick.DrawItem, priority:6);
```

```
    unExposeItem.icon = RightClick.texturesCache.GetTexture("MapMagic/Popup/UnExpose");
    if (expose != null) unExposeItem.onClick = //() => { if (gen.exposed==null) gen.exposed=new Exposed();
     () =>
     {
      //graph.exposed.Unexpose(gen, valField);
      graph.exposed.Remove(gen.id, expose.fieldName, expose.channel);
      GraphWindow.current.Focus();
      GraphWindow.current.Repaint();
     };
    valItems.subItems.Add(unExposeItem);


    return valItems;
   }
  }
 }
```

```
ï»¿
using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using System.Reflection;
using UnityEngine;
using UnityEditor;

using Den.Tools;
using Den.Tools.Tasks;
using Den.Tools.GUI;

using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes;
using MapMagic.Terrains;

using MapMagic.Nodes.GUI;

namespace MapMagic.Previews
{
 public static class PreviewDraw
 {
  public static Color BackgroundColor =>  StylesCache.isPro ?
   new Color(0.3f, 0.3f, 0.3f, 1) :
```

```
 new Color(0.4f, 0.4f, 0.4f, 1);
private static Material textureRawMat;



public static void DrawPreview (IOutlet<object> outlet)
{
 IPreview preview = PreviewManager.GetPreview(outlet);
 if (preview == null)
 {
  preview = PreviewManager.CreatePreview(outlet);
  if (preview == null) return; //for unknown types
  if (GraphWindow.current.mapMagic is MapMagicObject mapMagicObject)
   preview.SetObject(outlet, mapMagicObject.PreviewData);
 }

 //preview itself
 using (Cell.Full)
 {
  Color backColor = StylesCache.isPro ?
    new Color(0.33f, 0.33f, 0.33f, 1) :
    new Color(0.4f, 0.4f, 0.4f, 1);
  Draw.Rect(BackgroundColor);  //background in case no preview, or preview object was not assigned

  if (preview != null) preview.DrawInGraph();
 }
```

```
 //clock/na
 if (GraphWindow.current.mapMagic!=null)
 {
  if (preview.Stage==PreviewStage.Generating  ||  (preview.Stage==PreviewStage.Blank && GraphWindo
   using (Cell.Full) Draw.Icon(UI.current.textures.GetTexture("MapMagic/PreviewSandClock"));
  else if (preview.Stage==PreviewStage.Blank)
   using (Cell.Full) Draw.Icon(UI.current.textures.GetTexture("MapMagic/PreviewNA"));
 }

 //terrain buttons
 if (preview != null)
  using (Cell.Full)
   TerrainWindowButtons(preview);
}



private static void TerrainWindowButtons (IPreview preview)
{
 using (Cell.LineStd)
 {
  using (Cell.RowPx(20))
  {
   if (preview.Terrain != null)
   {
    if (Draw.Button(UI.current.textures.GetTexture("MapMagic/Icons/PreviewToTerrainActive"), visible:false
     PreviewManager.RemoveAllFromTerrain();
```

```
    }
    else
    {
     if (Draw.Button(UI.current.textures.GetTexture("MapMagic/Icons/PreviewToTerrain"), visible:false))
     {
      PreviewManager.RemoveAllFromTerrain();
      if (GraphWindow.current.mapMagic is MapMagicObject mapMagicObject)
      {
       TerrainTile previewTile = mapMagicObject.PreviewTile;
       preview.ToTerrain(previewTile?.main?.terrain, previewTile?.draft?.terrain);
      }
     }
    }
   }

    using (Cell.RowPx(20))
     if (Draw.Button(UI.current.textures.GetTexture("MapMagic/Icons/PreviewToWindow"), visible:false))
      PreviewManager.GetCreateWindow(preview);

    //Cell.EmptyRow();
  }
 }


 public static void DrawGenerateMarkInWindow (PreviewStage stage, Vector2 center)
 /// Draws non-scaled N/A or clock icon in the center of the preview window if necessary
```

```
  {
   Texture2D tex;
   switch (stage)
   {
    case PreviewStage.Blank: tex = UI.current.textures.GetTexture("MapMagic/PreviewNA"); break;
    case PreviewStage.Generating: tex = UI.current.textures.GetTexture("MapMagic/PreviewSandClock"); b
    default: tex = null; break;
   }

   if (tex != null)
    using (Cell.Full)
     using (Cell.Custom(center, Vector2.one))
   {
    Cell.current.pixelSize = new Vector2( tex.width/UI.current.scrollZoom.zoom, tex.height/UI.current.scrollZ
    Cell.current.pixelOffset -= Cell.current.pixelSize / 2;

    Draw.Texture(tex);
   }
  }
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using System.Reflection;
using UnityEngine;
using UnityEditor;

using Den.Tools;
using Den.Tools.Matrices;
//using Den.Tools.Segs;
using Den.Tools.Tasks;
using Den.Tools.GUI;
using Den.Tools.Matrices.Window;

using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes;
using MapMagic.Terrains;

using MapMagic.Nodes.GUI;

namespace MapMagic.Previews
{
 public class MatrixPreview : IPreview
 {
```

```
public bool colorized = true;

public bool relief = true;


public MatrixWorld matrix = null;  //to pass to thread

public CoordRect activeRect;

private int margins;

private byte[] bytes = null;  //to pass from thread to apply

public Texture2D tex = null;  //used by Preview.height


public PreviewStage Stage { get; set; } = PreviewStage.Blank;


private static Material guiMat;

private static Material terrainMat;


public BaseMatrixWindow Window { get; set; }


private Terrain terrain;

private Terrain draftTerrain;

public Terrain Terrain => terrain;



#region Generate


 public void Clear ()

 {

  Stage = PreviewStage.Blank;
```

```
 Window?.Repaint();
}

public void SetObject (IOutlet<object> outlet, TileData data)
{
 if (data != null)
  SetObject((MatrixWorld)data.ReadOutletProduct(outlet), data.area);
}



public void SetObject (MatrixWorld matrix, Area area)
{
 this.matrix = matrix;
 this.activeRect = area.active.rect;
 this.margins = area.Margins;

 Stage = PreviewStage.Generating;

 ThreadManager.Enqueue(ExecuteInThread, priority:-1000);
}



public void ExecuteInThread ()
{
 if (matrix == null) { Stage=PreviewStage.Blank; return; }
```

```
 bytes = new byte[matrix.rect.Count*2];
 matrix.ExportRaw16(bytes, matrix.rect.offset, matrix.rect.size);

 CoroutineManager.Enqueue(ApplyInMain, priority:-1000);
}

public void ApplyInMain ()
{
 if (bytes==null) return;

 int textureSize = (int)Mathf.Sqrt(bytes.Length / 2);

 if (tex==null || tex.width != textureSize || tex.height != textureSize)
  tex = new Texture2D(textureSize, textureSize, TextureFormat.R16, false, true);

 tex.LoadRawTextureData(bytes);
 tex.Apply();

 Stage = PreviewStage.Ready;
 Window?.Repaint();
}

#endregion
```

```
#region Terrain


 public void ToTerrain (Terrain terrain, Terrain draftTerrain)
 {
  this.terrain = terrain;
  this.draftTerrain = draftTerrain;


  terrain.drawHeightmap = false;
  if (draftTerrain != null) draftTerrain.drawHeightmap = false;


  Terrain substituteTerrain = GetSubstituteTerrain(terrain);
  if (substituteTerrain == null) substituteTerrain = CreateSubstituteTerrain(terrain);


  substituteTerrain.terrainData = terrain.terrainData; //in case was previewing other terrain


  if (terrainMat == null)
  {
   Shader shader;
   #if UNITY_2019_2_OR_NEWER
   Material defaultTerrainMat = UnityEngine.Rendering.GraphicsSettings.defaultRenderPipeline?.defaultT
   string pipelineName = UnityEngine.Rendering.GraphicsSettings.defaultRenderPipeline?.name;


   if ( (defaultTerrainMat != null  &&  defaultTerrainMat.shader.name.Contains("Universal Render Pipeline"
     (pipelineName != null && pipelineName.Contains("URP")) ) //"URP-HighFidelity"
     shader = Shader.Find("MapMagic/TerrainPreviewURP");
```

```
  else if ( (defaultTerrainMat != null  &&  defaultTerrainMat.shader.name.Contains("HDRP")) ||  //"HDRP/T
    (pipelineName != null && pipelineName.Contains("HDR")) )  //"HDRenderPipelineAsset"
   shader = Shader.Find("MapMagic/TerrainPreviewHDRP");

  else
  #endif
   shader = Shader.Find("MapMagic/TerrainPreview");

  terrainMat = new Material(shader);
 }
 #if !UNITY_2019_2_OR_NEWER
 substituteTerrain.materialType = Terrain.MaterialType.Custom;
 #endif

 substituteTerrain.materialTemplate = terrainMat;

 substituteTerrain.basemapDistance = 100000; //disabling base map since SRP preview shader does not

 terrainMat.SetTexture("_Preview", tex);
 terrainMat.SetInt("_Margins", margins);
}


public void ClearTerrain ()
{
 if (terrain == null) return;
```

```
 terrain.drawHeightmap = true;
 if (draftTerrain != null) draftTerrain.drawHeightmap = true;

 Terrain substituteTerrain = GetSubstituteTerrain(terrain);

 if (substituteTerrain != null)
 {
  Material previewMaterial = substituteTerrain.materialTemplate;
  if (previewMaterial != null) GameObject.DestroyImmediate(previewMaterial);

  GameObject.DestroyImmediate(substituteTerrain.gameObject);
 }

 terrain = null;
}


private static Terrain GetSubstituteTerrain (Terrain terrain)
{
 Transform tileTfm = terrain.transform.parent;
 Transform previewTfm = tileTfm.Find("Preview Terrain");
 if (previewTfm == null) return null;
 return previewTfm.GetComponent<Terrain>();
}
```

```
private static Terrain CreateSubstituteTerrain (Terrain terrain)
{
 GameObject go = new GameObject();
 go.name = "Preview Terrain";
 go.hideFlags = HideFlags.DontSaveInBuild | HideFlags.DontSaveInEditor;
 go.tag = "EditorOnly";
 go.transform.parent = terrain.transform.parent;
 go.transform.localPosition = Vector3.zero;
 Terrain previewTerrain = go.AddComponent<Terrain>();
 previewTerrain.terrainData = terrain.terrainData; //it will be assigned once more after in case using of ex

// StopPreviewOnLoad stopScript = go.AddComponent<StopPreviewOnLoad>();
// stopScript.active = true;

 previewTerrain.drawInstanced = terrain.drawInstanced; //set only if terrain data assigned
 previewTerrain.heightmapPixelError = terrain.heightmapPixelError;
 previewTerrain.drawTreesAndFoliage = false;
 #if UNITY_2019_1_OR_NEWER
 previewTerrain.shadowCastingMode = UnityEngine.Rendering.ShadowCastingMode.Off;
 #else
 previewTerrain.castShadows = false;
 #endif

 return previewTerrain;
}
```

```
[RuntimeInitializeOnLoadMethod, UnityEditor.InitializeOnLoadMethod]
static void ClearTerrainsOnLoad ()
{
 Terrain[] terrains = Resources.FindObjectsOfTypeAll<Terrain>(); //will return HideFlags.DontSave object
 for (int i=terrains.Length-1; i>=0; i--)
 {
  Terrain terrain = terrains[i];
  if (AssetDatabase.Contains(terrain) || AssetDatabase.Contains(terrain.gameObject)) continue;
   //FindObjectsOfTypeAll will return assets too

  //removing preview terrain (if left after script re-compile)
  if (terrain.gameObject.name == "Preview Terrain"  &&
   terrain.transform.parent != null  &&
   terrain.transform.parent.GetComponent<TerrainTile>() != null)
    { GameObject.DestroyImmediate(terrain.gameObject); continue; }

  //enabling drawHeightmap to all MM terrains
  if (!terrain.drawHeightmap  &&
   terrain.transform.parent != null  &&
   terrain.transform.parent.GetComponent<TerrainTile>() != null)
    terrain.drawHeightmap = true;
 }
}
```

```
#endregion

#region Window

 public BaseMatrixWindow CreateWindow ()
 {
  PreviewWindow window = ScriptableObject.CreateInstance<PreviewWindow>();

  window.plugins = new IPlugin[] {
   new StatsPlugin(),
   new ViewPlugin(),
   new PixelPlugin(),
   new SlicePlugin(),
   new ExportPlugin() {margins=margins} };
  window.preview = this;
  window.colorize = true;
  window.relief = true;
  window.name = "Map Preview";

  return window;
 }

 [System.Serializable]
 public class PreviewWindow : BaseMatrixWindow, IPreviewWindow
 {
  public IPreview Preview { get{return preview;} set{preview = value as MatrixPreview;} }
```

```
  public MatrixPreview preview;

  public override Matrix Matrix => preview?.matrix;
  public override Texture2D PreviewTexture => preview.tex;

  public ulong SerializedGenId
  {
   get{ return serializedGenId; }
   set{ serializedGenId = value; }
  }
  public ulong serializedGenId;


  protected override void DrawPreview ()
  {
   base.DrawPreview();

   PreviewDraw.DrawGenerateMarkInWindow(preview.Stage, MatrixRect.center);
  }
 }

#endregion

#region GUI

 public void DrawInGraph ()
```

```
  {
   if (tex == null) Draw.Rect(PreviewDraw.BackgroundColor);
   else
   {
    Draw.MatrixPreviewTexture(tex, colorized, relief, margins:margins);
    Draw.MatrixPreviewReliefSwitch(ref colorized, ref relief);
   }
  }

 #endregion
}
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using System.Reflection;
using UnityEngine;
using UnityEditor;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.Matrices.Window;
using Den.Tools.Tasks;
using Den.Tools.GUI;

using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes;
using MapMagic.Terrains;

using MapMagic.Nodes.GUI;

namespace MapMagic.Previews
{
 public class ObjectsPreview : IPreview
 {
  //temporary objects to pass to thread
```

```
public TransitionsList trns = null;

public MatrixWorld heights = null;

public Vector3 worldPos;

public Vector3 worldSize;

public int count; //number of obs within worldPos/worldSize rect


//mesh data passing from thread to main

private Vector3[] vertices; //relative coords from 0 to 1

private Vector3[] directions; //Y-types directions to create a triangle in shader

private int[] tris;


private Mesh mesh;


public bool relativeHeight = true;

private static Material heightTexMat;

private static Material meshMat;

private static Material terrainMat; //same shader as meshMat, but has rect and heightmap assigned (and

public PreviewStage Stage { get; set; } = PreviewStage.Blank;

public BaseMatrixWindow Window { get; set; }

public Terrain Terrain { get; set; }


public static readonly Color guiGizmoColor = new Color(0.3f,1,0,1);



#region Generate
```

```
public void Clear ()
{
 Stage = PreviewStage.Blank;
 Window?.Repaint();
}

public void SetObject (IOutlet<object> outlet, TileData data)
{
 if (data != null)
  SetObject((TransitionsList)data.ReadOutletProduct(outlet), data.heights, data.area);
}



public void SetObject (TransitionsList posTab, MatrixWorld heights, Area area)
{
 this.trns = posTab;
 this.heights = heights;
 this.worldPos = (Vector3)area.active.worldPos;
 this.worldSize = (Vector3)area.active.worldSize;

 Stage = PreviewStage.Generating;
 ThreadManager.Enqueue(ExecuteInThread, priority:-1000);
}
```

```
public void ExecuteInThread ()
{
 if (trns == null) { Stage = PreviewStage.Blank; return; }

 count = trns.CountInRect((Vector2D)worldPos, (Vector2D)worldSize);

 vertices = new Vector3[trns.count * 6];
 directions = new Vector3[trns.count * 6];
 tris = new int[trns.count * 6];

 int i = 0;
 for (int t=0; t<trns.count; t++)
 {
  Vector3 pos = trns.arr[t].pos;
  pos -= worldPos;
  pos = new Vector3 (pos.x/worldSize.x, pos.y, pos.z/worldSize.z);
 // if (heights != null) pos.y = pos.y/heights.worldSize.y;
  pos.y = 0;

  vertices[i] = vertices[i+1] = vertices[i+2] = vertices[i+3] = vertices[i+4] = vertices[i+5] = pos;

  directions[i]   = new Vector3(0, 2, 1); //background has Z of 1
  directions[i+1] = new Vector3(-4.5f, -7, 1);
  directions[i+2] = new Vector3(4.5f, -7, 1);

  directions[i+3] = new Vector3(0, 0, 0);
```

```
  directions[i+4] = new Vector3(-3f, -6, 0);
  directions[i+5] = new Vector3(3f, -6, 0);

  tris[i] = i;
  tris[i+1] = i+1;
  tris[i+2] = i+2;
  tris[i+3] = i+3;
  tris[i+4] = i+4;
  tris[i+5] = i+5;

  i+=6;
 }

 CoroutineManager.Enqueue(ApplyInMain, priority:-1000);
}


public void ApplyInMain ()
{
 if (vertices==null || tris==null) return;

 if (mesh == null) { mesh = new Mesh(); mesh.MarkDynamic(); }
 if (vertices.Length < mesh.vertices.Length) mesh.triangles = new int[0]; //otherwise "The supplied vertex
 mesh.vertices = vertices;
 mesh.normals = directions;
 mesh.triangles = tris;
```

```
  Stage = PreviewStage.Ready;
  Window?.Repaint();
 }

#endregion

#region Terrain

 public void ToTerrain (Terrain terrain, Terrain draftTerrain)
 {
  #if UNITY_2019_1_OR_NEWER
  SceneView.duringSceneGui -= DrawTerrainPreview;
  SceneView.duringSceneGui += DrawTerrainPreview;
  #else
  SceneView.onSceneGUIDelegate -= DrawTerrainPreview;
  SceneView.onSceneGUIDelegate += DrawTerrainPreview;
  #endif

  Terrain = terrain;
 }


 public void ClearTerrain ()
 {
  #if UNITY_2019_1_OR_NEWER
```

```
 SceneView.duringSceneGui -= DrawTerrainPreview;
 #else
 SceneView.onSceneGUIDelegate -= DrawTerrainPreview;
 #endif

 Terrain = null;
}

public void DrawTerrainPreview (SceneView sceneView) => DrawTerrainPreview();
public void DrawTerrainPreview ()
{
 if (mesh == null || Terrain == null) return;

 if (terrainMat == null)
 {
  terrainMat = new Material( Shader.Find("MapMagic/ObjectPreview") );
  terrainMat.SetColor("_Color", ObjectsPreview.guiGizmoColor);
  terrainMat.SetColor("_BackColor", new Color(0,0,0,1));
  terrainMat.SetFloat("_Size", 1.42f);
  terrainMat.SetFloat("_Flip", 0);
 }

 if (PreviewManager.heightOutputPreview?.tex != null)
  terrainMat.SetTexture("_Heightmap", PreviewManager.heightOutputPreview.tex);

 Vector3 position = Terrain.transform.position;
```

```
  Vector3 size = Terrain.terrainData.size;

  Matrix4x4 prs = Matrix4x4.TRS(
   position,
   Quaternion.identity,
   size );

  if (terrainMat.HasProperty("_Rect"))
   terrainMat.SetVector("_Rect", new Vector4(position.x, position.y, size.x, size.y));
   //not _ClipRect, since mesh is drawn in 0-1 range clipping it in shader

  terrainMat.SetPass(0);
  Graphics.DrawMeshNow(mesh, prs);
 }

#endregion

#region Window

 public BaseMatrixWindow CreateWindow ()
 {
  ObjectsPreviewWindow window = ScriptableObject.CreateInstance<ObjectsPreviewWindow>();

  window.plugins = new IPlugin[] {
   new ViewPlugin() };
  window.preview = this;
```

```
 window.colorize = false;
 window.relief = true;
 window.name = "Object Preview";

 return window;
}

public class ObjectsPreviewWindow : BaseMatrixWindow, IPreviewWindow
{
 public IPreview Preview { get{return preview;} set{preview = value as ObjectsPreview;} }
 public ObjectsPreview preview;

 public override Matrix Matrix => PreviewManager.heightOutputPreview?.matrix;
 public override Texture2D PreviewTexture => PreviewManager.heightOutputPreview?.tex;

 public ulong SerializedGenId
 {
  get{ return serializedGenId; }
  set{ serializedGenId = value; }
 }
 public ulong serializedGenId;


 protected override void DrawPreview ()
 {
  base.DrawPreview();
```

```
if (preview.mesh == null)
{
 PreviewDraw.DrawGenerateMarkInWindow(PreviewStage.Blank, Vector2.zero);
 return;
}

CoordRect matrixRect = PreviewManager.heightOutputPreview?.matrix!=null ?
 PreviewManager.heightOutputPreview.matrix.rect :
 new CoordRect(0,0,0,0);
CoordRect activeRect = PreviewManager.heightOutputPreview?.matrix!=null ?
 PreviewManager.heightOutputPreview.activeRect :
 new CoordRect(0,0,0,0);

//height background
using (Cell.Custom( ToMatrixRect(matrixRect) ))
{

 Texture2D heightTex = PreviewManager.heightOutputPreview!=null ? PreviewManager.heightOutputPr
 Draw.MatrixPreviewTexture(heightTex, colorize:false, relief:true);
}

//preview itself
using (Cell.Custom( ToMatrixRect(activeRect) ))
{
 if (meshMat == null)
```

```
    {
     meshMat = new Material( Shader.Find("MapMagic/ObjectPreview") );
     meshMat = UI.current.textures.GetMaterial("MapMagic/ObjectPreview");
     meshMat.SetColor("_Color", ObjectsPreview.guiGizmoColor);
     meshMat.SetColor("_BackColor", new Color(0,0,0,1));
     meshMat.SetFloat("_Size", 1.01f);
     meshMat.SetFloat("_Flip", 1);
    }

    Draw.Mesh(preview.mesh, meshMat, clip:false);
   }

   PreviewDraw.DrawGenerateMarkInWindow(preview.Stage, MatrixRect.center);
   if (PreviewManager.heightOutputPreview!=null)
    PreviewDraw.DrawGenerateMarkInWindow(PreviewManager.heightOutputPreview.Stage, MatrixRect.c
  }
 }

#endregion

#region GUI

 public void DrawInGraph ()
 {
  if (mesh == null) { Draw.Rect(PreviewDraw.BackgroundColor); return; }
```

```
//height background
Texture2D heightTex = PreviewManager.heightOutputPreview!=null ? PreviewManager.heightOutputPre
Draw.MatrixPreviewTexture(heightTex, colorize:false, relief:true);

//preview itself
if (meshMat == null)
{
 meshMat = new Material( Shader.Find("MapMagic/ObjectPreview") );
 meshMat = UI.current.textures.GetMaterial("MapMagic/ObjectPreview");
 meshMat.SetColor("_Color", ObjectsPreview.guiGizmoColor);
 meshMat.SetColor("_BackColor", new Color(0,0,0,1));
 meshMat.SetFloat("_Size", 1.01f);
 meshMat.SetFloat("_Flip", 1);
 meshMat.SetFloat("_Offset", 0);
}

Draw.Mesh(mesh, meshMat);

//objects count
using (Cell.Full)
{
 Cell.EmptyLine();
 using (Cell.LineStd) Draw.BackgroundRightLabel(count.ToString(), style:UI.current.styles.whiteLabel, ri
}

//use heightmap
```

```
    /*using (Cell.Full)
    {
     Cell.EmptyRow();
     using (Cell.RowPx(12))
     {
      Cell.EmptyLinePx(3);
      using (Cell.LinePx(12))
      {
       Texture2D icon;
       if (relativeHeight) icon = UI.current.textures.GetTexture("DPUI/TexCh/Heightmap");
       else icon = UI.current.textures.GetTexture("DPUI/TexCh/Flat");

       if (Draw.Button(icon, visible:false)) relativeHeight = !relativeHeight;
      }
      Cell.EmptyLine();
     }
     Cell.EmptyRowPx(3);
    }*/
   }

  #endregion
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using System.Reflection;
using UnityEngine;
using UnityEditor;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.Matrices.Window;
//using Den.Tools.Segs;
using Den.Tools.Splines;
using Den.Tools.Tasks;
using Den.Tools.GUI;

using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes;
using MapMagic.Terrains;

using MapMagic.Nodes.GUI;

namespace MapMagic.Previews
{
 public enum PreviewStage { Blank, Generating, Ready }
```

```
public interface IPreview
{
 void SetObject (IOutlet<object> outlet, TileData data);
 void Clear ();

 PreviewStage Stage { get; }

 void DrawInGraph ();

 BaseMatrixWindow Window { get; set; }
 BaseMatrixWindow CreateWindow ();

 Terrain Terrain { get; }
 void ToTerrain (Terrain terrain, Terrain draftTerrain);
 void ClearTerrain ();
}

public interface IPreviewType<T> where T: class { }


public static class PreviewManager
{
 private static Dictionary<IOutlet<object>,IPreview> all = new Dictionary<IOutlet<object>,IPreview>();
 //public static ConditionalWeakTable<IOutlet,Preview> all = new ConditionalWeakTable<IOutlet,Preview>
```

```
public  static MatrixPreview heightOutputPreview = new MatrixPreview();



public static IPreview GetPreview (IOutlet<object> outlet)
{
 if (all.TryGetValue(outlet, out IPreview preview)) return preview;
 else return null;
}



public static IPreview CreatePreview (IOutlet<object> outlet)
{
 IPreview preview = null;
 if (outlet is IOutlet<MatrixWorld>) preview = new MatrixPreview();
 else if (outlet is IOutlet<TransitionsList>) preview = new ObjectsPreview();
 else if (outlet is IOutlet<SplineSys>) preview = new SplinePreview();
 else return null;

 if (all.ContainsKey(outlet)) all[outlet] = preview;
 else all.Add(outlet, preview);

 return preview;
}



public static void SetObject (IOutlet<object> outlet, TileData data)
```

```
{
 if (all.TryGetValue(outlet, out IPreview preview))
  preview.SetObject(outlet, data);
}



public static void SetObjectsAll (TileData data)
{
 foreach (var kvp in all)
 {
  IOutlet<object> outlet = kvp.Key;
  IPreview preview = kvp.Value;

  preview.SetObject(outlet, data);
 }
}

public static void Clear (IOutlet<object> outlet)
{
 if (all.TryGetValue(outlet, out IPreview preview))
  preview.Clear();
}

public static void ClearAll ()
{
 foreach (IPreview preview in all.Values)
```

```
  preview.Clear();
}




public static void RemoveAllFromTerrain ()
{
 foreach (IPreview preview in all.Values)
  preview.ClearTerrain();
}

public static void ChangeTerrain (Terrain newTerrain, Terrain draftTerrain)
/// Removes preview from old terrain and assigns to new one
{
 foreach (IPreview preview in all.Values)
  if (preview.Terrain != null)
  {
   preview.ClearTerrain();
   preview.ToTerrain(newTerrain, draftTerrain);
  }
}



public static BaseMatrixWindow GetCreateWindow (IPreview preview)
/// Tries to find opened window by preview and updates it, and opens new window if non is found
{
```

```
 IOutlet<object> outlet = PreviewManager.GetOutlet(preview);

 ulong previewId = outlet.Gen.id;

 BaseMatrixWindow[] windows = Resources.FindObjectsOfTypeAll<BaseMatrixWindow>();

 BaseMatrixWindow window = null;
 for (int i=0; i<windows.Length; i++)
 {
  if (windows[i] is IPreviewWindow serGuid  &&  serGuid.SerializedGenId == previewId)
   { window = windows[i]; break; }
 }

 if (window == null)
 {
  window = preview.CreateWindow();
  if (window is IPreviewWindow serGuid) serGuid.SerializedGenId = previewId;
  window.ShowTab();
 }

 window.Show();

 return window;
}
```

```
public static IPreview Get (ulong genId)
{
 foreach (var kvp in all)
  if (kvp.Key.Gen.id == genId)
   return kvp.Value;
 return null;
}

public static IOutlet<object> GetOutlet (IPreview preview)
{
 foreach (var kvp in all)
  if (kvp.Value == preview)
   return kvp.Key;
 return null;
}


[RuntimeInitializeOnLoadMethod, UnityEditor.InitializeOnLoadMethod]
static void LoadPreviews ()
/// Loads all of the previews for current MapMagic object in scene
/// And finds proper windows to all of the previews
{
 MapMagicObject mapMagic = GameObject.FindObjectOfType<MapMagicObject>();
 if (mapMagic == null  ||  mapMagic.graph==null) return;
 foreach (Generator gen in mapMagic.graph.generators)
  if (gen.guiPreview  &&  gen is IOutlet<object> outlet)
```

```
   CreatePreview(outlet);


 BaseMatrixWindow[] windows = Resources.FindObjectsOfTypeAll<BaseMatrixWindow>();
 foreach (BaseMatrixWindow window in windows)
 {
  if (!(window is IPreviewWindow previewWindow)) continue;


  ulong id = previewWindow.SerializedGenId;


  foreach (var kvp in all)
   if (kvp.Key.Gen.id == id)
   {
    kvp.Value.Window = window;
    previewWindow.Preview = kvp.Value;
   }
 }
}



[RuntimeInitializeOnLoadMethod, UnityEditor.InitializeOnLoadMethod]
static void Subscribe ()
{
 /// Assigning height preview on height change
 Graph.OnOutputFinalized += (type, data, applyData, stop) =>
 {
  if (data.isPreview && typeof(Nodes.MatrixGenerators.HeightOutput200).IsAssignableFrom(type))
```

```
  heightOutputPreview.SetObject(data.heights, data.area);
};

/// Starting to generate a preview on node change
/// Note this is called in thread
Graph.OnAfterNodeGenerated += (gen, data) =>
{
 if (data.isPreview  &&  gen is IOutlet<object> outlet)
  SetObject(outlet, data);
};

/// When node cleared - disabling n/a mark
Graph.OnBeforeNodeCleared += (gen, data) =>
{
 if (!data.isPreview) return;

 if (gen is Nodes.MatrixGenerators.HeightOutput200)
  heightOutputPreview?.Clear();

 if (gen is IOutlet<object> outlet)
  Clear(outlet);
};

/// Forces all preview to re-generate on selecting new preview tile
TerrainTile.OnPreviewAssigned += (data) =>
{
```

```
    if (data != null && !data.isPreview)
    {
     heightOutputPreview.SetObject(data.heights, data.area);
     SetObjectsAll(data);
    }

    if (GraphWindow.current.mapMagic is MapMagicObject mapMagicObject)
     ChangeTerrain(mapMagicObject.PreviewTile.main?.terrain, mapMagicObject.PreviewTile.draft?.terrain)
   };
  }
 }

 public interface IPreviewWindow
 {
  ulong SerializedGenId { get; set; }  //to keep references of windows to preview on code re-compile
  IPreview Preview { get; set; }
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using System.Reflection;
using UnityEngine;
using UnityEditor;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.Matrices.Window;
//using Den.Tools.Segs;
using Den.Tools.Splines;
using Den.Tools.Tasks;
using Den.Tools.GUI;

using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes;
using MapMagic.Terrains;

using MapMagic.Nodes.GUI;

namespace MapMagic.Previews
{
 public class SplinePreview : IPreview
```

```
{
 [NonSerialized] public SplineSys splineSys = null;  //to pass to thread
 [NonSerialized] public PolyLine polyLine;  //in thread and in apply

 [NonSerialized] public Mesh nodesMesh;
 [NonSerialized] public Vector3[] nodesMeshVerts;
 [NonSerialized] public Vector2[] nodesMeshUvs;
 [NonSerialized] public int[] nodesMeshTris;

 public Vector2D worldPos;
 public Vector2D worldSize;
 public float worldHeight;

 public PreviewStage Stage { get; set; } = PreviewStage.Blank;
 public BaseMatrixWindow Window { get; set; }
 public Terrain Terrain { get; set; }

 private static Material heightTexMat;
 private static Material lineMat;
 private static Material terrainLineMat;
 private static Material terrainNodesMat;
 private static Texture2D lineTex;

 public static readonly Color guiGizmoColor = new Color(1f,0.5f,0,1);

 #region Generate
```

```
public void Clear ()
{
 Stage = PreviewStage.Blank;
 Window?.Repaint();
}

public void SetObject (IOutlet<object> outlet, TileData data)
{
 if (data == null) return;

 this.splineSys = (SplineSys)data.ReadOutletProduct(outlet);
 this.worldPos = data.area.active.worldPos;
 this.worldSize = data.area.active.worldSize;
 this.worldHeight = data.globals.height;

 Stage = PreviewStage.Generating;
 ThreadManager.Enqueue(ExecuteInThread, priority:-1000);
}


public void ExecuteInThread ()
{
 if (splineSys == null) { Stage = PreviewStage.Blank; return; }

 //auto-line
```

```
 /*Vector3[][] nodes = new Vector3[splineSys.splines.Length][];
 for (int s=0; s<splineSys.splines.Length; s++)
  nodes[s] = splineSys.splines[s].nodes;

 if (polyLine == null) polyLine = new PolyLine(0);
 polyLine.SetPointsThread(nodes);*/

 //line
 Vector3[][] points = splineSys.GetAllPoints(resPerUnit:0.2f, minRes:3, maxRes:20);

 for (int l=0; l<points.Length; l++)
  for (int p=0; p<points[l].Length; p++)
   points[l][p] = new Vector3(points[l][p].x/worldSize.x, points[l][p].y/worldHeight, points[l][p].z/worldSize.z)

 if (polyLine == null) polyLine = new PolyLine(0);
 polyLine.SetPointsThread(points);

 //nodes
 PrepareNodesMeshArrays();

 CoroutineManager.Enqueue(ApplyInMain, priority:-1000);
}

private void PrepareNodesMeshArrays ()
{
```

```
/*auto-line
int nodesNum = 0;
for (int s=0; s<splineSys.splines.Length; s++)
 nodesNum += splineSys.splines[s].nodes.Length;

//vertices
if (nodesMeshVerts == null  ||  nodesMeshVerts.Length != nodesNum*4)
 nodesMeshVerts = new Vector3[nodesNum*4];

int i = 0;
for (int s=0; s<splineSys.splines.Length; s++)
 for (int n=0; n<splineSys.splines[s].nodes.Length; n++)
 {
  Vector3 pos = splineSys.splines[s].nodes[n];
  pos = new Vector3(pos.x/worldSize.x, pos.y/worldHeight, pos.z/worldSize.z);

  for (int v=0; v<4; v++)
   nodesMeshVerts[i*4+v] = pos;
  i++;
 } */

int nodesNum = splineSys.NodesCount;

//vertices
if (nodesMeshVerts == null  ||  nodesMeshVerts.Length != nodesNum*4)
 nodesMeshVerts = new Vector3[nodesNum*4];
```

```
int i = 0;
for (int l=0; l<splineSys.lines.Length; l++)
 for (int n=0; n<splineSys.lines[l].NodesCount; n++)
 {
  Vector3 pos = splineSys.lines[l].GetNodePos(n);
  pos = new Vector3(pos.x/worldSize.x, pos.y/worldHeight, pos.z/worldSize.z);

  for (int v=0; v<4; v++)
   nodesMeshVerts[i*4+v] = pos;
  i++;
 }

//uvs
if (nodesMeshUvs == null  ||  nodesMeshUvs.Length != nodesNum*4)
{
 nodesMeshUvs = new Vector2[nodesNum*4];

 for (i=0; i<nodesNum; i++)
 {
  int v = i*4;
  nodesMeshUvs[v] = new Vector2(-1,-1);
  nodesMeshUvs[v+1] = new Vector2(-1,1);
  nodesMeshUvs[v+2] = new Vector2(1,1);
  nodesMeshUvs[v+3] = new Vector2(1,-1);
 }
```

```
 }

 //tris
 if (nodesMeshTris == null  ||  nodesMeshTris.Length != nodesNum*6)
 {
  nodesMeshTris = new int[nodesNum*6];

  for (i=0; i<nodesNum; i++)
  {
   int v = i*4;
   int t = i*6;

   nodesMeshTris[t] = v+0;
   nodesMeshTris[t+1] = v+2;
   nodesMeshTris[t+2] = v+3;

   nodesMeshTris[t+3] = v+0;
   nodesMeshTris[t+4] = v+1;
   nodesMeshTris[t+5] = v+2;
  }
 }

}

public void ApplyInMain ()
```

```
{
 if (polyLine == null) return;

 polyLine.SetPointsApply();

 if (nodesMesh == null) { nodesMesh = new Mesh(); nodesMesh.MarkDynamic(); }

 if (nodesMeshVerts.Length != nodesMesh.vertexCount)
 {
  if (nodesMeshVerts.Length < nodesMesh.vertexCount) //resetting uv and tris to avoid "Mesh.vertices is
  {
   nodesMesh.uv = new Vector2[0];
   nodesMesh.triangles = new int[0];
  }
  nodesMesh.vertices = nodesMeshVerts;
  nodesMesh.uv = nodesMeshUvs;
  nodesMesh.triangles = nodesMeshTris;
 }
 else
  nodesMesh.vertices = nodesMeshVerts; //just changing verts

 Stage = PreviewStage.Ready;
 Window?.Repaint();
}

#endregion
```

```
#region Terrain

 public void ToTerrain (Terrain terrain, Terrain draftTerrain)
 {
  #if UNITY_2019_1_OR_NEWER
  SceneView.duringSceneGui -= DrawTerrainPreview;
  SceneView.duringSceneGui += DrawTerrainPreview;
  #else
  SceneView.onSceneGUIDelegate -= DrawTerrainPreview;
  SceneView.onSceneGUIDelegate += DrawTerrainPreview;
  #endif

  Terrain = terrain;
 }



 public void ClearTerrain ()
 {
  #if UNITY_2019_1_OR_NEWER
  SceneView.duringSceneGui -= DrawTerrainPreview;
  #else
  SceneView.onSceneGUIDelegate -= DrawTerrainPreview;
  #endif

  Terrain = null;
```

```
}

public void DrawTerrainPreview (SceneView sceneView) => DrawTerrainPreview();
public void DrawTerrainPreview ()
{
 if (polyLine == null || Terrain == null) return;

 Vector3 position = Terrain.transform.position;
 Vector3 size = Terrain.terrainData.size;

 Matrix4x4 prs = Matrix4x4.TRS(
  position,
  Quaternion.identity,
  size );

 //line mesh
 if (lineTex == null) lineTex = Resources.Load("DPUI/PolyLineTex") as Texture2D;
 if (terrainLineMat == null)
 {
  terrainLineMat = new Material( Shader.Find("Hidden/DPLayout/PolyLine") );

  terrainLineMat.SetTexture("_MainTex", lineTex);
  terrainLineMat.SetColor("_Color", guiGizmoColor);
  terrainLineMat.SetFloat("_Width", 5);
  terrainLineMat.SetFloat("_Offset", 0.01f);
  //terrainLineMat.SetInt("_ZTest", 1);
```

```
 #if MM_DOC
 terrainLineMat.SetFloat("_Width", 10);
 #endif
}

terrainLineMat.SetPass(0);
Graphics.DrawMeshNow(polyLine.mesh, prs);

//nodes mesh
if (terrainNodesMat == null)
{
 terrainNodesMat = new Material( Shader.Find("Hidden/DPLayout/PolyLineBillboards") );
 terrainNodesMat.SetColor("_Color", guiGizmoColor);
 terrainNodesMat.SetFloat("_Size", 4);
 terrainNodesMat.SetFloat("_Offset", 0.02f);
 //terrainNodesMat.SetInt("_ZTest", 1);

 #if MM_DOC
 terrainNodesMat.SetFloat("_Size", 6); //4
 #endif
}

terrainNodesMat.SetPass(0);
Graphics.DrawMeshNow(nodesMesh, prs);
}
```

```
#endregion

#region Window

 public BaseMatrixWindow CreateWindow ()
 {
  SplinePreviewWindow window = ScriptableObject.CreateInstance<SplinePreviewWindow>();

  window.plugins = new IPlugin[] {
   new ViewPlugin() };
  window.preview = this;
  window.colorize = false;
  window.relief = true;
  window.name = "Spline Preview";

  return window;
 }

 public class SplinePreviewWindow : BaseMatrixWindow, IPreviewWindow
 {
  public IPreview Preview { get{return preview;} set{preview = value as SplinePreview;} }
  public SplinePreview preview;

  public override Matrix Matrix => PreviewManager.heightOutputPreview?.matrix;
  public override Texture2D PreviewTexture => PreviewManager.heightOutputPreview?.tex;
```

```
public ulong SerializedGenId
{
 get{ return serializedGenId; }
 set{ serializedGenId = value; }
}
public ulong serializedGenId;



protected override void DrawPreview ()
{
 base.DrawPreview();

 if (preview.polyLine?.mesh == null)
 {
  PreviewDraw.DrawGenerateMarkInWindow(PreviewStage.Blank, Vector2.zero);
  return;
 }

 CoordRect matrixRect = PreviewManager.heightOutputPreview?.matrix!=null ?
  PreviewManager.heightOutputPreview.matrix.rect :
  new CoordRect(0,0,0,0);
 CoordRect activeRect = PreviewManager.heightOutputPreview?.matrix!=null ?
  PreviewManager.heightOutputPreview.activeRect :
  new CoordRect(0,0,0,0);
```

```
//height background
using (Cell.Custom( ToMatrixRect(matrixRect) ))
{

 Texture2D heightTex = PreviewManager.heightOutputPreview!=null ? PreviewManager.heightOutputPr
 Draw.MatrixPreviewTexture(heightTex, colorize:false, relief:true);
}

//preview itself
using (Cell.Custom( ToMatrixRect(activeRect) ))
{
 if (lineTex == null) lineTex = Resources.Load("DPUI/PolyLineTex") as Texture2D;
 if (lineMat == null)
 {
  lineMat = new Material( Shader.Find("Hidden/DPLayout/PolyLine") );

  lineMat.SetTexture("_MainTex", lineTex);
  lineMat.SetColor("_Color", guiGizmoColor);
  lineMat.SetFloat("_Width", 1.5f);
  //lineMat.SetFloat("_Offset", offset);
  //lineMat.SetFloat("_NumPoints", numPoints-1);
  //lineMat.SetFloat("_Dotted", dottedSpace);
 }

 Draw.Mesh(preview.polyLine.mesh, lineMat, clip:false);
}
```

```
   PreviewDraw.DrawGenerateMarkInWindow(preview.Stage, MatrixRect.center);
   if (PreviewManager.heightOutputPreview!=null)
    PreviewDraw.DrawGenerateMarkInWindow(PreviewManager.heightOutputPreview.Stage, MatrixRect.c
  }
 }

#endregion

#region GUI

 public void DrawInGraph ()
 {
  if (splineSys == null  ||  polyLine == null  ||  polyLine.mesh == null) { Draw.Rect(PreviewDraw.Backgroun

  //height background
  Texture2D heightTex = PreviewManager.heightOutputPreview!=null ? PreviewManager.heightOutputPre
  Draw.MatrixPreviewTexture(heightTex, colorize:false, relief:true);

  //preview itself
  if (Event.current.type != EventType.Repaint) return;

  if (lineTex == null) lineTex = Resources.Load("DPUI/PolyLineTex") as Texture2D;
  if (lineMat == null)
  {
   lineMat = new Material( Shader.Find("Hidden/DPLayout/PolyLine") );
```

```
      lineMat.SetTexture("_MainTex", lineTex);
      lineMat.SetColor("_Color", guiGizmoColor);
      lineMat.SetFloat("_Width", 1.5f);
      //lineMat.SetFloat("_Offset", offset);
      //lineMat.SetFloat("_NumPoints", numPoints-1);
      //lineMat.SetFloat("_Dotted", dottedSpace);
     }

     Draw.Mesh(polyLine.mesh, lineMat);

     //DrawBeizer();
    }


   #endregion
  }
 }
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;

using Den.Tools;
using Den.Tools.Matrices;
using MapMagic.Nodes;
using MapMagic.Terrains;
using MapMagic.Core;

namespace MapMagic.Products
{
 public delegate void FinalizeAction (TileData data, StopToken stop);
 public delegate void ApplyAction (TileData data, Terrain terrain, StopToken stop);

 [Serializable, StructLayout (LayoutKind.Sequential)] //to pass to native
 public class StopToken
 /// Data is the same in all chunk's threads. Stop token is unique per thread.
 {
  public bool stop;
  public bool restart;
 }

 public class TileData
```

```
{
 public Area area;
 public Globals globals;
 public Noise random; //a ref to the one in parentGraph (to make it possible generate with no graph)

 public ulong graphChangeVersion; //copy from graph to find if cluster should be re-generated
 public int graphId;

 public bool isPreview; //should this generate be used as a preview?
 public bool isDraft; //is this terrain low-detail (to avoid applying objects and grass)?

 //per-graph products
 private Dictionary<ulong,ulong> lastVersion = new Dictionary<ulong, ulong>();
 private Dictionary<ulong,float> progress = new Dictionary<ulong,float>();
 private Dictionary<ulong,object> prepare = new Dictionary<ulong, object>();
 private Dictionary<ulong,object> products = new Dictionary<ulong,object>(); //per-outlet intermediate resu
 private Dictionary<ulong, (object,object,object)> outputs = new Dictionary<ulong, (object,object,object)>()

 //per-tile products (refs copied to subDatas)
 private HashSet<FinalizeAction> finalizeMarks = new HashSet<FinalizeAction>();
 private Dictionary<Type,IApplyData> applyMarks = new Dictionary<Type,IApplyData>(); //finalize marks a

 public MatrixWorld heights = null; //last heights applied to floor objects

 //sub-datas
 public Dictionary<ulong,TileData> subDatas = new Dictionary<ulong,TileData>();
```

```
public TileData parentData = null;

public MatrixWorld biomeMask = null;

//IDEA: these and products are the only thing that changes in biome sub-data.

//finalizeMarks and applyMarks are per-tile, not per-biome (using root to store and read them)

//Maybe send it with separate argument (and make sub-datas in product set?)




#region Hierarchy


 public TileData CreateLoadSubData (ulong biomeId, MatrixWorld mask)

 /// Creates/loads sub-data and assigns biome mask

 {

  if (!subDatas.TryGetValue(biomeId, out TileData subData))  //sometimes stored area is null on clearing,

  {

   subData = (TileData)this.MemberwiseClone();

   //taking finalize, apply, height by reference from parent data


   subData.lastVersion = new Dictionary<ulong, ulong>();

   subData.products = new Dictionary<ulong, object>();

   subData.prepare = new Dictionary<ulong, object>();

   subData.products = new Dictionary<ulong, object>();

   subData.outputs = new Dictionary<ulong, (object, object, object)>();


   subData.subDatas = new Dictionary<ulong,TileData>();
```

```
  subData.parentData = this;


  subDatas.Add(biomeId, subData);
 }


 subData.biomeMask = mask;


 if (subData.area == null) subData.area = area;
 if (subData.globals == null) subData.globals = globals;
 if (subData.random == null) subData.random = random;
 //these were added to fix node remove bug when using function. For some reason stored data on clearin


 return subData;
}



public TileData CreateLoadSubData (ulong biomeId)  =>  CreateLoadSubData(biomeId, this.biomeMask)
/// Creates/loads sub-data, but takes current mask reference (for function)



public TileData GetSubData (ulong biomeId)
/// Creates (or loads) sub-data for this biome unit
{
 subDatas.TryGetValue(biomeId, out TileData subData);
 return subData;
}
```

```
public TileData Root
{get{
 if (parentData != null)
  return parentData.Root;
 else
  return this;
}}


public IEnumerable<TileData> AllSubs (bool includeItself=false)
{
// if (includeItself)
//  yield return this;

 foreach (var kvp in subDatas)  //top-level first
  yield return kvp.Value;

 foreach (var kvp in subDatas)
 {
  TileData subData = kvp.Value;
  foreach (TileData subSub in subData.AllSubs())
   yield return subSub;
 }
}
```

```
#endregion

    #region Version Ready

 public bool IsReady (ulong genId, ulong version)
 {
  bool inDict = lastVersion.TryGetValue(genId, out ulong lastVer);
  if (inDict)
   return version == lastVer;
  return false;
 }

 public bool IsReady (Generator gen)
 {
  bool inDict = lastVersion.TryGetValue(gen.id, out ulong lastVer);
  if (inDict)
   return gen.version == lastVer;
  return false;
 }

 public bool VersionChanged (Generator gen)
 /// True only if this node version was changed (to determine this node change for biome)
 {
  bool inDict = lastVersion.TryGetValue(gen.id, out ulong lastVer);
  if (inDict)
```

```
  return true;
 return gen.version == lastVer;
}

#if MM_DEBUG
public ulong? Version (Generator gen)
{
 bool inDict = lastVersion.TryGetValue(gen.id, out ulong lastVer);
 if (!inDict)
  return null;
 else
  return lastVer;
}
#endif

public void MarkReady (ulong genId, ulong version)
{
 if (!lastVersion.ContainsKey(genId))
  lastVersion.Add(genId, version);
 else
  lastVersion[genId] = version;
}

public void MarkReady (Generator gen)
{
 if (!lastVersion.ContainsKey(gen.id))
```

```
  lastVersion.Add(gen.id, gen.version);
 else
  lastVersion[gen.id] = gen.version;
}

public void ClearReady (ulong genId)  =>  lastVersion.Remove(genId);

public void ClearReady (Generator gen)  =>  lastVersion.Remove(gen.id);

public bool AreAllRelevantReady (Graph graph, bool inSubs=false)
{
 foreach (Generator gen in graph.RelevantGenerators(isDraft))
 {
  bool inDict = lastVersion.TryGetValue(gen.id, out ulong lastVer);

  if (!inDict)
   return false;
  else
   if (gen.version != lastVer)
    return false;
 }

 if (inSubs)
  foreach (IBiome biome in graph.UnitsOfType<IBiome>())
  {
   Graph subGraph = biome.SubGraph;
```

```
   if (subGraph == null)
    continue;

   TileData subData;
   if (!subDatas.TryGetValue(biome.Id, out subData))
    continue;

   if (!subData.AreAllRelevantReady(subGraph, inSubs))
    return false;
  }

 return true;
}



public bool AllOutputsReady (Graph graph, OutputLevel level, bool inSubs=false)
/// Are all enabled output nodes with level of Level or Both are marked as ready in data
/// Similar to AreAllRelevantReady, but using proven OutputLevel
{
 foreach (OutputGenerator outGen in graph.GeneratorsOfType<OutputGenerator>())
 {
  //if enabled level output is NOT ready
  if (outGen.enabled &&
   outGen.OutputLevel.HasFlag(level))
   {
    bool inDict = lastVersion.TryGetValue(outGen.id, out ulong lastVer);
```

```
   if (!inDict)
    return false;
   else
    if (outGen.version != lastVer)
     return false;
  }
}

if (inSubs)
 foreach (IBiome biome in graph.UnitsOfType<IBiome>())
 {
  Graph subGraph = biome.SubGraph;
  if (subGraph == null)
   continue;

  //HACK: check biome for outputs is right, but not always work with current clear/ready system
  //Fails in draft when cleared while generating biome - some internal outputs might not be ready
  //While biome switches to ready on finish, and it is skipped with next wave of generate (after clear)
  //So drafts are using hacky simplified check
  //Update: might work okay with new clear sys
  if (isDraft)
  {
   if (!IsReady(biome.Gen))
    return false;
  }
```

```
   else
   {
    TileData subData;
    if (!subDatas.TryGetValue(biome.Id, out subData))
     continue;

    if (!subData.AllOutputsReady(subGraph, level, inSubs))
     return false;
   }
  }

 return true;
}



public string[] DebugReadyVar => DebugReady(Graph.debugGraph, false);
public string[] DebugReady (Graph rootGraph, bool useGraphName=true)
/// Returns the names of graphs, generators and layers that are marked as ready
/// Disable graph name when running from thread
{
 string[] debugs = new string[lastVersion.Count];

 int i=0;
 foreach (ulong id in lastVersion.Keys)
 {
```

```
   debugs[i] = rootGraph.DebugUnitName(id,useGraphName);
   if (debugs[i] == null)
    debugs[i] = $"Null id:{Id.ToString(id)}";

   i++;
  }

  return debugs;
 }

#endregion

      #region Progress

      public void SetProgress (ICustomComplexity cc, float progress)
 {
  Generator gen = (Generator)cc;
  if (this.progress.ContainsKey(gen.id)) this.progress[gen.id] = progress;
  else this.progress.Add(gen.id, progress);
 }

 public float GetProgress (ICustomComplexity cc)
 {
  Generator gen = (Generator)cc;
  if (this.progress.TryGetValue(gen.id, out float progress)) return progress;
  else return 0;
```

```
}

#endregion

#region Prepare

public T ReadPrepare<T> (Generator gen) where T: class  =>  (T)ReadPrepare(gen.id);

public object ReadPrepare (ulong genId)
{
 if (prepare.TryGetValue(genId, out object obj)) return obj;
 return null;
}

public void StorePrepare (ulong genId, object obj)
{
 if (obj == null)
  RemovePrepare (genId);

 if (prepare.ContainsKey(genId)) prepare[genId] = obj;
 else prepare.Add(genId, obj);
}

public void RemovePrepare (Generator gen)   =>  RemoveProduct(gen.id);

public void RemovePrepare (ulong genId)  =>  prepare.Remove(genId);
```

```
 public int PrepareCount => prepare.Count;

 public void ClearPrepare () => prepare.Clear();

#endregion

#region Products

 public T ReadInletProduct<T> (IInlet<T> inlet) where T: class =>  (T)ReadProduct(inlet.LinkedOutletId);
 public T ReadOutletProduct<T> (IOutlet<T> outlet) where T: class  =>  (T)ReadProduct(outlet.Id);

 public object ReadProduct (ulong outletId)
 {
  if (outletId == 0) return null; //not connected

  if (products.TryGetValue(outletId, out object obj)) return obj;
  return null;
 }

 public void StoreProduct<T> (IOutlet<T> outlet, T product) where T: class =>  StoreProduct(outlet.Id, pro

 public void StoreInletProduct (IInlet<object> inlet, object product)  =>
  //for generator tester
  StoreProduct(inlet.Id, product);
```

```
public void StoreProduct (ulong outletId, object obj)
{
 if (obj == null)
  RemoveProduct (outletId);

 if (products.ContainsKey(outletId)) products[outletId] = obj;
 else products.Add(outletId, obj);
}


public void RemoveProduct (IOutlet<object> outlet)   =>  RemoveProduct(outlet.Id);

public void RemoveProduct (ulong outletId)  =>  products.Remove(outletId);

public int ProductsCount  =>  products.Count;

public void ClearProducts ()  => products.Clear();

public string[] DebugProductsVar => DebugProducts(Graph.debugGraph, false);
public string[] DebugProducts (Graph rootGraph, bool useGraphName=true)
/// Returns the names of graphs, generators and layers that have stored products
/// Disable graph name when running from thread
{
 string[] debugs = new string[products.Count];
```

```
 int i=0;
 foreach (ulong id in products.Keys)
 {
  string unitName = rootGraph.DebugUnitName(id,useGraphName);
  if (unitName == null) unitName = "Null";

  debugs[i] = unitName + " product:" + products[id];
  i++;
 }

 return debugs;
}

#endregion

#region Outputs

 public void StoreOutput (IUnit unit, object key, object prototype, object product) =>
  StoreOutput(unit.Id, key, prototype, product);
  //key is any kind of object. Action, type, string, anything. Returns output if keys in get and store are equa
  //prototype: generator or layer

 public void StoreOutput (ulong outputId, object key, object prototype, object obj)
 {
  if (outputs.ContainsKey(outputId)) outputs[outputId] = (key, prototype, obj);
  else outputs.Add(outputId, (key, prototype, obj));
```

```
}


public IEnumerable<(TPrototype,TProduct,MatrixWorld)> Outputs<TPrototype,TProduct,TMask> (object
{
 foreach ((object prototype, object product, MatrixWorld mask) in Outputs(key, inSubs:inSubs))
  yield return ((TPrototype)prototype, (TProduct)product, mask);
}


public void GatherOutputs<TPrototype,TProduct> (object key, out TPrototype[] prototypes, out TProduct[
where TProduct: class
//For all textures outputs - they blend arrays of matrices
{
 List<TPrototype> prototypesList = new List<TPrototype>();
 List<TProduct> productsList = new List<TProduct>();
 List<MatrixWorld> masksList = new List<MatrixWorld>();

 foreach ((object prototype, object product, MatrixWorld mask) in Outputs(key, inSubs:inSubs))
 {
  prototypesList.Add((TPrototype)prototype);
  productsList.Add((TProduct)product);
  masksList.Add(mask);
 }

 prototypes = prototypesList.ToArray();
```

```
 products = productsList.ToArray();
 masks = masksList.ToArray();
}



public IEnumerable<(object prototype, object product, MatrixWorld biomeMask)> Outputs (object key, bo
{
 foreach (var kvp in outputs)
 {
  (object k, object p, object o) = kvp.Value;
  if (k==key)
   yield return (p, o, biomeMask);
 }

 if (inSubs)
  foreach (TileData sub in subDatas.Values)
   foreach (var ppb in sub.Outputs(key, true))
    yield return ppb;
}



public int OutputsCount (object key, bool inSubs=false)
{
 int count = 0;
```

```
 foreach (var kvp in outputs)
 {
  (object k, object p, object o) = kvp.Value;
  if (k==key)
   count++;
 }

 if (inSubs)
  foreach (var kvp in subDatas)
   count += kvp.Value.OutputsCount(key, true);

 return count;
}



public void RemoveOutput (ulong id, bool inSubs=false)
{
 outputs.Remove(id);

 if (inSubs)
  foreach (var kvp in subDatas)
   kvp.Value.RemoveOutput(id, inSubs);
}

#endregion
```

```
#region Finalize / Apply

 public void MarkFinalize (FinalizeAction action, StopToken stop)
 {
  if (!finalizeMarks.Contains(action))
   //lock (finalizeMarks)
   {
    //Debug.Log("Adding " + finalizeMarks.GetVersion() + " stop:" + stop.stop);
    finalizeMarks.Add(action);
   }
 }

 public void RemoveFinalize (FinalizeAction action)   => finalizeMarks.Remove(action);

 public void ClearFinalize ()
 {
  //lock (finalizeMarks)
  {
   //Debug.Log("ClearFinalize " + finalizeMarks.GetVersion());
   finalizeMarks.Clear();
  }
 }

 public IEnumerable<FinalizeAction> MarkedFinalizeActions (StopToken stop) //finalize and apply are retu
 {
  lock (finalizeMarks)
```

```
 {
  FinalizeAction heightAction = Nodes.MatrixGenerators.HeightOutput200.finalizeAction;

  if (finalizeMarks.Contains(heightAction))
   yield return heightAction;

  foreach (FinalizeAction action in finalizeMarks)
  {
   Debug.Log("Iterating " + finalizeMarks.GetVersion() + " stop:" + stop.stop);

   if (action != heightAction)
    yield return action;
  }
 }
}

public int FinalizeMarksCount => finalizeMarks.Count;

public FinalizeAction DequeueFinalize ()
/// Some finalize actions might be added while finalizing (like objects while finalizing height)
/// And this is thread safe (IEnumerable isn't)
/// So using this instead of IEnumerable
{
 //lock (finalizeMarks)
 {
  //returnning height
```

```
  FinalizeAction heightAction = Nodes.MatrixGenerators.HeightOutput200.finalizeAction;
  if (finalizeMarks.Contains(heightAction))
  {
   finalizeMarks.Remove(heightAction);
   return heightAction;
  }

  //returning other actions
  FinalizeAction action = null;
  foreach (FinalizeAction iAction in finalizeMarks)
   { action = iAction; break; }

  finalizeMarks.Remove(action);
  return action;
 }
}



public void MarkApply (IApplyData data)
{
 Type type = data.GetType();
 lock (applyMarks) //added from thread, but read from main
 {
  if (applyMarks.ContainsKey(type))
   applyMarks[type] = data;
  else
```

```
   applyMarks.Add(type, data);
 }
}

public int ApplyMarksCount  =>  applyMarks.Count;

public T ApplyOfType<T> () where T: class,IApplyData
{
 if (applyMarks.TryGetValue(typeof(T), out IApplyData data))
  return (T)data;
 else return null;
}

public IApplyData DequeueApply ()
{
 IApplyData topPriorityApply = null;
 int topPriority = -1;

 lock (applyMarks) //otherwise can cause "Collection was modified; enumeration operation may not execu
  foreach (IApplyData data in applyMarks.Values)
  //TODO: enumerators here (especially this one)
  {
   int priority = GetApplyPriority(data);
   if (priority > topPriority)
    { topPriority=priority; topPriorityApply=data; }
  }
```

```
  applyMarks.Remove(topPriorityApply.GetType());
  return topPriorityApply;
 }

 private int GetApplyPriority (IApplyData data)
 {
  switch (data)
  {
   case Nodes.MatrixGenerators.HeightOutput200.ApplySetData setApply: return 10;
   case Nodes.MatrixGenerators.HeightOutput200.ApplySplitData splitApply: return 9;
   case Nodes.MatrixGenerators.HeightOutput200.ApplyTexData texApply: return 8;
   case Nodes.MatrixGenerators.TexturesOutput200.ApplyData texturesApply: return 7;
   case Nodes.MatrixGenerators.GrassOutput200.ApplyData texturesApply: return 6;
  }
  return 0;
 }

#endregion



public void Clear (bool clearApply=true, bool inSubs=false)
/// Clears all of the unnecessary data in playmode
{
 lastVersion.Clear();
```

```
 products.Clear();
 prepare.Clear();
 outputs.Clear();

 lock (finalizeMarks)
  finalizeMarks.Clear();

 if (clearApply) applyMarks.Clear();

 heights = null; //if (heights != null) heights.Clear(); //clear is faster, but tends to miss an error

 if (inSubs)
 {
  foreach (TileData subData in subDatas.Values)
   subData.Clear(clearApply:clearApply, inSubs:inSubs);
  subDatas.Clear();
 }
}


public void Remove (Generator gen, bool inSubs=false)
{
 lastVersion.Remove(gen.id);
 products.Remove(gen.id);
 prepare.Remove(gen.id);
 outputs.Remove(gen.id);
```

```
 if (gen is IMultiOutlet mulOutGen)
  foreach (IOutlet<object> outlet in mulOutGen.Outlets())
   foreach (TileData set in AllSubs())
    set.products.Remove(outlet.Id);

 if (inSubs)
  foreach (TileData subData in subDatas.Values)
   subData.Remove(gen, inSubs:inSubs);
}



public void ClearStray (Graph graph, bool inSubs=false)
/// Clears products that are not in graph
/// Performance checked: 40ms per 1000 iterations on TutorialSplines graph
{
 HashSet<ulong> unitsIds = new HashSet<ulong>();
 foreach (IUnit unit in graph.AllUnits())
  unitsIds.Add(unit.Id);

 lastVersion.RemoveNotContained(unitsIds);
 products.RemoveNotContained(unitsIds);
 prepare.RemoveNotContained(unitsIds);
 outputs.RemoveNotContained(unitsIds);

 if (inSubs)
```

```
    foreach ((Graph subGraph, TileData subData) in Graph.AllGraphsDatas(graph,this))
      subData.ClearStray(subGraph, inSubs:inSubs);
  }



 }
}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using MapMagic.Nodes;

namespace MapMagic.Terrains
{
  [Serializable]
  public class Area
  {
   [Serializable]
   public class Dimensions
   {
    public CoordRect rect;
    public Vector2D worldPos;
    public Vector2D worldSize;

    public Dimensions () {} //for serializer

    public Dimensions (CoordRect rect, Vector2D worldPos, Vector2D worldSize)
     { this.rect = rect; this.worldPos = worldPos; this.worldSize = worldSize; }

    public Dimensions (Vector2D worldPos, Vector2D worldSize, int resolution)
```

```
{
 this.worldPos = worldPos; this.worldSize = worldSize;

 Vector3 pixelSize = 1f * (Vector3)worldSize / resolution;
 Coord offset = new Coord( Mathf.FloorToInt(worldPos.x / pixelSize.x), Mathf.FloorToInt(worldPos.z / pi
 this.rect = new CoordRect(offset, new Coord(resolution,resolution));
}

public Vector3 PixelSize {get{ return new Vector3(worldSize.x/(rect.size.x-1), 1, worldSize.z/(rect.size.z-

public override string ToString () { return "Rect:" + rect.ToString() + " Pos:" + worldPos + " Size:" + worlc

public Vector3 CoordToWorld (int x, int z)
{
 //finding relative percent
 float percentX = 1f * (x - rect.offset.x) / rect.size.x;
 float percentZ = 1f * (z - rect.offset.z) / rect.size.z;

 //get map coordinates
 float worldX = percentX*worldSize.x + worldPos.x;
 float worldZ = percentZ*worldSize.z + worldPos.z;

 return new Vector3(worldX, 0, worldZ);
}

public bool Contains (Vector3 pos)
```

```
 {
  if (pos.x < worldPos.x || pos.x > worldPos.x+worldSize.x ||
    pos.z < worldPos.z || pos.z > worldPos.z+worldSize.z) return false;
  return true;
 }
}

public Dimensions active;
public Dimensions full;

public int Margins {get{ return (full.rect.size.x-active.rect.size.x)/2; }}
public Vector3 PixelSize {get{ return active.PixelSize; }}
public override string ToString() { return "Active:(" + active.ToString() + "), Full:(" + full.ToString() + ")"; }

public Coord Coord { get{ return new Coord(active.rect.offset.x/active.rect.size.x, active.rect.offset.z/acti


private static CoordRect WorldToPixels (Vector3 worldPos, Vector3 worldSize, int resolution)
{
 Vector3 pixelSize = 1f * worldSize / resolution;
 Coord activeOffset = new Coord( Mathf.FloorToInt(worldPos.x / pixelSize.x), Mathf.FloorToInt(worldPos
 return new CoordRect(activeOffset, new Coord(resolution,resolution));
}


private static Dimensions GetFullDimensions (Dimensions active, int margins)
```

```
{
 Vector3 pixelSize = active.PixelSize;
 return new Dimensions(
  rect: new CoordRect( new Coord(active.rect.offset.x-margins, active.rect.offset.z-margins), new Coord(
  worldPos: new Vector2D(active.worldPos.x - margins*pixelSize.x, active.worldPos.z - margins*pixelSiz
  worldSize: new Vector2D(active.worldSize.x + margins*pixelSize.x*2, active.worldSize.z + margins*pix
}

private static Dimensions GetActiveDimensions (Dimensions full, int margins)
{
 Vector3 pixelSize = full.PixelSize;
 return new Dimensions(
  rect: new CoordRect(full.rect.offset.x+margins, full.rect.offset.z+margins, full.rect.size.x-margins*2, full.
  worldPos: new Vector2D(full.worldPos.x + margins*pixelSize.x, full.worldPos.z + margins*pixelSize.z),
  worldSize: new Vector2D(full.worldSize.x - margins*pixelSize.x*2, full.worldSize.z - margins*pixelSize.z
}



public Area () { } //for serializer

public Area (Vector2D activeWorldPos, Vector2D activeWorldSize, CoordRect activePixelRect, int margi
{
 active = new Dimensions(
  rect: activePixelRect,
  worldPos: activeWorldPos,
  worldSize: activeWorldSize);
```

```
 full = GetFullDimensions(active, margins);
}

public Area (CoordRect activeWorldRect, CoordRect activePixelRect, int margins)
{
 active = new Dimensions(
  rect: activePixelRect,
  worldPos: activeWorldRect.offset.vector2d,
  worldSize: activeWorldRect.size.vector2d);

 full = GetFullDimensions(active, margins);
}

public Area (Vector2D activeWorldPos, Vector2D activeWorldSize, int activeResolution, int margins)
{
 active = new Dimensions(
  rect: WorldToPixels((Vector3)activeWorldPos, (Vector3)activeWorldSize, activeResolution),
  worldPos: activeWorldPos,
  worldSize: activeWorldSize);

 full = GetFullDimensions(active, margins);
}

public Area (Terrain terrain, int margins=2)
{
```

```
 Vector2D terrainPos = (Vector2D)terrain.transform.position;
 Vector2D terrainSize = (Vector2D)terrain.terrainData.size;

 active = new Dimensions(
  rect: WorldToPixels((Vector3)terrainPos, (Vector3)terrainSize, terrain.terrainData.heightmapResolution
  worldPos: terrainPos,
  worldSize: terrainSize);

 full = GetFullDimensions(active, margins);
}

public Area (Coord coord, int activeResolution, int margins, Vector2D activeWorldSize)
{
 active = new Dimensions(
  rect: new CoordRect( coord.x*activeResolution, coord.z*activeResolution, activeResolution, activeReso
  worldPos: new Vector2D(coord.x*activeWorldSize.x, coord.z*activeWorldSize.z),
  worldSize: new Vector2D(activeWorldSize.x, activeWorldSize.z));

 full = GetFullDimensions(active, margins);
}

public Area (Area src)
{
 active = new Dimensions(src.active.rect, src.active.worldPos, src.active.worldSize);
 full = new Dimensions(src.full.rect, src.full.worldPos, src.full.worldSize);
}
```

```
public static Area FromActive (Vector2D activeWorldPos, Vector2D activeWorldSize, int activeResolutio
{
 Area area = new Area();
 area.active = new Dimensions(
  rect: WorldToPixels((Vector3)activeWorldPos, (Vector3)activeWorldSize, activeResolution),
  worldPos: activeWorldPos,
  worldSize: activeWorldSize);
 area.full = GetFullDimensions(area.active, margins);
 return area;
}

public static Area FromFull (Vector2D fullWorldPos, Vector2D fullWorldSize, int fullResolution, int margi
{
 Area area = new Area();
 area.full = new Dimensions(
  rect: WorldToPixels((Vector3)fullWorldPos, (Vector3)fullWorldSize, fullResolution),
  worldPos: fullWorldPos,
  worldSize: fullWorldSize);
 area.active = GetActiveDimensions(area.full, margins);
 return area;
}


public void MoveTo (Vector2D activeWorldPos)
{
```

```
      active = new Dimensions(
       rect: WorldToPixels((Vector3)activeWorldPos, (Vector3)active.worldSize, active.rect.size.x),
       worldPos: activeWorldPos,
       worldSize: active.worldSize);

      full = GetFullDimensions(active, Margins);
     }

     public void MoveTo (Coord coord, int activeResolution, Vector2D activeWorldPos)
     {
      active.rect = new CoordRect( coord.x*activeResolution, coord.z*activeResolution, activeResolution, acti
      active.worldPos = new Vector2D(coord.x*activeWorldPos.x, coord.z*activeWorldPos.z);
      active.worldSize = activeWorldPos;

      full = GetFullDimensions(active, Margins);
     }
    }
}
```

```
using System;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools;
using Den.Tools.Matrices;

namespace MapMagic.Terrains
{
 [ExecuteInEditMode]
 public class DirectMatricesHolder : MonoBehaviour, ISerializationCallbackReceiver
 /// For each tile stores dictionary of matrices generated by Direct Maps Output node
 {
  [NonSerialized] public DictionaryOrdered<string,MatrixWorld> maps = new DictionaryOrdered<string,Matr

  #region Matrix Helpers

   private MatrixWorld this[string name]
   {get{
    if (maps.TryGetValue(name, out MatrixWorld matrix))
     return matrix;
    else
     throw new Exception($"Matrix with the name '{name}' is not found in dictionary");
   }}
```

```
private static void CheckPosition (MatrixWorld matrix, float x, float z)
{
 if (!matrix.ContainsWorldValue(x,z))
  throw new Exception($"Position {x},{z} is out of matrix bounds");
}


/// Checks if maps contain position given in world units (X and Z only).
public bool ContainsPosition (float x, float z)
{
 if (maps.Count == 0)
  return false;
 else
  return maps[0].ContainsWorldValue(x,z);
}

public bool ContainsPosition (Vector3 pos)
{
 if (maps.Count == 0)
  return false;
 else
  return maps[0].ContainsWorldValue(pos.x, pos.z);
}


/// Selects the matrix by name (or number) and returns value at position given in world units.
```

```
/// Position is floored to pixels, no interpolation
public float ValueAtPosition (string name, float x, float z)  =>  this[name].GetWorldValue(x,z);
public float ValueAtPosition (string name, Vector3 pos)  =>  this[name].GetWorldValue(pos.x, pos.z);
public float ValueAtPosition (int num, float x, float z)  =>   maps[num].GetWorldValue(x,z);
public float ValueAtPosition (int num, Vector3 pos)  =>   maps[num].GetWorldValue(pos.x, pos.z);


/// Selects the matrix by number and returns value at position given in world units.
/// Position is interpolated between closest pixels. Use this for precise operations.
public float ValueAtPositionInterpolated (string name, float x, float z)  =>  this[name].GetWorldInterpolate
public float ValueAtPositionInterpolated (string name, Vector3 pos)  =>  this[name].GetWorldInterpolated
public float ValueAtPositionInterpolated (int num, float x, float z)  =>   maps[num].GetWorldInterpolatedVa
public float ValueAtPositionInterpolated (int num, Vector3 pos)  =>   maps[num].GetWorldInterpolatedVal


/// Finds all values at given world position, and returns them as name -> value dictionary. No interpolation
public Dictionary<string,float> AllValuesAtPosition (float x, float z)
{
 Dictionary<string,float> result = new Dictionary<string, float>();
 foreach (var kvp in maps)
 {
  MatrixWorld matrix = kvp.Value;
  CheckPosition(matrix, x, z);

  result.Add(kvp.Key, matrix.GetWorldValue(x,z));
 }
```

```
  return result;
 }

#endregion



#region Examples

 /// Finds proper holder based on coordinate world position.
 /// Null if not found.
 /// Uses GameObject.FindObjectOfType so it's not quick. Just an example.
 public static DirectMatricesHolder FindHolder (float x, float z)
 {
  DirectMatricesHolder[] holders = GameObject.FindObjectsOfType<DirectMatricesHolder>();

  foreach (DirectMatricesHolder holder in holders)
  {
   if (holder.ContainsPosition(x,z))
    return holder;
  }

  return null;
 }
```

```
/// Finds a proper holder and evaluates position on a map with given name.
/// In case we want to know biome or land texture value player currently in/on.
/// 0 if player is out of any terrain.
public static float FindValueAtPosition (string name, float x, float z)
{
 DirectMatricesHolder holder = FindHolder(x, z);

 if (holder != null)
  return holder.ValueAtPosition(name, x, z);
 else
  return 0;
}

/// Finds a proper holder and evaluates position on a map with given name.
/// Iterates directly on MapMagic tiles, so might be faster than previous one (however I have not checked)
public static float FindValueAtPosition (MapMagic.Core.MapMagicObject mapMagicObject, string name,
{
 TerrainTile tile = mapMagicObject.tiles.FindByWorldPosition(x, z);
 if (tile == null)
  return 0;

 DirectMatricesHolder holder = tile.ActiveTerrain.GetComponent<DirectMatricesHolder>();
 if (holder == null)
  return 0;

 return holder.ValueAtPosition(name, x, z);
```

```
    }

  #endregion

  #region Serialization
  [SerializeField] private string[] serNames;
    [SerializeField] private MatrixWorld[] serMaps;
    public void OnBeforeSerialize () => (serNames,serMaps) = maps.Serialize();
    public void OnAfterDeserialize () => maps.Deserialize(serNames,serMaps);
  #endregion
 }
}
```

```
using System;
using UnityEngine;

using Den.Tools;

namespace MapMagic.Terrains
{
 [ExecuteInEditMode]
 public class DirectTexturesHolder : MonoBehaviour, ISerializationCallbackReceiver
 /// Short version of MaterialPropertySerializer
 /// For each tile stores an array of textures generated by Direct Textures Output node
 {
  public Vector2D position;
  public Vector2D size;
  [NonSerialized] public DictionaryOrdered<string,Texture2D> textures = new DictionaryOrdered<string, Te

  /// Checks whether this holder contains the world position
  public bool ContainsPosition(float x, float z)
  {
   return  x > position.x  &&   x < position.x+size.x  &&
     z > position.z  &&   z < position.z+size.z;
  }

  /// Returns texture by name, null if not found.
  public Texture2D this[string name]
  {get{
```

```
 if (textures.TryGetValue(name, out Texture2D tex))
  return tex;
 else
  return null;
}}

#region Examples

 /// Finds proper holder based on coordinate world position.
 /// Null if not found.
 /// Uses GameObject.FindObjectOfType so it's not quick. Just an example.
 public static DirectTexturesHolder FindHolder (float x, float z)
 {
  DirectTexturesHolder[] holders = GameObject.FindObjectsOfType<DirectTexturesHolder>();

  foreach (DirectTexturesHolder holder in holders)
  {
   if (holder.ContainsPosition(x,z))
    return holder;
  }

  return null;
 }


 /// Finds a proper holder and evaluates position on a map with given name.
```

```
/// In case we want to know biome or land texture value player currently in/on.
/// 0 if player is out of any terrain.
public static Texture2D FindTexture (string name, float x, float z)
{
 DirectTexturesHolder holder = FindHolder(x, z);

 if (holder != null)
  return holder[name];
 else
  return null;
}

/// Finds a proper holder and evaluates position on a map with given name.
/// Iterates directly on MapMagic tiles, so might be faster than previous one (however I have not checked)
public static Texture2D FindTexture (MapMagic.Core.MapMagicObject mapMagicObject, string name, flo
{
 TerrainTile tile = mapMagicObject.tiles.FindByWorldPosition(x, z);
 if (tile == null)
  return null;

 DirectTexturesHolder holder = tile.ActiveTerrain.GetComponent<DirectTexturesHolder>();
 if (holder == null)
  return null;

 return holder[name];
}
```

```
  #endregion


  #region Serialization
   [SerializeField] private string[] serNames;
   [SerializeField] private Texture2D[] serTextures;
   public void OnBeforeSerialize () => (serNames,serTextures) = textures.Serialize();
   public void OnAfterDeserialize () => textures.Deserialize(serNames,serTextures);
  #endregion
 }
}
```

```
ï»¿using System;
using UnityEngine;

using Den.Tools;

namespace MapMagic.Terrains
{
 [ExecuteInEditMode]
 public class MaterialPropertySerializer : MonoBehaviour
 /// Serializes the assigned properties. To serialize property it should be assigned via MaterialPropertySeria
 {
  [SerializeField] private Terrain terrain;
  [SerializeField] private string[] names = new string[0];
  [SerializeField] private Texture2D[] textures = new Texture2D[0];

  [NonSerialized] private MaterialPropertyBlock matProps;


  /*public void OnEnable()
  {
   if (terrain == null) return;

   //terrain.GetSplatMaterialPropertyBlock(matProps);
   matProps = new MaterialPropertyBlock();

   for (int i=0; i<textures.Length; i++)
```

```
  matProps.SetTexture(names[i], textures[i]);

 terrain.SetSplatMaterialPropertyBlock(matProps);

 //re-enabling terrain. Otherwise Unity will crash on changing material values
 //terrain.enabled = false;
 //terrain.enabled = true;
}*/


//public void OnEnable () //called after each lod change, causing blinking
public void Start ()
{
 if (terrain == null) return;

 matProps = new MaterialPropertyBlock();

 for (int i=0; i<textures.Length; i++)
  matProps.SetTexture(names[i], textures[i]);

 terrain.SetSplatMaterialPropertyBlock(matProps);

 #if UNITY_EDITOR

  #if UNITY_2019_3_OR_NEWER
  terrain.enabled = false;
```

```
  UnityEditor.SceneView.RepaintAll();
  terrain.enabled = true;
  #else
  updateVisibility = true;
  #endif

 #endif
}

//Re-enabling terrain (in next frame?) Otherwise Unity will crash on changing material values
public bool updateVisibility = false;
public void OnDrawGizmos ()
{
 if (updateVisibility)
 {
  if (terrain.enabled) terrain.enabled = false;
  else { terrain.enabled = true; updateVisibility = false; }
 }

 //Debug.Log(terrain.enabled);
}



public void SetTexture (string name, Texture2D texture)
{
```

```
 if (terrain == null) terrain = GetComponent<Terrain>();
 if (matProps == null) matProps = new MaterialPropertyBlock();

 matProps.SetTexture(name, texture);

 int index = names.Find(name);
 if (index < 0)
 {
  ArrayTools.Add(ref names, name);
  ArrayTools.Add(ref textures, texture);
 }
 else
  textures[index] = texture;
}

public Texture2D GetTexture (string name)
{
 if (name == null) return null;
 if (terrain == null) terrain = GetComponent<Terrain>();
 if (matProps == null) matProps = new MaterialPropertyBlock();

 int index = names.Find(name);
 if (index < 0) return null;
 else return textures[index];
}
```

```
  public void Apply ()
  {
   terrain.SetSplatMaterialPropertyBlock(matProps);
  }
 }

}
```

```
ï»¿using UnityEngine;
using System;
using System.Threading;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using MapMagic.Core;


namespace MapMagic.Terrains
{
 [Serializable]
 public class TerrainSettings
 {
  //terrain settings
  [Val(name="Auto Connect", cat="AutoConnect")] public bool allowAutoConnect = true;
  [Val(name="Grouping ID", cat="AutoConnect")] public int groupingID = 0;

  [Val(name="Base Map Dist.", cat="BaseMap")] public int baseMapDist = 1000;
  [Val(name="Show Base Map", cat="BaseMap")] public bool showBaseMap = true;
  [Val(name="Base Map Resolution", cat="BaseMap")] public int baseMapResolution = 1024;

  [Val(name="Draw Instanced", cat="Terrain")] public bool drawInstanced = true;
  [Val(name="Pixel Error", cat="Terrain")] public int pixelError = 1;
```

```
#if UNITY_2019_1_OR_NEWER
[Val(name="Cast Shadows", cat="Terrain")] public UnityEngine.Rendering.ShadowCastingMode shadowC
#else
[Val(name="Cast Shadows", cat="Terrain")] public bool castShadows = false;
#endif
[Val(name="Reflection Probes", cat="Terrain")] public UnityEngine.Rendering.ReflectionProbeUsage refle

[Val(name="Editor Render Flags", cat="Misc")] public TerrainRenderFlags editorRenderFlags = TerrainRe
[Val(name="Maximim LOD", cat="Misc")] public int heightmapMaximumLOD = 0;

//[Val(name="Physics Thickness", cat="Terrain")] public int physicsThickness = 1;

//materials
//[Val(name="Material Type", cat="Materials")] public Terrain.MaterialType materialType = Terrain.Materia
//[Val(name="Custom Material", cat="Materials")] public bool useCustomTerrainMaterial = true;
[Val(name="Material Template", cat="Materials", type=typeof(Material))] public Material material = null;
//[Val(name="Assign Material Copy", cat="Materials")] public bool copyMaterial = true;

//details and trees
[Val(name="Draw Detail", cat="Grass")]  public bool detailDraw = true;
[Val(name="Detail Distance", cat="Grass")] public float detailDistance = 80;
[Val(name="Detail Density", cat="Grass")] public float detailDensity = 1;

[Val(name="Tree Distance", cat="Trees")] public float treeDistance = 1000;
[Val(name="Billboard Start", cat="Trees")] public float treeBillboardStart = 200;
[Val(name="Fade Length", cat="Trees")]  public float treeFadeLength = 5;
```

```
[Val(name="Max Full LOD Trees", cat="Trees")] public int treeFullLod = 150;
[Val(name="Tree LOD Bias Multiplier", cat="Trees")] public float treeLODBiasMultiplier = 1;
[Val(name="Bake Light Probes For Trees", cat="Trees")] public bool bakeLightProbesForTrees = false;
[Val(name="Remove Light Probe Ringing", cat="Trees")] public bool deringLightProbesForTrees = true;
#if UNITY_2021_3_OR_NEWER
[Val(name="Preserve Tree Prototype Layers", cat="Trees")] public bool preserveTreePrototypeLayers = f
#endif

[Val(name="Wind Speed", cat="WindTint")]  public float windSpeed = 0.5f;
[Val(name="Wind Bending", cat="WindTint")]  public float windSize = 0.5f;  //bending is size and size is be
[Val(name="Wind Size", cat="WindTint")]   public float windBending = 0.5f;
[Val(name="Grass Tint", cat="WindTint")]  public Color grassTint = Color.gray;



//copy
[Val(name="Copy Layers", cat="Copy")] public bool guiCopy = false;
[Val(name="Copy Tags", cat="Copy")] public bool copyLayersTags = true;
[Val(name="Copy Components", cat="Copy")] public bool copyComponents = false;



public void ApplyAll (Terrain terrain)
{
 ApplySettings(terrain);
 ApplyMaterial(terrain);
 CopyLayersTagsComponents(terrain);
}
```

```
public void ApplySettings (Terrain terrain)
{
 //terrain.legacyShininess
 //terrain.legacySpecular
 terrain.allowAutoConnect = allowAutoConnect;
 terrain.groupingID = groupingID;
 terrain.editorRenderFlags = editorRenderFlags;
 terrain.drawInstanced = drawInstanced;
 terrain.heightmapPixelError = pixelError;
 terrain.basemapDistance = showBaseMap ? baseMapDist : int.MaxValue;
 if (terrain.terrainData.baseMapResolution != baseMapResolution)
  terrain.terrainData.baseMapResolution = baseMapResolution; //will generate base map, takes some tim
 #if UNITY_2019_1_OR_NEWER
 terrain.shadowCastingMode = shadowCastingMode;
 #else
 terrain.castShadows = castShadows;
 #endif
 //terrain.collectDetailPatches
 //terrain.patchBoundsMultiplier
 terrain.reflectionProbeUsage = reflectionProbeUsage;
 //terrain.lightmapIndex
 //terrain.realtimeLightmapIndex
 //terrain.lightmapScaleOffset
 //terrain.realtimeLightmapScaleOffset
 //terrain.freeUnusedRenderingResources
```

```
 terrain.heightmapMaximumLOD = heightmapMaximumLOD;


 terrain.drawTreesAndFoliage = detailDraw;
 terrain.detailObjectDistance = detailDistance;
 terrain.detailObjectDensity = detailDensity;
 terrain.treeDistance = treeDistance;
 terrain.treeBillboardDistance = treeBillboardStart;
 terrain.treeCrossFadeLength = treeFadeLength;
 terrain.treeLODBiasMultiplier = treeLODBiasMultiplier;
 terrain.treeMaximumFullLODCount = treeFullLod;
 #if UNITY_EDITOR
 terrain.bakeLightProbesForTrees = bakeLightProbesForTrees;
 terrain.deringLightProbesForTrees = deringLightProbesForTrees;
 #endif
 #if UNITY_2021_3_OR_NEWER
 terrain.preserveTreePrototypeLayers = preserveTreePrototypeLayers;
 #endif


 terrain.terrainData.wavingGrassSpeed = windSpeed;
 terrain.terrainData.wavingGrassAmount = windSize;
 terrain.terrainData.wavingGrassStrength = windBending;
 terrain.terrainData.wavingGrassTint = grassTint;
}


public void ApplyMaterial (Terrain terrain)
/// Sets the terrain custom material state and assigns a copy of material reference if needed. Returns assi
```

```
{
 #if !UNITY_2019_2_OR_NEWER
 terrain.materialType = Terrain.MaterialType.Custom;
 #endif

 terrain.materialTemplate = material;
}

public void CopyLayersTagsComponents (Terrain terrain)
/// Copies assigned layers, tags and components from MapMagic object to terrain
{
 //getting MapMagic game object
 GameObject go = terrain.gameObject;
 GameObject mmGo = go.transform.parent.parent.gameObject;

 //copy layer, tag, scripts from mm to terrains
 if (copyLayersTags)
 {
  go.layer = mmGo.layer;
  go.isStatic = mmGo.isStatic;
  try { go.tag = mmGo.tag; } catch { Debug.LogError("MapMagic: could not copy object tag"); }
 }
 if (copyComponents)
 {
  MonoBehaviour[] components = mmGo.GetComponents<MonoBehaviour>();
  for (int i=0; i<components.Length; i++)
```

```
    {
      if (components[i] is MapMagicObject || components[i] == null) continue; //if MapMagic itself or script not
      if (terrain.gameObject.GetComponent(components[i].GetType()) == null) ReflectionExtensions.CopyCo
    }
   }
  }
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Threading.Tasks;
using UnityEngine;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Tasks;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes;

namespace MapMagic.Terrains
{

 //[System.Serializable]
 public class TerrainTile : MonoBehaviour, ITile, ISerializationCallbackReceiver
 // TODO: when using Voxeland, use an area or a special VoxelandTile with the same interface
 {
  public MapMagicObject mapMagic;  //each tile belongs to only one mm object, it could not be changed or

  public Coord coord = new Coord(int.MaxValue, int.MaxValue);
  public float distance = -1;  //distance in chunks from the center of the deploy rects
  public int Priority => (int)(-distance*100);
```

```
public bool preview = true;


public TerrainData defaultTerrainData;


public Rect WorldRect => new Rect(coord.x*mapMagic.tileSize.x, coord.z*mapMagic.tileSize.z, mapMagi
public Vector2D Min => new Vector2D(coord.x*mapMagic.tileSize.x, coord.z*mapMagic.tileSize.z);
public Vector2D Max => new Vector2D((coord.x+1)*mapMagic.tileSize.x, (coord.z+1)*mapMagic.tileSize.z
public bool ContainsWorldPosition(float x, float z)
{
 Vector2D worldPos = new Vector2D(coord.x*mapMagic.tileSize.x, coord.z*mapMagic.tileSize.z);
 return  x > worldPos.x  &&   x < worldPos.x+mapMagic.tileSize.x  &&
   z > worldPos.z  &&   z < worldPos.z+mapMagic.tileSize.z;
}


public static Action<TerrainTile, TileData> OnBeforeTileStart;
public static Action<TerrainTile, TileData> OnBeforeTilePrepare;
public static Action<TerrainTile, TileData, StopToken> OnBeforeTileGenerate;
public static Action<TerrainTile, TileData, StopToken> OnTileFinalized; //tile event
public static Action<TerrainTile, TileData, StopToken> OnTileApplied;  //TODO: rename to OnTileComple
public static Action<MapMagicObject> OnAllComplete;
public static Action<TerrainTile, bool, bool> OnLodSwitched;
public static Action<TileData> OnPreviewAssigned; //preview tile changed
public static Action<TerrainTile> OnTileMoved;


public static Action<TerrainTile> OnBeforeResetTerrain; //mainly for Lock, to save and return stored data
public static Action<TerrainTile> OnAfterResetTerrain;
```

```
[System.Serializable]
public class DetailLevel
{
 [NonSerialized] public TileData data; //also assigned on before serialize
 public Terrain terrain;
 public EdgesSet edges = new EdgesSet(); //edges are serializable, while data is not

 public bool generateStarted = true; //to avoid starting generate for the second time
 public bool generateReady = false; //used to control progress bar and lod switch, does not affect task pla
 public bool applyReady = false;  //practice shows two bools better than stage enum

 [NonSerialized] public StopToken stop;  //a tag to stop last assigned task
 [NonSerialized] public ThreadManager.Task task;
 [NonSerialized] public CoroutineManager.Task coroutine;
 [NonSerialized] public Stack<CoroutineManager.Task> applyMainCoroutines;
 [NonSerialized] public CoroutineManager.Task applyDraftCoroutine;
 [NonSerialized] public CoroutineManager.Task switchLodCoroutine; //should be cancelled somehow, but

 public DetailLevel (TerrainTile tile, bool isDraft) { data=new TileData(); terrain = tile.CreateTerrain(isDraft)
 public void Remove () { data?.Clear(inSubs:true); if (terrain!=null) GameObject.DestroyImmediate(terrain
}

[NonSerialized] public DetailLevel main;
[NonSerialized] public DetailLevel draft;
```

```
//serializing on onbeforeserialize

public ObjectsPool objectsPool;

public bool guiMain;
public bool guiDraft;



public Terrain GetTerrain (bool isDraft)  =>  isDraft ? draft?.terrain : main?.terrain;
public bool ContainsTerrain (Terrain terrain)  =>  terrain==draft?.terrain  || terrain==main?.terrain;

public Terrain ActiveTerrain
/// Setting null will disable both terrains
{
 get{
  if (main!=null && main.terrain != null  &&  main.terrain.isActiveAndEnabled)
   return main.terrain;
  if (draft!=null && draft.terrain != null  &&  draft.terrain.isActiveAndEnabled)
   return draft.terrain;
  return null;
 }

 set{
  if (main!=null && value==main.terrain)
  {
   if (main.terrain != null && !main.terrain.isActiveAndEnabled)
```

```
  {
   main.terrain.gameObject.SetActive(true);
   //main.terrain.Flush(); //this is required to set neighbors
  }
  if (draft !=null && draft.terrain != null && draft.terrain.isActiveAndEnabled) draft.terrain.gameObject.SetA
 }
 else if (draft!=null && value==draft.terrain)
 {
  if (main!=null && main.terrain != null && main.terrain.isActiveAndEnabled) main.terrain.gameObject.Set
  if (draft.terrain != null && !draft.terrain.isActiveAndEnabled)
  {
   draft.terrain.gameObject.SetActive(true);
   //draft.terrain.Flush();
  }
 }
 else
 {
  if (main?.terrain != null && main.terrain.isActiveAndEnabled)
   main.terrain.gameObject.SetActive(false);

  if (draft?.terrain != null && draft.terrain.isActiveAndEnabled)
   draft.terrain.gameObject.SetActive(false);
 }
}
}
```

```
public void SwitchLod ()
/// Changes detail level based on main and draft avaialability and readyness
/// Doesn't start generate (it's done by Dist), only welds drafts (not mains)
{
 if (this == null) return; //happens after scene switch

 Profiler.BeginSample("Switch Lod");

 bool useMain = main!=null;
 bool useDraft = draft!=null;
 //if both using main
 //if none disabling terrain

 //in editor
 #if UNITY_EDITOR
 if (!MapMagicObject.isPlaying)
 {
  //if both detail levels are used - choosing the one should be displayed
  if (useMain && useDraft)
  {
   //if generating Draft in DraftData - switching to draft
   if (draft.data!=null  &&  mapMagic?.graph!=null  &&  !draft.data.AllOutputsReady(mapMagic.graph, Outp
    useMain = false;

   //if generating Both in MainData - switching to draft too
```

```
  if (main.data!=null  &&  mapMagic?.graph!=null  &&  !main.data.AllOutputsReady(mapMagic.graph, Out
   useMain = false;

  //if dragging graph dragfield - do not switch from draft back to main
  if (mapMagic.guiDraggingField  &&  ActiveTerrain == draft.terrain)
   useMain = false;
 }
}
else
#endif

//if playmode
{
 //default case with drafts
 if (mapMagic.draftsInPlaymode)
 {
  if ((int)distance > mapMagic.mainRange)  useMain = false;
  if ((int)distance > mapMagic.tiles.generateRange  &&  mapMagic.hideFarTerrains)  useDraft = false;
 }

 //case no drafts at all
 else
 {
  if ((int)distance > mapMagic.tiles.generateRange  &&  mapMagic.hideFarTerrains)  useMain = false;
  useDraft = false;
 }
```

```
 //hiding just moved terrains
 if (main!=null  &&  !main.applyReady) useMain = false;
 if (draft!=null  &&  !draft.applyReady) useDraft = false;

 //if main is not ready and using drafts
 if (useMain  &&  useDraft  &&  !main.applyReady) useMain = false;
}

//debugging
//string was = ActiveTerrain==main.terrain ? "main" : (ActiveTerrain==draft.terrain ? "draft" : "null");
//string replaced = useMain ? "main" : (useDraft ? "draft" : "null");
//Debug.Log("Switching lod. Was " + was + ", replaced with " + replaced);
//if (was == "draft" && replaced == "main")
// Debug.Log("Test");

//finding if lod switch is for real and switching active terrain
Terrain newActiveTerrain;
if (useMain) newActiveTerrain = main.terrain;
else if (useDraft) newActiveTerrain = draft.terrain;
else newActiveTerrain = null;

bool lodSwitched = false;
if (ActiveTerrain != newActiveTerrain)
{
 lodSwitched = true;
```

```
 ActiveTerrain = newActiveTerrain;
}

//disabling objects
bool objsEnabled = useMain; // || (useDraft && mapMagic.draftsIfObjectsChanged);
bool currentObjsEnabled = objectsPool.isActiveAndEnabled;
if (!objsEnabled && currentObjsEnabled) objectsPool.gameObject.SetActive(false);
if (objsEnabled && !currentObjsEnabled) objectsPool.gameObject.SetActive(true);



//welding
//TODO: check active terrain to know if the switch is for real
if (lodSwitched &&
 mapMagic.tiles.Contains(coord) ) //otherwise error on SwitchLod called from Generate (when tile has be
{
 if (useMain)
 {
  Weld.WeldSurroundingDraftsToThisMain(mapMagic.tiles, coord);
  Weld.WeldCorners(mapMagic.tiles, coord);

  //Weld.SetNeighbors(mapMagic.tiles, coord);
  //Unity calls Terrain.SetConnectivityDirty on each terrain enable or disable that resets neighbors
  //using autoConnect instead. AutoConnect is a crap but neighbors are broken
 }
 else if (useDraft  &&  draft.applyReady)
  Weld.WeldThisDraftWithSurroundings(mapMagic.tiles, coord);
```

```
 }

 if (lodSwitched) OnLodSwitched?.Invoke(this, useMain, useDraft);

 //CoroutineManager.Enqueue( ()=>Weld.SetNeighbors(mapMagic.tiles, coord) );
 //CoroutineManager.Enqueue( mapMagic.Tmp );
 //mapMagic.Tmp();

 Profiler.EndSample();
}


public void ResetTerrain ()
/// Removes terrain and children, re-constructing tile. Used to clear some output
{
 OnBeforeResetTerrain?.Invoke(this);

 bool hasMain = main!=null;
 bool hasDraft = draft!=null;

 //removing all children
 for (int i=transform.childCount-1; i>-0; i--)
  GameObject.DestroyImmediate(transform.GetChild(i).gameObject);

 //creating new
 if (hasMain) main = new DetailLevel(this, isDraft:false);
```

```
 if (hasDraft) draft = new DetailLevel(this, isDraft:true);
 CreateObjectsPool();

 OnAfterResetTerrain?.Invoke(this);
}

#region ITile

 public static TerrainTile Construct (MapMagicObject mapMagic)
 {
  Profiler.BeginSample("Construct Internal");

  GameObject go = new GameObject();
  go.transform.parent = mapMagic.transform;
  TerrainTile tile = go.AddComponent<TerrainTile>();
  tile.mapMagic = mapMagic;

  //tile.Resize(mapMagic.tileSize, (int)mapMagic.tileResolution, mapMagic.tileMargins, (int)mapMagic.lodF

  //creating detail levels in playmode (for editor Pin us used)
  if (MapMagicObject.isPlaying) //if (UnityEditor.EditorApplication.isPlayingOrWillChangePlaymode)
  {
   tile.main = new DetailLevel(tile, isDraft:false); //tile created in any case and generated at the backgroun

   if (mapMagic.draftsInPlaymode)
```

```
    tile.draft = new DetailLevel(tile, isDraft:true);
  }

  //creating objects pool
  tile.CreateObjectsPool();

  Profiler.EndSample();

  return tile;
}



public void Pin (bool asDraftOnly)
{
  if (mapMagic.draftsInEditor && draft==null)
    draft = new DetailLevel(this, isDraft:true);

  if (!asDraftOnly && main==null)
    main = new DetailLevel(this, isDraft:false);

  if (asDraftOnly && main!=null)
    { main.Remove(); main=null; }
}



public void Move (Coord newCoord, float newRemoteness)
```

```
{
 coord = newCoord;

 //if (IsGenerating) //stopping anyway just in case
  StopGenerate();

 //clearing
 main?.data?.Clear(inSubs:true);
 draft?.data?.Clear(inSubs:true);

 if (main!=null) { main.applyReady = false; main.generateReady = false;  main.generateStarted = false; }
 if (draft != null) { draft.applyReady = false; draft.generateReady = false; draft.generateStarted = false; }

 ActiveTerrain = null; //disabling terrains

 //resizing (if needed)
 Vector3 size = (Vector3)mapMagic.tileSize;
 Vector3 position = new Vector3(coord.x*size.x, 0, coord.z*size.z);

 if (main!=null  &&  main.terrain != null  &&  main.terrain.terrainData.size != new Vector3 (size.x, main.terr
  main.terrain.terrainData.size = new Vector3(size.x, main.terrain.terrainData.size.y, size.z);

 if (draft!=null && draft.terrain != null  &&  draft.terrain.terrainData.size != new Vector3 (size.x, draft.terrain
  draft.terrain.terrainData.size = new Vector3(size.x, draft.terrain.terrainData.size.y, size.z);

 //moving
```

```
 transform.localPosition = position;
 gameObject.name = "Tile " + coord.x + "," + coord.z;

 //switch Dist (on each move)
 Dist(newRemoteness);

 OnTileMoved?.Invoke(this);
}

public void Dist (float newRemoteness)
{
 distance = newRemoteness;

 if (MapMagicObject.isPlaying)
 {
  if (main != null  &&
   !main.generateStarted  &&
   (int)distance <= mapMagic.mainRange)
     StartGenerate(mapMagic.graph, generateMain:true, generateLod:false);

  if (draft != null  &&
   !draft.generateStarted  &&
   (int)distance <= mapMagic.tiles.generateRange)
     StartGenerate(mapMagic.graph, generateMain:false, generateLod:true);
```

```
  //switching lod in playmode
  if (coord != new Coord(int.MaxValue, int.MaxValue))  //skipping tiles that were just created to avoid show
   SwitchLod();
 }

 else //editor mode
 {
  if (draft != null  &&  !draft.generateStarted) StartGenerate(mapMagic.graph, generateMain:false, genera
  if (main != null  &&  !main.generateStarted) StartGenerate(mapMagic.graph, generateMain:true, genera
 }

 //TODO: switch tasks priorities
}



public void Remove ()
{
 StopGenerate();

 #if UNITY_EDITOR
 if (!UnityEditor.EditorApplication.isPlaying)
  GameObject.DestroyImmediate(gameObject);
 else
 #endif
  GameObject.Destroy(gameObject);
}
```

```
public bool IsNull {get{ return this==(UnityEngine.Object)null || this.Equals(null) || gameObject==null || ga

//public bool Equals(TerrainTile tile) { return (object)this == (object)tile; }



public void Resize ()
{
 Move(coord, distance);
 //yep, it will change the tile size, including the height
}



public Terrain CreateTerrain (bool isDraft)
{
 GameObject go = new GameObject();
 go.transform.parent = transform;
 go.transform.localPosition = new Vector3(0,0,0);
 go.name = isDraft ? "Draft Terrain" : "Main Terrain";

 Terrain terrain = go.AddComponent<Terrain>();
 TerrainCollider terrainCollider = go.AddComponent<TerrainCollider>();

 TerrainData terrainData;
 TerrainData template = Resources.Load<TerrainData>("MapMagicDefaultTerrainData");
```

```
 if (template != null)
  terrainData = GameObject.Instantiate(template);
 else
  terrainData = new TerrainData();

 terrain.terrainData = terrainData;
 terrainCollider.terrainData = terrainData;
 terrainData.size = (Vector3)mapMagic.tileSize;

 mapMagic.terrainSettings.ApplyAll(terrain);
 terrain.groupingID = isDraft ? -2 : -1;

 return terrain;
}

public void CreateObjectsPool ()
{
 GameObject poolGo = new GameObject();
 poolGo.transform.parent = transform;
 poolGo.transform.localPosition = new Vector3();
 poolGo.name = "Objects";
 objectsPool = poolGo.AddComponent<ObjectsPool>();
}

#endregion
```

```
#region Async/Task

 /*private Task draftTask;
 private Task mainTask;

 private bool reGenDraft;


 public async Task GenerateAsync (Graph graph, bool genMain, bool genDraft)
 {
  if (draft != null  &&  genDraft) draftTask = GenerateDraftAsync(graph);
  if (main != null  &&  genMain) mainTask = GenerateMainAsync(graph);

  if (draft != null  &&  genDraft) await draftTask;
  if (main != null  &&  genMain) await mainTask;

  SwitchLod();
 }


 public async Task GenerateDraftAsync (Graph graph)
 {
  if (draftTask != null && !draftTask.IsCompleted)
   { reGenDraft = true; return; }
```

```
 draftTask = GenerateDraftAsyncInternal(graph);
 await draftTask;


 if (reGenDraft)
 {
  reGenDraft = false;
  draftTask = GenerateDraftAsyncInternal(graph);
  await draftTask;
 }
}

public async Task GenerateDraftAsyncInternal (Graph graph)
{
 //cancel the task that's already running
 if (draftTask != null && !draftTask.IsCompleted)
 {
  //draft.Data.stop = true; //don't stop draft, make it refresh constantly


  //but make it don't wait if it wasn't started


  //await draftTask;
 }


 draft.data.area = new Area(coord, (int)mapMagic.draftResolution, mapMagic.draftMargins, mapMagic.til
 draft.data.parentGraph = graph;
 draft.data.random = graph.random;
```

```
draft.data.isPreview = false; //don't preview draft in any case
draft.data.isDraft = false;
//draft.Data.stop = false;

//draft.Data.parentGraph.CheckClear(draft.Data);
await Task.Run( ()=> draft.data.parentGraph.CheckClear(draft.data) );

draft.data.parentGraph.Prepare(draft.data, main.terrain);

await Task.Run (() =>
{
 draft.data.parentGraph.Generate(draft.data);
 draft.data.parentGraph.Finalize(draft.data);
});

//draft.Data.parentGraph.Generate(draft.Data);
//draft.Data.parentGraph.Finalize(draft.Data);

//if (draft.Data.stop) return;

if (draft.terrain == null) draft.terrain = CreateTerrain("Draft Terrain");

while (draft.data.ApplyCount != 0)
{
 ITerrainData apply = draft.data.DequeueApply(); //this will remove apply from the list
 apply.Apply(draft.terrain);
```

```
 }
}



public async Task GenerateMainAsync (Graph graph)
{
 //cancel the task that's already running
 if (mainTask != null && !mainTask.IsCompleted)
 {
  //draft.Data.stop = true; //don't stop draft, make it refresh constantly
  await mainTask;
 }

 main.data.area = new Area(coord, (int)mapMagic.tileResolution, mapMagic.tileMargins, mapMagic.tileSi
 main.data.parentGraph = graph;
 main.data.random = graph.random;
 main.data.isPreview = preview;
 main.data.isDraft = false;
 //main.Data.stop = false;

 //clear changed nodes for main data first to see if draft should be switched
 await Task.Run( ()=> main.data.parentGraph.CheckClear(main.data) );

 //prepare
 main.data.parentGraph.Prepare(main.data, main.terrain);
```

```
 //generate
 await Task.Run ( ()=>
 {
  main.data.parentGraph.Generate(main.data);
  main.data.parentGraph.Finalize(main.data);

  //saving last generated results to use as preview
  //if (main.data.isPreview)
  // main.data.parentGraph.lastGeneratedResults.Target = main.data.products; //TODO: to MapMagic?

  //merging locks (by event?)
  //for (int l=0; l<main.data.lockReads.Count; l++)
  // main.data.lockReads[l].MergeLocks(main.data.terrainApply);
 });

 //if (main.Data.stop) return;

 if (main.terrain == null) main.terrain = CreateTerrain("Main Terrain");

 while (main.data.ApplyCount != 0)
 {
  ITerrainData apply = main.data.DequeueApply(); //this will remove apply from the list
  apply.Apply(main.terrain);
 }
}*/
```

```
#endregion



#region Threaded


 public void StartGenerate (Graph graph, bool generateMain=true, bool generateLod=true)
 /// Starts generating tile in a separate thread (or just enqueues it if `launch` is set to false)
 {
  if (graph==null) return;


  //starting draft
  if (generateLod  &&  draft != null)
  {
   if (draft.data == null) draft.data = new TileData();
   draft.data.area = new Area(coord, (int)mapMagic.draftResolution, mapMagic.draftMargins, mapMagic.ti
   draft.data.globals = mapMagic.globals;
   draft.data.random = graph.random;
   draft.data.isPreview = false; //don't preview draft in any case
   draft.data.isDraft = true;


   //if (draft.coroutines == null) draft.coroutines = new Stack<CoroutineManager.Task>();
   //while (draft.coroutines.Count != 0)
   // CoroutineManager.Stop(draft.coroutines.Pop());


   draft.generateStarted = true;
   draft.applyReady = false;
```

```
  draft.generateReady = false;


  EnqueueTask(draft, graph, Priority+1000, "Draft");
 }

 //starting main
 if (generateMain  &&  main != null)
 {
  if (main.data == null) main.data = new TileData();
  main.data.area = new Area(coord, (int)mapMagic.tileResolution, mapMagic.tileMargins, mapMagic.tileS
  main.data.globals = mapMagic.globals;
  main.data.random = graph.random;
  main.data.isPreview = mapMagic.PreviewData==main.data;
  main.data.isDraft = false;

  main.generateStarted = true;
  main.applyReady = false;
  main.generateReady = false;

  StopEnqueueTask(main, graph, Priority, "Main");
  //EnqueueTask(main, graph, Priority, "Main");
 }

 SwitchLod(); //switching to draft if needed
}
```

```
private void EnqueueTask (DetailLevel det, Graph graph, int priority=0, string name="Task")
/// Will run task no matter if previous task is running (draft-style)
{
 if (det.task == null)
 {
  det.stop = new StopToken();
  det.task = new ThreadManager.Task() {
   action = ()=>Generate(graph, this, det, det.stop), //graph captured, stop isn't
   priority = priority,
   name = name + " " + coord };
 }

 det.task.priority = priority;

 if (det.task.Active) det.stop.restart = true;
 else
 {
  if (!det.task.Enqueued)
  {
   Prepare(graph, this, det);
   ThreadManager.Enqueue(det.task);
  }
 }
}
```

```
private void StopEnqueueTask (DetailLevel det, Graph graph, int priority=0, string name="Task")
/// Will stop previous task before running
{
 if (det.applyMainCoroutines == null) det.applyMainCoroutines = new Stack<CoroutineManager.Task>();
 while (det.applyMainCoroutines.Count != 0)
  CoroutineManager.Stop(det.applyMainCoroutines.Pop());

 if (det.switchLodCoroutine != null)
  CoroutineManager.Stop(det.switchLodCoroutine);

 if (det.coroutine != null)
  CoroutineManager.Stop(det.coroutine);

 if (det.task != null  &&  det.task.Active)
 {
  det.stop.stop = true;
  //and forget about this task
 }

 if (det.task == null  ||  !det.task.Enqueued)
 {
  Prepare(graph, this, main);

  det.stop = new StopToken();
  StopToken stop = det.stop; //closure var
  det.task = new ThreadManager.Task() {
```

```
  action = ()=>Generate(graph, this, det, stop),
  priority = priority,
  name = name + " " + coord };
 ThreadManager.Enqueue(det.task);
}
//do nothing if task enqueued

det.task.priority = priority;



//Alternative:
/*if (det.task != null)
{
 ThreadManager.Dequeue(det.task); //if enqueued
 det.stop.stop = true;     //if active
 //and forget about this task
}

Prepare(graph, this, main);

det.stop = new StopToken();
StopToken mainStop = det.stop; //closure var
det.task = new ThreadManager.Task() {
 action = ()=>Generate(graph, this, main, mainStop),
 priority = Priority,
 name = "Main " + coord };
```

```
    ThreadManager.Enqueue(det.task);*/
   }

   private void Prepare (Graph graph, TerrainTile tile, DetailLevel det)
   {
    det.edges.ready = false;

    OnBeforeTilePrepare?.Invoke(tile, det.data);

    graph.Prepare(det.data, det.terrain);
    //was using data's parent graph
   }


   private void Generate (Graph graph, TerrainTile tile, DetailLevel det, StopToken stop)
   /// Note that referencing det.task is illegal since task could be changed
   {
    OnBeforeTileGenerate?.Invoke(tile, det.data, stop);

    //do not return (for draft) until the end (apply)
//    if (!stop.stop) graph.CheckClear(det.data, stop);
    if (!stop.stop) graph.Generate(det.data, stop);
    if (!stop.stop) graph.Finalize(det.data, stop);

    //finalize event
    OnTileFinalized?.Invoke(tile, det.data, stop);
```

```
//flushing products for playmode (all except apply)
if (MapMagicObject.isPlaying)
 det.data.Clear(clearApply:false, inSubs:true);

//welding (before apply since apply will flush 2d array)
if (!stop.stop) Weld.ReadEdges(det.data, det.edges);
if (!stop.stop) Weld.WeldEdgesInThread(det.edges, tile.mapMagic.tiles, tile.coord, det.data.isDraft);
if (!stop.stop) Weld.WriteEdges(det.data, det.edges);

//enqueue apply
//was: while the playmode is applied on SwitchLod to avoid unnecessary lags for main

if (det.data.isDraft)
 det.coroutine = CoroutineManager.Enqueue(()=>ApplyNow(det,stop), Priority+1000, "ApplyNow " + coo

else //main
{
 IEnumerator coroutine = ApplyRoutine(det, stop);
 det.coroutine = CoroutineManager.Enqueue(coroutine, Priority, "ApplyRoutine " + coord);
}

det.generateReady = true;
}
```

```
private void ApplyNow (DetailLevel det, StopToken stop)
{
 if (this == null) return;

 if (stop==null || !stop.stop)
 {
  while (det.data.ApplyMarksCount != 0)
  {
   var appDat = det.data.DequeueApply();
   appDat.Apply(det.terrain);
  }

  //MapMagicObject.OnTileApplied?.Invoke(this, det.data, stop);

  det.applyReady = true; //enabling ready before switching lod (otherwise will leave draft)

  SwitchLod();

  OnTileApplied?.Invoke(this, det.data, stop);

  //if (!mapMagic.IsGenerating()) //won't be called since this couroutine still left
  if (!ThreadManager.IsWorking && CoroutineManager.IsQueueEmpty)
   OnAllComplete?.Invoke(mapMagic);
 }

 if (stop.restart)
```

```
 {
  stop.restart=false;
  //Prepare(graph, this, det);
  if (!det.task.Enqueued) ThreadManager.Enqueue(det.task);
 }
}



private IEnumerator ApplyRoutine (DetailLevel det, StopToken stop)
{
 if (this == null) yield break;

 if (stop==null || !stop.stop)
 {
  while (det.data.ApplyMarksCount != 0)
  {
   if (stop!=null && stop.stop) yield break;

   IApplyData apply = det.data.DequeueApply(); //this will remove apply from the list
            //coroutines guarantee FIFO
   if (apply is IApplyDataRoutine)
   {
    IEnumerator routine = (apply as IApplyDataRoutine).ApplyRoutine(det.terrain);
    while (true)
    {
     if (stop!=null && stop.stop) yield break;
```

```
    bool move = routine.MoveNext();
    yield return null;

    if (!move) break;
   }
  }
  else
  {
   apply.Apply(det.terrain);
   yield return null;
  }
 }
}

if (stop==null || !(stop.stop || stop.restart)) //can't set ready when restart enqueued
{
 det.applyReady = true; //enabling ready before switching lod (otherwise will leave draft)

 SwitchLod();

 OnTileApplied?.Invoke(this, det.data, stop);

 //if (!mapMagic.IsGenerating()) //won't be called since this couroutine still left
 if (!ThreadManager.IsWorking && CoroutineManager.IsQueueEmpty)
  OnAllComplete?.Invoke(mapMagic);
```

```
 }

 if (stop!=null && stop.restart)
 {
  stop.restart=false;
  //Prepare(graph, this, det);
  if (!det.task.Enqueued) ThreadManager.Enqueue(det.task);
 }
}

public void StopGenerate ()
{
 if (main != null) StopGenerate(main);
 if (draft != null) StopGenerate(draft);
}

private void StopGenerate (DetailLevel det)
{
 if (det.task != null)
 {
  det.stop.stop = true;
  det.stop.restart = false;
  ThreadManager.Dequeue(det.task);
 }
```

```
 if (det.applyMainCoroutines != null)
  foreach (CoroutineManager.Task coroutine in main.applyMainCoroutines)
    CoroutineManager.Stop(coroutine);

 det.task = null;
 if (det.stop != null) det.stop.stop = true; //det.stop = null;
}



public (float progress, float max) GetProgress (Graph graph, float generateComplexity, float applyComple
{
 float progress = 0;
 float max = 0;

 if (main != null  &&  main.generateStarted)
 {
  max += generateComplexity + applyComplexity;

  if (main.generateReady) progress += generateComplexity;
  else if (main.data != null)  progress += graph.GetGenerateProgress(main.data);

  if (main.applyReady) progress += applyComplexity;
  else if (main.data != null) progress += graph.GetApplyProgress(main.data);
 }
```

```
 if (draft != null  &&  draft.generateStarted)
 {
  max += 2;
  if (draft.generateReady) progress ++;
  if (draft.applyReady) progress ++;
 }

 return (progress, max);
}



public bool IsGenerating
{get{
 if (main != null  &&  main.generateStarted  &&  !main.applyReady) return true;
 if (draft != null  &&  draft.generateStarted  &&  !draft.applyReady) return true;
 return false;
}}

public bool Ready
{get{
 if (main != null  &&  (!main.applyReady || !main.generateReady)) return false;
 if (draft != null  &&  (!draft.applyReady || !draft.generateReady)) return false;
 return true;
}}

//public bool ReadyDraft
```

```
 // {get{ return draft!=null && draft.stage != DetailLevel.Stage.Blank && draft.stage != DetailLevel.Stage.Re

#endregion



#region Serialization

 [SerializeField] private DetailLevel serialized_main;
 [SerializeField] private bool serialized_mainNull;

 [SerializeField] private DetailLevel serialized_draft;
 [SerializeField] private bool serialized_draftNull;

 public void OnBeforeSerialize ()
 {
  serialized_main = main;
  serialized_mainNull = main==null;

  serialized_draft = draft;
  serialized_draftNull = draft==null;
 }



 public void OnAfterDeserialize ()
 {
  if (!serialized_mainNull)
```

```
  {
   main = serialized_main;
   //main.data = new TileData(); //data is not serialized, so it will be null

   if (!main.applyReady || !main.generateReady) //resetting ready state if it's not completely generated
    { main.applyReady = false; main.generateReady = false; }
  }

  if (!serialized_draftNull)
  {
   draft = serialized_draft;
   //draft.data = new TileData();

   if (!draft.applyReady || !draft.generateReady) //resetting ready state if it's not completely generated
    { draft.applyReady = false; draft.generateReady = false; }
  }
 }

#endregion



public void OnDrawGizmos_Tmp ()
{
 Gizmos.color = Color.blue;
 Vector3 center = (Vector3)(coord.vector2d * mapMagic.tileSize.x + mapMagic.tileSize/2);
 Gizmos.DrawWireCube(center, (Vector3)mapMagic.tileSize);
```

```
center.y += 150;

//active terrain
Gizmos.color = Color.red;
if (draft != null && ActiveTerrain == draft.terrain) Gizmos.color = Color.yellow;
if (main != null && ActiveTerrain == main.terrain) Gizmos.color = Color.green;
Gizmos.DrawCube(center + new Vector3(-150,0,0), new Vector3(60,60,60));

//main state
Gizmos.color = Color.black;
if (main != null)
{
 Gizmos.color = Color.green;
 if (!main.applyReady)
 {
  if (main.task.Enqueued) Gizmos.color = Color.red;
  if (main.task.Active) Gizmos.color = new Color(0.8f, 0.3f, 0, 1);

  if (main.applyMainCoroutines != null)
   foreach (CoroutineManager.Task coroutine in main.applyMainCoroutines)
    if (coroutine.Active || coroutine.Enqueued) Gizmos.color = Color.yellow;
 }
}
Gizmos.DrawSphere(center + new Vector3(-30,0,0), 60);
```

```
//draft state
Gizmos.color = Color.black;
if (draft != null)
{
 Gizmos.color = Color.green;
 if (!draft.applyReady)
 {
  if (draft.task.Enqueued) Gizmos.color = Color.red;
  if (draft.task.Active) Gizmos.color = new Color(0.8f, 0.3f, 0, 1);

  if (draft.applyMainCoroutines != null)
   foreach (CoroutineManager.Task coroutine in draft.applyMainCoroutines)
    if (coroutine.Active || coroutine.Enqueued) Gizmos.color = Color.yellow;
 }
}
Gizmos.DrawSphere(center + new Vector3(90,0,0), 40);

//lod switch enqueued
if (CoroutineManager.IsNameEnqueued("LodSwitch " + coord)) Gizmos.color = Color.red;
else if (CoroutineManager.IsNameActive("LodSwitch " + coord)) Gizmos.color = Color.yellow;
else Gizmos.color = Color.green;
Gizmos.DrawSphere(center + new Vector3(180,0,0), 30);

//data size
/*Gizmos.color = Color.gray;
int dataSize = 0;
```

```
    if (main!=null) dataSize += main.data.Count();
    if (draft!=null) dataSize += draft.data.Count();
    dataSize *= 10;
    Gizmos.DrawCube(center + new Vector3(0,0,-120), new Vector3(dataSize,30,30));*/
   }
  }

}
```

```
ï»¿using UnityEngine;
using System;
using System.Threading;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;

using MapMagic.Nodes;
using MapMagic.Terrains;

namespace MapMagic.Core
{
 [Serializable]
 public class TerrainTileManager : TileManager<TerrainTile>, ISerializationCallbackReceiver
 {
  [SerializeField] public TerrainTile[] customTiles = new TerrainTile[0];
  public Dictionary<Coord,TerrainTile> pinned = new Dictionary<Coord,TerrainTile>();


  public void Pin (Coord coord, bool asDraft, MonoBehaviour holder=null)
  /// Creates new tile at the coord if it's empty and pin it
  {
   grid.TryGetValue(coord, out TerrainTile tile);

   if (tile == null)
```

```
 {
  tile = ConstructTile(holder);
  grid.Add(coord, tile);
 }

 else
  tile.Pin(asDraft);

 tile.Pin(asDraft);
 tile.Move(coord, camCoords != null ? GetRemoteness(coord,camCoords) : 0);

 if (!pinned.ContainsKey(coord))
  pinned.Add(coord, tile);
}


public void Unpin (Coord coord)
/// Clears pin flag for tile at the coord and re-deploys grid to remove it if needed
{
 if (!pinned.ContainsKey(coord)) return;

 pinned.Remove(coord);

 //re-deploying to find out if this tile should be removed or left as unpinned
 //if (camCoords != null)
 // Deploy(camCoords, pinned, holder:null); //deploying without holder since it shouldn't create new tiles ar
```

```
 //no deploy was performed - removing pinned
 //else
 {
  grid[coord].Remove();
  grid.Remove(coord);
 }
}


public void Deploy (Coord[] camCoords, MonoBehaviour holder=null)
 { Deploy(camCoords, pinned, holder); }

public IEnumerable<TerrainTile> All ()
{
 foreach (TerrainTile tile in base.Tiles())
  yield return tile;

 for (int i=0; i<customTiles.Length; i++)
  yield return customTiles[i];
}

/*public TerrainTile PreviewTile
{
 get
 {
```

```
  foreach (TerrainTile tile in All())
   if (tile.preview) return tile;
  return null;
 }
 set
 {
  foreach (TerrainTile tile in All())
  {
   if (tile == value) tile.preview = true;
   else tile.preview = false;
  }
 }
}*/

public IEnumerable<Rect> AllWorldRects ()
/// Map-Magic relative rects actually (in MM coordsys)
{
 foreach (TerrainTile tile in All())
  yield return tile.WorldRect;
}

public IEnumerable<Terrain> AllActiveTerrains ()
{
 foreach (TerrainTile tile in All())
  yield return tile.ActiveTerrain;
}
```

```
public void PinCustom (TerrainTile tile)
{
 if (!customTiles.Contains(tile))
  ArrayTools.Add(ref customTiles, tile);
}

public void UnpinCustom (TerrainTile tile)
{
 if (customTiles.Contains(tile))
  ArrayTools.Remove(ref customTiles, tile);
}



public override void RemoveNulls ()
{
 base.RemoveNulls();

 for (int i=customTiles.Length-1; i>=0; i--)
 {
  if (customTiles[i]==null || customTiles[i].IsNull)
   ArrayTools.RemoveAt(ref customTiles, i);
 }
}

public override TerrainTile Closest ()
```

```
/// Using cached distances instead of re-calculating hem
/// If mainOnly enabled then checking only tiles containing main data
{
 float minDist = int.MaxValue;
 TerrainTile minTile = default;

 foreach (var kvp in grid)
 {
  TerrainTile tile = kvp.Value;
  if (tile.distance < minDist) { minDist=tile.distance; minTile=kvp.Value; }
 }

 return minTile;
}

public TerrainTile ClosestMain ()
/// Same as above, but iterates only in tiles with main data
{
 float minDist = int.MaxValue;
 TerrainTile minTile = default;

 foreach (var kvp in grid)
 {
  TerrainTile tile = kvp.Value;
  if (tile.main==null) continue;
  if (tile.distance < minDist) { minDist=tile.distance; minTile=kvp.Value; }
```

```
 }

 return minTile;
}

public TerrainTile FindByWorldPosition (float x, float z)
{
 foreach (TerrainTile tile in All())
 {
  if (tile.ContainsWorldPosition(x,z))
   return tile;
 }

 return null;
}

public TerrainTile FindByTerrain (Terrain terrain)
{
 foreach (TerrainTile tile in All())
 {
  if (tile.main?.terrain == terrain)
   return tile;
  if (tile.draft?.terrain == terrain)
   return tile;
 }
```

```
 return null;
}



#region Serialization

public Coord[] serializedPinnedCoords = new Coord[0];

 public override void OnBeforeSerialize ()
 {
  base.OnBeforeSerialize();

  if (serializedPinnedCoords.Length != pinned.Count)
   serializedPinnedCoords = new Coord[pinned.Count];

  int i=0;
  foreach (var kvp in pinned)
  {
   serializedPinnedCoords[i] = kvp.Key;
   i++;
  }
 }

 public override void OnAfterDeserialize ()
 {
  base.OnAfterDeserialize();
```

```
    for (int i=0; i<serializedPinnedCoords.Length; i++)
    {
     Coord coord = serializedPinnedCoords[i];
     if (grid.TryGetValue(coord, out TerrainTile tile))
     {
      if (!pinned.ContainsKey(coord))
        pinned.Add(coord, tile);
     }
    }
   }

  #endregion
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Threading.Tasks;
using UnityEngine;

using Den.Tools;
using MapMagic.Core;
using MapMagic.Products;
using MapMagic.Nodes;
using Den.Tools.Matrices;
using MapMagic.Nodes.MatrixGenerators;

namespace MapMagic.Terrains
{
 [System.Serializable]
 public class EdgesSet
 {
  public bool ready = false;

  public Edges heightEdges = new Edges(0,0);
  public Edges[] splatEdges;
  public Edges[] controlEdges;

  public Lowered lowered;
```

```
}

[System.Serializable]
public class Edges
{
 public float[] arr_X;
 public float[] arr_Z;
 public float[] arr_x;
 public float[] arr_z;

 public Edges (int sizeX, int sizeZ)
 {
  arr_x = new float[sizeX]; arr_X = new float[sizeX];
  arr_z = new float[sizeZ]; arr_Z = new float[sizeZ];
 }

 public bool IsEmpty => arr_x.Length==0 && arr_X.Length==0 && arr_z.Length==0 && arr_Z.Length==0;

 public int SizeX {get{ return arr_x.Length; }}
 public int SizeZ {get{ return arr_z.Length; }}

 public float[] GetArr (Coord dir)
 {
  if (dir.x==0  &&  dir.z==-1) return arr_x;
  else if (dir.x==0  &&  dir.z==1) return arr_X;
  else if (dir.x==-1  &&  dir.z==0) return arr_z;
```

```
 else if (dir.x==1  &&  dir.z==0) return arr_Z;
 else return null;
}

#region Read/Write

 public void ReadFloats2D (float[,] heights2D)
 {
  int sizeX = heights2D.GetLength(1);
  int sizeZ = heights2D.GetLength(0);

  if (arr_x.Length != sizeX) arr_x = new float[sizeX];
  if (arr_X.Length != sizeX) arr_X = new float[sizeX];
  if (arr_z.Length != sizeZ) arr_z = new float[sizeZ];
  if (arr_Z.Length != sizeZ) arr_Z = new float[sizeZ];

  for (int x=0; x<sizeX; x++)
  {
   arr_x[x] = heights2D[0, x];
   arr_X[x] = heights2D[sizeZ-1, x];
  }

  for (int z=0; z<sizeZ; z++)
  {
   arr_z[z] = heights2D[z, 0];
   arr_Z[z] = heights2D[z, sizeX-1];
```

```
 }
}


public void WriteFloats2D (float[,] heights2D)
{
 int sizeX = heights2D.GetLength(1);
 int sizeZ = heights2D.GetLength(0);

 for (int x=0; x<sizeX; x++)
 {
  heights2D[0, x] = arr_x[x];
  heights2D[sizeZ-1, x] = arr_X[x];
 }

 for (int z=0; z<sizeZ; z++)
 {
  heights2D[z, 0] = arr_z[z];
  heights2D[z, sizeX-1] = arr_Z[z];
 }
}


public void ReadSplitFloats2D (float[][,] heights2DSplits)
{
 int sizeX = 0; for (int i=0; i<heights2DSplits.Length; i++) sizeX += heights2DSplits[i].GetLength(0);
```

```
int sizeZ = heights2DSplits[0].GetLength(1);

if (arr_x.Length != sizeX) arr_x = new float[sizeX];
if (arr_X.Length != sizeX) arr_X = new float[sizeX];
if (arr_z.Length != sizeZ) arr_z = new float[sizeZ];
if (arr_Z.Length != sizeZ) arr_Z = new float[sizeZ];

float[,] last = heights2DSplits[heights2DSplits.Length-1];
int lastWidth = last.GetLength(0);

for (int x=0; x<sizeX; x++)
{
 arr_x[x] = heights2DSplits[0][0, x];
 arr_X[x] = last[lastWidth-1, x];
}

int offset = 0;
for (int i=0; i<heights2DSplits.Length; i++)
{
 int segLength = heights2DSplits[i].GetLength(0);
 for (int z=0; z<segLength; z++)
 {
  arr_z[offset] = heights2DSplits[i][z, 0];
  arr_Z[offset] = heights2DSplits[i][z, sizeZ-1];

  offset++;
```

```
  }
 }
}


public void WriteSplitFloats2D (float[][,] heights2D)
{
 int sizeX = arr_x.Length;
 int sizeZ = arr_z.Length;

 float[,] last = heights2D[heights2D.Length-1];
 int lastWidth = last.GetLength(0);
 for (int x=0; x<sizeX; x++)
 {
  heights2D[0][0, x] = arr_x[x];
  last[lastWidth-1, x] = arr_X[x];
 }

 int offset = 0;
 for (int i=0; i<heights2D.Length; i++)
 {
  int segLength = heights2D[i].GetLength(0);
  for (int z=0; z<segLength; z++)
  {
   heights2D[i][z, 0] = arr_z[offset];
   heights2D[i][z, sizeZ-1] = arr_Z[offset];
```

```
   offset++;
  }
 }
}


public void ReadRawFloat (byte[] bytes)
{
 int res = (int)Mathf.Sqrt(bytes.Length/4);

 if (arr_x.Length != res) arr_x = new float[res];
 if (arr_X.Length != res) arr_X = new float[res];
 if (arr_z.Length != res) arr_z = new float[res];
 if (arr_Z.Length != res) arr_Z = new float[res];

 Matrix.FloatToBytes converter = new Matrix.FloatToBytes();

 for (int x=0; x<res; x++)
 {
  int pos = x*4;
  converter.b0 = bytes[pos];
  converter.b1 = bytes[pos +1];
  converter.b2 = bytes[pos +2];
  converter.b3 = bytes[pos +3];
  arr_x[x] = converter.f*2; //texture requires halved value for some reason
```

```
  pos = (x + res*(res-1) )*4;
  converter.b0 = bytes[pos];
  converter.b1 = bytes[pos +1];
  converter.b2 = bytes[pos +2];
  converter.b3 = bytes[pos +3];
  arr_X[x] = converter.f*2;
 }

 for (int z=0; z<res; z++)
 {
  int pos = z*res*4;
  converter.b0 = bytes[pos];
  converter.b1 = bytes[pos +1];
  converter.b2 = bytes[pos +2];
  converter.b3 = bytes[pos +3];
  arr_z[z] = converter.f*2;

  pos = (z*res + res-1)*4;
  converter.b0 = bytes[pos];
  converter.b1 = bytes[pos +1];
  converter.b2 = bytes[pos +2];
  converter.b3 = bytes[pos +3];
  arr_Z[z] = converter.f*2;
 }
}
```

```
public void WriteRawFloat (byte[] bytes)
{
 int res = (int)Mathf.Sqrt(bytes.Length/4);

 Matrix.FloatToBytes converter = new Matrix.FloatToBytes();

 for (int x=0; x<res; x++)
 {
  converter.f = arr_x[x]/2; //texture requires halved value for some reason
  int pos = x*4;
  bytes[pos] = converter.b0;
  bytes[pos +1] = converter.b1;
  bytes[pos +2] = converter.b2;
  bytes[pos +3] = converter.b3;

  converter.f = arr_X[x]/2;
  pos = (x + res*(res-1) )*4;
  bytes[pos] = converter.b0;
  bytes[pos +1] = converter.b1;
  bytes[pos +2] = converter.b2;
  bytes[pos +3] = converter.b3;
 }

 for (int z=0; z<res; z++)
```

```
 {
  converter.f = arr_z[z]/2;
  int pos = z*res*4;
  bytes[pos] = converter.b0;
  bytes[pos +1] = converter.b1;
  bytes[pos +2] = converter.b2;
  bytes[pos +3] = converter.b3;

  converter.f = arr_Z[z]/2;
  pos = (z*res + res-1)*4;
  bytes[pos] = converter.b0;
  bytes[pos +1] = converter.b1;
  bytes[pos +2] = converter.b2;
  bytes[pos +3] = converter.b3;
 }
}


public void ReadDelta2D (float[,] heights2D)
/// Reads not the edge values, but the delta between edges and 2nd pixel from edge
{
 int sizeX = heights2D.GetLength(1);
 int sizeZ = heights2D.GetLength(0);

 if (arr_x.Length != sizeX) arr_x = new float[sizeX];
 if (arr_X.Length != sizeX) arr_X = new float[sizeX];
```

```
 if (arr_z.Length != sizeZ) arr_z = new float[sizeZ];
 if (arr_Z.Length != sizeZ) arr_Z = new float[sizeZ];

 for (int x=0; x<sizeX; x++)
 {
  arr_x[x] = heights2D[0,x] - heights2D[1,x];
  arr_X[x] = heights2D[sizeZ-1,x] - heights2D[sizeZ-2,x];
 }

 for (int z=0; z<sizeZ; z++)
 {
  arr_z[z] = heights2D[z,0] - heights2D[z,1];
  arr_Z[z] = heights2D[z, sizeX-1] - heights2D[z, sizeX-2];
 }
}

public void ReadFloats3D (float[,,] splats2D, int ch)
{
 int sizeX = splats2D.GetLength(1);
 int sizeZ = splats2D.GetLength(0);

 if (arr_x.Length != sizeX) arr_x = new float[sizeX];
 if (arr_X.Length != sizeX) arr_X = new float[sizeX];
 if (arr_z.Length != sizeZ) arr_z = new float[sizeZ];
 if (arr_Z.Length != sizeZ) arr_Z = new float[sizeZ];
```

```
 for (int x=0; x<sizeX; x++)
 {
  arr_x[x] = splats2D[0, x, ch];
  arr_X[x] = splats2D[sizeZ-1, x, ch];
 }

 for (int z=0; z<sizeZ; z++)
 {
  arr_z[z] = splats2D[z, 0, ch];
  arr_Z[z] = splats2D[z, sizeX-1, ch];
 }
}

public void ReadSplats (float[,,] splats, int ch)
{
 int sizeX = splats.GetLength(1);
 int sizeZ = splats.GetLength(0);

 if (arr_x.Length != sizeX) arr_x = new float[sizeX];
 if (arr_X.Length != sizeX) arr_X = new float[sizeX];
 if (arr_z.Length != sizeZ) arr_z = new float[sizeZ];
 if (arr_Z.Length != sizeZ) arr_Z = new float[sizeZ];

 for (int x=0; x<sizeX; x++)
 {
  arr_x[x] = splats[0, x, ch];
```

```
  arr_X[x] = splats[sizeZ-1, x, ch];
 }

 for (int z=0; z<sizeZ; z++)
 {
  arr_z[z] = splats[z, 0, ch];
  arr_Z[z] = splats[z, sizeX-1, ch];
 }
}

public void WriteSplats (float[,,] splats, int ch)
{
 int sizeX = splats.GetLength(1);
 int sizeZ = splats.GetLength(0);

 for (int x=0; x<sizeX; x++)
 {
  splats[0, x, ch] = arr_x[x];
  splats[sizeZ-1, x, ch] = arr_X[x];
 }

 for (int z=0; z<sizeZ; z++)
 {
  splats[z, 0, ch] = arr_z[z];
  splats[z, sizeX-1, ch] = arr_Z[z];
```

```
 }
}



public void ReadColors (Color[] colors, int ch)
{
 int res = (int)Mathf.Sqrt(colors.Length);

 if (arr_x.Length != res) arr_x = new float[res];
 if (arr_X.Length != res) arr_X = new float[res];
 if (arr_z.Length != res) arr_z = new float[res];
 if (arr_Z.Length != res) arr_Z = new float[res];

 for (int x=0; x<res; x++)
 {
  int pos = x;
  arr_x[x] = colors[x][ch];
  arr_X[x] = colors[x + res*(res-1)][ch];
 }

 for (int z=0; z<res; z++)
 {
  arr_z[z] = colors[z*res][ch];
  arr_Z[z] = colors[z*res + res-1][ch];
 }
}
```

```
  public void WriteColors (Color[] colors, int ch)
  {
   int res = (int)Mathf.Sqrt(colors.Length);

   for (int x=0; x<res; x++)
   {
    int pos = x;
    colors[x][ch] = arr_x[x];
    colors[x + res*(res-1)][ch] = arr_X[x];
   }

   for (int z=0; z<res; z++)
   {
    colors[z*res][ch] = arr_z[z];
    colors[z*res + res-1][ch] = arr_Z[z];
   }
  }

 #endregion
}

[System.Serializable]
public struct Lowered
///Gets/sets lowered state by direction coord
```

```
{
 [SerializeField] private bool lowered_x, lowered_X, lowered_z, lowered_Z; //edges where the draft is weld


 public bool this[Coord dir, Coord thisCoord]
 {
  get{
   bool val = false;
   if (dir.x==0  &&  dir.z==-1) val = lowered_x;
   else if (dir.x==0  &&  dir.z==1) val = lowered_X;
   else if (dir.x==-1  &&  dir.z==0) val = lowered_z;
   else if (dir.x==1  &&  dir.z==0) val = lowered_Z;

   Debug.Log("CHECK Low state for " + thisCoord + " dir " + dir + " is " + val);
   return val; }
 }

 public bool this[Coord dir]
 {
  get{
   if (dir.x==0  &&  dir.z==-1) return lowered_x;
   else if (dir.x==0  &&  dir.z==1) return lowered_X;
   else if (dir.x==-1  &&  dir.z==0) return lowered_z;
   else if (dir.x==1  &&  dir.z==0) return lowered_Z;
   else return false; }
  set{
   if (dir.x==0  &&  dir.z==-1) lowered_x = value;
```

```
   else if (dir.x==0  &&  dir.z==1) lowered_X = value;
   else if (dir.x==-1  &&  dir.z==0) lowered_z = value;
   else if (dir.x==1  &&  dir.z==0) lowered_Z = value; }
 }
}



public static class Weld
{
 private static readonly Coord[] weldDirections = new Coord[] { new Coord(1,0), new Coord(-1,0), new Coo
 private static readonly Coord[] cornerDirections = new Coord[] { new Coord(1,1), new Coord(-1,1), new Co

 public static Action<TileData, EdgesSet> ReadEdgesCustom;
 public static Action<TileData, EdgesSet> WriteEdgesCustom;
 //to perform welding operation in other assembles like MicroSplat

 public static void WeldEdges (Edges src, Coord srcCoord, Edges dst, Coord dstCoord)
 {
  Coord dir = srcCoord-dstCoord;

  if (dir.x==0  &&  dir.z==1)
   WeldArrays(src.arr_x, dst.arr_X);

  else if (dir.x==0  &&  dir.z==-1)
   WeldArrays(src.arr_X, dst.arr_x);
```

```
 else if (dir.x==1  &&  dir.z==0)
  WeldArrays(src.arr_z, dst.arr_Z);

 else if (dir.x==-1  &&  dir.z==0)
  WeldArrays(src.arr_Z, dst.arr_z);
}


public static void WeldArrays (float[] src, float[] dst, bool lowerOnMismatch=false)
{
 //if arrays size match - simply copy them
 if (src.Length == dst.Length)
  Array.Copy(src, dst, src.Length);
  //for (int i=0; i<src.Length; i++) dst[i] = dst[i]+0.1f;

 //welding draft with the main tile
 else if (src.Length > dst.Length)
 {
  int srcStep = (src.Length-1) / (dst.Length-1);

  //assigning value
  for (int i=0; i<dst.Length; i++)
   dst[i] = src[i*srcStep];

  //avoiding holes (lowering edges)
  if (lowerOnMismatch)
```

```
  {
   float prevLower = 0;
   for (int i=0; i<dst.Length-1; i++)
   {
    float nextLower = 0;
    for (int j=0; j<srcStep; j++)
    {
     float percent = 1f*j / srcStep;
     float interpolatedDst = dst[i]*(1-percent) + dst[i+1]*percent;
     float realSrc = src[i*srcStep + j];
     float delta = interpolatedDst - realSrc;
     if (delta>nextLower) nextLower = delta;
    }

    dst[i] -= prevLower>nextLower ? prevLower : nextLower;

    prevLower = nextLower;
   }
  }
 }
}


public static void WeldArraysDebug (float[] src, float[] dst, bool lowerOnMismatch=false)
{
 //if (lowerOnMismatch)
```

```
 // for (int i=0; i<dst.Length-1; i++)
 //  dst[i] = 0;
}


#region Weld in Thread

 public static void ReadEdges (TileData thisData, EdgesSet thisEdges)
 /// Reads apply data and converts to edges. Edges are ready to use after this stage.
 {
  HeightOutput200.ApplySetData heightDataFull = thisData.ApplyOfType<HeightOutput200.ApplySetData
  if (heightDataFull != null)
   thisEdges.heightEdges.ReadFloats2D(heightDataFull.heights2D);

  HeightOutput200.ApplySplitData heightSplitData = thisData.ApplyOfType<HeightOutput200.ApplySplitD
  if (heightSplitData != null)
   thisEdges.heightEdges.ReadSplitFloats2D(heightSplitData.heights2DSplits);

  #if UNITY_2019_1_OR_NEWER
  HeightOutput200.ApplyTexData heightTexData = thisData.ApplyOfType<HeightOutput200.ApplyTexDat
  if (heightTexData != null)
   thisEdges.heightEdges.ReadRawFloat(heightTexData.texBytes);
  #endif


  TexturesOutput200.ApplyData texturesData = thisData.ApplyOfType<TexturesOutput200.ApplyData>();
```

```
 if (texturesData != null && texturesData.splats!=null)
 {
  int numChs = texturesData.splats.GetLength(2);
  if (thisEdges.splatEdges==null || thisEdges.splatEdges.Length != numChs)
   Array.Resize(ref thisEdges.splatEdges, numChs);

  for (int ch=0; ch<numChs; ch++)
  {
   if (thisEdges.splatEdges[ch] == null)
    thisEdges.splatEdges[ch] = new Edges(0,0);

   thisEdges.splatEdges[ch].ReadSplats(texturesData.splats, ch);
  }
 }

 ReadEdgesCustom?.Invoke(thisData, thisEdges);

 thisEdges.ready = true;
}


public static void WeldEdgesInThread (EdgesSet thisEdges, TerrainTileManager tileManager, Coord coo
/// Welds edges (does not affect data)
{
 if (tileManager == null) return;
```

```
for (int d=0; d<weldDirections.Length; d++)
{
 TerrainTile neigTile = tileManager[coord + weldDirections[d]];
 if (neigTile == null  ||
  (isDraft  &&  neigTile.draft == null)  ||  (isDraft  &&  !neigTile.draft.generateReady)  ||
  (!isDraft  &&  neigTile.main  == null)  ||  (!isDraft  &&  !neigTile.main.generateReady))
   continue;
   //if null or not generate-ready

 EdgesSet neigEdges = isDraft ? neigTile.draft?.edges : neigTile.main?.edges;
 if (neigEdges==null || !neigEdges.ready) continue;

 //height
 if (neigEdges.heightEdges != null && neigEdges.heightEdges.SizeX == thisEdges.heightEdges.SizeX)
  Weld.WeldEdges(neigEdges.heightEdges, neigTile.coord, thisEdges.heightEdges, coord);

 //splats
 if (neigEdges.splatEdges != null  &&  thisEdges.splatEdges != null  &&  thisEdges.splatEdges.Length ==
  for (int i=0; i<thisEdges.splatEdges.Length; i++)
 {
  if (thisEdges.splatEdges[i].SizeX == neigEdges.splatEdges[i].SizeX)
   Weld.WeldEdges(neigEdges.splatEdges[i], neigTile.coord, thisEdges.splatEdges[i], coord);
 }

 //textures
 if (neigEdges.controlEdges != null && thisEdges.controlEdges != null && thisEdges.controlEdges.Lengt
```

```
   for (int i=0; i<thisEdges.controlEdges.Length; i++)
  {
   if (thisEdges.controlEdges[i].SizeX == neigEdges.controlEdges[i].SizeX)
    Weld.WeldEdges(neigEdges.controlEdges[i], neigTile.coord, thisEdges.controlEdges[i], coord);
  }
 }
}


public static void WriteEdges (TileData thisData, EdgesSet thisEdges)
/// Writes edges back to apply data after they have been changed
{
 HeightOutput200.ApplySetData heightDataFull = thisData.ApplyOfType<HeightOutput200.ApplySetData
 if (heightDataFull != null)
  thisEdges.heightEdges.WriteFloats2D(heightDataFull.heights2D);

 HeightOutput200.ApplySplitData heightSplitData = thisData.ApplyOfType<HeightOutput200.ApplySplitD
 if (heightSplitData != null)
  thisEdges.heightEdges.WriteSplitFloats2D(heightSplitData.heights2DSplits);

 #if UNITY_2019_1_OR_NEWER
 HeightOutput200.ApplyTexData heightTexData = thisData.ApplyOfType<HeightOutput200.ApplyTexDat
 if (heightTexData != null)
  thisEdges.heightEdges.WriteRawFloat(heightTexData.texBytes);
 #endif
```

```
  TexturesOutput200.ApplyData texturesData = thisData.ApplyOfType<TexturesOutput200.ApplyData>();
  if (texturesData != null && texturesData.splats!=null)
  {
   int numChs = texturesData.splats.GetLength(2);
   if (thisEdges.splatEdges==null || thisEdges.splatEdges.Length != numChs)
    Array.Resize(ref thisEdges.splatEdges, numChs);

   for (int ch=0; ch<numChs; ch++)
    thisEdges.splatEdges[ch].WriteSplats(texturesData.splats, ch);
  }

  WriteEdgesCustom?.Invoke(thisData, thisEdges);
 }

#endregion


#region Weld on LOD switch

 public static void WeldSurroundingDraftsToThisMain (TerrainTileManager tileManager, Coord coord)
 /// Welds neig draft terrains to this one when switched on main, lowering draft edges
 /// Called in main thread on switch lods
 /// On drafts only because of the performance reasons
 {
  if (tileManager == null) return;
```

```
 TerrainTile thisTile = tileManager[coord];


 for (int d=0; d<weldDirections.Length; d++)
 {
  TerrainTile neigTile = tileManager[coord + weldDirections[d]];
  if (neigTile?.draft == null) continue;
  if (!neigTile.draft.applyReady || !neigTile.draft.generateReady) continue;


  if (neigTile.ActiveTerrain == neigTile.draft.terrain)
  {
   if (neigTile.draft.edges.lowered[-weldDirections[d]]) continue;


   WeldDraftToMain(thisTile, neigTile, weldDirections[d]);
   neigTile.draft.edges.lowered[-weldDirections[d]] = true;
  }
 }
}



public static void WeldThisDraftWithSurroundings (TerrainTileManager tileManager, Coord coord)
/// Welds this tile to tiles around it on switch to draft, lowering if surround is main or restoring weld if draft
/// This will restore edges in surrounding drafts too
{
 if (tileManager == null) return;
 TerrainTile thisTile = tileManager[coord];
```

```
for (int d=0; d<weldDirections.Length; d++)
{
 Coord dir = weldDirections[d];

 TerrainTile neigTile = tileManager[coord + weldDirections[d]];
 if (neigTile == null) continue;

 //restoring edges
 if (neigTile.draft != null  &&  neigTile.ActiveTerrain == neigTile.draft.terrain)
 {
  if (neigTile.draft.edges.lowered[-dir])
  {
   RestoreDraftWeld(neigTile, -dir);
   neigTile.draft.edges.lowered[-dir] = false;
  }

  if (thisTile.draft.edges.lowered[dir])
  {
   RestoreDraftWeld(thisTile, dir);
   thisTile.draft.edges.lowered[dir] = false;
  }
 }

 //lowering this edges
 else if (neigTile.main != null  &&  neigTile.ActiveTerrain == neigTile.main.terrain)
 {
```

```
    if (thisTile.draft.edges.lowered[dir]) continue;
    WeldDraftToMain(neigTile, thisTile, -dir);
    thisTile.draft.edges.lowered[dir] = true;
   }
  }
}

private static void RestoreDraftWeld (TerrainTile welded, Coord weldDir)
/// Restoring height edges if they were lowered by welding with main
{
 EdgesSet edges = welded.draft.edges;
 float[] arr = edges.heightEdges.GetArr(weldDir);
 Weld.ApplyToTerrain(welded.draft.terrain.terrainData, arr, weldDir);
}

private static void WeldDraftToMain (TerrainTile reference, TerrainTile welded, Coord weldDir)
/// Lowers edges on welding to main tile
{
 if (welded.draft.edges.heightEdges.IsEmpty)
  welded.draft.edges.heightEdges.ReadFloats2D(welded.draft.terrain.terrainData.GetHeights(0,0,welded.
  //for an unknown reason ready chunks in scene do not have their edges (generated or saved). Leads to

 EdgesSet refEdges = reference.main.edges;
 EdgesSet weldEdges = welded.draft.edges;
```

```
 //loading saved edges big array
 float[] refArr = refEdges.heightEdges.GetArr(weldDir);

 //duplicating neig saved edges short array
 float[] weldArr = weldEdges.heightEdges.GetArr(-weldDir);
 float[] weldArrCopy = new float[weldArr.Length];
 //Array.Copy(weldArr, weldArrCopy, weldArr.Length); //will be re-filled anyways

 //welding and apply to terrain
 Weld.WeldArrays(refArr, weldArrCopy, lowerOnMismatch:true);

 Weld.ApplyToTerrain(welded.draft.terrain.terrainData, weldArrCopy, -weldDir);

}


private static void ApplyToTerrain (TerrainData terrData, float[] arr, Coord dir)
{
 if (arr.Length == 0)
  throw new Exception("Empty weld array. Possibly terrain has not been generated yet");

 int size = terrData.heightmapResolution;

 if (dir.x==-1  &&  dir.z==0)
 {
```

```
 float[,] arr_x_2D = new float[size,1];
 for (int i=0; i<size; i++) arr_x_2D[i,0] = arr[i];
 terrData.SetHeightsDelayLOD(0,0,arr_x_2D);
}

else if (dir.x==1  &&  dir.z==0)
{
 float[,] arr_x_2D = new float[size,1];
 for (int i=0; i<size; i++) arr_x_2D[i,0] = arr[i];
 terrData.SetHeightsDelayLOD(size-1,0,arr_x_2D);
}

else if (dir.x==0  &&  dir.z==-1)
{
 float[,] arr_z_2D = new float[1,size];
 for (int i=0; i<size; i++) arr_z_2D[0,i] = arr[i];
 terrData.SetHeightsDelayLOD(0,0,arr_z_2D);
}

else if (dir.x==0  &&  dir.z==1)
{
 float[,] arr_z_2D = new float[1,size];
 for (int i=0; i<size; i++) arr_z_2D[0,i] = arr[i];
 terrData.SetHeightsDelayLOD(0,size-1,arr_z_2D);
}
}
```

```
public static void WeldCorners (TerrainTileManager tileManager, Coord coord, bool isDraft=false)
/// Iterates all terrain at the corner and chooses the first occurrence value.
{
 if (tileManager == null) return;
 TerrainTile thisTile = tileManager[coord];
 Terrain thisTerrain = isDraft ? thisTile.draft?.terrain : thisTile.main?.terrain;
 TerrainData thisData = thisTerrain.terrainData;
 if (thisTerrain == null) return;
 int resolution = thisTerrain.terrainData.heightmapResolution;
 float[,] tempArr = new float[1,1];

 for (int c=0; c<cornerDirections.Length; c++)
 {
  Coord corner = (cornerDirections[c]+1) / 2;
  //float thisHeight = thisData.GetHeight((corner.x+1)/2, (corner.z+1)/2);

  //reading and selecting maximum
  for (int d=0; d<cornerDirections.Length; d++)
  {
   Coord dir = (cornerDirections[d]+1) / 2;

   TerrainTile cornerTile = tileManager[coord + dir+corner-1];
   if (cornerTile == null || cornerTile == thisTile) continue;
```

```
    Terrain cornerTerrain = isDraft ? cornerTile.draft?.terrain : cornerTile.main?.terrain;
    if (cornerTerrain==null || !cornerTerrain.isActiveAndEnabled) continue;
    TerrainData cornerData = cornerTerrain.terrainData;

    float cornerHeight = cornerData.GetHeight((1-dir.x) * (resolution-1), (1-dir.z) * (resolution-1)) / cornerDa
    tempArr[0,0] = cornerHeight;
    thisData.SetHeightsDelayLOD(corner.x * (resolution-1), corner.z * (resolution-1), tempArr);

    break;
   }
  }
 }

#endregion


#region Setting Neightbor

 //It all should work, but Unity calls Terrain.SetConnectivityDirty on each terrain enable or disable

 public static void SetNeighbors (TerrainTileManager tileManager, Coord coord)
 {
  if (tileManager == null) return;
  TerrainTile thisTile = tileManager[coord];

  //foreach (TerrainTile tile in tileManager.Tiles())
```

```
TerrainTile tile = tileManager[coord];
coord = tile.coord;
{
 Terrain terrain = tile.main?.terrain; //isDraft ? tile.draft?.terrain : tile.main?.terrain;
 //if (terrain == null || !terrain.isActiveAndEnabled) continue;


 //for (int d=0; d<weldDirections.Length; d++)
 //unroll

 Terrain leftNeighbor = tileManager[ new Coord(coord.x-1, coord.z) ]?.main?.terrain;
 if (leftNeighbor!=null && leftNeighbor.isActiveAndEnabled)
 {
  terrain.SetNeighbors(leftNeighbor, terrain.topNeighbor, terrain.rightNeighbor, terrain.bottomNeighbor);
  leftNeighbor.SetNeighbors(leftNeighbor.leftNeighbor, leftNeighbor.topNeighbor, terrain, leftNeighbor.bo
 }

 Terrain rightNeighbor = tileManager[ new Coord(coord.x+1, coord.z) ]?.main?.terrain;
 if (rightNeighbor!=null && rightNeighbor.isActiveAndEnabled)
 {
  terrain.SetNeighbors(terrain.leftNeighbor, terrain.topNeighbor, rightNeighbor, terrain.bottomNeighbor);
  rightNeighbor.SetNeighbors(terrain, rightNeighbor.topNeighbor, rightNeighbor.rightNeighbor, rightNeigh
 }

 Terrain bottomNeighbor = tileManager[ new Coord(coord.x, coord.z-1) ]?.main?.terrain;
 if (bottomNeighbor!=null && bottomNeighbor.isActiveAndEnabled)
```

```
  {
   terrain.SetNeighbors(terrain.leftNeighbor, terrain.topNeighbor, terrain.rightNeighbor, bottomNeighbor);
   bottomNeighbor.SetNeighbors(bottomNeighbor.leftNeighbor, terrain, bottomNeighbor.rightNeighbor, bo
  }

  Terrain topNeighbor = tileManager[ new Coord(coord.x, coord.z+1) ]?.main?.terrain;
  if (topNeighbor!=null && topNeighbor.isActiveAndEnabled)
  {
   terrain.SetNeighbors(terrain.leftNeighbor, topNeighbor, terrain.rightNeighbor, terrain.bottomNeighbor);
   topNeighbor.SetNeighbors(topNeighbor.leftNeighbor, topNeighbor.topNeighbor, topNeighbor.rightNeigh
  }
 }
}


public static void SetNeighborsAll (TerrainTileManager tileManager)
{
 System.Diagnostics.Stopwatch timer = null;
 timer = new System.Diagnostics.Stopwatch();
 timer.Start();

 if (tileManager == null) return;
 //TerrainTile thisTile = tileManager[coord];

 foreach (TerrainTile tile in tileManager.Tiles())
  SetNeighbors(tileManager, tile.coord);
```

```
    timer.Stop();
    Debug.Log("Neighboring in " + timer.Elapsed.TotalMilliseconds + "ms" + " (" + timer.ElapsedMilliseconds
   }


   private static Terrain GetNeighbor (Terrain terrain, Coord dir)
   {
    if (dir.x==0  &&  dir.z==-1) return terrain.bottomNeighbor;
    else if (dir.x==0  &&  dir.z==1) return terrain.topNeighbor;
    else if (dir.x==-1  &&  dir.z==0) return terrain.leftNeighbor;
    else if (dir.x==1  &&  dir.z==0) return terrain.rightNeighbor;
    else return null;
   }

  #endregion
 }
}
```

```
ï»¿
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.Matrices;

using MapMagic.Nodes;
using MapMagic.Terrains;



namespace MapMagic.Core.GUI
{
 [CustomEditor(typeof(DirectMatricesHolder))]
 public class DirectMatricesHolderInspector : Editor
 {
  UI ui = new UI();

  public override void  OnInspectorGUI ()
  {
   ui.Draw(DrawGUI, inInspector:true);
  }
```

```
public void DrawGUI ()
{
 DirectMatricesHolder holder = (DirectMatricesHolder)target;

 using (Cell.Line)
 {
  Cell layersCell = Cell.current;

  using (Cell.LinePx(0))
   LayersEditor.DrawLayersThemselves(Cell.current,
    holder.maps.Count,
    onDraw:n => DrawLayer(holder.maps.GetKeyByNum(n), holder.maps[n]) );
 }
}

private static void DrawLayer (string name, Matrix map)
{
 Cell.EmptyLinePx(4);

 using (Cell.LineStd)
 {
  Cell.EmptyRowPx(4);
  //using (Cell.RowPx(64)) Draw.TextureIcon(texture);
  using (Cell.Row) Draw.Label(name);
  Cell.EmptyRowPx(4);
 }
```

```
    Cell.EmptyLinePx(4);
  }

 }//class

}//namespace
```

```
ï»¿
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;

using MapMagic.Nodes;
using MapMagic.Terrains;



namespace MapMagic.Core.GUI
{
 [CustomEditor(typeof(DirectTexturesHolder))]
 public class DirectTexturesHolderInspector : Editor
 {
  UI ui = new UI();

  public override void  OnInspectorGUI ()
  {
   ui.Draw(DrawGUI, inInspector:true);
  }

  public void DrawGUI ()
```

```
{
 DirectTexturesHolder holder = (DirectTexturesHolder)target;

 using (Cell.Line)
 {
  Cell layersCell = Cell.current;

  using (Cell.LinePx(0))
   LayersEditor.DrawLayersThemselves(Cell.current,
    holder.textures.Count,
    onDraw:n => DrawLayer(holder.textures.GetKeyByNum(n), holder.textures[n]) );
 }
}

private static void DrawLayer (string name, Texture2D texture)
{
 Cell.EmptyLinePx(4);

 using (Cell.LinePx(64))
 {
  Cell.EmptyRowPx(4);
  using (Cell.RowPx(64)) Draw.TextureIcon(texture);
  using (Cell.Row) Draw.Label(name);
  Cell.EmptyRowPx(4);
 }
```

```
    Cell.EmptyLinePx(4);
  }


 }//class


}//namespace
```

```
ï»¿
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;

using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.SceneEdit;

using MapMagic.Core;


///Draws tile frames around terrains

namespace MapMagic.Terrains.GUI
{
 public static class FrameDraw
 {
  public static readonly Color standardColor = new Color(0.3f, 0.5f, 0.8f, 1);
  public static readonly Color pinColor = new Color(0.5f, 0.7f, 1, 1);
  public static readonly Color selectPreviewColor = new Color(0.2f, 0.4f, 0.7f, 1);
  public static readonly Color previewColor = new Color(0.15f, 0.35f, 0.6f, 1);
```

```
public static readonly Color exportColor = new Color(0.31f, 0.55f, 0.26f);

public static readonly Color unpinColor = new Color(1,0,0,1);

public const float dotsPerSide = 9.5f;

public const float defaultZOffset = 0.05f;


public static int width = 5;


private static Texture2D clockIconTex;


private static ConditionalWeakTable<Terrain,PolyLine> terrainLinesCache = new ConditionalWeakTable<


#region Event


 [RuntimeInitializeOnLoadMethod, UnityEditor.InitializeOnLoadMethod]

 static void Subscribe ()

 {

  TerrainTile.OnTileApplied += UpdateTile_OnTileApplied;

 }


 private static void UpdateTile_OnTileApplied (TerrainTile tile, Products.TileData tileData, Products.StopT

 {

  Terrain terrain = tile.GetTerrain(tileData.isDraft);

  if (terrain == null) return; //seems to be happen when stopping playmode while tile generating


  if (!terrainLinesCache.TryGetValue(terrain, out PolyLine polyLine))

  {
```

```
    polyLine = CreateTerrainLine(terrain);
    if (polyLine != null) //if weak table has null it can't convert and crashes Unity
     terrainLinesCache.Add(terrain, polyLine);
   }

   else
   {
    Vector3[] lineArr = CreateLinePoints(terrain);
    if (polyLine.MaxPoints < lineArr.Length)
    {
     polyLine = CreateTerrainLine(terrain);
     terrainLinesCache.Remove(terrain);

     if (polyLine != null)
      terrainLinesCache.Add(terrain, polyLine);
    }

    else
     polyLine.SetPoints(lineArr);
   }
  }

#endregion



public static void DrawSceneGUI (MapMagicObject mapMagic)
```

```
{
 if (Event.current.type == EventType.Repaint)
 {
  Profiler.BeginSample("Drawing Frames");

  //drawing standard tiles
  foreach (TerrainTile tile in mapMagic.tiles.All())
  {
   if (!tile.Ready)
    DrawClock(tile, mapMagic.transform);

   if (tile != mapMagic.PreviewTile)
   {
    Terrain activeTerrain = tile.ActiveTerrain;
    if (activeTerrain != null)
     DrawTerrainFrame(activeTerrain, standardColor, dotted:tile.main==null);
   }
  }

  //preview tile above them
  if (mapMagic.PreviewTile != null &&  mapMagic.PreviewTile.main!=null)
  {
   DrawTerrainFrame(mapMagic.PreviewTile.main.terrain, previewColor, false, offset:defaultZOffset*1.5f);
  }

  Profiler.EndSample();
```

```
 }
}



public static void DrawClock (TerrainTile tile, Transform parent=null)
{
 if (clockIconTex == null) clockIconTex = Resources.Load<Texture2D>("MapMagic/PreviewSandClock");

 Rect tileRect =  tile.WorldRect;
 Vector3 tileCenter = new Vector3(tileRect.center.x, 0, tileRect.center.y);
 if (parent != null) tileCenter = parent.TransformPoint(tileCenter);

 Terrain activeTerrain = tile.ActiveTerrain;
 if (activeTerrain != null) tileCenter.y = activeTerrain.terrainData.bounds.center.y;
 Vector2 screenPos = HandleUtility.WorldToGUIPoint(tileCenter);

 Handles.BeginGUI();
 UnityEngine.GUI.DrawTexture( new Rect(screenPos.x-clockIconTex.width/2, screenPos.y-clockIconTex.
 Handles.EndGUI();
}



public static void DrawFrame (Coord coord, Vector3 tileSize, Color color, bool dotted, Dictionary<Coord,T
/// Drawing a frame on terrain (if found in terrainsLut) or on the ground level
{
 //drawing terrain
```

```
 Terrain terrain = null;
 if (terrainsLut != null  &&  terrainsLut.ContainsKey(coord))
  terrain = terrainsLut[coord].ActiveTerrain;

 if (terrain != null)
  DrawTerrainFrame(terrain, color, dotted, offset);

 //drawing empty
 else
  DrawEmptyFrame(coord.vector3.Mul(tileSize), new Vector3(tileSize.x,0,tileSize.z), color, dotted, offset, p
}

public static void DrawEmptyFrame (Vector3 start, Vector3 size, Color color, bool dotted, float offset=defa
{
 Vector3[] framePoints = new Vector3[] {
  start,
  start + new UnityEngine.Vector3(0,0,size.z),
  start + size,
  start + new UnityEngine.Vector3(size.x,0,0),
  start};

 if (parent != null)
  for (int i=0; i<framePoints.Length; i++)
   framePoints[i] = parent.TransformPoint(framePoints[i]);
```

```
 PolyLine.InstantLine(framePoints, color, width, dotted ? size.x/dotsPerSide : 0, offset:offset);
 //Handles.DrawAAPolyLine(width, framePoints);
}


public static void DrawTerrainFrame (Terrain terrain, Color color, bool dotted, float offset=defaultZOffset)
{
 if (!terrainLinesCache.TryGetValue(terrain, out PolyLine polyLine))
 {
  polyLine = CreateTerrainLine(terrain);
  if (polyLine != null) //if weak table has null it can't convert and crashes Unity
   terrainLinesCache.Add(terrain, polyLine);
 }

 polyLine.DrawLine(color, width, dotted ? terrain.terrainData.size.x/dotsPerSide : 0, offset:offset, parent:te
}


private static PolyLine CreateTerrainLine (Terrain terrain)
{
 Vector3[] lineArr = CreateLinePoints(terrain);
 PolyLine polyLine = new PolyLine(lineArr.Length);
 polyLine.SetPoints(lineArr);

 return polyLine;
}
```

```
private static Vector3[] CreateLinePoints (Terrain terrain)
{
 TerrainData data = terrain.terrainData;
 int heightmapResolution = data.heightmapResolution;
 int resolution = heightmapResolution; //Mathf.Min(heightmapResolution, 128);

 Vector3 terrainPos = terrain.transform.localPosition;
 Vector3 terrainSize = data.size;

 Vector3[] lineArr = new Vector3[resolution*4 - 3]; //dnw 3 works

 float[,] heights = data.GetHeights(0,0, 1, heightmapResolution);
 for (int x=0; x<heightmapResolution; x++)
 {
  float val = heights[x,0]; //x and z swaped

  float percent = 1f * x / (heightmapResolution-1);
  lineArr[x] = new Vector3(terrainPos.x, val*terrainSize.y + terrainPos.y, terrainPos.z + percent*terrainSize
 }

 heights = data.GetHeights(0,heightmapResolution-1, heightmapResolution, 1);
 for (int z=1; z<heightmapResolution; z++)
 {
  float val = heights[0,z]; //x and z swaped
```

```
   float percent = 1f * z / (heightmapResolution-1);
   lineArr[heightmapResolution-1 + z] = new Vector3(terrainPos.x + percent*terrainSize.x, val*terrainSize.y
  }

  heights = data.GetHeights(heightmapResolution-1, 0, 1, heightmapResolution);
  for (int x=1; x<heightmapResolution; x++)
  {
   float val = heights[heightmapResolution-1-x,0]; //x and z swaped

   float percent = 1 - 1f * x / (heightmapResolution-1);
   lineArr[(heightmapResolution-1)*2 + x] = new Vector3(terrainPos.x + terrainSize.x, val*terrainSize.y + te
  }

  heights = data.GetHeights(0,0, heightmapResolution, 1);
  for (int z=1; z<heightmapResolution; z++)
  {
   float val = heights[0,heightmapResolution-1-z]; //x and z swaped

   float percent = 1 - 1f * z / (heightmapResolution-1);
   lineArr[(heightmapResolution-1)*3 + z] = new Vector3(terrainPos.x + percent*terrainSize.x, val*terrainSi
  }

  return lineArr;
 }
}
```

```
}
```

```
ï»¿using System;
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;

using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.SceneEdit;

using MapMagic.Core;



namespace MapMagic.Terrains.GUI
{
 public static class PinDraw
 {
  private static readonly Color pinButtonColor = new Color(0.5f, 0.58f, 0.7f);
  private static readonly Color previewButtonColor = new Color(0.22f, 0.42f, 0.69f);
  private static readonly Color exportButtonColor = new Color(0.31f, 0.55f, 0.26f);
  private static readonly Color unpinButtonColor = new Color(0.68f, 0.14f, 0.15f);

  private const float margins = 0;
  private const int numSteps = 100;
```

```
private const float width = 4;

private static PolyLine polyLine; //to draw frames
private static Texture2D lineTex;

public enum SelectionMode { none, pin, pinLowres, pinExisting, selectPreview, export, exportCopy, expor



public static void DrawInspectorGUI (MapMagicObject mapMagic, ref SelectionMode selectionMode, ref b
{
 Cell.EmptyLinePx(2);

 using (Cell.LinePx(25)) DrawPinButton(ref selectionMode, SelectionMode.pin, "MapMagic/Icons/Pin", "M
 using (Cell.LinePx(25)) DrawPinButton(ref selectionMode, SelectionMode.pinLowres, "MapMagic/Icons/P
 //using (Cell.LinePx(25)) DrawPinButton(ref selectionMode, SelectionMode.pinExisting, "MapMagic/Icons

 Cell.EmptyLinePx(5);
 using (Cell.LinePx(25)) DrawPinButton(ref selectionMode, SelectionMode.selectPreview, "MapMagic/Ico

 Cell.EmptyLinePx(5);
 using (Cell.LinePx(25))
 {
  bool saveDraftCl = saveDraft;
  /*void DrawSaveDraft()
  {
   using (Cell.RowPx(60))
```

```
    {
     Cell.EmptyLine();
     using (Cell.LineStd) Draw.ToggleLeft(ref saveDraftCl, "Draft");
     Cell.EmptyLine();
    }
   }*/
   DrawPinButton(ref selectionMode, SelectionMode.export, "MapMagic/Icons/Export", "MapMagic/PinButt
   saveDraft = saveDraftCl;

   //save draft will cause additional save file dilog, and actually usless
  }
  using (Cell.LinePx(25)) DrawPinButton(ref selectionMode, SelectionMode.exportCopy, "MapMagic/Icons/

  using (Cell.LinePx(25)) DrawPinButton(ref selectionMode, SelectionMode.exportCluster, "MapMagic/Icon



  Cell.EmptyLinePx(5);
  using (Cell.LinePx(25)) DrawPinButton(ref selectionMode, SelectionMode.unpin, "MapMagic/Icons/Unpin

  Cell.EmptyLinePx(2);
 }

 private static void DrawPinButton (ref SelectionMode selectionMode, SelectionMode buttonMode, string ic
 {
  using (Cell.Padded(0,0,-1,-1))
  {
```

```
bool isPinning = selectionMode==buttonMode;

Draw.CheckButton(ref isPinning, visible:false);

GUIStyle style = UI.current.textures.GetElementStyle(isPinning ? buttonName+"_pressed" : buttonName
Draw.Element(style);

Cell.EmptyRowPx(10);

using (Cell.RowPx(30))
{
 //using (Cell.Padded(1,1,1,1))
 //{
 // if (isPinning) { color = color/2; color.a=1; }
 // Draw.Rect(color);
 //}

 Texture2D icon = UI.current.textures.GetTexture(iconName);
 Draw.Icon(icon, scale:iconScale);
}

using (Cell.Row) Draw.Label(label, style:UI.current.styles.middleLabel);

additionalElement?.Invoke();

if (Cell.current.valChanged)
```

```
  {
   if (isPinning) selectionMode = buttonMode;
   else selectionMode = SelectionMode.none;
  }
 }
}


private static bool DrawButton (ref SelectionMode selectionMode, SelectionMode buttonMode, string icon
// Taking same visual appearance from DrawPinButton, but it's pressed momentarily
// Was going to be used for Cluster
{
 using (Cell.Padded(0,0,-1,-1))
 {
  bool isPressed = Draw.Button(visible:false);
  bool isPrePressed = Draw.IsButtonPrePressed;

  if (isPressed || isPrePressed) selectionMode = SelectionMode.none;

  GUIStyle style = UI.current.textures.GetElementStyle(isPrePressed ? buttonName+"_pressed" : buttonN
  Draw.Element(style);

  Cell.EmptyRowPx(10);

  using (Cell.RowPx(30))
  {
```

```
    Texture2D icon = UI.current.textures.GetTexture(iconName);
    Draw.Icon(icon, scale:iconScale);
   }

   using (Cell.Row) Draw.Label(label, style:UI.current.styles.middleLabel);

   additionalElement?.Invoke();

   return isPressed;
  }
 }

public static void DrawSceneGUI (MapMagicObject mapMagic, ref SelectionMode selectionMode, bool sa
{
 if (selectionMode!=SelectionMode.none  &&  !Event.current.alt)
 {
  Dictionary<Coord,TerrainTile> tilesLut = new Dictionary<Coord,TerrainTile>(mapMagic.tiles.grid);

  //returning if no scene veiw (right after script compile)
  SceneView sceneview = UnityEditor.SceneView.lastActiveSceneView;
  if (sceneview==null || sceneview.camera==null) return;

  //disabling selection
  HandleUtility.AddDefaultControl(GUIUtility.GetControlID(FocusType.Passive));

  //canceling any pin on esc, alt or right-click
```

```
if (Event.current.keyCode == KeyCode.Escape  ||  Event.current.alt)
{
 selectionMode = SelectionMode.none;
 Select.CancelFrame();
// EditorWindow.GetWindow FindObjectOfType<Core.GUI.MapMagicInspector>().Repaint();
}

//preparing the sets of both custom and tile terrains
HashSet<Terrain> pinnedCustomTerrains = new HashSet<Terrain>(); //custom terrains that were alread
foreach (TerrainTile tile in mapMagic.tiles.customTiles)
{
 if (tile.main != null) pinnedCustomTerrains.Add(tile.main.terrain);
 if (tile.draft != null) pinnedCustomTerrains.Add(tile.draft.terrain);
}

HashSet<Terrain> pinnedTileTerrains = new HashSet<Terrain>();
foreach (var kvp in tilesLut)
{
 TerrainTile tile = kvp.Value;
 if (tile.main != null) pinnedTileTerrains.Add(tile.main.terrain);
 if (tile.draft != null) pinnedTileTerrains.Add(tile.draft.terrain);
}

if (selectionMode == SelectionMode.pin)
{
 Handles.color = FrameDraw.pinColor;
```

```
 List<Coord> selectedCoords = TerrainAiming.SelectTiles((Vector3)mapMagic.tileSize, false, tilesLut, ma
 if (selectedCoords != null)
 {
  UnityEditor.Undo.RegisterFullObjectHierarchyUndo(mapMagic.gameObject, "MapMagic Pin Terrains");

  foreach (Coord coord in selectedCoords)
   mapMagic.tiles.Pin(coord, false, mapMagic);

  foreach (Coord coord in selectedCoords)
   UnityEditor.Undo.RegisterCreatedObjectUndo(mapMagic.tiles[coord].gameObject, "MapMagic Pin Ter
 }
}

if (selectionMode == SelectionMode.pinLowres)
{
 Handles.color = FrameDraw.pinColor;

 List<Coord> selectedCoords = TerrainAiming.SelectTiles((Vector3)mapMagic.tileSize, true, tilesLut, ma
 if (selectedCoords != null)
 {
  UnityEditor.Undo.RegisterFullObjectHierarchyUndo(mapMagic.gameObject, "MapMagic Pin Draft Terra

  foreach (Coord coord in selectedCoords)
   mapMagic.tiles.Pin(coord, true, mapMagic);
```

```
  foreach (Coord coord in selectedCoords)
   UnityEditor.Undo.RegisterCreatedObjectUndo(mapMagic.tiles[coord].gameObject, "MapMagic Pin Ter
 }
}

if (selectionMode == SelectionMode.pinExisting)
{
 //excluding tiles
 HashSet<Terrain> possibleTerrains = new HashSet<Terrain>();
 Terrain[] allTerrains = GameObject.FindObjectsOfType<Terrain>();
 foreach (Terrain terrain in allTerrains)
  if (!pinnedTileTerrains.Contains(terrain)) possibleTerrains.Add(terrain);

 HashSet<Terrain> selectedTerrains = TerrainAiming.SelectTerrains(possibleTerrains, FrameDraw.pinC
 if (selectedTerrains != null)
 {
  UnityEditor.Undo.RegisterFullObjectHierarchyUndo(mapMagic.gameObject, "MapMagic Pin Terrains");

  foreach (Terrain terrain in selectedTerrains)
  {
   if (pinnedCustomTerrains.Contains(terrain)) continue;
   terrain.transform.parent = mapMagic.transform;

   TerrainTile tile = terrain.gameObject.GetComponent<TerrainTile>();
   if (tile == null) tile = terrain.gameObject.AddComponent<TerrainTile>();
   tile.main.terrain = terrain;
```

```
    //tile.main.area = new Terrains.Area(terrain);
    //tile.main.use = true;
    //tile.lodData = null;

    mapMagic.tiles.PinCustom(tile);
    mapMagic.StartGenerate(tile);
   }

  }
}



if (selectionMode == SelectionMode.selectPreview)
{
 //hash set of all pinned terrains contains main data
 HashSet<Terrain> pinnedMainTerrains = new HashSet<Terrain>();
 foreach (var kvp in tilesLut)
 {
  TerrainTile tile = kvp.Value;
  if (tile.main != null) pinnedMainTerrains.Add(tile.main.terrain);
 }
 pinnedMainTerrains.UnionWith(pinnedCustomTerrains);

 HashSet<Terrain> selectedTerrains = TerrainAiming.SelectTerrains(pinnedMainTerrains, FrameDraw.s
 if (selectedTerrains != null && selectedTerrains.Count == 1)
 {
```

```
UnityEditor.Undo.RegisterCompleteObjectUndo(mapMagic, "MapMagic Select Preview");

Terrain selectedTerrain = selectedTerrains.Any();

//clearing preview
if (selectedTerrain == mapMagic.AssignedPreviewTerrain)
{
 mapMagic.ClearPreviewTile();
 TerrainTile.OnPreviewAssigned(mapMagic.PreviewData);
}

//assigning new
else
 foreach (var kvp in tilesLut)
 {
  TerrainTile tile = kvp.Value;
  if (tile.main?.terrain == selectedTerrain  &&  mapMagic.AssignedPreviewTerrain != selectedTerrain)
  {
   mapMagic.AssignPreviewTile(tile);
   mapMagic.AssignedPreviewData.isPreview = true;
   TerrainTile.OnPreviewAssigned(mapMagic.AssignedPreviewData);

   UI.RepaintAllWindows();
  }
 }
}
```

```
}



if (selectionMode == SelectionMode.export  ||  selectionMode == SelectionMode.exportCopy)
{
 HashSet<Terrain> allTerrains = new HashSet<Terrain>(pinnedTileTerrains);
 allTerrains.UnionWith(pinnedCustomTerrains);

 HashSet<Terrain> selectedTerrains = TerrainAiming.SelectTerrains(allTerrains, FrameDraw.exportColo
 if (selectedTerrains != null)
  foreach (Terrain terrain in selectedTerrains)
  {
   if (!terrain.gameObject.activeSelf)
    continue;

   //finding coordinate to get proper name
   bool coordFound = false;
   Coord coord = new Coord();
   foreach (var kvp in tilesLut)
    if (kvp.Value.ContainsTerrain(terrain))
    {
     coordFound = true;
     coord = kvp.Key;
    }
   string terrainName = coordFound ? $"Terrain {coord.x}, {coord.z}" : terrain.name;
```

```
    bool asCopy = selectionMode==SelectionMode.exportCopy;
    SaveTerrainData(terrain.terrainData, terrainName, asCopy:asCopy);

    if (saveDraft)
    {
     TerrainTile tile = mapMagic.tiles.FindByTerrain(terrain);
     Terrain draftTerrain = tile.draft?.terrain;
     if (draftTerrain != null)
      SaveTerrainData(draftTerrain.terrainData, terrainName + "_draft", asCopy:asCopy);
    }
  }
}



if (selectionMode == SelectionMode.exportCluster)
{
 HashSet<Terrain> allTerrains = new HashSet<Terrain>(pinnedTileTerrains);
 allTerrains.UnionWith(pinnedCustomTerrains);

 HashSet<Terrain> selectedTerrains = TerrainAiming.SelectTerrains(allTerrains, FrameDraw.exportColo
 if (selectedTerrains != null && selectedTerrains.Count == 1)
  MapMagic.Core.GUI.MapMagicInspector.OnClusterExported?.Invoke(mapMagic, selectedTerrains.Any
  //Clusters.ClusterAssetInspector.CreateViaDialog(mapMagic, selectedTerrains.Any().transform.position
}
```

```
if (selectionMode == SelectionMode.unpin)
{
 HashSet<Terrain> possibleTerrains = new HashSet<Terrain>(pinnedTileTerrains);
 possibleTerrains.UnionWith(pinnedCustomTerrains);

 HashSet<Terrain> selectedTerrains = TerrainAiming.SelectTerrains(possibleTerrains, FrameDraw.unpi
 if (selectedTerrains != null)
 {
  UnityEditor.Undo.RegisterFullObjectHierarchyUndo(mapMagic.gameObject, "MapMagic Unpin Terrains

  foreach (Terrain terrain in selectedTerrains)
  {
   //terrain-to-coord lut (to pick tile terrain coord faster)
   Dictionary<Terrain, Coord> terrainToCoordLut = new Dictionary<Terrain, Coord>();
   foreach (var kvp in tilesLut)
   {
    Coord coord = kvp.Key;
    TerrainTile tile = kvp.Value;
    if (tile.main != null) terrainToCoordLut.Add(tile.main.terrain, coord);
    if (tile.draft != null) terrainToCoordLut.Add(tile.draft.terrain, coord);
   }

   //if it's tile
   if (terrainToCoordLut.ContainsKey(terrain))
   {
    Coord coord = terrainToCoordLut[terrain];
```

```
        mapMagic.tiles.Unpin(coord);
      }

      //if it's custom
      if (pinnedCustomTerrains.Contains(terrain))
      {
        TerrainTile tileComponent = terrain.gameObject.GetComponent<TerrainTile>();
        mapMagic.tiles.UnpinCustom(tileComponent);
        GameObject.DestroyImmediate(tileComponent);
      }
     }

    }
   }

   //redrawing scene
   SceneView.lastActiveSceneView.Repaint();
  }
}


private static bool SaveTerrainData (TerrainData terrainData, string terrainName, bool asCopy=false)
/// Returns false on some error or if user click "cancel"|
{
 TerrainData origTerrainData = terrainData;
 if (asCopy) terrainData = terrainData.Copy();
```

```
string fileName = terrainName.Replace(",", "");
if (asCopy) fileName += " Copy";

if (AssetDatabase.Contains(terrainData))
{
 Debug.Log($"Skipping {terrainName} since it's already has it's data stored in a file.\nUse 'Save Project' t
 return true;
 //can't save asset to the other file
}

string savePath = UnityEditor.EditorUtility.SaveFilePanel(
 $"Save {terrainName}",
 "Assets",
 fileName,
 "asset");

if (savePath == null  ||  savePath.Length==0)
 return false; //clicked cancel

savePath = savePath.Replace(Application.dataPath, "Assets");

//if it's not original terrain data
if (AssetDatabase.LoadAssetAtPath(savePath, typeof(TerrainData)) == origTerrainData)
{
 Debug.Log($"Could not overwrite {terrainName} original data with it's copy. This will make terrain data n
```

```
    return true;
   }

   float[,,] splats = terrainData.GetAlphamaps(0,0,terrainData.alphamapResolution, terrainData.alphamapR
   //backup splats - for some reason they will be removed

   AssetDatabase.DeleteAsset(savePath);  //this will mark terrain data as 'null' in all terrains using it (thus w
   AssetDatabase.CreateAsset(terrainData, savePath);
   AssetDatabase.SaveAssets();

   terrainData.SetAlphamaps(0,0,splats);  //re-assign splats
   AssetDatabase.SaveAssets();

   return true;
  }
 }
}
```

```
ï»¿
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;

using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.GUI;
using Den.Tools.SceneEdit;

using MapMagic.Core;

namespace MapMagic.Terrains.GUI
{
 public static class TerrainAiming
 {
  public static List<Coord> SelectTiles (Vector3 tileSize, bool dotted, Dictionary<Coord,TerrainTile> terrains
  /// Selects tile coordinates via click or selection frame
  /// Returns null if selection was not finally made
  /// Could use only the terrainsLut, but that's not so intuitive
  {
   Select.UpdateFrame();

   List<Coord> framedCoords = GetTilesInFrame(Select.frameRect, tileSize, parent);
```

```
 //displaying frame
 foreach (Coord coord in framedCoords)
  FrameDraw.DrawFrame(coord, tileSize, FrameDraw.pinColor, dotted, terrainsLut, FrameDraw.defaultZO

 //returning selected
 if (Select.justReleased || (!Select.isFrame  &&  Event.current.type==EventType.MouseUp  &&  Event.curr
  return framedCoords;

 return null;
}


public static HashSet<Terrain> SelectTerrains (HashSet<Terrain> possibleTerrains, Color color, bool dott
/// Selects terrains via click or selection frame
{
 Select.UpdateFrame();

 HashSet<Terrain> framedTerrains = GetTerrainsInFrame(Select.frameRect, possibleTerrains);

 //displaying frame
 foreach (Terrain terrain in framedTerrains)
  FrameDraw.DrawTerrainFrame(terrain, color, dotted, FrameDraw.defaultZOffset*2);

 //returning selected
 if (Select.justReleased || (!Select.isFrame  &&  Event.current.type==EventType.MouseUp  &&  Event.curr
```

```
  return framedTerrains;


 return null;
}



private static HashSet<Terrain> GetTerrainsInFrame (Rect screenFrame, HashSet<Terrain> possibleTer
{
 HashSet<Terrain> terrains = new HashSet<Terrain>();

 //if frame is small (or single click) selecting terrains by raycast
 Vector2[] screenFrameCorners = new Vector2[] {
  new Vector2(screenFrame.x, screenFrame.y),
  new Vector2(screenFrame.x, screenFrame.y+screenFrame.size.y),
  new Vector2(screenFrame.x+screenFrame.size.x, screenFrame.y),
  new Vector2(screenFrame.x+screenFrame.size.x, screenFrame.y+screenFrame.size.y) };

 for (int c=0; c<screenFrameCorners.Length; c++)
 {
  Ray worldRay = HandleUtility.GUIPointToWorldRay(screenFrameCorners[c]);
  RaycastHit hit = GetAimedTerrainHit(worldRay, possibleTerrains);
  if (hit.collider == null) continue;
  Terrain terrain = hit.collider.gameObject.GetComponent<Terrain>();
  if (!terrains.Contains(terrain)) terrains.Add(terrain);
 }
```

```
 //selecting terrains by their bounding boxes if the frame is larger than one click
 if (screenFrame.size.x > 10 && screenFrame.size.y > 10)
 {
  foreach (Terrain terrain in possibleTerrains)
  {
   Bounds bounds = new Bounds(terrain.transform.position + terrain.terrainData.size/2, terrain.terrainData

   if (IsBoundingBoxInFrame(bounds, screenFrame)  &&  !terrains.Contains(terrain))
    terrains.Add(terrain);
  }
 }

 //debug
 //List<Terrain> terrainsList = new List<Terrain>(terrains);
 //for (int i=0; i<terrainsList.Count; i++)
 //{
 // Vector3 size = terrainsList[i].terrainData.size;
 // Vector3 pos = terrainsList[i].transform.position;
 // Handles.DrawWireCube(pos+size/2,size);
 //}

 return terrains;
}


private static List<Coord> GetTilesInFrame (Rect screenFrame, Vector3 tileSize, Transform parent=null)
```

```
{
 //transforming frame into 4 world points
 Vector2[] screenFrameCorners = new Vector2[] {
  new Vector2(screenFrame.x, screenFrame.y),
  new Vector2(screenFrame.x, screenFrame.y+screenFrame.size.y),
  new Vector2(screenFrame.x+screenFrame.size.x, screenFrame.y),
  new Vector2(screenFrame.x+screenFrame.size.x, screenFrame.y+screenFrame.size.y) };

 Vector3[] worldFrameCorners = new Vector3[4];
 for (int c=0; c<4; c++)
 {
  Ray worldRay = HandleUtility.GUIPointToWorldRay(screenFrameCorners[c]);
  worldFrameCorners[c] = GetAimPosAtZeroLevel(worldRay);

  if (parent != null)
   worldFrameCorners[c] = parent.InverseTransformPoint(worldFrameCorners[c]);
 }

 //all possible coords rect
 CoordRect allTilesRect = new CoordRect( Coord.Floor(worldFrameCorners[0].Div(tileSize)), new Coord(
 allTilesRect.Encapsulate( Coord.Floor(worldFrameCorners[1].Div(tileSize)) );
 allTilesRect.Encapsulate( Coord.Floor(worldFrameCorners[2].Div(tileSize)) );
 allTilesRect.Encapsulate( Coord.Floor(worldFrameCorners[3].Div(tileSize)) );
 allTilesRect.size.x ++;
 allTilesRect.size.z ++;
```

```
 //intersecting each tile with frame
 List<Coord> tilesInFrame = new List<Coord>();
 Coord min = allTilesRect.Min;
 Coord max = allTilesRect.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   Coord tile = new Coord(x,z);
   if (IsInFrame(tile, tileSize, screenFrame, worldFrameCorners, parent))
    tilesInFrame.Add(tile);
  }

 //debug
 //for (int i=0; i<tilesInFrame.Count; i++)
 //{
 // Coord tile = tilesInFrame[i];
 // Vector3 pos = tile.vector3*tileSize + new Vector3(tileSize/2, 0, tileSize/2);
 // Vector3 size = new Vector3(tileSize, 0, tileSize);
 // Handles.DrawWireCube(pos,size);
 //}

 return tilesInFrame;
}
```

```
private static bool IsInFrame (Coord tileCoord, Vector3 tileSize, Rect screenFrame, Vector3[] worldFrame
{
 //looking if any of the tile corners lies within screen frame in screen space
 Vector3 cor =  tileCoord.vector3.Mul(tileSize);
 if (parent !=null) cor = parent.TransformPoint(cor);
 Vector2 p1 = HandleUtility.WorldToGUIPoint( cor );
 Vector2 p2 = HandleUtility.WorldToGUIPoint( cor + new Vector3(0,0,tileSize.z) );
 Vector2 p3 = HandleUtility.WorldToGUIPoint( cor + new Vector3(tileSize.x,0,0) );
 Vector2 p4 = HandleUtility.WorldToGUIPoint( cor + new Vector3(tileSize.x,0,tileSize.z) );

 if (LineRectIntersection(p1, p2, screenFrame)) return true;
 if (LineRectIntersection(p2, p3, screenFrame)) return true;
 if (LineRectIntersection(p3, p4, screenFrame)) return true;
 if (LineRectIntersection(p4, p1, screenFrame)) return true;

 //looking if any if the screen frame corner rays pass through tile in world space
 Rect worldRect = new Rect(new Vector2(tileCoord.x*tileSize.x, tileCoord.z*tileSize.z), new Vector2(tileSi
 for (int c=0; c<worldFrame.Length; c++)
 {
  if (worldRect.Contains( new Vector2(worldFrame[c].x, worldFrame[c].z) ))
   return true;
 }

 return false;
}
```

```
private static bool LineRectIntersection (Vector2 a, Vector2 b, Rect r)
/// Finds an intersection between the line and rect in 2D
{
 Vector2 min = Vector2.Min(a,b);
 Vector2 max = Vector2.Max(a,b);

 if (r.xMin > max.x || r.xMax < min.x ||
  r.yMin > max.y || r.yMax < min.y)
   return false;

 if (r.xMin < min.x && max.x < r.xMax) return true;
 if (r.yMin < min.y && max.y < r.yMax) return true;

 //line function
 float yAtRectLeft = a.y - (r.xMin - a.x) * ((a.y - b.y) / (b.x - a.x));
 float yAtRectRight = a.y - (r.xMax - a.x) * ((a.y - b.y) / (b.x - a.x));

 if (r.yMax < yAtRectLeft && r.yMax < yAtRectRight) return false;
 if (r.yMin > yAtRectLeft && r.yMin > yAtRectRight) return false;

 return true;
}

public static Vector3 GetAimPosAtZeroLevel (Ray aimRay)
```

```
/// Finds ray intersection with zero level and returns a coord
{
 //aiming coord tile (aim position at zero level)
 aimRay.direction = aimRay.direction.normalized;
 float aimDist = aimRay.origin.y / (-aimRay.direction.y);
 return aimRay.origin + aimRay.direction*aimDist;

}

public static RaycastHit GetAimedTerrainHit (Ray aimRay, HashSet<Terrain> possibleTerrains=null, Hash
/// Finds an aimed terrain from the list of all possible terrains (will not aim the terrain that is not in list)
{
 RaycastHit[] hits = Physics.RaycastAll(aimRay, Mathf.Infinity);
 for (int h=0; h<hits.Length; h++)
 {
  Terrain hitTerrain = hits[h].collider.gameObject.GetComponent<Terrain>();
  if (hitTerrain == null) continue;
  if (possibleTerrains!=null && !possibleTerrains.Contains(hitTerrain)) continue;
  if (ignoredTerrains!=null && ignoredTerrains.Contains(hitTerrain)) continue;

  return hits[h];
 }

 return new RaycastHit();
}
```

```
private static bool IsBoundingBoxInFrame (Bounds bounds, Rect screenFrame)
{
 //if bounding box edges intersect frame
 Vector3 min = bounds.min;
 Vector3 max = bounds.max;

 Vector2 a = HandleUtility.WorldToGUIPoint(min);
 Vector2 b = HandleUtility.WorldToGUIPoint( new Vector3(max.x, min.y, min.z) );
 Vector2 c = HandleUtility.WorldToGUIPoint( new Vector3(max.x, min.y, max.z) );
 Vector2 d = HandleUtility.WorldToGUIPoint( new Vector3(min.x, min.y, max.z) );

 Vector2 A = HandleUtility.WorldToGUIPoint( new Vector3(min.x, max.y, min.z) );
 Vector2 B = HandleUtility.WorldToGUIPoint( new Vector3(max.x, max.y, min.z) );
 Vector2 C = HandleUtility.WorldToGUIPoint( max );
 Vector2 D = HandleUtility.WorldToGUIPoint( new Vector3(min.x, max.y, max.z) );

 if (LineRectIntersection(a, b, screenFrame)) return true;
 if (LineRectIntersection(b, c, screenFrame)) return true;
 if (LineRectIntersection(c, d, screenFrame)) return true;
 if (LineRectIntersection(d, a, screenFrame)) return true;

 if (LineRectIntersection(A, B, screenFrame)) return true;
 if (LineRectIntersection(B, C, screenFrame)) return true;
 if (LineRectIntersection(C, D, screenFrame)) return true;
 if (LineRectIntersection(D, A, screenFrame)) return true;
```

```
    if (LineRectIntersection(a, A, screenFrame)) return true;
    if (LineRectIntersection(b, B, screenFrame)) return true;
    if (LineRectIntersection(c, C, screenFrame)) return true;
    if (LineRectIntersection(d, D, screenFrame)) return true;

    //looking if any if the screen frame corner rays pass through bb in world space
    //TODO, but not actually required - we are checking ray intersection anyways

    return false;
  }
 }
}
```

```
ï»¿
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;

using MapMagic.Nodes;
using MapMagic.Terrains;



namespace MapMagic.Core.GUI
{
 [CustomEditor(typeof(TerrainTile))]
 public class TerrainTileInspector : Editor
 {
  UI ui = new UI();


  /*[RuntimeInitializeOnLoadMethod, UnityEditor.InitializeOnLoadMethod]
  static void Subscribe ()
  {
   MapMagicObject.OnLodSwitched += (TerrainTile t,bool m, bool d) => SceneView.RepaintAll();
  }*/
```

```
//when selected
/*public void OnSceneGUI ()
{

}*/


public override void  OnInspectorGUI ()
{
 ui.Draw(DrawGUI, inInspector:true);
}

static readonly string[] currentLodNames = new string[] { "None", "Draft", "Main" };

public void DrawGUI ()
{
 TerrainTile tile = (TerrainTile)target;

 using (Cell.LinePx(0))
 {
  //Cell.current.disabled = true;

  using (Cell.LineStd) Draw.Field(tile.coord, "Coord");
  using (Cell.LineStd) Draw.Field(tile.distance, "Remoteness");
```

```
 using (Cell.LineStd) Draw.Field(tile.Priority, "Priority");
}

Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref tile.guiDraft, "Draft", isLeft:true))
  if (tile.guiDraft)
  {
   using (Cell.LineStd) Draw.Toggle(tile.draft==null, "Is Null");

   if (tile.draft!=null)
   {
    using (Cell.LineStd) Draw.ObjectField(tile.draft.terrain, "Terrain");
    using (Cell.LineStd) Draw.Toggle(tile.draft.generateStarted, "Generate Started");
    using (Cell.LineStd) Draw.Toggle(tile.draft.generateReady, "Generate Ready");
    using (Cell.LineStd) Draw.Toggle(tile.draft.applyReady, "Apply Ready");
    using (Cell.LineStd) Draw.Toggle(tile.draft.edges!=null && !tile.draft.edges.heightEdges.IsEmpty, "Edg

    if (tile.draft.data != null  &&  tile.mapMagic.graph != null)
     using (Cell.LineStd) Draw.Toggle(tile.draft.data.AllOutputsReady(tile.mapMagic.graph, OutputLevel.D
   }

   using (Cell.LineStd)
    if (Draw.Button("New"))
    {
     tile.draft = new TerrainTile.DetailLevel(tile, isDraft:true);
```

```
      tile.StartGenerate(tile.mapMagic.graph);
     }


    using (Cell.LineStd)
     if (Draw.Button("Remove"))
     {
      tile.draft.Remove();
      tile.draft=null;
      tile.SwitchLod();
     }
   }


Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref tile.guiMain, "Main", isLeft:true))
  if (tile.guiMain)
  {
   using (Cell.LineStd) Draw.Toggle(tile.main==null, "Is Null");


   if (tile.main!=null)
   {
    using (Cell.LineStd) Draw.ObjectField(tile.main.terrain, "Terrain");
    using (Cell.LineStd) Draw.Toggle(tile.main.generateStarted, "Generate Started");
    using (Cell.LineStd) Draw.Toggle(tile.main.generateReady, "Generate Ready");
    using (Cell.LineStd) Draw.Toggle(tile.main.applyReady, "Apply Ready");
```

```
    if (tile.main.data != null  &&  tile.mapMagic.graph != null)
     using (Cell.LineStd) Draw.Toggle(tile.main.data.AllOutputsReady(tile.mapMagic.graph, OutputLevel.D
   }

   using (Cell.LineStd)
    if (Draw.Button("New"))
    {
     tile.main = new TerrainTile.DetailLevel(tile, isDraft:false);
     tile.StartGenerate(tile.mapMagic.graph);
    }

   using (Cell.LineStd)
    if (Draw.Button("Remove"))
    {
     tile.main.Remove();
     tile.main=null;
     tile.SwitchLod();
    }
  }

Cell.EmptyLinePx(4);



Terrain activeTerrain = tile.ActiveTerrain;
int currentLodNum = 0;
if (activeTerrain != null  &&  tile.draft != null  &&  activeTerrain == tile.draft.terrain) currentLodNum = 1;
```

```
if (activeTerrain != null  &&  tile.main != null  &&  activeTerrain == tile.main.terrain) currentLodNum = 2;
using (Cell.LineStd)
{
 Draw.PopupSelector(ref currentLodNum, currentLodNames, "Current Detail");
 if (Cell.current.valChanged)
 {
  switch (currentLodNum)
  {
   case 1: tile.ActiveTerrain = tile.draft.terrain; break;
   case 2: tile.ActiveTerrain = tile.main.terrain; break;
   default: tile.ActiveTerrain = null; break;
  }
 }
}



Cell.EmptyLinePx(4);


using (Cell.LineStd)
{
 bool newPreview = Draw.ToggleLeft(tile.mapMagic.PreviewTile == tile, "Selected for Preview");
 if (Cell.current.valChanged)
 {
  if (newPreview) tile.mapMagic.AssignPreviewTile(tile);
  else tile.mapMagic.ClearPreviewTile();
 }
```

```
    }

    using (Cell.LineStd) Draw.Toggle(tile.Ready, "Ready");

    float generateComplexity = tile.mapMagic.graph.GetGenerateComplexity();
    float applyComplexity = tile.mapMagic.graph.GetApplyComplexity();
    using (Cell.LineStd) Draw.DualLabel("Generate Complexity", generateComplexity.ToString());
    using (Cell.LineStd) Draw.DualLabel("Apply Complexity", applyComplexity.ToString());
    (float progress, float maxProgress) = tile.GetProgress(tile.mapMagic.graph, generateComplexity, applyC
    using (Cell.LineStd) Draw.DualLabel("Progress", progress.ToString() + " of " + maxProgress.ToString());
  }

 }//class

}//namespace
```

```
ï»¿using System;
using System.Collections.Generic;

namespace Den.Tools
{
 //surprisingly Unity's ArrayUtility is an Editor calss
 //so this is it's analog to use in builds

 public static class ArrayTools
 {
  #region Array

   static public void RemoveAt<T> (ref T[] array, int num) { array = RemoveAt(array, num); }
   static public T[] RemoveAt<T> (T[] array, int num)
   {
    if (num >= array.Length || num < 0) num = array.Length-1; //note that negative pos will remove LAST ele

    T[] newArray = new T[array.Length-1];

    if (num!=0) Array.Copy(array, newArray, num);
    if (num!=array.Length) Array.Copy(array, num+1, newArray, num, newArray.Length-num);

    return newArray;
   }

   static public void Remove<T> (ref T[] array, T obj) where T : class  {array = Remove(array, obj); }
```

```
static public T[] Remove<T> (T[] array, T obj) where T : class
{
 int num = Find<T>(array, obj);
 return RemoveAt<T>(array,num);
}

static public void Add<T> (ref T[] array, T element) { array = Add(array, element); }
static public T[] Add<T> (T[] array, T element)
{
 if (array==null || array.Length==0)
  return new T[] { element };

 T[] newArray = new T[array.Length+1];
 Array.Copy(array, newArray, array.Length);

 newArray[array.Length] = element;

 return newArray;
}

static public void Add<T> (ref T[] array, T element1, T element2) { array = Add(array, element1, element2
static public T[] Add<T> (T[] array, T element1, T element2)
//Just adds to elements instead of array
{
 if (array==null || array.Length==0)
  return new T[] { element1, element2 };
```

```
 T[] newArray = new T[array.Length+2];
 Array.Copy(array, newArray, array.Length);

 newArray[array.Length] = element1;
 newArray[array.Length+1] = element2;

 return newArray;
}

static public void Add<T> (ref T[] array, T element1, T element2, T element3) { array = Add(array, elemen
static public T[] Add<T> (T[] array, T element1, T element2, T element3)
//Just adds to elements instead of array
{
 if (array==null || array.Length==0)
  return new T[] { element1, element2, element3 };

 T[] newArray = new T[array.Length+3];
 Array.Copy(array, newArray, array.Length);

 newArray[array.Length] = element1;
 newArray[array.Length+1] = element2;
 newArray[array.Length+2] = element3;

 return newArray;
}
```

```
static public void AddRange<T> (ref T[] array, T[] other) { array = AddRange(array, other); }
static public T[] AddRange<T> (T[] array, T[] other)
{
 if (array==null || array.Length==0)
 {
  T[] newArray = new T[other.Length];
  Array.Copy(other, newArray, other.Length);
  return newArray;
 }

 else
 {
  T[] newArray = new T[array.Length+other.Length];
  Array.Copy(array, newArray, array.Length);
  Array.Copy(other, 0, newArray, array.Length, other.Length);
  return newArray;
 }
}

static public void AddLayer<T> (ref T[,] array, T[,] otherArray, int channel) { array = AddLayer(array, oth
static public T[,] AddLayer<T> (T[,] array, T[,] otherArray, int channel)
{
 int lengthX = array.GetLength(0);
 int lengthZ = array.GetLength(1);
 int numChannels = array.GetLength(2);
```

```
 T[,,] newArray = new T[lengthX,lengthZ,numChannels+1];
 Array.Copy(array, newArray, lengthX*lengthZ*numChannels);
 CopyLayer(otherArray, newArray, channel, numChannels);

 return newArray;
}

static public void Insert<T> (ref T[] array, int pos, T element) { array = Insert(array, pos, element); }
static public void Insert<T> (ref T[] array, int pos, Func<int,T> createElement) { array = Insert(array, pos,
static public T[] Insert<T> (T[] array, int pos, T element)
{
 if (array==null || array.Length==0)
  return new T[] {element};

 if (pos > array.Length || pos < 0) pos = array.Length; //note that negative pos will ADD element

 T[] newArray = new T[array.Length+1];

 if (pos != 0) Array.Copy(array, newArray, pos);
 if (pos != array.Length) Array.Copy(array, pos, newArray, pos+1, array.Length-pos);

 newArray[pos] = element;

 return newArray;
}
```

```
static public void InsertRemoveLast<T> (this T[] array, int pos, T element)
{
 Array.Copy(array, pos, array, pos+1, array.Length-pos-1);
 array[pos] = element;
}

static public T[] InsertRange<T> (T[] array, int after, T[] add)
{
 //if (array==null || array.Length==0) { return add; } //should create a copy anyways

 if (after > array.Length || after<0) after = array.Length;

 T[] newArray = new T[array.Length+add.Length];

 if (after != 0) Array.Copy(array, newArray, after);
 Array.Copy(add, 0, newArray, after, add.Length);
 if (after != array.Length) Array.Copy(array, after, newArray, after+add.Length, array.Length-after);

 return newArray;
}

static public void Resize<T> (ref T[] array, int newSize, Func<int,T> createElement=null) { array = Resize
static public T[] Resize<T> (T[] array, int newSize, Func<int,T> createElement=null)
{
 //if (array.Length == newSize) return array; //should create a copy anyways
```

```
 T[] newArray = new T[newSize];
 Array.Copy(array, newArray, newSize<array.Length? newSize : array.Length);

 if (newSize > array.Length && createElement != null)
 {
  for (int i=array.Length; i<newSize; i++)
   newArray[i] = createElement(i);
 }

 return newArray;
}

static public void Resize<T> (ref T[] array, int newSize, T def) { array = Resize(array, newSize, def); }
static public T[] Resize<T> (T[] array, int newSize, T def)
{
 //if (array.Length == newSize) return array; //should create a copy anyways

 T[] newArray = new T[newSize];
 Array.Copy(array, newArray, newSize<array.Length? newSize : array.Length);

 if (newSize > array.Length)
 {
  for (int i=array.Length; i<newSize; i++)
   newArray[i] = def;
 }
```

```
 return newArray;
}

static public void ResizeLayers<T> (ref T[,,] array, int newSize, T def) { array = ResizeLayers(array, newS
static public T[,,] ResizeLayers<T> (T[,,] array, int newSize, T def)
{
 int oldSize = array.GetLength(2);
 int sizeX = array.GetLength(0);
 int sizeZ = array.GetLength(1);

 T[,,] newArray = new T[array.GetLength(0), array.GetLength(1), newSize];
 //Array.Copy(array, 0, newArray, 0, newSize<oldSize? newArray.Length : array.Length); //could be used

 for (int x=0; x<sizeX; x++)
  for (int z=0; z<sizeZ; z++)
   for (int i=0; i<newSize; i++)
   {
    T val = i<oldSize ? array[x,z,i] : def;
    newArray[x,z,i] = val;
   }

 return newArray;
}

static public void CopyLayer<T> (T[,,] src, T[,,] dst, int srcNum, int dstNum)
```

```
{
 int sizeX = src.GetLength(0);
 int sizeZ = src.GetLength(1);

 for (int x=0; x<sizeX; x++)
  for (int z=0; z<sizeZ; z++)
   dst[x,z,dstNum] = src[x,z,srcNum];
}

static public void Append<T> (ref T[] array, T[] additional) { array = Append(array, additional); }
static public T[] Append<T> (T[] array, T[] additional)
{
 T[] newArray = new T[array.Length+additional.Length];
 for (int i=0; i<array.Length; i++) { newArray[i] = array[i]; }
 for (int i=0; i<additional.Length; i++) { newArray[i+array.Length] = additional[i]; }
 return newArray;
}

static public void Switch<T> (T[] array, int num1, int num2)
{
 if (num1<0 || num1>=array.Length || num2<0 || num2 >=array.Length) return;

 T temp = array[num1];
 array[num1] = array[num2];
 array[num2] = temp;
}
```

```
static public void Switch<T> (T[] array, T obj1, T obj2) where T : class
{
 int num1 = Find<T>(array, obj1);
 int num2 = Find<T>(array, obj2);
 Switch<T>(array, num1, num2);
}

static public void Replace<T> (this T[] array, T obj1, T obj2) where T : IEquatable<T>
{
 for (int i=0; i<array.Length; i++)
  if (array[i].Equals(obj1))
   array[i] = obj2;
}

static public void ReplaceNum<T> (this T[] array, int n1, int n2) where T : IEquatable<T>
{
 T obj1 = array[n1];
 T obj2 = array[n2];

 for (int i=0; i<array.Length; i++)
  if (array[i].Equals(obj1))
   array[i] = obj2;
}

static public void Move<T> (T[] array, int src, int dst)
```

```
{
 T srcVal = array[src];
 if (src < dst) //moving down
 {
  for (int i=src; i<dst; i++)
   array[i] = array[i+1];
 }
 if (src > dst) //moving up
 {
  for (int i=src; i>dst; i--)
   array[i] = array[i-1];
 }
 array[dst] = srcVal;
}



static public T[] Truncated<T> (this T[] src, int length)
{
 T[] dst = new T[length];
 for (int i=0; i<length; i++) dst[i] = src[i];
 return dst;
}

public static bool Equals<T> (T[] a1, T[] a2) where T : class
{
 if (a1.Length != a2.Length) return false;
```

```
 for (int i=0; i<a1.Length; i++)
  if (a1[i] != a2[i]) return false;
 return true;
}

public static bool EqualsEquatable<T> (T[] a1, T[] a2) where T : IEquatable<T>
{
 if (a1.Length != a2.Length) return false;
 for (int i=0; i<a1.Length; i++)
  if (!Equals(a1[i],a2[i])) return false;
 return true;
}

public static bool EqualsVector3 (UnityEngine.Vector3[] a1, UnityEngine.Vector3[] a2, float delta=float.Ep
{
 if (a1==null || a2==null || a1.Length != a2.Length) return false;
 for (int i=0; i<a1.Length; i++)
 {
  float dist = a1[i].x-a2[i].x;
  if (!(dist<delta && -dist<delta)) return false;

  dist = a1[i].y-a2[i].y;
  if (!(dist<delta && -dist<delta)) return false;

  dist = a1[i].z-a2[i].z;
  if (!(dist<delta && -dist<delta)) return false;
```

```
 }
 return true;
}

//public static int Find(this Array array, object obj)
//{
// for (int i=0; i<array.Length; i++)
//  if (array.GetValue(i) == obj) return i;
// return -1;
//}
//when comparing two (object)strings VS says "true", Unity says "false". Maybe there are some other cas



static public int Find<T> (this T[] array, T obj) //where T : class   where T : IEquatable<T>
{
 for (int i=0; i<array.Length; i++)
  if (Equals(array[i],obj)) return i;
 return -1;
}

static public int Find<T> (this T[] array, Predicate<T> func) //where T : class   where T : IEquatable<T>
{
 for (int i=0; i<array.Length; i++)
  if (func(array[i])) return i;
 return -1;
}
```

```
static public T FindMember<T> (this T[] array, Func<T,bool> func) where T : class
{
 for (int i=0; i<array.Length; i++)
  if (func(array[i])) return array[i];
 return null;
}

public static int FindCount<T>(this T[] array, T obj)
{
 int count = 0;
 for (int i=0; i<array.Length; i++)
  if (Equals(array[i],obj)) count++;
 return count;
}

public static List<int> FindAllIndexes<T> (this T[] array, Func<T,bool> func)
/// Returns all instances
{
 List<int> result = new List<int>();
 for (int i=0; i<array.Length; i++)
  if (func(array[i])) result.Add(i);
 return result;
}

public static TT FindMemberOfType<T,TT> (this T[] array)
```

```
{
 for (int i=0; i<array.Length; i++)
  if (array[i] is TT tt) return tt;
 return default;
}

public static T FindSmallest<T> (this T[] array, Func<T,float> func)
{
 if (array.Length == 0) return default(T);
 if (array.Length == 1) return array[0];

 float smallestVal = float.MaxValue;
 int smallestIndex = -1;
 for (int i=0; i<array.Length; i++)
 {
  float val = func(array[i]);
  if (val < smallestVal)
   { smallestVal = val; smallestIndex = i; }
 }

 return array[smallestIndex];
}

public static T FindBiggest<T> (this T[] array, Func<T,float> func)
{
 if (array.Length == 0) return default(T);
```

```
 if (array.Length == 1) return array[0];

 float biggesttVal = float.MinValue;
 int biggestIndex = -1;
 for (int i=0; i<array.Length; i++)
 {
  float val = func(array[i]);
  if (val > biggesttVal)
   { biggesttVal = val; biggestIndex = i; }
 }

 return array[biggestIndex];
}

static public bool Contains<T>(this T[] array, T obj)
{
 if (array == null) return false;  //wierd case
 if (Array.IndexOf(array, obj) >= 0) return true;
 else return false;
}

static public bool Contains<T>(this T[] array, Predicate<T> func)
{
 if (array == null) return false;  //wierd case
 if (Find(array, func) >= 0) return true;
 else return false;
```

```
}

static public bool ContainsNull<T>(this T[] array) where T: class
{
 if (array == null) return false;  //wierd case
 if (Array.IndexOf(array, null) >= 0) return true;
 else return false;
}

static public bool AllNull<T>(this T[] array) where T: class
{
 if (array == null) return false;  //wierd case
 for (int i=0; i<array.Length; i++)
  if (array[i]!=null) return false;
 return true;
}


static public T Any<T> (this T[] array) where T : class
{
 for (int i=0; i<array.Length; i++)
  if (array[i]!=null) return array[i];
 return null;
}

static public int Max (this int[] array)
```

```
{
 int max = int.MinValue;
 for (int i=0; i<array.Length; i++)
  if (array[i] > max) max = array[i];
 return max;
}

static public bool Empty<T> (this T[] array) where T : class
{
 for (int i=0; i<array.Length; i++)
  if (array[i]!=null) return false;
 return true;
}

static public void RemoveAll<T> (ref T[] array, T obj) where T : class  {array = RemoveAll(array, obj); }
static public T[] RemoveAll<T> (T[] array, T obj) where T : class
/// Removes all the occurrences of obj in array
{
 bool[] remove = new bool[array.Length];
 int removeCount = 0;

 for (int i=0; i<array.Length; i++)
  if (Equals(array[i],obj)) { removeCount++; remove[i] = true; }

 T[] newArr = new T[array.Length-removeCount];
```

```
 int counter = 0;
 for (int i=0; i<array.Length; i++)
 {
  if (remove[i]) continue;

  newArr[counter] = array[i];
  counter++;
 }

 return newArr;
}

static public void RemoveAllFunc<T> (ref T[] array,  Func<T,bool> func) where T : class  {array = Remov
static public T[] RemoveAllFunc<T> (T[] array, Func<T,bool> func) where T : class
/// Finds all elementsz with a callback and removes them
{
 bool[] remove = new bool[array.Length];
 int removeCount = 0;

 for (int i=0; i<array.Length; i++)
  if (func(array[i])) { removeCount++; remove[i] = true; }

 T[] newArr = new T[array.Length-removeCount];

 int counter = 0;
 for (int i=0; i<array.Length; i++)
```

```
 {
  if (remove[i]) continue;

  newArr[counter] = array[i];
  counter++;
 }

 return newArr;
}

static public T[] RemoveNulls<T>(T[] array) where T: class
{
 if (array == null) return array;
 if (Array.IndexOf(array, null) >= 0) return array;

 int numNulls = 0;
 for (int i=0; i<array.Length; i++)
  if (array[i]==null) numNulls++;

 T[] newArr = new T[array.Length - numNulls];

 int c = 0;
 for (int i=0; i<array.Length; i++)
 {
  if (array[i]==null) continue;
```

```
  newArr[c] = array[i];
  c++;
 }

 return newArr;
}

static public bool MatchExactly<T> (T[] arr1, T[] arr2)
/// Returns true if arr1 and arr2 have same elements, in same order
{
 if (arr1.Length != arr2.Length) return false;

 for (int i=0; i<arr1.Length; i++)
  if (!Equals(arr1[i], arr2[i])) return false;

 return true;
}

static public bool MatchElements<T> (T[] arr1, T[] arr2)
/// Returns true if arr1 and arr2 have same elements, but in different order
/// This works even if elements null or duplicating
{
 if (arr1.Length != arr2.Length) return false;

 bool[] matchList = new bool[arr1.Length];
```

```
 for (int i=0; i<arr1.Length; i++)
  for (int j=0; j<arr2.Length; j++)
  {
   if (matchList[j]) continue;
   if (Equals(arr1[i], arr2[j])) { matchList[j] = true; break; }
  }

 for (int i=0; i<matchList.Length; i++)
  if (!matchList[i]) return false;
 return true;
}



static public void AddIfNotContains<T> (ref T[] array, T element) { array = AddIfNotContains(array, eleme
static public T[] AddIfNotContains<T> (T[] array, T element)
/// Adds new element only if it was not found in array
{
 if (array.FindCount(element) == 0) return Add(array, element);
 else return array;
}

static public void AddRangeIfNotContains<T> (ref T[] array, IEnumerable<T> add) { array = AddRangeIfN
static public T[] AddRangeIfNotContains<T> (T[] array, IEnumerable<T> add)
/// Adds each of new element only if it was not found in array
/// Does not guarantee uniqueness: if add contains 2 same elements, they will be added both
{
```

```
 List<T> elementsToAdd = new List<T>();
 foreach (T element in add)
  if (!array.Contains(element)) elementsToAdd.Add(element);

 if (elementsToAdd.Count == 0)
  return array;

 else
  return AddRange(array, elementsToAdd.ToArray());
}



static public bool IsEmpty<T> (T[] array) where T: class
{
 for (int i=0; i<array.Length; i++)
  if (array[i] != null) return false;
 return true;
}

static public void Rewrite<T> (List<T> src, ref T[] dst)
/// Fills an array with list values. Will not re-create array if it's count has not changed
{
 if (dst.Length != src.Count) dst = new T[src.Count];
 for (int i=0; i<dst.Length; i++)
  dst[i] = src[i];
}
```

```
static public void Rewrite<T1,T2> (List<T1> src, ref T2[] dst, Func<T1,T2> fn)
/// Same as rewrite, but calls fn each time
{
 if (dst.Length != src.Count) dst = new T2[src.Count];
 for (int i=0; i<dst.Length; i++)
  dst[i] = fn(src[i]);
}

static public T[] Process<T> (this T[] arr, Func<int,T> func)
/// Calls func on each arry member. Linq sux
{
 for (int i=0; i<arr.Length; i++)
  arr[i] = func(i);
 return arr;
}

static public TDst[] Select<TSrc, TDst> (this TSrc[] src, Func<TSrc,TDst> fn)
{
 TDst[] dst = new TDst[src.Length];
 for (int a=0; a<src.Length; a++)
  dst[a] = fn(src[a]);
 return dst;
}

static public T[] Convert<T,Y> (Y[] src)
```

```
{
 T[] result = new T[src.Length];
 for (int i=0; i<src.Length; i++) result[i] = (T)(object)(src[i]);
 return result;
}

static public T[] Convert<T,Y> (ICollection<Y> src)
{
 Y[] tmpArr = new Y[src.Count];
 src.CopyTo(tmpArr, 0);
 return Convert<T,Y>(tmpArr);
}

static public void FillNulls<T> (this T[] arr, Func<T> func) where T: class
{
 for (int a=0; a<arr.Length; a++)
  if (arr[a] == null) arr[a] = func();
}

static public void Fill<T> (this T[] arr, T val)
{
 for (int a=0; a<arr.Length; a++)
  arr[a] = val;
}

static public string ToStringMemberwise(this Array array, string separator=", ")
```

```
{
 string s = "";
 if (array.Length == 0) return s;
 for (int i=0; i<array.Length; i++)
 {
  object val = array.GetValue(i);
  s += val!=null? val.ToString() : "";
  if (i != array.Length-1) s += separator;
 }
 return s;
}


static public void RandomMix<T> (this T[] array, int iterations=2)
{
 for (int it=0; it<iterations; it++)
 {
  for (int a=0; a<array.Length; a++)
  {
   int b = UnityEngine.Random.Range(0, array.Length);
   if (a==b) continue;

   T tmp = array[b];
   array[b] = array[a];
   array[a] = tmp;
  }
 }
```

```
}

static public T Last<T> (this T[] array) where T: class =>
 array.Length != 0 ? array[array.Length-1] : null;

static public T[] Copy<T> (this T[] array)
{
 T[] newArr = new T[array.Length];
 Array.Copy(array, newArr, array.Length);
 return newArr;
}

static public T[][] CopyJagged<T> (this T[][] array)
{
 T[][] newArr = new T[array.Length][];
 for (int i=0; i<array.Length; i++)
  newArr[i] = Copy(array[i]);
 return newArr;
}

static public TA[] Where<TC,TA> (this ICollection<TC> collection, Func<TC,TA> fn)
{
 TA[] array = new TA[collection.Count];

 int i = 0;
 foreach (TC elem in collection)
```

```
 {
  array[i] = fn(elem);
  i++;
 }

 return array;
}

#endregion

#region Array Sorting

 static public void QSort (float[] array) { QSort(array, 0, array.Length-1); }
 static public void QSort (float[] array, int l, int r)
 {
  float mid = array[l + (r-l) / 2]; //(l+r)/2
  int i = l;
  int j = r;

  while (i <= j)
  {
   while (array[i] < mid) i++;
   while (array[j] > mid) j--;
   if (i <= j)
   {
    float temp = array[i];
```

```
   array[i] = array[j];
   array[j] = temp;

   i++; j--;
  }
 }
 if (i < r) QSort(array, i, r);
 if (l < j) QSort(array, l, j);
}

static public void QSort<T> (T[] array, float[] reference) { QSort(array, reference, 0, reference.Length-1); }
static public void QSort<T> (T[] array, float[] reference, int l, int r)
{
 float mid = reference[l + (r-l) / 2]; //(l+r)/2
 int i = l;
 int j = r;

 while (i <= j)
 {
  while (reference[i] < mid) i++;
  while (reference[j] > mid) j--;
  if (i <= j)
  {
   float temp = reference[i];
   reference[i] = reference[j];
   reference[j] = temp;
```

```
   T tempT = array[i];
   array[i] = array[j];
   array[j] = tempT;

   i++; j--;
  }
 }
 if (i < r) QSort(array, reference, i, r);
 if (l < j) QSort(array, reference, l, j);
}

static public void QSort<T> (List<T> list, float[] reference) { QSort(list, reference, 0, reference.Length-1); }
static public void QSort<T> (List<T> list, float[] reference, int l, int r)
{
 float mid = reference[l + (r-l) / 2]; //(l+r)/2
 int i = l;
 int j = r;

 while (i <= j)
 {
  while (reference[i] < mid) i++;
  while (reference[j] > mid) j--;
  if (i <= j)
  {
   float temp = reference[i];
```

```
   reference[i] = reference[j];
   reference[j] = temp;

   T tempT = list[i];
   list[i] = list[j];
   list[j] = tempT;

   i++; j--;
  }
 }
 if (i < r) QSort(list, reference, i, r);
 if (l < j) QSort(list, reference, l, j);
}

static public int[] Order (int[] array, int[] order=null, int max=0, int steps=1000000, int[] stepsArray=null) //r
{
 if (max==0) max=array.Length;
 if (stepsArray==null) stepsArray = new int[steps+1];
 else steps = stepsArray.Length-1;

 //creating starts array
 int[] starts = new int[steps+1];
 for (int i=0; i<max; i++) starts[ array[i] ]++;

 //making starts absolute
 int prev = 0;
```

```
    for (int i=0; i<starts.Length; i++)
     { starts[i] += prev; prev = starts[i]; }

    //shifting starts
    for (int i=starts.Length-1; i>0; i--)
     { starts[i] = starts[i-1]; }
    starts[0] = 0;

    //using magic to compile order
    if (order==null) order = new int[max];
    for (int i=0; i<max; i++)
    {
     int h = array[i]; //aka height
     int num = starts[h];
     order[num] = i;
     starts[h]++;
    }
    return order;
   }


  #endregion
 }

}
```

```
ï»¿using UnityEngine;
using System;

namespace Den.Tools
{



 public class AutoSubscribe
 {
  public static void Subscribe ()
  {

  }
 }

 public interface IAutoSubscriber
 {
  //void Subscribe ();
 }



}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;

namespace Den.Tools
{
 [Serializable]
 public class AnimCurve
 {
  [Serializable]
  public struct Point
  {
   public float time;
   public float val;
   public float inTangent;
   public float outTangent;

   public Point (Keyframe key) { time=key.time; val=key.value; inTangent=key.inTangent; outTangent=key.o
   public Point (float time, float val, float inTangent, float outTangent)
    { this.time=time; this.val=val; this.inTangent=inTangent; this.outTangent=outTangent; }
  }

  public Point[] points = new Point[0];

  public AnimCurve () { }
  public AnimCurve (AnimationCurve animCurve) { ReadAnimCurve(animCurve); }
```

```
public AnimCurve (Point[] points) { this.points = points; }


public void ReadAnimCurve (AnimationCurve animCurve)
{
 if (points.Length != animCurve.keys.Length)
  points = new Point[animCurve.keys.Length];

 for (int p=0; p<points.Length; p++)
  points[p] = new Point(animCurve.keys[p]);
}

public void WriteToAnimCurve (AnimationCurve animCurve)
{
 if (points.Length != animCurve.keys.Length)
  animCurve.keys = new Keyframe[points.Length];

 for (int p=0; p<points.Length; p++)
 {
  Point point = points[p];
  animCurve.keys[p] = new Keyframe(point.time, point.val, point.inTangent, point.outTangent);
 }
}

public float Evaluate (float time, int segment=-1)
{
 if (time <= points[0].time) return points[0].val;
```

```
if (time >= points[points.Length-1].time) return points[points.Length-1].val;

for (int p=0; p<points.Length-1; p++)
{
 if (time > points[p].time && time <= points[p+1].time)
 {
  Point prev = points[p];
  Point next = points[p+1];

  float delta = next.time - prev.time;
  float relativeTime = (time - prev.time) / delta;

  float timeSq = relativeTime * relativeTime;
  float timeCu = timeSq * relativeTime;

  float a = 2*timeCu - 3*timeSq + 1;
  float b = timeCu - 2*timeSq + relativeTime;
  float c = timeCu - timeSq;
  float d = -2*timeCu + 3*timeSq;

  return a*prev.val + b*prev.outTangent*delta + c*next.inTangent*delta + d*next.val;
 }
 else continue;
}

return 0;
```

```
 }
}

[Serializable]
public class Curve
{
 [Serializable]
 public class Node
 {
  public Vector2 pos;

  public float inTangent;
  public float outTangent;

  public bool linear;

  public Node () { }
  public Node (Vector2 pos) { this.pos=pos; inTangent=0; outTangent=0; linear = false; }
  public Node (Vector2 pos, float inTangent, float outTangent)
   { this.pos=pos; this.inTangent=inTangent; this.outTangent=outTangent; linear = false; }
  public Node (Node src) { pos=src.pos; inTangent=src.inTangent; outTangent=src.outTangent; linear=src.
 }

 public Node[] points = new Node[0];
 public float[] lut = new float[256];
```

```
public Curve () { }
public Curve (AnimationCurve animCurve) { ReadAnimCurve(animCurve); Refresh(); }
public Curve (Vector2 pos1, Vector2 pos2) { points = new Node[] { new Node(pos1), new Node(pos2) }; R
public Curve (Node[] points) { this.points = points; }
public Curve (Curve src)
{
 points = new Node[src.points.Length];
 Copy(src,this);
}

public static void Copy (Curve src, Curve dst)
{
 if (dst.points.Length != src.points.Length)
  dst.points = new Node[src.points.Length];

 for (int i=0; i<src.points.Length; i++)
  dst.points[i] = new Node(src.points[i]);
}

public Vector2[] GetPositions ()
{
 Vector2[] positions = new Vector2[points.Length];

 for (int i=0; i<points.Length; i++)
  positions[i] = points[i].pos;
```

```
 return positions;
}

public void ReadAnimCurve (AnimationCurve animCurve)
{
 if (points.Length != animCurve.keys.Length)
  points = new Node[animCurve.keys.Length];

 for (int p=0; p<points.Length; p++)
  points[p] = new Node( new Vector2(animCurve.keys[p].time,animCurve.keys[p].value), animCurve.keys[
}

public void WriteToAnimCurve (AnimationCurve animCurve)
{
 if (points.Length != animCurve.keys.Length)
  animCurve.keys = new Keyframe[points.Length];

 for (int p=0; p<points.Length; p++)
 {
  Node point = points[p];
  animCurve.keys[p] = new Keyframe(point.pos.x, point.pos.y, point.inTangent, point.outTangent);
 }
}

public float EvaluatePrecise (float time)
{
```

```
 if (time <= points[0].pos.x) return points[0].pos.y;
 if (time >= points[points.Length-1].pos.x) return points[points.Length-1].pos.y;

 for (int p=0; p<points.Length-1; p++)
 {
  if (time > points[p].pos.x && time <= points[p+1].pos.x)
  {
   float delta = points[p+1].pos.x - points[p].pos.x;
   float relativeTime = (time - points[p].pos.x) / delta;

   return EvaluatePrecise(points[p], points[p+1], relativeTime);
  }
 }

 return 0;
}


public static float EvaluatePrecise (Node prev, Node next, float x)
{
 if (prev.linear && next.linear)
  return prev.pos.y * (1-x)  +  next.pos.y * x;

 float delta = next.pos.x - prev.pos.x;
```

```
float pp = 3*x*x - 2*x*x*x;
float posInterpolation = prev.pos.y * (1-pp)  +  next.pos.y * pp;

float pt = x*(1-x) * x;
float it = x*(1-x) * (1-x);  //x*(2*x - x*x -1);

float tanInterpolation = prev.outTangent*it  +  (-next.inTangent)*pt; //unity outTangent is inverted and mat
posInterpolation += tanInterpolation*delta;


//bias values to set the changeable length tangents
/*float nBias = next.linear ? 0 : 1;
float pBias = prev.linear ? 0 : 1;

float pp = (x*x*nBias) * (1-x) +  (1-i*i*pBias) * x;
float posInterpolation = prev.pos.y * (1-pp)  +  next.pos.y * pp;

float pt = x*x*x - x*x;
float it = i*i*i - i*i;
float tanInterpolation = (-prev.outTangent)*it*pBias  +  next.inTangent*pt*nBias; //unity outTangent is inve
posInterpolation += tanInterpolation*delta;*/


/*float pPosFactor = prev.linear ? 1  :  1 + x - 2*x*x;
float nPosFactor = next.linear ? 1  :  3*x - 2*x*x;
```

```
float pTanFactor = prev.linear ? 0  :  x*(1-x);

float nTanFactor = next.linear ? 0  :  x*(1-x);


float prevVal = prev.linear ?

 prev.pos.y :

 prev.pos.y * (1 + x - 2*x*x)  +  prev.outTangent*delta * x*(1-x);


float nextVal = next.linear ?

 next.pos.y :

 next.pos.y * (3*x - 2*x*x)  +  (-next.inTangent)*delta * x*(1-x);


return prevVal*(1-x) + nextVal*x;*/



/*if (prev.linear)

{

 float linInterpolation = prev.pos.y*(1-x)  +  next.pos.y*x;

 posInterpolation = linInterpolation*(1-x) + posInterpolation*x;

}

else if (next.linear)

{

 float linInterpolation = prev.pos.y*(1-x)  +  next.pos.y*x;

 posInterpolation = posInterpolation*(1-x) + linInterpolation*x;

}*/
```

```
 return posInterpolation;
}



public void UpdateLut ()
/// Fills the lut array with baked curve evaluations
{
 for (int i=0; i<lut.Length; i++)
  lut[i] = EvaluatePrecise(1f*i / (lut.Length-1));
  //TODO: can speed this up
}



public float EvaluateLuted (float input)
/// Using the baked evaluations to quickly get value
{
 float step = 1f / (lut.Length-1);

 int prevNum = (int)(input/step);
 int nextNum = prevNum+1;

 float prevX = prevNum * step;
 float percent = (input-prevX) / step;

 float prevY = lut[prevNum];
 float nextY = lut[nextNum];
```

```
 return prevY*(1-percent) + nextY*percent;
}



public void ResetNodeTangents (int num)
{
 Vector2 pos = points[num].pos;
 float tan = 0;

 Vector2 prevPos = points[num-1].pos;
 Vector2 nextPos = points[num+1].pos;

 float prevTan =  (pos.y - prevPos.y) / (pos.x - prevPos.x);
 float nextTan =  (pos.y - nextPos.y) / (pos.x - nextPos.x);

 float p = (pos.x - prevPos.x) / (nextPos.x - prevPos.x);

 p = (pos-prevPos).magnitude / ((pos-nextPos).magnitude + (pos-prevPos).magnitude);

 p = 2*p*p*p - 3*p*p  +  2*p;

 tan = prevTan*(1-p) + nextTan*p;

 //photoshop-style tangents
 //float avgTan = (nextPos.y - prevPos.y) / (nextPos.x - prevPos.x);
 //tan = avgTan;
```

```
 points[num].outTangent = tan;
 points[num].inTangent = tan; //in Unity animcurve in and out tangents are not inverted
}


public void ResetFirstTangent ()
{
 //should be done after all of the other tangents set up
 if (points.Length == 2) { points[0].outTangent = 1; return; }

 Vector2 pos = points[0].pos;
 Vector2 nPos = points[1].pos;

 float delta = nPos.x - pos.x;
 Vector2 nTanDir = new Vector2(1, points[1].inTangent);
 Vector2 aimPos = nPos - nTanDir*delta/2;
 float tan = (pos.y - aimPos.y) / (pos.x - aimPos.x);  //next tan only

 points[0].outTangent = tan;
}

public void ResetLastTangent ()
{
 if (points.Length == 2) { points[points.Length-1].inTangent = 1; return; }
```

```
 Vector2 pos = points[points.Length-1].pos;
 Vector2 pPos = points[points.Length-2].pos;

 float delta = pPos.x - pos.x;
 Vector2 nTanDir = new Vector2(1, points[1].inTangent);
 Vector2 aimPos = pPos - nTanDir*delta/2;
 float tan = (aimPos.y-pos.y) / (aimPos.x - pos.x);  //prev tan only

 points[points.Length-1].inTangent = tan;
}

public void Refresh (bool updateLut = true)
{
 points[0].linear = true;
 points[points.Length-1].linear = true;

 for (int i=1; i<points.Length-1; i++)
  ResetNodeTangents(i);

 ResetFirstTangent();
 ResetLastTangent();
 //ResetBeginEndTangents();

 if (updateLut)
  UpdateLut();
}
```

```
}



public class CurveBeizer
{
 public class Node
 {
  public Vector2 pos;
  public Vector2 inTangent;
  public Vector2 outTangent;

  public enum TangentMode { Linear, Smooth, Correlated, Manual }
  public TangentMode tangentMode = TangentMode.Smooth;

  public Node () { }
  public Node (Vector2 pos) { this.pos = pos; }
  public Node (Vector2 pos, Vector2 inTangent, Vector2 outTangent)
   { this.pos = pos; this.inTangent = inTangent; this.outTangent = outTangent; }

  public static Vector2 GetPoint (Node start, Node end, float p)
  {
   float ip = 1f-p;
   return ip*ip*ip*start.pos + 3*p*ip*ip*(start.pos+start.outTangent) + 3*p*p*ip*(end.pos+end.inTangent) + p
  }
 }
```

```
public Node[] points = new Node[0];
public float[] lut = new float[256]; //prepared array to get Y value quickly. Note that conatins last point, so u

public CurveBeizer () { }
public CurveBeizer (Vector2 pos1, Vector2 pos2) { points = new Node[] { new Node(pos1), new Node(pos
public CurveBeizer (Node[] points) { this.points = points; }

/*public float Evaluate (float time, int segment=-1)
/// returns spline Y coordinate using input X
{
 if (time <= points[0].pos.x) return points[0].pos.y;
 if (time >= points[points.Length-1].pos.x) return points[points.Length-1].pos.y;

 for (int i=0; i<points.Length-1; i++)
 {
  if (time > points[i].pos.x && time <= points[i+1].pos.x)
  {
   //segment nodes
   Node start = points[i];
   Node end = points[i+1];

   //x percent
   float delta = end.pos.x - start.pos.x;
   float p = (time - start.pos.x) / delta;
   float ip = 1f-p;
```

```
   //evaluating
   Vector2 pos = ip*ip*ip*start.pos + 3*p*ip*ip*(start.pos+start.outTangent) + 3*p*p*ip*(end.pos+end.inTan
   return pos.y;
  }
  else continue;
 }

 return 0;
}*/

public float Evaluate (float input)
{
 float step = 1f / (lut.Length-1);

 int prevNum = (int)(input/step);
 int nextNum = prevNum+1;

 float prevX = prevNum * step;
 float percent = (input-prevX) / step;

 float prevY = lut[prevNum];
 float nextY = lut[nextNum];
 return prevY*(1-percent) + nextY*percent;
}
```

```
/*public Vector2[][] TestLut (int steps=10)
{
 //pass 1: evaluating beizer curves
 Vector2[][] perSegmentPoses = new Vector2[points.Length][];

 int stepsPerSegment = steps / points.Length + 3;
 if (stepsPerSegment > steps) stepsPerSegment = steps; //if only one segment

 for (int s=0; s<points.Length-1; s++)
 {
  Vector2[] poses = new Vector2[stepsPerSegment];
  perSegmentPoses[s] = poses;

  Node start = points[s];
  Node end = points[s+1];

  for (int i=0; i<stepsPerSegment; i++)
  {
   float p = 1f*i / (stepsPerSegment-1);
   float ip = 1f-p;
   poses[i] = ip*ip*ip*start.pos + 3*p*ip*ip*(start.pos+start.outTangent) + 3*p*p*ip*(end.pos+end.inTangent
  }
 }

 return perSegmentPoses;
```

```
}*/



public float[] GenerateLut (int steps=256)
{
 float[] lut = new float[steps];
 GenerateLut(lut);
 return lut;
}

public float[] GenerateLut () { GenerateLut(lut); return lut; }

public float[] GenerateLut (float[] lut)
{
 int steps = lut.Length;

 //pass 1: evaluating beizer curves
 Vector2[][] perSegmentPoses = new Vector2[points.Length][];

 int stepsPerSegment = steps / points.Length + 3;
 if (stepsPerSegment > steps) stepsPerSegment = steps; //if only one segment

 for (int s=0; s<points.Length-1; s++)
 {
  Vector2[] poses = new Vector2[stepsPerSegment];
  perSegmentPoses[s] = poses;
```

```
 Node start = points[s];
 Node end = points[s+1];

 for (int i=0; i<stepsPerSegment; i++)
 {
  float p = 1f*i / (stepsPerSegment-1);
  float ip = 1f-p;
  poses[i] = ip*ip*ip*start.pos + 3*p*ip*ip*(start.pos+start.outTangent) + 3*p*p*ip*(end.pos+end.inTangent
  //IDEA: replace with float operations to speed up
 }
}

//pass 2: interpolating poses depending on X position
for (int i=0; i<steps-1; i++)
{
 float xPos = 1f*i / (steps-1);

 //finding proper segment
 int segNum = 0;
 for (int s=0; s<points.Length-1; s++)
 {
  if (xPos > points[s].pos.x && xPos <= points[s+1].pos.x)
   { segNum = s; break; }
 }
```

```
 //finding proper sub-segment (pos) number
 Vector2[] poses = perSegmentPoses[segNum];
 int num = 0;
 for (int p=0; p<poses.Length-1; p++)
 {
  if (xPos > poses[p].x && xPos <= poses[p+1].x)
   { num = p; break; }
 }

 //interpolating pos
 float pStart = poses[num].x;
 float pEnd = poses[num+1].x;
 float pRange = pEnd - pStart;
 float percent = (xPos - pStart) / pRange;
 float val = poses[num].y*(1-percent) + poses[num+1].y*percent;

 //out-of-range
 if (xPos < points[0].pos.x) val = points[0].pos.y;
 if (xPos > points[points.Length-1].pos.x) val = points[points.Length-1].pos.y;

 lut[i] = val;
}

//setting the final lut val to the last position
lut[lut.Length-1] = points[points.Length-1].pos.y;
```

```
 return lut;
}



public void RefreshNodeTangents (int num)
{
 Node node = points[num];

 Vector2 prevPos = num!=0 ? points[num-1].pos : points[num].pos;
 Vector2 nextPos = num!=points.Length-1 ? points[num+1].pos : points[num].pos;
 Vector2 pos = points[num].pos;

 //start / end
 if (num == 0 || num == points.Length-1)
 {
  node.inTangent = new Vector2(0,0);
  node.outTangent = new Vector2(0,0);
 }

 //linear
 if (node.tangentMode == Node.TangentMode.Linear)
 {
  node.inTangent = (prevPos*0.333f + node.pos*0.667f) - node.pos;
  node.outTangent = (nextPos*0.333f + node.pos*0.667f) - node.pos;
 }
```

```
 //auto
 if (node.tangentMode == Node.TangentMode.Smooth)
 {
  Vector2 prevNormalized = (prevPos-pos).normalized + pos;
  Vector2 nextNormalized = (nextPos-pos).normalized + pos;

  Vector2 startDirNorm = (nextNormalized - prevNormalized).normalized;
  Vector2 endDirNorm = (prevNormalized - nextNormalized).normalized;

  node.outTangent = startDirNorm * (nextPos.x-pos.x)/2; //( (pos-nextPos).magnitude * 0.333f );
  node.inTangent = endDirNorm * (pos.x-prevPos.x)/2; //( (pos-prevPos).magnitude * 0.333f );
 }
}

public void Refresh ()
{
 for (int i=0; i<points.Length; i++)
  RefreshNodeTangents(i);

 //lut
 GenerateLut();
}

public void ReadAnimCurve (AnimationCurve animCurve)
{
```

```
    if (points.Length != animCurve.keys.Length)
      points = new Node[animCurve.keys.Length];

    for (int p=0; p<points.Length; p++)
      points[p] = new Node(new Vector2(animCurve.keys[p].time, animCurve.keys[p].value), new Vector2(-0.1

    Refresh();
   }


  }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections.Generic;

namespace Den.Tools
{
 public static class DebugGizmos
 {
  public class GizmoGroup
  {
   public bool enabled = true;
   public List<IGizmo> gizmos = new List<IGizmo>();

   public bool colorOverride;
   public Color color;
  }

  static public Dictionary<string, GizmoGroup> allGizmos = new Dictionary<string, GizmoGroup>();

  #region Gizmos
   public interface IGizmo
   {
    void Draw ();
    Color Color { get; set; }
   }
```

```
struct LineGizmo : IGizmo
{
 public Vector3 from;
 public Vector3 to;
 public Color Color { get; set; }

 public void Draw ()
 {
  #if UNITY_EDITOR
  UnityEditor.Handles.DrawLine(from, to);
  #endif
 }
}

struct PolyLineGizmo : IGizmo
{
 public Vector3[] points;
 public Color Color { get; set; }

 public void Draw ()
 {
  #if UNITY_EDITOR
  UnityEditor.Handles.DrawPolyLine(points);
  #endif
 }
}
```

```
struct SphereGizmo : IGizmo
{
 public Vector3 pos;
 public float radius;
 public Color Color { get; set; }

 public void Draw ()
 {
  #if UNITY_EDITOR
  UnityEditor.Handles.SphereHandleCap(0, pos, Quaternion.identity, radius, EventType.Repaint);
  #endif
 }
}

struct RectGizmo : IGizmo
{
 public Vector2D pos;
 public Vector2D size;
 public Color Color { get; set; }

 public void Draw ()
 {
  #if UNITY_EDITOR
  UnityEditor.Handles.DrawLine((Vector3)pos, (Vector3)pos + new Vector3(size.x,0,0));
  UnityEditor.Handles.DrawLine((Vector3)pos + new Vector3(size.x,0,0), (Vector3)(pos+size));
```

```
  UnityEditor.Handles.DrawLine((Vector3)(pos+size), (Vector3)pos + new Vector3(0,0,size.z));
  UnityEditor.Handles.DrawLine((Vector3)pos+new Vector3(0,0,size.z), (Vector3)pos);
  #endif
 }
}

struct GridGizmo : IGizmo
{
 public Vector3 pos;
 public Vector3 size;
 public int cellsX;
 public int cellsZ;
 public Color Color { get; set; }

 public void Draw ()
 {
  #if UNITY_EDITOR
  float cellSizeX = size.x / cellsX;
  float cellSizeZ = size.z / cellsZ;

  for (int x=0; x<=cellsX; x++)
   UnityEditor.Handles.DrawLine(
    pos + new Vector3(x*cellSizeX, 0, 0),
    pos + new Vector3(x*cellSizeX, 0, size.z));

  for (int z=0; z<=cellsZ; z++)
```

```
    UnityEditor.Handles.DrawLine(
      pos + new Vector3(0, 0, z*cellSizeX),
      pos + new Vector3(size.x, 0, z*cellSizeX));
   #endif
  }
 }

struct DotGizmo : IGizmo
{
  public Vector3 pos;
  public float size;
  public Color Color { get; set; }

  public void Draw ()
  {
   #if UNITY_EDITOR
   float size3d = UnityEditor.HandleUtility.GetHandleSize(pos) * size / 100;
   UnityEditor.Handles.DotHandleCap(0, pos, Quaternion.identity, size3d, EventType.Repaint);
   #endif
  }
 }

struct CircleGizmo : IGizmo
{
  public Vector3 pos;
  public float radius;
```

```
 public Color Color { get; set; }


 public void Draw ()
 {
  #if UNITY_EDITOR
  Camera cam = UnityEditor.SceneView.currentDrawingSceneView.camera;
  Vector3 camPos = cam.transform.position;
  Vector3 camUp = cam.transform.up;
  Vector3 camFront = cam.orthographic ? cam.transform.forward : (camPos-pos).normalized;
  UnityEditor.Handles.DrawWireArc(pos, camFront, camUp, 360, radius);
  #endif
 }
}

struct MultiLineGizmo : IGizmo
{
 public (Vector3,Vector3)[] lines;
 public Color Color { get; set; }


 public void Draw ()
 {
  #if UNITY_EDITOR
  for (int i=0; i<lines.Length; i++)
   UnityEditor.Handles.DrawLine(lines[i].Item1, lines[i].Item2);
  #endif
 }
```

```
 }

 struct LabelGizmo : IGizmo
 {
  public Vector3 pos;
  public string label;
  public Color Color { get; set; }

  #if UNITY_EDITOR
  private static GUIStyle style = new GUIStyle(UnityEditor.EditorStyles.label);
  #endif

  public void Draw ()
  {
   #if UNITY_EDITOR
   style.normal.textColor = Color;
   UnityEditor.Handles.Label(pos, label, style);
   #endif
  }
 }

#endregion


#region Add
```

```
public static void DrawLine (string name, Vector3 from, Vector3 to, Color color=new Color(), bool additive
{
 LineGizmo gizmo = new LineGizmo() { from=from, to=to };
 AddGizmo(name, gizmo, color, additive:additive);
}

public static void DrawRay (string name, Vector3 pos, Vector3 dir, Color color=new Color(), bool additive
{
 LineGizmo gizmo = new LineGizmo() { from=pos, to=pos+dir };
 AddGizmo(name, gizmo, color, additive:additive);
}

public static void DrawPolyLine (string name, Vector3[] points, Color color=new Color(), bool additive=fals
{
 PolyLineGizmo gizmo = new PolyLineGizmo() { points=points };
 AddGizmo(name, gizmo, color, additive:additive);
}

public static void DrawSphere (string name, Vector3 pos, float radius, Color color=new Color(), bool addi
{
 SphereGizmo gizmo = new SphereGizmo() { pos=pos, radius=radius };
 AddGizmo(name, gizmo, color, additive:additive);
}

public static void DrawDot (string name, Vector3 pos, float size, Color color=new Color(), bool additive=fa
{
```

```
 DotGizmo gizmo = new DotGizmo() { pos=pos, size=size };
 AddGizmo(name, gizmo, color, additive:additive);
}

public static void DrawCircle (string name, Vector3 pos, float radius, Color color=new Color(), bool additiv
{
 CircleGizmo gizmo = new CircleGizmo() { pos=pos, radius=radius };
 AddGizmo(name, gizmo, color, additive:additive);
}

public static void DrawGrid (string name, Vector3 worldPos, Vector3 worldSize, int cellsX, int cellsZ, Colo
{
 GridGizmo gizmo = new GridGizmo() { pos=worldPos, size=worldSize, cellsX=cellsX, cellsZ=cellsZ };
 AddGizmo(name, gizmo, color, additive:additive);
}

public static void DrawGrid (string name, CoordRect rect, Vector3 worldPos, Vector3 worldSize, Color co
{
 GridGizmo gizmo = new GridGizmo() { pos=worldPos, size=worldSize, cellsX=rect.size.x, cellsZ=rect.siz
 AddGizmo(name, gizmo, color, additive:additive);
}

public static void DrawRect (string name, Vector2D worldPos, Vector2D worldSize, Color color=new Colo
{
 RectGizmo gizmo = new RectGizmo() { pos=worldPos, size=worldSize };
 AddGizmo(name, gizmo, color, additive:additive);
```

```
}

public static void DrawRect (string name, Vector3 worldPos, Vector3 worldSize, Color color=new Color(),
{
 RectGizmo gizmo = new RectGizmo() { pos=new Vector2D(worldPos.x, worldPos.z), size=new Vector2D
 AddGizmo(name, gizmo, color, additive:additive);
}

public static void DrawMultipleLines (string name, (Vector3,Vector3)[] lines, Color color=new Color(), boo
{
 MultiLineGizmo gizmo = new MultiLineGizmo() { lines=lines };
 AddGizmo(name, gizmo, color, additive:additive);
}

public static void DrawLabel (string name, Vector3 worldPos, string label, Color color=new Color(), bool a
{
 LabelGizmo gizmo = new LabelGizmo() { pos=worldPos, label=label };
 AddGizmo(name, gizmo, color, additive:additive);
}


public static void AddGizmo (string name, IGizmo gizmo, Color color, bool additive=false)
{
 if (color.r==0 && color.g==0 && color.b==0 && color.a==0)
  color = Color.white;
 gizmo.Color = color;
```

```
  if (allGizmos.TryGetValue(name, out GizmoGroup group))
  {
   if (!additive) group.gizmos.Clear();
   group.gizmos.Add(gizmo);

   allGizmos[name] = group;
  }

  else
  {
   group = new GizmoGroup();
   group.gizmos.Add(gizmo);
   allGizmos.Add(name, group);
  }
 }

 public static void Clear (string name) => allGizmos.Remove(name);
 public static void Clear () => allGizmos.Clear();


#endregion


#region Draw
```

```
#if UNITY_EDITOR
[RuntimeInitializeOnLoadMethod, UnityEditor.InitializeOnLoadMethod]
static void Subscribe ()
{
 #if UNITY_2019_1_OR_NEWER
 UnityEditor.SceneView.duringSceneGui += Draw;
 #else
 UnityEditor.SceneView.onSceneGUIDelegate += Draw;
 #endif
}

//[UnityEditor.DrawGizmo(UnityEditor.GizmoType.NonSelected | UnityEditor.GizmoType.Active)]
static public void Draw (UnityEditor.SceneView sceneView) //(Transform objectTransform, UnityEditor.Gi
{
 foreach (var kvp in allGizmos)
 {
  GizmoGroup group = kvp.Value;

  if (!group.enabled)
   continue;

  if (group.colorOverride)
   UnityEditor.Handles.color = group.color;

  for (int i=0; i<group.gizmos.Count; i++)
  {
```

```
    if (!group.colorOverride)
     UnityEditor.Handles.color = group.gizmos[i].Color;

    group.gizmos[i].Draw();
   }
  }

  //allGizmos.Clear();
  //will show gizmos only the frame they were assigned
 }
#endif

#endregion



#region Other

 public static Action<Matrices.Matrix, string> onPreviewMatrix;
 public static Action<Matrices.Matrix, string> onPreviewMatrixAdd;

 public static void ToMatrixPreview (this Matrices.Matrix matrix, string name=null) { onPreviewMatrix?.Invo
 public static void ToMatrixPreviewCopy (this Matrices.Matrix matrix, string name=null) { onPreviewMatrix
 public static void ToMatrixPreviewAdd (this Matrices.Matrix matrix, string name=null) { onPreviewMatrixA

 public static void ToMatrixPreviewMainthread (this Matrices.Matrix matrix, string name=null) { Den.Tools.
 public static void ToMatrixPreviewCopyMainthread (this Matrices.Matrix matrix, string name=null)
```

```
{
 Matrices.Matrix m2 = new Matrices.Matrix(matrix);
 Den.Tools.Tasks.CoroutineManager.Enqueue(() => onPreviewMatrix?.Invoke(m2, name) );
}
public static void ToMatrixPreviewAddMainthread (this Matrices.Matrix matrix, string name=null) { Den.To



public static Vector3 PositionHandle (Vector3 src)
{
 #if UNITY_EDITOR

 Vector3 dst = UnityEditor.Handles.PositionHandle(src, Quaternion.identity);

 if (src!=dst)
 {
  if (UnityEditor.Selection.activeObject != null)
   UnityEditor.Undo.RecordObject(UnityEditor.Selection.activeObject, "Position Changed");
  return dst;
 }
 else
  return src;

 #else
 return src;
 #endif
}
```

```
    public static Vector2D PositionHandle (Vector2D src) => (Vector2D)PositionHandle((Vector3)src);


   #endregion
  }
 }
```

```
ï»¿using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System;
using Den.Tools.Matrices;
using System.Runtime.InteropServices;

namespace Den.Tools
{
 public static class Erosion
 {
  #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

   //il2cpp doesn't see Matrix2D<int>
   [Serializable, StructLayout (LayoutKind.Sequential)]
   public class MatrixInt
   {
    public CoordRect rect; //never assign it's size manually, use ChangeRect
    public int count;
    public int pos;
    public int[] arr;

    public static implicit operator MatrixInt (Matrix2D<int> src)
     { return new MatrixInt(src); }

    public  MatrixInt (Matrix2D<int> src)
```

```
  { rect=src.rect; count=src.count; pos=src.pos; arr=src.arr; }
 }

#else

 public struct Cross
 {
  //public float xz; public float z; public float Xz;
  //public float x; public float c; public float X;
  //public float xZ; public float Z; public float XZ;

  public float[] vals;

  public static MooreCross Zero () { return new MooreCross() { vals = new float[5] }; }

  public Cross (float[] m, int i, int sizeX)
  {
   //xz = m[i-1-sizeX]; z = m[i-sizeX];  Xz = m[i+1-sizeX];
   //x = m[i-1];   c = m[i];   X = m[i+1];
   //xZ = m[i-1+sizeX]; Z = m[i+sizeX];  XZ = m[i+1+sizeX];

   vals = new float[]
   {
        m[i-sizeX],
    m[i-1],   m[i],   m[i+1],
        m[i+sizeX],
```

```
 };
}

public Cross (float val) { vals = new float[5]; for (int i=0; i<5; i++) vals[i] = val; }

public void SetToMatrix (float[] m, int i, int sizeX)
{
        m[i-sizeX] = vals[0];
  m[i-1] = vals[1];  m[i] = vals[2];   m[i+1] = vals[3];
        m[i+sizeX] = vals[4];
}

public void AddToMatrix (float[] m, int i, int sizeX)
{
        m[i-sizeX] += vals[0];
  m[i-1] += vals[1];  m[i] += vals[2];   m[i+1] += vals[3];
        m[i+sizeX] += vals[4];
}

public static Cross operator + (Cross c1, Cross c2) { for (int i=0; i<5; i++) c1.vals[i] += c2.vals[i]; return c1
public static Cross operator + (Cross c1, float f) { for (int i=0; i<5; i++) c1.vals[i] += f; return c1; }
public static Cross operator - (float f, Cross c) { for (int i=0; i<5; i++) c.vals[i] = f-c.vals[i]; return c; }
public static Cross operator - (Cross c1, float f) { for (int i=0; i<5; i++) c1.vals[i] -=f; return c1; }
public static Cross operator - (Cross c1, Cross c2) { for (int i=0; i<5; i++) c1.vals[i] -= c2.vals[i]; return c1;
public static Cross operator / (Cross c, float f) { for (int i=0; i<5; i++) c.vals[i] = c.vals[i]/f; return c; }
public static Cross operator * (Cross c, float f) { for (int i=0; i<5; i++) c.vals[i] = c.vals[i]*f; return c; }
```

```
public static Cross operator * (float f, Cross c) { for (int i=0; i<5; i++) c.vals[i] = c.vals[i]*f; return c; }

public float Min () { float min=2000000000; for (int i=0; i<5; i++) if (vals[i]<min) min = vals[i]; return min; }
public float MinSides () { float min=2000000000; for (int i=0; i<5; i++) { if (i==2) continue; if (vals[i]<min) m
public float MaxSides () { float max=-2000000000; for (int i=0; i<5; i++) { if (i==2) continue; if (vals[i]>max
public float Sum () { float sum=0; for (int i=0; i<5; i++) sum += vals[i]; return sum; }
public float SumSides () { float sum=0; for (int i=0; i<5; i++) { if (i==2) continue; sum += vals[i];} return su
public float Avg () { return Sum() / 5f; }
public float AvgSides () { return  SumSides() / 4f; }

public static Cross ClampMax (Cross c, float f) { for (int i=0; i<5; i++) c.vals[i] = Mathf.Max(f,c.vals[i]); retu
public static Cross ClampMin (Cross c, float f) { for (int i=0; i<5; i++) c.vals[i] = Mathf.Min(f,c.vals[i]); retur

public static Cross Pour (Cross height, float liquid)
{
 //if (liquid < 0.0000001f) return new Cross(0);

 //initial avg scatter
 float sum = height.Sum() + liquid;
 float avg = sum / 5;

 Cross pour = new Cross(avg);
 pour -= height;
 pour = ClampMax(pour, 0);
 //now liquids sum is larger than original
```

```
  //lowering all of the liquid cells
  int liquidCellsCount = 0;
  float currentLiquidSum = 0;
  for (int i=0; i<5; i++)
  {
   float val = pour.vals[i];
   if (val > 0.0001f) liquidCellsCount++;
   currentLiquidSum += val;
  }
  if (liquidCellsCount == 0) return pour; //should not happen
  float lowerAmount = (pour.Sum() - liquid) / liquidCellsCount;
  pour = pour - lowerAmount;
  pour = ClampMax(pour, 0);

  //in most cases now the delta is 0, but sometimes it's still needs to be adjusted
  if (Mathf.Abs(pour.Sum() - liquid) > 0.00001f)
  {
   if (Mathf.Abs(pour.Sum()) < 0.000001f) return new Cross(0); //this is 100% needed
   float factor = liquid / pour.Sum();
   pour *= factor;
  }

  return pour;
 }
}
public struct MooreCross
```

```
{
 //public float xz; public float z; public float Xz;
 //public float x; public float c; public float X;
 //public float xZ; public float Z; public float XZ;

 public float[] vals;

 public static MooreCross Zero () { return new MooreCross() { vals = new float[9] }; }

 public MooreCross (float[] m, int i, int sizeX)
 {
  //xz = m[i-1-sizeX]; z = m[i-sizeX];  Xz = m[i+1-sizeX];
  //x = m[i-1];   c = m[i];   X = m[i+1];
  //xZ = m[i-1+sizeX]; Z = m[i+sizeX];  XZ = m[i+1+sizeX];

  vals = new float[]
  {
   m[i-1-sizeX], m[i-sizeX],  m[i+1-sizeX],
   m[i-1],   m[i],   m[i+1],
   m[i-1+sizeX], m[i+sizeX],  m[i+1+sizeX]
  };
 }

 public void AddToMatrix (float[] m, int i, int sizeX)
 {
   m[i-1-sizeX] += vals[0]; m[i-sizeX] += vals[1];  m[i+1-sizeX] += vals[2];
```

```
    m[i-1] += vals[3];   m[i] += vals[4];   m[i+1] += vals[5];
    m[i-1+sizeX] += vals[6]; m[i+sizeX] += vals[7];  m[i+1+sizeX] += vals[8];
  }

  public void SetToMatrix (float[] m, int i, int sizeX)
  {
    m[i-1-sizeX] = vals[0]; m[i-sizeX] = vals[1];  m[i+1-sizeX] = vals[2];
    m[i-1] = vals[3];   m[i] = vals[4];   m[i+1] = vals[5];
    m[i-1+sizeX] = vals[6]; m[i+sizeX] = vals[7];  m[i+1+sizeX] = vals[8];
  }

  public static MooreCross operator + (MooreCross c1, MooreCross c2) { for (int i=0; i<9; i++) c1.vals[i] +=
  public static MooreCross operator - (float f, MooreCross c) { for (int i=0; i<9; i++) c.vals[i] = f-c.vals[i]; retu
  public static MooreCross operator / (MooreCross c, float f) { for (int i=0; i<9; i++) c.vals[i] = c.vals[i]/f; retu
  public static MooreCross operator * (MooreCross c, float f) { for (int i=0; i<9; i++) c.vals[i] = c.vals[i]*f; retu
  public static MooreCross operator * (float f, MooreCross c) { for (int i=0; i<9; i++) c.vals[i] = c.vals[i]*f; retu

  public static MooreCross ClampMax (MooreCross c, float f) { for (int i=0; i<9; i++) c.vals[i] = Mathf.Max(f,
  public static MooreCross ClampMin (MooreCross c, float f) { for (int i=0; i<9; i++) c.vals[i] = Mathf.Min(f,c

  public float Sum () { float sum=0; for (int i=0; i<9; i++) sum += vals[i]; return sum; }
 }
#endif

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "SetOrder")]
```

```
 public static extern void SetOrder(Matrix refm, MatrixInt order);
#else
 public static void SetOrder (Matrix refMatrix, Matrix2D<int> order)
 {
  int length = refMatrix.count;
  for (int i=0; i<length; i++) order.arr[i] = i;
  float[] refHeights = new float[length];
  Array.Copy(refMatrix.arr, refHeights, length);
  Array.Sort(refHeights, order.arr);

 // int[] orderCopy = new int[refArray.Length];
 // Array.Copy(orderArray, orderCopy, length);
 // for (int i=0; i<orderCopy.Length; i++) orderArray[i] = orderCopy[orderCopy.Length-1-i];

 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MaskBorders")]
 public static extern void MaskBorders(MatrixInt order);
#else
 public static void MaskBorders (Matrix2D<int> order)
 {
  for (int j=0; j<order.count; j++)
  {
```

```
   int pos = order.arr[j];

   int x = pos / order.rect.size.x;
   int z = pos % order.rect.size.x;

   if (x==0 || z==0 || x==order.rect.size.x-1 || z==order.rect.size.z-1) order.arr[j] = -1;
  }
 }
#endif

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "CreateTorrents"
 public static extern void CreateTorrents(Matrix heights, MatrixInt order, Matrix torrents);
#else
 public static void CreateTorrents (Matrix heights, Matrix2D<int> order, Matrix torrents)
 {
   CoordRect rect = heights.rect;
   for (int i=0; i<heights.count; i++) torrents.arr[i] = 1; //casting initial rain

   for (int j=heights.count-1; j>=0; j--)
   {
    //finding column ordered by height
    int pos = order.arr[j];
    if (pos<0) continue;
```

```
    MooreCross height = new MooreCross(heights.arr, pos, rect.size.x);
    MooreCross torrent = new MooreCross(torrents.arr, pos, rect.size.x); //moore
    if (torrent.vals[4] > 200000000) torrent.vals[4] = 200000000;

    MooreCross delta = height.vals[4] - height;
    delta = MooreCross.ClampMax(delta, 0);

    MooreCross percents = MooreCross.Zero();
    float sum = delta.Sum();
    if (sum>0.00001f) percents = delta / sum;

    MooreCross newTorrent = percents*torrent.vals[4];
    newTorrent.AddToMatrix(torrents.arr, pos, rect.size.x);
   }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "Erode")]
 public static extern void Erode(Matrix heights, Matrix torrents, Matrix mudflow, MatrixInt order,
  float erosionDurability = 0.9f, float erosionAmount = 1, float sedimentAmount = 0.5f);
#else
 public static void Erode (Matrix heights, Matrix torrents, Matrix mudflow, Matrix2D<int> order,
  float erosionDurability=0.9f, float erosionAmount=1f, float sedimentAmount=0.5f)
```

```
  {
    CoordRect rect = heights.rect;
    for (int i=0; i<mudflow.count; i++) mudflow.arr[i] = 0;

    for (int j=heights.count-1; j>=0; j--)
    {
     //finding column ordered by height
     int pos = order.arr[j];
     if (pos<0) continue;

     Cross height = new Cross(heights.arr, pos, rect.size.x);
     float h_min = height.Min();

     //getting height values
//     float[] m = heights; int i=pos; int sizeX = rect.size.x;
//     float h = m[i]; float hx = m[i-1]; float hX = m[i+1]; float hz = m[i-sizeX]; float hZ = m[i+sizeX];

     //height minimum
//     float h_min = h;
//     if (hx<h_min) h_min=hx; if (hX<h_min) h_min=hX; if (hz<h_min) h_min=hz; if (hZ<h_min) h_min=hZ;

     //erosion line
     float erodeLine = (heights.arr[pos] + h_min)/2f; //halfway between current and maximum height
     if (heights.arr[pos] < erodeLine) continue;
```

```
    //raising soil
    float raised = heights.arr[pos] - erodeLine;
    float maxRaised = raised*(torrents.arr[pos]-1) * (1-erosionDurability);
    if (raised > maxRaised) raised = maxRaised;
    raised *= erosionAmount;

    //saving arrays
    heights.arr[pos] -= raised;
    mudflow.arr[pos] += raised * sedimentAmount;
    //if (erosion != null) erosion.array[pos] += raised; //and writing to ref
   }

   //for (int i=0; i<heights.Length; i++)
   // if (float.IsNaN(heights[i])) Debug.Log("NaN");
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "TransferSettleM
 public static extern void TransferSettleMudflow(Matrix heights, Matrix mudflow, Matrix sediments, MatrixI
#else
 public static void TransferSettleMudflow(Matrix heights, Matrix mudflow, Matrix sediments, Matrix2D<int>
 {
  TransferMudflow (heights, mudflow, sediments, order, erosionFluidityIterations);
```

```
 SettleMudflow (heights, mudflow, order, ruffle:1f);
}

public static void TransferMudflow (Matrix heights, Matrix mudflow, Matrix sediments, Matrix2D<int> orde
{
 CoordRect rect = heights.rect;
 for (int i=0; i<sediments.count; i++) sediments.arr[i] = 0;

 #region Settling sediment

  for (int l=0; l<erosionFluidityIterations; l++)
  for (int j=heights.count-1; j>=0; j--)
  {
   //finding column ordered by height
   int pos = order.arr[j];
   if (pos<0) continue;

   Cross height = new Cross(heights.arr, pos, rect.size.x);
   Cross sediment = new Cross(mudflow.arr, pos, rect.size.x);

   float sedimentSum = sediment.Sum();
   if (sedimentSum < 0.00001f) continue;

   Cross pour = Cross.Pour(height, sedimentSum);

   pour.SetToMatrix(mudflow.arr, pos, rect.size.x);
```

```
   if (sediments != null) pour.AddToMatrix(sediments.arr, pos, rect.size.x);
  }

  //for (int i=0; i<heights.Length; i++)
  // if (float.IsNaN(heights[i])) Debug.Log("NaN");

  #endregion
 }

 public static void SettleMudflow (Matrix heights, Matrix mudflow, Matrix2D<int> order, float ruffle=0.1f)
 {


  //int seed = 12345;
  for(int j=heights.count-1; j>=0; j--)
  {
   //writing heights
   heights.arr[j] += mudflow.arr[j];

   /*seed = 214013*seed + 2531011;
   float random = ((seed>>16)&0x7FFF) / 32768f;

   int pos = order[j];
   if (pos<0) continue;

   //float[] m = heights; int sizeX = rect.size.x;
```

```
    //float h = m[pos]; float hx = m[pos-1]; float hX = m[pos+1]; float hz = m[pos-sizeX]; float hZ = m[pos+siz
    Cross height = new Cross(heights, pos, rect.size.x);

    //smoothing sediments a bit
    float s = mudflow[pos];
    if (s > 0.0001f)
    {
     float smooth = s/2f; if (smooth > 0.75f) smooth = 0.75f;
     heights[pos] = heights[pos]*(1-smooth) + height.AvgSides()*smooth;
    }

    else
    {
     float maxHeight = height.MaxSides();
     float minHeight = height.MinSides();
     float randomHeight = random*(maxHeight-minHeight) + minHeight;
    // heights[pos] = heights[pos]*(1-ruffle) + randomHeight*ruffle;
    }*/
   }
  }
 #endif

}//class
}//namespace
```

```
ï»¿using System;
//using System.Net.NetworkInformation;
using UnityEngine;

namespace Den.Tools
{
 public static class Id
 {
  private static ulong idVersion;

  public static ulong Generate ()
  {
   //timestamp (40 bits, 35 years of milliseconds)
   ulong dateTimeCode = 0;
   TimeSpan dateTime = DateTime.Now - new DateTime(2020,1,1);
   dateTimeCode = (ulong)(dateTime.TotalMilliseconds % 0xFFFFFFFFFF);

   //session number (12 bits, 16383)
   //alternative to machine id
   //#if UNITY_EDITOR
   //ulong sessionCode = (ulong)(UnityEditor.EditorAnalyticsSessionInfo.sessionCount % 0xFFF);
   //#else
   ulong sessionCode = 0;
   //#endif
   //doesn't work on serialization
```

```
 //version (12 bits, 16383)
 //in case several generators are created in one millisecond with script.
 idVersion++;
 if (idVersion >= 0xFFF) idVersion = 1;

 return (dateTimeCode << 40) | (sessionCode << 12) | idVersion;
}


public static double GetIdTimestamp (this ulong id) => (id >> 40) / 4.0;
public static ulong GetIdSession (this ulong id) => (id>>12) & 0xFFFF;
public static ulong GetIdVersion (this ulong id) => id & 0xFFFF;

public static byte[] ToByteArray (ulong id)
{
 byte[] arr = new byte[8];
 for (int i=0; i<8; i++)
  arr[7-i] = (byte)((id >> (i*8)) | 0xFF);
 return arr;
}

public static string ToString (ulong id)
{
 string str = "";
 for (int i=13; i>=0; i--)
 {
```

```
  int e = (int)((id >> (i*5)) & 0x3F);
  char c = ByteToCharLut[e];
  str += c;
 }
 return str;
}

private static string ByteToCharLut = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQ



public static ulong MachineId ()
{
 //machine mac address
 /*int macAddressCode = 0;
 byte[] macAddressBytes = null;
 foreach (NetworkInterface nic in NetworkInterface.GetAllNetworkInterfaces())
  if (nic.OperationalStatus == OperationalStatus.Up && nic.NetworkInterfaceType != NetworkInterfaceTyp
  {
   macAddressBytes = nic.GetPhysicalAddress().GetAddressBytes();
   break;
  }
 foreach (byte bt in macAddressBytes)
  macAddressCode = macAddressCode ^ bt;*/

 //machine name (if mac address is 0)
 /*int machineNameCode = 0;
```

```
    string machineName = Environment.MachineName;
    foreach (char ch in machineName)
     machineNameCode = machineNameCode ^ Convert.ToByte(ch);*/

    //process id
    //int pid = System.Diagnostics.Process.GetCurrentProcess().Id;

    return 0;
   }
  }
 }
```

```
ï»¿using UnityEngine;
using System;
using System.Runtime.InteropServices;

namespace Den.Tools
{
 [Serializable, StructLayout (LayoutKind.Sequential)] //to pass to native
 public class Noise
 {
  [SerializeField] private int seed; //the seed this noise was initialized with, for informational purpose
  [SerializeField] private int subSeed; //the seed that could be changed without re-building permutation. Nev

  public readonly int permutationCount;
  private readonly int permutationCountMinusOne;
  private readonly uint[] permutation;

  public int Seed
  {
   get{ return seed; }
   set{ FillPermutation(value); seed = value; }
  }

  public int SubSeed
  {
   get{ return subSeed; }
   set{ subSeed = value & permutationCountMinusOne; }
```

```
}

private static readonly int[] gradX = { 1, -1,  1, -1,  1, -1,  0,  0 };
private static readonly int[] gradZ = { 0,  0,  1,  1, -1, -1,  1, -1 };



public Noise (int seed, int permutationCount=16384)
{
 this.permutation = new uint[permutationCount*2];
 this.permutationCount = permutationCount;
 this.permutationCountMinusOne = permutationCount - 1;
 this.seed = seed;
 FillPermutation(seed);
}

public Noise (Noise noise)
{
 this.permutation = noise.permutation;
 this.permutationCount = noise.permutationCount;
 this.permutationCountMinusOne = noise.permutationCountMinusOne;
 this.seed = noise.seed;
 this.subSeed = noise.subSeed;
}

public Noise (Noise noise, int subSeed)
```

```
{
 this.permutation = noise.permutation;
 this.permutationCount = noise.permutationCount;
 this.permutationCountMinusOne = noise.permutationCountMinusOne;
 this.seed = noise.seed;
 this.subSeed = subSeed & permutationCountMinusOne;
}



private void FillPermutation (int seed)
{
 //consists of 2 parts: one for random, and one repeats itself

 //random part
 int permutationCount = permutation.Length / 2;
 for (int i=0; i<permutationCount; i++)
 {
  //random
  seed = 214013*seed + 2531011;
  float val = ((seed>>16)&0x7FFF) / 32768f; //ASAP: increase 32768

  permutation[i] = (uint)(val*permutationCount);
 }

 //clone part
 for (int i=0; i<permutationCount; i++)
```

```
  permutation[i+permutationCount] = permutation[i];
}

public static int Pair (int k1, int k2)
/// Using Cantor function to uniquely encode two numbers
{
 int t = (k1+k2)*(k1+k2+1);
 return t/2 + k2;
}

public float Random (int x)
{
 x = x & permutationCountMinusOne; //permutationCountMinusOne = 11111111, so this will throw away a

 x = (int)permutation[x + permutation[subSeed]];  //x+permutation[] is never bigger than permutation length

 return 1f*x / permutationCount;
}

public float Random (int x, int z)
{
 x = x & permutationCountMinusOne;
 z = z & permutationCountMinusOne;

 x = (int)permutation[x + permutation[z + permutation[subSeed]]];
```

```
 return 1f*x / permutationCount;
}

public float Random (int x, int y, int z)
{
 x = x & permutationCountMinusOne;
 y = y & permutationCountMinusOne;
 z = z & permutationCountMinusOne;

 x = (int)permutation[x + permutation[z + permutation[y + permutation[subSeed]]]];

 return 1f*x / permutationCount;
}

public float Random (int x, int y, int z, int w)
{
 x = x & permutationCountMinusOne;
 y = y & permutationCountMinusOne;
 z = z & permutationCountMinusOne;
 w = w & permutationCountMinusOne;

 x = (int)permutation[x + permutation[z + permutation[y + permutation[w + permutation[subSeed]]]]];

 return 1f*x / permutationCount;
}
```

```
public int RandomInt (int x)
{
 x = x & permutationCountMinusOne;
 x = (int)permutation[x + permutation[subSeed]];
 return x;
}

public float Linear (float x, float z)
{
 //cell (i) and percent (f)
 int xi = x>0? (int)x : (int)x-1;
 int zi = z>0? (int)z : (int)z-1;

 float xf = x-xi;
 float zf = z-zi;

 xi = xi & permutationCountMinusOne;
 zi = zi & permutationCountMinusOne;
 //int f = ((xi^zi)/permutationCount^xi) & permutationCountMinusOne;

 //random corners
 uint permSeed = permutation[subSeed];
 uint aa = permutation[ permutation[ permSeed + xi] + zi ];
 uint ab = permutation[ permutation[ permSeed + xi] + zi+1];
 uint ba = permutation[ permutation[ permSeed + xi+1] + zi];
 uint bb = permutation[ permutation[ permSeed + xi+1] + zi+1];
```

```
 //fade
 float xfade = 3*xf*xf - 2*xf*xf*xf;  //xf*xf*xf*(xf* (xf*6 - 15) + 10);
 float zfade = 3*zf*zf - 2*zf*zf*zf;  //zf*zf*zf*(zf* (zf*6 - 15) + 10);

 //interpolation
 float x1 = aa*(1-xfade) + ba*xfade;
 float x2 = ab*(1-xfade) + bb*xfade;
 float z2 = x1*(1-zfade) + x2*zfade;

 return 1f*z2 / permutationCount;
}

public float Perlin (float x, float z)
{
 //cell (i) and percent (f)
 int xi = x>0? (int)x : (int)x-1;
 int zi = z>0? (int)z : (int)z-1;

 float xf = x-xi;
 float zf = z-zi;

 xi = xi & permutationCountMinusOne;
 zi = zi & permutationCountMinusOne;

 //hash
```

```
 int permSeed = (int)permutation[subSeed];
 int aa = (int)( permutation[ permutation[permSeed+xi] + zi ] )&0x7;
 int ab = (int)( permutation[ permutation[permSeed+xi] + zi+1] )&0x7;
 int ba = (int)( permutation[ permutation[permSeed+xi+1] + zi] )&0x7;
 int bb = (int)( permutation[ permutation[permSeed+xi+1] + zi+1] )&0x7;

 //grad and dot
 float aa_gd = gradX[aa]*xf + gradZ[aa]*zf;
 float ab_gd = gradX[ab]*xf + gradZ[ab]*(zf-1);
 float ba_gd = gradX[ba]*(xf-1) + gradZ[ba]*zf;
 float bb_gd = gradX[bb]*(xf-1) + gradZ[bb]*(zf-1);

 //fade
 float xfade = xf*xf*xf*(xf* (xf*6 - 15) + 10); //3*xf*xf - 2*xf*xf*xf;
 float zfade = zf*zf*zf*(zf* (zf*6 - 15) + 10); //3*zf*zf - 2*zf*zf*zf;

 //interpolation
 float x1 = aa_gd*(1-xfade) + ba_gd*xfade;
 float x2 = ab_gd*(1-xfade) + bb_gd*xfade;
 float z2 = x1*(1-zfade) + x2*zfade;

 return (z2+1) / 2;
}


public float Simplex (float x, float z) // based on Stefan Gustavson's code (http://webstaff.itn.liu.se/~stegu/
```

```
{
 float f2 = 0.5f*(1.732050807568877f-1.0f);
 float g2 = (3.0f-1.732050807568877f)/6.0f;
 float s = (x+z) * f2; // Hairy factor for 2D

 int i = (x+s)>0? (int)(x+s) : (int)(x+s)-1;
 int j = (z+s)>0? (int)(z+s) : (int)(z+s)-1;

 float t = (i+j) * g2;

 float X0 = i-t;
 float Z0 = j-t;
 float x0 = x-X0;
 float z0 = z-Z0;

 // For the 2D case, the simplex shape is an equilateral triangle.
 int i1 = x0>z0? 1 : 0;
 int j1 = x0>z0? 0 : 1;

 float x1 = x0 - i1 + g2;
 float z1 = z0 - j1 + g2;
 float x2 = x0 - 1 + 2*g2;
 float z2 = z0 - 1 + 2*g2;

 // Work out the hashed gradient indices of the three simplex corners
 int ii = i & permutationCountMinusOne;
```

```
 int jj = j & permutationCountMinusOne;

 uint permSeed = permutation[subSeed];
 uint gi0 = permutation[ii+permutation[jj+permSeed]] % 8;
 uint gi1 = permutation[ii+i1+permutation[jj+j1+permSeed]] % 8;
 uint gi2 = permutation[ii+1+permutation[jj+1+permSeed]] % 8;

 // Calculate the contribution from the three corners
 float n0, n1, n2;

 float t0 = 0.5f - x0*x0 - z0*z0;
 if(t0<0) n0 = 0;
 else n0 = t0*t0*t0*t0 * (gradX[gi0]*x0 + gradZ[gi0]*z0);

 float t1 = 0.5f - x1*x1-z1*z1;
 if(t1<0) n1 = 0;
 else n1 = t1*t1*t1*t1 * (gradX[gi1]*x1 + gradZ[gi1]*z1);

 float t2 = 0.5f - x2*x2-z2*z2;
 if(t2<0) n2 = 0;
 else n2 = t2*t2*t2*t2 * (gradX[gi2]*x2 + gradZ[gi2]*z2);

 return (float)(70.0 * (n0 + n1 + n2) + 1) / 2;
}
```

```
[DllImport ("NativePlugins", EntryPoint = "NoiseFractal")]
public static extern float NativeFractal (uint[] p, int pc, float x, float y, float size, int iterations, float detail, fl

public float Fractal (float x, float y, float size, int iterations=-1, float detail=0.5f, float turbulence=0, int type=
{
 //if (native)
 // return NativeFractal(permutation, permutationCount, x, y, size, iterations, detail, turbulence, type, subS

 float result = 0; //0.5f;
 float freq = size;
 float amp = 1;
 float absTurbulence = turbulence>0? turbulence : -turbulence;

 //get number of iterations
 if (iterations < 0) iterations = (int)Mathf.Log(size,2) + 1; //+1 max size iteration

 //subSeed = subSeed & permutationCount-iterations-4-1;
 int permSeed = (int)permutation[subSeed];

 //applying noise
 freq = size;
 for (int i=0; i<iterations;i++)
 {
  //freq++; //return it back for backwards compatibility
  float val = 0;
  switch (type)
```

```
{
 case -1:

  val = Mathf.PerlinNoise(
   x/freq + (permutation[permSeed+0+i]+permutation[permSeed+1+i])/1000f,
   y/freq + (permutation[permSeed+2+i]+permutation[permSeed+3+i])/1000f);
  break;
 case 0:
  val = Mathf.PerlinNoise(
   x/freq + (permutation[permSeed+0+i]+permutation[permSeed+1+i]),
   y/freq + (permutation[permSeed+2+i]+permutation[permSeed+3+1]));
  break;  // +arg should not be int, noise repeats itself
 case 1: val = Linear(x/freq, y/freq); break;
 case 2: val = Perlin(x/freq, y/freq); break;
 case 3: val = Simplex(x/freq, y/freq); break;
 default: val = 0; break; //should not happen
}

//turbulence
if (absTurbulence > 0.001f)
{
 float turb = val*2 - 1;
 if (turb<0) turb = -turb;
 if (turbulence>0) turb = 1-turb;

 val = val*(1-absTurbulence) + turb*absTurbulence;
```

```
  }

  //standard mode
  result += val*amp;

  freq = freq / 2f;
  amp *= detail; //detail is 0.5 by default
 }

 return result * (1-detail);  //NOTE: factor is 0 on the large detail
}

[System.Obsolete] public float OverlayFractal (int x, int y, float persistence=2, int iterations=10, float turbu
{
 float result = 0.5f;

 float freq = Mathf.Pow(2,iterations);
 float amp = 1;

 for (int i=iterations-1; i>=0; i--) //symbolize iterating from high-size noise to low one
 {
  float val = 0;
  switch (type)
  {
   case 0: val = Mathf.PerlinNoise(x/freq, y/freq); break;
```

```
     case 1: val = Linear(x/freq, y/freq); break;
     case 2: val = Perlin(x/freq, y/freq); break;
     case 3: val = Simplex(x/freq, y/freq); break;
     default: val = 0; break; //should not happen
    }

    val = (val-0.5f)*amp + 0.5f;

    if (result > 0.5f) result = 1-2*(1-result)*(1-val); //(1 - (1-2*(perlin-0.5f)) * (1-result));
    else result = 2*val*result;

    freq /= 2;
    amp /= persistence;
   }

   return result;//result/ampMax;
  }
 }
}
```

```
ï»¿using UnityEngine;
using UnityEngine.Profiling;
using System;
using System.Collections;
using System.Collections.Generic;

namespace Den.Tools
{
 [System.Serializable]
 public class ObjectsPool : MonoBehaviour
 {
  [System.Serializable]
  public class Prototype //: IEquatable<Prototype>
  {
   public GameObject prefab;

   public bool allowReposition = true;
   public bool instantiateClones = false;

   //public bool useRotation = true;
   //public bool rotateYonly = false;
   public bool regardPrefabRotation = false;

   //public bool useScale = true;
   //public bool scaleYonly = false;
   public bool regardPrefabScale = false;
```

```
/*public bool Equals (Prototype p)
{
 Debug.Log("Equals");

 return prefab == p.prefab &&
  allowReposition == p.allowReposition &&
  instantiateClones == p.instantiateClones;
}*/

public override int GetHashCode ()
 { return prefab.GetHashCode(); }

public Prototype () { }
public Prototype (Prototype src)
{
 prefab = src.prefab;
 allowReposition = src.allowReposition;
 instantiateClones = src.instantiateClones;
 //useRotation = src.useRotation;
 //rotateYonly = src.rotateYonly;
 regardPrefabRotation = src.regardPrefabRotation;
 //useScale = src.useScale;
 //scaleYonly = src.scaleYonly;
 regardPrefabScale = src.regardPrefabScale;
```

```
 }
}

[System.Serializable]
private class Pool
{
 [SerializeField] public Prototype prototype; //readonly, but readonly is not serialized
 [SerializeField] public List<GameObject> instances;

 //statistics
 private int created = 0;
 private int moved = 0;
 private int deleted = 0;
 public string stats {get{ return "created:" + created + " moved:" + moved + " deleted:" + deleted; }}
 public void ResetStats () { created=0; moved=0; deleted=0; }

 public Pool (Prototype prototype)
 {
  this.prototype = new Prototype(prototype);
  this.instances = new List<GameObject>();
 }


 private void ClearNulls ()
 /// Clearing removed items
 {
```

```
 for (int i=instances.Count-1; i>=0; i--)
 {
  if (instances[i] == null)
   instances.RemoveAt(i);
 }
}

private void ClampCount (int targetCount)
/// And removing unused instances
{
 int instancesCount = instances.Count;
 if (instances.Count < targetCount) return;

 for (int i=targetCount; i<instancesCount; i++)
 {
  #if UNITY_EDITOR
  if (!UnityEditor.EditorApplication.isPlaying)
   GameObject.DestroyImmediate(instances[i].gameObject);
  else
  #endif
   GameObject.Destroy(instances[i].gameObject);

  deleted++;
 }

 instances.RemoveRange(targetCount, instancesCount-targetCount);
```

```
}



private void AppendCount (int targetCount, Transform parent=null)

/// Will create new items until it reach target count

{

 int instancesCount = instances.Count;


 moved += instancesCount;


 for (int i=instancesCount; i<targetCount; i++)

 {

  GameObject gobj = InstantiateObject();

  gobj.transform.parent = parent;

  gobj.hideFlags = parent.gameObject.hideFlags;


  instances.Add(gobj);

  created++;

 }

}



private void MoveItems (List<Transition> drafts, int startNum, int endNum)

{

 for (int i=startNum; i<endNum; i++)

 {
```

```
  GameObject gobj = instances[i];
  Transform tfm = gobj.transform;
  Transition draft = drafts[i];

  tfm.localPosition = draft.pos; //transformation.MultiplyPoint3x4(draft.pos);

  if (prototype.regardPrefabRotation) tfm.localRotation = draft.rotation * prototype.prefab.transform.rotatio
  else tfm.localRotation = draft.rotation;

  if (prototype.regardPrefabScale) tfm.transform.localScale = draft.scale.x * prototype.prefab.transform.lo
  else tfm.transform.localScale = draft.scale;
 }
}


public void Reposition (List<Transition> drafts, Transform parent=null)
{
 ResetStats();
 ClearNulls();

 int draftsCount = drafts.Count;
 int instancesCount = instances.Count;

 if (draftsCount < instances.Count)
  ClampCount(draftsCount);
```

```
 else if (draftsCount > instances.Count)
  AppendCount(draftsCount, parent);

 MoveItems(drafts, 0, draftsCount);
}

public IEnumerator RepositionRoutine (List<Transition> drafts, Transform parent=null, int objsPerIteratio
{
 ResetStats();
 ClearNulls();

 int draftsCount = drafts.Count;
 int instancesCount = instances.Count;
 int iterations = draftsCount / objsPerIteration + 1;

 if (draftsCount < instances.Count)
  ClampCount(draftsCount);

 for (int i=0; i<iterations; i++)
 {
  Profiler.BeginSample("Pool Apply");

  int startNum = i * objsPerIteration;
  int endNum = (i+1) * objsPerIteration;
  if (endNum > draftsCount) endNum = draftsCount;
```

```
  Profiler.BeginSample("Pool Instantiate");
  if (endNum > instances.Count)
   AppendCount(endNum, parent);
  Profiler.EndSample();

  Profiler.BeginSample("Pool Move");
  MoveItems(drafts, startNum, endNum);
  Profiler.EndSample();

  Profiler.EndSample();

  yield return null;
 }
}

public GameObject InstantiateObject ()
{
 GameObject gobj = null;

 #if UNITY_EDITOR
 if (!prototype.instantiateClones &&
  !UnityEditor.EditorApplication.isPlaying &&
  UnityEditor.PrefabUtility.GetPrefabAssetType(prototype.prefab)!=UnityEditor.PrefabAssetType.NotAPre
   gobj = (GameObject)UnityEditor.PrefabUtility.InstantiatePrefab(prototype.prefab);
```

```
  else
    gobj = GameObject.Instantiate(prototype.prefab);
  #else
  gobj = GameObject.Instantiate(prototype.prefab);
  #endif

  return gobj;
 }

 public void Clear ()
 {
  if (instances == null) return; //when error occurred on pool creation
  int instancesCount = instances.Count;
  for (int i=0; i<instancesCount; i++)
  {
   #if UNITY_EDITOR
   if (!UnityEditor.EditorApplication.isPlaying)
    GameObject.DestroyImmediate(instances[i].gameObject);
   else
   #endif
    GameObject.Destroy(instances[i].gameObject);
  }
  instances.Clear();
 }
}
```

```
//private Dictionary<Prototype,Pool> pools = new Dictionary<Prototype,Pool>();
[SerializeField] private Pool[] pools = new Pool[0];



public void SetPrototypes (Prototype[] prototypes)
/// Will clear and remove pools that are not in array and add new pools with array prototypes
/// Will set the pools order exactly as it was given in the array
{
 //clearing pools with null prefab (rare case)
 for (int i=pools.Length-1; i>=0; i--)
  if (pools[i]==null || pools[i].prototype==null || pools[i].prototype.prefab==null || pools[i].prototype.prefab.Ec
  {
   pools[i].Clear();
   ArrayTools.RemoveAt(ref pools, i);
  }

 //creating the dictionary of used prototypes/pools
 //TODO: use ClearPrortotypes?
 Dictionary<Prototype,Pool> usedPools = new Dictionary<Prototype,Pool>(pools.Length);//, new Prototyp
 for (int i=0; i<pools.Length; i++)
  if (!usedPools.ContainsKey(pools[i].prototype)) //avoiding two pools with one prototype (could be created
   usedPools.Add(pools[i].prototype, pools[i]);

 //reposition
```

```
 Pool[] newPools = new Pool[prototypes.Length];
 for (int i=0; i<prototypes.Length; i++)
 {
  if (usedPools.TryGetValue(prototypes[i], out Pool pool))
   usedPools.Remove(prototypes[i]);
  else
   pool = new Pool(prototypes[i]);

  //other than instantiateClones and allowReposition (they are in comparer) should be copied
  pool.prototype.regardPrefabRotation = prototypes[i].regardPrefabRotation;
  pool.prototype.regardPrefabScale = prototypes[i].regardPrefabScale;

  newPools[i] = pool;
 }

 //clearing unused
 if (usedPools.Count != 0)
  foreach (var kvp in usedPools)
   kvp.Value.Clear();

 pools = newPools;
}


public void ClearPrototypes (Prototype[] prototypes)
/// Will clear and remove pools that are in array
```

```
{
 HashSet<Prototype> prototypesHashSet = new HashSet<Prototype>(prototypes);//, new PrototypeComp

 List<Pool> newPools = new List<Pool>();
 for (int i=0; i<pools.Length; i++)
 {
  if (prototypesHashSet.Contains(pools[i].prototype))
   pools[i].Clear();
  else
   newPools.Add(pools[i]);
 }

 pools = newPools.ToArray();
}


public void Reposition (Prototype[] prototypes, List<Transition>[] drafts)
{
 //excluding 0-count from input drafts and prototypes
 for (int i=drafts.Length-1; i>0; i--)
  if (drafts[i].Count == 0)
   { ArrayTools.RemoveAt(ref prototypes,i); ArrayTools.RemoveAt(ref drafts,i); }

 SetPrototypes(prototypes); //will set the pools count and order as given in prototypes array

 for (int i=0; i<pools.Length; i++)
```

```
  pools[i].Reposition(drafts[i], this.transform);
}



public IEnumerator RepositionRoutine (Prototype[] prototypes, List<Transition>[] drafts, int objsPerIteratio
{
 //excluding 0-count from input drafts and prototypes
 for (int i=drafts.Length-1; i>=0; i--)
  if (drafts[i].Count == 0)
   { ArrayTools.RemoveAt(ref prototypes,i); ArrayTools.RemoveAt(ref drafts,i); }

 SetPrototypes(prototypes); //will set the pools count and order as given in prototypes array

 for (int i=0; i<pools.Length; i++)
 {
  IEnumerator e = pools[i].RepositionRoutine(drafts[i], this.transform, objsPerIteration);
  while (e.MoveNext()) { yield return null; }
 }
}



public class PrototypeComparer : IEqualityComparer<Prototype>
{
 public bool Equals (Prototype p1, Prototype p2)
 {
  return p1.prefab == p2.prefab &&
```

```
    p1.allowReposition == p2.allowReposition &&
    p1.instantiateClones == p2.instantiateClones;
  }

  public int GetHashCode (Prototype p)
  {
   if (p==null || p.prefab==null || p.prefab.Equals(null))
    return 0;

   return p.prefab.GetHashCode() +
    (p.allowReposition ? 0 : 1) +
    (p.instantiateClones ? 0 : 2);
  }
}



public void Depool (GameObject obj)
/// Removes an object from the pool without actually deleting it
{
 for (int p=0; p<pools.Length; p++)
  pools[p].instances.Remove(obj);
 //SOON: check performance
}
```

```
/*#region Serialization

 [SerializeField] private Pool[] serializedPools;

 public void OnBeforeSerialize ()
 {
  if (serializedPools==null || serializedPools.Length!=pools.Count) serializedPools = new Pool[pools.Count

  int i = 0;
  foreach (var kvp in pools)
  {
   Pool pool = kvp.Value;
   serializedPools[i] = pool;

   i++;
  }
 }

 public void OnAfterDeserialize ()
 {
  if (serializedPools==null) return;

  pools.Clear();

  for (int i=0; i<serializedPools.Length; i++)
```

```
    {
      Pool pool = serializedPools[i];
      if (pool.prototype == null) continue;
      if (pool.instances[i] == null || pool.instances.Count==0) continue;

      pools.Add(pool.prototype, pool);
     }
    }

  #endregion*/
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;

namespace Den.Tools
{
 public class DictionaryOrdered<TKey,TValue> :
  IDictionary<TKey, TValue>, IList<TKey>,
  ICollection<KeyValuePair<TKey, TValue>>, ICollection<TKey>,  ICollection,
  IEnumerable<KeyValuePair<TKey, TValue>>, IEnumerable<TKey>,  IEnumerable,
  IReadOnlyCollection<KeyValuePair<TKey, TValue>>, IReadOnlyCollection<TKey>

 /// Maintains the order of keys, and has access by number
 {
  protected Dictionary<TKey,TValue> dict = new Dictionary<TKey, TValue>();
  protected List<TKey> order = new List<TKey>();

  public DictionaryOrdered () { }
  public DictionaryOrdered (IEqualityComparer<TKey> comparer) { dict=new Dictionary<TKey,TValue>(cor
  public DictionaryOrdered (int capacity) { dict=new Dictionary<TKey, TValue>(capacity); order=new List<T
  public DictionaryOrdered (int capacity, IEqualityComparer<TKey> comparer) { dict=new Dictionary<TKey,
  public DictionaryOrdered (DictionaryOrdered<TKey,TValue> other) { dict=new Dictionary<TKey,TValue>(

  public Dictionary<TKey,TValue>.ValueCollection Values => dict.Values;
  public Dictionary<TKey,TValue>.KeyCollection Keys => dict.Keys;
  ICollection<TValue> IDictionary<TKey, TValue>.Values => dict.Values;
```

```
ICollection<TKey> IDictionary<TKey, TValue>.Keys => dict.Keys;

public int Count => order.Count;

public IEqualityComparer<TKey> Comparer => dict.Comparer;


public TValue this[TKey key]

{

 get => dict[key];

 set => dict[key]=value;

}


public TValue this[int num]

{

 get => dict[order[num]];

 set => dict[order[num]]=value;

}


TKey IList<TKey>.this[int num]

{

 get => order[num];

 set => order[num]=value;

}


public TKey GetKeyByNum(int num) => order[num];

public void SetKeyByNum(int num, TKey val) => order[num] = val;


public bool TryGetValue (TKey key, out TValue value) => dict.TryGetValue(key, out value);
```

```
public void Add (TKey key, TValue value)
{
 dict.Add(key, value); //dict goes first just to avoid adding to order on excepton
 order.Add(key);
}

public void Add (KeyValuePair<TKey,TValue> kvp)
{
 dict.Add(kvp.Key, kvp.Value);
 order.Add(kvp.Key);
}

public void Add (TKey key)  /// Adds default value
{
 dict.Add(key, default(TValue));
 order.Add(key);
}

public bool TryAdd (TKey key, TValue value)
{
 if (dict.ContainsKey(key)) return false;

 dict.Add(key, value); //dict goes first just to avoid adding to order on excepton
 order.Add(key);
```

```
 return true;
}

public void Insert (int index, TKey key, TValue value)
{
 dict.Add(key, value);
 order.Insert(index, key);
}

public void Insert (int index, TKey key)  /// Adds default value
{
 dict.Add(key, default(TValue));
 order.Insert(index, key);
}

public int IndexOf (TKey key) => order.IndexOf(key);

public void Clear ()
{
 dict.Clear();
 order.Clear();
}

public bool Remove (TKey key)
{
 if (!dict.ContainsKey(key))
```

```
  return false;

 int index = order.IndexOf(key);
 if (index<0)
  return false;

 dict.Remove(key);
 order.RemoveAt(index);
 return true;
}

public bool Remove (KeyValuePair<TKey,TValue> kvp)
{
 if (!((ICollection<KeyValuePair<TKey,TValue>>)dict).Contains(kvp))
  return false;

 int index = order.IndexOf(kvp.Key);
 if (index<0)
  return false;

 dict.Remove(kvp.Key);
 order.RemoveAt(index);
 return true;
}

public void RemoveAt (int index)
```

```
{
 dict.Remove(order[index]);
 order.RemoveAt(index);
}

public void Switch (int n1, int n2)
{
 TKey temp = order[n1];
 order[n1] = order[n2];
 order[n2] = temp;
}

public bool ContainsKey (TKey key) => dict.ContainsKey(key);
public bool ContainsValue (TValue value) => dict.ContainsValue(value);
public bool Contains (TKey key) => dict.ContainsKey(key);
public bool Contains (KeyValuePair<TKey,TValue> kvp) => ((ICollection<KeyValuePair<TKey,TValue>>)

public void CopyTo (Array array, int index)
{
 int max = array.Length<(order.Count+index) ? array.Length : order.Count+index;
 for (int i=index; i<max; i++)
 {
  TKey key = order[i-index];
  array.SetValue(new KeyValuePair<TKey,TValue> (key, dict[key]), i);
 }
```

```
}

public void CopyTo (KeyValuePair<TKey,TValue>[] array, int index) => CopyTo((Array)array, index);
public void CopyTo (TKey[] array, int index) => order.CopyTo(array,index);


bool ICollection.IsSynchronized => ((ICollection)dict).IsSynchronized;
object ICollection.SyncRoot => ((ICollection)dict).SyncRoot;
public bool IsReadOnly => ((ICollection<KeyValuePair<TKey,TValue>>)dict).IsReadOnly && ((ICollection

public IEnumerator<KeyValuePair<TKey,TValue>> GetEnumerator()
{
 foreach (TKey key in order)
  yield return new KeyValuePair<TKey,TValue> (key, dict[key]);
}
IEnumerator IEnumerable.GetEnumerator()
{
 foreach (TKey key in order)
  yield return new KeyValuePair<TKey,TValue> (key, dict[key]);
}
IEnumerator<TKey> IEnumerable<TKey>.GetEnumerator() => order.GetEnumerator();


#region Extensions

 public DictionaryOrdered (TKey[] keys, TValue[] values)
```

```
/// Creates new dictionary from keys and values
/// Doesn't check keys array for duplicates
{
 if (keys.Length != values.Length)
  throw new Exception("Could not create ordered dictionary: keys and values lengths differ");

 dict=new Dictionary<TKey, TValue>(keys.Length);
 for (int i=0; i<keys.Length; i++)
  dict.Add(keys[i], values[i]);

 order=new List<TKey>(keys.Length);
 order.AddRange(keys);
}



public DictionaryOrdered (TKey[] keys)
/// Creates new empty dictionary with keys structure. Fills all values as null
/// Doesn't check keys array for duplicates
{
 dict=new Dictionary<TKey, TValue>(keys.Length);
 order=new List<TKey>(keys.Length);

 dict=new Dictionary<TKey, TValue>(keys.Length);
 for (int i=0; i<keys.Length; i++)
  dict.Add(keys[i], default);
```

```
 order=new List<TKey>(keys.Length);
 order.AddRange(keys);
}


public void TakeMatchingValuesFrom (DictionaryOrdered<TKey,TValue> src)
/// If src has a same key it's values is copied to the dict
{
 foreach (TKey key in order)
 {
  if (src.dict.TryGetValue(key, out TValue val))
   dict[key] = val; //iterating order, so dict should not change
 }
}

#endregion

#region Serialization

public (TKey[], TValue[]) Serialize ()
{
 int count = Count;
 TKey[] keys = new TKey[Count];
 TValue[] vals = new TValue[Count];

 for (int i=0; i<count; i++)
```

```
 {
  TKey key = order[i];
  keys[i] = key;
  vals[i] = dict[key];
 }

 return (keys, vals);
}

public void Serialize (ref TKey[] keys, ref TValue[] vals)
{
 int count = Count;
 if (keys.Length != count) keys = new TKey[Count];
 if (vals.Length != count) vals = new TValue[Count];

 for (int i=0; i<count; i++)
 {
  TKey key = order[i];
  keys[i] = key;
  vals[i] = dict[key];
 }
}

public void Deserialize (TKey[] keys, TValue[] values)
{
 if (keys.Length != values.Length)
```

```
      throw new Exception("Could not deserialize ordered dictionary: keys and values lengths differ");


     Clear();


     for (int i=0; i<keys.Length; i++)
      Add(keys[i], values[i]);
    }


    public void ReCreateDictionary ()
    /// Unity 2021.2 Beta cannot use re-serialized dictionary. Cannot find key that definately in it. Probably so
    /// This one re-creates dictionary from scratch
    {
     var newdict = new Dictionary<TKey, TValue>();
     foreach (var kvp in dict)
      newdict.Add(kvp.Key, kvp.Value);
     dict = newdict;
    }


   #endregion
  }
 }
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.GUI;

namespace Den.Tools
{
 public static class Pathfinding
 {
  public class Factors
  {
   [Val("Incline")] public float incline = 0.25f;
   public float lowland = 0.25f;
   public float highland = 0;
   public float straighten = 0.25f;
   public float distance = 0.25f; //actually not used, but moving other factors to 0 will make spline as short as

   private float this[int num]
   {get{
    switch (num)
    {
     case 0: return incline;
```

```
  case 1: return lowland;
  case 2: return highland;
  case 3: return straighten;
  default: return distance;
 }
}
set{
 switch (num)
 {
  case 0: incline = value; break;
  case 1: lowland = value; break;
  case 2: highland = value; break;
  case 3: straighten = value; break;
  default: distance = value; break;
 }
}}



private void SetNormalizedVal (int num, float val)
{
 if (val<0) val = 0;
 if (val>1) val = 1;

 this[num] = 0;
 float sum = incline + lowland + highland + straighten + distance;
```

```
  for (int i=0; i<5; i++)
    this[i] = sum!=0 ? this[i]/sum * (1-val) : (1-val)/4;

  this[num] = val;
 }


 public float Incline {set{ SetNormalizedVal(0,value); }}
 public float Lowland {set{ SetNormalizedVal(1,value); }}
 public float Highland {set{ SetNormalizedVal(2,value); }}
 public float Straighten {set{ SetNormalizedVal(3,value); }}
 public float Distance {set{ SetNormalizedVal(4,value); }}
}


public class FixedList<T>
/// More handy in natives, faster in managed
{
 public T[] arr;
 public T count;

 public FixedList (int capacity) { arr = new T[capacity]; }
 public FixedList (T[] arr) { this.arr = arr; }
}
```

```
private static readonly Coord[] neigDiagonalFirst = new Coord[] {
 new Coord(-1,-1), new Coord(-1,1), new Coord(1,1), new Coord(1,-1),
 new Coord(0,-1), new Coord(-1,0), new Coord(0,1), new Coord(1,0) };

private static readonly Coord[] neigLineFirst = new Coord[] {
 new Coord(0,-1), new Coord(-1,0), new Coord(0,1), new Coord(1,0),
 new Coord(-1,-1), new Coord(-1,1), new Coord(1,1), new Coord(1,-1) };

private static readonly bool[] directionRnd = new bool[] {true, false, false, false, true, false, true, false, true
 true, true, true, false, true, true, true, false, true, false, true, true, false, false, false, false, false, false, fals
 false, false, true, true, true, false, false, true, true, true, true, false, false, false, true, false, false, false, fal
 true, false, true, false, true, false, true, false, true, false, false, true, false, true, true, false, false, true, fals
 true, true, true, false, true, true, false, true, true, false, false, false, false, false, true, false, true, false, true
 false, true, true, false, false, true, false, false, true, true, false, false, false, false, true};


public static float CalcWeight (Coord coord, Coord dir, MatrixWorld heights, Matrix mask, Factors factors)
/// returns 0-infinity range, where 0 is no friction, 1 is the standard step, infinity is impassable
{
 CoordRect rect = heights != null ? heights.rect : mask.rect;
 int pos = (coord.z-rect.offset.z)*rect.size.x + coord.x - rect.offset.x;

 int nPos = pos + rect.size.x*dir.z + dir.x;

 float diagonal = 1;
 if (dir.x*dir.z != 0) diagonal = 1.414213562373f;
```

```
float distanceFactor = diagonal;


float maskFactor = 1;
if (mask != null)
{
 maskFactor = mask.arr[nPos]; //range 0-1 (0 is standard step, 1 is impassable)


 if (maskFactor > 0.999f) maskFactor = float.MaxValue; //float.PositiveInfinity;
 else maskFactor = 1 / (1-mask.arr[nPos]) + 1;
 // 0-1 -> 1-Inf
}


float inclineFactor = 1;
//float highlandFactor = 1;
//float lowlandFactor = 1;
if (heights != null)
{
 float nHeight = heights.arr[nPos];


 float pixelSize = heights.worldSize.x/heights.rect.size.x;
 float elevation = nHeight - heights.arr[pos]; //thisHeight;
 if (elevation < 0) elevation = -elevation;
 elevation *= heights.worldSize.y;
 elevation = elevation / (pixelSize*diagonal); //since diagonal is less steep
```

```
  inclineFactor = elevation / factors.incline;  //0 - 1_or_more, 0 is standard, 1 is impassable
  if (inclineFactor > 0.999f) inclineFactor = float.MaxValue;
  else inclineFactor = 1 / (1-inclineFactor);

  //inclineFactor = Mathf.Atan(elevation) / (Mathf.PI/2); //range 0-1 (less is more passable)
 }

 float directionFactor = 1; //directionRnd[nPos%directionRnd.Length] ? 1 : 0.999f;
 //adding epsilon random factor to avoid long "cornered" look when factors are equal

 //transforming
 float factor = maskFactor * distanceFactor * inclineFactor * directionFactor;
 return factor;// != 1 ? 1/(1-factor) : float.MaxValue;
}


public static void FillDirs (Matrix2D<Coord> dirs, Coord to, MatrixWorld heights, Matrix mask, Factors fact
 Matrix weights=null, FixedList<int> changedPoses=null, FixedList<int> newChangedPoses=null,
 int maxIterations = -1)
/// Using Dijkstra to calculate directions matrix
/// Re-writing weights, dirs and changedposes
/// Returns null if path could not be found (in manhattan dist * 2 cells)
{
 CoordRect rect = heights!=null ? heights.rect : mask.rect;
 Coord rectMin = rect.offset; Coord rectMax = rect.offset + rect.size;
```

```
if (weights == null) weights = new Matrix(rect);
if (dirs == null) dirs = new Matrix2D<Coord>(rect);

if (changedPoses == null) changedPoses = new FixedList<int>(capacity:10000);
if (newChangedPoses == null) newChangedPoses = new FixedList<int>(capacity:10000);

weights.Fill(float.MaxValue); //clearing arrays (migh be used for previous pathfinding)
weights[to] = 0;

dirs.Fill(new Coord());

changedPoses.arr[0] = rect.GetPos(to);
changedPoses.count = 1;

if (maxIterations<0) maxIterations = rect.size.x+rect.size.z;
Coord min = rect.Min; Coord max = rect.Max;
for (int i=0; i<maxIterations; i++)
{
 for (int c=0; c<changedPoses.count; c++)
 {
  int pos = changedPoses.arr[c];
  Coord coord = rect.GetCoord(pos);

  if (coord.x < rectMin.x  ||  coord.x > rectMax.x-1  ||
   coord.z < rectMin.z  ||  coord.z > rectMax.z-1) return;
```

```
 float weight = weights.arr[pos];

 for (int d=0; d<8; d++)
 {
  Coord nCoord = neigDiagonalFirst[d];

  if ((coord.x==rectMin.x && nCoord.x==-1) || (coord.x==rectMax.x-1 && nCoord.x==1) ||
   (coord.z==rectMin.z && nCoord.z==-1) || (coord.z==rectMax.z-1 && nCoord.z==1)) continue;

  int nPos = pos + rect.size.x*nCoord.z + nCoord.x;

  float nWeight = weights.arr[nPos];
  float nNewWeight = CalcWeight(coord, nCoord, heights, mask, factors) + weight;

  if (nNewWeight < nWeight-0.0001f)  //using an epsilon delta to avoid overwriting nearly equal values
  {
   weights.arr[nPos] = nNewWeight;
   dirs.arr[nPos] = nCoord;

   newChangedPoses.arr[newChangedPoses.count] = nPos;
   newChangedPoses.count++;
  }
 }
}

FixedList<int> tempList = changedPoses;
```

```
    changedPoses = newChangedPoses;
    newChangedPoses = tempList;
    newChangedPoses.count = 0;
   }
  }


  public static Coord[] DrawPath (Matrix2D<Coord> dirs, Coord from, Coord to)
  {
   List<Coord> path = new List<Coord>();
   path.Add(from);
   Coord curr = from;
   for (int i=0; i<1000; i++)
   {
    curr -= dirs[curr];
    path.Add(curr);

    if (curr == to) break;
   }

   return path.ToArray();
  }
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace Den.Tools
{
 [System.Serializable]
 public class PosTab : ICloneable
 {
//make child of Matrix2
//id is number in list

//2rect matrix
// - child of matrix
// - standard operators are map sized
// - worldRect rect, and GetWorldPoint fn

/*  [System.Serializable]
  public struct Pos
  {
   public float x;
   public float z;
   public float height;
   public float rotation;
   public float inclineX;
```

```
 public float inclineZ;
 public float size;
 public int type;
 public int id; //num to apply per-object random that does not depend of object coords. 0 if pos is null. Note
}*/

[System.Serializable]
public struct Cell
{
 public CoordRect rect;
 public Transition[] poses;  //SOMEDAY: list? (currently replacing list features)
 public int count;

 public int GetPosNum (float x, float z)
 {
  for (int i=0; i<count; i++)
  {
   //if (Vector2.FloatsEqual(poses[i].pos.x, x) && Vector2.FloatsEqual(poses[i].pos.z, z))
   float dx = poses[i].pos.x-x; if (dx < 0) dx = -dx;
   float dz = poses[i].pos.z-z; if (dz < 0) dz = -dz;

   if (dx < 0.0001f && dz < 0.0001f)
    return i;
  }
  return -1;
 }
```

```
}

public readonly CoordRect rect;
public readonly Vector3 pos;
public readonly Vector3 size;
public Matrix2D<Cell> cells;
public readonly int resolution; //number of cells
public readonly Coord cellSize;

public int totalCount = 0;
public int idCounter = 1; //always increases, not changes if pos removed


public PosTab (Vector3 pos, Vector3 size, int resolution)
{
 //if rect%resolution!=0 the last cell size will be lower than usual

 this.resolution = resolution;
 this.rect = new CoordRect(Coord.Round(pos), Coord.Round(size));
 this.pos = pos;
 this.size = size;

 cells = new Matrix2D<Cell>(resolution, resolution);

 cellSize = new Coord( Mathf.CeilToInt(1f*rect.size.x/resolution), Mathf.CeilToInt(1f*rect.size.z/resolution)
```

```
 for (int x=0; x<resolution; x++)
  for (int z=0; z<resolution; z++)
 {
  Cell cell = new Cell();
  cell.rect = new CoordRect(
   x*cellSize.x + rect.offset.x,
   z*cellSize.z + rect.offset.z,
   Mathf.Min(cellSize.x, Mathf.Max(0, rect.size.x - x*cellSize.x)),
   Mathf.Min(cellSize.z, Mathf.Max(0, rect.size.z - z*cellSize.z)) );
  cell.rect.offset.x = Mathf.Min(cell.rect.offset.x, rect.offset.x+rect.size.x);
  cell.rect.offset.z = Mathf.Min(cell.rect.offset.z, rect.offset.z+rect.size.z);
  cells[x,z] = cell;
 }
}


public PosTab Copy ()
{
 PosTab copy = new PosTab(pos, size, resolution);
 for (int c=0; c<cells.arr.Length; c++)
 {
  copy.cells.arr[c].count = cells.arr[c].count;
  copy.cells.arr[c].rect = cells.arr[c].rect;

  if (cells.arr[c].poses == null) continue;
  copy.cells.arr[c].poses = new Transition[cells.arr[c].poses.Length];
```

```
  Array.Copy(cells.arr[c].poses, copy.cells.arr[c].poses, cells.arr[c].poses.Length);
 }
 copy.totalCount = totalCount;
 copy.idCounter = idCounter;
 return copy;
}

public object Clone () //IClonable
 { return Copy(); }

private Coord GetCellCoord (float x, float z, bool throwExceptions=true)
{
 int ix = (int)((x-rect.offset.x)/cellSize.x);
 int iz = (int)((z-rect.offset.z)/cellSize.z); //no need to process negative values

 if (throwExceptions && (ix > cells.rect.size.x || iz > cells.rect.size.z)) throw new Exception("Out of cells ra

 return new Coord(ix, iz);
}

private int GetCellNum (float x, float z, bool throwExceptions=true)
{
 int ix = (int)((x-rect.offset.x)/cellSize.x);
 int iz = (int)((z-rect.offset.z)/cellSize.z); //no need to process negative values

 if (throwExceptions && (ix > cells.rect.size.x || iz > cells.rect.size.z)) throw new Exception("Out of cells ra
```

```
 int n = iz*cells.rect.size.x + ix;

 if (throwExceptions && (n < 0)) throw new Exception("Could not find object at coord " + x + "," + z);

 return n;

}



public void Add (PosTab tab)
/// Combines two hash sets in current
{
 foreach (Transition trs in tab.All())
  Add(trs);
}



public void Add (float x, float z)
{
 Transition trs = new Transition(x,z);
 Add(trs);
}



public void Add (Transition trs)
```

```
{
 //checking in range
 if (!rect.Contains(trs.pos)) return;

 idCounter++;
 //if (idCounter > 2147000000) idCounter = 1;
 trs.id = idCounter;
 if (trs.hash == 0) //if hash not defined
  trs.hash = trs.id;

 int n = GetCellNum(trs.pos.x, trs.pos.z);

 //creating poses array
 if (cells.arr[n].poses == null) cells.arr[n].poses = new Transition[1];

 //resizing poses array
 if (cells.arr[n].poses.Length == cells.arr[n].count)
 {
  Transition[] newPoses = new Transition[cells.arr[n].count*4];
  Array.Copy(cells.arr[n].poses, newPoses, cells.arr[n].count);
  cells.arr[n].poses = newPoses;
 }

 //adding to array
 cells.arr[n].poses[ cells.arr[n].count ] = trs;
```

```
 cells.arr[n].count++;
 totalCount++;
}

public void Add (List<Transition> transitions)
{
 for (int i=0; i<transitions.Count; i++)
  Add(transitions[i]);
}


public void Add (TransitionsList transitions)
{
 for (int t=0; t<transitions.count; t++)
  Add(transitions.arr[t]);
}


public void Remove (int cellNum, int posNum)
{
 //swapping given pos num with the last one
 cells.arr[cellNum].poses[posNum] = cells.arr[cellNum].poses[ cells.arr[cellNum].count-1 ];
 cells.arr[cellNum].poses[ cells.arr[cellNum].count-1 ].hash = 0;

 cells.arr[cellNum].count--;
 totalCount--;
```

```
 //shrinking array length
 if (cells.arr[cellNum].count == 0) cells.arr[cellNum].poses = null;
 else if (cells.arr[cellNum].count < cells.arr[cellNum].poses.Length / 2)
 {
  Transition[] newPoses = new Transition[cells.arr[cellNum].count];
  Array.Copy(cells.arr[cellNum].poses, newPoses, cells.arr[cellNum].count);
  cells.arr[cellNum].poses = newPoses;
 }
}

public void RemoveAt (float x, float z)
{
 int c = GetCellNum(x,z);
 int n = cells.arr[c].GetPosNum(x,z);

 if (n!=-1) Remove(c,n);
}

public void Move (int cellNum, int posNum, float newX, float newZ)
///Moves pos to the new coordinates, preserving all other data
{
 int newCellNum = GetCellNum(newX,newZ);

 //if moving withing same cell
 if (newCellNum == cellNum)
```

```
 {
  cells.arr[cellNum].poses[posNum].pos.x = newX;
  cells.arr[cellNum].poses[posNum].pos.z = newZ;

  bool inRange =  newX >= rect.offset.x &&
   newZ >= rect.offset.z &&
   newX < rect.offset.x+rect.size.x &&
   newZ < rect.offset.z+rect.size.z;
  if (!inRange) Remove(cellNum, posNum);
  //if (!inRange) throw new Exception("Pos out of range: " + newX + "," + newZ + " rect:" + rect.ToString());
 }

 //removing from other cell and adding to new
 else
 {
  Transition trn = cells.arr[cellNum].poses[posNum];
  Remove(cellNum, posNum);

  trn.pos.x = newX;
  trn.pos.z = newZ;

  Add(trn);
 }
}

public void GetAndMove (float oldX, float oldZ, float newX, float newZ)
```

```
{
 int c = GetCellNum(oldX,oldZ);
 int n = cells.arr[c].GetPosNum(oldX,oldZ);

 if (n < 0) throw new Exception("Could not find object at coord " + oldX + "," + oldZ + " cell num:" + c);

 Move(c,n, newX, newZ);
}

public bool Exists (float x, float z)
///Checks if there an object at specified coord. Mainly for test purpose
{
 int c = GetCellNum(x,z, throwExceptions:false);
 if (c > cells.arr.Length) return false;

 int n = cells.arr[c].GetPosNum(x,z);

 return n >= 0;
}

public void Flush ()
///Removes unused array tails, reducing PosTab size (up to 4 times)
{
 for (int c=0; c<cells.arr.Length; c++)
 {
  if (cells.arr[c].poses == null) continue;
```

```
  if (cells.arr[c].count == 0) cells.arr[c].poses = null;

  if (cells.arr[c].poses.Length > cells.arr[c].count)
  {
   Transition[] newPoses = new Transition[cells.arr[c].count];
   Array.Copy(cells.arr[c].poses, newPoses, cells.arr[c].count);
   cells.arr[c].poses = newPoses;
  }
 }
}


public Transition Closest (float x, float z, float minDist=0, float maxDist=2147000000, Predicate<Transition
///Finds the closest Pos to given x and z in all cells. Use minDist=epsilon to exclude self.
{
 //alternative way: keep a bool array of process cells, and increase rect each iteration (skipping processed

 float minDistSq = maxDist*maxDist;
 Transition closestPos = new Transition() { hash=0 };

 Coord center = GetCellCoord(x,z);

 //finding cell search limit
 int maxP = (int)(maxDist / cellSize.x * 1.42f + 1);
```

```
int cellsToBounds = center.x>center.z? center.x : center.z; //a _maximum_ distance from center to cells r
if (cells.rect.size.x-center.x > cellsToBounds) cellsToBounds = cells.rect.size.x-center.x;
if (cells.rect.size.z-center.z > cellsToBounds) cellsToBounds = cells.rect.size.z-center.z;

if (maxP > cellsToBounds) maxP = cellsToBounds;
maxP++;

int minP = (int)(minDist / cellSize.x * 0.7f);

//looking in perimeters
for (int p=0; p<maxP; p++)
{
 ClosestInPerimeter(ref minDistSq, ref closestPos, center, p, x,z,minDist,maxDist, filterFn);

 //if closest found at least - checking 2 perimeters more
 if (closestPos.hash != 0)
 {
  int maxPleft = (int)( (Mathf.Sqrt(minDistSq) / cellSize.x) * 0.3f);  //0.3 is 1-0.7071
  if (maxPleft < 2) maxPleft = 2;

  for (int p2=0; p2<=maxPleft; p2++)
   ClosestInPerimeter(ref minDistSq, ref closestPos, center, p+p2, x,z,minDist,maxDist, filterFn);

  break;
 }
}
```

```
 return closestPos;
}



public void ClosestInPerimeter (ref float minDistSq, ref Transition closestPos, Coord center, int perimSize
///finds the closest in a rectangular (square) perimeter. Just a helper for Closest. PerDist is the perimeter s
{
 if (perimSize == 0) //in current cell
 {
  Cell curCell = cells[center];
  for (int i=0; i<curCell.count; i++)
  {
   float curDistSq = (curCell.poses[i].pos.x-x)*(curCell.poses[i].pos.x-x) + (curCell.poses[i].pos.z-z)*(curCe
   if (curDistSq<minDistSq &&
    curDistSq>=minDist*minDist &&
    (filterFn==null || filterFn(curCell.poses[i])) )
     { minDistSq=curDistSq; closestPos=curCell.poses[i]; }
  }
 }

 else //in perimeter
 {
  for (int s=0; s<perimSize; s++)
   foreach (Coord c in center.DistanceStep(s,perimSize))
  {
```

```
   //checking cell in range
   if (!(c.x >= cells.rect.offset.x && c.x < cells.rect.offset.x + cells.rect.size.x &&
         c.z >= cells.rect.offset.z && c.z < cells.rect.offset.z + cells.rect.size.z)) continue;


   Cell curCell = cells[c];
   for (int i=0; i<curCell.count; i++)
   {
    float curDistSq = (curCell.poses[i].pos.x-x)*(curCell.poses[i].pos.x-x) + (curCell.poses[i].pos.z-z)*(curCe
    if (curDistSq<minDistSq &&
     curDistSq>=minDist*minDist &&
     (filterFn==null || filterFn(curCell.poses[i])))
      { minDistSq=curDistSq; closestPos=curCell.poses[i]; }
   }
  }
 }
}



public Transition ClosestDebug (float x, float z, float minDist=0, float maxDist=20000000000)
///Finds closest iterating in all cells and objects. Hust fn to test Closest.
{
 float minDistSq = maxDist;
 Transition closestPos = new Transition() { hash=0 };


 for (int c=0; c<cells.arr.Length; c++)
 {
```

```
  Cell curCell = cells.arr[c];
  for (int i=0; i<curCell.count; i++)
  {
   float curDistSq = (curCell.poses[i].pos.x-x)*(curCell.poses[i].pos.x-x) + (curCell.poses[i].pos.z-z)*(curCe
   if (curDistSq<minDistSq && curDistSq>=minDist*minDist) { minDistSq=curDistSq; closestPos=curCell.p
  }
 }

 return closestPos;
}

public void TwoClosest (out Transition minTrs1, out Transition minTrs2)
/// Finds two most closest objects of all
{
 float minDist = float.MaxValue;
 minTrs1 = default;
 minTrs2 = default;
 foreach (Transition trs in All())
 {
  Transition closest = Closest(trs.pos.x, trs.pos.z, minDist:0.0001f);
  float dist = (trs.pos - closest.pos).sqrMagnitude;
  if (dist < minDist) { minDist = dist; minTrs1 = trs; minTrs2 = closest; }
 }
 //note that if t1 has t2 as closest does NOT mean that t1 is closest for t2.
}
```

```
#region MapMagic functions


 public static PosTab Combine (params PosTab[] posTabs)
 //NOTE: combine ids are not unique
 {
  if (posTabs.Length==0) return null;


  PosTab any = ArrayTools.Any(posTabs);
  if (any == null) return null;
  PosTab result = new PosTab(any.pos, any.size, any.resolution);


  for (int i=0; i<posTabs.Length; i++)
  {
   PosTab posTab = posTabs[i];
   if (posTab == null) continue;


   for (int c=0; c<posTab.cells.arr.Length; c++)
   {
    Cell cell = posTab.cells.arr[c];
    for (int p=0; p<cell.count; p++)
     result.Add(cell.poses[p]);
   }
  }
```

```
  return result;
 }



#endregion



public IEnumerable<Transition> All ()
{
 for (int c=0; c<cells.arr.Length; c++)
 {
  Cell cell = cells.arr[c];
  for (int i=0; i<cell.count; i++)
   yield return cell.poses[i];
 }
}

public Transition Any ()
{
 for (int c=0; c<cells.arr.Length; c++)
 {
  Cell cell = cells.arr[c];
  if (cell.count != 0)
   return cell.poses[0];
 }
 return default;
```

```
}

public int GetCountInRect (Vector3 rectPos, Vector3 rectSize)
{
 int count = 0;

 for (int c=0; c<cells.arr.Length; c++)
 {
  Cell cell = cells.arr[c];
  for (int i=0; i<cell.count; i++)
  {
   if (cell.poses[i].pos.x < rectPos.x || cell.poses[i].pos.x > rectPos.x+rectSize.x ||
    cell.poses[i].pos.z < rectPos.z || cell.poses[i].pos.z > rectPos.z+rectSize.z)
     continue;
   count ++;
  }
 }

 return count;
}


public IEnumerable<int> CellNumsInRect(Vector2 min, Vector2 max, bool inCenter=true) //obsolete?
{
 int minX = (int)((min.x-rect.offset.x) / cellSize.x);
 int minY = (int)((min.y-rect.offset.z) / cellSize.z);
```

```
 int maxX = (int)((max.x-rect.offset.x) / cellSize.x);
 int maxY = (int)((max.y-rect.offset.z) / cellSize.z);

 minX = Mathf.Max(0, minX); minY = Mathf.Max(0, minY);
 maxX = Mathf.Min(resolution-1, maxX); maxY = Mathf.Min(resolution-1, maxY);

 //processing all the rect
 if (inCenter)
  for (int x=minX; x<=maxX; x++)
   for (int y=minY; y<=maxY; y++)
    yield return y*resolution + x;

 //borders only
 else
 {
  for (int x=minX; x<=maxX; x++) { yield return minY*resolution + x; yield return maxY*resolution + x; }
  for (int y=minY; y<=maxY; y++) { yield return y*resolution + minX; yield return y*resolution + maxX; }
 }
}

public CoordRect CellsWithinBounds (Vector2 min, Vector2 max)
{
 CoordRect rect = new CoordRect();
 min.x-=this.rect.offset.x;  if (min.x<0) min.x--;  rect.offset.x = (int)(float)(min.x/cellSize.x);
 min.y-=this.rect.offset.z;  if (min.y<0) min.y--; rect.offset.z = (int)(float)(min.y/cellSize.z);
 max.x-=this.rect.offset.x;  if (max.x<0) max.x--; rect.MaxX = (int)(float)(max.x/cellSize.x + 1f);
```

```
 max.y-=this.rect.offset.z;  if (max.y<0) max.y--; rect.MaxZ = (int)(float)(max.y/cellSize.z + 1f);


 rect = CoordRect.Intersected(rect, new CoordRect(0,0,resolution,resolution));


 rect.Clamp(new Coord(0,0), new Coord(resolution,resolution));  //original resolution-1


 return rect;
}


public void RemoveObjsInRange(float posX, float posZ, float range)
{
 Rect rect = new Rect(posX-range, posZ-range, range*2, range*2);
 rect = CoordinatesExtensions.Intersect(rect, this.rect);


 foreach (int c in CellNumsInRect(rect.min, rect.max))
 {
  for (int p=cells.arr[c].count-1; p>=0; p--)
  {
   float distSq = (cells.arr[c].poses[p].pos.x-posX)*(cells.arr[c].poses[p].pos.x-posX) + (cells.arr[c].poses[p]
   if (distSq < range*range) Remove(c,p);
  }
 }
}


public bool IsAnyObjInRange(float posX, float posZ, float range)
{
```

```
 Vector2 min = new Vector2(posX-range, posZ-range);
 Vector2 max = new Vector2(posX+range, posZ+range);

 //foreach (int c in CellNumsInRect(min,max)) {
 CoordRect cellsCoords = CellsWithinBounds(min,max);
 Coord cMin = cellsCoords.Min; Coord cMax = cellsCoords.Max;
 for (int cx=cMin.x; cx<cMax.x; cx++)
  for (int cz=cMin.z; cz<cMax.z; cz++)
 {
  int c = cz*resolution + cx;

  for (int p=cells.arr[c].count-1; p>=0; p--)
  {
   float distSq = (cells.arr[c].poses[p].pos.x-posX)*(cells.arr[c].poses[p].pos.x-posX) + (cells.arr[c].poses[p]
   if (distSq < range*range) return true;
  }
 }
 return false;
}


public List<Transition> ToList ()
{
 List<Transition> transitions = new List<Transition>();

 for (int c=0; c<cells.arr.Length; c++)
```

```
  {
   Cell cell = cells.arr[c];
   transitions.AddRange(cell.poses);
  }

  return transitions;
 }


 public TransitionsList ToTransitionsList ()
 {
  TransitionsList trns = new TransitionsList(); //capacity totalCount

  for (int c=0; c<cells.arr.Length; c++)
  {
   Cell cell = cells.arr[c];
   for (int i=0; i<cell.count; i++)
    trns.Add(cells.arr[c].poses[i]);
  }

  return trns;
 }
}
}
```

```
ï»¿using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using System;
using System.Runtime.InteropServices;
using System.Runtime.Serialization;

namespace Den.Tools
{
 [Serializable]
 public static class Serializer
 {
  public interface IAlternativeType
  {
   Type AlternativeSerializationType { get; }
  }

  public interface ICustomSerialization
  {
   void PostprocessAfterSerialize (Object serObj, Dictionary<object,Object> allSerialized);
   void PreprocessBeforeDeserialize (Object serObj, Object[] allSerialized, object[] allDeserialized);
  }

  public sealed class NoCopyAttribute : Attribute { }
```

```
[Serializable]
[StructLayout(LayoutKind.Explicit, Size=16)]
public struct Value
{
 [SerializeField][FieldOffset(0)] public byte t;
 [SerializeField][FieldOffset(8)] long v; //the only field serialized. Otherwise all values will be written

 [NonSerialized][FieldOffset(8)] bool boolVal;
 [NonSerialized][FieldOffset(8)] byte byteVal;
 [NonSerialized][FieldOffset(8)] sbyte sbyteVal;
 [NonSerialized][FieldOffset(8)] char charVal;
 [NonSerialized][FieldOffset(8)] decimal decimalVal;
 [NonSerialized][FieldOffset(8)] double doubleVal;
 [NonSerialized][FieldOffset(8)] float floatVal;
 [NonSerialized][FieldOffset(8)] int intVal;
 [NonSerialized][FieldOffset(8)] uint uintVal;
 [NonSerialized][FieldOffset(8)] long longVal;
 [NonSerialized][FieldOffset(8)] ulong ulongVal;
 [NonSerialized][FieldOffset(8)] short shortVal;
 [NonSerialized][FieldOffset(8)] ushort ushortVal;

 [NonSerialized][FieldOffset(8)] int reference; //same as intVal

 //reference t is 255
```

```
//using value as a reference int number
public static implicit operator Value(int i) { return new Value() { t=255, intVal=i }; }
public static implicit operator int(Value v)
{
 if (v.t != 255)
  throw new Exception("Value t:" + v.t  + " v:" + v.v + " is used like a reference");
 return v.intVal;
}


//using as an universal value
public void Set (object obj, Type type)
{
 if (type == typeof(int))  { intVal = (int)obj;  t = 1; }
 else if (type == typeof(float)) { floatVal = (float)obj; t = 2; }
 else if (type == typeof(bool)) { boolVal = (bool)obj; t = 3; }
 else if (type == typeof(byte)) { byteVal = (byte)obj; t = 4; }
 else if (type == typeof(char)) { charVal = (char)obj; t = 5; }
 else if (type == typeof(uint)) { uintVal = (uint)obj; t = 6; }
 else if (type == typeof(sbyte)) { sbyteVal = (sbyte)obj; t = 7; }
 else if (type == typeof(decimal)) { decimalVal = (decimal)obj; t = 8; }
 else if (type == typeof(double)){ doubleVal = (double)obj; t = 9; }
 else if (type == typeof(long)) { longVal = (long)obj; t = 10; }
 else if (type == typeof(ulong)) { ulongVal = (ulong)obj; t = 11; }
 else if (type == typeof(short)) { shortVal = (short)obj; t = 12; }
 else if (type == typeof(ushort)){ ushortVal = (ushort)obj; t = 13; }
```

```
 else throw new Exception("Could not set value for " + type);
}

public object Get (Type type) // use stored type instead
{
 if (type == typeof(bool)) return boolVal;
 else if (type == typeof(byte)) return byteVal;
 else if (type == typeof(sbyte)) return sbyteVal;
 else if (type == typeof(char)) return charVal;
 else if (type == typeof(decimal)) return decimalVal;
 else if (type == typeof(double)) return doubleVal;
 else if (type == typeof(float)) return floatVal;
 else if (type == typeof(int)) return intVal;
 else if (type == typeof(uint)) return uintVal;
 else if (type == typeof(long)) return longVal;
 else if (type == typeof(ulong)) return ulongVal;
 else if (type == typeof(short)) return shortVal;
 else if (type == typeof(ushort)) return ushortVal;
 else throw new Exception("Could not get value for " + type);
}

public object Get () // use stored type instead
{
 switch (t)
 {
```

```
  case 1: return intVal;
  case 2: return floatVal;
  case 3: return boolVal;
  case 4: return byteVal;
  case 5: return charVal;
  case 6: return uintVal;
  case 7: return sbyteVal;
  case 8: return decimalVal;
  case 9: return doubleVal;
  case 10: return longVal;
  case 11: return ulongVal;
  case 12: return shortVal;
  case 13: return ushortVal;
  default: throw new Exception("Could not get value for " + t);
 }
}

public bool CheckType (Type type)
{
 if (t==0) return false;

 switch (t)
 {
  case 1: return type==typeof(int);
  case 2: return type==typeof(float);
  case 3: return type==typeof(bool);
```

```
   case 4: return type==typeof(byte);
   case 5: return type==typeof(char);
   case 6: return type==typeof(uint);
   case 7: return type==typeof(sbyte);
   case 8: return type==typeof(decimal);
   case 9: return type==typeof(double);
   case 10: return type==typeof(long);
   case 11: return type==typeof(ulong);
   case 12: return type==typeof(short);
   case 13: return type==typeof(ushort);
   case 255: return !type.IsPrimitive;
   default: throw new Exception("Could not get value for " + t);
  }
 }



 public override bool Equals (object obj)
 {
  if (!(obj is Value)) return false;
  return t==((Value)obj).t && v==((Value)obj).v;
 }
 public override int GetHashCode () { return base.GetHashCode(); }


}
```

```
[Serializable]
public class Object
{
 public int refId = -1;  //number of this object in <object,Object> dictionary. -1 is null
 public string type = null;
 public string altType = null;
 public string[] fields = null;
 public Value[] values = null;
 public string special = null; //to serialize strings, Types, FieldInfos
 public UnityEngine.Object uniObj = null; //to serialize Unity Objects

 public static Object Null {get{ return new Object() { refId=-1, type=null }; }}
}




public static Object[] Serialize (object obj, Action<object,Object> onAfterSerialize=null)
/// Stores objects in Object[] array
{
 Dictionary<object,Object> serializedDict = new Dictionary<object, Object>();
 SerializeObject(obj, serializedDict, onAfterSerialize:onAfterSerialize);

 Object[] serialized = null;
 CopyObjectsToArray(serializedDict, ref serialized);

 return serialized;
```

```
}


public static object Deserialize (Object[] serialized, Action<Object> onBeforeDeserialize=null)
/// Reads objects from stored array
{
 if (serialized.Length == 0) return null;
 object[] deserialized = new object[serialized.Length];
 return DeserializeObject(0, serialized, deserialized, onBeforeDeserialize:onBeforeDeserialize);
}


public static object DeepCopy (object obj)
{
 Dictionary<object,Object> serializedDict = new Dictionary<object, Object>();
 SerializeObject(obj, serializedDict);

 Object[] serialized = null;
 CopyObjectsToArray(serializedDict, ref serialized);

 object[] deserialized = new object[serialized.Length];
 return DeserializeObject(0, serialized, deserialized);
}


public static object DeepCopyPreservingRefs (object obj, Dictionary<object,object> srcDstRefs)
```

```
{
 //serializing
 Dictionary<object,Object> serializedDict = new Dictionary<object, Object>();
 SerializeObject(obj, serializedDict);

 //manually copy to array with dst reused
 object[] sources = new object[serializedDict.Count]; //to append src-dst dict easily
 Object[] serialized = new Object[serializedDict.Count];
 object[] reused = new object[serializedDict.Count]; //the objects from dst that should be used

 foreach (var kvp in serializedDict)
 {
  object src = kvp.Key;
  Object ser = kvp.Value;
  srcDstRefs.TryGetValue(src, out object dst);

  if (serialized[ser.refId] != null)
   throw new Exception("Objects with the same id: " + ser.refId);

  sources[ser.refId] = src;
  serialized[ser.refId] = ser;
  reused[ser.refId] = dst;
 }

 //deserializing
 object[] deserialized = new object[serialized.Length];
```

```
 object result = DeserializeObject(0, serialized, deserialized, reused);


 //appending src-dst dict
 srcDstRefs.Clear(); //to get rid of unused references. Added later. Not sure if this line will break all, but it's
 for (int i=0; i<sources.Length; i++)
 {
  if (!srcDstRefs.ContainsKey(sources[i]))
   srcDstRefs.Add(sources[i], deserialized[i]);


  else if (srcDstRefs[sources[i]] != deserialized[i])
   throw new Exception("Deserialized and srcDstRefs mismatch");
 }



 return result;
}



private static void CopyObjectsToArray (Dictionary<object,Object> serializedDict, ref Object[] serialized)
/// Will copy Objects using their refIds (not randomly like standard dict.Values). Will check id duplicates btw
{
 if (serialized==null  ||  serialized.Length!=serializedDict.Count) serialized = new Object[serializedDict.Cou
 else { for (int i=0; i<serialized.Length; i++) serialized[i] = null; }


 foreach (var kvp in serializedDict)
 {
```

```
  Object serObj = kvp.Value;

  if (serialized[serObj.refId] != null)
   throw new Exception("Objects with the same id: " + serObj.refId);

  serialized[serObj.refId] = serObj;
 }
}

private static Object SerializeObject (object obj, Dictionary<object,Object> serialized, bool skipNoCopyAtt
///If skipNotCopyAttribute will skip all fields marked with NoCopy attribute
{
 //null
 if (obj == null)
  return Object.Null;

 //if (obj is UnityEngine.Object && (UnityEngine.Object)obj == (UnityEngine.Object)null)
 // return Object.Null;
 //in some cases treats other graph as null. Better skip nulls on deserialize

 //already serialized
 if (serialized.TryGetValue(obj, out Object serObj)) return serObj;

 //type ignored
 //if (obj is ) return Object.Null;
```

```
serObj = new Object();
serialized.Add(obj, serObj); //adding before calling any other Serialize to prevent infinite loops
serObj.refId = serialized.Count - 1;

Type type = obj.GetType();
serObj.type = type.AssemblyQualifiedName;

//string
if (type == typeof(string))
 serObj.special = (string)obj;

//guid
if (type == typeof(Guid))
 serObj.special = ((Guid)obj).ToString();

//unity object
//else if (type.IsSubclassOf(typeof(UnityEngine.Object))  &&  !type.IsSubclassOf(typeof(UnityEngine.Scrip
else if (type.IsSubclassOf(typeof(UnityEngine.Object)))
 serObj.uniObj = (UnityEngine.Object)obj;

//serializer
else if (type == typeof(Object[]))
 throw new Exception("Serializer is trying to serialize serializer objects. This is causing infinite loop.");

//reflections
```

```
else if (type.IsSubclassOf(typeof(MemberInfo)))
{
 if (type.IsSubclassOf(typeof(Type))) //could be MonoType
  serObj.special = ((Type)obj).AssemblyQualifiedName;

 else
 {
  MemberInfo mi = (MemberInfo)obj;
  serObj.special = mi.Name + ", " + mi.DeclaringType.AssemblyQualifiedName;
 }
}

//primitives (if they were assigned to object field)
else if (type.IsPrimitive)
{
 serObj.values = new Value[1];
 serObj.values[0].Set(obj,type);
}

//animation curve
else if (type == typeof(AnimationCurve))
{
 AnimationCurve curve = (AnimationCurve)obj;

 serObj.values = new Value[3];
 serObj.values[0] = SerializeObject(curve.keys, serialized, onAfterSerialize:onAfterSerialize).refId;
```

```
 serObj.values[1].Set((int)curve.preWrapMode, typeof(int));
 serObj.values[2].Set((int)curve.postWrapMode, typeof(int));
}

//array
else if (type.IsArray)
{
 Array array = (Array)obj;

 serObj.fields = null;
 serObj.values = new Value[array.Length];

 Type elementType = type.GetElementType();
 bool elementTypeIsPrimitive = elementType.IsPrimitive;

 for (int i=0; i<array.Length; i++)
 {
  object val = array.GetValue(i);

  Value serVal = new Value();
  if (elementTypeIsPrimitive) serVal.Set(val, elementType);
  else serVal = SerializeObject(val, serialized, onAfterSerialize:onAfterSerialize).refId;

  serObj.values[i] = serVal;
 }
}
```

```
//another more relable Unity Object null check
else if (type == typeof(UnityEngine.Object))
{
 FieldInfo ptrField = type.GetField("m_CachedPtr", BindingFlags.Public | BindingFlags.NonPublic | Bindin
 IntPtr ptr = (IntPtr)ptrField.GetValue(obj);
 if (ptr == IntPtr.Zero)
  return Object.Null;
}

//class/struct
else
{
 if (obj is ISerializationCallbackReceiver)
  ((ISerializationCallbackReceiver)obj).OnBeforeSerialize();

 if (obj is IAlternativeType altTypeObj)
  serObj.altType = altTypeObj.AlternativeSerializationType.AssemblyQualifiedName;

 FieldInfo[] fields = type.GetFields(BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instance)

 //special cases when derive from list - adding base class values
 if (obj is IList || obj is IDictionary)
   ArrayTools.Append(ref fields, type.BaseType.GetFields(BindingFlags.Public | BindingFlags.NonPublic

 List<FieldInfo> usedFields = new List<FieldInfo>();
```

```
for (int f=0; f<fields.Length; f++)
{
 FieldInfo field = fields[f];

 if (field.IsLiteral) continue; //leaving constant fields blank
 if (field.FieldType.IsPointer) continue; //skipping pointers (they make unity crash. Maybe require unsafe)
 if (field.IsNotSerialized) continue;

 if (skipNoCopyAttribute)
 {
  object[] noCopyAttributes = field.GetCustomAttributes(typeof(NoCopyAttribute), false);
  if (noCopyAttributes.Length != 0) continue;  //SOMEDAY: make NoCopy a serializer control attribute wi
 }

 usedFields.Add(field);
}
int usedFieldsCount = usedFields.Count;

serObj.fields = new string[usedFieldsCount];
serObj.values = new Value[usedFieldsCount];

for (int f=0; f<usedFieldsCount; f++)
{
 FieldInfo field = usedFields[f];

 serObj.fields[f] = field.Name;
```

```
    object val = field.GetValue(obj);

    Value serVal = new Value();
    if (field.FieldType.IsPrimitive) serVal.Set(val, field.FieldType);
    else serVal = SerializeObject(val, serialized, onAfterSerialize:onAfterSerialize).refId;

    serObj.values[f] = serVal;
   }

   if (obj is ICustomSerialization customObj)
    customObj.PostprocessAfterSerialize(serObj, serialized);
  }

  onAfterSerialize?.Invoke(obj, serObj);

  return serObj;
 }

 private static object DeserializeObject (int refId, Object[] serialized, object[] deserialized, object[] reuse = n
 /// Loading object from serialized list.
 /// Will try to get object from reuse at the same position first. Note that reuse exists before the serialization
 {
  //null
  if (refId < 0)
```

```
 return null;

//already deserialized
if (deserialized[refId] != null) return deserialized[refId];

Object serObj = serialized[refId];
onBeforeDeserialize?.Invoke(serObj);

if (serObj.type == null  ||  serObj.refId<0) //refid could be changed on onBeforeDeserialize
 return null;

Type type = Type.GetType(serObj.type);

//trying to load alternative type
if (type == null  &&  serObj.altType != null)
 type = Type.GetType(serObj.altType);

//if still no type - maybe it's from the other assembly?
if (type == null  &&  serObj.type.Contains(","))
{
 string shortName = serObj.type.Substring(0, serObj.type.IndexOf(','));
 type = Type.GetType(shortName);
 if (type == null)
 {
  foreach (Assembly ass in AppDomain.CurrentDomain.GetAssemblies())
  {
```

```
    type = ass.GetType(shortName);
    if (type!=null)
     break;
   }
  }
}

//string
if (type == typeof(string))
 return serObj.special;

//guid
if (type == typeof(Guid) && serObj.special.Length!=0)
 return Guid.Parse(serObj.special);

//trying to load from core module
#if !UNITY_2019_2_OR_NEWER
if (type == null)
 type = Type.GetType($"{serObj.type}, UnityEngine.CoreModule");
if (type == null)
 type = Type.GetType($"{serObj.type}, UnityEngine.TerrainModule");
if (type == null)
 type = Type.GetType($"{serObj.altType}, UnityEngine.CoreModule");
#endif

//not exist anymore
```

```
if (type == null)
{
 #if MM_DEBUG
  Debug.LogError("Could not find type for: " + serObj.type);
 return null;
 #else
  throw new Exception("Could not find type for: " + serObj.type);
 #endif
}

//unity object
//else if (type.IsSubclassOf(typeof(UnityEngine.Object))  &&  !type.IsSubclassOf(typeof(UnityEngine.Scrip
else if (type.IsSubclassOf(typeof(UnityEngine.Object)))
{
 //if (serObj.uniObj == (UnityEngine.Object)null) return null; //to avoid casting "null" to Texture2D or some
 //doesnt work in deserialize thread, so here is workaround:
 if (serObj.uniObj.GetType() == typeof(UnityEngine.Object))
  return null;
 //yet don't know a case when pure Object could be assigned, so considering it as null

 return serObj.uniObj;
}

//serializer

//reflections
```

```
else if (type.IsSubclassOf(typeof(MemberInfo)))
{
 if (type.IsSubclassOf(typeof(Type))) //could be MonoType
  return Type.GetType(serObj.special);

 else
 {
  int d = serObj.special.IndexOf(", ");
  string mn = serObj.special.Substring(0, d);
  string tn = serObj.special.Substring(d+2);

  Type mt = Type.GetType(tn);
  return mt.GetField(mn, BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instance);
 }
}

//primitives (if they were assigned to object field)
else if (type.IsPrimitive)
 return serObj.values[0].Get();

//animation curve
else if (type == typeof(AnimationCurve))
{
 AnimationCurve curve;
 if (reuse==null || reuse[refId]==null) curve = new AnimationCurve();
 else curve = (AnimationCurve)reuse[refId];
```

```
 deserialized[refId] = curve;

 curve.keys = (Keyframe[])DeserializeObject(serObj.values[0], serialized, deserialized, reuse, onBeforeD
 curve.preWrapMode = (WrapMode)serObj.values[1].Get();
 curve.postWrapMode = (WrapMode)serObj.values[2].Get();

 return curve;
}

//array
else if (type.IsArray)
{
 Type elementType = type.GetElementType();
 bool elementTypeIsPrimitive = elementType.IsPrimitive;

 if (serObj.values.Length>0 && !serObj.values[0].CheckType(elementType))
  throw new Exception("Value was saved as a reference, but loading as a primitive (or vice versa)");

 Array array;
 if (reuse==null || reuse[refId]==null)
  array = (Array)Activator.CreateInstance(type, serObj.values.Length);
 else array = (Array)reuse[refId];

 deserialized[refId] = array;
```

```
 for (int i=0; i<array.Length; i++)
 {
  Value serVal = serObj.values[i];

  if (elementTypeIsPrimitive)
   array.SetValue( serVal.Get(), i);

  else if (serVal >= 0)
  {
   object val = DeserializeObject(serVal, serialized, deserialized, reuse, onBeforeDeserialize);
   array.SetValue(val, i);
  }
 }

 return array;
}


//class/struct
else
{
 object obj;
 if (reuse==null || reuse[refId]==null)
  //obj = FormatterServices.GetUninitializedObject(type);  // will not set default values
  obj = Activator.CreateInstance(type); // require constructor. Key/Value collection doesnt have one. Dicti
 else obj = reuse[refId];
```

```
if (obj is ICustomSerialization customObj)
 customObj.PreprocessBeforeDeserialize(serObj, serialized, deserialized);

deserialized[refId] = obj;

if (serObj.values == null)
 return null;
 //happens on de-serializing null Unity objects

for (int v=0; v<serObj.values.Length; v++)
{
 FieldInfo field = type.GetField(serObj.fields[v], BindingFlags.Public | BindingFlags.NonPublic | BindingFl

 //special cases when derive from list - adding base class values
 if (field==null && (obj is IList || obj is IDictionary) )
  field = type.BaseType.GetField(serObj.fields[v], BindingFlags.Public | BindingFlags.NonPublic | Binding

 if (field==null) continue; //if new variable was set in code at this position

 if (field.IsNotSerialized) continue; //although it was serialized when data was saved, it could be not seria
 //if (!serObj.values[v].CheckType(field.FieldType))
 // throw new Exception("Value was saved as a reference, but loading as a primitive (or vice versa)");

 Value serVal = serObj.values[v];
```

```
   if (field.FieldType.IsPrimitive)
   {
    if (!serVal.CheckType(field.FieldType)) continue; //in case 'new' keywork of other type was used on valu
    field.SetValue(obj, serVal.Get());
   }

   else if (serVal >= 0) //if -1 then leaving null/default in field
   {
    object val = DeserializeObject(serVal, serialized, deserialized, reuse, onBeforeDeserialize);
    if (val!=null  &&  val.GetType() != field.FieldType  &&  !field.FieldType.IsAssignableFrom(val.GetType())
     { Debug.LogWarning($"Serializer: Could not convert {val.GetType()} to {field.FieldType}. Using default
    field.SetValue(obj, val);
   }
  }

  if (obj is ISerializationCallbackReceiver)
   ((ISerializationCallbackReceiver)obj).OnAfterDeserialize();

  return obj;
 }
}



public enum ClassMatch { None, ShouldMatch, ShouldDiffer }
public static bool CheckMatch (object src, object dst, HashSet<object> chkd, ClassMatch checkIfSameRe
/// Checks if all fields are equal to test the serialization/copy results
```

```
/// sameReference checks if classes AreSame
{
 if (chkd==null) chkd = new HashSet<object>();

 if (src==null && dst==null) return true;
 if (src==null && dst!=null) throw new Exception("Src is null while dst isn't"); //return false;
 if (src!=null && dst==null) throw new Exception("Dst is null while src isn't"); //return false;

 Type type = src.GetType();

 if (type.IsPrimitive) { if (!src.Equals(dst)) throw new Exception("Primitives not equal"); }
 else if (type == typeof(string)) { if (src!=dst) throw new Exception("String not equal"); }
 else if (type.IsSubclassOf(typeof(Type))) { if (src!=dst) throw new Exception("Types not equal"); }
 else if (type.IsSubclassOf(typeof(FieldInfo))) { if (src!=dst) throw new Exception("Field Infos not equal"); }
 else if (type.IsSubclassOf(typeof(UnityEngine.Object))) { if (src!=dst) throw new Exception("Unity objects
 else if (type == typeof(AnimationCurve))
 {
  if (checkIfSameReference==ClassMatch.ShouldDiffer && src==dst)
   throw new Exception("Value references match");

  if (checkIfSameReference==ClassMatch.ShouldMatch && src!=dst)
   throw new Exception("Value references differ");

  AnimationCurve srcCurve = (AnimationCurve)src;
  AnimationCurve dstCurve = (AnimationCurve)dst;
  if (srcCurve.keys.Length != dstCurve.keys.Length) throw new Exception("Anim Curve length differ"); //re
```

```
 for (int i=0; i<srcCurve.keys.Length; i++)
  if (srcCurve.keys[i].time != dstCurve.keys[i].time  ||  srcCurve.keys[i].value != dstCurve.keys[i].value) retu


 return true;
}
else if (type.IsArray || type.BaseType.IsArray)
{
 if (checkIfSameReference==ClassMatch.ShouldDiffer && src==dst)
  throw new Exception("Value references match");


 if (checkIfSameReference==ClassMatch.ShouldMatch && src!=dst)
  throw new Exception("Value references differ");


 Array srcArray = (Array)src;
 Array dstArray = (Array)dst;


 if (chkd.Contains(srcArray)) return true;
 chkd.Add(srcArray);


 if (srcArray.Length != dstArray.Length) return false;


 for (int i=0; i<dstArray.Length; i++)
  if (!CheckMatch(srcArray.GetValue(i), dstArray.GetValue(i), chkd, checkIfSameReference)) return false;


 return true;
}
```

```
else
{
 if (checkIfSameReference==ClassMatch.ShouldDiffer && src==dst)
  throw new Exception("Value references match");

 if (checkIfSameReference==ClassMatch.ShouldMatch && !src.Equals(dst))
  throw new Exception("Value references differ");

 if (chkd.Contains(src)) return true;
 chkd.Add(src);

 FieldInfo[] fields = type.GetFields(BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instance)
 for (int i=0; i<fields.Length; i++)
 {
  if (fields[i].IsLiteral) continue; //leaving constant fields blank
  if (fields[i].FieldType.IsPointer) continue; //skipping pointers (they make unity crash. Maybe require unsa
  if (fields[i].IsNotSerialized) continue;

  object srcVal = fields[i].GetValue(src);
  object dstVal = fields[i].GetValue(dst);

  bool match = CheckMatch(srcVal, dstVal, chkd, checkIfSameReference);
  if (!match) return false;
 }
 return true;
}
```

```
    return true;
  }
 }

}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;

namespace Den.Tools
{
 [System.Serializable]
 [StructLayout (LayoutKind.Sequential)] //to pass to native
 public struct Coord
 {
  public int x;
  public int z;

  public int this[int c] { get => c==0 ? x : z;  set { if (x==0) x=value; else z=value; } }

  public static bool operator > (Coord c1, Coord c2) { return c1.x>c2.x && c1.z>c2.z; }
  public static bool operator < (Coord c1, Coord c2) { return c1.x<c2.x && c1.z<c2.z; }
  public static bool operator == (Coord c1, Coord c2) { return c1.x==c2.x && c1.z==c2.z; }
  public static bool operator != (Coord c1, Coord c2) { return c1.x!=c2.x || c1.z!=c2.z; }
  public static Coord operator + (Coord c, int s) { return  new Coord(c.x+s, c.z+s); }
  public static Coord operator + (Coord c1, Coord c2) { return  new Coord(c1.x+c2.x, c1.z+c2.z); }
  public static Coord operator - (Coord c) { return  new Coord(-c.x, -c.z); }
  public static Coord operator - (Coord c, int s) { return  new Coord(c.x-s, c.z-s); }
  public static Coord operator - (Coord c1, Coord c2) { return  new Coord(c1.x-c2.x, c1.z-c2.z); }
```

```
public static Coord operator * (Coord c, int s) { return  new Coord(c.x*s, c.z*s); }
public static Vector2 operator * (Coord c, Vector2 s) { return  new Vector2(c.x*s.x, c.z*s.y); }
public static Vector3 operator * (Coord c, Vector3 s) { return  new Vector3(c.x*s.x, s.y, c.z*s.z); }
public static Coord operator * (Coord c1, Coord c2) { return  new Coord((int)(c1.x*c2.x), (int)(c1.z*c2.z)); }
public static Coord operator * (Coord c, float s) { return  new Coord((int)(c.x*s), (int)(c.z*s)); }
public static Coord operator / (Coord c, int s) { return  new Coord(c.x/s, c.z/s); }
public static Coord operator / (Coord c, float s) { return  new Coord((int)(c.x/s), (int)(c.z/s)); }

public override bool Equals(object obj) { if (obj is Coord co) return co.x==x && co.z==z; return false; }
public override int GetHashCode() {return x*10000000 + z;}

public int Minimal {get{ return x<z ? x : z; } }
public int Maximal {get{ return x>z ? x : z; } }
public int SqrMagnitude {get{ return x*x + z*z; } }
public float Magnitude {get{ return Mathf.Sqrt(x*x + z*z); } }

public Vector4 vector4 {get{ return new Vector4(x,0,z,0); } }
public Vector3 vector3 {get{ return new Vector3(x,0,z); } }
public Vector2 vector2 {get{ return new Vector2(x,z); } }
public Vector2D vector2d {get{ return new Vector2D(x,z); } }

public static explicit operator Coord(Vector2D v) => new Coord((int)v.x, (int)v.z); //no flooring, handling ne
public static explicit operator Vector2D(Coord c) => new Vector2D(c.x, c.z);
public static explicit operator Vector3(Coord c) => new Vector3(c.x, 0, c.z);
public static explicit operator Coord(int i) => new Coord(i, i);
```

```
public static Coord zero {get{ return new Coord(0,0); }}

public Coord (int x, int z) { this.x=x; this.z=z; }
public Coord (int x) { this.x=x; this.z=x; }

#region Cell Operations

 public static Coord PickCell (int ix, int iz, int cellRes)
 {
  int x = ix/cellRes;
  if (ix<0 && ix!=x*cellRes) x--;

  int z = iz/cellRes;
  if (iz<0 && iz!=z*cellRes) z--;

  return new Coord(x,z);
 }

 public static Coord PickCell (Coord c, int cellRes) { return PickCell(c.x, c.z, cellRes); }

 public static Coord PickCellByPos (float fx, float fz, float cellSize=1)
 {
  int x = (int)(fx/cellSize);
  if (fx<0 && fx!=x*cellSize) x--;

  int z = (int)(fz/cellSize);
```

```
 if (fz<0 && fz!=z*cellSize) z--;

 return new Coord (x,z);
}

 public static Coord PickCellByPos (Vector3 v, float cellSize=1) { return PickCellByPos(v.x, v.z, cellSize); }



#endregion

#region Rounding

 //use v/tileSize instead of special tileSize methods

 public static Coord Floor (Vector3 v)
 {
  if (v.x<0) v.x--;  if (v.z<0) v.z--;
  return new Coord((int)(float)v.x, (int)(float)v.z);
 }

 public static Coord Floor (Vector2D v)
 {
  if (v.x<0) v.x--;  if (v.z<0) v.z--;
  return new Coord((int)(float)v.x, (int)(float)v.z);
 }
```

```
public static Coord Floor (float x, float z)
{
 if (x<0) x--;  if (z<0) z--;
 return new Coord((int)(float)x, (int)(float)z);
}

public static Coord Ceil (Vector2D v)
/// Ceil is NOT Floor(v+1):
/// Math.Ceiling(-2f)=-2,  Math.Floor(-2f+1)=-1,  Math.Floor(-2f)+1=-1
{
 if (v.x<0) v.x--;  if (v.z<0) v.z--;
 return new Coord((int)(float)(v.x + 1f), (int)(float)(v.z + 1f));
}

public static Coord Ceil (float x, float z)
{
 if (x<0) x--;  if (z<0) z--;
 return new Coord((int)(float)(x + 1f), (int)(float)(z + 1f));
}

public static Coord Round (Vector3 v)
{
 if (v.x<0) v.x--;  if (v.z<0) v.z--;
 return new Coord((int)(float)(v.x + 0.5f), (int)(float)(v.z + 0.5f));
}
```

```
 public static Coord Round (Vector2D v)
 {
  if (v.x<0) v.x--;  if (v.z<0) v.z--;
  return new Coord((int)(float)(v.x + 0.5f), (int)(float)(v.z + 0.5f));
 }

 public static Coord Round (float x, float z)
 {
  if (x<0) x--;  if (z<0) z--;
  return new Coord((int)(float)(x + 0.5f), (int)(float)(z + 0.5f));
 }



#endregion

public void Clamp (int sizeX, int sizeZ)
{
 if (x > sizeX) x = sizeX;
 if (z > sizeZ) z = sizeZ;
}

public void ClampPositive ()
 { x = Mathf.Max(0,x); z = Mathf.Max(0,z); }

public void ClampByRect (CoordRect rect)
```

```
// Closest coordinate within rect
{
 if (x<rect.offset.x) x = rect.offset.x; if (x>=rect.offset.x+rect.size.x) x = rect.offset.x+rect.size.x-1;
 if (z<rect.offset.z) z = rect.offset.z; if (z>=rect.offset.z+rect.size.z) z = rect.offset.z+rect.size.z-1;
}

static public Coord Min (Coord c1, Coord c2)
{
 //return new Coord(Mathf.Min(c1.x,c2.x), Mathf.Min(c1.z,c2.z));
 int minX = c1.x<c2.x? c1.x : c2.x;
 int minZ = c1.z<c2.z? c1.z : c2.z;
 return new Coord(minX, minZ);
}
static public Coord Max (Coord c1, Coord c2)
{
 //return new Coord(Mathf.Max(c1.x,c2.x), Mathf.Max(c1.z,c2.z));
 int maxX = c1.x>c2.x? c1.x : c2.x;
 int maxZ = c1.z>c2.z? c1.z : c2.z;
 return new Coord(maxX, maxZ);

}

public Coord BaseFloor (int cellSize) //tested
{
 return new Coord(
  x>=0 ? x/cellSize : (x+1)/cellSize-1,
```

```
  z>=0 ? z/cellSize : (z+1)/cellSize-1 );
}



public override string ToString()
{
 return (base.ToString() + " x:" + x + " z:" + z);
}



public static float Distance (Coord c1, Coord c2)
/// Standard Euclidean distance
{
 int distX = c1.x - c2.x; //if (distX < 0) distX = -distX; //there should be a reason I've added this. Don't really
 int distZ = c1.z - c2.z; //if (distZ < 0) distZ = -distZ;
 return Mathf.Sqrt(distX*distX + distZ*distZ);
}

public static float DistanceSq (Coord c1, Coord c2)
{
 int distX = c1.x - c2.x; if (distX < 0) distX = -distX;
 int distZ = c1.z - c2.z; if (distZ < 0) distZ = -distZ;
 return distX*distX + distZ*distZ;
}

public static int DistanceAxisAligned (Coord c1, Coord c2)
```

```
/// Chebyshev Max distance
{
 int distX = c1.x - c2.x; if (distX < 0) distX = -distX;
 int distZ = c1.z - c2.z; if (distZ < 0) distZ = -distZ;
 return distX>distZ? distX : distZ;
}

public static int DistanceManhattan (Coord c1, Coord c2)
{
 int distX = c1.x - c2.x; if (distX < 0) distX = -distX;
 int distZ = c1.z - c2.z; if (distZ < 0) distZ = -distZ;
 return distX+distZ;
}

//TODO: test
public static int DistanceAxisAligned (Coord c, CoordRect rect) //NOT manhattan dist. offset and size are
{
 //finding x distance
 int distPosX = rect.offset.x - c.x;
 int distNegX = c.x - rect.offset.x - rect.size.x;

 int distX;
 if (distPosX >= 0) distX = distPosX;
 else if (distNegX >= 0) distX = distNegX;
 else distX = 0;
```

```
 //finding z distance
 int distPosZ = rect.offset.z - c.z;
 int distNegZ = c.z - rect.offset.z - rect.size.z;

 int distZ;
 if (distPosZ >= 0) distZ = distPosZ;
 else if (distNegZ >= 0) distZ = distNegZ;
 else distZ = 0;

 //returning the maximum(!) distance
 if (distX > distZ) return distX;
 else return distZ;
}

public static float DistanceAxisPriority (Coord c1, Coord c2)
/// Whole number is an axis (Chebyshev) max dist, and remains is a remoteness from axis. Useful to set p
{
 int distX = c1.x - c2.x; if (distX < 0) distX = -distX;
 int distZ = c1.z - c2.z; if (distZ < 0) distZ = -distZ;

 int max = distX>distZ? distX : distZ;
 int min = distX<distZ? distX : distZ;

 return max + 1f*min/(max+1);
}
```

```
public IEnumerable<Coord> DistanceStep (int i, int dist) //4+4 terrains, no need to use separetely
{
 yield return new Coord(x-i, z-dist);
 yield return new Coord(x-dist, z+i);
 yield return new Coord(x+i, z+dist);
 yield return new Coord(x+dist, z-i);

 yield return new Coord(x+i+1, z-dist);
 yield return new Coord(x-dist, z-i-1);
 yield return new Coord(x-i-1, z+dist);
 yield return new Coord(x+dist, z+i+1);
}

public IEnumerable<Coord> DistancePerimeter (int dist) //a circular square border sorted by distance
{
 for (int i=0; i<dist; i++)
  foreach (Coord c in DistanceStep(i,dist)) yield return c;
}

public IEnumerable<Coord> DistanceArea (int maxDist)
{
 yield return this;
 for (int i=0; i<maxDist; i++)
  foreach (Coord c in DistancePerimeter(i)) yield return c;
}
```

```
public IEnumerable<Coord> DistanceArea (CoordRect rect) //same as distance are, but clamped by rect
{
 int maxDist = Mathf.Max(x-rect.offset.x, rect.Max.x-x, z-rect.offset.z, rect.Max.z-z) + 1;

 if (rect.Contains(this)) yield return this;
 for (int i=0; i<maxDist; i++)
  foreach (Coord c in DistancePerimeter(i))
   if (rect.Contains(c)) yield return c;
}

public static IEnumerable<Coord> MultiDistanceArea (Coord[] coords, int maxDist)
{
 if (coords.Length==0) yield break;

 for (int c=0; c<coords.Length; c++) yield return coords[c];

 for (int dist=0; dist<maxDist; dist++)
  for (int i=0; i<dist; i++)
   for (int c=0; c<coords.Length; c++)
    foreach (Coord c2 in coords[c].DistanceStep(i,dist)) yield return c2;
}

public Vector3 ToVector3 (float cellSize) { return new Vector3(x*cellSize, 0, z*cellSize); }
public Vector2 ToVector2 (float cellSize) { return new Vector2(x*cellSize, z*cellSize); }
public Rect ToRect (float cellSize) { return new Rect(x*cellSize, z*cellSize, cellSize, cellSize); }
```

```
public CoordRect ToCoordRect (int cellSize) { return new CoordRect(x*cellSize, z*cellSize, cellSize, cellS

public float GetFalloff (Vector2D center, float radius, float hardness, int smooth=1)
/// Gets current pixel's falloff percent for stamps
/// Smooth is iterational
{
 float distSq = (x-center.x)*(x-center.x) + (z-center.z)*(z-center.z);
 if (distSq > radius*radius) return 0;
 if (distSq < radius*hardness * radius*hardness) return 1;

 float dist = Mathf.Sqrt(distSq);

 float hardRadius = radius*hardness;
 float fallof = 1 - (dist - hardRadius) / (radius - hardRadius);  //remaining dist / transition, inversed

 for (int s=0; s<smooth; s++)
  fallof = 3*fallof*fallof - 2*fallof*fallof*fallof;

 return fallof;
}

public float GetInterpolatedPercent (Vector2D pos)
/// Gets influence of the pos on current coord
/// Used instead of GetFallof when brush size is lower than one pixel
{
 float xp = pos.x - x;  if (xp<0) xp = -xp; if (xp>1) return 0;
```

```
 float zp = pos.z - z;  if (zp<0) zp = -zp; if (zp>1) return 0;
 return (1-xp)*(1-zp);
}

public float GetInterpolatedFalloff (Vector2D center, float radius, float hardness, int smooth=1)
/// Automatically switches between GetFalloff and GetInterpolatedPercent to get a neat right falloff of this
{
 if (radius > 2)
  return GetFalloff(center, radius, hardness, smooth);

 else if (radius > 1) return
  GetFalloff(center, radius, hardness, smooth) * ((radius-1)) +
  GetInterpolatedPercent(center) * (2-radius);

 else
  return GetInterpolatedPercent(center) * radius;
}

//serialization
public string Encode () { return "x=" + x + " z=" + z; }
public void Decode (string[] lineMembers) { x=(int)lineMembers[2].Parse(typeof(int)); z=(int)lineMembers[
}



[System.Serializable]
```

```
[StructLayout (LayoutKind.Sequential)] //to pass to native
public struct CoordRect
{
 public Coord offset;
 public Coord size;

 public enum TileMode { Clamp=0, Tile=1, PingPong=2 } //see Tile region

 //public int radius; //not related with size, because a clamped CoordRect should have non-changed radius

 public CoordRect (Coord offset, Coord size) { this.offset = offset; this.size = size; }
 public CoordRect (int offsetX, int offsetZ, int sizeX, int sizeZ) { this.offset = new Coord(offsetX,offsetZ); thi
 public CoordRect (float offsetX, float offsetZ, float sizeX, float sizeZ) { this.offset = new Coord((int)offsetX,
 public CoordRect (Rect r) { offset = new Coord((int)r.x, (int)r.y); size = new Coord((int)r.width, (int)r.height)
 public CoordRect (Coord center, int radius) { this.offset = center-radius; this.size = new Coord(radius*2,ra
 public CoordRect (Vector2D center, float radius)
 {
  Coord centerCoord = Coord.Round(center);
  int radiusInt = (int)(radius+1);
  this.offset = centerCoord-radiusInt;
  this.size = new Coord(radiusInt + 1 + radiusInt); //1 for the dimensions of the centerCoord tile
 }

 public Coord Max { get { return offset+size; } set { size = value-offset; } }
 public int MaxX { get { return offset.x+size.x; } set { size.x = value-offset.x; } }
 public int MaxZ { get { return offset.z+size.z; } set { size.z = value-offset.z; } }
```

```
public Coord Min { get { return offset; } set { offset = value; } }
public Coord Center { get { return offset + size/2; } }
public Vector3 CenterVector3 { get { return new Vector3(offset.x + size.x/2f, 0, offset.z + size.z/2f); } }
public int Count { get {return size.x*size.z;} }



public override bool Equals(object obj) { return base.Equals(obj); }
public override int GetHashCode() {return offset.x*100000000 + offset.z*1000000 + size.x*1000+size.z;}

public int GetPos (Coord c) { return (c.z-offset.z)*size.x + c.x - offset.x; }
public int GetPos (int x, int z) { return (z-offset.z)*size.x + x - offset.x; }

public Coord GetCoord (int pos)
{
 int z = pos/size.x + offset.z;
 int x = pos - (z-offset.z)*size.x + offset.x;
 return new Coord(x,z);
}

public static bool operator > (CoordRect c1, CoordRect c2) { return c1.size>c2.size; }
public static bool operator < (CoordRect c1, CoordRect c2) { return c1.size<c2.size; }
public static bool operator == (CoordRect c1, CoordRect c2) { return c1.offset==c2.offset && c1.size==c2.
public static bool operator != (CoordRect c1, CoordRect c2) { return c1.offset!=c2.offset || c1.size!=c2.size
public static CoordRect operator * (CoordRect c, int s) { return  new CoordRect(c.offset*s, c.size*s); }
public static CoordRect operator * (CoordRect c, float s) { return  new CoordRect(c.offset*s, c.size*s); }
public static CoordRect operator / (CoordRect c, int s) { return  new CoordRect(c.offset/s, c.size/s); }
```

```
public Vector4 vector4 {get{ return new Vector4(offset.x,offset.z,size.x,size.z); } }

public static explicit operator CoordRect(Vector4 vec) => new CoordRect((int)vec.x, (int)vec.y, (int)vec.z, (
public static explicit operator Vector4(CoordRect cr) => new Vector4(cr.offset.x, cr.offset.z, cr.size.x, cr.siz
public static explicit operator CoordRect(Rect r) => new CoordRect((int)r.x, (int)r.y, (int)r.width, (int)r.heigh
public static explicit operator Rect(CoordRect cr) => new Rect(cr.offset.x, cr.offset.z, cr.size.x, cr.size.z);

public void Expand (int v) { offset.x-=v; offset.z-=v; size.x+=v*2; size.z+=v*2; }
public CoordRect Expanded (int v) { return new CoordRect(offset.x-v, offset.z-v, size.x+v*2, size.z+v*2); }
public void Contract (int v) { offset.x+=v; offset.z+=v; size.x-=v*2; size.z-=v*2; }
public CoordRect Contracted (int v) { return new CoordRect(offset.x+v, offset.z+v, size.x-v*2, size.z-v*2);

public void Clamp (Coord min, Coord max)
{
 Coord oldMax = Max;
 offset = Coord.Max(min, offset);
 size = Coord.Min(max-offset, oldMax-offset);
 size.ClampPositive();
}

public void ClampLine (ref Coord from, ref Coord to)
/// changes from and to so that they are within rect
{
 if (this.Contains(from) && this.Contains(to)) return;
```

```
 Vector2 dir = (from-to).vector2.normalized;
}

public static CoordRect Intersected (CoordRect c1, CoordRect c2)
{
 c1.Clamp(c2.Min, c2.Max);
 return c1;
}

public static CoordRect Intersected (CoordRect c1, CoordRect c2, CoordRect c3)
/// Finds the minimum intersection of 3 rects
{
 c1.Clamp(c2.Min, c2.Max);
 c1.Clamp(c3.Min, c3.Max);
 return c1;
}

public static bool IsIntersecting (CoordRect c1, CoordRect c2)
{
 if (c2.Contains(c1.offset.x, c1.offset.z) || c2.Contains(c1.offset.x+c1.size.x, c1.offset.z) || c2.Contains(c1.c
 if (c1.Contains(c2.offset.x, c2.offset.z) || c1.Contains(c2.offset.x+c2.size.x, c2.offset.z) || c1.Contains(c2.c

 return false;
}

public static CoordRect Combined (CoordRect rect1, CoordRect rect2)
```

```
{
 Coord min = Coord.Min(rect1.offset, rect2.offset);
 Coord max = Coord.Max(rect1.Max, rect2.Max);
 return new CoordRect(min, max-min);
}

public static CoordRect Combined (CoordRect[] rects)
{
 Coord min=new Coord(2000000000, 2000000000); Coord max=new Coord(-2000000000, -2000000000)
 for (int i=0; i<rects.Length; i++)
 {
  if (rects[i].offset.x < min.x) min.x = rects[i].offset.x;
  if (rects[i].offset.z < min.z) min.z = rects[i].offset.z;
  if (rects[i].offset.x + rects[i].size.x > max.x) max.x = rects[i].offset.x + rects[i].size.x;
  if (rects[i].offset.z + rects[i].size.z > max.z) max.z = rects[i].offset.z + rects[i].size.z;
 }
 return new CoordRect(min, max-min);
}

public void Encapsulate (Coord coord)
/// Resizes this rect so that coord is included
{
 if (coord.x < offset.x) { size.x += offset.x-coord.x; offset.x = coord.x; }
 if (coord.x > offset.x+size.x) { size.x = coord.x-offset.x; }

 if (coord.z < offset.z) { size.z += offset.z-coord.z; offset.z = coord.z; }
```

```
 if (coord.z > offset.z+size.z) { size.z = coord.z-offset.z; }
}



public static CoordRect WorldToPixel (Vector2D worldPos, Vector2D worldSize, Vector2D pixelSize, bool
/// Converts world rect to 0-aligned pixel/grid rect
/// Can convert to pixels and cells as well
{
 if (pixelSize.x == 0  ||  pixelSize.z == 0)
  throw new Exception("Cell size is zero");

 Coord min; Coord max;
 if (inclusive)
 {
  min = new Coord(
   Mathf.FloorToInt(worldPos.x/pixelSize.x),
   Mathf.FloorToInt(worldPos.z/pixelSize.z) );
  max = new Coord(
   Mathf.CeilToInt((worldPos.x+worldSize.x)/pixelSize.x),
   Mathf.CeilToInt((worldPos.z+worldSize.z)/pixelSize.z) );
 }
 else
 {
  min = new Coord(
   Mathf.CeilToInt(worldPos.x/pixelSize.x),
   Mathf.CeilToInt(worldPos.z/pixelSize.z) );
```

```
  max = new Coord(
    Mathf.FloorToInt((worldPos.x+worldSize.x)/pixelSize.x),
    Mathf.FloorToInt((worldPos.z+worldSize.z)/pixelSize.z) );
 }

 return new CoordRect(min, max-min);
}




#region Contains

public bool Contains (Coord coord)
 /// Checking if coord is within coordrect
 {
  return (coord.x >= offset.x && coord.x < offset.x + size.x &&
     coord.z >= offset.z && coord.z < offset.z + size.z);
 }

 public bool Contains (int x, int z)
 {
  return (x- offset.x >= 0 && x- offset.x < size.x &&
     z- offset.z >= 0 && z- offset.z < size.z);
 }

 public bool Contains (float x, float z)
```

```
{
 return (x- offset.x >= 0 && x- offset.x < size.x &&
  z- offset.z >= 0 && z- offset.z < size.z);
}

public bool Contains (Vector2 pos)
{
 return (pos.x- offset.x >= 0 && pos.x- offset.x < size.x &&
   pos.y- offset.z >= 0 && pos.y- offset.z < size.z);
}

public bool Contains (Vector3 pos)
{
 return (pos.x- offset.x >= 0 && pos.x- offset.x < size.x &&
   pos.z- offset.z >= 0 && pos.z- offset.z < size.z);
}

public bool Contains (float x, float z, float margins)
/// Contracts the coordrect by margins and checks contains
{
 return (x- offset.x >= margins && x- offset.x < size.x-margins &&
   z- offset.z >= margins && z- offset.z < size.z-margins);
}

public bool Contains (CoordRect r) //tested
{
```

```
 return  r.offset.x >= offset.x && r.offset.x+r.size.x <= offset.x+size.x &&
   r.offset.z >= offset.z && r.offset.z+r.size.z <= offset.z+size.z;
}

public bool ContainsOrIntersects (CoordRect r) //tested
{
 return  r.offset.x > offset.x-r.size.x && r.offset.x+r.size.x < offset.x+size.x+r.size.x &&
   r.offset.z > offset.z-r.size.z && r.offset.z+r.size.z < offset.z+size.z+r.size.z;
}

#endregion



#region Tiling

public Coord Tile (Coord coord, TileMode tileMode)
/// Returns the corresponding coord within the matrix rect
{
 //transferring to zero-based coord
 coord.x -= offset.x;
 coord.z -= offset.z;

 switch (tileMode)
 {
  //case TileMode.Once:
  // if (coord.x < 0 || coord.x >= size.x) coord.x = -1;
```

```
// if (coord.z < 0 || coord.z >= size.z) coord.z = -1;
// break;

case TileMode.Clamp:
 if (coord.x < 0) coord.x = 0;
 if (coord.x >= size.x) coord.x = size.x - 1;
 if (coord.z < 0) coord.z = 0;
 if (coord.z >= size.z) coord.z = size.z - 1;
 break;

case TileMode.Tile:
 coord.x = coord.x % size.x;
 if (coord.x < 0) coord.x= size.x + coord.x;
 coord.z = coord.z % size.z;
 if (coord.z < 0) coord.z= size.z + coord.z;
 break;

case TileMode.PingPong:
 coord.x = coord.x % (size.x*2);
 if (coord.x < 0) coord.x = size.x*2 + coord.x;
 if (coord.x >= size.x) coord.x = size.x*2 - coord.x - 1;

 coord.z = coord.z % (size.z*2);
 if (coord.z<0) coord.z=size.z*2 + coord.z;
 if (coord.z>=size.z) coord.z = size.z*2 - coord.z - 1;
 break;
```

```
 }



 coord.x += offset.x;
 coord.z += offset.z;

 return coord;
}



public void Tile (ref Coord coord, TileMode tileMode)
/// Returns the corresponding coord within the matrix rect
{
 //transferring to zero-based coord
 coord.x -= offset.x;
 coord.z -= offset.z;

 switch (tileMode)
 {
  //case TileMode.Once:
  // if (coord.x < 0 || coord.x >= size.x) coord.x = -1;
  // if (coord.z < 0 || coord.z >= size.z) coord.z = -1;
  // break;

  case TileMode.Clamp:
   if (coord.x < 0) coord.x = 0;
```

```
  if (coord.x >= size.x) coord.x = size.x - 1;
  if (coord.z < 0) coord.z = 0;
  if (coord.z >= size.z) coord.z = size.z - 1;
  break;

 case TileMode.Tile:
  coord.x = coord.x % size.x;
  if (coord.x < 0) coord.x= size.x + coord.x;
  coord.z = coord.z % size.z;
  if (coord.z < 0) coord.z= size.z + coord.z;
  break;

 case TileMode.PingPong:
  coord.x = coord.x % (size.x*2);
  if (coord.x < 0) coord.x = size.x*2 + coord.x;
  if (coord.x >= size.x) coord.x = size.x*2 - coord.x - 1;

  coord.z = coord.z % (size.z*2);
  if (coord.z<0) coord.z=size.z*2 + coord.z;
  if (coord.z>=size.z) coord.z = size.z*2 - coord.z - 1;
  break;
}


coord.x += offset.x;
coord.z += offset.z;
```

```
}



/*public Vector2 Tile (Vector2 vec, TileMode tileMode)
/// Returns the corresponding coord within the matrix rect
{
 switch (tileMode)
 {
  case TileMode.Clamp | TileMode.Once: return TileClamp(vec);
  case TileMode.Tile: return TileRepeat(vec);
  case TileMode.PingPong: return TilePingPong(vec);
  default: return vec;
 }
}

private Coord TileClamp (Coord coord)
{
 if (coord.x < offset.x) coord.x = offset.x;
 if (coord.x >= offset.x + size.x) coord.x = offset.x + size.x - 1;

 if (coord.z < offset.z) coord.z = offset.z;
 if (coord.z >= offset.z + size.z) coord.z = offset.z + size.z - 1;

 return coord;
}
```

```
private Vector2 TileClamp (Vector2 vec)
{
 if (vec.x < offset.x) vec.x = offset.x;
 if (vec.x >= offset.x + size.x) vec.x = offset.x + size.x - 1;

 if (vec.y < offset.z) vec.y = offset.z;
 if (vec.y >= offset.z + size.z) vec.y = offset.z + size.z - 1;

 return vec;
}

private Coord TileRepeat (Coord coord)
{
 coord.x -= offset.x;
 coord.x = coord.x % size.x;
 if (coord.x < 0) coord.x= size.x + coord.x;
 coord.x += offset.x;

 coord.z -= offset.z;
 coord.z = coord.z % size.z;
 if (coord.z < 0) coord.z= size.z + coord.z;
 coord.z += offset.z;

 return coord;
}
```

```
private Vector2 TileRepeat (Vector2 vec)
{
 vec.x -= offset.x;
 vec.x = vec.x % size.x;
 if (vec.x < 0) vec.x= size.x + vec.x;
 vec.x += offset.x;

 vec.y -= offset.z;
 vec.y = vec.y % size.z;
 if (vec.y < 0) vec.y= size.z + vec.y;
 vec.y += offset.z;

 return vec;
}

private Coord TilePingPong (Coord coord)
{
 coord.x -= offset.x;
 coord.x = coord.x % (size.x*2);
 if (coord.x < 0) coord.x = size.x*2 + coord.x;
 if (coord.x >= size.x) coord.x = size.x*2 - coord.x - 1;
 coord.x += offset.x;

 coord.z -= offset.z;
 coord.z = coord.z % (size.z*2);
```

```
 if (coord.z<0) coord.z=size.z*2 + coord.z;
 if (coord.z>=size.z) coord.z = size.z*2 - coord.z - 1;
 coord.z += offset.z;

 return coord + offset;
}

private Vector2 TilePingPong (Vector2 vec)
{
 vec.x -= offset.x;
 vec.x = vec.x % (size.x*2);
 if (vec.x < 0) vec.x = size.x*2 + vec.x;
 if (vec.x >= size.x) vec.x = size.x*2 - vec.x - 1;
 vec.x += offset.x;

 vec.y -= offset.z;
 vec.y = vec.y % (size.z*2);
 if (vec.y<0) vec.y=size.z*2 + vec.y;
 if (vec.y>=size.z) vec.y = size.z*2 - vec.y - 1;
 vec.y += offset.z;

 return vec;
}*/

#endregion
```

```
public override string ToString()
{
 return (base.ToString() + ": offsetX:" + offset.x + " offsetZ:" + offset.z + " sizeX:" + size.x + " sizeZ:" + size
}

public void DrawGizmo ()
{
 #if UNITY_EDITOR
 Vector3 s = size.ToVector3(1);
 Vector3 o = offset.ToVector3(1);
 Gizmos.DrawWireCube(o + s/2, s);
 #endif
}


#region Obsolete

 [Obsolete]
 public static CoordRect PickIntersectingCells (CoordRect rect, int cellRes)
 {
  int rectMaxX = rect.offset.x+rect.size.x;
  int rectMaxZ = rect.offset.z+rect.size.z;

  int minX = rect.offset.x/cellRes; if (rect.offset.x<0 && rect.offset.x%cellRes!=0) minX--;
  int minZ = rect.offset.z/cellRes; if (rect.offset.z<0 && rect.offset.z%cellRes!=0) minZ--;
  int maxX = rectMaxX/cellRes; if (rectMaxX>=0 && rectMaxX%cellRes!=0) maxX++;
```

```
 int maxZ = rectMaxZ/cellRes; if (rectMaxZ>=0 && rectMaxZ%cellRes!=0) maxZ++;


 return new CoordRect (minX, minZ, maxX-minX, maxZ-minZ);
}
//public static CoordRect PickIntersectingCells (Coord center, int range, int cellRes=1) { return PickInters


[Obsolete]
public static CoordRect PickIntersectingCellsByPos (float rectMinX, float rectMinZ, float rectMaxX, float r
{
 int minX = (int)(rectMinX/cellSize); if (rectMinX<0 && rectMinX!=minX*cellSize) minX--;
 int minZ = (int)(rectMinZ/cellSize); if (rectMinZ<0 && rectMinZ!=minZ*cellSize) minZ--;
 int maxX = (int)(rectMaxX/cellSize); if (rectMaxX>=0 && rectMaxX!=maxX*cellSize) maxX++;
 int maxZ = (int)(rectMaxZ/cellSize); if (rectMaxZ>=0 && rectMaxZ!=maxZ*cellSize) maxZ++;


 return new CoordRect (minX, minZ, maxX-minX, maxZ-minZ);
}
[Obsolete] public static CoordRect PickIntersectingCellsByPos (Vector3 pos, float range, float cellSize=1)
[Obsolete] public static CoordRect PickIntersectingCellsByPos (Rect rect, float cellSize=1) { return PickIn



[Obsolete]
public CoordRect MapSized (int resolution)
{
 CoordRect tem = new CoordRect(
  Mathf.RoundToInt( offset.x / (1f * size.x / resolution)  ),
  Mathf.RoundToInt( offset.z / (1f * size.z / resolution) ),
```

```
  resolution,
  resolution );
 return tem;
}


[Obsolete]
public int GetPos (float x, float z)
///Rounds coordinates and gets value. To get interpolated value use GetInterpolated
{
 int ix = (int)(x+0.5f); if (x<1) ix--;
 int iz = (int)(z+0.5f); if (z<1) iz--;
 return (iz-offset.z)*size.x + ix - offset.x;
}


//serialization
[Obsolete] public string Encode () { return "offsetX=" + offset.x + " offsetZ=" + offset.z + " sizeX=" + size.x
[Obsolete] public void Decode (string[] lineMembers)
{
 offset.x=(int)lineMembers[2].Parse(typeof(int)); offset.z=(int)lineMembers[3].Parse(typeof(int));
 size.x=(int)lineMembers[4].Parse(typeof(int)); size.z=(int)lineMembers[5].Parse(typeof(int));
}


/*public Vector2 ToWorldspace (Coord coord, Rect worldRect)
```

```
{
 return new Vector2 ( 1f*(coord.x-offset.x)/size.x * worldRect.width + worldRect.x,
       1f*(coord.z-offset.z)/size.z * worldRect.height + worldRect.y);  //percentCoord*worldWidth + worldOff
}

public Coord ToLocalspace (Vector2 pos, Rect worldRect)
{
 return new Coord ( (int) ((pos.x-worldRect.x)/worldRect.width * size.x + offset.x),
       (int) ((pos.y-worldRect.y)/worldRect.height * size.z + offset.z) ); //percentPos*size + offset
}*/


/*public IEnumerable<Coord> Cells (int cellSize) //coordinates of the cells inside this rect
{
 //transforming to cell-space
 Coord min = offset/cellSize;
 Coord max = (Max-1)/cellSize + 1;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
 {
  yield return new Coord(x,z);
 }
}*/

#endregion
```

```
}


[System.Serializable]
[StructLayout (LayoutKind.Sequential)] //to pass to native
public struct Coord3D
{
 public int x;
 public int y;
 public int z;

 public static bool operator > (Coord3D c1, Coord3D c2) { return c1.x>c2.x && c1.y>c2.y && c1.z>c2.z; }
 public static bool operator < (Coord3D c1, Coord3D c2) { return c1.x<c2.x && c1.y<c2.y && c1.z<c2.z; }
 public static bool operator == (Coord3D c1, Coord3D c2) { return c1.x==c2.x && c1.y==c2.y && c1.z==c2.
 public static bool operator != (Coord3D c1, Coord3D c2) { return c1.x!=c2.x || c1.y!=c2.y || c1.z!=c2.z; }
 public static Coord3D operator + (Coord3D c, int s) { return  new Coord3D(c.x+s, c.y+s, c.z+s); }
 public static Coord3D operator + (Coord3D c1, Coord3D c2) { return  new Coord3D(c1.x+c2.x, c1.y+c2.y,
 public static Coord3D operator - (Coord3D c) { return  new Coord3D(-c.x, -c.y, -c.z); }
 public static Coord3D operator - (Coord3D c, int s) { return  new Coord3D(c.x-s, c.y-s, c.z-s); }
 public static Coord3D operator - (Coord3D c1, Coord3D c2) { return  new Coord3D(c1.x-c2.x, c1.y-c2.y, c
 public static Coord3D operator * (Coord3D c, int s) { return  new Coord3D(c.x*s, c.y*s, c.z*s); }
 public static Vector3 operator * (Coord3D c, Vector3 s) { return  new Vector3(c.x*s.x, c.y*s.y, c.z*s.z); }
 public static Coord3D operator * (Coord3D c1, Coord3D c2) { return  new Coord3D(c1.x*c2.x, c1.y*c2.y, c
 public static Coord3D operator * (Coord3D c, float s) { return  new Coord3D((int)(c.x*s), (int)(c.y*s), (int)(c
 public static Coord3D operator / (Coord3D c, int s) { return  new Coord3D(c.x/s, c.y/s, c.z/s); }
 public static Coord3D operator / (Coord3D c, float s) { return new Coord3D((int)(c.x/s), (int)(c.y/s), (int)(c.z
```

```
 public static readonly Coord3D up = new Coord3D(0,1,0); //TODO: could be changed externally
 public static readonly Coord3D down = new Coord3D(0,-1,0);
 public static readonly Coord3D front = new Coord3D(0,0,1);
 public static readonly Coord3D back = new Coord3D(0,0,-1);
 public static readonly Coord3D left = new Coord3D(-1,0,0);
 public static readonly Coord3D right = new Coord3D(1,0,0);

 public override bool Equals(object obj) { if (obj is Coord3D co) return co.x==x && co.y==y && co.z==z; ret
 public override int GetHashCode() {return x*1000000 + y*1000 + z;}

 public int Minimal {get{ int xz = x<z ? x : z; return xz<y ? xz : y; } }
 public int Maximal {get{ int xz = x>z ? x : z; return xz>y ? xz : y; } }
 public int SqrMagnitude {get{ return x*x + y*y + z*z; } }
 public float Magnitude {get{ return Mathf.Sqrt(x*x + y*y + z*z); } }

 public static explicit operator Coord3D(Vector3 v) => new Coord3D((int)v.x, (int)v.y, (int)v.z); //no flooring,
 public static explicit operator Vector3(Coord3D c) => new Vector3(c.x, c.y, c.z);

 public static Coord3D zero {get{ return new Coord3D(0,0,0); }}

 public Coord3D (int x, int y, int z) { this.x=x; this.y=y; this.z=z; }

 public override string ToString() => $"{base.ToString()} x:{x}, y{y}, z:{z}";
}
```

```
[Serializable]
[StructLayout (LayoutKind.Sequential)] //to pass to native
public struct Vector2D : IEquatable<Vector2D>
{
 public float x;
 public float z;

 public const float kEpsilon = 1E-05F;
 public const float kEpsilonNormalSqrt = 1E-15F;

 public float this[int c] { get => c==0 ? x : z;  set { if (x==0) x=value; else z=value; } }

 public Vector2D (float x, float z) { this.x=x; this.z=z; }
 public Vector2D (float v) { this.x=v; this.z=v; }

 public static readonly Vector2D zero = new Vector2D(0,0);
 public static readonly Vector2D one = new Vector2D(1,1);

 public static Vector2D Zero { get; } = new Vector2D(0f, 0f);
 public static Vector2D One { get; } = new Vector2D(1f, 1f);
 public static Vector2D PositiveInfinity { get; } = new Vector2D(float.PositiveInfinity, float.PositiveInfinity);
 public static Vector2D NegativeInfinity { get; } = new Vector2D(float.NegativeInfinity, float.NegativeInfinity)

 public float SqrMagnitude => x*x + z*z;
 public float Magnitude => (float)Math.Sqrt((double)(x*x + z*z));
```

```
public Vector2D Normalized
{get{
 float m = (float)Math.Sqrt((double)(x*x + z*z));
 return m>1E-05f ? new Vector2D(x/m, z/m) : new Vector2D(0,0);
}}

public void ClampPositive ()
{
 if (x<0) x=0;
 if (z<0) z=0;
}

public static float Dot (Vector2D lhs, Vector2D rhs)
{
 return lhs.x * rhs.x + lhs.z * rhs.z;
}

public static float Distance (Vector2D a, Vector2D b)
{
 float num = a.x - b.x;
 float num2 = a.z - b.z;
 return (float)Math.Sqrt((double)(num * num + num2 * num2));
}

public static Vector2D Lerp (Vector2D a, Vector2D b, float t)
{
```

```
 if (t>1) t=1;
 if (t<0) t=0;
 return new Vector2D(a.x + (b.x - a.x) * t, a.z + (b.z - a.z) * t);
}

public static Vector2D Min (Vector2D lhs, Vector2D rhs)
{
 return new Vector2D(lhs.x<rhs.x ? lhs.x : rhs.x, lhs.z<rhs.z ? lhs.z : rhs.z);
}

public static Vector2D Max (Vector2D lhs, Vector2D rhs)
{
 return new Vector2D(lhs.x>rhs.x ? lhs.x : rhs.x, lhs.z>rhs.z ? lhs.z : rhs.z);
}

public static Vector2D Normalize (Vector3 v)
{
 float m = (float)Math.Sqrt((double)(v.x*v.x + v.z*v.z));
 return m>1E-05f ? new Vector2D(v.x/m, v.z/m) : new Vector2D(0,0);
}

public void Normalize ()
{
 float m = (float)Math.Sqrt((double)(x*x + z*z));
 if (m>1E-05f) {x=x/m; z=z/m;}
 else {x=0; z=0;}
```

```
}

public override int GetHashCode () { return x.GetHashCode() ^ z.GetHashCode() << 2; }
public bool Equals (Vector2D other) { return x == other.x && z == other.z; }
public override bool Equals (object other)
{
 if (!(other is Vector2D otherV)) return false;
 return x == otherV.x && z == otherV.z;
}

public static (Vector2D,Vector2D) Intersected (Vector2D pos1, Vector2D size1, Vector2D pos2, Vector2D
/// Copy of CoordRect intersection
{
 Vector2D pos3 = Vector2D.Max(pos1, pos2);
 Vector2D size3 = Vector2D.Min(pos1+size1, pos2+size2) - pos3;
 size3.ClampPositive();
 return (pos3, size3);
}

public static bool Intersects (Vector2D pos1, Vector2D size1, Vector2D pos2, Vector2D size2)
/// Finds if two world rects intersecting
{
 Vector2D pos = Vector2D.Max(pos1, pos2);
 Vector2D size = Vector2D.Min(pos1+size1, pos2+size2) - pos;

 if (size.x > 0 && size.z > 0)
```

```
  return true;
 else
  return false;
}

public static bool Contains (Vector2D pos, Vector2D size, Vector2D pos2)
/// Checks if pos2 within pos-size rect
{
 return (pos2.x>pos.x && pos2.x<pos.x+size.x &&
   pos2.z>pos.z && pos2.z<pos.z+size.z);
}


public override string ToString() { return string.Format("({0:F1}, {1:F1})", x, z); }

public static Vector2D operator + (Vector2D a, Vector2D b) => new Vector2D(a.x + b.x, a.z + b.z);
public static Vector2D operator + (Vector2D a, float b) => new Vector2D(a.x + b, a.z + b);
public static Vector2D operator - (Vector2D a, Vector2D b) => new Vector2D(a.x - b.x, a.z - b.z);
public static Vector2D operator - (Vector2D a, float b) => new Vector2D(a.x-b, a.z-b);
public static Vector2D operator * (Vector2D a, Vector2D b) => new Vector2D(a.x * b.x, a.z * b.z);
public static Vector2D operator / (Vector2D a, Vector2D b) => new Vector2D(a.x / b.x, a.z / b.z);
public static Vector2D operator - (Vector2D a) => new Vector2D(0f - a.x, 0f - a.z);
public static Vector2D operator * (Vector2D a, float d) => new Vector2D(a.x * d, a.z * d);
public static Vector2D operator * (float d, Vector2D a) => new Vector2D(a.x * d, a.z * d);
public static Vector2D operator / (Vector2D a, float d) => new Vector2D(a.x / d, a.z / d);
public static Vector2D operator / (float d, Vector2D a) => new Vector2D(d / a.x, d / a.z);
```

```
 public static Vector3 operator * (Vector3 a, Vector2D b) => new Vector3(a.x*b.x, a.y, a.z*b.z);
 public static Vector3 operator / (Vector3 a, Vector2D b) => new Vector3(a.x/b.x, a.y, a.z/b.z);
 public static Vector3 operator + (Vector3 a, Vector2D b) => new Vector3(a.x+b.x, a.y, a.z+b.z);
 public static Vector3 operator - (Vector3 a, Vector2D b) => new Vector3(a.x-b.x, a.y, a.z-b.z);
 public static Vector3 operator * (Vector2D a, Vector3 b) => new Vector3(a.x*b.x, b.y, a.z*b.z);
 public static Vector3 operator / (Vector2D a, Vector3 b) => new Vector3(a.x/b.x, b.y, a.z/b.z);
 public static Vector3 operator + (Vector2D a, Vector3 b) => new Vector3(a.x+b.x, b.y, a.z+b.z);
 public static Vector3 operator - (Vector2D a, Vector3 b) => new Vector3(a.x-b.x, b.y, a.z-b.z);

 public static bool operator == (Vector2D lhs, Vector2D rhs)
 {
  float num = lhs.x - rhs.x;
  float num2 = lhs.z - rhs.z;
  return num * num + num2 * num2 < 9.99999944E-11f;
 }
 public static bool operator != (Vector2D lhs, Vector2D rhs) => !(lhs == rhs);

 public static explicit operator Vector2D(Vector3 v) => new Vector2D(v.x, v.z);
 public static explicit operator Vector3(Vector2D v) => new Vector3(v.x, 0f, v.z);
 public static explicit operator Vector2D(Vector2 v) => new Vector2D(v.x, v.y);
 public static explicit operator Vector2(Vector2D v) => new Vector3(v.x, v.z);
 public static explicit operator Vector2D(float v) => new Vector2D(v, v);

 public Coord RoundToCoord () { return new Coord( (int)(float)(x<0 ? x-1 : x + 0.5f), (int)(float)(z<0 ? z-1 : z
}
```

```
[System.Serializable]
public struct PosRect
{
 public Vector3 offset;
 public Vector3 size;



}
```

```
[System.Serializable]
[StructLayout(LayoutKind.Explicit)]
public struct StructArray
{
 [FieldOffset(0)] public byte b0;
 [FieldOffset(1)] public byte b1;
 [FieldOffset(2)] public byte b2;
 [FieldOffset(3)] public byte b3;
 [FieldOffset(4)] public byte b4;
 [FieldOffset(5)] public byte b5;
 [FieldOffset(6)] public byte b6;
 [FieldOffset(7)] public byte b7;
```

```
[FieldOffset(0)] private long l;


public const int length = 8;


public byte this[int n]
{
 get { return (byte)((l>>n*8) & 0b_1111_1111); }
 set
 {
  l &= ~(((long)0b_1111_1111) << n*8); //erasing previous val
  l |= ((long)value)<<n*8;  //writing new one
 }
}


public float GetFloat (int n)
{
 long val = (l>>n*8) & 0b_1111_1111;
 return val / 255f;
}


public void SetFloat (int n, float f)
{
 long val = (int)(f*255);
 l &= ~(((long)0b_1111_1111) << n*8);
 l |= val<<n*8;
}
```

```
}



public struct StructArrayExtended<T>
///Stores first 4 T as a struct, and others as ref array
{
 public T i0;
 public T i1;
 public T i2;
 public T i3;

 public int count;
 private T[] others;


 public StructArrayExtended (int capacity)
 {
  i0=default; i1=default; i2=default; i3=default;
  others = null;
  count=0;
  Capacity = capacity;
 }


 public T this[int n]
```

```
{
 get
 {
  switch (n)
  {
   case 0: return i0;
   case 1: return i1;
   case 2: return i2;
   case 3: return i3;
   default: return others[n-4];
  }
 }
 set
 {
  switch (n)
  {
   case 0: i0 = value; break;
   case 1: i1 = value; break;
   case 2: i2 = value; break;
   case 3: i3 = value; break;
   default: others[n-4] = value; break;
  }
 }
}
```

```
public int Capacity
{
 get{ return 4 + (others!=null ? others.Length : 0); }

 set{
  if (value <= 4)
  {
   if (others!=null)
    others = null;
  }

  else
  {
   if (others==null) others = new T[value-4];
   else ArrayTools.Resize(ref others, value-4);
  }
 }
}

public void Add (T item)
{
 switch (count)
 {
  case 0: i0 = item; break;
  case 1: i1 = item; break;
```

```
  case 2: i2 = item; break;
  case 3: i3 = item; break;
  default:
   if (others == null) others = new T[4];
   if (others.Length <= count-4) ArrayTools.Resize(ref others, others.Length*2);
   others[count-4] = item;
   break;
 }
 count++;
}


public void AddRange (StructArrayExtended<T> other)
{
 for (int i=0; i<other.count; i++)
  this.Add(other[i]);
}


public int FindIndex (T item)
{
 if (i0.Equals(item)) return 0;
 if (i1.Equals(item)) return 1;
 if (i2.Equals(item)) return 2;
 if (i3.Equals(item)) return 3;
```

```
  if (others != null)
   for (int i=0; i<others.Length; i++)
    if (others[i].Equals(item)) return i+4;

  return -1;
 }
}


[System.Serializable]
[StructLayout(LayoutKind.Explicit)]
public struct SemVer
{
 public enum PRT { Release=4, RC=3, Beta=2, Alpha=1 };

 [FieldOffset(0)] public byte major;
 [FieldOffset(1)] public byte minor;
 [FieldOffset(2)] public byte prt;
 [FieldOffset(3)] public byte patch;

 [FieldOffset(0)] private int hash; //could be negative
 [FieldOffset(0)] private uint summary;


 public SemVer (byte major, byte minor, PRT prt, byte patch)
 {
```

```
 this.hash = 0;
 this.summary = 0;

 this.major = major;
 this.minor = minor;
 this.prt = (byte)prt;
 this.patch = patch;
}

public SemVer (byte major, byte minor, byte patch) : this (major, minor, PRT.Release, patch) { }

public string PRTtoString
{get{
 switch (prt)
 {
  default: return "";
  case 3: return "RC";
  case 2: return "B";
  case 1: return "A";
 }
}}

public override string ToString () => $"{major}.{minor}.{PRTtoString}{patch}";

public bool Equals (SemVer obj) => summary == obj.summary;
public override bool Equals (object obj) => summary == ((SemVer)obj).summary;
```

```
  public override int GetHashCode () => hash;


  public static bool operator == (SemVer v1, SemVer v2) => v1.summary == v2.summary;
  public static bool operator != (SemVer v1, SemVer v2) => v1.summary != v2.summary;
  public static bool operator < (SemVer v1, SemVer v2) => v1.summary < v2.summary;
  public static bool operator > (SemVer v1, SemVer v2) => v1.summary > v2.summary;
  public static bool operator <= (SemVer v1, SemVer v2) => v1.summary <= v2.summary;
  public static bool operator >= (SemVer v1, SemVer v2) => v1.summary >= v2.summary;
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Profiling;

namespace Den.Tools
{

 public interface ITile
 {
  //Coord Coord { set; }
  //bool Pinned { set; }
  //int Distance { get; set; } //from deploy rects centers, in chunks
  bool IsNull { get; } //if main object was removed externally. Checking on ad, remove and deploy

  //static ITile Construct (object holder);
  void Move (Coord coord, float dist);
  void Dist (float dist);
  void Remove ();
 }

 public class TileManager<T> : ISerializationCallbackReceiver where T: ITile//, IEquatable<T>
 {
  public Dictionary<Coord,T> grid = new Dictionary<Coord,T>();
  public object gridLocker = new object();
```

```
public bool allowMove = false;

public bool generateInfinite = true;

public int generateRange = 2;

public int retainMargin = 1;


public bool genAroundMainCam = true;

public bool genAroundObjsTag = false;

public string genAroundTag = null;


[System.NonSerialized] protected Coord[] camCoords = null;

//[System.NonSerialized] protected CoordRect[] deployRects;  //used to find chunks difference and for Un



public T this[Coord coord]

{get{

 if (grid.TryGetValue(coord, out T t)) return t;

 else return default(T);

}}


public T this[int x, int z] {get{ return this[ new Coord(x,z) ]; }}



public bool Contains (Coord coord)

/// Checks if tile is contained in hash dictionary.

{
```

```
 return grid.ContainsKey(coord);
}



protected T ConstructTile (MonoBehaviour holder)
{
 return (T)typeof(T).GetMethod("Construct", System.Reflection.BindingFlags.Static | System.Reflection.Bi
 //TODO: make smth with it
}



public IEnumerable<T> Tiles ()
{
 foreach (KeyValuePair<Coord,T> kvp in grid)
  yield return kvp.Value;
}



public virtual T Closest ()
{
 float minDist = int.MaxValue;
 T minTile = default;

 foreach (var kvp in grid)
 {
  if (camCoords == null) return kvp.Value;
```

```
  Coord coord = kvp.Key;
  float dist = GetRemoteness(coord, camCoords);
  if (dist<minDist) { minDist=dist; minTile=kvp.Value; }
 }

 return minTile;
}



#region Per-frame/Update

 public void Update (Vector3 tileSize, Dictionary<Coord,T> pinned=null, MonoBehaviour holder=null, bool
 {
  Profiler.BeginSample("Remove Nulls");
  RemoveNulls(); //excluding removed objects
  Profiler.EndSample();

  Profiler.BeginSample("RefreshCamCoords");
  bool camCoordsChanged = RefreshCamCoords(tileSize.x, holder);
  if (!camCoordsChanged || camCoords.Length==0) { Profiler.EndSample(); return; }
  Profiler.EndSample();

  Profiler.BeginSample("Deploy");
```

```
 if (!distsOnly && generateInfinite) Deploy(camCoords, pinned:pinned, holder:holder);
 Profiler.EndSample();

 Profiler.BeginSample("ChangeDists");
 ChangeDists(camCoords);
 Profiler.EndSample();
}

public void ReDeploy (Vector3 tileSize, Dictionary<Coord,T> pinned=null, MonoBehaviour holder=null)
{
 RemoveNulls(); //excluding removed objects
 RefreshCamCoords(tileSize.x);
 Deploy(camCoords, pinned:pinned, holder:holder);
 ChangeDists(camCoords);
}

private bool RefreshCamCoords (float tileSize, MonoBehaviour holder=null)
/// Gets a list of camera (or tagged objects) positions. Uses a cached camPoses array. Returns true if ca
{
 bool coordsChanged = false;

 #if UNITY_EDITOR
 if (!UnityEditor.EditorApplication.isPlaying)
 {
```

```
 if (UnityEditor.SceneView.lastActiveSceneView?.camera==null || UnityEditor.SceneView.lastActiveScen
  camCoords = new Coord[0];

 else
 {
  Vector3 sceneCamPos = UnityEditor.SceneView.lastActiveSceneView.camera.transform.position;
  Coord sceneCamCoord = Coord.Floor(sceneCamPos.x/tileSize, sceneCamPos.z/tileSize);

  if (camCoords==null || camCoords.Length!=1) { camCoords = new Coord[1]; coordsChanged = true; }
  if (camCoords[0] != sceneCamCoord) { camCoords[0] = sceneCamCoord; coordsChanged = true; }
 }
}

else
#endif
{
 //finding objects with tag
 GameObject[] taggedObjects = null;
 if (genAroundObjsTag)
  taggedObjects = GameObject.FindGameObjectsWithTag(genAroundTag);

 //calculating cams array length and rescaling it
 int camsLength = 0;
 if (genAroundMainCam) camsLength++;
 if (taggedObjects !=null) camsLength += taggedObjects.Length;
```

```
if (camCoords == null || camsLength != camCoords.Length) { camCoords = new Coord[camsLength]; co


if (camsLength == 0)
 //throw new Exception("TileManager: No Camera in scene to generate tiles.");
 return coordsChanged;


//filling cams array
int counter = 0;
if (genAroundMainCam)
{
 Camera mainCam = Camera.main;
 if (mainCam == null) mainCam = GameObject.FindObjectOfType<Camera>(); //in case it was destroye
 if (mainCam != null) //if still no camera
 {
  Vector3 camPos = mainCam.transform.position;
  if (holder != null) camPos = holder.transform.InverseTransformPoint(camPos);


  Coord camCoord = Coord.Floor(camPos.x/tileSize, camPos.z/tileSize);
  if (camCoords[0] != camCoord) { camCoords[0] = camCoord; coordsChanged = true; }
  counter++;
 }
}
if (taggedObjects != null)
 for (int i=0; i<taggedObjects.Length; i++)
 {
  Vector3 objPos = taggedObjects[i].transform.position;
```

```
    if (holder != null) objPos = holder.transform.InverseTransformPoint(objPos);

    Coord objCoord = Coord.Floor(objPos.x/tileSize, objPos.z/tileSize);
    if (camCoords[i+counter] != objCoord) { camCoords[i+counter] = objCoord; coordsChanged = true; }
   }
  }

  return coordsChanged;
 }

#endregion



#region Deploy

 public virtual void ChangeDists (Coord[] camCoords)
 /// Fast deploy that changes distances only
 {
  foreach (var kvp in grid)
  {
   Coord coord = kvp.Key;
   T tile = kvp.Value;

   tile.Dist( GetRemoteness(coord, camCoords) );
  }
 }
```

```
public virtual void Deploy (Coord[] camCoords, Dictionary<Coord,T> pinned=null, MonoBehaviour holder
/// Creates all tiles within createRect, removes tiles outside removeRect. Tries to move tiles instead of cre
/// Note that all rects contain chunks, not world units
/// Holder is a parent object that called refresh, to parent created tiles
{
 CoordRect[] createRects = GetDeployRects(camCoords, generateRange);

 //it would be easier to create new grid and fill it then, but
 //no change should be made in original grid because of multithreading
 Dictionary<Coord,T> dstGrid = new Dictionary<Coord,T>();
 Dictionary<Coord,T> srcGrid = new Dictionary<Coord,T>(grid);

 //transferring pinned tiles to new grid
 Profiler.BeginSample("Transf Pin To New");
 if (pinned != null)
  foreach(KeyValuePair<Coord,T> kvp in pinned)
 {
  Coord coord = kvp.Key;
  T tile = kvp.Value;

  srcGrid.Remove(coord);
  dstGrid.Add(coord, tile);

  tile.Dist(GetRemoteness(coord, camCoords)); //calculating dist to every tile added to dstGrid
```

```
}
Profiler.EndSample();



//adding objects within create range + margin (on their respective coordinates)
Profiler.BeginSample("Adding Objs");
for (int r=0; r<createRects.Length; r++)
{
 CoordRect rect = createRects[r];
 //rect.Expand(retainMargin);
 Coord min = rect.Min-retainMargin; Coord max = rect.Max+retainMargin;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   Coord coord = new Coord(x,z);

   if (srcGrid.TryGetValue(coord, out T tile))
   {
    srcGrid.Remove(coord);
    dstGrid.Add(coord,tile);

    tile.Dist(GetRemoteness(coord, camCoords));
   }
  }
}
```

```
Profiler.EndSample();


//filling create rects empty areas with unused (or new) objects and moving them
Profiler.BeginSample("Fillin Empty");
Queue<T> pool = new Queue<T>(srcGrid.Values);
List<(T tile, Coord coord, float dist)> moved = new List<(T,Coord,float)>();
for (int r=0; r<createRects.Length; r++)
{
 CoordRect rect = createRects[r];
 Coord min = rect.Min; Coord max = rect.Max;


 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
 {
  Coord newCoord = new Coord(x,z);


  if (dstGrid.ContainsKey(newCoord)) continue;


  T tile;


  //moving
  if (pool.Count != 0  &&  allowMove)
  {
   //Coord oldCoord = srcGrid.AnyKey();
   //T tile = srcGrid[oldCoord];
   tile = pool.Dequeue();
```

```
  }

  //creating
  else
  {
   //Debug.Log("No tiles left. Creating. Coord:" + newCoord);
   Profiler.BeginSample("Construct Tile");
   tile = ConstructTile(holder);
   Profiler.EndSample();
  }

  dstGrid.Add(newCoord, tile);

  //tile.Move(newCoord, GetRemoteness(newCoord, camCoords)); //moving after according to their dista
  moved.Add( (tile, newCoord, GetRemoteness(newCoord, camCoords)) );
 }

 //HashSet<T> curChangedTiles = RelocateTiles(dstGrid, rect, pool, holder);
 //changedTiles.UnionWith(curChangedTiles);
}
Profiler.EndSample();

//calling remove fn on all other objs left (no need to remove from srcDict - just not including them in dst)
Profiler.BeginSample("Callin Remove");
while (pool.Count != 0)
{
```

```
 T tile = pool.Dequeue();
 tile.Remove();
}
Profiler.EndSample();

//calling Move function in order depending on remoteness
Profiler.BeginSample("Callin Move");
moved.Sort((x,y) =>
{
 float delta = x.dist-y.dist;
 if (delta > 0.00001f) return 1;
 else if (delta < -0.000001f) return -1;
 else return 0;
});

//assigning new grid and deployed rects
//this should be done before calling Move (moves calls MM welding, and welding reads grid)
lock (gridLocker)
 grid = dstGrid;

int movedCount = moved.Count;
for (int m=0; m<movedCount; m++)
 moved[m].tile.Move(moved[m].coord, moved[m].dist);
Profiler.EndSample();
}
```

```
#endregion

#region Helpers

 private static CoordRect[] GetDeployRects (Coord[] camCoords, int range)
 /// Converts each cam coord to chunk rect using the generate range
 {
  CoordRect[] deployRects = new CoordRect[camCoords.Length];

  for (int r=0; r<camCoords.Length; r++)
   deployRects[r] = new CoordRect(camCoords[r].x - range, camCoords[r].z - range, range*2 +1, range*2

  return deployRects;
 }



 protected static float GetRemoteness (Coord coord, Coord[] camCoords)
 /// Returns an axis/priority distance to the closest cam
 {
  float minDist = float.MaxValue;

  if (camCoords == null) return minDist;

  for (int r=0; r<camCoords.Length; r++)
  {
   float dist = Coord.DistanceAxisPriority(camCoords[r], coord);
```

```
  if (dist < minDist) minDist = dist;
 }

 return minDist;
}



public virtual void RemoveNulls ()
/// Removes tiles that were deleted externally from the collection
{
 List<Coord> removedCoords = null;

 foreach (KeyValuePair<Coord,T> kvp in grid)
 {
  T tile = kvp.Value;

  if (tile == null || tile.IsNull)
  {
   if (removedCoords == null) removedCoords = new List<Coord>(); //do not create list if there's nothing to
   removedCoords.Add(kvp.Key);
  }
 }

 if (removedCoords != null)
  foreach (Coord coord in removedCoords)
   grid.Remove(coord);
```

```
}


[Obsolete] private CoordRect WorldToChunksRect (CoordRect wrect, int size)
/// Not used anywhere, but contains a tested code just in case
{
 Coord cMin = new Coord(
  wrect.offset.x>=0 ? wrect.Min.x/size : (wrect.Min.x+1)/size-1,
  wrect.offset.z>=0 ? wrect.Min.z/size : (wrect.Min.z+1)/size-1 );

 Coord cMax = new Coord(
  wrect.offset.x+wrect.size.x>0 ? (wrect.offset.x+wrect.size.x-1)/size + 1 :  (wrect.offset.x+wrect.size.x)/siz
  wrect.offset.z+wrect.size.z>0 ? (wrect.offset.z+wrect.size.z-1)/size + 1 :  (wrect.offset.z+wrect.size.z)/siz
 //tested

 return new CoordRect(cMin, cMax-cMin);
}




#endregion


#region Serialization
//generics do not serialize. Derive to use it.
```

```
public T[] serializedTiles;
public Coord[] serializedCoords;


public virtual void OnBeforeSerialize ()
{
 if (serializedTiles == null || serializedTiles.Length != grid.Count) serializedTiles = new T[grid.Count];
 if (serializedCoords == null || serializedCoords.Length != grid.Count) serializedCoords = new Coord[grid.


 int counter = 0;
 foreach (var kvp in grid)
 {
  serializedTiles[counter] = kvp.Value;
  serializedCoords[counter] = kvp.Key;
  counter++;
 }
}


public virtual void OnAfterDeserialize ()
{
 Dictionary<Coord,T> newTiles = new Dictionary<Coord,T>();
 for (int i=0; i<serializedTiles.Length; i++)
 {
  if (serializedTiles[i] != null)
   newTiles.Add(serializedCoords[i], serializedTiles[i]);
 }
```

```
   lock (grid) { grid = newTiles; }
  }

 #endregion
}



public interface IPinTile : ITile { void Pin(); }

public class TilePinManager<T> : TileManager<T> where T: IPinTile, IEquatable<T>
{
 private Dictionary<Coord,T> pinned = new Dictionary<Coord,T>();



 public void Pin (Coord coord, MonoBehaviour holder=null)
 /// Creates new tile at the coord if it's empty and pin it
 {
  grid.TryGetValue(coord, out T tile);

  if (tile == null)
  {
   tile = ConstructTile(holder);
   grid.Add(coord, tile);

   tile.Pin();
   tile.Move(coord, camCoords != null ? GetRemoteness(coord,camCoords) : 0);
```

```
 }

 else
  tile.Pin();

 if (pinned.ContainsKey(coord))
  pinned.Add(coord, tile);
}



public void Unpin (Coord coord)
/// Clears pin flag for tile at the coord and re-deploys grid to remove it if needed
{
 if (!pinned.ContainsKey(coord)) return;

 pinned.Remove(coord);

 //re-deploying to find out if this tile should be removed or left as unpinned
 if (camCoords != null)
  Deploy(camCoords, pinned, holder:null); //deploying without holder since it shouldn't create new tiles any

 //no deploy was performed - removing pinned
 else
 {
  grid[coord].Remove();
  grid.Remove(coord);
```

```
    }
   }

   public void Deploy (Coord[] camCoords, MonoBehaviour holder=null)
    { Deploy(camCoords, pinned, holder); }
  }
 }
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using System.Diagnostics;
using System.Runtime.InteropServices;

namespace Den.Tools
{
 public static class Timer
 {
  #if UNITY_EDITOR_WIN
  [DllImport("Kernel32.dll")]
  private static extern int QueryPerformanceCounter(ref long count);
  [DllImport("Kernel32.dll")]
  private static extern int QueryPerformanceFrequency(ref long frequency);
  #endif


  public struct TimerInstance : IDisposable  //struct to avoid adding to GC while profiling
  {
   public string name;

   public int calls;
   public long total;
   public long fastest;
```

```
public long slowest;
public long current;
public bool logAfter;
public void AddCurrent (long t) => current += t; //to use as struct

public bool expanded; //is expanded in gui

public List<TimerInstance> subTimers;

public void AddTime (long delta)
{
 total += delta;
 fastest += delta;
 slowest += delta;

 int subsCount = subTimers.Count;
 for (int i=0; i<subsCount; i++)
  subTimers[i].AddTime(delta);
}

public override int GetHashCode () { return name.GetHashCode(); }
public string Log () { return Log("", 0); }
public string Log (string result, int tab)
{
 for (int i=0; i<tab; i++) result += "\t";
 result += name + " calls:" + calls +
```

```
  " total:" + TicksToMilliseconds(total).ToString("0.000") +
  " fastest:" + TicksToMilliseconds(fastest).ToString("0.000") +
  " average:" + TicksToMilliseconds(total/calls).ToString("0.000") + "\n";
 if (subTimers != null)
  for (int i=0; i<subTimers.Count; i++) result += subTimers[i].Log(result, tab+1);
 return result;
}

public int GetExpandedCount ()
{
 if (subTimers == null || !expanded) return 0;

 int subsCount = subTimers.Count;
 int count = subsCount;
 for (int i=0; i<subsCount; i++)
  count += subTimers[i].GetExpandedCount();

 return count;
}

/*public long SubsTime ()
{
 if (subTimers == null) return total;
 else
 {
  long sum = 0;
```

```
  int subsCount = subTimers.Count;
  for (int i=0; i<subsCount; i++)
   sum += subTimers[i].SubsTime();
  return sum;
 }
}*/

public long SelfTime ()
{
 if (subTimers == null) return total;

 long subsTime = 0;
 int subsCount = subTimers.Count;
 for (int i=0; i<subsCount; i++)
  subsTime += subTimers[i].total;

 return total - subsTime;
}

public void Dispose ()
{
 Stop(this);

 if (logAfter)
  Debug.Log(name + " total:" + TicksToMilliseconds(total).ToString("0.000"));
}
```

```
}



public static bool enabled;

public static long startTime;

public static TimerInstance[] active = new TimerInstance[maxActiveCount]; //TODO: use array to avoid GC
public static int activeCount = 0;
const int maxActiveCount = 1000;

public static List<TimerInstance> history = new List<TimerInstance>();

public static TimerInstance temp = new TimerInstance();

public static Action OnHistoryAdded;

public static void PauseActive (long stopTime)
{
 long delta = stopTime - startTime;

 //if (delta < 0)
 // throw new System.Exception("Negative timer value");

 for (int i=0; i<activeCount; i++)
  active[i].current += delta;
```

```
}

public static TimerInstance Start (string name)
{
 long stopTime = 0;
 #if UNITY_EDITOR_WIN
 QueryPerformanceCounter(ref stopTime);
 #else
 stopTime = System.Diagnostics.Stopwatch.GetTimestamp();
 #endif
 return Start(name, stopTime, false);
}

public static TimerInstance Start (string name, bool logAfter)
{
 long stopTime = 0;
 #if UNITY_EDITOR_WIN
 QueryPerformanceCounter(ref stopTime);
 #else
 stopTime = System.Diagnostics.Stopwatch.GetTimestamp();
 #endif
 return Start(name, stopTime, logAfter);
}

public static TimerInstance Start (string name, long stopTime, bool logAfter)
{
```

```
//stopping active timers
PauseActive(stopTime);

//profiling
UnityEngine.Profiling.Profiler.BeginSample(name);

if (!Timer.enabled) { return temp; }

TimerInstance timer;

//root timer
if (activeCount == 0)
 timer = new TimerInstance() { name=name, fastest=long.MaxValue };

else
{
 TimerInstance lastActive = active[activeCount-1];

 //try finding in active
 if (lastActive.subTimers != null)
  timer = lastActive.subTimers.Find(t => t.name==name);

 //creating new if not found
 else
 {
  timer = new TimerInstance() { name=name, fastest=long.MaxValue };
```

```
   if (lastActive.subTimers == null) lastActive.subTimers = new List<TimerInstance>();
   lastActive.subTimers.Add(timer);
  }
 }

 //activate
 if (activeCount == maxActiveCount-1)
  throw new Exception("Max Timer counter is reached");
 active[activeCount] = timer;
 activeCount++;

 //starting again
 #if UNITY_EDITOR_WIN
 QueryPerformanceCounter(ref startTime);
 #else
 startTime = System.Diagnostics.Stopwatch.GetTimestamp();
 #endif

 timer.logAfter = logAfter;
 return timer;
}


public static void Stop (TimerInstance timer)
{
 //stopping active timers
```

```
long stopTime = 0;
#if UNITY_EDITOR_WIN
QueryPerformanceCounter(ref stopTime);
#else
stopTime = System.Diagnostics.Stopwatch.GetTimestamp();
#endif

//profiling
UnityEngine.Profiling.Profiler.EndSample();

if (!Timer.enabled) return;

PauseActive(stopTime);

//check if the timer is active
if (activeCount == 0)
 throw new System.Exception("Trying to stop timer when there are no active timers running");
//if (active[activeCount-1] != timer)
// throw new System.Exception("Trying to stop non-active timer");

//removing timer from active
//active.RemoveAt(activeCount-1);
activeCount--;

//writing time
timer.calls ++;
```

```
 timer.total += timer.current;
 if (timer.current < timer.fastest) timer.fastest = timer.current;
 if (timer.current > timer.slowest) timer.slowest = timer.current;
 timer.current = 0;

 //if it is root - moving to history
 if (enabled && activeCount == 0)
 {
  history.Add(timer);
  if (OnHistoryAdded != null) OnHistoryAdded();
 }

 //starting all again
 #if UNITY_EDITOR_WIN
 QueryPerformanceCounter(ref startTime);
 #else
 startTime = System.Diagnostics.Stopwatch.GetTimestamp();
 #endif
}


public static double TicksToMilliseconds (long rawTicks)
{
 long frequency = 0;
 #if UNITY_EDITOR_WIN
 QueryPerformanceFrequency(ref frequency);
```

```
 #else
 frequency = System.Diagnostics.Stopwatch.Frequency;
 #endif



 return 1.0 * rawTicks / frequency * 1000;


 /*double dticks;
 if (Stopwatch.IsHighResolution)
 {
  dticks = rawTicks;
  //dticks = rawTicks / Stopwatch.Frequency;
 }
 else
  dticks = rawTicks;


 return dticks / 10000;*/
}



public static void Calibrate ()
{
 long minTme = long.MaxValue;
 for (int i=0; i<10; i++)
 {
  long startTime = 0;
```

```
#if UNITY_EDITOR_WIN
QueryPerformanceCounter(ref startTime);
#else
startTime = System.Diagnostics.Stopwatch.GetTimestamp();
#endif


for (int j=0; j<10000; j++)
{
 using (Timer.Start("Calibrate")) { }
 using (Timer.Start("Calibrate")) { }
 using (Timer.Start("Calibrate")) { }
 using (Timer.Start("Calibrate")) { }
 using (Timer.Start("Calibrate")) { }

 using (Timer.Start("Calibrate")) { }
 using (Timer.Start("Calibrate")) { }
 using (Timer.Start("Calibrate")) { }
 using (Timer.Start("Calibrate")) { }
 using (Timer.Start("Calibrate")) { }
}
long stopTime = 0;
#if UNITY_EDITOR_WIN
QueryPerformanceCounter(ref stopTime);
#else
stopTime = System.Diagnostics.Stopwatch.GetTimestamp();
#endif
```

```
    long delta = stopTime - startTime;
    if (delta < minTme) minTme = delta;
   }
   refinement = minTme/100000;
   history.Clear();
   UnityEngine.Debug.Log("Timer calibrated with refinement " + refinement);
  }
  public static long refinement = 0;
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Den.Tools.Matrices;

namespace Den.Tools
{
 [Serializable]
 public struct Transition
 {
  public Vector3 pos;
  public Quaternion rotation;
  public Vector3 scale;
  public float terrainHeight; //additional height for relative placement

  public int id;   //always unique for every posTab (sets on adding to posTab)
  public int hash; //stays the same on merging posTabs. The one the random depends on
  public int num;  //for brush: tree prototype in tree relocate or prefub instance num for obj relocate.

  [NonSerialized] public Transform instance; //to make brush remember the read objects

  public float Yaw
  {
   get{
```

```
  return Mathf.Acos(rotation.w) / Mathf.Deg2Rad * 2;
 }

 set{
  float halfVal = value/2 * Mathf.Deg2Rad;
  rotation.w = Mathf.Cos(halfVal);
  rotation.z = 0;
  rotation.y = Mathf.Sin(halfVal);
  rotation.x = 0;
 }
}

public (Vector2D,Vector2D) FrontRight2D
{
 get {
  float yaw = Yaw;
  Vector2D front = new Vector2D( Mathf.Sin(yaw*Mathf.Deg2Rad), Mathf.Cos(yaw*Mathf.Deg2Rad) );
  Vector2D right = new Vector2D(front.z, -front.x);
  return (front,right);
 }
}

public Transition (float x, float z)
/// Adds empty transition at coordinates
{
 pos = new Vector3(x,0,z);
```

```
 rotation = new Quaternion(0,0,0,1);
 scale = new Vector3(1,1,1);
 terrainHeight = 0;
 id = 0;
 hash = 0;
 num = 0;
 instance =null;
}


public Transition (float x, float y, float z)
/// Adds empty transition at coordinates
{
 pos = new Vector3(x,y,z);
 rotation = new Quaternion(0,0,0,1);
 scale = new Vector3(1,1,1);
 terrainHeight = 0;
 id = 0;
 hash = 0;
 num = 0;
 instance =null;
}


public Transition (Transform transform)
{
```

```
 pos = transform.position; //MM is generating world positions
 rotation = transform.rotation;
 scale = transform.localScale;
 terrainHeight = 0;
 id = 0;
 hash = 0;
 num = 0;
 instance = transform;
}

public Transition (TreeInstance tree, Vector3 terrainPos, Vector3 terrainSize, Transform prototypePrefab=
{
 pos = new Vector3(tree.position.x*terrainSize.x, tree.position.y*terrainSize.y, tree.position.z*terrainSize.z
 rotation = Quaternion.Euler(0, tree.rotation, 0);
 scale = new Vector3(tree.widthScale, tree.heightScale, tree.widthScale);
 terrainHeight = 0;
 id = 0;
 hash = 0;
 num = tree.prototypeIndex;
 instance = prototypePrefab;
}

/*public static Transition operator * (Transition trs, float val)
{
 trs.pos *= val;
```

```
  trs.rotation *= val;
 }*/
}


[System.Serializable]
public class TransitionsList : ICloneable
{
 public Transition[] arr = new Transition[4];
 public int count = 0;

 private int idCounter;


 public TransitionsList () { }
 public TransitionsList (int count) { arr=new Transition[count]; this.count=count; }
 //public TransitionsList (int count, int startId) { arr=new Transition[count]; this.count=count; idCounter=star
 public TransitionsList (TransitionsList src) { arr=new Transition[src.arr.Length]; Array.Copy(src.arr, arr, src


 public void Add (TransitionsList other)
 /// Combines two lists in current
 {
  for (int t=0; t<other.count; t++)
   Add(other.arr[t]);
  //TODO: avoid resizing array several times
```

```
}



public void Add (float x, float z)
{
 Transition trs = new Transition(x,z);
 Add(trs);
}



public void Add (Transition trs)
{
 idCounter++;
 if (idCounter > 0x7FFFFFFE) idCounter = 1;
 trs.id = idCounter;

 if (trs.hash == 0) //if hash not defined
  trs.hash = trs.id;

 if (arr.Length <= count)
  SetCapacity(arr.Length*2);

 arr[count] = trs;
 count++;
}
```

```
private void SetCapacity (int newCapacity)
{
 Transition[] newArr = new Transition[newCapacity];
 Array.Copy(arr, newArr, arr.Length);
 arr = newArr;
}


public Vector3 Min ()
{
 Vector3 min = new Vector3(float.MaxValue, float.MaxValue, float.MaxValue);
 for (int t=0; t<count; t++)
 {
  if (arr[t].pos.x < min.x) min.x = arr[t].pos.x;
  if (arr[t].pos.y < min.y) min.y = arr[t].pos.y;
  if (arr[t].pos.z < min.z) min.z = arr[t].pos.z;
 }
 return min;
}


public Vector3 Max ()
{
 Vector3 max = new Vector3(float.MinValue, float.MinValue, float.MinValue);
 for (int t=0; t<count; t++)
```

```
 {
  if (arr[t].pos.x > max.x) max.x = arr[t].pos.x;
  if (arr[t].pos.y > max.y) max.y = arr[t].pos.y;
  if (arr[t].pos.z > max.z) max.z = arr[t].pos.z;
 }
 return max;
}



public int CountInRect (Vector2D pos, Vector2D size)
{
 int c = 0;
 for (int t=0; t<count; t++)
 {
  if (arr[t].pos.x > pos.x  &&  arr[t].pos.x < pos.x+size.x  &&
   arr[t].pos.z > pos.z  &&  arr[t].pos.z < pos.z+size.z)
    c ++;
 }
 return c;
}



public object Clone ()
{
 return new TransitionsList() { arr=(Transition[])(arr.Clone()), count=count };
}
```

```
 #region Per-Transition Operations

  public static void Mask (TransitionsList src, TransitionsList dst, MatrixWorld mask, Noise random, bool in
  /// Adds masked (or unmasked if invert) objects to dst
  {
   for (int t=0; t<src.count; t++)
   {
    Vector3 pos = src.arr[t].pos;

    if (pos.x <= mask.worldPos.x  &&   pos.x >= mask.worldPos.x+mask.worldSize.x  &&
     pos.z <= mask.worldPos.z  &&   pos.z >= mask.worldPos.x+mask.worldSize.z)
      continue; //do remove out of range objects?

    float val = mask.GetWorldValue(pos.x, pos.z);
    float rnd = random.Random(src.arr[t].hash);

    if (val<rnd && invert) dst.Add(src.arr[t]);
    if (val>=rnd && !invert) dst.Add(src.arr[t]);
   }
  }

 #endregion
}
```

}

```
ï»¿using System.Collections.Generic;

namespace Den.Tools
{
 public class DictionaryBidirectional<T1,T2>
 {
  public Dictionary<T1,T2> d1 = new Dictionary<T1,T2>();
  public Dictionary<T2,T1> d2 = new Dictionary<T2,T1>();

  public void Add (T1 t1, T2 t2)
  {
   d1.Add(t1, t2);
   d2.Add(t2, t1);
  }

  public void Add (T2 t2, T1 t1)
  {
   d1.Add(t1, t2);
   d2.Add(t2, t1);
  }

  public void ForceAdd (T1 t1, T2 t2)
  {
   if (d1.ContainsKey(t1)) d1[t1] = t2;
   else d1.Add(t1, t2);
```

```
 if (d2.ContainsKey(t2)) d2[t2] = t1;
 else d2.Add(t2, t1);
}

public void ForceAdd (T2 t2, T1 t1)
{
 if (d1.ContainsKey(t1)) d1[t1] = t2;
 else d1.Add(t1, t2);

 if (d2.ContainsKey(t2)) d2[t2] = t1;
 else d2.Add(t2, t1);
}

public T2 this[T1 t1]
{
 get{ return d1[t1]; }
 set{ d1[t1] = value; }
}

public T1 this[T2 t2]
{
 get{ return d2[t2]; }
 set{ d2[t2] = value; }
}

public bool TryGetValue (T1 t1, out T2 t2) { return d1.TryGetValue(t1, out t2); }
```

```
  public bool TryGetValue (T2 t2, out T1 t1) { return d2.TryGetValue(t2, out t1); }

  public bool ContainsKey (T1 t1) { return d1.ContainsKey(t1); }
  public bool ContainsKey (T2 t2) { return d2.ContainsKey(t2); }

  public void Clear ()
  {
   d1.Clear();
   d2.Clear();
  }
 }
}
```

```
ï»¿using System;
using System.Collections.Generic;

namespace Den.Tools
{
 public class TypeDict
 {
  private Dictionary<Type,object> dict = new Dictionary<Type,object>();

  public TypeDict () {}
  public TypeDict (object[] arr) { Add(arr); }


  public void Add<T> (T val)
   { dict.Add(typeof(T), val); }

  public void TryAdd<T> (T val)
  {
   if (dict.ContainsKey(typeof(T))) return;
   dict.Add(typeof(T), val);
  }

  public void ForceAdd<T> (T val)
  {
   if (dict.ContainsKey(typeof(T))) dict[typeof(T)] = val;
   else dict.Add(typeof(T), val);
```

```
}

public void Add (object[] arr)
{
 for (int i=0; i<arr.Length; i++)
 {
  Type type = arr[i].GetType();
  if (!dict.ContainsKey(type))
   dict.Add(type, arr[i]);
 }
}

public bool TryGetValue<T> (out T val)
{
 bool found = dict.TryGetValue(typeof(T), out object obj);
 val = (T)obj;
 return found;
}

public T GetValue<T> ()
 { return (T)dict[typeof(T)]; }

public bool ContainsKey<T> ()
 { return dict.ContainsKey(typeof(T)); }

public void Clear ()
```

```
  { dict.Clear(); }
}



public class TypeDict<T2>
///Note that values are not bound to key types
{
 private Dictionary<Type,T2> dict = new Dictionary<Type,T2>();

 public void Add<T1> (T2 val)
  { dict.Add(typeof(T1), val); }

 public void TryAdd<T1> (T2 val)
 {
  if (dict.ContainsKey(typeof(T1))) return;
  dict.Add(typeof(T1), val);
 }

 public void ForceAdd<T1> (T2 val)
 {
  if (dict.ContainsKey(typeof(T1))) dict[typeof(T1)] = val;
  else dict.Add(typeof(T1), val);
 }

 public bool TryGetValue<T1> (out T2 t2)
  { return dict.TryGetValue(typeof(T1), out t2); }
```

```
  public T2 GetValue<T1> ()
   { return dict[typeof(T1)]; }

  public bool ContainsKey<T1> ()
   { return dict.ContainsKey(typeof(T1)); }

  public void Clear ()
   { dict.Clear(); }
 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
using System.IO;

namespace Den.Tools
{

 public static class AssemblePlugins
 {



  [MenuItem("Assets/Assemble MapMagic")]
  private static void AssembleMapMagic ()
  {
   string path = EditorUtility.SaveFolderPanel("Assemble MapMagic To", "", "MapMagic");

   //Directory.Delete(path, true);
   //Debug.Log(AssetDatabase.proj
  }


  private static void CopyFile (string fromFolder, string toFolder, string name)
  {
```

```
  }



  private static void CopyFolder (string fromFolder, string toFolder, string name)
  {


  }
 }

}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
using System.Reflection;

namespace Den.Tools
{
 public static class EditorHacks
 {
  public static void SetIconForObject (UnityEngine.Object obj, Texture2D icon)
  {
   var flags = System.Reflection.BindingFlags.InvokeMethod | System.Reflection.BindingFlags.Static | Syst
   var argTypes = new System.Type[]{typeof(UnityEngine.Object), typeof(Texture2D)};
   var methodInfo = typeof(EditorGUIUtility).GetMethod("SetIconForObject", flags, null, argTypes, null);

   var args = new object[] { obj, icon };
   methodInfo?.Invoke(null, args);
  }


  public static Assembly EditorAssembly
   { get{ return Assembly.GetAssembly(typeof(Editor)); } }

  public static Type GetEditorType (string typeName) //type name with the namespace (UnityEditor.ObjectL
```

```
 { return EditorAssembly?.GetType(typeName); }



public static void SubscribeToListIconDrawCallback (Action<Rect,string,bool> action)
{
 Type type = GetEditorType("UnityEditor.ObjectListArea");
 EventInfo eventInfo = type.GetEvent("postAssetIconDrawCallback", BindingFlags.Static | BindingFlags.N

 Delegate handler = Delegate.CreateDelegate(eventInfo.EventHandlerType, action.Target, action.Method

 //eventInfo.AddEventHandler(null, handler);
 var addMethod = eventInfo.GetAddMethod(true);
 addMethod.Invoke(null, new[] {handler});
}

public static void SubscribeToTreeIconDrawCallback (Action<Rect,string> action)
{
 Type type = GetEditorType("UnityEditor.AssetsTreeViewGUI");
 EventInfo eventInfo = type.GetEvent("postAssetIconDrawCallback", BindingFlags.Static | BindingFlags.N

 Delegate handler = Delegate.CreateDelegate(eventInfo.EventHandlerType, action.Target, action.Method

 var addMethod = eventInfo.GetAddMethod(true);
 addMethod.Invoke(null, new[] {handler});
}
```

```
  public static void SubscribeToLabelDrawCallback (Func<Rect,string,bool, bool> func)
  // func return true if drawing occured (space will be redistributed if false)
  {
   Type type = GetEditorType("UnityEditor.ObjectListArea");
   EventInfo eventInfo = type.GetEvent("postAssetLabelDrawCallback", BindingFlags.Static | BindingFlags.

   Delegate handler = Delegate.CreateDelegate(eventInfo.EventHandlerType, func.Target, func.Method);

   //eventInfo.AddEventHandler(null, handler);
   var addMethod = eventInfo.GetAddMethod(true);
   addMethod.Invoke(null, new[] {handler});
  }

  //[RuntimeInitializeOnLoadMethod, UnityEditor.InitializeOnLoadMethod]
  //static void Subscribe()
  // { SceneView.duringSceneGui += DragGraphToScene; }
 }
}
```

```
ï»¿using System;
using System.IO;
using System.Reflection;
using UnityEditor;
using UnityEngine;

namespace Den.Tools
{
 #if MM_ExtendedTerrainInspector
 [CustomEditor(typeof(Terrain))]
 #endif
 public class ExtendedTerrainInspector : Editor
 {
  private Type inspectorType;
  [NonSerialized] private Editor inspector;

  private Terrain terrain;
  private TerrainCollider terrainCollider;

  private PropertyInfo selectedToolProperty;

  private Action onInspectorGUIAction;
  private Action onInspectorUpdateAction;
  private Action saveInspectorSettingsAction;
  private Action onEnableAction;
  private Action onDisableAction;
```

```
private Action<SceneView> onSceneGUIAction;
private Action<SceneView> onPreSceneGUIAction;
private Func<bool> isModificationAliveAction;

private Bounds strokeBounds;

//public static WeakEvent<Terrain,Bounds,int> StrokeFinished = new WeakEvent<Terrain,Bounds,int>();
public static Action<Terrain,Bounds,int> StrokeFinished;

public void Init ()
{
 //inspector type
 Assembly a = Assembly.GetAssembly(typeof(Editor));
 inspectorType = ArrayTools.FindMember(a.GetTypes(), x => x.Name=="TerrainInspector");

 if (inspectorType==null)
  throw new Exception("Could not load TerrainInspector type");

 //creating a temporary editor
 if (inspector == null)
  inspector = CreateEditor(targets, inspectorType);

 //terrain
 terrain = (Terrain)target;
 terrainCollider = terrain.GetComponent<TerrainCollider>();
```

```
        //properties
        selectedToolProperty = inspectorType.GetProperty("selectedTool",  BindingFlags.Instance | BindingFlags

        //methods
        MethodInfo onInspectorGUIMethod = inspectorType.GetMethod("OnInspectorGUI");
        MethodInfo onInspectorUpdateMethod = inspectorType.GetMethod("OnInspectorUpdate",  BindingFlags.
        MethodInfo saveInspectorSettingsMethod = inspectorType.GetMethod("SaveInspectorSettings", Binding
        MethodInfo onEnableMethod = inspectorType.GetMethod("OnEnable");
        MethodInfo onDisableMethod = inspectorType.GetMethod("OnDisable");

        //caching most used methods to actions
        onInspectorGUIAction = (Action)Delegate.CreateDelegate(typeof(Action), inspector, onInspectorGUIMeth
        onInspectorUpdateAction = (Action)Delegate.CreateDelegate(typeof(Action), inspector, onInspectorUpda
        saveInspectorSettingsAction = (Action)Delegate.CreateDelegate(typeof(Action), inspector, saveInspecto
        onEnableAction = (Action)Delegate.CreateDelegate(typeof(Action), inspector, onEnableMethod);
        onDisableAction = (Action)Delegate.CreateDelegate(typeof(Action), inspector, onDisableMethod);
        isModificationAliveAction = (Func<bool>)Delegate.CreateDelegate(typeof(Func<bool>), inspector, inspec

        onSceneGUIAction = (Action<SceneView>)Delegate.CreateDelegate(typeof(Action<SceneView>), inspe

        //initializing
        //MethodInfo initializeMethod = inspectorType.GetMethod("Initialize", BindingFlags.Instance | BindingFlag
        //initializeMethod.Invoke(inspector, null);
    }
```

```
public void OnSceneGUI ()
{
 Event current = Event.current;
 int controlID = GUIUtility.GetControlID("TerrainEditor".GetHashCode(), FocusType.Passive);

 if (!Event.current.alt &&
  current.button == 0 &&
  isModificationAliveAction() )
 {
  Vector3 brushPos = BrushPos();
  Bounds brushBounds = new Bounds(brushPos, Vector3.zero);

  float pixelSize = 1f * terrain.terrainData.size.x / Mathf.Min(terrain.terrainData.heightmapResolution, terra
  brushBounds.Expand( EditorPrefs.GetInt("TerrainBrushSize") * pixelSize );

  switch (Event.current.type)
  {
   case EventType.MouseDown:
    strokeBounds = brushBounds;
    break;
   case EventType.MouseDrag:
    strokeBounds.Encapsulate(brushBounds);
    break;
   case EventType.MouseUp:
    //saveInspectorSettingsAction();
    if (StrokeFinished != null)
```

```
      StrokeFinished(terrain, strokeBounds, 1);
     break;
   }

   Handles.color = Color.red;
   Handles.DrawWireCube(strokeBounds.center, strokeBounds.extents*2);
  }
}



public void OnDisable ()
{
 onDisableAction();
 onDisableAction(); //ondisabling twice to remove onprescenegui delegates (somehow they are added twi

 DestroyImmediate(inspector); //otherwise it will still be calling onscenegui
 inspector = null;
}



public void OnEnable ()
{
 if (inspector == null) Init();
 onEnableAction();
}
```

```
  public override void OnInspectorGUI ()
  {
   if (inspector == null) Init();
   onInspectorGUIAction();
  }


  public Vector3 BrushPos ()
  {
   Ray ray = HandleUtility.GUIPointToWorldRay(Event.current.mousePosition);
   RaycastHit hit;
   if (terrainCollider.Raycast(ray, out hit, Mathf.Infinity))
    return hit.point;
   else
    return strokeBounds.center; //returning somewhere within stroke
  }
 }

}



//properties:
//UnityEditor.TerrainTool selectedTool,
//System.Boolean   canEditMultipleObjects,
```

//UnityEngine.Object target,

//UnityEngine.Object[] targets,

//System.Int32   referenceTargetIndex,

//System.String   targetTitle,

//UnityEditor.SerializedObject serializedObject,

//System.Boolean   isInspectorDirty,

//UnityEditor.IPreviewable preview,

//UnityEditor.PropertyHandlerCache propertyHandlerCache,

//System.String   name,

//UnityEngine.HideFlags hideFlags

//methods:

//Single PercentSlider(UnityEngine.GUIContent, Single, Single, Single),

//Void CheckKeys(),

//Void LoadBrushIcons(),

//Void Initialize(),

//Void LoadInspectorSettings(),

//Void SaveInspectorSettings(),

//Void OnEnable(),

//Void OnDisable(),

//TerrainTool get_selectedTool(),

//Void set_selectedTool(TerrainTool),

//Void MenuButton(UnityEngine.GUIContent, System.String, Int32),

//Int32 AspectSelectionGrid(Int32, UnityEngine.Texture[], Int32, UnityEngine.GUIStyle, System.String, Boo

//Rect GetBrushAspectRect(Int32, Int32, Int32, Int32 ByRef),

//Int32 AspectSelectionGridImageAndText(Int32, UnityEngine.GUIContent[], Int32, UnityEngine.GUIStyle,

//Void LoadSplatIcons(), Void LoadTreeIcons(), Void LoadDetailIcons(),

//Void ShowTrees(), Void ShowDetails(), Void ShowSettings(), Void ShowRaiseHeight(), Void ShowSmoot

//Void ResizeDetailResolution(UnityEngine.TerrainData, Int32, Int32),

//Void ShowUpgradeTreePrototypeScaleUI(),

//Void InitializeLightingFields(),

//Void RenderLightingFields(),

//Void OnInspectorUpdate(),

//Void OnInspectorGUI(),

//UnityEditor.Brush GetActiveBrush(Int32),

//Boolean Raycast(Vector2 ByRef, Vector3 ByRef),

//Boolean HasFrameBounds(),

//Bounds OnGetFrameBounds(),

//Boolean IsModificationToolActive(),

//Boolean IsBrushPreviewVisible(),

//Void DisableProjector(),

//Void UpdatePreviewBrush(),

//Void OnSceneGUICallback(UnityEditor.SceneView),

//Void OnPreSceneGUICallback(UnityEditor.SceneView)

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;

namespace Den.Tools
{
 public class TimerWindow : EditorWindow
 {
  const int toolbarHeight = 18;
  const int scrollWidth = 15;
  const int lineHeight = 18;
  const float namesWidthPercent = 0.4f;
  Vector2 scrollPosition;
  int selectedLine = 1;

  Timer.TimerInstance groupTimer;
  bool groupEnabled = false;

  bool prevEnabled;

  public void OnEnable ()
  {
   Timer.OnHistoryAdded -= Repaint;
   Timer.OnHistoryAdded += Repaint;
  }
```

```
public void OnGUI ()
{
 //toolbar/header
 if (Event.current.type == EventType.Repaint)
  EditorStyles.toolbar.Draw(new Rect(0,0,position.width, toolbarHeight), new GUIContent(), 0);

 //Record
 Timer.enabled = EditorGUI.Toggle(new Rect(5,0,50,toolbarHeight), Timer.enabled, style:EditorStyles.too
 EditorGUI.LabelField(new Rect(9,1,50,toolbarHeight), "Record", style:EditorStyles.miniBoldLabel);

 //Group
 bool newGroupEnabled = EditorGUI.Toggle(new Rect(55,0,50,toolbarHeight), groupEnabled, style:Editor
 if (newGroupEnabled && groupEnabled) //just pressed
 {
  prevEnabled = Timer.enabled;
  Timer.enabled = true;
  groupTimer = Timer.Start("Record Group " + Timer.history.Count);
  groupEnabled = true;
 }
 if (!newGroupEnabled && groupEnabled)
 {
  groupTimer.Dispose();
  groupEnabled = false;
  Timer.enabled = prevEnabled;
 }
```

```
EditorGUI.LabelField(new Rect(63,1,50,toolbarHeight), "Group", style:EditorStyles.miniBoldLabel);

//Clear
if (UnityEngine.GUI.Button(new Rect(105,0,50,toolbarHeight), "Clear", style:EditorStyles.toolbarButton))
 Timer.history.Clear();

DrawHeaderValues(new Rect(0, 0, position.width-scrollWidth, toolbarHeight));

//calculating lines number
int historyCount = Timer.history.Count;
int linesCount = historyCount;
for (int h=0; h<historyCount; h++)
 linesCount += Timer.history[h].GetExpandedCount();

//drawing timers
scrollPosition = UnityEngine.GUI.BeginScrollView(
 position:new Rect(0, toolbarHeight, position.width, position.height-toolbarHeight),
 scrollPosition:scrollPosition,
 viewRect:new Rect(0, 0, position.width-scrollWidth, linesCount*lineHeight),
 alwaysShowHorizontal:false,
 alwaysShowVertical:true);

int lineNum = 0;
for (int h=0; h<historyCount; h++)
{
 Timer.TimerInstance timer = Timer.history[h];
```

```
  DrawTimer(new Rect(0,lineNum*lineHeight,position.width-scrollWidth, lineHeight), timer, ref lineNum, 0,
 }

 UnityEngine.GUI.EndScrollView();
}


public void DrawHeaderValues (Rect rect)
{
 float namesWidth = rect.width * namesWidthPercent;
 float valuesWidth = rect.width - namesWidth;
 float rowWidth = valuesWidth / 7;

 //if (Event.current.type == EventType.Repaint)
 // for (int i=0; i<5; i++)
 //  EditorStyles.toolbarButton.Draw(new Rect(rect.x+rowWidth*i+namesWidth, rect.y, rowWidth+1, rect.he
 //   isHover:false, isActive:false, on:false, hasKeyboardFocus:false);

 Rect row = new Rect(rect.x+namesWidth, rect.y, rowWidth+1, rect.height);
 if (Event.current.type == EventType.Repaint)
  EditorStyles.toolbarButton.Draw(row, isHover:false, isActive:false, on:false, hasKeyboardFocus:false);
 EditorGUI.LabelField(new Rect(row.x+2, row.y+1, row.width, row.height), "Calls", style:EditorStyles.miniL

 row.x += rowWidth;
 if (Event.current.type == EventType.Repaint)
  EditorStyles.toolbarButton.Draw(row, isHover:false, isActive:false, on:false, hasKeyboardFocus:false);
```

```
EditorGUI.LabelField(new Rect(row.x+2, row.y+1, row.width, row.height), "Total", style:EditorStyles.miniL

row.x += rowWidth;
if (Event.current.type == EventType.Repaint)
 EditorStyles.toolbarButton.Draw(row, isHover:false, isActive:false, on:false, hasKeyboardFocus:false);
EditorGUI.LabelField(new Rect(row.x+2, row.y+1, row.width, row.height), "Self", style:EditorStyles.miniLa

row.x += rowWidth;
if (Event.current.type == EventType.Repaint)
 EditorStyles.toolbarButton.Draw(row, isHover:false, isActive:false, on:false, hasKeyboardFocus:false);
EditorGUI.LabelField(new Rect(row.x+2, row.y+1, row.width, row.height), "Average", style:EditorStyles.m

row.x += rowWidth;
if (Event.current.type == EventType.Repaint)
 EditorStyles.toolbarButton.Draw(row, isHover:false, isActive:false, on:false, hasKeyboardFocus:false);
EditorGUI.LabelField(new Rect(row.x+2, row.y+1, row.width, row.height), "Fastest", style:EditorStyles.mi

row.x += rowWidth;
if (Event.current.type == EventType.Repaint)
 EditorStyles.toolbarButton.Draw(row, isHover:false, isActive:false, on:false, hasKeyboardFocus:false);
EditorGUI.LabelField(new Rect(row.x+2, row.y+1, row.width, row.height), "Slowest", style:EditorStyles.m

row.x += rowWidth;
if (Event.current.type == EventType.Repaint)
 EditorStyles.toolbarButton.Draw(row, isHover:false, isActive:false, on:false, hasKeyboardFocus:false);
EditorGUI.LabelField(new Rect(row.x+2, row.y+1, row.width, row.height), "Fast*Num", style:EditorStyles.
```

```
}



public void DrawTimer (Rect rect, Timer.TimerInstance timer, ref int lineNum, int offset=0, int num=-1)
{
 float namesWidth = rect.width * namesWidthPercent;
 float valuesWidth = rect.width - namesWidth;
 float rowWidth = valuesWidth / 7;
 float numWidth = 25;
 float tabWidth = 10;

 if (rect.y - scrollPosition.y >= 0  &&  rect.y - scrollPosition.y < position.height)
 {

  //selection
  if (Event.current.type == EventType.MouseDown && Event.current.button == 0 && rect.Contains(Event.c
  {
   selectedLine = lineNum;
   Repaint();
  }

  if (selectedLine == lineNum && Event.current.type == EventType.Repaint)
  {
   EditorGUI.BeginDisabledGroup(true);
   EditorStyles.helpBox.Draw(rect, new GUIContent(), 0);
   EditorGUI.EndDisabledGroup();
```

```
}

//disabling if fastest value is too low
EditorGUI.BeginDisabledGroup(timer.fastest < 10);

//name
if (num>=0)
 EditorGUI.LabelField(new Rect(rect.x, rect.y, numWidth, rect.height), num.ToString());

float namePos = offset*tabWidth + numWidth;
Rect nameRect = new Rect(rect.x+namePos, rect.y, namesWidth-namePos, rect.height);
if (timer.subTimers != null)
 timer.expanded = EditorGUI.Foldout(nameRect, timer.expanded, timer.name);
else
 EditorGUI.LabelField(new Rect(nameRect.x+12, nameRect.y, nameRect.width-12, nameRect.height), ti

//values
Rect row = new Rect(rect.x+namesWidth, rect.y, rowWidth, rect.height);
EditorGUI.LabelField(row, timer.calls.ToString());

row.x += rowWidth;
EditorGUI.LabelField(row, Timer.TicksToMilliseconds(timer.total).ToString("0.000"));

row.x += rowWidth;
EditorGUI.LabelField(row, Timer.TicksToMilliseconds(timer.SelfTime()).ToString("0.000"));
```

```
  row.x += rowWidth;
  EditorGUI.LabelField(row, Timer.TicksToMilliseconds(timer.total / timer.calls).ToString("0.000"));

  row.x += rowWidth;
  EditorGUI.LabelField(row, Timer.TicksToMilliseconds(timer.fastest).ToString("0.000"));

  row.x += rowWidth;
  EditorGUI.LabelField(row, Timer.TicksToMilliseconds(timer.slowest).ToString("0.000"));

  row.x += rowWidth;
  EditorGUI.LabelField(row, Timer.TicksToMilliseconds(timer.fastest * timer.calls).ToString("0.000"));

  EditorGUI.EndDisabledGroup();
 }

 //sub-timers
 int origLineNum = lineNum;
 lineNum++;
 if (timer.expanded && timer.subTimers!=null)
 {
  for (int i=0; i<timer.subTimers.Count; i++)
   DrawTimer(new Rect(rect.x, rect.y+lineHeight*(lineNum-origLineNum), rect.width, rect.height), timer.sub
 }
}
```

```
  [MenuItem ("Window/Timers")]
  public static void ShowEditor ()
  {
   EditorWindow.GetWindow<TimerWindow>("Timers");
  }
 }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection; //to copy properties
using UnityEngine.SceneManagement;

namespace Den.Tools
{
 static public class AssetsExtensions
 {
  public static string GUID (this UnityEngine.Object obj)
  {
   #if UNITY_EDITOR
   string path = UnityEditor.AssetDatabase.GetAssetPath(obj);
   if (path==null || path.Length==0) return "";
   string guid = UnityEditor.AssetDatabase.AssetPathToGUID(path);
   if (guid==null || guid.Length==0) return ""; //should not return null
   return guid;
   #else
   Debug.LogError("GUID does not work in build");
   return "";
   #endif
  }

  #if UNITY_EDITOR
```

```
public static UnityEditor.AssetImporter GetImporter (this UnityEngine.Object obj)
{
 string path = UnityEditor.AssetDatabase.GetAssetPath(obj);
 if (path==null || path.Length==0) return null;
 return UnityEditor.AssetImporter.GetAtPath(path);
}
#endif

#if UNITY_EDITOR
public static UnityEditor.AssetImporter GetImporter (string guid)
{

 string path = UnityEditor.AssetDatabase.GUIDToAssetPath(guid);
 if (path==null || path.Length==0) return null;
 return UnityEditor.AssetImporter.GetAtPath(path);

}
#endif

public static T GUIDtoObj<T> (this string guid) where T: UnityEngine.Object
{
 #if UNITY_EDITOR
 if (guid==null || guid.Length==0) return null;
 string sourcePath = UnityEditor.AssetDatabase.GUIDToAssetPath(guid);
 if (sourcePath.Length==0) return null;
 return UnityEditor.AssetDatabase.LoadAssetAtPath<T>(sourcePath);
```

```
 #else
 Debug.LogError("GUIDtObj does not work in build");
 return null;
 #endif
}


public static string[] GetUserData (this UnityEngine.Object obj, string param)
{
 #if UNITY_EDITOR
 UnityEditor.AssetImporter importer = obj.GetImporter();
 if (importer == null) return null;
 return GetUserData(importer, param);
 #else
 Debug.LogError("GetUserData does not work in build");
 return null;
 #endif
}

public static string[] GetUserData (string guid, string param)
{
 #if UNITY_EDITOR
 UnityEditor.AssetImporter importer = GetImporter(guid);
 if (importer == null) return null;
 return GetUserData(importer, param);
 #else
```

```
 Debug.LogError("GetUserData does not work in build");
 return null;
 #endif
}

#if UNITY_EDITOR
public static string[] GetUserData (this UnityEditor.AssetImporter importer, string param)
{
 string userData = importer.userData;
 if (userData==null) return null;
 if (userData.Length==0) return new string[0];

 string[] userDataSplit = userData.Split('\n', ';');
 for (int i=0; i<userDataSplit.Length; i++)
 {
  if (userDataSplit[i].StartsWith(param + ":"))
  {
   userDataSplit[i] = userDataSplit[i].Remove(0, param.Length+1);
   return userDataSplit[i].Split(',');
  }
 }

 return new string[0];
}
#endif
```

```
public static void SetUserData (this UnityEngine.Object obj, string param, string[] data, bool reload = false
{
 #if UNITY_EDITOR
 UnityEditor.AssetImporter importer = obj.GetImporter();
 if (importer == null) return;
 SetUserData(importer, param, data, reload);
 #else
 Debug.LogError("SetUserData does not work in build");
 #endif
}

public static void SetUserData (string guid, string param, string[] data, bool reload = false)
{
 #if UNITY_EDITOR
 UnityEditor.AssetImporter importer = GetImporter(guid);
 if (importer == null) return;
 SetUserData(importer, param, data, reload);
 #else
 Debug.LogError("SetUserData does not work in build");
 #endif
}

#if UNITY_EDITOR
public static void SetUserData (this UnityEditor.AssetImporter importer, string param, string[] data, bool re
{
 char endline = '\n'; //';'
```

```
string userData = importer.userData;
string[] userDataSplit = userData.Split('\n', ';');

//preparing new data line
if (data == null) data = new string[0];
string newDataString = param + ":" + data.ToStringMemberwise(separator:",");

//param line number (-1 if not found)
int numInSplit = -1;
for (int i=0; i<userDataSplit.Length; i++)
 if (userDataSplit[i].StartsWith(param + ":"))
  numInSplit = i;

//erasing empty data
if (numInSplit >= 0 && data.Length == 0)
 ArrayTools.RemoveAt(ref userDataSplit, numInSplit);

//replacing line
if (numInSplit >= 0 && data.Length != 0)
 userDataSplit[numInSplit] = newDataString;

//adding new line
if (numInSplit == -1 && data.Length != 0)
 ArrayTools.Add(ref userDataSplit, newDataString);
```

```
 //to string
 string newUserData = "";
 for (int i=0; i<userDataSplit.Length; i++)
 {
  if (userDataSplit[i].Length == 0) continue;
  newUserData += userDataSplit[i];
  if (i!=userDataSplit.Length-1) newUserData += endline;
 }

 //writing
 if (newUserData != userData)
 {
  importer.userData = newUserData;

  UnityEditor.EditorUtility.SetDirty(importer);
  UnityEditor.AssetDatabase.WriteImportSettingsIfDirty(importer.assetPath);
  if (reload) UnityEditor.AssetDatabase.Refresh();
 }
}
#endif


public static void Reimport (this UnityEngine.Object obj)
{
 #if UNITY_EDITOR
 string path = UnityEditor.AssetDatabase.GetAssetPath(obj);
```

```
 if (path==null || path.Length==0) return;
 UnityEditor.AssetImporter importer = UnityEditor.AssetImporter.GetAtPath(path);
 importer.userData = importer.userData;
 UnityEditor.EditorUtility.SetDirty(importer);
 importer.SaveAndReimport();
 #else
 Debug.LogError("Reimport does not work in build");
 #endif
}



public static int GetDirtyId (this Scene scene)
{
 //Scene scene = SceneManager.GetActiveScene();
 if (dirtyIdProp == null)
  dirtyIdProp = typeof(Scene).GetProperty("dirtyID", BindingFlags.Instance | BindingFlags.NonPublic);
 return (int)dirtyIdProp.GetValue(scene, new object[0]);
}
private static PropertyInfo dirtyIdProp;

public static int GetDirtyId ()
/// Gets summary dirty id of all scenes
{
 int dirtySum = 0;
 int scenesCount = SceneManager.sceneCount;
 for (int s=0; s<scenesCount; s++)
```

```
 {
  Scene scene = SceneManager.GetSceneAt(s);
  dirtySum += scene.GetDirtyId();
 }
 return dirtySum;
}



public static Texture2D GetNormalTexture (this Texture2D diffuse)
/// Tries to load texture with the postfix _n or _normal and same name as this texture
{
 #if UNITY_EDITOR
 string path = UnityEditor.AssetDatabase.GetAssetPath(diffuse);
 if (path == null)
  return null;

 //megasplat
 if (path.Contains("_diff."))
  path = path.Replace("_diff.", "_norm.");

 //microsplat
 else if (path.Contains("_albedo."))
  path = path.Replace("_albedo.", "_norm.");

 //cts
```

```
 else if (path.Contains("_A_Sm."))
  path = path.Replace("_A_Sm.", "_N.");

 else if (path.Contains("_A_Sm_Dry."))
  path = path.Replace("_A_Sm_Dry.", "_N.");

 //mapmagic
 else path = path.Replace(".", "_nrm.");

 return UnityEditor.AssetDatabase.LoadAssetAtPath<Texture2D>(path);
 #else
 return null;
 #endif
}


public static Texture GetMainTexture (this GameObject gameObject) =>
 gameObject.GetComponent<Renderer>()?.material?.mainTexture;


public static bool IsAsset (this GameObject obj)
{
 #if UNITY_EDITOR
 return UnityEditor.AssetDatabase.Contains(obj);
 #else
 return false;
```

```
   #endif
  }

  public static bool IsNull (this UnityEngine.Object unityObj)
  /// Can check if unity obj is null in thread
  /// Just checking unityObj == null will throw an error if object was removed (available in main thread)
  {
   //return (UnityEngine.Object)unityObj != (UnityEngine.Object)null;
   try { return unityObj == null; }
   catch (Exception) { return false; }
  }
 }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection; //to copy properties

namespace Den.Tools
{
 static public class CoordinatesExtensions
 {
  public static bool InRange (this Rect rect, Vector2 pos)
  {
   return (rect.center - pos).sqrMagnitude < (rect.width/2f)*(rect.width/2f);
   //return rect.Contains(pos);
  }

  public static Vector3 ToDir (this float angle) { return new Vector3( Mathf.Sin(angle*Mathf.Deg2Rad), 0, Ma
  public static float ToAngle (this Vector3 dir) { return Mathf.Atan2(dir.x, dir.z) * Mathf.Rad2Deg; }

  public static Vector3 Mul (this Vector3 v, Vector3 m) { return new Vector3(v.x*=m.x, v.y*=m.y, v.z*=m.z); }
  public static Vector3 Div (this Vector3 v, Vector3 m) { return new Vector3(v.x/=m.x, v.y/=m.y, v.z/=m.z); }

  public static Vector3 V3 (this Vector2 v2) { return new Vector3(v2.x, 0, v2.y); }
  public static Vector2 V2 (this Vector3 v3) { return new Vector2(v3.x, v3.z); }
  public static Vector3 ToV3 (this float f) { return new Vector3(f,f, f); }
  public static Vector4 ToV4 (this Rect r) { return new Vector4(r.x, r.y, r.width, r.height); }
```

```
public static Quaternion EulerToQuat (this Vector3 v) { Quaternion rotation = Quaternion.identity; rotation.
public static Quaternion EulerToQuat (this float f) { Quaternion rotation = Quaternion.identity; rotation.eule

public static Vector3 Direction (this float angle) { return new Vector3( Mathf.Sin(angle*Mathf.Deg2Rad), 0
public static float Angle (this Vector3 dir) { return Mathf.Atan2(dir.x, dir.z) * Mathf.Rad2Deg; }

public static Rect Clamp (this Rect r, float p) { return new Rect(r.x, r.y, r.width*p, r.height); }
public static Rect ClampFromLeft (this Rect r, float p) { return new Rect(r.x+r.width*(1f-p), r.y, r.width*p, r.
public static Rect Clamp (this Rect r, int p) { return new Rect(r.x, r.y, p, r.height); }
public static Rect ClampFromLeft (this Rect r, int p) { return new Rect(r.x+(r.width-p), r.y, p, r.height); }

public static Vector3 Pow (this Vector3 v, float pow) => new Vector3( Mathf.Pow(v.x,pow), Mathf.Pow(v.y,

public static Rect Intersect (Rect r1, Rect r2)
{
 Rect result = new Rect(0,0,0,0);

 result.x = Mathf.Max(r1.x, r2.x);
 result.y = Mathf.Max(r1.y, r2.y);

 result.max = new Vector2(
  Mathf.Min(r1.max.x, r2.max.x),
  Mathf.Min(r1.max.y, r2.max.y) );

 if (result.size.x<0) result.size = new Vector2(0, result.size.y);
```

```
 if (result.size.y<0) result.size = new Vector2(result.size.y, 0);

 return result;
}

public static Rect Intersect (Rect r1, CoordRect r2)
{
 Rect result = new Rect(0,0,0,0);

 result.x = Mathf.Max(r1.x, r2.offset.x);
 result.y = Mathf.Max(r1.y, r2.offset.z);

 result.max = new Vector2(
  Mathf.Min(r1.max.x, r2.offset.x+r2.size.x),
  Mathf.Min(r1.max.y, r2.offset.z+r2.size.z) );

 if (result.size.x<0) result.size = new Vector2(0, result.size.y);
 if (result.size.y<0) result.size = new Vector2(result.size.y, 0);

 return result;
}

public static Rect ToRect(this Vector3 center, float range) { return new Rect(center.x-range, center.z-rang

public static Vector3 Average (this Vector3[] vecs) { Vector3 result = Vector3.zero; for (int i=0; i<vecs.Leng
```

```
public static bool Intersects (this Rect r1, Rect r2)
{
 Vector2 r1min = r1.min; Vector2 r1max = r1.max;
 Vector2 r2min = r2.min; Vector2 r2max = r2.max;
 if (r2max.x < r1min.x || r2min.x > r1max.x || r2max.y < r1min.y || r2min.y > r1max.y) return false;
 else return true;
}

public static bool Intersects (this Rect r1,  Vector2 pos, Vector2 size)
{
 Vector2 r1min = r1.min; Vector2 r1max = r1.max;
 Vector2 r2min = pos; Vector2 r2max = pos+size;
 if (r2max.x < r1min.x || r2min.x > r1max.x || r2max.y < r1min.y || r2min.y > r1max.y) return false;
 else return true;
}

public static bool Intersects (this Rect r1, Rect[] rects)
{
 for (int i=0; i<rects.Length; i++)
  if (r1.Intersects(rects[i])) return true;
 return false;
}

public static bool Contains (this Rect r1, Rect r2)
{
```

```
 Vector2 r1min = r1.min; Vector2 r1max = r1.max;
 Vector2 r2min = r2.min; Vector2 r2max = r2.max;
 if (r2min.x > r1min.x && r2max.x < r1max.x && r2min.y > r1min.y && r2max.y < r1max.y) return true;
 else return false;
}

public static bool Contains (this Rect r, Vector2 pos, Vector2 size)
{
 Vector2 r1min = r.min; Vector2 r1max = r.max;
 Vector2 r2min = pos; Vector2 r2max = pos+size;
 if (r2min.x > r1min.x && r2max.x < r1max.x && r2min.y > r1min.y && r2max.y < r1max.y) return true;
 else return false;
}

public static bool ContainsOrIntersects (this Rect r1, Rect r2)
{
 Vector2 r1min = r1.min; Vector2 r1max = r1.max;
 Vector2 r2min = r2.min; Vector2 r2max = r2.max;
 if (r2min.x>r1max.x || r2min.y>r1max.y || r2max.x<r1min.x || r2max.y<r1min.y) return false;
 else return true;
}

public static bool ContainsOrIntersects (this Rect r1, Rect r2, float padding)
{
 Vector2 r1min = r1.min; Vector2 r1max = r1.max;
 Vector2 r2min = r2.min; Vector2 r2max = r2.max;
```

```
 if (r2min.x>r1max.x+padding || r2min.y>r1max.y+padding || r2max.x<r1min.x-padding || r2max.y<r1min.y
 else return true;
}

public static Rect Extended (this Rect r, float f) { return new Rect(r.x-f, r.y-f, r.width+f*2, r.height+f*2); }

public static Rect Extended (this Rect rect, float r, float l, float t, float b) { return new Rect(rect.x-l, rect.y-t,

public static Rect Encapsulate (this Rect r, Rect n)
{
 if (n.xMin < r.xMin) r.xMin = n.xMin;
 if (n.yMin < r.yMin) r.yMin = n.yMin;
 if (n.xMax > r.xMax) r.xMax = n.xMax;
 if (n.yMax > r.yMax) r.yMax = n.yMax;
 return r;
}

public static Rect TurnNonNegative (this Rect r)
{
 float width = r.width;
 if (width < 0) { r.width = -width; r.x += width; }

 float height = r.height;
 if (height < 0) { r.height = -height; r.y += height; }

 return r;
```

```
}



public static float DistToRectCenter (this Vector3 pos, float offsetX, float offsetZ, float size)
{
 float deltaX = pos.x - (offsetX+size/2); float deltaZ = pos.z - (offsetZ+size/2); float dist = deltaX*deltaX + d
 return Mathf.Sqrt(dist);
}

public static float DistToRectAxisAligned (this Vector3 pos, float offsetX, float offsetZ, float size) //NOT ma
{
 //finding x distance
 float distPosX = offsetX - pos.x;
 float distNegX = pos.x - offsetX - size;

 float distX;
 if (distPosX >= 0) distX = distPosX;
 else if (distNegX >= 0) distX = distNegX;
 else distX = 0;

 //finding z distance
 float distPosZ = offsetZ - pos.z;
 float distNegZ = pos.z - offsetZ - size;

 float distZ;
 if (distPosZ >= 0) distZ = distPosZ;
```

```
 else if (distNegZ >= 0) distZ = distNegZ;
 else distZ = 0;

 //returning the maximum(!) distance
 if (distX > distZ) return distX;
 else return distZ;
}



public static float DistToRectCenter (this Vector3[] poses, float offsetX, float offsetZ, float size)
{
 float minDist = 200000000;
 for (int p=0; p<poses.Length; p++)
 {
  float dist = poses[p].DistToRectCenter(offsetX, offsetZ, size);
  if (dist < minDist) minDist = dist;
 }
 return minDist;
}

public static float DistToRectAxisAligned (this Vector3[] poses, float offsetX, float offsetZ, float size)
{
 float minDist = 200000000;
 for (int p=0; p<poses.Length; p++)
 {
```

```
  float dist = poses[p].DistToRectAxisAligned(offsetX, offsetZ, size);
  if (dist < minDist) minDist = dist;
 }
 return minDist;
}

public static float DistAxisAligned (this Vector3 center, Vector3 pos)
{
 float distX = center.x - pos.x; if (distX<0) distX = -distX;
 float distZ = center.z - pos.z; if (distZ<0) distZ = -distZ;

 //returning the maximum(!) distance
 if (distX > distZ) return distX;
 else return distZ;
}



/*public static Coord ToCoord (this Vector3 pos, float cellSize, bool ceil=false) //to use in object grid
{
 if (!ceil) return new Coord(
  Mathf.FloorToInt((pos.x) / cellSize),
  Mathf.FloorToInt((pos.z) / cellSize) );
 else return new Coord(
  Mathf.CeilToInt((pos.x) / cellSize),
  Mathf.CeilToInt((pos.z) / cellSize) );
```

```
}*/

[Obsolete("Use Coord.Round(v)")] public static Coord RoundToCoord (this Vector2 pos) //to use in spatia
{
 int posX = (int)(pos.x + 0.5f); if (pos.x < 0) posX--; //snippet for RoundToInt
 int posZ = (int)(pos.y + 0.5f); if (pos.y < 0) posZ--;
 return new Coord(posX, posZ);
}

[Obsolete("Use Coord.Floor(v/cellSize)")] public static Coord FloorToCoord (this Vector3 pos, float cellSiz
[Obsolete("Use Coord.Ceil(v/cellSize)")] public static Coord CeilToCoord (this Vector3 pos, float cellSize)
[Obsolete("Use Coord.Round(v/cellSize)")] public static Coord RoundToCoord (this Vector3 pos, float cell
public static CoordRect ToCoordRect (this Vector3 pos, float range, float cellSize) //this one works with Te
{
 Coord min = new Coord( Mathf.FloorToInt((pos.x-range)/cellSize), Mathf.FloorToInt((pos.z-range)/cellSiz
 Coord max = new Coord( Mathf.FloorToInt((pos.x+range)/cellSize), Mathf.FloorToInt((pos.z+range)/cellS
 return new CoordRect(min, max-min);
}

public static CoordRect GetHeightRect (this Terrain terrain)
{
 float pixelSize = terrain.terrainData.size.x / terrain.terrainData.heightmapResolution;

 int posX = (int)(terrain.transform.localPosition.x/pixelSize + 0.5f); if (terrain.transform.localPosition.x < 0)
 int posZ = (int)(terrain.transform.localPosition.z/pixelSize + 0.5f); if (terrain.transform.localPosition.z < 0)
```

```
 return new CoordRect(posX, posZ, terrain.terrainData.heightmapResolution, terrain.terrainData.heightma
}

public static Rect GetWorldRect (this Terrain terrain)
{
 return new Rect(
  terrain.transform.position.x,
  terrain.transform.position.z,
  terrain.terrainData.size.x,
  terrain.terrainData.size.z);
}

public static float[,] SafeGetHeights (this TerrainData data, int offsetX, int offsetZ, int sizeX, int sizeZ)
{
 if (offsetX<0) { sizeX += offsetX; offsetX=0; } if (offsetZ<0) { sizeZ += offsetZ; offsetZ=0; } //Not Tested!
 int res = data.heightmapResolution;
 if (sizeX+offsetX > res) sizeX = res-offsetX; if (sizeZ+offsetZ > res) sizeZ = res-offsetZ;
 return data.GetHeights(offsetX, offsetZ, sizeX, sizeZ);
}

public static float[,,] SafeGetAlphamaps (this TerrainData data, int offsetX, int offsetZ, int sizeX, int sizeZ)
{
 if (offsetX<0) { sizeX += offsetX; offsetX=0; } if (offsetZ<0) { sizeZ += offsetZ; offsetZ=0; } //Not Tested!
 int res = data.alphamapResolution;
 if (sizeX+offsetX > res) sizeX = res-offsetX; if (sizeZ+offsetZ > res) sizeZ = res-offsetZ;
 return data.GetAlphamaps(offsetX, offsetZ, sizeX, sizeZ);
```

```
}

public static float GetInterpolated (this float[,] array, float x, float z)
/// Gets value in-between pixels using linear interpolation. X and Z are swapped.
{
 //neig coords
 int px = (int)x; if (x<0) px--; //because (int)-2.5 gives -2, should be -3
 int nx = px+1;

 int pz = (int)z; if (z<0) pz--;
 int nz = pz+1;

 //reading values
 float val_pxpz = array[px, pz];  //x and z swapped
 float val_nxpz = array[px, nz];
 float val_pxnz = array[nx, pz];
 float val_nxnz = array[nx, nz];

 float percentX = x-px;
 float percentZ = z-pz;

 float val_fz = val_pxpz*(1-percentX) + val_nxpz*percentX;
 float val_cz = val_pxnz*(1-percentX) + val_nxnz*percentX;
 float val = val_fz*(1-percentZ) + val_cz*percentZ;

 return val;
```

```
}
```

```
public static bool Equal (Vector3 v1, Vector3 v2)
{
 return Mathf.Approximately(v1.x, v2.x) &&
   Mathf.Approximately(v1.y, v2.y) &&
   Mathf.Approximately(v1.z, v2.z);
}

public static bool Equal (Ray r1, Ray r2)
{
 return Equal(r1.origin, r2.origin) && Equal(r1.direction, r2.direction);
}


public static Vector3[] InverseTransformPoint (this Transform tfm, Vector3[] points)
{
 for (int c=0; c<points.Length; c++) points[c] = tfm.InverseTransformPoint(points[c]);
 return points;
}

public static Vector3 GetCenter (this Vector3[] poses)
```

```
{
 if (poses.Length == 0) return new Vector3();
 if (poses.Length == 1) return poses[0];

 float x=0; float y=0; float z=0;
 for (int i=0; i<poses.Length; i++)
 {
  x+=poses[i].x;
  y+=poses[i].y;
  z+=poses[i].z;
 }
 return new Vector3(x/poses.Length, y/poses.Length, z/poses.Length);
}

public static bool Approximately (Rect r1, Rect r2)
{
 return  Mathf.Approximately(r1.x, r2.x) &&
   Mathf.Approximately(r1.y, r2.y) &&
   Mathf.Approximately(r1.width, r2.width) &&
   Mathf.Approximately(r1.height, r2.height);
}

public static IEnumerable<Vector3> CircleAround (this Vector3 center, float radius, int numPoints, bool en
{
 float radianStep = 2*Mathf.PI / numPoints;
 if (endWhereStart) numPoints++;
```

```
 for (int i=0; i<numPoints; i++)
 {
  float angle = i*radianStep;
  Vector3 dir = new Vector3( Mathf.Sin(angle), 0, Mathf.Cos(angle) );
  yield return center + dir*radius;
 }
}

public static bool IntersectsLine (this Rect rect, Vector2 from, Vector2 to)
{
 Vector2 rectMin = rect.position;
 Vector2 rectMax = rect.max;

 //if line is absolutely out of the rect
 if ((from.x < rectMin.x  &&  to.x < rectMin.x)  ||
  (from.x > rectMax.x  &&  to.x > rectMax.x)  ||
  (from.y < rectMin.y  &&  to.y < rectMin.y)  ||
  (from.y > rectMax.y  &&  to.y > rectMax.y))
   return false;

 //if line is absolutely within rect (combined with abs out)
 if ((from.x < rectMax.x  &&  to.x < rectMax.x)  ||
  (from.x > rectMin.x  &&  to.x > rectMin.x)  ||
  (from.y < rectMax.y  &&  to.y < rectMax.y)  ||
  (from.y > rectMin.y  &&  to.y > rectMin.y))
   return true;
```

```
 Vector2 rectCorner3 = new Vector2(rectMin.x, rectMax.y);
 Vector2 rectCorner4 = new Vector2(rectMax.x, rectMin.y);

 bool hand1 = rectMin.Handness(from,to) > 0;
 bool hand2 = rectMax.Handness(from,to) > 0;
 bool hand3 = rectCorner3.Handness(from,to) > 0;
 bool hand4 = rectCorner4.Handness(from,to) > 0;

 if (hand1 && hand2 && hand3 && hand4) return false;
 if (!hand1 && !hand2 && !hand3 && !hand4) return false;

 return true;
}

public static float Handness (this Vector2 point, Vector2 from, Vector2 to)
/// Determines whether point lays on left or on right of the line
{
 return (point.x-from.x)*(to.y-from.y) - (point.y-from.y)*(to.x-from.x);
}


public static float DistanceToLine (this Vector3 point, Vector3 lineStart, Vector3 lineEnd)
/// Just helper fn for optimize, isn't related with beizer
{
 Vector3 lineDir = lineStart-lineEnd;
```

```
 float lineLengthSq = lineDir.x*lineDir.x + lineDir.y*lineDir.y + lineDir.z*lineDir.z;
 float lineLength = Mathf.Sqrt(lineLengthSq);
 float startDistance = (lineStart-point).magnitude;
 float endDistance = (lineEnd-point).magnitude;

 //finding height of triangle
 float halfPerimeter = (startDistance + endDistance + lineLength) / 2;
 float square = Mathf.Sqrt( halfPerimeter*(halfPerimeter-endDistance)*(halfPerimeter-startDistance)*(halfP
 float height = 2/lineLength * square;

 //dealing with out of line cases
 float distFromStartSq = startDistance*startDistance - height*height;
 float distFromEndSq = endDistance*endDistance - height*height;

 if (distFromStartSq > lineLengthSq && distFromStartSq > distFromEndSq) return endDistance;
 else if (distFromEndSq > lineLengthSq) return startDistance;
 else return height;
}

public static void ClampLine (this Rect rect, ref Vector2 from, ref Vector2 to, bool checkBothWays=true)
/// Cuts the line, leaving only the segment which is within the rect
/// If checkBothWays disabled just cutting out from-to-rect segment, leaving rect-to-to
{
 Vector2 dir = from-to;

 Vector2 rectMin = rect.position;
```

```
Vector2 rectMax = rect.max;


if (from.x < rectMin.x)
{
 float ratio = (rectMin.x-from.x) / (to.x-from.x); //the number of times big triangle smaller
 from.y = (to.y-from.y) * ratio + from.y;
 from.x = rectMin.x;
}


if (from.x > rectMax.x)
{
 float ratio = (rectMax.x-from.x) / (to.x-from.x);
 from.y = (to.y-from.y) * ratio + from.y;
 from.x = rectMax.x;
}


if (from.y < rectMin.y)
{
 float ratio = (rectMin.y-from.y) / (to.y-from.y);
 from.x = (to.x-from.x) * ratio + from.x;
 from.y = rectMin.y;
}


if (from.y > rectMax.y)
{
 float ratio = (rectMax.y-from.y) / (to.y-from.y);
```

```
    from.x = (to.x-from.x) * ratio + from.x;
    from.y = rectMax.y;
   }

   if (checkBothWays)
    ClampLine(rect, ref to, ref from, checkBothWays=false);
  }
 }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection; //to copy properties


namespace Den.Tools
{
 static public class Extensions
 {


  public static void RemoveChildren (this Transform tfm)
  {
   for (int i=tfm.childCount-1; i>=0; i--)
   {
    Transform child = tfm.GetChild(i);

    #if UNITY_EDITOR
    if (!UnityEditor.EditorApplication.isPlaying)
     GameObject.DestroyImmediate(child.gameObject);
    else
    #endif
     GameObject.Destroy(child.gameObject);
   }
```

```
}

public static Transform FindChildRecursive (this Transform tfm, string name)
{
 int numChildren = tfm.childCount;

 for (int i=0; i<numChildren; i++)
  if (tfm.GetChild(i).name == name) return tfm.GetChild(i);

 for (int i=0; i<numChildren; i++)
 {
  Transform result = tfm.GetChild(i).FindChildRecursive(name);
  if (result != null) return result;
 }

 return null;
}

public static void ToggleDisplayWireframe (this Transform tfm, bool show)
{
 #if UNITY_EDITOR
 #if !UNITY_5_5_OR_NEWER
```

```
  UnityEditor.EditorUtility.SetSelectedWireframeHidden(tfm.GetComponent<Renderer>(), !show);
  int childCount = tfm.childCount;
  for (int c=0; c<childCount; c++) tfm.GetChild(c).ToggleDisplayWireframe(show);
 #else
  UnityEditor.EditorUtility.SetSelectedRenderState(tfm.GetComponent<Renderer>(), show? UnityEditor.E
  int childCount = tfm.childCount;
  for (int c=0; c<childCount; c++) tfm.GetChild(c).ToggleDisplayWireframe(show);
 #endif
 #endif
}

public static int ToInt (this Coord coord)
{
 int absX = coord.x<0? -coord.x : coord.x;
 int absZ = coord.z<0? -coord.z : coord.z;

 return ((coord.z<0? 1000000000 : 0) + absX*30000 + absZ) * (coord.x<0? -1 : 1);
}

public static Coord ToCoord (this int hash)
{
 int absHash = hash<0? -hash : hash;
 int sign = (absHash/1000000000)*1000000000;

 int absX = (absHash - sign)/30000;
 int absZ = absHash - sign - absX*30000;
```

```
 return new Coord(hash<0? -absX : absX, sign==0? absZ : -absZ);
}

public static TValue[] ToArray<TKey,TValue> (this Dictionary<TKey,TValue>.ValueCollection values)
{
 TValue[] arr = new TValue[values.Count];
 values.CopyTo(arr, 0);
 return arr;
}

public static TKey[] ToArray<TKey,TValue> (this Dictionary<TKey,TValue>.KeyCollection keys)
{
 TKey[] arr = new TKey[keys.Count];
 keys.CopyTo(arr, 0);
 return arr;
}

public static T[] ToArray<T> (this ICollection<T> col)
{
 T[] arr = new T[col.Count];
 col.CopyTo(arr, 0);
 return arr;
}

public static T[] ToArray<T> (this HashSet<T> vals)
```

```
{
 T[] arr = new T[vals.Count];
 vals.CopyTo(arr, 0);
 return arr;
}

public static void AddRange<TKey,TValue> (this Dictionary<TKey,TValue> dict, TKey[] keys, TValue[] val
{
 for (int i=0; i<keys.Length; i++)
  dict.Add(keys[i], values[i]);
}

public static void TryAdd<TKey,TValue> (this Dictionary<TKey,TValue> dict, TKey key, TValue value) { if

public static void ForceAdd<TKey,TValue> (this Dictionary<TKey,TValue> dict, TKey key, TValue value)
{
 if (dict.ContainsKey(key))
  dict[key] = value;
 else dict.Add(key, value);
}

public static void TryRemove<TKey,TValue> (this Dictionary<TKey,TValue> dict, TKey key) { if (dict.Cont

public static void RemoveNotContained<TKey,TValue> (this Dictionary<TKey,TValue> dict, HashSet<TK
/// Removes everything but what is inside Contained
{
```

```
 List<TKey> keysToRemove = null;

 foreach (TKey key in dict.Keys)
  if (!contained.Contains(key))
  {
   if (keysToRemove == null) keysToRemove = new List<TKey>(); //create list only if there's something to
   keysToRemove.Add(key);
  }

 if (keysToRemove != null)
  foreach (TKey key in keysToRemove)
   dict.Remove(key);
}

public static void RemoveWhere<TKey,TValue> (this Dictionary<TKey,TValue> dict, Predicate<TKey> pr
{
 List<TKey> keysToRemove = null;

 foreach (TKey key in dict.Keys)
  if (predicate(key))
  {
   if (keysToRemove == null) keysToRemove = new List<TKey>(); //create list only if there's something to
   keysToRemove.Add(key);
  }

 if (keysToRemove != null)
```

```
  foreach (TKey key in keysToRemove)
    dict.Remove(key);
}

public static TValue CheckGet<TKey,TValue> (this Dictionary<TKey,TValue> dict, TKey key)
{
 if (dict.ContainsKey(key)) return dict[key];
 else return default(TValue);
}
public static TKey AnyKey<TKey,TValue> (this Dictionary<TKey,TValue> dict)
{
 foreach (KeyValuePair<TKey,TValue> kvp in dict)
  return kvp.Key;
 return default(TKey);
}
public static TValue AnyValue<TKey,TValue> (this Dictionary<TKey,TValue> dict)
{
 foreach (KeyValuePair<TKey,TValue> kvp in dict)
  return kvp.Value;
 return default(TValue);
}
public static T Any<T> (this HashSet<T> hashSet)
{
 foreach (T val in hashSet)
  return val;
 return default(T);
```

```
}

public static void RemoveAfter<T> (this List<T> list, int num)
{
 list.RemoveRange(num+1, list.Count-num-1);
}

public static T ElementOfType<T> (this IEnumerable arr) where T : class
{
 foreach (object mem in arr)
  if (mem is T objMem)
   return objMem;
 return default;
}

public static bool MinDist<T1,T2> (this Dictionary<T1,T2> dict, Func<T1,float> distEvaluator, out float min
{
 bool minFound = false;
 minDist = int.MaxValue;
 minValue = default;
 foreach (var kvp in dict)
 {
  float dist = distEvaluator(kvp.Key);
  if (dist < minDist)
  {
   minDist = dist;
```

```
   minFound = true;
   minValue = kvp.Value;
  }
 }
 return minFound;
}

public static void AddRange<T> (this HashSet<T> set, T[] objs) { for (int i=0; i<objs.Length; i++) set.Add(o
public static void CheckAdd<T> (this HashSet<T> set, T obj) { if (!set.Contains(obj)) set.Add(obj); }
public static void CheckRemove<T> (this HashSet<T> set, T obj) { if (set.Contains(obj)) set.Remove(obj);
public static void SetState<T> (this HashSet<T> set, T obj, bool state)
{
 if (state && !set.Contains(obj)) set.Add(obj);
 if (!state && set.Contains(obj)) set.Remove(obj);
}

public static void Normalize (this float[,,] array, int pinnedLayer)
{
 int maxX = array.GetLength(0); int maxZ = array.GetLength(1); int numLayers = array.GetLength(2);
 for (int x=0; x<maxX; x++)
  for (int z=0; z<maxZ; z++)
 {
  float othersSum = 0;

  for (int i=0; i<numLayers; i++)
  {
```

```
   if (i==pinnedLayer) continue;
   othersSum += array[x,z,i];
  }

  float pinnedValue = array[x,z,pinnedLayer];
  if (pinnedValue > 1) { pinnedValue = 1; array[x,z,pinnedLayer] = 1; }
  if (pinnedValue < 0) { pinnedValue = 0; array[x,z,pinnedLayer] = 0; }

  float othersTargetSum = 1 - pinnedValue;
  float factor = othersSum>0? othersTargetSum / othersSum : 0;

  for (int i=0; i<numLayers; i++)
  {
   if (i==pinnedLayer) continue;
    array[x,z,i] *= factor;
  }
 }

}

public static void DrawDebug (this Vector3 pos, float range=1, Color color=new Color())
{
 if (color.a<0.001f) color = Color.white;
 Debug.DrawLine(pos + new Vector3(-1,0,1)*range, pos + new Vector3(1,0,1)*range, color);
 Debug.DrawLine(pos + new Vector3(1,0,1)*range, pos + new Vector3(1,0,-1)*range, color);
 Debug.DrawLine(pos + new Vector3(1,0,-1)*range, pos + new Vector3(-1,0,-1)*range, color);
```

```
 Debug.DrawLine(pos + new Vector3(-1,0,-1)*range, pos + new Vector3(-1,0,1)*range, color);
}

public static void DrawDebug (this Rect rect, Color color=new Color())
{
 if (color.a<0.001f) color = Color.white;
 Debug.DrawLine( new Vector3(rect.x,0,rect.y),       new Vector3(rect.x+rect.width,0,rect.y),    color);
 Debug.DrawLine( new Vector3(rect.x+rect.width,0,rect.y),    new Vector3(rect.x+rect.width,0,rect.y+rect.h
 Debug.DrawLine( new Vector3(rect.x+rect.width,0,rect.y+rect.height), new Vector3(rect.x,0,rect.y+rect.he
 Debug.DrawLine( new Vector3(rect.x,0,rect.y+rect.height),    new Vector3(rect.x,0,rect.y),       color);
}


public static Transform AddChild (this Transform tfm, string name="", Vector3 offset=new Vector3())
{
 GameObject go = new GameObject();
 go.name = name;
 go.transform.parent = tfm;
 go.transform.localPosition = offset;

 return go.transform;
}

public static T CreateObjectWithComponent<T> (string name="", Transform parent=null, Vector3 offset=n
{
 GameObject go = new GameObject();
```

```
 if (name != null)
 if (parent != null) go.transform.parent = parent.transform;
 go.transform.localPosition = offset;

 return go.AddComponent<T>();
}



public static float EvaluateMultithreaded (this AnimationCurve curve, float time)
{
 int keyCount = curve.keys.Length;

 if (time <= curve.keys[0].time) return curve.keys[0].value;
 if (time >= curve.keys[keyCount-1].time) return curve.keys[keyCount-1].value;

 int keyNum = 0;
 for (int k=0; k<keyCount-1; k++)
 {
  if (curve.keys[keyNum+1].time > time) break;
  keyNum++;
 }

 float delta = curve.keys[keyNum+1].time - curve.keys[keyNum].time;
 float relativeTime = (time - curve.keys[keyNum].time) / delta;
```

```
 float timeSq = relativeTime * relativeTime;
 float timeCu = timeSq * relativeTime;

 float a = 2*timeCu - 3*timeSq + 1;
 float b = timeCu - 2*timeSq + relativeTime;
 float c = timeCu - timeSq;
 float d = -2*timeCu + 3*timeSq;

 return a*curve.keys[keyNum].value + b*curve.keys[keyNum].outTangent*delta + c*curve.keys[keyNum+1
}

public static bool IdenticalTo (this AnimationCurve c1, AnimationCurve c2)
{
 if (c1==null || c2==null) return false;
 if (c1.keys.Length != c2.keys.Length) return false;

 int numKeys = c1.keys.Length;
 for (int k=0; k<numKeys; k++)
 {
  if (c1.keys[k].time != c2.keys[k].time ||
   c1.keys[k].value != c2.keys[k].value ||
   c1.keys[k].inTangent != c2.keys[k].inTangent ||
   c1.keys[k].outTangent != c2.keys[k].outTangent)
    return false;
 }
```

```
 return true;
}

public static Keyframe[] Copy (this Keyframe[] src)
{
 Keyframe[] dst = new Keyframe[src.Length];
 for (int k=0; k<src.Length; k++)
 {
  dst[k].value = src[k].value;
  dst[k].time = src[k].time;
  dst[k].inTangent = src[k].inTangent;
  dst[k].outTangent = src[k].outTangent;
 }
 return dst;
}

public static AnimationCurve Copy (this AnimationCurve src)
{
 AnimationCurve dst = new AnimationCurve();
 dst.keys = src.keys.Copy();
 return dst;
}
```

```
public static object Parse (this string s, Type t)
{
 //better than creating xml serializer each time. Reverse to "ToString" function

 if (s.Contains("=")) s = s.Remove(0, s.IndexOf('=')+1); //removing everything before =

 object r = null;
 if (t == typeof(float)) r = float.Parse(s);
 else if (t == typeof(int)) r = int.Parse(s);
 else if (t == typeof(bool)) r = bool.Parse(s);
 else if (t == typeof(string)) r = s;
 else if (t == typeof(byte)) r = byte.Parse(s);
 else if (t == typeof(short)) r = short.Parse(s);
 else if (t == typeof(long)) r = long.Parse(s);
 else if (t == typeof(double)) r = double.Parse(s);
 else if (t == typeof(char)) r = char.Parse(s);
 else if (t == typeof(decimal)) r = decimal.Parse(s);
 else if (t == typeof(sbyte)) r = sbyte.Parse(s);
 else if (t == typeof(uint)) r = uint.Parse(s);
 else if (t == typeof(ulong)) r = ulong.Parse(s);
 else return null;

 return r;
}
```

```
public static bool isPlaying
{get{
 #if UNITY_EDITOR
  return UnityEditor.EditorApplication.isPlaying; //if not playing
 #else
  return true;
 #endif
}}

public static bool IsEditor ()
{
 #if UNITY_EDITOR
  return
   !UnityEditor.EditorApplication.isPlaying; //if not playing
   //(UnityEditor.EditorWindow.focusedWindow != null && UnityEditor.EditorWindow.focusedWindow.GetT
   //UnityEditor.SceneView.lastActiveSceneView == UnityEditor.EditorWindow.focusedWindow; //if scene v
 #else
  return false;
 #endif
}

public static bool IsSelected (Transform transform)
{
 #if UNITY_EDITOR
  return UnityEditor.Selection.activeTransform == transform;
```

```
 #else
  return false;
 #endif
}

public static Camera GetMainCamera ()
{
 if (IsEditor())
 {
  #if UNITY_EDITOR
  if (UnityEditor.SceneView.lastActiveSceneView==null) return null;
  else return UnityEditor.SceneView.lastActiveSceneView.camera;
  #else
  return null;
  #endif
 }
 else
 {
  Camera mainCam = Camera.main;
  if (mainCam==null) mainCam = GameObject.FindObjectOfType<Camera>(); //in case it was destroyed c
  return mainCam;
 }
}

public static Vector3[] GetCamPoses (bool genAroundMainCam=true, string genAroundTag=null, Vector3
{
```

```
if (IsEditor())
{
 #if UNITY_EDITOR
 if (UnityEditor.SceneView.lastActiveSceneView==null || UnityEditor.SceneView.lastActiveSceneView.ca
 if (camPoses==null || camPoses.Length!=1) camPoses = new Vector3[1];
 camPoses[0] = UnityEditor.SceneView.lastActiveSceneView.camera.transform.position;
 #else
 camPoses = new Vector3[1];
 #endif
}
else
{
 //finding objects with tag
 GameObject[] taggedObjects = null;
 if (genAroundTag!=null && genAroundTag.Length!=0) taggedObjects = GameObject.FindGameObjectsW

 //calculating cams array length and rescaling it
 int camPosesLength = 0;
 if (genAroundMainCam) camPosesLength++;
 if (taggedObjects !=null) camPosesLength += taggedObjects.Length;

 if (camPosesLength == 0) { Debug.LogError("No Main Camera to deploy"); return new Vector3[0]; }
 if (camPoses == null || camPosesLength != camPoses.Length) camPoses = new Vector3[camPosesLen

 //filling cams array
 int counter = 0;
```

```
  if (genAroundMainCam)
  {
   Camera mainCam = Camera.main;
   if (mainCam==null) mainCam = GameObject.FindObjectOfType<Camera>(); //in case it was destroyed
   camPoses[0] = mainCam.transform.position;
   counter++;
  }
  if (taggedObjects != null)
   for (int i=0; i<taggedObjects.Length; i++) camPoses[i+counter] = taggedObjects[i].transform.position;
 }

 return camPoses;
}

public static Vector2 GetMousePosition ()
{
 if (IsEditor())
 {
  #if UNITY_EDITOR
  UnityEditor.SceneView sceneview = UnityEditor.SceneView.lastActiveSceneView;
  if (sceneview==null || sceneview.camera==null || Event.current==null) return Vector2.zero;
  Vector2 mousePos = Event.current.mousePosition;
  mousePos = new Vector2(mousePos.x/sceneview.camera.pixelWidth, mousePos.y/sceneview.camera.p
  #if UNITY_5_4_OR_NEWER
  mousePos *= UnityEditor.EditorGUIUtility.pixelsPerPoint;
  #endif
```

```
  mousePos.y = 1 - mousePos.y;
  return mousePos;
  #else
  return Input.mousePosition;
  #endif
 }
 else return Input.mousePosition;
}

public static void GizmosDrawFrame (Vector3 center, Vector3 size, int resolution, float level = 30)
{
 Vector3 offset = center-size/2;

 Vector3 prevP1=Vector3.zero; Vector3 prevP2=Vector3.zero;
 for (float x=0; x < size.x+0.0001f; x += 1f*size.x/resolution)
 {
  RaycastHit hit = new RaycastHit();

  Vector3 p1 = new Vector3(offset.x+x, 10000, offset.z);
  if (Physics.Raycast(new Ray(p1, Vector3.down*20000), out hit, 20000)) p1.y = hit.point.y;
  else if (Physics.Raycast(new Ray(p1+new Vector3(1,0,0), Vector3.down*20000), out hit, 20000)) p1.y =
  else if (Physics.Raycast(new Ray(p1+new Vector3(-1,0,0), Vector3.down*20000), out hit, 20000)) p1.y =
  else if (Physics.Raycast(new Ray(p1+new Vector3(0,0,1), Vector3.down*20000), out hit, 20000)) p1.y =
  else if (Physics.Raycast(new Ray(p1+new Vector3(0,0,-1), Vector3.down*20000), out hit, 20000)) p1.y =
  else p1.y = level;
  if (x>0.0001f) Gizmos.DrawLine(prevP1, p1);
```

```
 prevP1 = p1;

 Vector3 p2 = new Vector3(offset.x+x, 10000, offset.z+size.z);
 if (Physics.Raycast(new Ray(p2, Vector3.down*20000), out hit, 20000)) p2.y = hit.point.y;
 else if (Physics.Raycast(new Ray(p2+new Vector3(1,0,0), Vector3.down*20000), out hit, 20000)) p2.y =
 else if (Physics.Raycast(new Ray(p2+new Vector3(-1,0,0), Vector3.down*20000), out hit, 20000)) p2.y =
 else if (Physics.Raycast(new Ray(p2+new Vector3(0,0,1), Vector3.down*20000), out hit, 20000)) p2.y =
 else if (Physics.Raycast(new Ray(p2+new Vector3(0,0,-1), Vector3.down*20000), out hit, 20000)) p2.y =
 else p2.y = level;
 if (x>0.0001f) Gizmos.DrawLine(prevP2, p2);
 prevP2 = p2;
}

for (float z=0; z < size.z+0.0001f; z += 1f*size.z/resolution)
{
 RaycastHit hit = new RaycastHit();

 Vector3 p1 = new Vector3(offset.x, 10000, offset.z+z);
 if (Physics.Raycast(new Ray(p1, Vector3.down*20000), out hit, 20000)) p1.y = hit.point.y;
 else if (Physics.Raycast(new Ray(p1+new Vector3(1,0,0), Vector3.down*20000), out hit, 20000)) p1.y =
 else if (Physics.Raycast(new Ray(p1+new Vector3(-1,0,0), Vector3.down*20000), out hit, 20000)) p1.y =
 else if (Physics.Raycast(new Ray(p1+new Vector3(0,0,1), Vector3.down*20000), out hit, 20000)) p1.y =
 else if (Physics.Raycast(new Ray(p1+new Vector3(0,0,-1), Vector3.down*20000), out hit, 20000)) p1.y =
 else p1.y = level;
 if (z>0.0001f) Gizmos.DrawLine(prevP1, p1);
 prevP1 = p1;
```

```
  Vector3 p2 = new Vector3(offset.x+size.x, 10000, offset.z+z);
  if (Physics.Raycast(new Ray(p2, Vector3.down*20000), out hit, 20000)) p2.y = hit.point.y;
  else if (Physics.Raycast(new Ray(p2+new Vector3(1,0,0), Vector3.down*20000), out hit, 20000)) p2.y =
  else if (Physics.Raycast(new Ray(p2+new Vector3(-1,0,0), Vector3.down*20000), out hit, 20000)) p2.y =
  else if (Physics.Raycast(new Ray(p2+new Vector3(0,0,1), Vector3.down*20000), out hit, 20000)) p2.y =
  else if (Physics.Raycast(new Ray(p2+new Vector3(0,0,-1), Vector3.down*20000), out hit, 20000)) p2.y =
  else p2.y = level;
  if (z>0.0001f) Gizmos.DrawLine(prevP2, p2);
  prevP2 = p2;
 }
}

public static void Planar (this Mesh mesh, float size, int resolution)
{
 float step = size / resolution;

 Vector3[] verts = new Vector3[(resolution+1)*(resolution+1)];
 Vector2[] uvs = new Vector2[verts.Length];
 int[] tris = new int[resolution*resolution*2*3];

 int vertCounter = 0;
 int triCounter = 0;
 for (float x=0; x<size+0.001f; x+=step) //including max
  for (float z=0; z<size+0.001f; z+=step)
 {
```

```
  verts[vertCounter] = new Vector3(x,0,z);
  uvs[vertCounter] = new Vector2(x/size, z/size);

  if (x>0.001f && z>0.001f)
  {
   tris[triCounter] = vertCounter-(resolution+1);  tris[triCounter+1] = vertCounter-1;     tris[triCounter+2] = ve
   tris[triCounter+3] = vertCounter-1;     tris[triCounter+4] = vertCounter-(resolution+1); tris[triCounter+5] =
   triCounter += 6;
  }

  vertCounter++;
 }

 mesh.Clear();
 mesh.vertices = verts;
 mesh.uv = uvs;
 mesh.triangles = tris;
}

/* //use layout instead
public static T Save<T> (this T data, string label="Save Data as Unity Asset", string fileName="Data.asse
{
 #if UNITY_EDITOR
 //finding path
 string path= UnityEditor.EditorUtility.SaveFilePanel(label, "Assets", fileName, "asset");
 if (path==null || path.Length==0) return data;
```

```
 //releasing data on re-save
 T newData = data;
 if (UnityEditor.AssetDatabase.Contains(data)) newData = (T)data.Clone();

 //saving
 path = path.Replace(Application.dataPath, "Assets");
 if (undoObj != null) UnityEditor.Undo.RecordObject(undoObj, undoName); //TODO: undo is actually not re
 //undoObj.setDirty = !undoObj.setDirty;
 UnityEditor.AssetDatabase.CreateAsset(newData, path);
 if (undoObj != null) UnityEditor.EditorUtility.SetDirty(undoObj);

 return newData;
 #else
 return data;
 #endif
}

public static T LoadAsset<T> (string label="Load Unity Asset", string[] filters=null) where T : UnityEngine.O
{
 #if UNITY_EDITOR
 if (filters==null && typeof(T).IsSubclassOf(typeof(Texture))) filters = new string[] { "Textures", "PSD,TIFF,
 ArrayTools.Add(ref filters, "All files");
 ArrayTools.Add(ref filters, "*");

 string path= UnityEditor.EditorUtility.OpenFilePanelWithFilters(label, "Assets", filters);
```

```
 if (path!=null && path.Length!=0)
 {
  path = path.Replace(Application.dataPath, "Assets");
  T asset = (T)UnityEditor.AssetDatabase.LoadAssetAtPath(path, typeof(T));
  return asset;
 }
 return null;
 #endif
}*/

/*public static object Invoke (object target, string methodName, params object[] paramArray)
{
 Type type = target.GetType();
 MethodInfo methodInfo = type.GetMethod(methodName);
 return methodInfo.Invoke(target, paramArray);
}

public static object StaticInvoke (string className, string methodName, params object[] paramArray)
{
 Type type = Type.GetType(className);
 MethodInfo methodInfo = type.GetMethod(methodName, BindingFlags.Public | BindingFlags.Static);
 return methodInfo.Invoke(null, new object[] { paramArray });
}*/

public static string LogBinary (this int src)
{
```

```
 string result = "";
 for (int i=0; i<32; i++)
 {
  if (i%4==0) result=" "+result;
  result = (src & 0x1) + result;


  src = src >> 1;
 }
 return result;
}

public static string ToStringArray<T> (this T[] array)
{
 string result = "";
 for (int i=0; i<array.Length; i++)
 {
  result += array[i].ToString();
  if (i!=array.Length-1) result += ",";
 }
 return result;
}

public static Color[] ToColors (this Vector4[] src)
{
 Color[] dst = new Color[src.Length];
```

```
 for (int i=0; i<src.Length; i++)
  dst[i] = src[i];
 return dst;
}

public static Texture2D GetBiggestTexture (this Texture2D[] textures)
/// Finds the biggest texture in array. Useful to create TextureArrays
{
 int maxResolution = 0;
 int maxNum = -1;

 for (int i=0; i<textures.Length; i++)
 {
  if (textures[i]==null) continue;

  if (textures[i].width > maxResolution) { maxResolution = textures[i].width; maxNum = i; }
  if (textures[i].height > maxResolution) { maxResolution = textures[i].height; maxNum = i; }
 }

 if (maxNum >=0) return (textures[maxNum]);
 else return null;
}

/*public static string ToStringMemberwise<T> (this List<T> list)
/// prints list as one, two, three
{
```

```
 string result = "";
 for (int i=0; i<list.Count; i++)
 {
  result += list[i];
  if (i!=list.Count-1) result += ", ";
 }
 return result;
}*/

public static string ToStringMemberwise<T> (this IEnumerable list, Func<T,string> toStringFn=null)
/// prints list as one, two, three
{
 string result = "";
 foreach (T obj in list)
 {
  if (toStringFn==null) result += obj.ToString();
  else result += toStringFn(obj);
  result += ", ";
 }
 if (result.Length>=2) result = result.Substring(0, result.Length-2);
 return result;
}


public static string Nicify (this string camelCase)
{
```

```
 string titleCase = System.Text.RegularExpressions.Regex.Replace(camelCase, @"(\B[A-Z])", @" $1"); //
 return titleCase.Substring(0, 1).ToUpper() + titleCase.Substring(1);
}



public static void CheckSetInt (this Material mat, string name, int val) { if (mat.HasProperty(name)) mat.Se
public static void CheckSetFloat (this Material mat, string name, float val) { if (mat.HasProperty(name)) ma
public static void CheckSetTexture (this Material mat, string name, Texture tex) { if (mat.HasProperty(nam
public static void CheckSetVector (this Material mat, string name, Vector4 val) { if (mat.HasProperty(name
public static void CheckSetColor (this Material mat, string name, Color val) { if (mat.HasProperty(name)) m



public class WeakRefComparer<T> : IEqualityComparer<WeakReference<T>> where T : class
{
 public bool Equals ( WeakReference<T> wr2, T t1)
 {
  //if (!wr1.TryGetTarget(out T t1)) return false;
  if (!wr2.TryGetTarget(out T t2)) return false;
  return t1==t2;
 }

 public bool Equals (T t1, WeakReference<T> wr2)
 {
  //if (!wr1.TryGetTarget(out T t1)) return false;
  if (!wr2.TryGetTarget(out T t2)) return false;
```

```
  return t1==t2;
 }

 public bool Equals (WeakReference<T> wr1, WeakReference<T> wr2)
 {
  if (!wr1.TryGetTarget(out T t1)) return false;
  if (!wr2.TryGetTarget(out T t2)) return false;
  return t1==t2;
 }

 public int GetHashCode (WeakReference<T> obj)
 {
  if (!obj.TryGetTarget(out T t)) return 0;
  return t.GetHashCode();
 }
}

public static int Max<T> (this IEnumerable<T> enumerable, Func<T,int> getNumFn)
{
 int max = int.MinValue;
 foreach (T element in enumerable)
 {
  int num = getNumFn(element);
  if (num > max)
   max = num;
 }
```

```
 return max;
}

public static uint Max<T> (this IEnumerable<T> enumerable, Func<T,uint> getNumFn)
{
 uint max = uint.MinValue;
 foreach (T element in enumerable)
 {
  uint num = getNumFn(element);
  if (num > max)
   max = num;
 }
 return max;
}

public static bool Contains<T> (this List<T> list, T val, out int index)
/// Checks if list contains value and returns it's index
{
 index = list.IndexOf(val);
 return index >= 0;
}

public static T GetInRange<T> (this List<T> list, int index) where T: class
/// Gets object if it is within list range, null if not
{
 if (index>0 && index<list.Count)
```

```
    return list[index];
   else
    return null;
  }

  public static void ExtendSet<T> (this List<T> list, T val, int index)
  /// Sets val at index, resizing list if necessary
  {
   if (list.Count-1 < index)
    list.AddRange( new T[index-(list.Count-1)] );
   list[index] = val;
  }



  #if UNITY_EDITOR
  public static UnityEditor.EditorWindow GetInspectorWindow ()
  {
   var editorAsm = typeof(UnityEditor.Editor).Assembly;
   var inspWndType = editorAsm.GetType("UnityEditor.InspectorWindow");
   return UnityEditor.EditorWindow.GetWindow(inspWndType);
  }
  #endif

 }//extensions
}//namespace
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection; //to copy properties

using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

namespace Den.Tools
{
 static public class ReflectionExtensions
 {
  public static object CallStaticMethodFrom (string assembly, string type, string method, params object[] pa
  /// Used for debug purposes only
  {
   // editor assembly is Assembly-CSharp-Editor (or typeof(CustomEditor).Assembly)
   // main is Assembly-CSharp

   Assembly a = Assembly.Load(assembly);
   Type t = a.GetType(type);
   return t.GetMethod(method).Invoke(null, parameters);
  }

  public static void GetPropertiesFrom<T1,T2> (this T1 dst, T2 src) where T1:class where T2:class
```

```
{
 PropertyInfo[] srcProps = src.GetType().GetProperties(BindingFlags.Instance | BindingFlags.Public | Binc
 PropertyInfo[] dstProps = src.GetType().GetProperties(BindingFlags.Instance | BindingFlags.Public | Binc

 for (int sp=0; sp<srcProps.Length; sp++)
  for (int dp=0; dp<dstProps.Length; dp++)
 {
  if (srcProps[sp].Name==dstProps[dp].Name && dstProps[dp].CanWrite)
   dstProps[dp].SetValue(dst, srcProps[sp].GetValue(src, null), null);
 }
       }


public static IEnumerable<FieldInfo> UsableFields (this Type type, bool nonPublic=false, bool includeStat
{
 BindingFlags flags = BindingFlags.Public | BindingFlags.Instance;
 if (nonPublic) flags = flags | BindingFlags.NonPublic;
 if (includeStatic) flags = flags | BindingFlags.Static;

 FieldInfo[] fields = type.GetFields(flags);
 for (int i=0; i<fields.Length; i++)
 {
  FieldInfo field = fields[i];
  if (field.IsLiteral) continue; //leaving constant fields blank
  if (field.FieldType.IsPointer) continue; //skipping pointers (they make unity crash. Maybe require unsafe)
  if (field.IsNotSerialized) continue;
```

```
  yield return field;
 }
}

public static IEnumerable<PropertyInfo> UsableProperties (this Type type, bool nonPublic=false, bool ski
{
 BindingFlags flags;
 if (nonPublic) flags = BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.Instance;
 else flags = BindingFlags.Public | BindingFlags.Instance;

 PropertyInfo[] properties = type.GetProperties(flags);
 for (int i=0;i<properties.Length;i++)
 {
  PropertyInfo prop = properties[i];
  if (!prop.CanWrite) continue;
  if (skipItems && prop.Name=="Item") continue; //ignoring this[x]

  yield return prop;
 }
}

public static IEnumerable<MemberInfo> UsableMembers (this Type type, bool nonPublic=false, bool skip
{
 BindingFlags flags;
 if (nonPublic) flags = BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.Instance;
```

```
 else flags = BindingFlags.Public | BindingFlags.Instance;


 FieldInfo[] fields = type.GetFields(flags);
 for (int i=0; i<fields.Length; i++)
 {
  FieldInfo field = fields[i];
  if (field.IsLiteral) continue; //leaving constant fields blank
  if (field.FieldType.IsPointer) continue; //skipping pointers (they make unity crash. Maybe require unsafe)
  if (field.IsNotSerialized) continue;

  yield return field;
 }

 PropertyInfo[] properties = type.GetProperties(flags);
 for (int i=0;i<properties.Length;i++)
 {
  PropertyInfo prop = properties[i];
  if (!prop.CanWrite) continue;
  if (skipItems && prop.Name=="Item") continue; //ignoring this[x]

  yield return prop;
 }
}


public static void PrintAllFields (this Type type, BindingFlags flags)
```

```
{
 FieldInfo[] fields = type.GetFields();
 for (int i=0; i<fields.Length; i++) Debug.Log(fields[i].Name + ", field, " + flags.ToString());

 PropertyInfo[] props = type.GetProperties(flags);
 for (int i=0; i<props.Length; i++) Debug.Log(props[i].Name + ", property, " + flags.ToString());

 MethodInfo[] methods = type.GetMethods(flags);
 for (int i=0; i<methods.Length; i++) Debug.Log(methods[i].Name + ", method, " + flags.ToString());
}

public static void PrintAllFields (this Type type)
{
 BindingFlags flags = BindingFlags.Public | BindingFlags.Instance;
 type.PrintAllFields(flags);

 flags = BindingFlags.NonPublic | BindingFlags.Instance;
 type.PrintAllFields(flags);

 flags = BindingFlags.Public | BindingFlags.Static;
 type.PrintAllFields(flags);

 flags = BindingFlags.NonPublic | BindingFlags.Static;
 type.PrintAllFields(flags);

 //type.PrintAllFields();
```

```
}

public static Component CopyComponent (Component src, GameObject go)
{
 System.Type type = src.GetType();

 Component dst = go.GetComponent(src.GetType());
 if (dst==null) dst = go.AddComponent(type);

 foreach (FieldInfo field in type.UsableFields(nonPublic:true)) field.SetValue(dst, field.GetValue(src));
 foreach (PropertyInfo prop in type.UsableProperties(nonPublic:true))
 {
  if (prop.Name == "name") continue;
  try {prop.SetValue(dst, prop.GetValue(src, null), null); }
  catch { }
 }

 return dst;
}

/*public static List<Type> GetAllChildTypes (this Type type)
{
 List<Type> result = new List<Type>();

 Assembly assembly = Assembly.GetAssembly(type);
 Type[] allTypes = assembly.GetTypes();
```

```
 for (int i=0; i<allTypes.Length; i++)
  if (allTypes[i].IsSubclassOf(type)) result.Add(allTypes[i]); //nb: IsAssignableFrom will return derived class

 return result;
}*/

[Obsolete] public static IEnumerable<Type> SubtypesEnumerable (this Type parent)
{
 Assembly assembly = Assembly.GetAssembly(parent);  //Assembly[] assembleys = AppDomain.CurrentD
 Type[] types = assembly.GetTypes();
 for (int t=0; t<types.Length; t++)
 {
  Type type = types[t];
  if (type.IsSubclassOf(parent) && !type.IsInterface && !type.IsAbstract) yield return type;
 }
}

public static Type[] Subtypes (this Type parent, bool allAssemblies=false)
{
 List<Type> children = new List<Type>();

 Assembly[] assembleys;
 if (allAssemblies) assembleys = AppDomain.CurrentDomain.GetAssemblies();
 else assembleys = new Assembly[] { Assembly.GetAssembly(parent) };

 foreach (Assembly assembly in assembleys)
```

```
 {
  Type[] types = assembly.GetTypes();
  for (int t=0; t<types.Length; t++)
  {
   Type type = types[t];
   if (type.IsInterface || type.IsAbstract) continue;

   if (type.IsSubclassOf(parent) || parent.IsAssignableFrom(type)) children.Add(type);
  }
 }
 return children.ToArray();
}

public static Type GetTerrainInspectorType ()
{
 //System.Reflection.Assembly a = System.Reflection.Assembly.Load("Assembly-CSharp-Editor");
 //return a.GetType("TerrainInspector"); //does not work

 #if UNITY_EDITOR
 Type[] tmpTypes = Assembly.GetAssembly(typeof(UnityEditor.Editor)).GetTypes();
 for (int i=tmpTypes.Length-1; i>=0; i--) //from the end
 {
  if (tmpTypes[i].Name=="TerrainInspector")
   return tmpTypes[i];
 }
 #endif
```

```
 return null;
}

public static object GetTerrainInspectorField (string fieldName, Type inspectorType=null)
{
 if (inspectorType==null) inspectorType = GetTerrainInspectorType();

 object[] editors = Resources.FindObjectsOfTypeAll(inspectorType);
 for (int i=0; i<editors.Length; i++)
 {
  PropertyInfo toolProp = inspectorType.GetProperty(fieldName, BindingFlags.Instance | BindingFlags.No
  object field = toolProp.GetValue(editors[i], null);
  if (field != null) return field;
 }
 return null;
}

public static void SetTerrainInspectorField (string fieldName, object obj, Type inspectorType=null)
{
 if (inspectorType==null) inspectorType = GetTerrainInspectorType();

 object[] editors = Resources.FindObjectsOfTypeAll(inspectorType);
 for (int i=0; i<editors.Length; i++)
 {
  PropertyInfo toolProp = inspectorType.GetProperty(fieldName, BindingFlags.Instance | BindingFlags.No
```

```
  toolProp.SetValue(editors[i], obj, null);

  //moving component up to refresh terrain tool state
  //UnityEditorInternal.ComponentUtility.MoveComponentUp(script);
 }
}

public static int GetVersion<T> (this HashSet<T> hashSet)
{
 FieldInfo field = typeof(HashSet<T>).GetField("_version", BindingFlags.NonPublic | BindingFlags.Instanc
 return (int)field.GetValue(hashSet);
}

public static int GetVersion<TK,TV> (this Dictionary<TK,TV> dict)
{
 FieldInfo field = typeof(Dictionary<TK,TV>).GetField("version", BindingFlags.NonPublic | BindingFlags.In
 return (int)field.GetValue(dict);
}


public static T GetAddComponent<T> (this GameObject go) where T:Component
{
 T c = go.GetComponent<T>();
 if (c==null) c = go.AddComponent<T>();
 return c;
```

```
  }



  public static void ReflectionReset<T> (this T obj)
  {
    Type type = obj.GetType();
    T empty = (T)Activator.CreateInstance(type);

    foreach (FieldInfo field in type.UsableFields(nonPublic:true)) field.SetValue(obj, field.GetValue(empty));
    foreach (PropertyInfo prop in type.UsableProperties(nonPublic:true)) prop.SetValue(obj, prop.GetValue(e
  }

 }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;

namespace Den.Tools
{
 public static class ScriptableAssetExtensions
 {
  public static T SaveAsset<T> (this T asset, string savePath=null, string filename="Data", string type="ass
  {
   #if UNITY_EDITOR
   if (savePath==null) savePath = UnityEditor.EditorUtility.SaveFilePanel(
    caption,
    "Assets",
    filename,
    type);
   if (savePath!=null && savePath.Length!=0)
   {
    savePath = savePath.Replace(Application.dataPath, "Assets");

    UnityEditor.AssetDatabase.CreateAsset(asset, savePath);
    if (asset is ISerializationCallbackReceiver) ((ISerializationCallbackReceiver)asset).OnBeforeSerialize();
    UnityEditor.AssetDatabase.SaveAssets();
```

```
  return asset;
 }
 #endif

 return null;
}

public static void SaveRawBytes (this byte[] bytes, string savePath=null, string filename="Data", string typ
{
 #if UNITY_EDITOR
 if (savePath==null) savePath = UnityEditor.EditorUtility.SaveFilePanel(
  "Save Data as Unity Asset",
  "Assets",
  filename,
  type);
 if (savePath!=null && savePath.Length!=0)
 {
  savePath = savePath.Replace(Application.dataPath, "Assets");
  System.IO.File.WriteAllBytes(savePath, bytes);
 }
 #endif
}

public static void SaveTexture (this Texture2D tex, string savePath=null, string filename="Texutre", string
{
```

```
 #if UNITY_EDITOR
 if (savePath==null) savePath = UnityEditor.EditorUtility.SaveFilePanel(
  caption, "Assets", filename, type);

 if (savePath!=null && savePath.Length!=0)
  tex.SaveAsPNG(savePath);
 #endif
}

public static T ReleaseAsset<T> (this T asset, string savePath=null) where T : ScriptableObject, ISerializa
{
 #if UNITY_EDITOR
 asset = ScriptableObject.Instantiate<T>(asset);
 #endif

 return asset;
}

public static T LoadAsset<T> (string label="Load Unity Asset", string[] filters=null) where T : UnityEngine.
{
 #if UNITY_EDITOR
 if (filters == null)
 {
  if (typeof(T).IsSubclassOf(typeof(Texture))) filters = new string[] { "Textures", "PSD,TIFF,TIF,JPG,TGA,P
  if (typeof(T) == typeof(Transform) || typeof(T) == typeof(Mesh)) filters = new string[] { "Meshes", "FBX,DA
  if (typeof(T) == typeof(TerrainData)) filters = new string[] { "TerrainData", "ASSET" };
```

```
 }
 ArrayTools.Add(ref filters, "All files");
 ArrayTools.Add(ref filters, "*");

 string path= UnityEditor.EditorUtility.OpenFilePanelWithFilters(label, "Assets", filters);
 if (path!=null && path.Length!=0)
 {
  path = path.Replace(Application.dataPath, "Assets");
  T asset = (T)UnityEditor.AssetDatabase.LoadAssetAtPath(path, typeof(T));
  return asset;
 }
 #endif
 return null;
}


//object selector wrapper
static Type objectSelectorType;

public static object GetObjectSelector ()
{
 if (objectSelectorType == null) objectSelectorType = Type.GetType("UnityEditor.ObjectSelector,UnityEdit

 PropertyInfo getProperty = objectSelectorType.GetProperty("get", BindingFlags.Public | BindingFlags.Sta
 object objSelector = getProperty.GetValue(null,null);
```

```
 return objSelector;
}

public static int GetObjectSelectorId ()
{
 if (objectSelectorType == null) objectSelectorType = Type.GetType("UnityEditor.ObjectSelector,UnityEdit
 object objectSelector = GetObjectSelector();

 FieldInfo idField = objectSelectorType.GetField("objectSelectorID", BindingFlags.Instance | BindingFlags

 return (int)idField.GetValue(objectSelector);
}

public static object GetObjectSelectorObject ()
{
 if (objectSelectorType == null) objectSelectorType = Type.GetType("UnityEditor.ObjectSelector,UnityEdit
 object objectSelector = GetObjectSelector();

 MethodInfo objectMethod =  objectSelectorType.GetMethod("GetCurrentObject", BindingFlags.Static | Bi
 Debug.Log(objectMethod.Invoke(null,null));
 return objectMethod.Invoke(null,null);

 //Debug.Log(objectMethod.Invoke(objectSelector, new object[0]));

 //return objectMethod.Invoke(objectSelector, new object[0]);
}
```

```
public static void ShowObjectSelector (Type objType, int id=12345, bool allowSceneObjects=false, Action
{
 #if UNITY_EDITOR

 if (objectSelectorType == null) objectSelectorType = Type.GetType("UnityEditor.ObjectSelector,UnityEdit
 object objectSelector = GetObjectSelector();

 MethodInfo showMethod = objectSelectorType.GetMethod(
  "Show",
  BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.DeclaredOnly,
  null,
  CallingConventions.Any,
  new Type[] { typeof(Type), typeof(UnityEditor.SerializedProperty), typeof(bool), typeof(List<int>), typeof(
  null);
 showMethod.Invoke(objectSelector, new object[]{objType, null, allowSceneObjects, null, onClosed, onUp

 FieldInfo idField = objectSelectorType.GetField("objectSelectorID", BindingFlags.Instance | BindingFlags
 idField.SetValue(objectSelector,id);

 #endif
}



public static class ObjectSelectorWrapper
```

```
{
 #if UNITY_EDITOR

 private static System.Type T;
 private static bool oldState = false;
 static ObjectSelectorWrapper()
 {
  T = System.Type.GetType("UnityEditor.ObjectSelector,UnityEditor");
 }

/* private static UnityEditor.EditorWindow Get()
 {
  PropertyInfo P = T.GetProperty("get", BindingFlags.Public | BindingFlags.Static);
  return P.GetValue(null,null) as UnityEditor.EditorWindow;
 }

 public static void ShowSelector(System.Type aRequiredType)
 {
  MethodInfo ShowMethod = T.GetMethod("Show",BindingFlags.Public | BindingFlags.Instance | BindingF
  ShowMethod.Invoke(Get (), new object[]{null,aRequiredType,null, true});
 }
 public static T GetSelectedObject<T>() where T : UnityEngine.Object
 {
  MethodInfo GetCurrentObjectMethod =  T.GetMethod("GetCurrentObject",BindingFlags.Static | BindingF
  return GetCurrentObjectMethod.Invoke(null,null) as T;
 }*/
```

```
 public static bool isVisible
 {
  get
  {
   PropertyInfo P = T.GetProperty("isVisible", BindingFlags.Public | BindingFlags.Static);
   return (bool)P.GetValue(null,null);
  }
 }
 public static bool HasJustBeenClosed()
 {
  bool visible = isVisible;
  if (visible != oldState && visible == false)
  {
   oldState = false;
   return true;
  }
  oldState = visible;
  return false;
 }

 #endif
}
```

```
 }

}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection; //to copy properties


namespace Den.Tools
{
 public enum TerrainControlType { Height, Splats, Grass }

 static public class TerrainExtensions
 {
  public static int GetResolution (this Terrain terrain, TerrainControlType controlType)
  {
   TerrainData terrainData = terrain.terrainData;

   switch (controlType)
   {
    case TerrainControlType.Height: return terrainData.heightmapResolution;
    case TerrainControlType.Splats: return terrainData.alphamapResolution;
    case TerrainControlType.Grass: return terrainData.detailResolution;
    default: return 0;
   }
  }
```

```
public static Vector2D PixelSize (this Terrain terrain, TerrainControlType controlType)
{
 Vector2D worldSize = (Vector2D)terrain.terrainData.size;
 int resolution = GetResolution(terrain, controlType);
 return worldSize / (resolution-1); //since terrain size looses half pixel from both sides
}


public static Vector2D PixelSize (this Terrain terrain, int resolution)
{
 Vector2D worldSize = (Vector2D)terrain.terrainData.size;
 return worldSize / (resolution-1); //since terrain size looses half pixel from both sides
}


public static CoordRect PixelRect (this Terrain terrain, Vector2D worldPos, Vector2D worldSize, TerrainC
/// Convering offset/size rect to pixel rect
{
 Vector2D pixelSize = PixelSize(terrain, controlType);
 return CoordRect.WorldToPixel(worldPos, worldSize, pixelSize);
}


public static CoordRect PixelRect (this Terrain terrain, Vector2D worldPos, Vector2D worldSize, int resolu
{
```

```
 Vector2D pixelSize = PixelSize(terrain, resolution);
 return CoordRect.WorldToPixel(worldPos, worldSize, pixelSize);
}


public static CoordRect PixelRect (this Terrain terrain, TerrainControlType controlType)
/// Getting whole terrain pixel rect
{
 int resolution = GetResolution(terrain, controlType);
 Vector2D pixelSize = PixelSize(terrain, controlType);

 return new CoordRect(
  Mathf.RoundToInt(terrain.transform.position.x/pixelSize.x),
  Mathf.RoundToInt(terrain.transform.position.z/pixelSize.z),
  resolution,
  resolution);
}


public static CoordRect PixelRect (this Terrain terrain, int resolution)
/// Getting whole terrain pixel rect
{
 Vector2D pixelSize = PixelSize(terrain, resolution);

 return new CoordRect(
  Mathf.RoundToInt(terrain.transform.position.x/pixelSize.x),
```

```
  Mathf.RoundToInt(terrain.transform.position.z/pixelSize.z),
  resolution,
  resolution);
}



/*public static CoordRect HeightAlignedPixelRect (this Terrain terrain, Vector2D worldPos, Vector2D worl
/// Returns non-heightmap rect. It's size perfectly corresponds with height resolution (if it's twice smaller -
{
 int splatsResolution = GetResolution(terrain, controlType);
 if (splatsResolution == 0) //no output of this type is assigned
  splatsResolution = terrain.terrainData.heightmapResolution;

 int heightResolution = terrain.terrainData.heightmapResolution;
 float ratio = HeightAlignedRatio(heightResolution, splatsResolution);

 CoordRect heightRect = PixelRect(terrain, worldPos, worldSize, TerrainControlType.Height);
 return heightRect * ratio;
}*/



public static float HeightAlignedRatio (int heightResolution, int splatsResolution)
/// Returns proper ratio for HeightAlignedPixelRect
/// Tested (MiscTests)
{
 float ratio = 1f * splatsResolution / heightResolution;
```

```
 //trying to find perfect ratio (0.25, 0.5, 1, 2, etc)
 if (splatsResolution > heightResolution)
 {
  int pRatio = (int)(ratio+0.5f);
  int restoredRes = heightResolution*pRatio;
  if (restoredRes-splatsResolution >= -pRatio  ||  restoredRes-splatsResolution <= pRatio)
   ratio = pRatio;
 }
 else
 {
  int invpRatio = (int)(1f/ratio + 0.5f);
  int restoredRes = splatsResolution*invpRatio;
  if (restoredRes-splatsResolution >= -invpRatio  ||  restoredRes-splatsResolution <= invpRatio)
   ratio = 1f / invpRatio;
 }

 return ratio;
}



public static void FastResize (this Terrain terrain, int resolution, Vector3 size)
{
 //setting resolution and THEN terrain size is too laggy
 //so making this trick to resize terrain or change res
```

```
 if ((terrain.terrainData.size-size).sqrMagnitude > 0.01f || terrain.terrainData.heightmapResolution != resol
 {
  if (resolution <= 64) //brute force
  {
   terrain.terrainData.heightmapResolution = resolution;
   terrain.terrainData.size = new Vector3(size.x, size.y, size.z);
  }

  else //setting res 64, re-scaling to 1/64, and then changing res
  {
   terrain.terrainData.heightmapResolution = 65;
   terrain.Flush(); //otherwise unity crushes without an error
   int resFactor = (resolution-1) / 64;
   terrain.terrainData.size = new Vector3(size.x/resFactor, size.y, size.z/resFactor);
   terrain.terrainData.heightmapResolution = resolution;
  }
 }
}


public static bool Contains (this Terrain terrain, Vector3 pos)
{
 Vector3 tpos = terrain.transform.position;
 Vector3 tsize = terrain.terrainData.size;
 if (pos.x>tpos.x && pos.z>tpos.z && pos.x<tpos.x+tsize.x && pos.z<tpos.z+tsize.z) return true;
 else return false;
```

```
}


public static float SampleAverageHeight (this Terrain terrain, Vector3 pos, int pixelExtent)
{
 TerrainData terrainData = terrain.terrainData;
 int heightmapResolution = terrainData.heightmapResolution;
 Coord pixelPos = new Coord();
 pixelPos.x = (int)((pos.x-terrain.transform.position.x) / terrainData.size.x * heightmapResolution);
 pixelPos.z = (int)((pos.z-terrain.transform.position.z) / terrainData.size.z * heightmapResolution);
 CoordRect pixelRect = new CoordRect(pixelPos, pixelExtent);
 pixelRect = CoordRect.Intersected(pixelRect, new CoordRect(0,0,heightmapResolution,heightmapResolu

 float avg = 0;

 float[,] heights = terrainData.GetHeights(pixelRect.offset.x, pixelRect.offset.z, pixelRect.size.x, pixelRect.

 for (int x=0; x<pixelRect.size.x; x++)
  for (int z=0; z<pixelRect.size.z; z++)
   avg += heights[z,x];

 return avg / (pixelRect.size.x*pixelRect.size.z) * terrainData.size.y;
}


public static UnityEngine.Object Object (this DetailPrototype prot)
```

```
{
 if (prot.renderMode == DetailRenderMode.VertexLit)
  return prot.prototype;

 else
  return prot.prototypeTexture;
}

public static TerrainData Copy (this TerrainData src)
{
 TerrainData dst = new TerrainData();

 dst.heightmapResolution = src.heightmapResolution;
 dst.SetHeights(0,0, src.GetHeights(0,0,src.heightmapResolution,src.heightmapResolution));

 dst.terrainLayers = src.terrainLayers;
 dst.alphamapResolution = src.alphamapResolution;
 dst.SetAlphamaps(0,0, src.GetAlphamaps(0,0, src.alphamapResolution, src.alphamapResolution));

 dst.detailPrototypes = src.detailPrototypes;
 dst.SetDetailResolution(src.detailResolution, src.detailResolutionPerPatch);
 int numGrass = src.detailPrototypes.Length;
 for (int i=0; i<numGrass; i++)
  dst.SetDetailLayer(0,0,i, src.GetDetailLayer(0,0,src.detailResolution, src.detailResolution, i));

 dst.treePrototypes = src.treePrototypes;
```

```
 dst.treeInstances = src.treeInstances;

 dst.size = src.size; //after changing all resolutions

 return dst;
}


public static bool CheckSplatsSum (this TerrainData data, out string error)
// True if sum is 1 for each pixel
{
 int resolution = data.alphamapResolution;
 float[,,] splats = data.GetAlphamaps(0,0, resolution, resolution);
 int numChannels = splats.GetLength(2);

 for (int x=0; x<resolution; x++)
  for (int z=0; z<resolution; z++)
  {
   float sum = 0;

   for (int c=0; c<numChannels; c++)
    sum += splats[x,z,c];

   if (sum < 0.999f || sum > 1.01f)
   {
    error = $"Sum not equals to 1 at {x}, {z}, sum: {sum}";
```

```
      return false;
    }
   }


  error = null;
  return true;
 }
}
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Den.Tools.Matrices;

namespace Den.Tools
{
 public static class TextureExtensions
 {
   public static bool IsReadable (this Texture2D tex)
   {
    try
    {
     tex.GetPixel(0,0);
     return true;
    }
    catch
    {
     return false;
    }
   }

   public static bool IsLinear (this Texture tex)
   {
    #if UNITY_EDITOR
```

```
/*    //this works for saved texture, but not for texture2darray:
    //string path = UnityEditor.AssetDatabase.GetAssetPath(tex);
    //UnityEditor.AssetImporter importer = UnityEditor.TextureImporter.GetAtPath(path);
    //Debug.Log(importer.GetType());
    //UnityEditor.TextureImporter teximporter = (UnityEditor.TextureImporter)importer;
    //return teximporter.sRGBTexture;

    System.Reflection.Assembly a = typeof(UnityEditor.EditorWindow).Assembly;
    System.Type t = a.GetType("UnityEditor.TextureUtil");
    string s = (string)t.GetMethod("GetTextureColorSpaceString").Invoke(null, new object[] { tex });
    return s=="Linear";
    //you know the better way to get texture array color space?
    //let me know mail@denispahunov.ru
*/
    return false;

    #else
    return false;
    #endif
   }



   public static Texture2D ReadableClone (this Texture2D tex)
   {
    Texture2D readTex = new Texture2D(tex.width, tex.height, tex.format, true, linear:tex.IsLinear());
```

```
 Graphics.CopyTexture(tex,readTex);


 readTex.Apply(updateMipmaps:false);
 return readTex;
}


public static Texture2D UncompressedClone (this Texture2D tex)
/// Just using UnityEditor.EditorUtility.CompressTexture will not convert compressed to compressed (DXT
/// Texture should be readable
{
 int mipmapCount = tex.mipmapCount;
 Texture2D resultTex = new Texture2D(tex.width, tex.height, TextureFormat.ARGB32, mipmapCount!=1,
 for (int m=0; m<mipmapCount; m++)
 {
  Color[] colors = tex.GetPixels(m);
  resultTex.SetPixels(colors,m);
 }
 resultTex.Apply(updateMipmaps:false);
 return resultTex;
}


static public Texture2D ResizedClone (this Texture2D tex, int newWidth, int newHeight)
/// Texture should be be readable and not compressed
{
 Texture2D newTex = new Texture2D(newWidth, newHeight, TextureFormat.ARGB32, true, linear:tex.IsL
```

```
 newTex.name = tex.name;

 Color[] pixels = tex.GetPixels();
 pixels = pixels.ResizeColorArray(tex.width, tex.height, newWidth, newHeight);
 newTex.SetPixels(pixels);

 newTex.Apply(updateMipmaps:true);
 return newTex;
}



public static Color[] ResizeColorArray (this Color[] srcColors, int oldWidth, int oldHeight, int newWidth, int
/// Helper for GetPixelsResize
{
 Color[] dstColors = new Color[newWidth*newHeight];

 Matrix src = new Matrix( new CoordRect(0,0,oldWidth, oldHeight) );
 Matrix dst = new Matrix( new CoordRect(0,0, newWidth, newHeight) );

 //reds
 for (int i=0; i<srcColors.Length; i++)
  src.arr[i] = srcColors[i].r;

 MatrixOps.Resize(src, dst);

 for (int i=0; i<dstColors.Length; i++)
```

```
 dstColors[i].r = dst.arr[i];

//greens
for (int i=0; i<srcColors.Length; i++)
 src.arr[i] = srcColors[i].g;

MatrixOps.Resize(src, dst);

for (int i=0; i<dstColors.Length; i++)
 dstColors[i].g = dst.arr[i];

//blues
//when I was arrested I was dressed in black they put me on a train and they took me back
for (int i=0; i<srcColors.Length; i++)
 src.arr[i] = srcColors[i].b;

MatrixOps.Resize(src, dst);

for (int i=0; i<dstColors.Length; i++)
 dstColors[i].b = dst.arr[i];

//alphas
for (int i=0; i<srcColors.Length; i++)
 src.arr[i] = srcColors[i].a;

MatrixOps.Resize(src, dst);
```

```
 for (int i=0; i<dstColors.Length; i++)
  dstColors[i].a = dst.arr[i];

 return dstColors;
}

public static Texture2D ColorTexture (int width, int height, Color color, bool linear=false)
{
 Texture2D result = new Texture2D(width, height, TextureFormat.ARGB32, true, linear:linear);
 result.Colorize(color);
 return result;
}

public static void Colorize (this Texture2D tex, Color color)
{
 Color[] pixels = tex.GetPixels();
 for (int i=0;i<pixels.Length;i++) pixels[i] = color;
 tex.SetPixels(pixels);
 tex.Apply();
}

public static Texture2D Clone (this Texture2D src)
{
```

```
 Texture2D dst = new Texture2D(src.width, src.height, src.format, src.mipmapCount!=1);
 Graphics.CopyTexture(src, dst);
 dst.Apply(updateMipmaps:false);
 return dst;

}



public static void ClearAlpha (this Texture2D tex)
{
 Color[] colors = tex.GetPixels();
 for (int i=0; i<colors.Length; i++)
  colors[i].a = 1;  //clear alpha is white alpha
 tex.SetPixels(colors);
 tex.Apply();
}

public static void ApplyGamma (this Texture2D tex, float gamma=2.2f)
{
 //IDEA: use raw pixel data
 float invVal = 1 / gamma;
 Color[] colors = tex.GetPixels();
 for (int i=0; i<colors.Length; i++)
  colors[i] = new Color(
   Mathf.Pow(colors[i].r, invVal),
   Mathf.Pow(colors[i].g, invVal),
```

```
   Mathf.Pow(colors[i].b, invVal),
   colors[i].a);
 tex.SetPixels(colors);
 tex.Apply();
}



public static void RestoreNormalmap (this Texture2D tex)
{
 Color[] colors = tex.GetPixels();
 for (int i=0; i<colors.Length; i++)
 {
  Vector2 normXY = new Vector2(colors[i].g*2 - 1, colors[i].a*2 - 1);
  float normZ = Mathf.Sqrt(1 - Mathf.Clamp01( Vector3.Dot(normXY, normXY)));
  colors[i] = new Color(
   colors[i].g,
   colors[i].a,
   normZ/2 + 0.5f,
   1);
 }
 tex.SetPixels(colors);
 tex.Apply();
}



public static void Multiply (this Texture2D tex, Color color, bool multiplyAlpha=false)
```

```
{
 Color[] colors = tex.GetPixels();
 for (int i=0; i<colors.Length; i++)
 {
  colors[i].r = colors[i].r * color.r; //(r*cr)*ca + r*(1-ca),
  colors[i].g = colors[i].g * color.g;
  colors[i].b = colors[i].b * color.b;
  if (multiplyAlpha)
   colors[i].a = colors[i].a * color.a;
 }
 tex.SetPixels(colors);
 tex.Apply();
}



public static void SaveAsPNG (this Texture2D origTex, string savePath, bool linear=false, bool normal=fa
{
 Texture2D tex = origTex;

 if (!tex.IsReadable()) tex = tex.ReadableClone();
 if (tex.format.IsCompressed()) tex = tex.UncompressedClone();

 if (linear)
 {
  if (tex==origTex) tex=tex.Clone();
  tex.ApplyGamma(0.4545454545454545f);
```

```
  tex.Apply(updateMipmaps:false);
 }

 if (normal)
 {
  if (tex==origTex) tex=tex.Clone();
  tex.RestoreNormalmap();
  tex.Apply(updateMipmaps:false);
 }

 savePath = savePath.Replace(Application.dataPath, "Assets");
 System.IO.File.WriteAllBytes(savePath, tex.EncodeToPNG());

 #if UNITY_EDITOR
 UnityEditor.AssetDatabase.Refresh();
 #endif
}

public static Hash128 GetHash (this Texture2D tex)
{
 #if UNITY_EDITOR

  #if UNITY_2017_3_OR_NEWER
  return tex.imageContentsHash;

  #else
```

```
UnityEditor.SerializedProperty property = new UnityEditor.SerializedObject (tex).FindProperty ("m_Imag

//by Broxxar
//https://answers.unity.com/questions/1249181/how-to-get-texture-image-contents-hash-property.html

if (property.type != "Hash128") {
 throw new Exception("SerializedProperty does not represent a Hash128 struct.");
}

var bytes = new byte[4][];

for (var i = 0; i < 4; i++) {
 bytes[i] = new byte[4];

 for (var j = 0; j < 4; j++) {
  property.Next(true);
  bytes[i][j] = (byte)property.intValue;
 }
}

var hash = new Hash128(
 BitConverter.ToUInt32(bytes[0], 0),
 BitConverter.ToUInt32(bytes[1], 0),
 BitConverter.ToUInt32(bytes[2], 0),
 BitConverter.ToUInt32(bytes[3], 0));
```

```
  return hash;
  #endif

 #else
 return new Hash128();
 #endif
}


#region Formats and Compression

 public static readonly HashSet<TextureFormat> uncompressedFormats = new HashSet<TextureFormat>
  new TextureFormat[] {
   TextureFormat.Alpha8, TextureFormat.ARGB32, TextureFormat.ARGB4444, TextureFormat.R16, Textu
   TextureFormat.RG16, TextureFormat.RGB24, TextureFormat.RGB565, TextureFormat.RGB9e5Float, T
   TextureFormat.RGBAFloat, TextureFormat.RGBAHalf, TextureFormat.RGFloat, TextureFormat.RHalf }

 public static bool IsCompressed (this TextureFormat format)
 {
  if (uncompressedFormats.Contains(format)) return false;
  else return true;
 }

 public enum TextureType { RGBA, RGB, Normal, Monochrome, MonochromeFloat, Manual };
```

```
public static TextureFormat AutoFormat (TextureType type, bool compressed)
{
 if (compressed)
 {
  //list of used texture formats: https://docs.unity3d.com/Manual/class-TextureImporterOverride.html
  //using high quality settings
  #if UNITY_EDITOR
  UnityEditor.BuildTarget buildTarget = UnityEditor.EditorUserBuildSettings.activeBuildTarget;
  switch (buildTarget)
  {
   case UnityEditor.BuildTarget.StandaloneWindows64:
   case UnityEditor.BuildTarget.StandaloneWindows:
   case UnityEditor.BuildTarget.WSAPlayer:
    UnityEngine.Rendering.GraphicsDeviceType[] deviceTypes = UnityEditor.PlayerSettings.GetGraphics
    for (int i=0; i<deviceTypes.Length; i++)
     if (deviceTypes[i] == UnityEngine.Rendering.GraphicsDeviceType.Direct3D11 ||
      deviceTypes[i] == UnityEngine.Rendering.GraphicsDeviceType.Direct3D12)
      {
       if (type==TextureType.Normal) return TextureFormat.BC5;
       else if (type==TextureType.Monochrome) return TextureFormat.BC4;
       else if (type==TextureType.MonochromeFloat) return TextureFormat.BC6H;
       else return TextureFormat.BC7;
      }
    return TextureFormat.DXT5;
   #if UNITY_2017_3_OR_NEWER
   case UnityEditor.BuildTarget.StandaloneOSX:
```

```
 #endif
 //case UnityEditor.BuildTarget.StandaloneLinuxUniversal:
 case UnityEditor.BuildTarget.PS4:
 case UnityEditor.BuildTarget.XboxOne:
  if (type==TextureType.Normal) return TextureFormat.BC5;
  else if (type==TextureType.Monochrome) return TextureFormat.BC4;
  else if (type==TextureType.MonochromeFloat) return TextureFormat.BC6H;
  else return TextureFormat.BC7;
 case UnityEditor.BuildTarget.Android:
  if (type==TextureType.RGBA) return TextureFormat.ETC2_RGBA8;
  else return TextureFormat.ETC2_RGB;
 case UnityEditor.BuildTarget.iOS: return TextureFormat.PVRTC_RGBA4;
 case UnityEditor.BuildTarget.tvOS: return TextureFormat.ASTC_4x4;
 default: return TextureFormat.DXT5;
 }
 #else
 return TextureFormat.DXT5;
 #endif
}
else
{
 switch (type)
 {
  case TextureType.RGB: return TextureFormat.RGB24;
  case TextureType.Normal: return TextureFormat.RG16;
  case TextureType.Monochrome: return TextureFormat.R8;
```

```
      case TextureType.MonochromeFloat: return TextureFormat.RFloat;
      default: return TextureFormat.RGBA32;
     }
    }
   }

  #endregion
 }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace Den.Tools
{
 public static class TextureArrayTools
 {

  public static void SetTexture (this Texture2DArray dstArr, Texture2D src, int dstCh, bool apply=true)
  {
   //Debug.Log("Setting Texture " + src.name + " " + System.IO.Path.GetFileName(UnityEditor.AssetDataba

   if (dstArr.depth <= dstCh) throw new System.IndexOutOfRangeException("Trying to set channel (" + dstC

   //quick case if size and format match
   if (src.width == dstArr.width  &&  src.height == dstArr.height  &&  src.format == dstArr.format)
   {
    Graphics.CopyTexture(src,0,  dstArr,dstCh);
    if (apply) dstArr.Apply(updateMipmaps:false);
    return;
   }

   if (!src.IsReadable()) src = src.ReadableClone();  //texture should be readable to uncompress
   if (src.format.IsCompressed()) src = src.UncompressedClone();
```

```
 if (src.width != dstArr.width  ||  src.height != dstArr.height)
  src = src.ResizedClone(dstArr.width, dstArr.height);

 #if UNITY_EDITOR

 #if UNITY_2018_3_OR_NEWER
 UnityEditor.EditorUtility.CompressTexture(src, dstArr.format, 100);  //de-compress and compress to chan
 #else
 UnityEditor.EditorUtility.CompressTexture(src, dstArr.format, TextureCompressionQuality.Best);  //de-cor
 #endif

 #else
 if (dstArr.format.IsCompressed()) src.Compress(true);
 #endif
 src.Apply(updateMipmaps:false);

 Graphics.CopyTexture(src,0,  dstArr,dstCh);
 if (apply) dstArr.Apply(updateMipmaps:false);
}

public static void SetTextureAlpha (this Texture2DArray dstArr, Texture2D src, Texture2D alpha, int dstCh
/// Sets RGB to src RGB, and A to desaturated alpha RGB
{
 if (alpha == null) { dstArr.SetTexture(src, dstCh, apply:apply); return; } //shortcut if alpha is null

 if (!src.IsReadable()) src = src.ReadableClone();
```

```
 if (src.format.IsCompressed()) src = src.UncompressedClone();
 if (src.width != dstArr.width  ||  src.height != dstArr.height) src = src.ResizedClone(dstArr.width, dstArr.hei

 if (!alpha.IsReadable()) alpha = alpha.ReadableClone();
 //if (!alpha.format.IsCompressed()) alpha = alpha.UncompressedClone();  //no need to change alpha form
 if (alpha.width != dstArr.width  ||  alpha.height != dstArr.height) alpha = alpha.ResizedClone(dstArr.width,



 Texture2D tmp = new Texture2D(src.width, src.height, TextureFormat.RGBA32, true, src.IsLinear());
 int mipmapCount = src.mipmapCount;
 for (int m=0; m<mipmapCount; m++)
 {
  Color[] srcColors = src.GetPixels(m);
  Color[] alphaColors = alpha.GetPixels(m);

  for (int i=0; i<srcColors.Length; i++)
   srcColors[i] = new Color(srcColors[i].r, srcColors[i].g, srcColors[i].b, alphaColors[i].r*0.3f + alphaColors[i
  tmp.SetPixels(srcColors,m);
 }

 tmp.Apply(updateMipmaps:false);
 dstArr.SetTexture(tmp, dstCh, apply:apply);
}

public static Texture2D GetTexture (this Texture2DArray srcArr, int srcCh, bool readable=true)
{
```

```
 Texture2D tex = new Texture2D(srcArr.width, srcArr.height, srcArr.format, true, linear:srcArr.IsLinear());
 Graphics.CopyTexture(srcArr,srcCh,  tex,0);
 tex.Apply(updateMipmaps:false, makeNoLongerReadable:!readable);
 return tex;
}



public static Texture2D[] GetTextures (this Texture2DArray srcArr)
{
 Texture2D[] result = new Texture2D[srcArr.depth];

 for (int ch=0; ch<srcArr.depth; ch++)
  result[ch] = GetTexture(srcArr, ch);

 return result;
}



public static Color GetPixel (this Texture2DArray srcArr, int x, int y, int ch)
/// Debug purpose only
{
 Texture2D tmp = srcArr.GetTexture(ch);
 return tmp.GetPixel(x,y);
}

public static void FillTexture (this Texture2DArray srcArr, Texture2D dst, int srcCh)
```

```
{
 if (srcArr.depth <= srcCh) throw new System.IndexOutOfRangeException("Trying to get channel (" + srcC

 //format and size match
 if (srcArr.format == dst.format  &&  srcArr.width == dst.width  &&  srcArr.height == dst.height)
 {
  Graphics.CopyTexture(srcArr,srcCh,  dst,0);
  dst.Apply(updateMipmaps:false);
  return;
 }

 Texture2D tmpTex = srcArr.GetTexture(srcCh, readable:true);
 if (tmpTex.format.IsCompressed()) tmpTex = tmpTex.UncompressedClone();  //uncompress to change fo

 if (tmpTex.width != dst.width  ||  tmpTex.height != dst.height)
  tmpTex = tmpTex.ResizedClone(dst.width, dst.height);

 #if UNITY_EDITOR

 #if UNITY_2018_3_OR_NEWER
 UnityEditor.EditorUtility.CompressTexture(tmpTex, dst.format, 100);  //de-compress and compress to cha
 #else
 UnityEditor.EditorUtility.CompressTexture(tmpTex, dst.format, TextureCompressionQuality.Best);  //de-co
 #endif
```

```
 #else
 if (dst.format.IsCompressed()) tmpTex.Compress(true);
 #endif
 tmpTex.Apply(updateMipmaps:false);

 Graphics.CopyTexture(tmpTex, dst);
 dst.Apply(updateMipmaps:false);
}



public static void CopyTexture (Texture2DArray srcArr, int srcCh,  Texture2DArray dstArr, int dstCh) { Cop

public static void CopyTextures (Texture2DArray srcArr, Texture2DArray dstArr, int length) { CopyTexture
public static void CopyTextures (Texture2DArray srcArr, int srcIndex, Texture2DArray dstArr, int dstIndex,
/// Bulk textures copy from index to index
{
 //format and size match
 if (srcArr.format == dstArr.format  &&  srcArr.width == dstArr.width  &&  srcArr.height == dstArr.height)
 {
  for (int i=0; i<length; i++)
   Graphics.CopyTexture(srcArr,srcIndex+i,  dstArr,dstIndex+i);
  dstArr.Apply(updateMipmaps:false);
  return;
 }

 Texture2D tmpTex = new Texture2D(dstArr.width, dstArr.height, dstArr.format, true, linear:srcArr.IsLinea
```

```
 for (int i=0; i<length; i++)
 {
  srcArr.FillTexture(tmpTex, srcIndex+i);
  Graphics.CopyTexture(tmpTex,0, dstArr,dstIndex+i);
 }

 dstArr.Apply(updateMipmaps:false);
}



public static void Add (ref Texture2DArray texArr, Texture2D tex) { var newArr = Add(texArr, tex); Rewrite
public static Texture2DArray Add (Texture2DArray texArr, Texture2D tex)
{
 Texture2DArray newArr = new Texture2DArray(texArr.width, texArr.height, texArr.depth+1, texArr.format
 newArr.name = texArr.name;

 CopyTextures(texArr, newArr, texArr.depth);

 newArr.SetTexture(tex, texArr.depth, apply:false);

 newArr.Apply(updateMipmaps:false);
 return newArr;
}



static public void Insert (ref Texture2DArray texArr, int pos, Texture2D tex) { var newArr = Insert(texArr, p
```

```
static public Texture2DArray Insert (Texture2DArray texArr, int pos, Texture2D tex)
{
 bool linear = texArr.IsLinear();
 if (texArr==null || texArr.depth==0)
 {
  texArr = new Texture2DArray(tex.width, tex.height, 1, texArr.format, true, linear:linear);
  texArr.filterMode = FilterMode.Trilinear;
  texArr.SetTexture(tex, 0, apply:false);
  return texArr;
 }

 if (pos > texArr.depth || pos < 0) pos = texArr.depth;

 Texture2DArray newArr = new Texture2DArray(texArr.width, texArr.height, texArr.depth+1, texArr.format
 newArr.name = texArr.name;

 if (pos != 0) CopyTextures(texArr, newArr, pos);
 if (pos != texArr.depth) CopyTextures(texArr, pos, newArr, pos+1, texArr.depth-pos);

 if (tex!=null) newArr.SetTexture(tex, pos, apply:false);

 newArr.Apply(updateMipmaps:false);
 return newArr;
}
```

```
static public Texture2DArray InsertRange (Texture2DArray texArr, int pos, Texture2DArray addArr)
{
 //if (texArr==null || texArr.depth==0) { return addArr; }
 if (pos > texArr.depth || pos < 0) pos = texArr.depth;

 Texture2DArray newArr = new Texture2DArray(texArr.width, texArr.height, texArr.depth+addArr.depth, te
 newArr.name = texArr.name;

 if (pos != 0) CopyTextures(texArr, newArr, pos);
 CopyTextures(addArr, 0, newArr, pos, addArr.depth);
 if (pos != texArr.depth) CopyTextures(texArr, pos, newArr, pos+addArr.depth, texArr.depth-pos);

 newArr.Apply(updateMipmaps:false);
 return newArr;
}

static public void Switch (ref Texture2DArray texArr, int num1, int num2) { Switch(texArr, num1, num2); Re
static public void Switch (this Texture2DArray texArr, int num1, int num2)
{
 if (num1<0 || num1>=texArr.depth || num2<0 || num2 >=texArr.depth) return;

 Texture2D temp = texArr.GetTexture(num1);
 CopyTexture(texArr,num2,  texArr,num1);
 texArr.SetTexture(temp,num2);
}
```

```
static public void Clear (this Texture2DArray texArr, int chNum)
/// Fills channel with blank without removing it
{
 Texture2D temp = new Texture2D(texArr.width, texArr.height, texArr.format, true, linear:texArr.IsLinear())
 texArr.SetTexture(temp,chNum);
}



static public void RemoveAt (ref Texture2DArray texArr, int num) { var newArr = RemoveAt(texArr, num);
static public Texture2DArray RemoveAt (Texture2DArray texArr, int num)
{
 if (num >= texArr.depth || num < 0) return texArr;

 Texture2DArray newArr = new Texture2DArray(texArr.width, texArr.height, texArr.depth-1, texArr.format,
 newArr.name = texArr.name;

 if (num!=0) CopyTextures(texArr, newArr, num);
 if (num!=texArr.depth) CopyTextures(texArr, num+1, newArr, num, newArr.depth-num);

 newArr.Apply(updateMipmaps:false);
 return newArr;
}



static public void ChangeCount (ref Texture2DArray texArr, int newSize) { var newArr = ChangeCount(tex
```

```
static public Texture2DArray ChangeCount (Texture2DArray texArr, int newSize)
{
 //if (texArr.depth == newSize) return texArr;

 Texture2DArray newArr = new Texture2DArray(texArr.width, texArr.height, newSize, texArr.format, true,
 newArr.name = texArr.name;

 int min = newSize<texArr.depth? newSize : texArr.depth;
 CopyTextures(texArr, newArr, min);

 newArr.Apply(updateMipmaps:false);
 return newArr;
}


static public Texture2DArray ResizedClone (this Texture2DArray texArr, int newWidth, int newHeight)
{
 Texture2DArray newArr = new Texture2DArray(newWidth, newHeight, texArr.depth, texArr.format, true, l
 newArr.name = texArr.name;

 for (int i=0; i<texArr.depth; i++)
  CopyTexture(texArr,i, newArr,i);

 newArr.Apply(updateMipmaps:false);
 return newArr;
}
```

```
static public Texture2DArray FormattedClone (this Texture2DArray texArr, TextureFormat format)
{
 Texture2DArray newArr = new Texture2DArray(texArr.width, texArr.height, texArr.depth, format, true, line
 newArr.name = texArr.name;

 int depth = texArr.depth;
 for (int i=0; i<depth; i++)
  CopyTexture(texArr,i, newArr,i);

 newArr.Apply(updateMipmaps:false);
 return newArr;
}

static public Texture2DArray LinearClone (this Texture2DArray texArr, bool linear)
{
 Texture2DArray newArr = new Texture2DArray(texArr.width, texArr.height, texArr.depth, texArr.format, tr
 newArr.name = texArr.name;

 int depth = texArr.depth;
 for (int i=0; i<depth; i++)
  CopyTexture(texArr,i, newArr,i);

 newArr.Apply(updateMipmaps:false);
 return newArr;
```

```
}

static public Texture2DArray WritableClone (this Texture2DArray texArr)
{
 Texture2DArray newArr = new Texture2DArray(texArr.width, texArr.height, texArr.depth, texArr.format, tr
 for (int i=0; i<texArr.depth; i++)
 {
  CopyTexture(texArr,i, newArr,i);
 }

 newArr.Apply(updateMipmaps:true);
 return newArr;
}


public static int GetMipMapCount (this Texture2DArray texArr)
{
 for (int i=0; i<100; i++)
 {
  try { texArr.GetPixels(0,i); }
  catch { return i; }
 }
 return -1;
}

public static bool IsReadWrite (this Texture2DArray texArr)
```

```
{
 try { texArr.SetPixels(null,0); }
 catch { return false; }
 return true;
}



public static void Rewrite (ref Texture2DArray texArr, Texture2DArray newArr)
{
 #if UNITY_EDITOR
 bool isSelected = UnityEditor.Selection.activeGameObject == texArr;

 UnityEditor.AssetImporter texImporter = texArr.GetImporter();
 if (texImporter == null) return;

 //string userData = texImporter.userData; //do not owerwrite userdata
 UnityEditor.EditorUtility.CopySerialized(newArr, texArr);
 UnityEditor.EditorUtility.SetDirty(texArr);
 UnityEditor.AssetDatabase.SaveAssets();

 if (isSelected) UnityEditor.Selection.activeObject = texArr;



 //texArr = UnityEditor.AssetDatabase.LoadAssetAtPath<Texture2DArray>(path);
 //texArr.GetImporter().userData = userData;
 #endif
```

```
}



#region GUID operations


/*public static Texture2D GetSource (this Texture2DArray texArr, int num, bool isAlpha=false)
{
 #if UNITY_EDITOR
 if (!UnityEditor.AssetDatabase.Contains(texArr))
  { Debug.Log("Texture Array is not an asset: could not get source tex"); return null; }

 string[] sourceGuids = texArr.GetUserData(isAlpha? "TexArr_alphaLayers" : "TexArr_sourceLayers");

 if (num < sourceGuids.Length)
  return sourceGuids[num].GUIDtoObj<Texture2D>();
 else
  return null;
 #else
 return null;
 #endif
}

public static void SetSource (this Texture2DArray texArr, Texture2D src, int num, bool isAlpha=false, bool
{
 #if UNITY_EDITOR
```

```
//checking if user data could be saved
if (!UnityEditor.AssetDatabase.Contains(texArr))
 { Debug.Log("Texture Array is not an asset: could not set source tex"); return; }
if (src != null  &&  !UnityEditor.AssetDatabase.Contains(src))
 { Debug.Log("Texture is not an asset: could not set source"); return; }



string arrGuid = texArr.GUID();
string[] sourceGuids = texArr.GetUserData(isAlpha? "TexArr_alphaLayers" : "TexArr_sourceLayers");



//removing link to arr in previous tex
Texture2D prevTex = texArr.GetSource(num, isAlpha);
if (prevTex != null)
{
 string prevTexGuid = sourceGuids[num];
 int usedCount = sourceGuids.FindCount(prevTexGuid);
 if (usedCount == 1) //if used once (>0) and only once (<2)
 {
  string[] prevTexData = prevTex.GetUserData(isAlpha? "TexArr_textureArray_asAlpha": "TexArr_texture
  ArrayTools.RemoveAll(ref prevTexData, arrGuid);
  prevTex.SetUserData(isAlpha? "TexArr_textureArray_asAlpha": "TexArr_textureArray_asSource", prev
 }
}
```

```
 //assigning link to arr in new tex
 if (src != null)
 {
  string[] srcData = src.GetUserData(isAlpha? "TexArr_textureArray_asAlpha": "TexArr_textureArray_asS
  if (!srcData.Contains(arrGuid))
   ArrayTools.Add(ref srcData, arrGuid);
  src.SetUserData(isAlpha? "TexArr_textureArray_asAlpha": "TexArr_textureArray_asSource", srcData, re
 }

 sourceGuids[num] = src.GUID();
 texArr.SetUserData(isAlpha? "TexArr_alphaLayers" : "TexArr_sourceLayers", sourceGuids);

 if (reload) UnityEditor.AssetDatabase.Refresh();
 #endif
}



public static void ClearSources (this Texture2DArray texArr, bool isAlpha=false, bool reload = true)
/// Clear user data and unlink source textures
{
 #if UNITY_EDITOR

 if (!UnityEditor.AssetDatabase.Contains(texArr))
  { Debug.Log("Texture Array is not an asset: could not set sources count"); return; }
```

```
 //unlinking previous sources
 string[] sourceGuids = texArr.GetUserData(isAlpha? "TexArr_alphaLayers" : "TexArr_sourceLayers");
 for (int i=0; i<sourceGuids.Length; i++) texArr.SetSource(null, i, isAlpha, reload:false);

 texArr.SetUserData(isAlpha? "TexArr_alphaLayers" : "TexArr_sourceLayers", new string[0]);
 if (reload) UnityEditor.AssetDatabase.Refresh();

 #endif
}

public static void ResizeSources (this Texture2DArray texArr, int count, bool isAlpha=false, bool reload=tr
/// This will erase all sources
{
 #if UNITY_EDITOR

 if (!UnityEditor.AssetDatabase.Contains(texArr))
  { Debug.Log("Texture Array is not an asset: could not set sources count"); return; }

 texArr.ClearSources(isAlpha, reload:false);

 //creating new
 string[] sourceGuids = new string[count];
 for (int i=0; i<sourceGuids.Length; i++) sourceGuids[i] = "";

 texArr.SetUserData(isAlpha? "TexArr_alphaLayers" : "TexArr_sourceLayers", sourceGuids);
```

```
    if (reload) UnityEditor.AssetDatabase.Refresh();


    #endif
   }*/


   #endregion


  }


}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections.Generic;
//using UnityEngine.Profiling;

namespace Den.Tools.GUI
{
 public sealed class ValAttribute : Attribute
 {
  public string name;
  public string cat;
  public bool display = true; //to expose but do not show automatically
  public bool isLeft; //for toggles
  public int priority = 0;
  public Type type;
  public bool allowSceneObject;
  public float min = float.MinValue;
  public float max = float.MaxValue;

  public FieldInfo field;
  public PropertyInfo prop;
  public MethodInfo method;

  public ValAttribute () { }
  public ValAttribute (string name) { this.name=name; }
  public ValAttribute (string name, string cat) { this.name=name; this.cat=cat; }
```

```
public ValAttribute (string name, float min, float max) { this.name=name; this.min=min; this.max=max; }
public ValAttribute (string name, float min) { this.name=name; this.min=min; }
public ValAttribute (string name, string cat, float min, float max) { this.name=name; this.cat=cat; this.min=

[System.NonSerialized] private static readonly Dictionary<Type, ValAttribute[]> attributesCaches = new D

public static ValAttribute[] GetAttributes (Type type)
{
 if (attributesCaches.TryGetValue(type, out ValAttribute[] attributes)) return attributes;

 List<ValAttribute> attList = new List<ValAttribute>();

 FieldInfo[] fields = type.GetFields();
 for (int f=0; f<fields.Length; f++)
 {
  ValAttribute valAtt = Attribute.GetCustomAttribute(fields[f], typeof(ValAttribute)) as ValAttribute;
  if (valAtt == null) continue;

  valAtt.field = fields[f];
  valAtt.type = fields[f].FieldType;

  attList.Add(valAtt);
 }

 PropertyInfo[] props = type.GetProperties();
```

```
for (int p=0; p<props.Length; p++)
{
 ValAttribute valAtt = Attribute.GetCustomAttribute(props[p], typeof(ValAttribute)) as ValAttribute;
 if (valAtt == null) continue;

 valAtt.prop = props[p];
 valAtt.type = props[p].PropertyType;

 attList.Add(valAtt);
}

MethodInfo[] methods = type.GetMethods();
for (int m=0; m<methods.Length; m++)
{
 ValAttribute valAtt = Attribute.GetCustomAttribute(methods[m], typeof(ValAttribute)) as ValAttribute;
 if (valAtt == null) continue;

 valAtt.method = methods[m];

 attList.Add(valAtt);
}

attributes = attList.ToArray();

//if (attributes != null)
// Array.Sort(attributes, (x,y) => 0);//y.priority.CompareTo(x.priority));
```

```
   attributesCaches.Add(type, attributes);
   return attributes;
  }



 }

 public sealed class EditorClassAttribute : Attribute
 {
  public static Dictionary<Type,Type> editors = new Dictionary<Type, Type>();

  public EditorClassAttribute (Type type)
  {
   UnityEngine.Debug.Log(type);
  }
 }
}

/*
namespace Plugins.GUI
{
 public class Val : Attribute
 {
  public string name;
  public string cat;
```

```
public Type type;
public bool isLeft; //for toggles
public int priority;
public bool allowSceneObject;

public FieldInfo field;
public PropertyInfo prop;
public MethodInfo method;

public Val () { }
public Val (string name) { this.name=name; }
public Val (string name, string cat) { this.name=name; this.cat=cat; }


[System.NonSerialized] private static readonly Dictionary<Type, Val[]> attributesCaches = new Dictionary

public static Val[] GetAttributes (Type type)
{
 if (attributesCaches.TryGetValue(type, out Val[] attributes)) return attributes;

 List<Val> attList = new List<Val>();

 FieldInfo[] fields = type.GetFields();
 for (int f=0; f<fields.Length; f++)
 {
  Val valAtt = Attribute.GetCustomAttribute(fields[f], typeof(Val)) as Val;
```

```
 if (valAtt == null) continue;

 valAtt.field = fields[f];
 valAtt.type = fields[f].FieldType;

 attList.Add(valAtt);
}

PropertyInfo[] props = type.GetProperties();
for (int p=0; p<props.Length; p++)
{
 Val valAtt = Attribute.GetCustomAttribute(props[p], typeof(Val)) as Val;
 if (valAtt == null) continue;

 valAtt.prop = props[p];
 valAtt.type = props[p].PropertyType;

 attList.Add(valAtt);
}

attributes = attList.ToArray();

//if (attributes != null)
// Array.Sort(attributes, (x,y) => 0);//y.priority.CompareTo(x.priority));

attributesCaches.Add(type, attributes);
```

```
    return attributes;
  }



 }
}
*/
```

```
ï»¿using System;
using System.Reflection;
using System.Collections.Generic;
//using UnityEngine.Profiling;

namespace Den.Tools.GUI
{
 public sealed class SpecialEditorAttribute : Attribute
 {
  public string className;
  public string actionName;

  public SpecialEditorAttribute (string className, string actionName) { this.className=className; this.act
  public SpecialEditorAttribute (string className, string actionName, string cat) { this.className=classNar

  [System.NonSerialized] private static readonly Dictionary<Type,Delegate> actionsCache = new Dictionary



  [System.NonSerialized] private static readonly Dictionary<Type, Delegate> delegateCaches = new Dictio
  [System.NonSerialized] private static readonly Dictionary<Type, MethodInfo> methodsCaches = new Dict

  private static MethodInfo GetEditorMethod (Type type)
  /// Using original (non-editor) type get gui action delegate
  {
   if (methodsCaches.TryGetValue(type, out MethodInfo editorMethod)) return editorMethod;
```

```
 SpecialEditorAttribute customEditorAttribute = Attribute.GetCustomAttribute(type, typeof(SpecialEditorAtt
 if (customEditorAttribute != null)
 {
  Type editorType = GetEditorType(type, customEditorAttribute.className);
  if (editorType != null)
  {
   editorMethod = editorType.GetMethod(customEditorAttribute.actionName);
   if (editorMethod == null)
    throw new Exception("Could not find method " + customEditorAttribute.actionName + " in " + customEd
  }
 }

 methodsCaches.Add(type, editorMethod);
 return editorMethod;
}

public static void Draw2 (object obj, Type nullObjType=null)
/// Type argument for in case of object == null
/// If object is null using nullObjType to determine it's type
{
 Type type = obj!=null ? obj.GetType() : nullObjType; //note that using direct object type when it's not null

 MethodInfo guiMethod = GetEditorMethod(type);
 if (guiMethod == null) return;

 Action<object> guiAction = Delegate.CreateDelegate(typeof(Action<object>), guiMethod) as Action<obje
```

```
 guiAction(obj);
}



public static void Draw<TO> (TO obj)
{
 Type type = obj.GetType();
 SpecialEditorAttribute customEditorAttribute = Attribute.GetCustomAttribute(type, typeof(SpecialEditorAtt
 if (customEditorAttribute == null) return;

 Type editorType = GetEditorType(type, customEditorAttribute.className);
 MethodInfo editorMethod = editorType.GetMethod(customEditorAttribute.actionName);
 if (editorMethod == null)
  throw new Exception("Could not find method " + customEditorAttribute.actionName + " in " + customEdit

 Action<TO> editorAction;
 if (actionsCache.ContainsKey(type)) editorAction = actionsCache[type] as Action<TO>;
 else
 {
  ParameterInfo[] args = editorMethod.GetParameters();
  if (args.Length != 1)
   throw new Exception("Special Editor: Number of method arguments (" + args.Length + ") doesn't match
  if (args[0].ParameterType != typeof(TO))
   throw new Exception("Special Editor: Arguments don't match: \n" +
    "\t" + args[0].ParameterType + " vs " + typeof(TO) );
```

```
  editorAction = Delegate.CreateDelegate(typeof(Action<TO>), editorMethod) as Action<TO>;
  actionsCache.Add(type,editorAction);
 }
 editorAction(obj);
}

public static void Draw<TO, T1> (TO obj, T1 t1)
{
 Type type = obj.GetType();
 SpecialEditorAttribute customEditorAttribute = Attribute.GetCustomAttribute(type, typeof(SpecialEditorAtt
 if (customEditorAttribute == null) return;

 Type editorType = GetEditorType(type, customEditorAttribute.className);
 MethodInfo editorMethod = editorType.GetMethod(customEditorAttribute.actionName);
 if (editorMethod == null)
  throw new Exception("Could not find method " + customEditorAttribute.actionName + " in " + customEdit

 Action<TO,T1> editorAction;
 if (actionsCache.ContainsKey(type)) editorAction = actionsCache[type] as Action<TO,T1>;
 else
 {
  ParameterInfo[] args = editorMethod.GetParameters();
  if (args.Length != 2)
   throw new Exception("Special Editor: Number of method arguments (" + args.Length + ") doesn't match
  if (args[0].ParameterType != typeof(TO)  ||  args[1].ParameterType != typeof(T1) )
   throw new Exception("Special Editor: Arguments don't match: \n" +
```

```
    "\t" + args[0].ParameterType + " vs " + typeof(TO) + "\n" +
    "\t" + args[1].ParameterType + " vs " + typeof(T1) );

  editorAction = Delegate.CreateDelegate(typeof(Action<TO,T1>), editorMethod) as Action<TO,T1>;
  actionsCache.Add(type,editorAction);
 }
 editorAction(obj, t1);
}


public static void Draw<TO, T1,T2> (TO obj, T1 t1, T2 t2)
{
 Type type = obj.GetType();
 SpecialEditorAttribute customEditorAttribute = Attribute.GetCustomAttribute(type, typeof(SpecialEditorAtt
 if (customEditorAttribute == null) return;

 Type editorType = GetEditorType(type, customEditorAttribute.className);
 MethodInfo editorMethod = editorType.GetMethod(customEditorAttribute.actionName);
 if (editorMethod == null)
  throw new Exception("Special Editor: Could not find method " + customEditorAttribute.actionName + " in

 Action<TO,T1,T2> editorAction;
 if (actionsCache.ContainsKey(type)) editorAction = actionsCache[type] as Action<TO,T1,T2>;
 else
 {
  ParameterInfo[] args = editorMethod.GetParameters();
```

```
 if (args.Length != 3)
  throw new Exception("Special Editor: Number of method arguments (" + args.Length + ") doesn't match
 if (args[0].ParameterType != typeof(TO)  ||  args[1].ParameterType != typeof(T1)  ||  args[2].ParameterTy
  throw new Exception("Special Editor: Arguments don't match: \n" +
   "\t" + args[0].ParameterType + " vs " + typeof(TO) + "\n" +
   "\t" + args[1].ParameterType + " vs " + typeof(T1) + "\n" +
   "\t" + args[2].ParameterType + " vs " + typeof(T2) );

 editorAction = Delegate.CreateDelegate(typeof(Action<TO,T1,T2>), editorMethod) as Action<TO,T1,T2>
 actionsCache.Add(type,editorAction);
}
editorAction(obj, t1, t2);
}


public static void Draw<TO, T1,T2,T3> (TO obj, T1 t1, T2 t2, T3 t3)
{
 Type type = obj.GetType();
 SpecialEditorAttribute customEditorAttribute = Attribute.GetCustomAttribute(type, typeof(SpecialEditorAtt
 if (customEditorAttribute == null) return;

 Type editorType = GetEditorType(type, customEditorAttribute.className);
 MethodInfo editorMethod = editorType.GetMethod(customEditorAttribute.actionName);
 if (editorMethod == null)
  throw new Exception("Special Editor: Could not find method " + customEditorAttribute.actionName + " in
```

```
 Action<TO,T1,T2,T3> editorAction;
 if (actionsCache.ContainsKey(type)) editorAction = actionsCache[type] as Action<TO,T1,T2,T3>;
 else
 {
  ParameterInfo[] args = editorMethod.GetParameters();
  if (args.Length != 4)
   throw new Exception("Special Editor: Number of method arguments (" + args.Length + ") doesn't match
  if (args[0].ParameterType != typeof(TO)  ||  args[1].ParameterType != typeof(T1)  ||  args[2].ParameterTy
   throw new Exception("Special Editor: Arguments don't match: \n" +
    "\t" + args[0].ParameterType + " vs " + typeof(TO) + "\n" +
    "\t" + args[1].ParameterType + " vs " + typeof(T1) + "\n" +
    "\t" + args[2].ParameterType + " vs " + typeof(T2) + "\n" +
    "\t" + args[3].ParameterType + " vs " + typeof(T3) );

  editorAction = Delegate.CreateDelegate(typeof(Action<TO,T1,T2,T3>), editorMethod) as Action<TO,T1,
  actionsCache.Add(type,editorAction);
 }
 editorAction(obj, t1, t2, t3);
}


public static Type GetEditorType (Type baseType, string editorTypeName)
{
 Type editorType = null;
 editorType = Type.GetType(editorTypeName);
 if (editorType != null) return editorType;
```

```
//finding type in it'a assembly
Assembly ass = baseType.Assembly; //There are only two hard things in Computer Science: cache invali
editorType = ass.GetType(editorTypeName);
if (editorType != null) return editorType;

//finding type in it's editor assembly
Assembly editorAss = null;
string assName = ass.GetName().Name;
string editorAssName = assName + "Editor";
foreach (var a in AppDomain.CurrentDomain.GetAssemblies())
{
 if (a.GetName().Name == editorAssName)
  { editorAss = a; break; }
}
if (editorAss != null)
{
 editorType = editorAss.GetType(editorTypeName);
 if (editorType != null) return editorType;
}

//finding type without namespace
if (editorType == null)
{
 Type[] assTypes = ass.GetTypes();
```

```
 for (int i=0; i<assTypes.Length; i++)
 {
  string typeName = assTypes[i].Name;
  int lastIndexOfDot = typeName.LastIndexOf('.');
  if (lastIndexOfDot >= 0)
   typeName = typeName.Substring(typeName.LastIndexOf('.'), typeName.Length);
  if (typeName == editorTypeName)
   { editorType = assTypes[i]; break; }
 }
}
if (editorType != null) return editorType;

//finding type without namespace in editor assembly
if (editorType == null && editorAss != null)
{
 Type[] assTypes = editorAss.GetTypes();

 for (int i=0; i<assTypes.Length; i++)
 {
  string typeName = assTypes[i].Name;
  int lastIndexOfDot = typeName.LastIndexOf('.');
  if (lastIndexOfDot >= 0)
   typeName = typeName.Substring(typeName.LastIndexOf('.'), typeName.Length);
  if (typeName == editorTypeName)
   { editorType = assTypes[i]; break; }
 }
```

```
    }
    return editorType;
  }
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using UnityEngine;


namespace Den.Tools.GUI
{

 public class Cell : IDisposable
 {
  #if UNITY_2019_3_OR_NEWER
  public const int lineHeight = 20;
  #else
  public const int lineHeight = 18;
  #endif

  public static List<Cell> activeStack = new List<Cell>();
  public static Cell current; // { get { return activeStack[activeStack.Count-1]; } }
  public static Cell Parent
  {get{
   int activeStackCount = activeStack.Count;
   if (activeStack.Count < 2) return null;
   else return activeStack[activeStack.Count-2];
  }}
```

```
public bool stackHor;
public bool stackVer;
public Vector2 relativeSize;
public Vector2 relativeOffset;
public Vector2 pixelSize;
public Vector2 pixelOffset;
//public Vector2 pixelResize; //same as offset for size. Adjust the final size with this value

public Vector2 minPixelSize;  // cell size could not be lesser than that because of this.pixelSize | contentsI
public Vector2 contentsPixelSize;  //only contents size, this.pixelSize could set bigger value
public Vector2 finalSize;
public Vector2 cellPosition;
public Vector2 worldPosition;


//public Rect cellRect;
//public Rect worldRect;

public List<Cell> subCells;
public int subCounter;

public IEnumerable SubCellsRecursively (bool includeSelf=false)
{
 if (includeSelf)
  yield return this;
```

```
    if (subCells ==null)
     yield break;

    //top level first
    int count = subCells.Count;
    for (int i=0; i<count; i++)
    //for (int i=0; i<subCounter; i++)
    //iterating real list, since most sub cells won't be active
     yield return subCells[i];

    for (int i=0; i<count; i++)
     foreach (Cell subSub in subCells[i].SubCellsRecursively())
      yield return subSub;
   }

//  public string name;
   public override string ToString() { return "Cell" + " " + worldPosition.x + "," + worldPosition.y + " " + finalSiz

   public bool disabled;
   public bool inactive; //same view, but controls (field drag, buttons) do not work
   public float fieldWidth = 0.5f;

   public bool isLayouted = true; //is this cells size taken into account when calculating parent size? (no for s

   public bool trackChange = true;
```

```
public bool valChanged;

[Flags] public enum Special { None=0, Label=1, Field=2, LabelField=4, Vector=8, VectorX=16, VectorY=3
public Special special; //Just determines the kind of cell by it's contents with Draw. Was made to integrate



public Vector2 InternalCenter {get{ return new Vector2(worldPosition.x+finalSize.x/2f, worldPosition.y+fina

public Rect InternalRect //note it's internal rect, not scaled with zoom!
{
 get{ return new Rect(worldPosition.x, worldPosition.y, finalSize.x, finalSize.y); }
 set{ worldPosition=value.position; finalSize=value.size; }
}

public bool Contains (Vector2 pos, int padding=0)
{
 return  pos.x > worldPosition.x-padding && pos.x < worldPosition.x+finalSize.x+padding &&
   pos.y > worldPosition.y-padding && pos.y < worldPosition.y+finalSize.y+padding;
}

public Rect GetRect (ScrollZoom scrollZoom=null, int padding=0, bool pixelPerfect=true)
{
 if (scrollZoom==null)
  return new Rect(
   worldPosition.x + padding,
   worldPosition.y + padding,
```

```
    finalSize.x - padding*2,
    finalSize.y - padding*2);
  else
   return scrollZoom.ToScreen(
    worldPosition.x + padding,
    worldPosition.y + padding,
    finalSize.x - padding*2,
    finalSize.y - padding*2,
    pixelPerfect:pixelPerfect);
}

public void MakeStatic ()
/// Discards scroll/zoom information for this cell, making it to be displayed at the same pos no matter of sc
{
 if (UI.current.scrollZoom != null)
 {
  pixelSize /= UI.current.scrollZoom.zoom;
  pixelOffset = (pixelOffset - UI.current.scrollZoom.scroll) / UI.current.scrollZoom.zoom;
 }
}



#region Layout

 public void CalculateMinContentsSize ()
 /// Gets minPixelSize (as well as contentsPixelSize btw)
```

```
/// From child to parent
{
 minPixelSize = pixelSize;
 //don't use assigned pixelSize anymore, it should not be changed with layout

 if (subCells == null) return;
 int subCount = subCells.Count;
 if (subCount == 0) return;

 contentsPixelSize = new Vector2();
 for (int s=0; s<subCount; s++)
 {
  Cell sub = subCells[s];

  if (sub.subCells != null)
   sub.CalculateMinContentsSize();
  else
   sub.minPixelSize = sub.pixelSize; //instead of CalculateMinContentsSize

  if (!sub.isLayouted)
   continue;

  if (sub.stackHor) contentsPixelSize.x += sub.minPixelSize.x;
  else contentsPixelSize.x = contentsPixelSize.x>sub.minPixelSize.x ? contentsPixelSize.x : sub.minPixe

  if (sub.stackVer) contentsPixelSize.y += sub.minPixelSize.y;
```

```
  else contentsPixelSize.y = contentsPixelSize.y>sub.minPixelSize.y ? contentsPixelSize.y : sub.minPixe
 }

 if (contentsPixelSize.x > minPixelSize.x) minPixelSize.x = contentsPixelSize.x;
 if (contentsPixelSize.y > minPixelSize.y) minPixelSize.y = contentsPixelSize.y;
}



public void CalculateSubRects ()
/// Sets finalPixelSize for child cells
/// From parent to children
{
 if (subCells == null) return;
 int subCount = subCells.Count;
 if (subCount == 0) return;

 //calculating relative sum
 float relativeSizeSumX=0; float relativeSizeSumY=0; //Vector2 relativeSizeSum = new Vector2();
 for (int s=0; s<subCount; s++)
 {
  Cell sub = subCells[s];
  if (sub.stackHor) relativeSizeSumX += sub.relativeSize.x;
  if (sub.stackVer) relativeSizeSumY += sub.relativeSize.y;
 }

 //calculating the number of remaining pixels for relative
```

```
float subsPixelsSumX=0; float subsPixelsSumY=0; //Vector2 subsPixelsSum = new Vector2();
for (int s=0; s<subCount; s++)
{
 Cell sub = subCells[s];
 subsPixelsSumX += sub.pixelSize.x; //using assigned size, ignoring subs min contents size
 subsPixelsSumY += sub.pixelSize.y;
}
//Vector2 relativePixelsLeft = new Vector2(finalSize.x - subsPixelsSum.x, finalSize.y - subsPixelsSum.y)
float relativePixelsLeftX = finalSize.x - subsPixelsSumX;
float relativePixelsLeftY = finalSize.y - subsPixelsSumY;

//setting rects
float currentPosX=0; float currentPosY=0;  //Vector2 currentPos = new Vector2();
for (int s=0; s<subCount; s++)
{
 Cell sub = subCells[s];

 //horizontal
 if (sub.stackHor)
 {
  float relativePixelsX = 0;
  if (relativeSizeSumX != 0 && sub.relativeSize.x != 0)
   relativePixelsX = sub.relativeSize.x/relativeSizeSumX * relativePixelsLeftX;

  sub.finalSize.x = sub.minPixelSize.x>relativePixelsX ? sub.minPixelSize.x : relativePixelsX ; //Mathf.Ma
  sub.cellPosition.x = currentPosX;
```

```
 currentPosX += sub.finalSize.x;
}
else
{
 sub.finalSize.x = //Mathf.Max(sub.minPixelSize.x, finalSize.x*sub.relativeSize.x+sub.pixelSize.x);
  sub.minPixelSize.x > finalSize.x*sub.relativeSize.x+sub.pixelSize.x ?
   sub.minPixelSize.x :
   finalSize.x*sub.relativeSize.x+sub.pixelSize.x;
 sub.cellPosition.x = finalSize.x * sub.relativeOffset.x;
}

//vertical
if (sub.stackVer)
{
 float relativePixelsY = 0;
 if (relativeSizeSumY != 0 && sub.relativeSize.y != 0)
  relativePixelsY = sub.relativeSize.y/relativeSizeSumY * relativePixelsLeftY;

 sub.finalSize.y = sub.minPixelSize.y>relativePixelsY ? sub.minPixelSize.y : relativePixelsY; //Mathf.Ma
 sub.cellPosition.y = currentPosY;

 currentPosY += sub.finalSize.y;
}
else
{
```

```
   sub.finalSize.y = //Mathf.Max(sub.minPixelSize.y, finalSize.y*sub.relativeSize.y+sub.pixelSize.y);
    sub.minPixelSize.y > finalSize.y*sub.relativeSize.y+sub.pixelSize.y ?
     sub.minPixelSize.y :
     finalSize.y*sub.relativeSize.y+sub.pixelSize.y;
   sub.cellPosition.y = finalSize.y * sub.relativeOffset.y;
  }

  sub.cellPosition += sub.pixelOffset;
  sub.worldPosition = worldPosition + sub.cellPosition;

  ///sub.finalSize += sub.pixelResize;

  if (sub.subCells != null)
   sub.CalculateSubRects();
 }
}



public void CalculateRootRects ()
/// Called on root cell to calculate all rects
/// From parent to children
{
 cellPosition.x = pixelOffset.x;
 cellPosition.y = pixelOffset.y;
 finalSize.x = minPixelSize.x;
 finalSize.y = minPixelSize.y;
```

```
 worldPosition = cellPosition;


 CalculateSubRects();
}



public void FillCellsUnderCursor (List<Cell> cells, Vector2 mousePos)
/// Finds all child cells containing mousePos
/// Should be done after layout
{
 if (mousePos.x > worldPosition.x  &&  mousePos.x < worldPosition.x+finalSize.x  &&
  mousePos.y > worldPosition.y  &&  mousePos.y < worldPosition.y+finalSize.y)
   cells.Add(this);


 if (subCells != null)
 {
  int subCount = subCells.Count;
  for (int s=0; s<subCount; s++)
  {
   Cell sub = subCells[s];
   subCells[s].FillCellsUnderCursor(cells, mousePos);
  }
 }
}
```

```
 public void UseEvenetsOnClick ()
 {
  if (Event.current.isMouse)
   Event.current.Use();
 }



#endregion



#region Factory / Dispose

 public static Cell GetCell ()
 {
  //trying to re-use existing cell
  Cell cell = null;
  if (current != null  &&  current.subCells != null)
  {
   int subsCount = current.subCells.Count;
   if (current.subCounter < subsCount)
   {
    cell = current.subCells[current.subCounter];
    current.subCounter ++;
   }
  }
```

```
 //creating new if could not re-use
 if (cell == null)
 {
  cell = new Cell();

  if (current != null)
  {
   if (current.subCells == null)
    current.subCells = new List<Cell>();

   current.subCells.Add(cell);
   current.subCounter ++;
  }
 }

 if (UI.current.layout) //all parameters should remain the same on repaint
  cell.Init(current);

 return cell;
}

public void Init (Cell parent)
{
 //resetting child counter
```

```
 subCounter = 0;

 //resetting parameters
 pixelOffset.x=0;  pixelOffset.y=0;
 pixelSize.x=0;   pixelSize.y=0;
 relativeSize.x=0;  relativeSize.y=0;
 relativeOffset.x=0;  relativeOffset.y=0;
 pixelOffset.x=0;  pixelOffset.y=0;
 stackHor = false;
 stackVer = false;
 isLayouted = true;
 special = Special.None;

 //copy properties
 trackChange = parent.trackChange;
 valChanged = false;
 disabled = parent.disabled;
 inactive = parent.inactive;
 fieldWidth = parent.fieldWidth;
}


//[MethodImpl(MethodImplOptions.AggressiveInlining)]
public void Activate ()
{
 activeStack.Add(this);
```

```
 current = this;
}



public void UpdateSubs ()
/// Removes extra subs and resets subCounter recursively
{
 if (subCells == null) return;

 int subsCount = subCells.Count;
 if (subCounter < subsCount-1)
  subCells.RemoveRange(subCounter, subsCount-1-subCounter);

 subCounter = 0;

 for (int s=0; s<subsCount; s++)
 {
  if (subCells[s].subCells != null)
   subCells[s].UpdateSubs();
 }
}



public void Dispose ()
{
```

```
 //if (activeStack[lastIndex] != this)
 // throw new Exception("Cell: Trying to de-activate cell that is currently non-active");

// int lastIndex = activeStack.Count-1;
// activeStack.RemoveAt(lastIndex);
// if (activeStack.Count != 0) current = activeStack[activeStack.Count-1];
// else current = null;


 //if (current != this)
 // throw new Exception("Cell: Trying to de-activate cell that is currently non-active");

 int activeStackCount = activeStack.Count;
 if (activeStackCount == 0)
  return; //otherwise EditorUtility.DisplayDialog will log an error

 activeStack.RemoveAt(activeStackCount-1); //removing last
 if (activeStackCount > 1) //if there's something in stack after remove
  current = activeStack[activeStackCount-2]; //then assigning the new last one
 else current = null;

 //removing from child list
 /*if (activeStack.Count != 0)
 {
  Cell prevActive = activeStack[activeStack.Count-1];
```

```
   //prevActive.subCells.Remove(this);
  }*/

  //removing extra children
  if (subCells != null)
  {
   int subsCount = subCells.Count;
   if (subCounter <= subsCount-1)
    subCells.RemoveRange(subCounter, subsCount-subCounter);
  }

  subCounter = 0;
 }



 public void Skip ()
 /// Emulates cell being drawn while it isn't (out of window, for instance)
 /// Prevents it being ruined on Dispose
 {
  if (subCells != null)
   subCounter = subCells.Count;
 }

#endregion
```

```
  #region Static Constructors

   public static Cell Line
   {get{
    Cell cell = GetCell();
    cell.stackVer = true;
    cell.relativeSize = new Vector2(1,1);
//    cell.pixelSize = new Vector2(0,0);
//    cell.pixelOffset = new Vector2(0,0);
    cell.Activate();
    return cell;
   }}


   public static Cell LinePx (float size)
   {
    Cell cell = GetCell();
    cell.stackVer = true;
    cell.relativeSize = new Vector2(1,0);
    cell.pixelSize = new Vector2(0,size);
//    cell.pixelOffset = new Vector2(0,0);
    cell.Activate();
    return cell;
   }

   public static Cell LineRel (float size)
```

```
    {
     Cell cell = GetCell();
     cell.stackVer = true;
     cell.relativeSize = new Vector2(1,size);
//     cell.pixelSize = new Vector2(0,0);
//     cell.pixelOffset = new Vector2(0,0);
     cell.Activate();
     return cell;
    }

    public static void EmptyLinePx (float size)
    {
     Cell cell = GetCell();
     cell.stackVer = true;
     cell.relativeSize = new Vector2(1,0);
     cell.pixelSize = new Vector2(0,size);
//     cell.pixelOffset = new Vector2(0,0);
     if (cell.subCells != null) cell.subCells.Clear();
    }

    public static void EmptyLineRel (float size)
    {
     Cell cell = GetCell();
     cell.stackVer = true;
     cell.relativeSize = new Vector2(1,size);
//     cell.pixelSize = new Vector2(0,0);
```

```
//    cell.pixelOffset = new Vector2(0,0);
     if (cell.subCells != null) cell.subCells.Clear();
    }

    public static void EmptyLine ()
    {
     Cell cell = GetCell();
     cell.stackVer = true;
     cell.relativeSize = new Vector2(1,1);
//    cell.pixelSize = new Vector2(0,0);
//    cell.pixelOffset = new Vector2(0,0);
     if (cell.subCells != null) cell.subCells.Clear();
    }


    public static Cell LineStd
    {get{
     Cell cell = GetCell();
     cell.stackVer = true;
     cell.relativeSize = new Vector2(1,0);
     cell.pixelSize = new Vector2(0, lineHeight);
//    cell.pixelOffset = new Vector2(0,0);
     cell.Activate();
     return cell;
    }}
```

```
    public static Cell Row
    {get{
     Cell cell = GetCell();
     cell.stackHor = true;
     cell.relativeSize = new Vector2(1,1);
     cell.Activate();
     return cell;
    }}


    public static Cell RowPx (float size)
    {
     Cell cell = GetCell();
     cell.stackHor = true;
     cell.relativeSize = new Vector2(0,1);
     cell.pixelSize = new Vector2(size, 0);
//     cell.pixelOffset = new Vector2(0,0);
     cell.Activate();
     return cell;
    }


    public static Cell RowRel (float size)
    {
     Cell cell = GetCell();
```

```
    cell.stackHor = true;
    cell.relativeSize = new Vector2(size,1);
//    cell.pixelSize = new Vector2(0,0);
//    cell.pixelOffset = new Vector2(0,0);
    cell.Activate();
    return cell;
   }


   public static void EmptyRowPx (float size)
   {
    Cell cell = GetCell();
    cell.stackHor = true;
    cell.relativeSize = new Vector2(0,1);
    cell.pixelSize = new Vector2(size, 0);
//    cell.pixelOffset = new Vector2(0,0);
    if (cell.subCells != null) cell.subCells.Clear();
   }


   public static void EmptyRowRel (float size)
   {
    Cell cell = GetCell();
    cell.stackHor = true;
    cell.relativeSize = new Vector2(size,1);
//    cell.pixelSize = new Vector2(0,0);
```

```
//    cell.pixelOffset = new Vector2(0,0);
     if (cell.subCells != null) cell.subCells.Clear();
    }

    public static void EmptyRow ()
    {
     Cell cell = GetCell();
     cell.stackHor = true;
     cell.relativeSize = new Vector2(1,1);
//    cell.pixelSize = new Vector2(0,0);
//    cell.pixelOffset = new Vector2(0,0);
     if (cell.subCells != null) cell.subCells.Clear();
    }


    public static Cell Custom (float sizeX, float sizeY)
    {
     Cell cell = GetCell();
     cell.pixelSize = new Vector2(sizeX, sizeY);
     cell.stackHor = false;
     cell.stackVer = false;
     cell.Activate();
     return cell;
    }

    public static Cell Custom (Vector2 pos, Vector2 size) => Custom(pos.x, pos.y, size.x, size.y);
```

```
    public static Cell Custom (float posX, float posY, float sizeX, float sizeY)
    {
     Cell cell = GetCell();
//    cell.relativeSize = new Vector2(0,0);
     cell.pixelSize = new Vector2(sizeX, sizeY);
     cell.pixelOffset = new Vector2(posX, posY);
     cell.Activate();
     return cell;
    }


    public static Cell Custom (Rect rect)
    {
     Cell cell = GetCell();
//    cell.relativeSize = new Vector2(0,0);
     cell.pixelSize = rect.size;
     cell.pixelOffset = rect.position;
     cell.Activate();
     return cell;
    }

    public static Cell Custom (Cell other)
    {
     Cell cell = GetCell();
     cell.relativeOffset = other.relativeOffset;
```

```
 cell.relativeSize = other.relativeSize;
 cell.isLayouted = other.isLayouted;
 cell.pixelOffset = other.pixelOffset;
 cell.pixelSize = other.pixelSize;
 cell.stackHor = other.stackHor;
 cell.stackVer = other.stackVer;
 cell.Activate();
 return cell;
}

public static Cell Static (float posX, float posY, float sizeX, float sizeY)
///Cell independent from current scroll zoom
{
 Cell cell = GetCell();
 cell.pixelSize = new Vector2(sizeX, sizeY);
 cell.pixelOffset = new Vector2(posX, posY);

 if (UI.current.scrollZoom != null)
 {
  cell.pixelSize /= UI.current.scrollZoom.zoom;
  cell.pixelOffset = (cell.pixelOffset - UI.current.scrollZoom.scroll) / UI.current.scrollZoom.zoom;
 }

 cell.Activate();
 return cell;
```

```
    }



    public static Cell Full
    {get{
     Cell cell = GetCell();
     cell.relativeSize = new Vector2(1,1);
//    cell.pixelSize = new Vector2(0,0);
     cell.pixelOffset = new Vector2(0,0);
     cell.Activate();
     return cell;
    }}



    public static Cell Padded (int left, int right, int top, int bottom) //padding will not be counted in min size
    {
     Cell cell = GetCell();
     cell.relativeSize = new Vector2(1,1);
     cell.pixelSize = new Vector2(-left-right, -top-bottom);
     cell.pixelOffset = new Vector2(left,top);
//    cell.pixelResize = new Vector2(-left-right, -top-bottom);
     cell.Activate();
     return cell;
    }
```

```
  public static Cell Padded (int padding)
  {
   Cell cell = GetCell();
   cell.relativeSize = new Vector2(1,1);
   cell.pixelSize = new Vector2(-padding*2, -padding*2);
   cell.pixelOffset = new Vector2(padding,padding);
//   cell.pixelResize = new Vector2(-padding*2, -padding*2);
   cell.Activate();
   return cell;
  }


  public static Cell Center (float width, float height)
  {
   Cell cell = GetCell();
   //cell.relativeSize = new Vector2(0,0);
   cell.relativeOffset = new Vector2(0.5f, 0.5f);
   cell.pixelSize = new Vector2(width, height);
   cell.pixelOffset = new Vector2(-width/2, -height/2);
//   cell.pixelResize = new Vector2(width, height);
   cell.Activate();
   return cell;
  }

  public static Cell Root (ref Cell cell, Vector2 min, Vector2 max)
  {
```

```
    if (activeStack.Count != 0)
     throw new Exception("Trying to create root cell with non-empty active stack");

    if (cell == null) cell = new Cell();
    cell.Activate(); //activeStack.Add(cell);

//    cell.relativeSize = new Vector2(0,0);
    cell.pixelSize = max-min;
    cell.pixelOffset = min;

    return cell;
   }


   public static Cell Root (ref Cell cell, Rect rect)
   {
    if (activeStack.Count != 0)
    {
     activeStack.Clear();
     throw new Exception("Trying to create root cell with non-empty active stack");
    }

    if (cell == null) cell = new Cell();
    cell.Activate(); //activeStack.Add(cell);
    cell.valChanged = false;
```

```
//    cell.relativeSize = new Vector2(0,0);
    cell.pixelSize = rect.size;
    cell.pixelOffset = rect.position;

    return cell;
   }

   public static Cell Root (ref Cell cell, float posX, float posY, float sizeX, float sizeY)
   {
    if (activeStack.Count != 0)
     throw new Exception("Trying to create root cell with non-empty active stack");

    if (cell == null) cell = new Cell();
    cell.Activate(); //activeStack.Add(cell);
    cell.valChanged = false;

//    cell.relativeSize = new Vector2(0,0);
    cell.pixelSize = new Vector2(sizeX, sizeY);
    cell.pixelOffset = new Vector2(posX, posY);

    return cell;
   }

  #endregion

 }
```

}

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using UnityEngine;


namespace Den.Tools.GUI
{
 public class CellObjs
 {
  // id (string)  -> cell -> object
  //    -> object -> cell

  private class TwoWayDict
  {
   public Dictionary<Cell, object> cellToObj = new Dictionary<Cell, object>();
   public Dictionary<object, Cell> objToCell = new Dictionary<object, Cell>();
  }

  private Dictionary<string, TwoWayDict> dict = new Dictionary<string, TwoWayDict>();


  public void ForceAdd (object obj, Cell cell, string id=null)
  {
   if (!dict.TryGetValue(id, out TwoWayDict twd))
```

```
 {
  twd = new TwoWayDict();
  dict.Add(id, twd);
 }

 if (!twd.cellToObj.ContainsKey(cell)) twd.cellToObj.Add(cell, obj);
 else twd.cellToObj[cell] = obj;

 if (!twd.objToCell.ContainsKey(obj)) twd.objToCell.Add(obj, cell);
 else twd.objToCell[obj] = cell;
}


public bool TryGetObject<T> (Cell cell, string id, out T tobj)
{
 if (!dict.TryGetValue(id, out TwoWayDict twd))
  { tobj=default; return false; }

 if (!twd.cellToObj.TryGetValue(cell, out object obj))
  { tobj=default; return false; }

 if (!(obj is T))
  { tobj=default; return false; }

 tobj = (T)obj;
 return true;
```

```
}



public bool TryGetCell (object obj, string id, out Cell cell)
{
 if (!dict.TryGetValue(id, out TwoWayDict twd))
  { cell=null; return false; }

 if (!twd.objToCell.TryGetValue(obj, out Cell objcell))
  { cell=null; return false; }

 cell = objcell;
 return true;
}



public T GetObject<T> (Cell cell, string id=null)
{
 if (!dict.TryGetValue(id, out TwoWayDict twd))
  return default;

 if (!twd.cellToObj.TryGetValue(cell, out object obj))
  return default;

 return (T)obj;
}
```

```
public Cell GetCell (object obj, string id)
{
 if (!dict.TryGetValue(id, out TwoWayDict twd))
  return null;

 if (!twd.objToCell.TryGetValue(obj, out Cell cell))
  return null;

 return cell;
}


public bool ContainsCell (Cell cell, string id)
{
 if (!dict.TryGetValue(id, out TwoWayDict twd))
  return false;

 return twd.cellToObj.ContainsKey(cell);
}


public IEnumerable<object> GetAllCells (string id)
{
 if (!dict.TryGetValue(id, out TwoWayDict twd))
```

```
    yield break;

   foreach (Cell cell in twd.cellToObj.Keys)
    yield return cell;
  }

  public IEnumerable<T> GetAllObjects<T> (string id)
  {
   if (!dict.TryGetValue(id, out TwoWayDict twd))
    yield break;

   foreach (object obj in twd.objToCell.Keys)
    yield return (T)obj;
  }


  public void Clear ()
  {
   foreach (TwoWayDict twd in dict.Values)  //leaving ids to avoid creating garbage
   {
    twd.cellToObj.Clear();
    twd.objToCell.Clear();
   }
  }
 }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;

namespace Den.Tools.GUI
{
 public static class DragDrop
 {
  public static object obj; //drag object
  public static string group; //and id. Drag if both obj and id match. Used in DragField, could be used in Laye

  public static Rect initialRect; //rect provided with CheckStart
  public static Vector2 initialMousePos;
  public static Vector2 prevMousePos;

  public static Vector2 totalDelta; //drag pos relative to initial position
  public static Vector2 currentDelta;

  public static object releasedObj;
  public static object startedObj;

  //execute order:
  // - TryDrag
```

```
// - TryRelease
// - TryStart - always the last
// int the end of frame - ResetOnUse

public static void Drag (object obj)
{

}

public static void ForceStart (object obj, Vector2 mousePos, Rect rect)
{
 DragDrop.obj = obj;
 startedObj = obj;

 initialMousePos = mousePos;
 initialRect = rect;
 prevMousePos = mousePos;
 totalDelta = new Vector2();
 currentDelta = new Vector2();

 UI.current.editorWindow?.Repaint();
}

public static bool TryStart (object obj, Vector2 mousePos, Rect rect)
{
 if (UI.current.layout) return false; //dragging could be started only in repaint (how we get rect otherwise?)
```

```
 if (Event.current.type==EventType.MouseDown  )
 if ( Event.current.button==0  &&  rect.Contains(mousePos))
 {
  ForceStart(obj, mousePos, rect);
  return true;
 }
 return false;
}
public static bool TryStart (object obj, int id, Vector2 mousePos, Rect rect)  { return TryStart(new DragObj
public static bool TryStart (object obj, int id, Rect rect)  { return TryStart(new DragObj(obj,id), rect); }
public static bool TryStart (object obj, Rect rect) { return TryStart(obj, Event.current.mousePosition, rect);



public static bool TryDrag (object obj, Vector2 mousePos)
/// Done before each gui layout. Change the static position values (does not move any cell or something)
/// Done both in !UI.layout and non-repaint
{
 if (DragDrop.obj==null  ||  !DragDrop.obj.Equals(obj))
  return false;

 else
 {
  totalDelta = mousePos - initialMousePos;
  currentDelta = mousePos - prevMousePos;
```

```
  prevMousePos = mousePos;

  UI.current.editorWindow?.Repaint();

  return true;
 }
}
public static bool TryDrag (object obj, int id, Vector2 mousePos)  { return TryDrag(new DragObj(obj,id), m
public static bool TryDrag (object obj) { return TryDrag(obj, Event.current.mousePosition); }


public static bool TryRelease (object obj, Vector2 mousePos)
//mousePos is actually not used, made for uniformity
//if not working in window - check for the OnGUI mouse hack
{
 if (UI.MouseUp && !UI.current.layout)
 //if (Event.current.rawType == EventType.MouseUp && !UI.current.layout)
 {
  if (DragDrop.obj==null  ||  !DragDrop.obj.Equals(obj))
   return false;

  else
  {
   releasedObj = DragDrop.obj;
   DragDrop.obj = null;
```

```
   DragDrop.group = null;


   UI.current.editorWindow?.Repaint();


   return true;
  }
 }

 return false;
}
public static bool TryRelease (object obj, int id, Vector2 mousePos)  { return TryRelease(new DragObj(ob
public static bool TryRelease (object obj) { return TryRelease(obj, Event.current.mousePosition); }



public static bool IsDragging (object obj)
///Doesn't perform drag, just check obj equality
 { return DragDrop.obj!=null  &&  DragDrop.obj.Equals(obj); }
public static bool IsDragging (object obj, int id) { return IsDragging(new DragObj(obj,id)); }
public static bool IsDragging () { return DragDrop.obj!=null; }

public static bool IsReleased (object obj) { return releasedObj!=null  &&  releasedObj.Equals(obj); }
public static bool IsReleased () { return releasedObj!=null; }

public static bool IsStarted (object obj)
 { return startedObj!=null  &&  startedObj.Equals(obj); }
public static bool IsStarted () => startedObj!=null;
```

```
public static void ResetTempObjs ()
///Called after every gui update
{
 //resetting drag/drop on event use
 if (Event.current.type == EventType.Used  &&  Event.current.rawType != EventType.MouseUp)
  obj = null;

 releasedObj = null;
 startedObj = null;
}



public static void ResetOnUse ()
{
 //resetting drag/drop on event use
 if (Event.current.type == EventType.Used  &&  Event.current.rawType != EventType.MouseUp)
  obj = null;
}

public struct DragObj
///Temporary object to be used on drag if performing several drags per cell/object
{
 public object obj;
 public int id;
```

```
 public DragObj (object o, int i) { obj=o; id=i; }
}



public static bool ResizeRect (
 object obj,
 Vector2 mousePos,
 ref Rect rect,
 int border=6,
 Vector2 minSize = new Vector2())
{
 bool resized = false;



 Rect leftRect = new Rect( rect.x - border/2, rect.y + border, border, rect.height-border*2 );

 Rect zoomedRect = UI.current.scrollZoom==null ? leftRect : UI.current.scrollZoom.ToScreen(leftRect.pos
 UnityEditor.EditorGUIUtility.AddCursorRect (zoomedRect, UnityEditor.MouseCursor.ResizeHorizontal);

 ObjId obj1 = new ObjId (obj, 1);
 if (TryDrag(obj1, mousePos))
 {
  rect.x = (initialRect).position.x + totalDelta.x;
  rect.width = (initialRect).width - totalDelta.x;
  if (rect.width<minSize.x)  { rect.x = (initialRect).xMax - minSize.x; rect.width = minSize.x; }
  resized = true;
```

```
}
if (TryRelease(obj1, mousePos))
 { resized = true; }
if (TryStart(obj1, mousePos, leftRect))
 { initialRect = rect; resized = true; } //re-assigning initial rect to full one, not left rect


Rect rightRect = new Rect( rect.x - border/2 + rect.width, rect.y + border, border, rect.height-border*2 );

zoomedRect = UI.current.scrollZoom==null ? rightRect : UI.current.scrollZoom.ToScreen(rightRect.positi
UnityEditor.EditorGUIUtility.AddCursorRect (zoomedRect, UnityEditor.MouseCursor.ResizeHorizontal);

ObjId obj2 = new ObjId (obj, 2);
if (TryDrag(obj2, mousePos))
{
 rect.width = initialRect.width + totalDelta.x;
 if (rect.width<minSize.x)  { rect.width = minSize.x; }
 resized = true;
}
if (TryRelease(obj2, mousePos))
 { resized = true; }
if (TryStart(obj2, mousePos, rightRect))
 { initialRect = rect; resized = true; }


Rect topRect = new Rect( rect.x + border, rect.y - border/2, rect.width-border*2, border );
```

```
zoomedRect = UI.current.scrollZoom==null ? topRect : UI.current.scrollZoom.ToScreen(topRect.position

UnityEditor.EditorGUIUtility.AddCursorRect (zoomedRect, UnityEditor.MouseCursor.ResizeVertical);


ObjId obj3 = new ObjId (obj, 3);

if (TryDrag(obj3, mousePos))

{

 rect.y = initialRect.position.y + totalDelta.y;

 rect.height = initialRect.height - totalDelta.y;

 if (rect.height<minSize.y)  { rect.y = initialRect.yMax - minSize.y; rect.height = minSize.y; }

 resized = true;

}

if (TryRelease(obj3, mousePos))

 { resized = true; }

if (TryStart(obj3, mousePos, topRect))

 { initialRect = rect; resized = true; }



Rect bottomRect = new Rect( rect.x + border, rect.y - border/2 + rect.height, rect.width-border*2, border )


zoomedRect = UI.current.scrollZoom==null ? bottomRect : UI.current.scrollZoom.ToScreen(bottomRect.p

UnityEditor.EditorGUIUtility.AddCursorRect (zoomedRect, UnityEditor.MouseCursor.ResizeVertical);


ObjId obj4 = new ObjId (obj, 4);

if (TryDrag(obj4, mousePos))

{
```

```
 rect.height = initialRect.height + totalDelta.y;
 if (rect.height<minSize.y)  { rect.height = minSize.y; }
 resized = true;
}
if (TryRelease(obj4, mousePos))
 { resized = true; }
if (TryStart(obj4, mousePos, bottomRect))
 { initialRect = rect; resized = true; }



Rect topLeftRect = new Rect( rect.x-border, rect.y-border, border*2, border*2);


zoomedRect = UI.current.scrollZoom==null ? topLeftRect : UI.current.scrollZoom.ToScreen(topLeftRect.p
UnityEditor.EditorGUIUtility.AddCursorRect (zoomedRect, UnityEditor.MouseCursor.ResizeUpLeft);


ObjId obj5 = new ObjId (obj, 5);
if (TryDrag(obj5, mousePos))
{
 rect.position = initialRect.position + totalDelta;
 rect.size = initialRect.size - totalDelta;
 if (rect.width<minSize.x)  { rect.x = initialRect.xMax - minSize.x; rect.width = minSize.x; }
 if (rect.height<minSize.y)  { rect.y = initialRect.yMax - minSize.y; rect.height = minSize.y; }
 resized = true;
}
if (TryRelease(obj5, mousePos))
 { resized = true; }
```

```
if (TryStart(obj5, mousePos, topLeftRect))
 { initialRect = rect; resized = true; }



Rect topRightRect = new Rect( rect.x-border + rect.width, rect.y-border, border*2, border*2);


zoomedRect = UI.current.scrollZoom==null ? topRightRect : UI.current.scrollZoom.ToScreen(topRightRe
UnityEditor.EditorGUIUtility.AddCursorRect (zoomedRect, UnityEditor.MouseCursor.ResizeUpRight);


ObjId obj6 = new ObjId (obj, 6);
if (TryDrag(obj6, mousePos))
{
 rect.position = initialRect.position + new Vector2(0,totalDelta.y);
 rect.size = initialRect.size + new Vector2(totalDelta.x, -totalDelta.y);
 if (rect.width<minSize.x)  { rect.width = minSize.x; }
 if (rect.height<minSize.y)  { rect.y = initialRect.yMax - minSize.y; rect.height = minSize.y; }
 resized = true;
}
if (TryRelease(obj6, mousePos))
 { resized = true; }
if (TryStart(obj6, mousePos, topRightRect))
 { initialRect = rect; resized = true; }



Rect bottomLeftRect = new Rect( rect.x-border, rect.y-border + rect.height, border*2, border*2);
```

```
zoomedRect = UI.current.scrollZoom==null ? bottomLeftRect : UI.current.scrollZoom.ToScreen(bottomLe
UnityEditor.EditorGUIUtility.AddCursorRect (zoomedRect,  UnityEditor.MouseCursor.ResizeUpRight);

ObjId obj7 = new ObjId (obj, 7);
if (TryDrag(obj7, mousePos))
{
 rect.position = initialRect.position + new Vector2(totalDelta.x,0);
 rect.size = initialRect.size + new Vector2(-totalDelta.x, totalDelta.y);
 if (rect.width<minSize.x)  { rect.x = initialRect.xMax - minSize.x; rect.width = minSize.x; }
 if (rect.height<minSize.y)  { rect.height = minSize.y; }
 resized = true;
}
if (TryRelease(obj7, mousePos))
 { resized = true; }
if (TryStart(obj7, mousePos, bottomLeftRect))
 { initialRect = rect; resized = true; }

Rect bottomRightRect = new Rect( rect.x-border + rect.width, rect.y-border + rect.height, border*2, borde

zoomedRect = UI.current.scrollZoom==null ? bottomRightRect : UI.current.scrollZoom.ToScreen(bottomR
UnityEditor.EditorGUIUtility.AddCursorRect (zoomedRect, UnityEditor.MouseCursor.ResizeUpLeft);

ObjId obj8 = new ObjId (obj, 8);
if (TryDrag(obj8, mousePos))
{
```

```
    rect.size = initialRect.size + totalDelta;
    if (rect.width<minSize.x)  { rect.width = minSize.x; }
    if (rect.height<minSize.y)  { rect.height = minSize.y; }
    resized = true;
   }
   if (TryRelease(obj8, mousePos))
    { resized = true; }
   if (TryStart(obj8, mousePos, bottomRightRect))
    { initialRect = rect; resized = true; }

   return resized;
  }

  private struct ObjId { public object i1; public int i2; public ObjId (object o, int i) {i1=o; i2=i;} }
  //will be used in ResizeCell instead TupleSet <object, int>
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
using System.Reflection;
using System.Reflection.Emit;
using System.Runtime.InteropServices;
using System.Linq;
using System.Linq.Expressions;
using Den.Tools;

namespace Den.Tools.GUI
{
 public static class Draw
 {
  const int fieldPadding = 1;
  const int buttonPadding = 1;
  public const int vectorXYWidth = 15; //width of 'width' and 'height' elements in vector field
  public const float vectorXYRelWidth = 0.11f;
  const int vectorWidthHeightWidth = 42; //width of 'width' and 'height' elements in vector field

  private static Material multiplyMat; //transparent, has no texture, draws over area and fill it with color
  private static Material textureIconMat;
  private static Material textureArrIconMat;
  private static Material textureScrollZoomMat;
```

```
private static Material textureMat;
private static Material gridMat;
private static Material textureRawMat;

private static Cell pressedButton;



#region Fast Fields

 public static void Label (string label, GUIStyle style=null) //TODO: optimize style?
 {
  if (UI.current.layout) return;
  if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

  Cell.current.special |= Cell.Special.Label;

  Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);
  if (style==null) style = UI.current.styles.label;

  if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true); //BeginDisabledGroup perform
  EditorGUI.LabelField(rect, label, style:style);
  if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();
 }


 public static float Field (float val, GUIStyle style=null)
```

```
{
 if (UI.current.layout) return val;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return val;

 Cell.current.special |= Cell.Special.Field;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);
 GUIStyle cstyle = style!=null ? style : UI.current.styles.field;

 if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true);

 float newVal = val;
 if (!Cell.current.inactive)
  newVal = EditorGUI.FloatField(rect, val, style:cstyle);
  //Generates some garbage (1-2kB, check with profile), but there's nothing I can do about it
 else
  EditorGUI.LabelField(rect, val.ToString(), style:cstyle);

 if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();

 //assigning new value only on enter, tab or mouseclick (delayed mode)
 if (!newVal.Equals(val))
 {
  UI.current.delayedFloat = newVal;
  UI.current.delayedCell = Cell.current;
 }
```

```
//    if (Event.current.keyCode == KeyCode.Return  &&  !UI.current.layout)
//      Debug.Log("True");



    if (UI.current.delayedCell == Cell.current  &&  (UI.FieldLostFocus || UI.current.editorWindow != EditorWi
    {
     UI.current.MarkChanged();
     val = UI.current.delayedFloat;

     if (UI.current.editorWindow!=null) UI.current.editorWindow.Repaint();

     UI.current.delayedCell = null;
    }

    return val;
   }



   public static float Field (float val, string label, float min=float.MinValue, float max=float.MaxValue)
   {
    if (UI.current.optimizeCells && !UI.current.IsInWindow())
     { Cell.current.Skip(); return val; }

    Cell.current.special |= Cell.Special.LabelField;
```

```
 //using (Cell.RowRel(1-Cell.current.fieldWidth))
 Cell cell = Cell.RowRel(1-Cell.current.fieldWidth);
 {
  Label(label);
  if (!Cell.current.inactive)
   val = DragValue(val);
  if (val>max) val=max; //checking after drag since it going to draw field with changed value
  if (val<min) val=min;
 }
 cell.Dispose();

 //using (Cell.RowRel(Cell.current.fieldWidth))
 cell = Cell.RowRel(Cell.current.fieldWidth);
 val = Field(val);
 cell.Dispose();

 if (val>max) val=max;
 if (val<min) val=min;
 return val;
}

public static void Field (ref float val) { val = Field(val); }
public static void Field (ref float val, string label) { val = Field(val, label); }

public static int Field (int val, GUIStyle style = null)
{
```

```
if (UI.current.layout) return val;
if (UI.current.optimizeElements && !UI.current.IsInWindow()) return val;

Cell.current.special |= Cell.Special.Field;

if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true);

Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);
GUIStyle cstyle = style!=null ? style : UI.current.styles.field;

if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();

int newVal = val;
if (!Cell.current.inactive)
 newVal = (int)EditorGUI.FloatField(rect, val, style:cstyle);
 //Generates some garbage (1-2kB, check with profile), but there's nothing I can do about it
else
 EditorGUI.LabelField(rect, val.ToString(), style:cstyle);

if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();

//assigning new value only on enter, tab or mouseclick (delayed mode)
if (!newVal.Equals(val))
{
 UI.current.delayedInt = newVal;
 UI.current.delayedCell = Cell.current;
```

```
 }

 if (UI.current.delayedCell == Cell.current  &&  (UI.FieldLostFocus || UI.current.editorWindow != EditorWi
 {
  UI.current.MarkChanged();
  val = UI.current.delayedInt;

  if (UI.current.editorWindow!=null) UI.current.editorWindow.Repaint();

  UI.current.delayedCell = null;
 }

 return val;
}

public static int Field (int val, string label, int min=int.MinValue, int max=int.MaxValue)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 Cell.current.special |= Cell.Special.LabelField;

 //using (Cell.RowRel(1-Cell.current.fieldWidth))
 Cell cell = Cell.RowRel(1-Cell.current.fieldWidth);
 {
  Label(label);
```

```
  val = DragValue(val);
 // if (val>max) val=max; //checking after drag since it going to draw field with changed value
 // if (val<min) val=min;
 }
 cell.Dispose();

 //using (Cell.RowRel(Cell.current.fieldWidth))
 cell = Cell.RowRel(Cell.current.fieldWidth);
  val = Field(val);
 cell.Dispose();

// if (val>max) val=max;
// if (val<min) val=min;

 return val;
}

public static void Field (ref int val) { val = Field(val); }
public static void Field (ref int val, string label) { val = Field(val, label); }

#endregion


#region Other Fields

//simple single field
```

```
public static T Field<T> (T val, Func<Rect,T,T> drawFn) //where T: IEquatable<T>
{
 if (UI.current.layout) return val;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return val;

 Cell.current.special |= Cell.Special.Field;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);

 if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true);

 T newVal = drawFn(rect, val);

 if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();

 if (
  (newVal==null && val!=null) ||
  (newVal!=null && val==null) ||
  (newVal!=null && val!=null && !newVal.Equals(val)) )
 {
  UI.current.MarkChanged();
  val = newVal;
 }

 return val;
}
```

```
//complex field label+val
public static T Field<T> (T val, string label, Func<Rect,T,T> drawFn, Func<T,T> dragFn=null) //where T:
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 Cell.current.special |= Cell.Special.LabelField;

 //using (Cell.RowRel(1-Cell.current.fieldWidth))
 Cell labelCell = Cell.RowRel(1-Cell.current.fieldWidth);
 {
  Label(label);
  if (dragFn != null) val = dragFn(val);
 }
 labelCell.Dispose();

 //using (Cell.RowRel(Cell.current.fieldWidth))
 Cell fieldCell = Cell.RowRel(Cell.current.fieldWidth); //won't work with texture/object fields
 val = Field(val, drawFn);
 fieldCell.Dispose();

 return val;
}

public static double Field (double val) { return Field(val, doubleFieldFn); }
```

```
public static void Field (ref double val) { val = Field(val, doubleFieldFn); }
public static double Field (double val, string label) { return Field(val, label, doubleFieldFn, doubleDragFn)
public static void Field (ref double val, string label) { val = Field(val, label, doubleFieldFn, doubleDragFn);
private static double DoubleFieldFn (Rect rect, double val) { return EditorGUI.DoubleField(rect, val, style:
private static readonly Func<Rect,double,double> doubleFieldFn = (rect,val) => EditorGUI.DoubleField(r
private static readonly Func<double,double> doubleDragFn = val => (double)DragValueInternal((float)val



public static string Field (string val) { return Field(val, stringFieldFn); }
public static void Field (ref string val) { val = Field(val, stringFieldFn); }
public static string Field (string val, string label) { return Field(val, label, stringFieldFn); }
public static void Field (ref string val, string label) { val = Field(val, label, stringFieldFn); }
 //could use return Field(val, EditorGUI.ColorField), but it creates new object each time, and therefore too
private static readonly Func<Rect,string,string> stringFieldFn = (rect,val) => EditorGUI.TextField(rect, val



//enums
public static Enum Field (Enum val, bool flags=false) { return Field(val, flags ? flagsFieldFn : enumFieldF
public static void Field (ref Enum val, bool flags=false) { val = Field(val, flags ? flagsFieldFn : enumFieldF
public static Enum Field (Enum val, string label, bool flags=false) { return Field(val, label, flags ? flagsFie
public static void Field (ref Enum val, string label, bool flags=false) { val = Field(val, label, flags ? flagsFie
private static Enum EnumFieldFn (Rect rect, Enum val)
{
 Enum result = EditorGUI.EnumPopup(rect, val, UI.current.styles.enumClose);
 Vector2 signPos = Cell.current.InternalCenter; signPos.x += Cell.current.finalSize.x/2 - 10; signPos.y-=1
 Icon(StylesCache.enumSign, signPos);
```

```
 return result;
}
private static readonly Func<Rect,Enum,Enum> enumFieldFn = EnumFieldFn;

private static Enum FlagsFieldFn (Rect rect, Enum val)
{
 Enum result = EditorGUI.EnumFlagsField(rect, val, UI.current.styles.enumClose);
 Vector2 signPos = Cell.current.InternalCenter; signPos.x += Cell.current.finalSize.x/2 - 10; signPos.y-=1
 Icon(StylesCache.enumSign, signPos);
 return result;
}
private static readonly Func<Rect,Enum,Enum> flagsFieldFn = FlagsFieldFn;


//generic enums
public static T Field<T> (T val, bool flags=false) where T:Enum { return (T)Field((Enum)val,flags); }
public static void Field<T> (ref T val, bool flags=false) where T:Enum { val = (T)Field((Enum)val,flags); }
public static T Field<T> (T val, string label, bool flags=false) where T:Enum { return (T)Field((Enum)val, la
public static void Field<T> (ref T val, string label, bool flags=false) where T:Enum { val = (T)Field((Enum)

public static bool Toggle (bool val) { return Field(val, toggleFieldFn); }
public static void Toggle (ref bool val) { val = Field(val, toggleFieldFn); }
public static bool Toggle (bool val, string label) { return Field(val, label, toggleFieldFn); }
public static void Toggle (ref bool val, string label) { val = Field(val, label, toggleFieldFn); }
private static readonly Func<Rect,bool,bool> toggleFieldFn = (rect,val) =>
{
```

```
 float maxSize = 16;
 if (UI.current.scrollZoom!=null) maxSize *= UI.current.scrollZoom.zoom;
 if (rect.width > maxSize) rect.width = maxSize;
 if (rect.height > maxSize) rect.height = maxSize;
 return EditorGUI.Toggle(rect, val, UI.current.styles.checkbox);
};


public static bool ToggleLeft (bool val, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 using (Cell.RowPx(18)) val = Field(val, toggleFieldFn);
 using (Cell.Row) Label(label);

 return val;
}

public static void ToggleLeft (ref bool val, string label) { val = ToggleLeft(val, label); }

public static void DualLabel (string label, string field) { Field(field, label, dualLabelFieldFn, null); }
private static string LabelFn (Rect rect, string label) { EditorGUI.LabelField(rect, label, style:UI.current.sty
private static readonly Func<Rect,string,string> dualLabelFieldFn = (rect,label) =>
 { EditorGUI.LabelField(rect, label, style:UI.current.styles.label); return label; };
```

```
public static void IconLabel (string label, Texture2D icon)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return; }

 using (Cell.RowPx(18)) Icon(icon);
 using (Cell.Row) Label(label);
}

public static T UniversalField<T> (T val) { return (T)UniversalField(val, typeof(T)); }

public static object UniversalField(object val, Type type)
{
 if (type == typeof(int)) return Field((int)val);
 else if (type == typeof(float)) return Field((float)val);
 else if (type == typeof(bool)) return Toggle((bool)val);
 else if (typeof(Enum).IsAssignableFrom(type)) return Field((Enum)val);
 else if (type == typeof(string)) return Field((string)val);
 else if (type == typeof(Color)) return Field((Color)val);
 else if (type == typeof(double)) return Field((double)val);
 else if (type == typeof(Vector2)) return Field((Vector2)val);
 else if (type == typeof(Vector3)) return Field((Vector3)val);
 else if (type == typeof(Vector4)) return Field((Vector4)val);
 else if (type == typeof(Vector2D)) return Field((Vector2D)val);
 else if (type == typeof(Rect)) return Field((Rect)val);
 else if (type == typeof(Coord)) return Field((Coord)val);
```

```
 else if (type == typeof(CoordRect)) return Field((CoordRect)val);
 else if (type == typeof(Coord3D)) return Field((Coord3D)val);
 else if (type == typeof(Texture2D)) return Field((Texture2D)val, true);
 else if (type == typeof(Transform)) return Field((Transform)val, true);
 else if (type == typeof(GameObject)) return Field((Transform)val, true);
 else if (type == typeof(Material)) return Field((Material)val, true);
 else if (typeof(UnityEngine.Object).IsAssignableFrom(type)) return ObjectField((UnityEngine.Object)val,

 return val;
}

public static object UniversalField (object val, Type type, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 if (type == typeof(int)) return Field((int)val, label);
 else if (type == typeof(float)) return Field((float)val, label);
 else if (type == typeof(bool)) return Toggle((bool)val, label);
 else if (typeof(Enum).IsAssignableFrom(type)) return Field((Enum)val, label);
 else if (type == typeof(string)) return Field((string)val, label);
 else if (type == typeof(Color)) return Field((Color)val, label);
 else if (type == typeof(double)) return Field((double)val, label);
 else if (type == typeof(Vector2)) return Field((Vector2)val, label);
 else if (type == typeof(Vector3)) return Field((Vector3)val, label);
 else if (type == typeof(Vector4)) return Field((Vector4)val, label);
```

```
 else if (type == typeof(Vector2D)) return Field((Vector2D)val, label);
 else if (type == typeof(Rect)) return Field((Rect)val, label);
 else if (type == typeof(Coord)) return Field((Coord)val, label);
 else if (type == typeof(CoordRect)) return Field((CoordRect)val, label);
 else if (type == typeof(Coord3D)) return Field((Coord3D)val, label);
 else if (type == typeof(Texture2D)) return Field((Texture2D)val, label, true);
 else if (type == typeof(Transform)) return Field((Transform)val, label, true);
 else if (type == typeof(GameObject)) return Field((Transform)val, label, true);
 else if (type == typeof(Material)) return Field((Material)val, label, true);
 else if (type == typeof(TerrainLayer)) return Field((TerrainLayer)val, label, true);
 else if (typeof(UnityEngine.Object).IsAssignableFrom(type)) return ObjectField((UnityEngine.Object)val,

 return val;
}

public static T UniversalField<T> (T val, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 Type type = typeof(T);

 using (Cell.RowRel(1-Cell.current.fieldWidth))
 {
  Label(label);
  if (type == typeof(int)) val = (T)(object)DragValue((int)(object)val);
```

```
  if (type == typeof(float)) val = (T)(object)DragValue((float)(object)val);
 }

 using (Cell.RowRel(Cell.current.fieldWidth))
  val = (T)UniversalField(val, type);

 return val;
}

#endregion



#region Non-generic Object Fields

public static UnityEngine.Object Field (UnityEngine.Object val, Type type, bool allowSceneObject)
{
 if (UI.current.layout) return val;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return val;
 if (StylesCache.objectPickerTex == null) return val; //happens on build

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);

 if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true);

 Rect shrinkedRect = new Rect(rect.x, rect.y+1, rect.width, rect.height-2);
 UnityEngine.Object newVal = EditorGUI.ObjectField(shrinkedRect, val, objType:type, allowSceneObject
```

```
if (StylesCache.isPro)
 EditorGUI.DrawRect(rect, new Color(0.22f, 0.22f, 0.22f));

if (Event.current.type == EventType.Repaint)
 UI.current.styles.field.Draw(rect, false, false, false, false);

string name = "None";
if (newVal != null) name = newVal.name.ToString();
name += " (" + type.Name + ")";

Rect labelRect = new Rect(rect.x+2, rect.y, rect.width-22, rect.height);
EditorGUI.LabelField(labelRect, name, UI.current.styles.label);

float zoom = UI.current.scrollZoom!=null ? UI.current.scrollZoom.zoom : 1;
Rect pickerRect = new Rect(rect.x+rect.width - 12*zoom, rect.y+rect.height/2-4*zoom, 8*zoom, 8*zoom)
UnityEngine.GUI.DrawTexture(pickerRect, StylesCache.objectPickerTex, ScaleMode.ScaleAndCrop);

if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();

if (
 (newVal==null && val!=null) ||
 (newVal!=null && val==null) ||
 (newVal!=null && val!=null && !newVal.Equals(val)) )
{
 UI.current.MarkChanged();
```

```
  val = newVal;
 }

 return val;
}

public static UnityEngine.Object Field (UnityEngine.Object val, string label, Type type, bool allowSceneO
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 using (Cell.RowRel(1-Cell.current.fieldWidth))
 //Cell labelCell = Cell.RowRel(1-Cell.current.fieldWidth);
  Label(label);
 //labelCell.Dispose();

 using (Cell.RowRel(Cell.current.fieldWidth))
 //Cell fieldCell = Cell.RowRel(Cell.current.fieldWidth); //won't work with texture/object fields
 val = Field(val, type, allowSceneObject);
 //fieldCell.Dispose();

 return val;
}

public static void Field (ref UnityEngine.Object val, Type type, bool allowSceneObject)
 { val = Field(val, type, allowSceneObject); }
```

```
 public static void Field (ref UnityEngine.Object val, string label, Type type, bool allowSceneObject)
  { val = Field(val, label, type, allowSceneObject); }

#endregion

#region Generic Object Fields

 public static T Field<T> (T val, Func<Rect,T,bool,T> drawFn, bool allowSceneObject) where T: UnityEng
 {
  if (UI.current.layout) return val;
  if (UI.current.optimizeElements && !UI.current.IsInWindow()) return val;

  Cell.current.special |= Cell.Special.Field;

  Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);

  if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true);
  T newVal = drawFn(rect, val, allowSceneObject);
  if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();

  if (
   (newVal==null && val!=null) ||
   (newVal!=null && val==null) ||
   (newVal!=null && val!=null && !newVal.Equals(val)) )
  {
```

```
  UI.current.MarkChanged();
  val = newVal;
 }

 return val;
}

public static T Field<T> (T val, string label, Func<Rect,T,bool,T> drawFn, bool allowSceneObject) where
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 Cell.current.special |= Cell.Special.LabelField;

 using (Cell.RowRel(1-Cell.current.fieldWidth))
 //Cell labelCell = Cell.RowRel(1-Cell.current.fieldWidth);
  Label(label);
 //labelCell.Dispose();

 using (Cell.RowRel(Cell.current.fieldWidth))
 //Cell fieldCell = Cell.RowRel(Cell.current.fieldWidth); //won't work with texture/object fields
 val = Field(val, drawFn, allowSceneObject);
 //fieldCell.Dispose();

 return val;
}
```

```
//known object fields
public static Texture2D Field (Texture2D val, bool allowSceneObject=false) { return Field(val, textureFiel
public static void Field (ref Texture2D val, bool allowSceneObject=false) { val = Field(val, textureFieldFn,
public static Texture2D Field (Texture2D val, string label, bool allowSceneObject=false) { return Field(val
public static void Field (ref Texture2D val, string label, bool allowSceneObject=false) { val = Field(val, lab
private static readonly Func<Rect,Texture2D,bool,Texture2D> textureFieldFn = ObjectFieldFn;

public static TerrainLayer Field (TerrainLayer val, bool allowSceneObject=false) { return Field(val, terrain
public static void Field (ref TerrainLayer val, bool allowSceneObject=false) { val = Field(val, terrainLayerF
public static TerrainLayer Field (TerrainLayer val, string label, bool allowSceneObject=false) { return Fiel
public static void Field (ref TerrainLayer val, string label, bool allowSceneObject=false) { val = Field(val, l
private static readonly Func<Rect,TerrainLayer,bool,TerrainLayer> terrainLayerFieldFn = ObjectFieldFn;

public static Material Field (Material val, bool allowSceneObject=false) { return Field(val, materialFieldFn,
public static void Field (ref Material val, bool allowSceneObject=false) { val = Field(val, materialFieldFn, a
public static Material Field (Material val, string label, bool allowSceneObject=false) { return Field(val, labe
public static void Field (ref Material val, string label, bool allowSceneObject=false) { val = Field(val, label,
private static readonly Func<Rect,Material,bool,Material> materialFieldFn = ObjectFieldFn;

public static Transform Field (Transform val, bool allowSceneObject=false) { return Field(val, transformFi
public static void Field (ref Transform val, bool allowSceneObject=false) { val = Field(val, transformFieldF
public static Transform Field (Transform val, string label, bool allowSceneObject=false) { return Field(val,
public static void Field (ref Transform val, string label, bool allowSceneObject=false) { val = Field(val, labe
```

```
private static readonly Func<Rect,Transform,bool,Transform> transformFieldFn = ObjectFieldFn;

public static GameObject Field (GameObject val, bool allowSceneObject=false) { return Field(val, gameO
public static void Field (ref GameObject val, bool allowSceneObject=false) { val = Field(val, gameObjectF
public static GameObject Field (GameObject val, string label, bool allowSceneObject=false) { return Field
public static void Field (ref GameObject val, string label, bool allowSceneObject=false) { val = Field(val, la
private static readonly Func<Rect,GameObject,bool,GameObject> gameObjectFieldFn = ObjectFieldFn;

//unknown object fields
public static T ObjectField<T> (T val, bool allowSceneObject=false) where T: UnityEngine.Object
 { return Field(val, ObjectFieldFn, allowSceneObject); }
public static void ObjectField<T> (ref T val, bool allowSceneObject=false) where T: UnityEngine.Object
 { val = Field(val, ObjectFieldFn, allowSceneObject); }
public static T ObjectField<T> (T val, string label, bool allowSceneObject=false) where T: UnityEngine.Ob
 { return Field(val, label, ObjectFieldFn, allowSceneObject); }
public static void ObjectField<T> (ref T val, string label, bool allowSceneObject=false) where T: UnityEng
 { val = Field(val, label, ObjectFieldFn, allowSceneObject); }

private static T ObjectFieldFn<T> (Rect rect, T val, bool allowSceneObject) where T: UnityEngine.Object
 (T)ObjectFieldFn(rect, val, typeof(T), allowSceneObject);


public static UnityEngine.Object ObjectField (UnityEngine.Object val, Type type, bool allowSceneObject=
{
 if (UI.current.layout) return val;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return val;
```

```
 Cell.current.special |= Cell.Special.Field;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);

 if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true);
 UnityEngine.Object newVal = ObjectFieldFn(rect, val, type, allowSceneObject);
 if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();

 if (
  (newVal==null && val!=null) ||
  (newVal!=null && val==null) ||
  (newVal!=null && val!=null && !newVal.Equals(val)) )
 {
  UI.current.MarkChanged();
  val = newVal;
 }

 return val;
}

public static UnityEngine.Object ObjectField (UnityEngine.Object val, string label, Type type, bool allowS
{
 if (UI.current.layout) return val;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return val;
```

```
 using (Cell.RowRel(1-Cell.current.fieldWidth)) Label(label);
 using (Cell.RowRel(Cell.current.fieldWidth)) val = ObjectField(val, type, allowSceneObject);

 return val;
}

private static UnityEngine.Object ObjectFieldFn (Rect rect, UnityEngine.Object val, Type type, bool allow
{
 Rect shrinkedRect = new Rect(rect.x, rect.y+1, rect.width, rect.height-2);
 UnityEngine.Object obj = EditorGUI.ObjectField(shrinkedRect, val, objType:type, allowSceneObjects:allo

 if (StylesCache.isPro)
  EditorGUI.DrawRect(rect, new Color(0.22f, 0.22f, 0.22f));

 if (Event.current.type == EventType.Repaint)
  UI.current.styles.field.Draw(rect, false, false, false, false);

 string name = "None";
 if (obj != null) name = obj.name.ToString();
 name += " (" + type.Name + ")";

 Rect labelRect = new Rect(rect.x+2, rect.y, rect.width-22, rect.height);
 EditorGUI.LabelField(labelRect, name, UI.current.styles.label);

 float zoom = UI.current.scrollZoom!=null ? UI.current.scrollZoom.zoom : 1;
 Rect pickerRect = new Rect(rect.x+rect.width - 12*zoom, rect.y+rect.height/2-4*zoom, 8*zoom, 8*zoom)
```

```
 UnityEngine.GUI.DrawTexture(pickerRect, StylesCache.objectPickerTex, ScaleMode.ScaleAndCrop);

 return obj;
}

#endregion


#region Color Fields

 public static Color Field (Color color, bool hdr=false)
 {
  if (UI.current.layout) return color;
  if (UI.current.optimizeElements && !UI.current.IsInWindow()) return color;

  Cell.current.special |= Cell.Special.Field;

  Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);

  if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true);

  //if (multiplyMat == null)
  // multiplyMat = new Material( Shader.Find("Hidden/DPLayout/Multiply") );
  //multiplyMat.SetColor("_Color", color);

  if (UnityEngine.GUI.Button(rect, "", GUIStyle.none))
```

```
{
 Assembly editorAssembly = Assembly.GetAssembly(typeof(EditorWindow));
 Type colorPickerType = editorAssembly.GetType("UnityEditor.ColorPicker");
 MethodInfo[] methods = colorPickerType.GetMethods(BindingFlags.Public | BindingFlags.Static);
 MethodInfo showMethod = methods.Single( m =>
 {
  if (m.Name != "Show") return false;
  ParameterInfo[] parameters = m.GetParameters();
  if (parameters.Length == 4 && parameters[0].ParameterType == typeof(Action<Color>)) return true;
  else return false;
 } );

 Cell currentCell = Cell.current;
 EditorWindow baseWindow = UI.current.editorWindow; //EditorWindow.focusedWindow;
 Action<Color> colorChangedCallback = c =>
 {
  colorChangeCell = currentCell;
  colorChangeColor = c;
  baseWindow.Repaint();
 };
 showMethod.Invoke(null, new object[] {colorChangedCallback, color, true, hdr});
}

if (Event.current.type == EventType.Repaint)
 UI.current.styles.field.Draw(rect, false, false, false, false);
```

```
 if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();

 if (colorChangeCell==Cell.current)
 {
  colorChangeCell = null;
  color = colorChangeColor;
  UI.current.MarkChanged();
 }

 //drawing color itself after it has been changed
 Rect colorRect = rect.Extended(-1);
 Rect rgbRect = colorRect;
 rgbRect.width -= 10;
 Rect aRect = colorRect;
 aRect.width = 10;
 aRect.x = rgbRect.position.x+rgbRect.width;

 EditorGUI.DrawRect(rgbRect, new Color(color.r, color.g, color.b));
 EditorGUI.DrawRect(aRect, new Color(color.a, color.a, color.a));

 return color;
}

public static Cell colorChangeCell; //null if color have not changed
public static Color colorChangeColor;
```

```
public static Color Field (Color val, string label) //where T: IEquatable<T>
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 using (Cell.RowRel(1-Cell.current.fieldWidth))
 {
  Label(label);

  float sum = val.r + val.g + val.b;
  float newSum = DragValueInternal(sum, 0.005f, exponentiality:1000, sensitivity:50000);
  if (newSum<0) newSum = 0;
  if (newSum>5) newSum = 5; //maximum value when color isn't white
  if (newSum>sum+0.00001f || newSum<sum-0.00001f)
  {
   val.r = val.r/sum * newSum;
   val.g = val.g/sum * newSum;
   val.b = val.b/sum * newSum;

   UI.current.MarkChanged();
  }
 }

 using (Cell.RowRel(Cell.current.fieldWidth))
  val = Field(val);
```

```
 return val;
}

public static void Field (ref Color val) { val = Field(val); }
public static void Field (ref Color val, string label) { val = Field(val, label); }

#endregion



#region Vectors

public static Vector2 Field (Vector2 val)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 Cell.current.special |= Cell.Special.Vector;

 Cell.current.fieldWidth = 0.8f;
 using (Cell.LineStd) { val.x = Field(val.x, "X"); Cell.current.special |= Cell.Special.VectorX; } //after field si
 using (Cell.LineStd) { val.y = Field(val.y, "Y"); Cell.current.special |= Cell.Special.VectorY; }

 return val;
}
```

```
public static Vector2 Field (Vector2 val, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }
 Cell.current.special |= Cell.Special.Vector;

 using (Cell.RowRel(1-Cell.current.fieldWidth))
 {
  using (Cell.Row) Label(label);
  using (Cell.RowPx(vectorXYWidth))
  {
   using (Cell.LineStd) { Label("X"); val.x = DragValue(val.x); Cell.current.special |= Cell.Special.VectorX;
   using (Cell.LineStd) { Label("Y"); val.y = DragValue(val.y); Cell.current.special |= Cell.Special.VectorY;
  }
 }
 using (Cell.RowRel(Cell.current.fieldWidth))
 {
  using (Cell.LineStd) { val.x = Field(val.x); Cell.current.special |= Cell.Special.VectorX; }
  using (Cell.LineStd) { val.y = Field(val.y); Cell.current.special |= Cell.Special.VectorY; }
 }

 return val;
}

public static void Field (ref Vector2 val) { val = Field(val); }
public static void Field (ref Vector2 val, string label) { val = Field(val, label); }
```

```
public static void Field (ref Vector2 val, string label, string xName, string yName, int xyWidth=15) { val = F
public static Vector2 Field (Vector2 val, string label, string xName, string yName, int xyWidth=15)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }
 Cell.current.special |= Cell.Special.Vector;

 using (Cell.RowRel(1-Cell.current.fieldWidth))
 {
  using (Cell.Row) Label(label);
  using (Cell.RowPx(xyWidth))
  {
   using (Cell.LineStd) { Label(xName); val.x = DragValue(val.x); Cell.current.special |= Cell.Special.Vecto
   using (Cell.LineStd) { Label(yName); val.y = DragValue(val.y); Cell.current.special |= Cell.Special.Vecto
  }
 }
 using (Cell.RowRel(Cell.current.fieldWidth))
 {
  using (Cell.LineStd) { val.x = Field(val.x); Cell.current.special |= Cell.Special.VectorX; }
  using (Cell.LineStd) { val.y = Field(val.y); Cell.current.special |= Cell.Special.VectorY; }
 }

 return val;
}
```

```
public static Vector3 Field (Vector3 val)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }
 Cell.current.special |= Cell.Special.Vector;

 Cell.current.fieldWidth = 0.8f;
 using (Cell.LineStd) { val.x = Field(val.x, "X"); Cell.current.special |= Cell.Special.VectorX; }
 using (Cell.LineStd) { val.y = Field(val.y, "Y"); Cell.current.special |= Cell.Special.VectorY; }
 using (Cell.LineStd) { val.z = Field(val.z, "Z"); Cell.current.special |= Cell.Special.VectorZ; }

 return val;
}

public static Vector3 Field (Vector3 val, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }
 Cell.current.special |= Cell.Special.Vector;

 using (Cell.RowRel(1-Cell.current.fieldWidth))
 {
  using (Cell.Row) Label(label);
  using (Cell.RowPx(vectorXYWidth))
  {
```

```
   using (Cell.LineStd) { Label("X"); val.x = DragValue(val.x); Cell.current.special |= Cell.Special.VectorX;
   using (Cell.LineStd) { Label("Y"); val.y = DragValue(val.y); Cell.current.special |= Cell.Special.VectorY;
   using (Cell.LineStd) { Label("Z"); val.z = DragValue(val.z); Cell.current.special |= Cell.Special.VectorZ;
  }
 }
 using (Cell.RowRel(Cell.current.fieldWidth))
 {
  using (Cell.LineStd) { val.x = Field(val.x); Cell.current.special |= Cell.Special.VectorX; }
  using (Cell.LineStd) { val.y = Field(val.y); Cell.current.special |= Cell.Special.VectorY; }
  using (Cell.LineStd) { val.z = Field(val.z); Cell.current.special |= Cell.Special.VectorZ; }
 }

 return val;
}

public static void Field (ref Vector3 val) { val = Field(val); }
public static void Field (ref Vector3 val, string label) { val = Field(val, label); }



public static Vector4 Field (Vector4 val)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }
 Cell.current.special |= Cell.Special.Vector;

 Cell.current.fieldWidth = 0.8f;
```

```
 using (Cell.LineStd) { val.x = Field(val.x, "X"); Cell.current.special |= Cell.Special.VectorX; }
 using (Cell.LineStd) { val.y = Field(val.y, "Y"); Cell.current.special |= Cell.Special.VectorY; }
 using (Cell.LineStd) { val.z = Field(val.z, "Z"); Cell.current.special |= Cell.Special.VectorZ; }
 using (Cell.LineStd) { val.w = Field(val.w, "W"); Cell.current.special |= Cell.Special.VectorW; }

 return val;
}

public static Vector4 Field (Vector4 val, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }
 Cell.current.special |= Cell.Special.Vector;

 using (Cell.RowRel(1-Cell.current.fieldWidth))
 {
  using (Cell.Row) Label(label);
  using (Cell.RowPx(vectorXYWidth))
  {
   using (Cell.LineStd) { Label("X"); val.x = DragValue(val.x); Cell.current.special |= Cell.Special.VectorX;
   using (Cell.LineStd) { Label("Y"); val.y = DragValue(val.y); Cell.current.special |= Cell.Special.VectorY;
   using (Cell.LineStd) { Label("Z"); val.z = DragValue(val.z); Cell.current.special |= Cell.Special.VectorZ;
   using (Cell.LineStd) { Label("W"); val.w = DragValue(val.w); Cell.current.special |= Cell.Special.VectorW
  }
 }
 using (Cell.RowRel(Cell.current.fieldWidth))
```

```
 {
  using (Cell.LineStd) { val.x = Field(val.x); Cell.current.special |= Cell.Special.VectorX; }
  using (Cell.LineStd) { val.y = Field(val.y); Cell.current.special |= Cell.Special.VectorY; }
  using (Cell.LineStd) { val.z = Field(val.z); Cell.current.special |= Cell.Special.VectorZ; }
  using (Cell.LineStd) { val.w = Field(val.w); Cell.current.special |= Cell.Special.VectorW; }
 }

 return val;
}

public static void Field (ref Vector4 val) { val = Field(val); }
public static void Field (ref Vector4 val, string label) { val = Field(val, label); }



public static Vector2D Field (Vector2D val)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }
 Cell.current.special |= Cell.Special.Vector;

 Cell.current.fieldWidth = 0.8f;
 using (Cell.LineStd) { val.x = Field(val.x, "X"); Cell.current.special |= Cell.Special.VectorX; }
 using (Cell.LineStd) { val.z = Field(val.z, "Z"); Cell.current.special |= Cell.Special.VectorZ; }

 return val;
}
```

```
public static Vector2D Field (Vector2D val, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }
 Cell.current.special |= Cell.Special.Vector;

 using (Cell.RowRel(1-Cell.current.fieldWidth))
 {
  using (Cell.Row) Label(label);
  using (Cell.RowPx(vectorXYWidth))
  {
   using (Cell.LineStd) { Label("X"); val.x = DragValue(val.x); Cell.current.special |= Cell.Special.VectorX;
   using (Cell.LineStd) { Label("Z"); val.z = DragValue(val.z); Cell.current.special |= Cell.Special.VectorZ;
  }
 }
 using (Cell.RowRel(Cell.current.fieldWidth))
 {
  using (Cell.LineStd) { val.x = Field(val.x); Cell.current.special |= Cell.Special.VectorX; }
  using (Cell.LineStd) { val.z = Field(val.z); Cell.current.special |= Cell.Special.VectorZ; }
 }

 return val;
}

public static void Field (ref Vector2D val) { val = Field(val); }
```

```
public static void Field (ref Vector2D val, string label) { val = Field(val, label); }



public static Rect Field (Rect val)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 Cell.current.fieldWidth = 0.7f;
 using (Cell.LineStd) val.x = Field(val.x, "Pos X");
 using (Cell.LineStd) val.y = Field(val.y, "Pos Y");
 using (Cell.LineStd) val.width = Field(val.width, "Width");
 using (Cell.LineStd) val.height = Field(val.height, "Height");

 return val;
}

public static Rect Field (Rect val, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 using (Cell.RowRel(1-Cell.current.fieldWidth))
 {
  using (Cell.Row) Label(label);
  using (Cell.RowPx(vectorWidthHeightWidth))
```

```
  {
   using (Cell.LineStd) { Label("Pos X"); val.x = DragValue(val.x); }
   using (Cell.LineStd) { Label("Pos Y"); val.y = DragValue(val.y); }
   using (Cell.LineStd) { Label("Width"); val.width = DragValue(val.width); }
   using (Cell.LineStd) { Label("Height"); val.height = DragValue(val.height); }
  }
 }
 using (Cell.RowRel(Cell.current.fieldWidth))
 {
  using (Cell.LineStd) val.x = Field(val.x);
  using (Cell.LineStd) val.y = Field(val.y);
  using (Cell.LineStd) val.width = Field(val.width);
  using (Cell.LineStd) val.height = Field(val.height);
 }

 return val;
}

public static void Field (ref Rect val) { val = Field(val); }
public static void Field (ref Rect val, string label) { val = Field(val, label); }


public static Coord Field (Coord val)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }
```

```
 Cell.current.special |= Cell.Special.Vector;

 Cell.current.fieldWidth = 0.8f;
 using (Cell.LineStd) val.x = Field(val.x, "X");
 using (Cell.LineStd) val.z = Field(val.z, "Z");

 return val;
}

public static Coord Field (Coord val, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }
 Cell.current.special |= Cell.Special.Vector;

 using (Cell.RowRel(1-Cell.current.fieldWidth))
 {
  using (Cell.Row) Label(label);
  using (Cell.RowPx(vectorXYWidth))
  {
   using (Cell.LineStd) { Label("X"); val.x = DragValue(val.x); Cell.current.special |= Cell.Special.VectorX;
   using (Cell.LineStd) { Label("Z"); val.z = DragValue(val.z); Cell.current.special |= Cell.Special.VectorZ;
  }
 }
 using (Cell.RowRel(Cell.current.fieldWidth))
 {
```

```
  using (Cell.LineStd) { val.x = Field(val.x); Cell.current.special |= Cell.Special.VectorX; }
  using (Cell.LineStd) { val.z = Field(val.z); Cell.current.special |= Cell.Special.VectorZ; }
 }

 return val;
}

public static void Field (ref Coord val) { val = Field(val); }
public static void Field (ref Coord val, string label) { val = Field(val, label); }


public static CoordRect Field (CoordRect val)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 Cell.current.fieldWidth = 0.7f;
 using (Cell.LineStd) val.offset.x = Field(val.offset.x, "Pos X");
 using (Cell.LineStd) val.offset.z = Field(val.offset.z, "Pos Z");
 using (Cell.LineStd) val.size.x = Field(val.size.x, "Size X");
 using (Cell.LineStd) val.size.z = Field(val.size.z, "Size Z");

 return val;
}

public static CoordRect Field (CoordRect val, string label)
```

```
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 using (Cell.RowRel(1-Cell.current.fieldWidth))
 {
  using (Cell.Row) Label(label);
  using (Cell.RowPx(vectorWidthHeightWidth))
  {
   using (Cell.LineStd) { Label("Pos X"); val.offset.x = DragValue(val.offset.x); }
   using (Cell.LineStd) { Label("Pos Z"); val.offset.z = DragValue(val.offset.z); }
   using (Cell.LineStd) { Label("Size X"); val.size.x = DragValue(val.size.x); }
   using (Cell.LineStd) { Label("Size Z"); val.size.z = DragValue(val.size.z); }
  }
 }
 using (Cell.RowRel(Cell.current.fieldWidth))
 {
  using (Cell.LineStd) val.offset.x = Field(val.offset.x);
  using (Cell.LineStd) val.offset.z = Field(val.offset.z);
  using (Cell.LineStd) val.size.x = Field(val.size.x);
  using (Cell.LineStd) val.size.z = Field(val.size.z);
 }

 return val;
}
```

```
public static void Field (ref CoordRect val) { val = Field(val); }
public static void Field (ref CoordRect val, string label) { val = Field(val, label); }



public static Coord3D Field (Coord3D val)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }
 Cell.current.special |= Cell.Special.Vector;

 Cell.current.fieldWidth = 0.8f;
 using (Cell.LineStd) { val.x = Field(val.x, "X");  Cell.current.special |= Cell.Special.VectorX; }
 using (Cell.LineStd) { val.y = Field(val.y, "Y");  Cell.current.special |= Cell.Special.VectorX; }
 using (Cell.LineStd) { val.z = Field(val.z, "Z");  Cell.current.special |= Cell.Special.VectorX; }

 return val;
}

public static Coord3D Field (Coord3D val, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }
 Cell.current.special |= Cell.Special.Vector;

 using (Cell.RowRel(1-Cell.current.fieldWidth))
```

```
 {
  using (Cell.Row) Label(label);
  using (Cell.RowPx(vectorXYWidth))
  {
   using (Cell.LineStd) { Label("X"); val.x = DragValue(val.x); Cell.current.special |= Cell.Special.VectorX;
   using (Cell.LineStd) { Label("Y"); val.y = DragValue(val.y); Cell.current.special |= Cell.Special.VectorY;
   using (Cell.LineStd) { Label("Z"); val.z = DragValue(val.z); Cell.current.special |= Cell.Special.VectorZ;
  }
 }
 using (Cell.RowRel(Cell.current.fieldWidth))
 {
  using (Cell.LineStd) { val.x = Field(val.x); Cell.current.special |= Cell.Special.VectorX; }
  using (Cell.LineStd) { val.y = Field(val.y); Cell.current.special |= Cell.Special.VectorY; }
  using (Cell.LineStd) { val.z = Field(val.z); Cell.current.special |= Cell.Special.VectorZ; }
 }

 return val;
}

#endregion


#region Class

 public static bool Class (object obj, string category=null, Action<FieldInfo,Cell> additionalAction=null)
 /// Draws all values of the class marked with Val attribute (and category)
```

```
/// if additionalAction defined performs it for each fo the fields
/// Returns true if anything has been drawn, false if empty
{
 Type type = obj.GetType();

 ValAttribute[] attributes = null;
 attributes = GetCachedVals(type);

 for (int a=0; a<attributes.Length; a++)
 {
  ValAttribute att = attributes[a];

  if (att.field == null) continue; //could be a property
  if (att.cat != category) continue; //null category is a category too. Not drawing all when category is null.

  Cell cell = Cell.LineStd;
  try
  {
   ClassField(att, obj);
   additionalAction?.Invoke(att.field, cell);
  }
  finally { cell.Dispose(); }
 }

 return attributes.Length != 0;
}
```

```
public static void ClassField (ValAttribute att, object obj)
{
 //wrapping GetValue to delegate to avoid boxing/unboxing objects (creates too much garbage)
 //not using dynamic since it's compiled to object anyways

 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return; }

 if (att.type == typeof(float))
 {
  if (UI.current.layout) EmptyField();
  else
  {
   float srcVal = Getter<float>.GetGetter(att.field)(obj);
   float dstVal = Field(srcVal, att.name, att.min, att.max);
   if (Cell.current.valChanged)  att.field.SetValue(obj, dstVal);
  }
 }

 else if (att.type == typeof(int))
 {
  if (UI.current.layout) EmptyField();
  else
  {
```

```
  int val = Getter<int>.GetGetter(att.field)(obj);
  val = Field(val, att.name,
    min:att.min<int.MinValue ? int.MinValue : (int)att.min,
    max:att.max>int.MaxValue ? int.MaxValue : (int)att.max); //(int)float.Max = -2000..0
  if (Cell.current.valChanged)  att.field.SetValue(obj, val);
 }
}

else if (att.type == typeof(bool))
{
 bool val = Getter<bool>.GetGetter(att.field)(obj);

 if (!att.isLeft) val = Toggle(val, att.name);
 else val = ToggleLeft(val, att.name);

 if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, val);
}

else if (att.type == typeof(string))
{
 if (UI.current.layout) EmptyField();
 else
 {
  string val = Getter<string>.GetGetter(att.field)(obj);
  val = Field(val ?? default, att.name);
  if (Cell.current.valChanged)  att.field.SetValue(obj, val);
```

```
 }
}

else if (att.type == typeof(Color))
{
 if (UI.current.layout) EmptyField();
 else
 {
  Color val = Getter<Color>.GetGetter(att.field)(obj);
  val = Field((Color)val, att.name);
  if (Cell.current.valChanged)  att.field.SetValue(obj, val);
 }
}

else if (att.type == typeof(double))
{
 if (UI.current.layout) EmptyField();
 else
 {
  double val = Getter<float>.GetGetter(att.field)(obj);
  val = Field(val, att.name);
  if (Cell.current.valChanged)  att.field.SetValue(obj, val);
 }
}

else if (typeof(Enum).IsAssignableFrom(att.type))
```

```
{
 Enum val = (Enum)att.field.GetValue(obj);  //Getter<Enum>.GetGetter(att.field)(obj); Invalid IL code
 val = Field(val, att.name);
 if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, val);
}

else if (att.type == typeof(Vector2))
{
 Vector2 val = Getter<Vector2>.GetGetter(att.field)(obj);
 val = Field(val, att.name);
 if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, val);
}

else if (att.type == typeof(Vector3))
{
 Vector3 val = Getter<Vector3>.GetGetter(att.field)(obj);
 val = Field(val, att.name);
 if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, val);
}

else if (att.type == typeof(Vector4))
{
 Vector4 val = Getter<Vector4>.GetGetter(att.field)(obj);
 val = Field(val, att.name);
 if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, val);
}
```

```
else if (att.type == typeof(Vector2D))
{
 Vector2D val = Getter<Vector2D>.GetGetter(att.field)(obj);
 val = Field(val, att.name);
 if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, val);
}

else if (att.type == typeof(Rect))
{
 Rect val = Getter<Rect>.GetGetter(att.field)(obj);
 val = Field(val, att.name);
 if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, val);
}

else if (att.type == typeof(Coord))
{
 Coord val = Getter<Coord>.GetGetter(att.field)(obj);
 val = Field(val, att.name);
 if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, val);
}

else if (att.type == typeof(CoordRect))
{
 CoordRect val = Getter<CoordRect>.GetGetter(att.field)(obj);
 val = Field(val, att.name);
```

```
 if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, val);
}

else if (att.type == typeof(Coord3D))
{
 Coord3D val = Getter<Coord3D>.GetGetter(att.field)(obj);
 val = Field(val, att.name);
 if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, val);
}

else if (att.type == typeof(Texture2D))
{
 if (UI.current.layout) EmptyField();
 else
 {
  Texture2D val = Getter<Texture2D>.GetGetter(att.field)(obj);
  val = Field(val, att.name);
  if (Cell.current.valChanged)  att.field.SetValue(obj, val);

 }
}

else if (att.type == typeof(Transform))
{
 if (UI.current.layout) EmptyField();
 else
```

```
 {
  Transform val = Getter<Transform>.GetGetter(att.field)(obj);
  val = Field(val ?? default, att.name, allowSceneObject:att.allowSceneObject);
  if (Cell.current.valChanged)  att.field.SetValue(obj, val);
 }
}

else if (att.type == typeof(GameObject))
{
 if (UI.current.layout) EmptyField();
 else
 {
  GameObject val = Getter<GameObject>.GetGetter(att.field)(obj);
  val = Field(val ?? default, att.name, allowSceneObject:att.allowSceneObject);
  if (Cell.current.valChanged)  att.field.SetValue(obj, val);
 }
}

else if (att.type == typeof(UnityEngine.Object) || typeof(UnityEngine.Object).IsAssignableFrom(att.type))
{
 UnityEngine.Object val = Getter<UnityEngine.Object>.GetGetter(att.field)(obj);
 val = Field(val ?? default, att.name, att.type, att.allowSceneObject);
 if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, val);
}

else if (att.type == typeof(AnimationCurve))
```

```
{
 AnimationCurve val = Getter<AnimationCurve>.GetGetter(att.field)(obj);
 using (Cell.Row) Label(att.name);
 using (Cell.Row) AnimationCurve (val);
 if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, val);
}

else if (att.type.IsArray)
{
 bool opened = true;
 Cell.EmptyLinePx(4);
 using (Cell.LineStd)
  using (new Draw.FoldoutGroup(ref opened, att.name, isLeft:true))
   if (opened)
   {
    Type elType = att.type.GetElementType();
    Array arr = (Array)att.field.GetValue(obj);

    for (int i=0; i<arr.Length; i++)
     using (Cell.LineStd)
     {
      if (elType == typeof(float)) arr.SetValue( Field((float)arr.GetValue(i), i.ToString()), i);
     }

    if (Cell.current.valChanged && !UI.current.layout)  att.field.SetValue(obj, arr);
   }
```

```
  Cell.EmptyLinePx(4);
 }

 else if (att.type.IsClass)// || type.IsValueType)
 {
  bool opened = true;
  Cell.EmptyLinePx(4);
  using (Cell.LineStd)
   using (new Draw.FoldoutGroup(ref opened, att.name, isLeft:true))
    if (opened)
    {
     object val =  Getter<object>.GetGetter(att.field)(obj); //field.GetValue(obj);
     Draw.Class(val);
    }
  Cell.EmptyLinePx(4);
 }

 else
 {
  object val = att.field.GetValue(obj);
  if (val != null) DualLabel(att.name, val.ToString());
  else DualLabel(att.name, "null");
 }
}

public static class Getter<T>
```

```
{
 static Dictionary<FieldInfo, Func<object,T>> cache = new Dictionary<FieldInfo, Func<object, T>>(); //ide

 public static Func<object,T> GetGetter (FieldInfo field)
 {
  //return cached if any
  if (cache.TryGetValue(field, out Func<object,T> getter)) return getter;

  //creating new one
  string methodName = field.ReflectedType.FullName + ".get_" + field.Name;
  DynamicMethod setterMethod = new DynamicMethod(methodName, typeof(T), new Type[1] { typeof(ob
  ILGenerator ilgen = setterMethod.GetILGenerator();
  ilgen.Emit(OpCodes.Ldarg_0);
  ilgen.Emit(OpCodes.Ldfld, field);
  ilgen.Emit(OpCodes.Ret); //I have no idea of what I'm doing.jpg
  getter = (Func<object, T>)setterMethod.CreateDelegate(typeof(Func<object, T>));

  cache.Add(field, getter);
  return getter;
 }
}

public static void EmptyField ()
/// Just initializes cells on layout
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
```

```
 { Cell.current.Skip(); return; }

 //Cell cell = Cell.LineStd;

 Cell.EmptyRowRel(1-Cell.current.fieldWidth);
 Cell.EmptyRowRel(Cell.current.fieldWidth);

 //cell.Dispose();
}

public static ValAttribute[] GetCachedVals (Type type)
{
 if (valsCaches.TryGetValue(type, out ValAttribute[] attributes)) return attributes;

 List<ValAttribute> attList = new List<ValAttribute>();

 FieldInfo[] fields = type.GetFields();
 for (int f=0; f<fields.Length; f++)
 {
  ValAttribute valAtt = Attribute.GetCustomAttribute(fields[f], typeof(ValAttribute)) as ValAttribute;
  if (valAtt == null) continue;

  valAtt.field = fields[f];
  valAtt.type = fields[f].FieldType;

  attList.Add(valAtt);
```

```
 }

 PropertyInfo[] props = type.GetProperties();
 for (int p=0; p<props.Length; p++)
 {
  ValAttribute valAtt = Attribute.GetCustomAttribute(props[p], typeof(ValAttribute)) as ValAttribute;
  if (valAtt == null) continue;

  valAtt.prop = props[p];
  valAtt.type = props[p].PropertyType;

  attList.Add(valAtt);
 }

 attributes = attList.ToArray();

 //if (attributes != null)
 // Array.Sort(attributes, (x,y) => 0);//y.priority.CompareTo(x.priority));

 valsCaches.Add(type, attributes);

 return attributes;
}

private static readonly Dictionary<Type, ValAttribute[]> valsCaches = new Dictionary<Type, ValAttribute[
//don't store attributes cache in runtime code
```

```
#endregion

#region Fields Caches

 public static void AddFieldToCellObj (FieldInfo field) => UI.current.cellObjs.ForceAdd(field, Cell.current, "

 public static void AddFieldToCellObj (Type type, string fieldName)
 {
  FieldInfo field = GetCachedField(type,fieldName);
  if (field == null)
  {
   field = type.GetField(fieldName, BindingFlags.Instance | BindingFlags.Public | BindingFlags.NonPublic);
   if (field == null) Debug.LogError($"Could not find GUI field for {type.Name}:{fieldName}");
   else SetCachedField(field, type, fieldName);
  }

  UI.current.cellObjs.ForceAdd(field, Cell.current, "Field");
 }

 private static FieldInfo GetCachedField (Type type, string fieldName)
 {
  if (fieldsCaches.TryGetValue(type, out var fieldsDict))
  {
   if (fieldsDict.TryGetValue(fieldName, out FieldInfo field))
    return field;
```

```
   else
    return null;
  }
  else
   return null;
 }

 private static void SetCachedField (FieldInfo field, Type type, string fieldName)
 {
  if (!fieldsCaches.ContainsKey(type))
   fieldsCaches.Add(type, new Dictionary<string,FieldInfo>());
  fieldsCaches[type].ForceAdd(fieldName, field);
 }

 private static readonly Dictionary<Type, Dictionary<string,FieldInfo>> fieldsCaches = new Dictionary<Typ

#endregion


#region Editor

 [AttributeUsage(AttributeTargets.Method, AllowMultiple = true)]
 public sealed class EditorAttribute : Attribute
 {
  public Type type;
  public string cat;
```

```
 public EditorAttribute (Type type) { this.type = type; }
 public EditorAttribute (Type type, string cat) { this.type = type; this.cat = cat; }
}



public static bool Editor (dynamic obj, object[] args=null, string cat=null)
/// Draws special class editor. Returns false if no editor found
{
 if (cachedEditors == null)
  PopulateCachedEditors();

 Type type = obj.GetType();
 cachedEditors.TryGetValue((type, cat), out Delegate editorAction);
 if (editorAction != null) Invoke(editorAction, obj, args);

 return editorAction != null;
}

private static void Invoke<T> (Delegate action, T obj, object[] args)
/// Can't invoke delegate directly, so using generic wrapper
/// Providing a T obj will let it know what type to use
{
 if (action is Action<T,object[]>)
  ((Action<T,object[]>)action).Invoke(obj, args);
```

```
 else
  ((Action<T>)action).Invoke(obj);
  // SNIPPET: Calling delegate of any type
}

/*private static Action<T> GetEditor<T>(T ignored) { return GetEditor<T>(); } //just to call generic  (SNIPP
public static Action<T> GetEditor<T> ()
{
 if (cachedEditors == null)
  PopulateCachedEditors();

 if (cachedEditors.TryGetValue(typeof(T), out Delegate editorAction)) return (Action<T>)editorAction;
 else return null;
}*/

private static Dictionary<(Type,string),Delegate> cachedEditors = null;




private static void PopulateCachedEditors ()
{
 cachedEditors = new Dictionary<(Type,string),Delegate>();
 Dictionary<EditorAttribute,MethodInfo> methodsDict = GetAllMethodsWithAttribute<EditorAttribute>();

 foreach (var kvp in methodsDict)
```

```
 {
  EditorAttribute editorAtt = kvp.Key;
  MethodInfo methodInfo = kvp.Value;

  if (cachedEditors.ContainsKey((editorAtt.type, editorAtt.cat))) continue;

  string methodName = methodInfo.ReflectedType.FullName + "." + methodInfo.Name;
  DynamicMethod editorMethod = new DynamicMethod(methodInfo.Name, typeof(void), new Type[1] { m

  //Type delegateType = Expression.GetDelegateType(Type[] {methodInfo.GetParameters());
  Delegate action = CreateDelegate(methodInfo); //Delegate.CreateDelegate(typeof(Action<dynamic>), n
  cachedEditors.Add((editorAtt.type, editorAtt.cat), action);
 }
}


static Delegate CreateDelegate (MethodInfo methodInfo)
/// SNIPPET: Creating delegate of any type from MethodInfo
{
 ParameterInfo[] pars = methodInfo.GetParameters();

 Type[] parTypes = new Type[pars.Length + 1]; //one for return type
 for (int i=0; i<pars.Length; i++)
  parTypes[i] = pars[i].ParameterType;
 parTypes[parTypes.Length-1] = methodInfo.ReturnType;
```

```
 var delegateType = Expression.GetDelegateType(parTypes);
 return Delegate.CreateDelegate(delegateType, null, methodInfo);
}



private static Dictionary<T,MethodInfo> GetAllMethodsWithAttribute<T> () where T: Attribute
{
 Dictionary<T,MethodInfo> dict = new Dictionary<T, MethodInfo>();
 string aName = typeof(Draw).Assembly.FullName;

 foreach (Assembly a in AppDomain.CurrentDomain.GetAssemblies())
 {
  bool isDependent = false;
  foreach (AssemblyName dan in a.GetReferencedAssemblies())
   if (dan.FullName == aName) { isDependent = true; break; }
  #if UNITY_2019_2_OR_NEWER //don't know if it will affect anything, but doint it just not to ruin everythi
  if (!isDependent) continue;
  #else
  if (!isDependent && a.FullName!=aName) continue;
  #endif

  foreach(Type t in a.GetTypes())
  {
   //if (!t.IsAbstract || !t.IsSealed) continue;

   foreach(MethodInfo m in t.GetMethods(BindingFlags.Static | BindingFlags.Public))
```

```
    foreach(Attribute att in m.GetCustomAttributes())
    {
     if (att is T tatt)
     {
      if (dict.ContainsKey(tatt))
       Debug.LogError("Editor method is defined twice. Attach to debug."); //don't throw exception
      else dict.Add(tatt, m);
     }
    }
  }
 }

 return dict;
}

#endregion



#region Other Elements

 public static bool IsButtonPrePressed =>  Cell.current == pressedButton;
 /// Should be called in the same cell with Draw.Button

 public static bool Button (string label, bool visible=true, GUIStyle style=null, MouseCursor cursor=0)
 {
  if (UI.current.layout) return false;
```

```
if (UI.current.optimizeElements && !UI.current.IsInWindow()) return false;

if (Event.current.type==EventType.MouseDown  &&  Event.current.button==0  &&  Cell.current.Contains
{
 pressedButton = Cell.current;
 UI.current.editorWindow?.Repaint();
}

if (visible  &&  Event.current.type == EventType.Repaint)
{
 if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true);
 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:buttonPadding);

 if (cursor != 0 && !Cell.current.disabled) UnityEditor.EditorGUIUtility.AddCursorRect (rect, cursor);
 if (style == null) style = UI.current.styles.button;
 style.Draw(rect, new GUIContent(label), isHover:false, isActive:false, on:pressedButton==Cell.current, h
 if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();
}

if (pressedButton == Cell.current  &&  Event.current.rawType == EventType.MouseUp)
{
 pressedButton = null;
 UI.current.editorWindow?.Repaint();

 if (Event.current.button==0  &&  Cell.current.Contains(UI.current.mousePos)  &&  !Cell.current.inactive)
 {
```

```
   UI.current.MarkChanged();
   return true;
  }
 }

 return false;
}


public static bool Button (bool visible=true, MouseCursor cursor=0)
{
 return Button("", visible:visible, cursor:cursor);
}


public static bool Button (Texture2D icon, float iconScale=1, bool visible=true, MouseCursor cursor=0)
{
 bool pressed = Button(visible:visible, cursor:cursor);
 if (icon != null) Icon(icon, scale:iconScale);
 return pressed;
}


public static bool Button (string label, Texture2D icon, int iconWidth=20, float iconScale=1, bool visible=tr
{
 bool pressed = Button(visible:visible, cursor:cursor);
```

```
 using (Cell.RowPx(iconWidth)) Icon(icon, scale:iconScale);
 using (Cell.Row) Label(label);
 return pressed;
}



public static bool CheckButton (bool val, bool visible = true)
{
 if (UI.current.layout) return val;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return val;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:buttonPadding);

 if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true);

 bool newVal;
 if (visible) newVal = UnityEngine.GUI.Toggle(rect, val, "", UI.current.styles.button);
 else newVal = UnityEngine.GUI.Toggle(rect, val, "", GUIStyle.none);

 if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();

 if (val != newVal)
 {
  UI.current.MarkChanged();
  val = newVal;
 }
```

```
 return val;
}


public static void CheckButton (ref bool val, bool visible = true)
 { val = CheckButton(val, visible:visible); }


public static bool CheckButton (bool val, string label, bool visible=true)
{
 val = CheckButton(val, visible:visible);
 Label(label);
 return val;
}

public static void CheckButton (ref bool val, string label, bool visible=true)
 { val = CheckButton(val, label, visible:visible); }


public static bool CheckButton (bool val, Texture2D iconOff, Texture2D iconOn, float iconScale=1, bool v
{
 val = CheckButton(val, visible:visible);
 Icon(val ? iconOff : iconOn, scale:iconScale);
 return val;
}
```

```
public static bool CheckButton (bool val, Texture2D icon, bool visible=true)
{
 val = CheckButton(val, visible:visible);
 Icon(icon);
 return val;
}

public static void CheckButton (ref bool val, Texture2D icon, bool visible=true)
{
 val = CheckButton(val, visible:visible);
 Icon(icon);
}

public static void CheckButton (ref bool val, Texture2D iconOff, Texture2D iconOn, bool visible=true)
 { val = CheckButton(val, iconOff, iconOn, visible:visible); }


public static bool CheckButton (bool val, string label, Texture2D iconOff, Texture2D iconOn, int iconWidth
{
 val = CheckButton(val, visible:visible);
 using (Cell.RowPx(iconWidth)) Icon(val ? iconOff : iconOn);
 using (Cell.Row) Label(label);
 return val;
}
```

```
public static void CheckButton (ref bool val, string label, Texture2D iconOff, Texture2D iconOn, int iconW
 { val = CheckButton(val, label, iconOff, iconOn, iconWidth:iconWidth, visible:visible); }



public static void Element (GUIStyle style)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;
 if (Event.current.type != EventType.Repaint) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom);

 style.Draw(rect, false, false, false ,false);
}

public static void Element (Rect rect, GUIStyle style)
{
 if (UI.current.layout) return;
 //if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;
 if (Event.current.type != EventType.Repaint) return;

 Rect scrRect = UI.current.scrollZoom!=null ? UI.current.scrollZoom.ToScreen(rect) : rect;

 style.Draw(scrRect, false, false, false ,false);
}
```

```
public static void Element (GUIStyle style, int padding)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;
 if (Event.current.type != EventType.Repaint) return;

 Rect scrRect = Cell.current.GetRect(UI.current.scrollZoom, padding);

 style.Draw(scrRect, false, false, false ,false);
}


public static void Icon (Texture icon, Vector2 center, Color color = new Color(), float scale=1)
/// Draws an icon in the given coord in native resolution (for zoom 1). Independent from cell
{
 if (UI.current.layout) return;
 if (icon == null) return; //happens when performing a build

 float zoom = UI.current.scrollZoom != null  ?  UI.current.scrollZoom.zoom  :  1;

 Rect rect = new Rect(center.x - icon.width/2f*scale, center.y - icon.height/2f*scale, icon.width*scale, icon
 if (UI.current.scrollZoom != null)
  rect = UI.current.scrollZoom.ToScreen(rect.position, rect.size);

 //non-cell releated rect, so using alternative optimize
 if (UI.current.optimizeElements)
```

```
 {
  Vector2 min = rect.min; Vector2 max = rect.max;
  if (max.x < -1 || max.y < -1 || min.x > Screen.width+1 || max.y > Screen.height+1) return; //Screen.width
 }

 if (color.a<0.001f) UnityEngine.GUI.DrawTexture(rect, icon, ScaleMode.ScaleAndCrop);
 else UnityEngine.GUI.DrawTexture(rect, icon, ScaleMode.ScaleAndCrop, true, 0, color, 0,0);
}

public static void Icon  (Texture icon, Color color = new Color(), float scale=1)
/// Draws an icon in the center of cell in native resolution (for zoom 1)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Icon(icon, Cell.current.InternalCenter, color, scale);
}


public static void Texture (Texture texture, Material mat = null, ScaleMode scaleMode = ScaleMode.Scal
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom);
```

```
 if (texture == null) texture = StylesCache.blankTex;
 if (texture == null) return; //doesn't load texture after build for some reason

 if (mat != null)
 {
  if (scaleMode != ScaleMode.ScaleToFit) UnityEditor.EditorGUI.DrawPreviewTexture(rect, texture, mat, s
  else UnityEditor.EditorGUI.DrawPreviewTexture(rect, texture, mat);
 }
 else UnityEngine.GUI.DrawTexture(rect, texture, ScaleMode.ScaleAndCrop); //UnityEditor.EditorGUI.Dra
}

public static void ColorizedTexture (Texture2D texture, Color color)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom);

 if (texture == null) texture = StylesCache.blankTex;

 UnityEngine.GUI.DrawTexture(rect, texture, ScaleMode.ScaleAndCrop, false, 1, color, 0,0); //UnityEditor
}

public static void ScrollableTexture (Texture2D texture, ScrollZoom scrollZoom)
{
```

```
if (UI.current.layout) return;
if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

Rect rect = Cell.current.GetRect(UI.current.scrollZoom);

if (Event.current.type == EventType.ScrollWheel && Cell.current.Contains(UI.current.mousePos))
 { scrollZoom.Zoom(Event.current.mousePosition-rect.position); Event.current.Use(); }

if (
 Event.current.type == EventType.MouseDown  &&  Event.current.button ==scrollZoom.scrollButton  &&
 Event.current.rawType == EventType.MouseUp ||
 scrollZoom.isScrolling)
  scrollZoom.Scroll();

//float rectAspect = rect.width / rect.height;
//float textureAspect = texture.width / texture.height;
//float scrollYfactor = textureAspect / rectAspect;

if (textureScrollZoomMat == null) textureScrollZoomMat = new Material( Shader.Find("Hidden/DPLayout
textureScrollZoomMat.SetFloat("_Scale", scrollZoom.zoom);
textureScrollZoomMat.SetFloat("_OffsetX", scrollZoom.scroll.x / rect.size.x);
textureScrollZoomMat.SetFloat("_OffsetY",  (1-scrollZoom.scroll.y) / rect.size.y + 1);
textureScrollZoomMat.SetTexture("_DispTex", texture);
textureScrollZoomMat.SetFloat("_CellSizeX", rect.size.x);
textureScrollZoomMat.SetFloat("_CellSizeY",  rect.size.y);
Shader.SetGlobalInt("_IsLinear", UnityEditor.PlayerSettings.colorSpace==ColorSpace.Linear ? 1 : 0);
```

```
 UnityEditor.EditorGUI.DrawPreviewTexture(rect, texture, textureScrollZoomMat, ScaleMode.StretchToF
}


public static void TextureIcon (Texture2D texture, int borderRadius = 3)
/// Draws a rounded texture preview at the cell background (no offset)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom);

 if (textureIconMat == null) textureIconMat = new Material( Shader.Find("Hidden/DPLayout/TextureIcon")
 textureIconMat.SetFloat("_Borders", 1/rect.width);
 Shader.SetGlobalInt("_IsLinear", UnityEditor.PlayerSettings.colorSpace==ColorSpace.Linear ? 1 : 0);

 if (texture == null) texture = StylesCache.blankTex;

 UnityEditor.EditorGUI.DrawPreviewTexture(rect, texture, textureIconMat);
}


public static void TextureIcon (Texture2DArray textureArr, int index, int borderRadius = 3)
/// Draws a rounded texture preview at the cell background (no offset)
{
```

```
 if (textureArr == null)
  { TextureIcon(null, borderRadius); return; }

 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom);

 if (textureArrIconMat == null) textureArrIconMat = new Material( Shader.Find("Hidden/DPLayout/Texture
 textureArrIconMat.SetFloat("_Borders", 1/rect.width);
 textureArrIconMat.SetFloat("_Index", index);
 textureArrIconMat.SetTexture("_MainTexArr", textureArr);
 Shader.SetGlobalInt("_IsLinear", UnityEditor.PlayerSettings.colorSpace==ColorSpace.Linear ? 1 : 0);

 UnityEditor.EditorGUI.DrawPreviewTexture(rect, StylesCache.blankTex, textureArrIconMat);
}


public static void MatrixPreviewTexture (Texture2D texture, bool colorize=false, bool relief=false, float mi
{
 if (textureRawMat==null) textureRawMat = new Material(Shader.Find("Hidden/MapMagic/TexturePreview

 textureRawMat.SetFloat("_Colorize", colorize ? 1 : 0);
 textureRawMat.SetFloat("_Relief", relief ? 1 : 0);
 textureRawMat.SetFloat("_MinValue", min);
 textureRawMat.SetFloat("_MaxValue", max);
```

```
 textureRawMat.SetInt("_Margins", margins);


 Texture(texture, textureRawMat);
}



public static void MatrixPreviewReliefSwitch (ref bool colorize, ref bool relief)
{
 using (Cell.Full)
 {
  Cell.EmptyRow();
  using (Cell.RowPx(12))
  {
   Cell.EmptyLine();
   using (Cell.LinePx(12))
   {
    Texture2D icon;
    if (colorize && relief) icon = UI.current.textures.GetTexture("DPUI/TexCh/BlackWhite");
    else icon = UI.current.textures.GetTexture("DPUI/TexCh/ColRelief");


    if (Draw.Button(icon, visible:false))
     { colorize = !colorize; relief = colorize; }
   }
   Cell.EmptyLinePx(3);
  }
  Cell.EmptyRowPx(3);
```

```
 }
}


public static void ProgressBar (float val, Color color)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom);

 Draw.Element(UI.current.styles.progressBarBackground);

 ProgressBarGauge(val, StylesCache.progressBarFill);
}


public static void ProgressBarGauge (float val, Texture2D fillTex=null, Color color=new Color())
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom);

 if (fillTex == null) fillTex = StylesCache.progressBarFill;
```

```
 if (multiplyMat == null) multiplyMat = new Material( Shader.Find("Hidden/DPLayout/Multiply") );

 Rect gaugeRect = rect;
 gaugeRect.width = gaugeRect.width * val;
 gaugeRect.width = Mathf.RoundToInt(gaugeRect.width);
 if (gaugeRect.width < 1) return;

 if (color.a < 0.00001f) color = new Color(0.4f, 0.65f, 1f);
 multiplyMat.SetColor("_Color", color);

 UnityEditor.EditorGUI.DrawPreviewTexture(gaugeRect, fillTex);
 UnityEditor.EditorGUI.DrawPreviewTexture(gaugeRect, fillTex, multiplyMat);
}


public static void URL (string label, string url)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);

 if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true);

 GUIStyle style = UI.current.styles.url;
 if (UnityEngine.GUI.Button(rect, label, style)) Application.OpenURL(url);
```

```
 UnityEditor.EditorGUIUtility.AddCursorRect (rect, UnityEditor.MouseCursor.Link);

 if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();
}

public static void Helpbox (string label, MessageType messageType = MessageType.None)
{
 //Element(UI.current.styles.foldoutBackground);
 //Label(label, UI.current.styles.helpBox);

 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);

 UnityEditor.EditorGUI.HelpBox(rect, label, messageType);
}

/*public static Enum EnumField (Enum val)
{
 if (UI.current.layout) return val;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return val;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);
```

```
 if (enumStyle == null)
 {
  Texture2D tex = Resources.Load("DPUI/Backgrounds/Enum") as Texture2D;
  enumStyle = new GUIStyle();
  enumStyle.normal.background = tex;
  enumStyle.border = new RectOffset(1,1,1,1);
  enumStyle.overflow.bottom += 1;
 }

 if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true);

 Enum newVal = EditorGUI.EnumPopup(rect, val, enumStyle);

 if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();

 if (!val.Equals(newVal))
 {
  UI.current.MarkChanged();
  val = newVal;
 }

 return val;
}
public static void EnumField (ref Enum val) { val = EnumField(val); }
```

```
public static Enum EnumField (Enum val, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 using(Cell.RowRel(1-Cell.current.fieldWidth))
  Label(label);

 using (Cell.RowRel(Cell.current.fieldWidth))
  val = EnumField(val);

 return val;
}
public static void EnumField (ref Enum val, string label) { val = EnumField(val, label); }*/



public static int PopupSelector (int selectedIndex, string[] displayedOptions)
{
 if (UI.current.layout) return selectedIndex;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return selectedIndex;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:buttonPadding);

 if (Cell.current.disabled) UnityEditor.EditorGUI.BeginDisabledGroup(true);
 int newIndex = UnityEditor.EditorGUI.Popup(rect, selectedIndex, displayedOptions, UI.current.styles.enu
 if (Cell.current.disabled) UnityEditor.EditorGUI.EndDisabledGroup();
```

```
 Vector2 signPos = Cell.current.InternalCenter; signPos.x += Cell.current.finalSize.x/2 - 10; signPos.y-=1
 Icon(StylesCache.enumSign, signPos);

 if (!newIndex.Equals(selectedIndex))
 {
  UI.current.MarkChanged();
  selectedIndex = newIndex;
 }

 return selectedIndex;
}

public static void PopupSelector (ref int selectedIndex, string[] displayedOptions)
 { selectedIndex = PopupSelector(selectedIndex, displayedOptions); }

public static int PopupSelector (int selectedIndex, string[] displayedOptions, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return selectedIndex; }

 using (Cell.RowRel(1-Cell.current.fieldWidth)) Label(label);
 using (Cell.RowRel(Cell.current.fieldWidth)) selectedIndex = PopupSelector(selectedIndex, displayedOp
```

```
 return selectedIndex;
}


public static void PopupSelector (ref int selectedIndex, string[] displayedOptions, string label) =>
 selectedIndex = PopupSelector(selectedIndex, displayedOptions, label);

public static T PopupSelector<T> (T selectedItem, T[] allItems, string[] displayedOptions, string label)
{
 int index = allItems.Find(selectedItem);
 int newIndex = PopupSelector(index, displayedOptions);

 if (newIndex != index  &&  newIndex >= 0)
  return allItems[newIndex];
 else
  return selectedItem;
}

public static T PopupSelector<T> (ref T selectedItem, T[] allItems, string[] displayedOptions, string label)
 selectedItem = PopupSelector(selectedItem, allItems, displayedOptions, label);


public static void TypeSelector<T> (ref T obj, string label, ref Type[] allTypes, ref string[] allNames, bool a
{
 if (allTypes == null)
 {
```

```
  allTypes = typeof(T).Subtypes(allAssemblies:allAssemblies);

  allNames = new string[allTypes.Length];
  for (int i=0; i<allNames.Length; i++)
   allNames[i] = allTypes[i].Name;
 }

 int selectedNum = obj != null ? allTypes.Find(obj.GetType()) : -1;
 Draw.PopupSelector(ref selectedNum, allNames, label);

 if (Cell.current.valChanged)
  obj = (T)Activator.CreateInstance(allTypes[selectedNum]);
}


public static void SilhouetteLabel (string label)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);

 rect.x+=1; rect.y+=1; EditorGUI.LabelField(rect, label, UI.current.styles.blackLabel);
 rect.y-=2; EditorGUI.LabelField(rect, label, UI.current.styles.blackLabel);
 rect.x-=2;  EditorGUI.LabelField(rect, label, UI.current.styles.blackLabel);
 rect.y+=2; EditorGUI.LabelField(rect, label, UI.current.styles.blackLabel);
```

```
 rect.x+=1; rect.y-=1; //EditorGUI.LabelField(rect, label, UI.current.styles.whiteLabel);

 rect.x+=1; EditorGUI.LabelField(rect, label, UI.current.styles.blackLabel);
 rect.x-=2; EditorGUI.LabelField(rect, label, UI.current.styles.blackLabel);
 rect.x+=1; rect.y-=1;  EditorGUI.LabelField(rect, label, UI.current.styles.blackLabel);
 rect.y+=2; EditorGUI.LabelField(rect, label, UI.current.styles.blackLabel);
 rect.y-=1; EditorGUI.LabelField(rect, label, UI.current.styles.whiteLabel);
}


public static void BackgroundRightLabel (string label, GUIStyle style=null, GUIStyle backStyle=null, float
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);
 if (style==null) style = UI.current.styles.label;

 float width = style.CalcSize( new GUIContent(label) ).x;
 rect.x += rect.width - width;
 rect.width = width;
 rect.x -= rightOffset * (UI.current.scrollZoom!=null ? UI.current.scrollZoom.zoom : 0);

 Rect backRect = rect;
 backRect.yMin += 2 * (UI.current.scrollZoom!=null ? UI.current.scrollZoom.zoom : 0);
 backRect.yMax -= 2 * (UI.current.scrollZoom!=null ? UI.current.scrollZoom.zoom : 0);
```

```
 backRect.x -= 1 * (UI.current.scrollZoom!=null ? UI.current.scrollZoom.zoom : 0);
 backRect.width += 3 * (UI.current.scrollZoom!=null ? UI.current.scrollZoom.zoom : 0);

 if (backStyle != null)
 {
  if (Event.current.type == EventType.Repaint)
   backStyle.Draw(backRect, false, false, false, false);
 }

 else EditorGUI.DrawRect(backRect, new Color(0,0,0,0.8f));

 EditorGUI.LabelField(rect, label, style:style);
}


public static string EditableLabel (string label, GUIStyle style=null)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return label; }

 Cell editLabelCell = Cell.current;

 using (Cell.Row)
 {
  if (style == null) style = UI.current.styles.middleLabel;
```

```
  if (activeEditLabelCell != editLabelCell)  //non-editable
   Label(label, style);
  else
   label = LabelEditField(label, style);
 }

 using (Cell.RowPx(20))
 {
  if (Button(StylesCache.pencilTex, visible:false, cursor:UnityEditor.MouseCursor.Link))
   activeEditLabelCell = editLabelCell;
 }

 return label;
}



public static void EditableLabel (ref string label)
 { label = EditableLabel(label); }



public static string EditableLabelRight (string label, GUIStyle style=null)
/// Same editable label, but placed to the right of pencil icon
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return label; }
```

```
 Cell editLabelCell = Cell.current;


 using (Cell.RowPx(20))
 {
  if (Button(StylesCache.pencilTex, visible:false, cursor:UnityEditor.MouseCursor.Link))
   activeEditLabelCell = editLabelCell;
 }


 using (Cell.Row)
 {
  if (style == null) style = UI.current.styles.middleLabel;


  if (activeEditLabelCell != editLabelCell)  //non-editable
   Label(label, style);
  else
   label = LabelEditField(label, style);
 }


 return label;
}



public static void EditableLabelRight (ref string label, GUIStyle style=null)
 { label = EditableLabelRight(label, style); }
```

```
public static void SearchLabel (ref string label, GUIStyle style=null, bool forceFocus=false)
 { label = SearchLabel(label, style, forceFocus); }



public static string SearchLabel (string label, GUIStyle style=null, bool forceFocus=false)
{
 GUIStyle backStyle = UI.current.textures.GetElementStyle("DPUI/Backgrounds/RoundField");
 Draw.Element(backStyle);

 using (Cell.Row)
 {
  Cell editCell = Cell.current;

  //enabling edit
  if (Cell.current.Contains(UI.current.mousePos)  &&  UI.current.mouseButton==0)
   activeEditLabelCell = Cell.current;

  UnityEditor.EditorGUIUtility.AddCursorRect (Cell.current.GetRect(UI.current.scrollZoom), UnityEditor.M

  using (Cell.RowPx(20))
   Draw.Icon(UI.current.textures.GetTexture("DPUI/Icons/ZoomSmall"), scale:0.5f);

  using (Cell.Row)
  {
   if (style == null) style = UI.current.styles.middleBlackLabel;
```

```
   if (forceFocus || activeEditLabelCell == editCell)  //editable
    label = LabelEditField(label, style);
   else
    Label(label, style);
  }
 }

 using (Cell.RowPx(22))
 {
  if (Button(UI.current.textures.GetTexture("DPUI/Icons/Close"), iconScale:0.5f, visible:false))
  {
   UI.current.MarkChanged();
   label = "";
   UnityEditor.EditorGUI.FocusTextInControl(null);
  }
 }

 return label;
}


public static string LabelEditField (string label, GUIStyle style=null)
{
 if (UI.current.layout) return label;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return label;
```

```
 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);

 if (style == null) style = UnityEditor.EditorStyles.label;

 //editing
 UnityEngine.GUI.SetNextControlName("LayerFoldoutNextFocus"); //to focus in text right after pressing e
 string newLabel = UnityEditor.EditorGUI.TextField(rect, label, style:style);
 UnityEditor.EditorGUI.FocusTextInControl("LayerFoldoutNextFocus");
 UI.current.editorWindow?.Repaint();

 //exit editing
 if (Event.current.keyCode==KeyCode.KeypadEnter || Event.current.keyCode==KeyCode.Return || Event
  (Event.current.type==EventType.MouseDown && !rect.Contains(Event.current.mousePosition))) //if clicl
   activeEditLabelCell = null;

 if (newLabel != label)
 {
  UI.current.MarkChanged();
  label = newLabel;
 }

 return label;
}

public static Cell activeEditLabelCell; //to know what label we are currently editing
```

```
public static void Grid (
 Color color,
 Color background = new Color(),
 int cellsNumX = 4,
 int cellsNumY = 4,
 bool fadeWithZoom = true)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom);
 Vector2 cellSize = new Vector2(rect.width / cellsNumX, rect.height / cellsNumY);
 DrawGrid(rect, cellSize, new Vector2(), color, background, 0.5f, 1,fadeWithZoom);
}


public static void StaticGrid (Rect displayRect,
 float cellSize,
 Color color,
 Color background = new Color(),
 bool fadeWithZoom = true)
/// Draws grid in window-related rect. Moves with zoom. Useful for backgrounds
{
 if (UI.current.layout) return;
```

```
//float dpiFactor = UI.current.DpiScaleFactor;
//displayRect.width = (int)(float)(displayRect.width*dpiFactor + 0.5f) / dpiFactor;

Vector2 dispCellSize = new Vector2(cellSize, cellSize);
if (UI.current.scrollZoom != null)
 dispCellSize *= UI.current.scrollZoom.zoom;
//dispCellSize.x = (int)(float)(dispCellSize.x+0.5f);
//dispCellSize.y = (int)(float)(dispCellSize.y+0.5f);

Vector2 dispOffset;
if (UI.current.scrollZoom != null)
 dispOffset = new Vector2(-UI.current.scrollZoom.scroll.x+displayRect.x, UI.current.scrollZoom.scroll.y-d
else
 dispOffset = new Vector2(displayRect.x, -displayRect.height-displayRect.y);
dispOffset.x += dispCellSize.x*10000;
dispOffset.y += dispCellSize.y*10000;  //hiding the line pass through 0-1 pixel in 10000 cells away

//dispOffset.x = (int)(float)(dispOffset.x+0.5f);
//dispOffset.y = (int)(float)(dispOffset.y+0.5f);

//displayRect.position *= dpiFactor;
//displayRect.size *= dpiFactor;

DrawGrid(displayRect, dispCellSize, dispOffset, color, background, 1, 0, fadeWithZoom);
}
```

```
private static void DrawGrid (Rect displayRect,
 Vector2 cellSize, Vector2 cellOffset,
 Color color, Color background,
 float lineOpacity, float bordersOpacity,
 bool fadeWithZoom)
/// Draws grid in window-related rect
{
 if (UI.current.layout) return;
 if (StylesCache.blankTex == null) return;  //happens when performing a build

 if (background.a == 0 && background.r == 0 && background.g == 0 && background.b == 0)
  background = new Color(color.r, color.g, color.b, 0); //to avoid blacking on fadeOnZoom

 if (gridMat == null) gridMat = new Material( Shader.Find("Hidden/DPLayout/Grid") );

 if (fadeWithZoom)
 {
  float clampZoom = UI.current.scrollZoom!=null ? UI.current.scrollZoom.zoom : 1;
  if (clampZoom > 1) clampZoom = 1;
  color = color*clampZoom  +  background*(1-clampZoom);
 }

 float dpiFactor = UI.current.DpiScaleFactor;

 gridMat.SetColor("_Color", color);
```

```
 gridMat.SetColor("_Background", background);

 gridMat.SetFloat("_CellSizeX", cellSize.x * dpiFactor);
 gridMat.SetFloat("_CellSizeY", cellSize.y * dpiFactor);
 gridMat.SetFloat("_CellOffsetX", cellOffset.x * dpiFactor);
 gridMat.SetFloat("_CellOffsetY", cellOffset.y * dpiFactor);

 gridMat.SetFloat("_LineOpacity", lineOpacity);
 gridMat.SetFloat("_BordersOpacity", bordersOpacity);

 gridMat.SetVector("_ViewRect", new Vector4(displayRect.x, displayRect.y, displayRect.size.x, displayRe

 UnityEditor.EditorGUI.DrawPreviewTexture(displayRect, StylesCache.blankTex, gridMat, ScaleMode.St
}


public static void StaticAxis (Rect displayRect, int pos, bool isVertical, Color color)
/// Infinite horizontal or vertical line
/// Draws 1-pixel rect in base coordinates (or nothing if it is out of cell)
/// Used to draw axis
/// StaticGrid and StaticAxis use displayRect (like (0, 0, Screen.width-toolbarWidth, Screen.height)) instea
{
 if (UI.current.layout) return;

 Rect lineRect = isVertical ?
  new Rect (
```

```
    pos*UI.current.scrollZoom.zoom + UI.current.scrollZoom.scroll.x,
    displayRect.y,
    1,
    displayRect.height) :
   new Rect (
    displayRect.x,
    pos*UI.current.scrollZoom.zoom + UI.current.scrollZoom.scroll.y,
    displayRect.width,
    1);

  UnityEditor.EditorGUI.DrawRect(lineRect, color);
}



public static void Histogram (float[] histogram, Vector4 color, Vector4 backColor)
{
  Material histogramMat = UI.current.textures.GetMaterial("Hidden/DPLayout/Histogram");
  histogramMat.SetFloatArray("_Histogram", histogram);
  histogramMat.SetVector("_Backcolor", backColor);
  histogramMat.SetVector("_Forecolor", color);
  histogramMat.SetInt("_HistogramLength", histogram.Length);

  Draw.Texture(null, histogramMat);
}
```

```
public static bool LayerChevron (
 int num,
 ref int expanded)
{
 int newexpanded = expanded;
 if (LayerChevron(expanded==num))
  newexpanded = num;
 else
  { if (expanded==num) newexpanded = -1; }

 if (expanded != newexpanded)
 {
  expanded = newexpanded;
  UI.RemoveFocusOnControl(); //otherwise still editing field in the other layer, and it got the previous valu
  return true;
 }

 return false;
}


public static bool LayerChevron (bool expanded)
{
 return CheckButton(expanded,
  UI.current.textures.GetTexture("DPUI/Chevrons/Down"),
  UI.current.textures.GetTexture("DPUI/Chevrons/Left"),
```

```
  iconScale:0.5f,
  visible:false);
}


public static void Equalizer (float[] arr, int length=-1)
{
 if (length < 0)
  length = arr.Length;

 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return; }

 for (int i=0; i<arr.Length; i++)
  using (Cell.Row) EqualizerElement(ref arr[i]);
}


public static void EqualizerElement (ref float val)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Texture2D diamonTex = StylesCache.diamonTex;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom);
```

```
 Rect lineRect = new Rect(rect.x+rect.width/2-1, rect.y, 2, rect.height);
 UnityEditor.EditorGUI.DrawRect(lineRect, Color.gray);

 Vector2 iconCenter = new Vector2(rect.x+rect.width/2, rect.y + (1-val)*rect.height);
 Rect iconRect = new Rect(iconCenter.x-diamonTex.width/2, iconCenter.y-diamonTex.height/2, diamonT
 UnityEngine.GUI.DrawTexture(iconRect, diamonTex, ScaleMode.ScaleAndCrop);

 if (DragDrop.TryDrag(Cell.current, UI.current.mousePos))
 {
  float newVal = 1-((DragDrop.initialMousePos.y + DragDrop.totalDelta.y - rect.y) / rect.height);
  if (newVal > 1) newVal = 1;
  if (newVal < 0) newVal = 0;
  if (val != newVal) UI.current.MarkChanged();
  val = newVal;

  UnityEditor.EditorGUI.LabelField( new Rect(iconRect.x-15, iconRect.y+iconRect.height, iconRect.width-

  iconRect = new Rect(iconCenter.x-diamonTex.width/2, iconCenter.y-diamonTex.height/2, diamonTex.w
 }
 DragDrop.TryStart(Cell.current, UI.current.mousePos, iconRect);
 DragDrop.TryRelease(Cell.current, UI.current.mousePos);

 UnityEngine.GUI.DrawTexture(iconRect, diamonTex, ScaleMode.ScaleAndCrop);
}
```

```
public static float FieldDragIcon (float val, Texture2D icon=null)
/// MM1-style field with a drag slider cursor icon
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return val; }

 Draw.Element(UI.current.styles.field, padding:fieldPadding);

 using (Cell.RowPx(20))
 {
  if (icon==null) icon = UI.current.textures.GetTexture("DPUI/Icons/Slider");
  Icon(icon);
  val = DragValue(val);
 }

 using (Cell.Row)
  val = Field(val, style:UI.current.styles.label);

 return val;
}

public static void FieldDragIcon (ref float val, Texture2D icon=null) { val = FieldDragIcon(val,icon); }


public static float IconField (float val, string label, Texture2D icon)
```

```
{
 using (Cell.RowRel(1-Cell.current.fieldWidth))
 {
  using (Cell.RowPx(icon.width)) Icon(icon);
  using (Cell.Row) Label(label);
  val = DragValue(val);
 }
 using (Cell.RowRel(Cell.current.fieldWidth))
  val = Field(val);

 return val;
}

public static void IconField (ref float val, string label, Texture2D icon) { val = IconField(val, label, icon); }



public static void Mesh (Mesh mesh, Material mat, bool clip=true)
/// Draws a mesh within a cell, presuming mesh fits in 0-1 bounds
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom);

 Matrix4x4 prs = Matrix4x4.TRS(
  new Vector3 (rect.position.x, rect.position.y+rect.height, 0),
```

```
 new Quaternion(0.7071067811865475f, 0, 0, 0.7071067811865475f),
 new Vector3(rect.size.x, 0, rect.size.y) );

if (mat.HasProperty("_ClipRect"))
{
 if (clip)
 {
  float dpiScaleFactor = UI.current.DpiScaleFactor;

  Rect clipRect = rect;



  clipRect.position *= dpiScaleFactor;
  clipRect.size *= dpiScaleFactor;
  clipRect.position += UI.current.subWindowRect.position * dpiScaleFactor;

  Rect containerRect = GetRootVisualContainerRect(UI.current.editorWindow);
  containerRect.position *= dpiScaleFactor;
  containerRect.size *= dpiScaleFactor;
  clipRect.position += containerRect.position;

  clipRect = CoordinatesExtensions.Intersect(containerRect, clipRect);

  //clipping by window
  //hacky there, but I'm tired of that stuff. Seems to be no way to get proper clip rect to shader, neither ge
 // Rect containerRect = GetRootVisualContainerRect(UI.current.editorWindow);
```

```
  // Rect windowRect = new Rect(0,0, containerRect.width, containerRect.height); //new Rect(10,10,1000
  // clipRect = CoordinatesExtensions.Intersect(windowRect, clipRect);
  // clipRect.y += containerRect.y + UI.current.subWindowRect.y;
  // clipRect.x += containerRect.x;

   mat.SetVector("_ClipRect", new Vector4(clipRect.x, clipRect.y, clipRect.xMax, clipRect.yMax));
  }
  else mat.SetVector("_ClipRect", new Vector4(0, 0, -1, -1));
 }

 mat.SetPass(0);
 Graphics.DrawMeshNow(mesh, prs);
}

private static Rect GetRootVisualContainerRect (EditorWindow window)
{
 #if UNITY_2019_1_OR_NEWER
 return window.rootVisualElement.layout;
 #else
 if (rvcLayoutProp == null)
 {
  Type winType = UI.current.editorWindow.GetType();
  rvcProp = winType.GetProperty("rootVisualContainer", BindingFlags.Instance | BindingFlags.NonPublic
 }
 object rvc = rvcProp.GetValue(UI.current.editorWindow);
 if (rvcLayoutProp == null)
```

```
 {
  Type rvcType = rvc.GetType();
  rvcLayoutProp = rvcType.GetProperty("layout", BindingFlags.Instance | BindingFlags.Public);
 }
 return (Rect)rvcLayoutProp.GetValue(rvc);
 #endif
}

#if !UNITY_2019_1_OR_NEWER
private static PropertyInfo rvcProp = null;
private static PropertyInfo rvcLayoutProp = null;
#endif

public static void Rect (Color color)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom);

 EditorGUI.DrawRect(rect, color);
}


public static void Rect (Rect rect, Color color)
{
```

```
 if (UI.current.layout) return;

 rect = UI.current.scrollZoom.ToScreen(rect);

 EditorGUI.DrawRect(rect, color);
}

public static void ToolbarSeparator ()
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom);

 EditorGUI.DrawRect(rect, StylesCache.isPro ? Color.black : new Color(0.57f, 0.57f, 0.57f, 1));
}

public static void DebugRect (int level=0)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 DebugRect(new Rect(Cell.current.worldPosition, Cell.current.finalSize), level);
}
```

```
public static void DebugRect (Rect rect, int level=0)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 if (UI.current.scrollZoom != null)
  rect = UI.current.scrollZoom.ToScreen(rect.position, rect.size);

 Color color = new Color(level/3f, 1-level/5f, 0, 0.5f);

 if (rect.width < 0 || rect.height < 0)
  color = new Color(0, 0, 0, 1f);

 if (multiplyMat == null)
  multiplyMat = new Material( Shader.Find("Hidden/DPLayout/Multiply") );
 multiplyMat.SetColor("_Color", color);

 EditorGUI.DrawPreviewTexture(rect, StylesCache.debugTex);
}

public static void DebugRect (Cell cell, int level=0)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;
```

```
 DebugRect(new Rect(cell.worldPosition, cell.finalSize), level);
}

public static void DebugRectRecursive (Cell cell, int level=0)
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 DebugRect(new Rect(cell.worldPosition, cell.finalSize), level);

 if (cell.subCells != null)
 {
  int childCount = cell.subCells.Count;
  for (int i=0; i<childCount; i++)
  {
   Cell child = cell.subCells[i];
   DebugRectRecursive(child, level+1);
  }
 }
}

public static void DebugRectRecursive ()
 { DebugRectRecursive(Cell.current); }

public static void DebugMousePos ()
{
```

```
 GetCursorPos(out Vector2Int hMousePos);
 Vector2 hwMousePos = new Vector2(hMousePos.x, hMousePos.y);
 hwMousePos -= UI.current.editorWindow.position.position;
 hwMousePos.y -= 20;
 Rect hwMouseRect = new Rect(hwMousePos.x-3, hwMousePos.y-3, 6,6);
 EditorGUI.DrawRect(hwMouseRect, Color.green);

 Rect mouseRect = new Rect(Event.current.mousePosition.x-3, Event.current.mousePosition.y-3, 6,6);
 EditorGUI.DrawRect(mouseRect, Color.red);
}

#endregion

#region Animation Curve

 private static AnimationCurve windowCurveRef = null;

 private static Type curveWindowType;
 private static Type GetCurveWindowType ()
 {
  if (curveWindowType == null) curveWindowType = typeof(EditorWindow).Assembly.GetType("UnityEdito
  return curveWindowType;
 }
```

```
public static void AnimationCurve (
 AnimationCurve src,
 Rect ranges=new Rect(),
 Color color = new Color())
{
 if (UI.current.layout) return;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return;

 Rect displayRect = Cell.current.GetRect(UI.current.scrollZoom);

 if (ranges.width < Mathf.Epsilon && ranges.height < Mathf.Epsilon) { ranges.width = 1; ranges.height = 1
 if (color.a == 0) color = Color.white;

 //recording undo on change if the curve editor window is opened (and this current curve is selected)
 try
 {
  Type curveWindowType = GetCurveWindowType();
  if (UI.current.editorWindow != null  &&  EditorWindow.focusedWindow.GetType() == curveWindowType)
  {
   AnimationCurve windowCurve = curveWindowType.GetProperty("curve").GetValue(EditorWindow.focu
   if (windowCurve == src)
   {
    if (windowCurveRef == null) windowCurveRef = windowCurve.Copy();
    if (!windowCurve.IdenticalTo(windowCurveRef))
    {
     Keyframe[] tempKeys = windowCurve.keys;
```

```
      windowCurve.keys = windowCurveRef.keys;
      UI.current.MarkChanged();

      windowCurve.keys = tempKeys;

      windowCurveRef = windowCurve.Copy();
     }
    }
   }
   else windowCurveRef = null;
  }
  catch {};

  if (Event.current.type!=EventType.MouseDown || Event.current.button!=1 || EditorWindow.focusedWindo
  //hack to allow right clicking on curve to expose it
   EditorGUI.CurveField(displayRect, src, color, ranges);
 }

#endregion


#region Foldout

 public static void Foldout (ref bool src, string label)
 {
  if (UI.current.optimizeCells && !UI.current.IsInWindow())
```

```
 { Cell.current.Skip(); return; }

 src = Draw.CheckButton(src, visible:false);

 using (Cell.Row)
  Draw.Label(label, UI.current.styles.boldLabel);

 using (Cell.RowPx(15))
  src = CheckButton(src,
   UI.current.textures.GetTexture("DPUI/Chevrons/SmallDown"),
   UI.current.textures.GetTexture("DPUI/Chevrons/SmallLeft"),
   visible:false);
}


public static void FoldoutLeft (ref bool src, string label)
{
 if (UI.current.optimizeCells && !UI.current.IsInWindow())
  { Cell.current.Skip(); return; }

 src = Draw.CheckButton(src, visible:false);

 using (Cell.RowPx(10))
  src = CheckButton(src,
   UI.current.textures.GetTexture("DPUI/Chevrons/SmallDown"),
   UI.current.textures.GetTexture("DPUI/Chevrons/SmallRight"),
```

```
   visible:false);

 using (Cell.Row)
  Draw.Label(label, UI.current.styles.boldLabel);
}



public struct FoldoutGroup : IDisposable
{
 Cell outCell;
 Cell innerCell;

 public FoldoutGroup (ref bool opened, string label, bool isLeft=false, GUIStyle style=null, bool backgrour
 {
  outCell = null;
  innerCell = null;

  if (style == null)
   style = UI.current.styles.foldoutBackground;

  if (backgroundWhileClosed || opened) Draw.Element(style);

  //Cell.EmptyLinePx(3);
  using (Cell.LineStd)
   using (Cell.Padded(padding,0,0,0))
```

```
  {
   Cell.current.trackChange = false;

   if (isLeft) Draw.FoldoutLeft(ref opened, label);
   else Draw.Foldout(ref opened, label);
  }

 if (opened)
 {
  outCell = Cell.Line;
  innerCell = Cell.Padded(padding + (isLeft? 10 : 0), 3, 0, 0); //Cell.Padded(padding + (isLeft? 10 : 0), 3, 

  //Do stuff
 }
}

public void Dispose ()
{
 //Do stuff

 Cell.EmptyLinePx(3);
 if (innerCell != null) innerCell.Dispose();
 if (outCell != null) outCell.Dispose();

 }
}
```

```
 //usage example:
 // using (Cell.LineStd)
 // {
 //  using (new Draw.FoldoutGroup(ref opened, "Foldout"))
 //  if (opened)
 //  {
 //   //do stuff
 //  }
 // }

#endregion

#region Scroll

 public struct ScrollGroup : IDisposable
 {
  Cell innerCell;
  ScrollZoom backupScrollZoom;
  Vector2 backupMousePos;

  public const float scrollWidth = 15;

  public ScrollGroup (ref float scroll, bool enabled=true)
  // if not enabled - skipping the scroll section
  {
```

```
if (!enabled)
 {innerCell=null; backupScrollZoom = UI.current.scrollZoom; backupMousePos = UI.current.mousePos;

float guizoom = UI.current.scrollZoom != null ? UI.current.scrollZoom.zoom : 1;
Vector2 guiscroll = UI.current.scrollZoom != null ? UI.current.scrollZoom.scroll : Vector2.zero;

Rect cellRect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);
Vector2 offset = Cell.current.worldPosition * guizoom;
Vector2 size = Cell.current.finalSize * guizoom;

backupScrollZoom = UI.current.scrollZoom;
UI.current.scrollZoom = new ScrollZoom();
UI.current.scrollZoom.zoom = backupScrollZoom.zoom;

backupMousePos = UI.current.mousePos;
UI.current.mousePos.y += scroll;
//Event.current.mousePosition = new Vector2(Event.current.);

innerCell = Cell.Custom(Vector2.zero, Cell.current.finalSize); //Cell.Custom(0,0,size.x-20,0); //Cell.Full;
innerCell.pixelSize.x -= scrollWidth;
innerCell.MakeStatic();
innerCell.isLayouted = false;

Rect internalRect = new Rect(offset, innerCell.InternalRect.size*guizoom);

if (!UI.current.layout)
```

```
    scroll = UnityEngine.GUI.BeginScrollView(cellRect, new Vector2(0,scroll), internalRect, alwaysShowHo

  }



  public void Dispose ()
  {
   if (innerCell != null)
   {
    if (!UI.current.layout)
     UnityEngine.GUI.EndScrollView();

    UI.current.scrollZoom = backupScrollZoom;
    UI.current.mousePos = backupMousePos;
    innerCell.Dispose();
   }
  }
 }

#endregion

#region DragValue

 public static float DragValue (float val)
 {
  float newVal = DragValueInternal(val, 0.01f);
```

```
 if (newVal > val + 0.000001f  ||  newVal < val - 0.000001f)
  UI.current.MarkChanged();

 return newVal;
}

public static int DragValue (int val)
{
 int newVal = (int)DragValueInternal(val, 1f, exponentiality:1000, sensitivity:5000);

 if (newVal != val)
  UI.current.MarkChanged();

 return newVal;
}

public static double DragValue (double val)
{
 double newVal = (double)DragValueInternal((float)val, 0.01f);

 if (newVal > val + 0.000001  ||  newVal < val - 0.000001)
  UI.current.MarkChanged();

 return newVal;
}
```

```
private static float DragValueInternal (float val, float minStep, float exponentiality=1, float sensitivity=1000
{
 if (UI.current.layout) return val;
 if (UI.current.optimizeElements && !UI.current.IsInWindow()) return val;
 if (Cell.current.inactive) return val;

 Cell cell = Cell.current;
 if (cell.disabled) return val;

 Rect rect = Cell.current.GetRect(UI.current.scrollZoom, padding:fieldPadding);
 rect.height += 2;

 //cursor
 UnityEditor.EditorGUIUtility.AddCursorRect (rect, UnityEditor.MouseCursor.SlideArrow);

 //dragging
 float newVal = val;

 if (DragDrop.TryStart(cell,  rect))
 {
  DragDrop.group = "DragField";
  origDragValue = val;

  //ChartWindow.Clear();
```

```
 //ChartWindow.Evaluate(x=>RaiseValue(origDragValue, x), 0, 10, 0.01f);
}

if (DragDrop.TryDrag(cell))
{
 float delta = DragDrop.totalDelta.x;

 delta = SwapCursor ((int)delta);

 newVal = RaiseValue(origDragValue, delta, exponentiality, sensitivity);
 newVal = RoundValue(newVal);

 UI.current.editorWindow?.Repaint();
 UI.RemoveFocusOnControl();
}

if (DragDrop.TryRelease(cell))
{
 #if UNITY_EDITOR_WIN
 swapTimes = 0;
 #endif

 UI.RemoveFocusOnControl();

 //UI.current.MarkChanged(); //changing on relase too to re-generate chunks in fullres
}
```

```
 return newVal;
}

public static float origDragValue;

public static float RaiseValue (float initialVal, float deltaSteps, float exponentiality=1, float sensitivity=100
{
 //converting exponential value to linear number of steps
 float sign = initialVal>=0? 1 : -1;
 float absVal = initialVal*sign;
 float steps = Mathf.Log(absVal+exponentiality, 2f) * sign;

 //modifying steps num
 steps += deltaSteps/sensitivity;

 //converting back to exponential
 sign = steps>=0? 1 : -1;
 float absSteps = steps*sign;
 return (Mathf.Pow(2f,absSteps)-exponentiality) * sign;
}

public static float RoundValue (float val, float minStep=0.01f)
{
 int sign = val>=0? 1 : -1;
 float absVal = val*sign;
```

```
 int step = 100;
 if (absVal > 10) step = 10;
 if (absVal > 100) step = 1;

 absVal = 1f*((int)(absVal*step)) / step;

 return (float)absVal*sign;
}


[DllImport("user32.dll")]
public static extern bool GetCursorPos(out Vector2Int lpPoint);

[DllImport("user32.dll")]
public static extern bool SetCursorPos(int x, int y);

#if UNITY_EDITOR_WIN
private static int swapTimes = 0;
#endif

public static float SwapCursor (int mouseX)
{
 #if UNITY_EDITOR_WIN
 Vector2Int screenMousePos;
 GetCursorPos(out screenMousePos);
```

```
 mouseX += swapTimes*Screen.currentResolution.width;

 if (screenMousePos.x == 0)
 {
  swapTimes --;
  SetCursorPos(Screen.currentResolution.width-2, screenMousePos.y);
 }

 if (screenMousePos.x == Screen.currentResolution.width-1)
 {
  swapTimes ++;
  SetCursorPos(1, screenMousePos.y);
 }
 #endif

 return mouseX;
}

public static void MoveCursorRight (float speed=10)
/// Slowly moves cursor to the right. Just to record videos
/// Speed - pixels per second
{
 double currentTime = (DateTime.Now-DateTime.Today).TotalMilliseconds;
 double deltaTime = currentTime - moveCursorTime;
```

```
    GetCursorPos(out Vector2Int mousePos);
    mousePos.x += (int)(deltaTime / 100f * speed + 0.5f);
    SetCursorPos(mousePos.x, mousePos.y);

    moveCursorTime = currentTime;
   }
   private static double moveCursorTime = -1;

  #endregion
 }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;

namespace Den.Tools.GUI
{
 public static class LayersEditor
 {
  const float stepAsideDist = 10; //the distance other layers step aside when dragging layer
  private static readonly RectOffset layerBackgroundOverflow = new RectOffset(0,0,0,1);

  static object dragLayerId = null; //the id of the system currently dragging. If null-nothing dragged. If not ma
  static object dragReleasedId = null;
  static int dragNum = -1;
  static int dragTo = -1; //dragNum;


  public static void DrawLayers<T> (
   ref T[] layers,
   Action<int> onDraw,
   Func<int,T> onCreate = null)
  {
   T[] newLayers = layers;
```

```
 DrawLayers(
  layers.Length,
  onDraw: onDraw,
  onAdd:n => ArrayTools.Insert(ref newLayers, n, onCreate!=null ? onCreate(n) : default ),
  onRemove: n => ArrayTools.RemoveAt(ref newLayers, n),
  onMove: (n,m) => ArrayTools.Switch(newLayers, n, m) );

 if (layers != newLayers)
  layers = newLayers;
}



public static void DrawLayers (
 int count,
 Action<int> onDraw,
 Action<int> onAdd = null,
 Action<int> onRemove = null,
 Action<int,int> onMove = null)
{
 //GUIStyle background = UI.current.textures.GetElementStyle(texturesFolder+"AddPanelEmpty");
 //Draw.Element(background);

 Cell layersCell = Cell.current;

 using (Cell.LinePx(0))
```

```
  DrawLayersThemselves(layersCell, count, onDraw);


 using (Cell.LinePx(20))
  DrawAddRemove(layersCell, onAdd, onRemove, onMove);
}

public static void DrawLayersThemselves (object id, int count, Action<int> onDraw)
/// Draws layers with the default backgrounds
{
 DrawLayersThemselves(id, count, onDraw,
  midBackground: UI.current.textures.GetElementStyle("DPUI/Layers/Mid", overflow:layerBackgroundOve
  topBackground: UI.current.textures.GetElementStyle("DPUI/Layers/Top", overflow:layerBackgroundOve
  botBackground: UI.current.textures.GetElementStyle("DPUI/Layers/Bot", overflow:layerBackgroundOver
  dragBackground: UI.current.textures.GetElementStyle("DPUI/Layers/Dragged", overflow:layerBackgroun
}


public static void DrawLayersThemselves (object id, int count, Action<int> onDraw, bool roundTop=true, b
/// Draws layers with the default backgrounds
{
 DrawLayersThemselves(id, count, onDraw,
  midBackground: UI.current.textures.GetElementStyle("DPUI/Layers/Mid", overflow:layerBackgroundOve
  topBackground: roundTop ?
   UI.current.textures.GetElementStyle("DPUI/Layers/Top", overflow:layerBackgroundOverflow) :
   UI.current.textures.GetElementStyle("DPUI/Layers/Mid", overflow:layerBackgroundOverflow),
  botBackground: roundBottom ?
```

```
   UI.current.textures.GetElementStyle("DPUI/Layers/Bot", overflow:layerBackgroundOverflow) :
   UI.current.textures.GetElementStyle("DPUI/Layers/BotSquare", overflow:layerBackgroundOverflow),
  dragBackground: UI.current.textures.GetElementStyle("DPUI/Layers/Dragged", overflow:layerBackgroun
}



public static void DrawLayersThemselves (
 object id,  //any equitable object that send in dragdrop. Should be the same as AddRemove button id. Us
 int count,
 Action<int> onDraw,
 GUIStyle dragBackground=null, GUIStyle topBackground=null, GUIStyle botBackground=null, GUIStyle m
{
 Cell cell = Cell.current;


 if (cell.subCounter != 0)
  throw new Exception("Using non-empty cell for layers");



 //space to insert dragged cell (stepAside)
 if (!UI.current.layout  &&  dragLayerId==id)
 {
  dragTo = FindDragTo();
  if (dragNum>=0) StepAside(dragNum, dragTo);
 }
```

```
//clearing all drag data after the full frame circle (i.e. before it was assigned with drag)
if (!UI.current.layout)
{
 if (dragLayerId == id)
 {
  dragLayerId = null;
  dragNum = -1;
 }
 if (dragReleasedId == id)
 {
  dragReleasedId = null;
  dragNum = -1;
 }
}


//drawing layers
for (int i=0; i<count; i++)
 using (Cell.LinePx(0))
{
 DragLayer(id, i);

 if (dragLayerId!=id || i!=dragNum)
  DrawLayer(i, count, onDraw, topBackground, botBackground, midBackground);

 if (dragLayerId==id && i==dragNum)
```

```
   DrawDraggedLayer(i, count, onDraw, dragBackground ?? midBackground);
 }
}



private static int FindDragTo ()
/// Finding where the layer is dragged using current cursor position
{
 if (UI.current.layout) return -1;

 Cell cell = Cell.current;
 int count = cell.subCells.Count;

 int to = dragNum;

 if (UI.current.mousePos.y < cell.worldPosition.y) return 0;
 else if (UI.current.mousePos.y > cell.worldPosition.y+cell.finalSize.y) return count-1;
 else
 {
  int num = 0; //using counter to skip dragged field
  for (int i=0; i<count; i++)
  {
   if (i == dragNum) continue; //cell is dragged - pos always within this cell

   Cell layerCell = cell.subCells[i];
```

```
   double start = layerCell.worldPosition.y;
   double mid = layerCell.worldPosition.y + layerCell.finalSize.y/2;
   double end = layerCell.worldPosition.y + layerCell.finalSize.y;

   if (i==0 && UI.current.mousePos.y <= mid) return 0;
   if (UI.current.mousePos.y > mid) to = num+1;

  // if (UI.mousePos.y >= start-stepAsideDist  &&  UI.mousePos.y < mid) { dragTo = num; break; }
  // if (UI.mousePos.y >= mid  &&  UI.mousePos.y < end+stepAsideDist) { dragTo = num+1; break; }

   num++;
  }
 }

 return to;
}


private static void StepAside (int dragNum, int dragTo)
/// Re-layouts cells creating an empty space. Shifting from dragTo to dragNum or vice versa
{
 if (UI.current.layout) return;
 if (dragNum<0 || dragTo<0) return; //happens when something went wrong with dragging

 Cell cell = Cell.current;
 int count = cell.subCells.Count;
```

```
 for (int i=Mathf.Min(dragNum,dragTo); i<count; i++)
 {
  Cell layerCell = cell.subCells[i];

  if (i>=dragTo && i<dragNum)
   layerCell.worldPosition.y += stepAsideDist;

  if (i<=dragTo && i>dragNum)
   layerCell.worldPosition.y -= stepAsideDist;

  layerCell.CalculateSubRects(); //re-layout cell
 }
}


private static void DragLayer (object id, int i)
{
 if (UI.current.layout) return;

 if (DragDrop.TryDrag((id,i), UI.current.mousePos))
 {
  Cell.current.worldPosition = DragDrop.initialRect.position + DragDrop.totalDelta;
  Cell.current.CalculateSubRects(); //re-layout cell

  dragLayerId = id;
```

```
  dragNum = i;
 }

 if (DragDrop.TryRelease((id,i), UI.current.mousePos))
 {
  dragReleasedId = id;
  dragNum = i;
 }

 if (DragDrop.TryStart((id,i), UI.current.mousePos, Cell.current.InternalRect))
 {
  dragLayerId = id;
  dragNum = i;
 }
}


private static void DrawLayer (int i, int count, Action<int> onDraw,
 GUIStyle topBackground=null, GUIStyle botBackground=null, GUIStyle midBackground=null)
{
 //background
 if (!UI.current.layout  &&  UI.current.textures != null)
 {
  GUIStyle style = null;
  if (i==0) style = topBackground ?? midBackground;
  else if (i==count-1) style = botBackground ?? midBackground;
```

```
  else style = midBackground;


  if (style!=null)
    Draw.Element(style);
 }

 //contents
 onDraw(i);
}



private static void DrawDraggedLayer (int i, int count, Action<int> onDraw, GUIStyle dragBackground)
{
 //drawing dragged in standard order
 if (!UI.current.layout)
 {
  //drag
  Cell.current.worldPosition = DragDrop.initialRect.position + DragDrop.totalDelta;
  //Cell.current.finalSize.x = cell.finalSize.x;
  Cell.current.CalculateSubRects(); //re-layout cell


  //background
  if (!UI.current.layout  &&  UI.current.textures != null  &&  dragBackground != null)
    Draw.Element(dragBackground);
 }
```

```
 onDraw(i);



 //and enqueue to draw once again
 Rect dragCellRect = Cell.current.InternalRect;
 int num = i; //to closure



 UI.current.DrawAfter(()=>
  {
   using (Cell.Custom(dragCellRect))
   {
    Cell.current.InternalRect = dragCellRect;
    Cell.current.CalculateSubRects(); //re-layout cell

    //background
    if (!UI.current.layout  &&  UI.current.textures != null  &&  dragBackground != null)
    Draw.Element(dragBackground);

    //contents
    onDraw(num);
   }
  }, 1);
}
```

```
public static void DrawAddRemove (
 object id,       //the same object as in DrawLayers
 Action<int> onAdd = null,
 Action<int> onRemove = null,
 Action<int,int> onMove = null)
{
 //add/remove
 bool draggedOnRemoveCell = false;

 Cell.EmptyRow();

 using (Cell.RowPx(70))
 {
  //Cell.current.worldPosition.y = cell.worldPosition.y + cell.finalSize.y - Cell.lineHeight;

  if (dragLayerId != id) //add when drag is disabled
  {
   Texture2D buttonIcon = UI.current.textures.GetTexture("DPUI/Layers/Add");
   if (Draw.Button(buttonIcon, visible:false))
   {
    UI.current.MarkChanged();
    onAdd(0);

    Event.current.Use(); //gui structure changed. Not necessary since button is used
   }
  }
```

```
 else //displaying remove when drag is enabled
 {
  Rect cellRect = Cell.current.InternalRect;
  draggedOnRemoveCell = cellRect.Contains(UI.current.mousePos);

  UI.current.DrawAfter(()=>
  {
   using (Cell.Custom(cellRect))
   {
    Cell.current.InternalRect = cellRect;

    Draw.Icon( draggedOnRemoveCell ?
     UI.current.textures.GetTexture("DPUI/Layers/RemoveBright") :
     UI.current.textures.GetTexture("DPUI/Layers/RemoveDark") );
   }
  }, 2);
 }
}

Cell.EmptyRowPx(2);

//releasing drag
if (dragReleasedId==id && dragNum>=0 && dragTo>=0)
{
 if (draggedOnRemoveCell)
```

```
  {
   UI.current.MarkChanged();
   onRemove(dragNum); //!invert ? dragNum : count-1-dragNum;

   Event.current.Use(); //to remove extra child in flush
  }

  else if (onMove != null && dragNum!=dragTo)  //switch is the last - it conflicts with remove
  {
   UI.current.MarkChanged();
   onMove(dragNum, dragTo); //!invert ? dragNum : count-1-dragNum,  //!invert ? dragTo : count-1-dragTo
  }

  //using dragged object
  dragReleasedId = null; dragNum = -1;

  //clearing after-draw (or will try to draw removed layer)
  UI.current.ClearDrawAfter();
 }
}


public static void DrawAddRemove (
 object id,
 string label,
 Action<int> onAdd = null,
```

```
    Action<int> onRemove = null,
    Action<int,int> onMove = null)
   {
    Cell.EmptyLine();
    using (Cell.LinePx(18)) Draw.Label(label); //placing in the center of cell to match the button font
    Cell.EmptyLine();

    using (Cell.Full)
     DrawAddRemove(id, onAdd, onRemove, onMove);
   }
  }
 }
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;

namespace Den.Tools.GUI
{
 public struct Padding
 {
  public float left;
  public float right;
  public float top;
  public float bottom;



  public Padding (float left, float right, float top, float bottom)
  {
   this.left = left;
   this.right = right;
   this.top = top;
   this.bottom = bottom;
  }

  public Padding (float hor, float vert)
```

```
  {
   this.left = hor;
   this.right = hor;
   this.top = vert;
   this.bottom = vert;
  }

  public Padding (float offset)
  {
   this.left = offset;
   this.right = offset;
   this.top = offset;
   this.bottom = offset;
  }
 }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections.Generic;
using UnityEditor;

namespace Den.Tools.GUI.Popup
{

 public class PopupMenu : UnityEditor.PopupWindowContent
  {



   public int minWidth = 100;


   public const int verticalOffset = 5; //HACK: first 5 pixels of popup window could not be clicked in Unity
   public const int verticalOffsetTmp = 4;


   static GUIStyle blackLabel;


   static private Texture2D background;
   static private Texture2D highlight;
   static private Texture2D triangle;
   static private Texture2D separator;


   public List<Item> items;
   public bool sortItems = true;
```

```
private Item lastItem;
private System.DateTime lastTimestart;
//private bool timeUsed = false;

private Item expandedItem;

private PopupMenu parent;

private PopupMenu expandedWindow = null;

private static Action nextFrameShow; //hack to show the window next frame on click

//void CloseMenuIfNotFocused () { if (UnityEditor.EditorWindow.focusedWindow.GetType() != typeof(Pop
//void OnEnable () { UnityEditor.EditorApplication.update += CloseMenuIfNotFocused; }
//void OnDisable () { UnityEditor.EditorApplication.update -= CloseMenuIfNotFocused; }



/*static public PopupMenu DrawPopup (List<Item> items, Vector2 pos, bool closeAllOther=false, bool so
{
 if (sort) Item.SortItems(items);
 PopupMenu popupWindow = new PopupMenu();
 popupWindow.items = items;
 popupWindow.minWidth = minWidth;
```

```
 popupWindow.parent = parent;
 PopupWindow.Show(new Rect(pos.x,pos.y, minWidth, 0), popupWindow);
 return popupWindow;
}*/

public void Show (Vector2 pos)
{
 if (sortItems) Item.SortItems(items);
 PopupWindow.Show(new Rect(pos.x,pos.y- verticalOffset, minWidth, 0), this);
}

public override Vector2 GetWindowSize()
{
 float height = 0;
 float width = 0;
 int count = items.Count;
 for (int i=0; i<count; i++)
 {
  height += items[i].height;
  if (items[i].width > width) width = items[i].width;
 }

 if (width != minWidth) width = minWidth;

 return new Vector2(width, height+ verticalOffsetTmp);
}
```

```
public Rect GetIconRect (Rect srcRect, Texture2D texture)
{
 Vector2 center = srcRect.center;
 return new Rect(
  center.x - texture.width/2f,
  center.y - texture.height/2f,
  texture.width,
  texture.height);
}

public override void OnGUI(Rect rect)
{
 //showing window next frame
 Action tmp = nextFrameShow;
 nextFrameShow = null; //because we can't null it after it has been called (new window will be started)
 tmp?.Invoke();

 //preparing textures
 if (background==null)
 {
  background = new Texture2D(1, 1, TextureFormat.RGBA32, false);
  background.SetPixel(0, 0, new Color(0.98f, 0.98f, 0.98f));
  background.Apply();
 }
```

```
if (highlight==null)
{
 highlight = new Texture2D(1, 1, TextureFormat.RGBA32, false);
 highlight.SetPixel(0, 0, new Color(0.6f, 0.7f, 0.9f));
 highlight.Apply();
}

Vector2 size = GetWindowSize();
Vector2 pos = new Vector2(0, verticalOffset);

//background
//if (Event.current.type == EventType.repaint) GUI.skin.box.Draw(fullRect, false, true, true, false);
UnityEngine.GUI.DrawTexture(new Rect(pos, size), background, ScaleMode.StretchToFill);

//list
float currentHeight = verticalOffsetTmp;
int itemsCount = items.Count;
for (int i=0; i<itemsCount; i++)
{
 Item currentItem = items[i];

 //rects
 Rect lineRect = new Rect(1, currentHeight+1, size.x-2, currentItem.height-2);
 currentHeight += currentItem.height;

 Rect offsetRect = new Rect(lineRect.x, lineRect.y, Item.lineHeight*currentItem.offset, lineRect.height);
```

```
Rect labelRect = new Rect(lineRect.x+offsetRect.width+3, lineRect.y+1, lineRect.width-offsetRect.width

//background
bool highlighted = lineRect.Contains(Event.current.mousePosition);
if (currentItem.disabled) highlighted = false;
if (highlighted) UnityEngine.GUI.DrawTexture(lineRect, highlight);
/*{
 //GUIStyle style = texturesCache.GetElementStyle(tex);
 //if (Event.current.type == EventType.Repaint) style.Draw(leftRect, false, false, false ,false);

 GUIStyle style = new GUIStyle();
 style.normal.background = highlight;
 style.border = new RectOffset(highlight.width/2, highlight.width/2, highlight.height/2, highlight.height/2);

 if (Event.current.type == EventType.Repaint) style.Draw(lineRect, false, false, false ,false);
}*/

//clicking
bool clicked = Event.current.rawType == EventType.MouseUp && Event.current.button == 0;
if (highlighted && clicked)
{
 currentItem.onClick?.Invoke();
 CloseRecursive();
 Event.current.Use();
}
```

```
//label
UnityEditor.EditorGUI.BeginDisabledGroup(currentItem.disabled);
//if (blackLabel == null) { blackLabel = new GUIStyle(UnityEditor.EditorStyles.label); blackLabel.normal.t
if (currentItem.onDraw != null)
 currentItem.onDraw(currentItem,lineRect);
else
 EditorGUI.LabelField(labelRect, currentItem.name);
UnityEditor.EditorGUI.EndDisabledGroup();

//separator
if (currentItem.isSeparator)
{
 if (currentItem.onDraw == null)
 {
  Rect separatorRect = new Rect(lineRect.x+3, lineRect.y, lineRect.width-6, 1);
  if (separator == null) separator = TextureExtensions.ColorTexture(2,2,new Color(0.3f, 0.3f, 0.3f, 1));
  UnityEngine.GUI.DrawTexture(separatorRect, separator, ScaleMode.ScaleAndCrop);
 }
 else
  currentItem.onDraw(currentItem, lineRect);
}

//chevron
if (currentItem.hasSubs)
{
 Rect rightRect = lineRect; rightRect.width = 10; rightRect.height = 10;
```

```
 rightRect.x = lineRect.x + lineRect.width - rightRect.width; rightRect.y = lineRect.y + lineRect.height/2 -
//UnityEditor.EditorGUI.LabelField(rightRect, "\u25B6");
if (triangle == null) triangle = Resources.Load("DPUI/Chevrons/SmallRight") as Texture2D;
UnityEngine.GUI.DrawTexture(GetIconRect(rightRect, triangle), triangle, ScaleMode.ScaleAndCrop);

//opening subsmenus
if (highlighted)
{
 //starting timer on selected item change
 if (currentItem != lastItem)
 {
  lastTimestart = System.DateTime.Now;
  lastItem = currentItem;
 }

 //when holding for too long
 double highlightTime = (System.DateTime.Now-lastTimestart).TotalMilliseconds;
 if ((highlightTime > 150 && expandedItem != currentItem) || clicked)
 {
  //re-opening expanded window
  if (expandedWindow != null && expandedWindow.editorWindow != null)
   expandedWindow.editorWindow.Close();

  expandedWindow = new PopupMenu() {
   items = currentItem.subItems,
   minWidth = minWidth,
```

```
      parent = this };
     expandedItem = currentItem;

     //nextFrameShow = () => expandedWindow.Show(lineRect.max-new Vector2(0,currentItem.height));
     expandedWindow.Show(lineRect.max-new Vector2(0,currentItem.height));
     editorWindow.Focus();

     //if (currentItem.subItems != null) PopupWindow.Show(new Rect(lineRect.max-new Vector2(0,currentI
    }
   }
  }
 }

 //#if (!UNITY_EDITOR_LINUX)
 this.editorWindow.Repaint();
 //#endif
}

public override void OnClose ()
{
 base.OnClose();

 if (parent != null && parent.expandedWindow == this)
 {
  parent.expandedWindow = null;
  parent.expandedItem = null;
```

```
    }

    //closing all of the expanded windows too
    if (expandedWindow != null && expandedWindow.editorWindow != null)
     expandedWindow.editorWindow.Close();
   }

   public void CloseRecursive ()
   {
    if (parent != null) parent.CloseRecursive();
    editorWindow.Close();
   }
  }

}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace Den.Tools.GUI
{
 [System.Serializable]
 public class ScrollZoom
 {
  public float zoom = 1;
  //public int zoomStage = 0; //number of scroll wheel ticks from zoom 1
  public float zoomStep = 0.0625f;
  public float minZoom = 0.25f;
  public float maxZoom = 2;
  public bool allowZoom = false; //read externally before zooming

  public int scrollWheelStep = 18;

  public Vector2 scroll = new Vector2(0, 0);
  public bool isScrolling = false;
  private Vector2 clickPos = new Vector2(0,0);
  private Vector2 clickScroll = new Vector2(0,0);
  public int scrollButton = 2;
  public bool roundScroll = true;
  public bool allowScroll = false;
```

```
public void Zoom ()
{
 if (!allowZoom) return;
 Zoom(Event.current.mousePosition);
}

public void Zoom (Vector2 pivot)
/// Zooms with mouse wheel
{

 if (Event.current == null) return;

 //reading control
 #if UNITY_EDITOR_OSX
 bool control = Event.current.command;
 #else
 bool control = Event.current.control;
 #endif

 float delta = 0;
 if (Event.current.type == EventType.ScrollWheel) delta = Event.current.delta.y / 3f;
 //else if (Event.current.type == EventType.MouseDrag && Event.current.button == 0 && control) delta = E
 //else if (control && Event.current.alt && Event.current.type==EventType.KeyDown && Event.current.key
 //else if (control && Event.current.alt && Event.current.type==EventType.KeyDown && Event.current.key
 if (Mathf.Abs(delta) < 0.001f) return;
```

```
//calculating current zoom stage - number of scroll wheel ticks from zoom 1
int zoomStage = 0;
if (zoom < 0.999f)
 zoomStage = -(int)((1-zoom) / zoomStep);
else if (zoom > 1.0001f)
{
 float tempZoom = 1;
 while (tempZoom < zoom)
 {
  tempZoom += zoomStep * (int)(tempZoom / 2 + 1) * 2;
  zoomStage ++;
 }
}

//new zoom
if (delta > 0) zoomStage--;
if (delta < 0) zoomStage++;

float newZoom = 1;
if (zoomStage < 0)
{
 int minStage = -(int)((1-minZoom) / zoomStep);
 if (zoomStage < minStage) zoomStage = minStage;
 newZoom = 1 - zoomStep*(-zoomStage);
}
```

```
 else if (zoomStage > 0)
 {
  for (int i=0; i<zoomStage; i++)
  {
   float val = zoomStep * (int)(newZoom / 2 + 1) * 2;
   if (newZoom + val > maxZoom + 0.0001f)
    break;
   newZoom += val;
  }
 }

 Zoom(newZoom, pivot);
}

public void Zoom (float zoomVal, Vector2 pivot)
/// Zooms to zoomVal
/// Pivot is usually mouse position
{
 //record mouse position in worldspace
 Vector2 worldMousePos = (pivot - scroll) / zoom;

 //changing zoom
 float zoomChange = zoomVal - zoom;
 zoom = zoomVal;

 //scrolling around pivot
```

```
 scroll -= worldMousePos * zoomChange;

 #if UNITY_EDITOR
 //UnityEditor.EditorWindow.focusedWindow?.Repaint();
 UI.current?.editorWindow?.Repaint();
 #endif

 if (roundScroll) RoundScroll();
}


public void ScrollWheel(int step = 3)
{
 float delta = 0;
 if (Event.current.type == EventType.ScrollWheel) delta = Event.current.delta.y / 3f;
 scroll.y -= delta * scrollWheelStep * step;
}


public void Scroll()
{
 if (!allowScroll) return;

 if (Event.current.type == EventType.MouseDown  &&  Event.current.button == scrollButton)
 {
  clickPos = Event.current.mousePosition;
```

```
 clickScroll = scroll;
 isScrolling = true;
}

if (Event.current.type == EventType.MouseDown  &&  Event.current.button == 0  &&  Event.current.alt)  /
{
 clickPos = Event.current.mousePosition;
 clickScroll = scroll;
 isScrolling = true;
}

if (UI.MouseUp) //(Event.current.rawType == EventType.MouseUp  ||  Event.current.rawType == EventTy
{
 isScrolling = false;
}

if (isScrolling)
 Scroll(clickScroll + Event.current.mousePosition - clickPos);
}


public void Scroll (Vector2 newScroll)
{
 scroll = newScroll;

 if (roundScroll) RoundScroll();
```

```
 UI.RemoveFocusOnControl(); //disabling text typing
 UI.current?.editorWindow?.Repaint();
}



public void RoundScroll ()
{
 float dpiFactor = UI.current?.DpiScaleFactor ?? 1;
 if (scroll.x < 0) scroll.x--; scroll.x = (int)(float)(scroll.x*dpiFactor + 0.50002f) / dpiFactor;   //adding epsilon
 if (scroll.y < 0) scroll.y--; scroll.y = (int)(float)(scroll.y*dpiFactor + 0.50002f) / dpiFactor;
}



public Vector2 GetWindowCenter (Vector2 windowSize)
/// returns the center of the screen(window) in base coordinates
{
 //return (windowSize/2 - scroll) / zoom;
 return  (windowSize/2) - scroll/zoom;
}


public void FocusWindowOn (Vector2 center, Vector2 windowSize)
{
 //Scroll( -center*zoom + windowSize/2 );
 Scroll( (windowSize/2 - center) * zoom );
}
```

```
#region ToScreen/ToInternal

 public Vector2 ToScreen(Vector2 pos)
 {
  return new Vector2(pos.x * zoom + scroll.x, pos.y * zoom + scroll.y);
 }

 public Rect ToScreen (Vector2 pos, Vector2 size, bool pixelPerfect=true)
 {
  return ToScreen(pos.x, pos.y, size.x, size.y, pixelPerfect);
 }

 public Rect ToScreen (Rect rect, bool pixelPerfect=true)
 {
  return ToScreen(rect.x, rect.y, rect.width, rect.height, pixelPerfect);
 }

 public Rect ToScreen (double minX, double minY, double sizeX, double sizeY,  bool pixelPerfect=false, b
 {
  float dpiFactor = UI.current.DpiScaleFactor;
  minX*=dpiFactor; minY*=dpiFactor; sizeX*=dpiFactor; sizeY*=dpiFactor;

  minX = minX*zoom + scroll.x*dpiFactor;
  minY = minY*zoom + scroll.y*dpiFactor;
```

```
sizeX *= zoom;
sizeY *= zoom;

if (pixelPerfect)
{
 double maxX = minX + sizeX;
 double maxY = minY + sizeY;

 if (minX < 0) minX--; minX = (int)(minX + 0.50002f);  //adding epsilon to prevent position flickering on cle
 if (minY < 0) minY--; minY = (int)(minY + 0.50002f);
 if (maxX < 0) maxX--; maxX = (int)(maxX + 0.50002f);
 if (maxY < 0) maxY--; maxY = (int)(maxY + 0.50002f);

 sizeX = maxX - minX;
 sizeY = maxY - minY;
}

if (sizePerfect)
{
 if (minX < 0) minX--; minX = (int)(minX + 0.50002f);  //adding epsilon to prevent position flickering on cle
 if (minY < 0) minY--; minY = (int)(minY + 0.50002f);
 if (sizeX < 0) sizeX--; sizeX = (int)(sizeX + 0.50002f);
 if (sizeY < 0) sizeY--; sizeY = (int)(sizeY + 0.50002f);
}

return new Rect((float)minX/dpiFactor, (float)minY/dpiFactor, (float)sizeX/dpiFactor, (float)sizeY/dpiFacto
```

```
}



public Rect ToInternal(Rect rect)
{
 Vector2 offset = new Vector2(
  (rect.x - scroll.x) / zoom,
  (rect.y - scroll.y) / zoom );
 Vector2 size = new Vector2(
  rect.width / zoom,
  rect.height / zoom);
 return new Rect(offset, size);
}

public Vector2 ToInternal(Vector2 pos)
{
 return new Vector2 (
  (pos.x - scroll.x) / zoom,
  (pos.y - scroll.y) / zoom );
}

public float ToInternal(float val)
{
 return val / zoom;
}
```

```
 public Vector2 RoundToZoom (Vector2 vec)
 /// Queer thing
 {
  vec.x /= zoom;
  vec.x = Mathf.Round(vec.x);
  vec.x *= zoom;

  vec.y /= zoom;
  vec.y = Mathf.Round(vec.y);
  vec.y *= zoom;

  return vec;
 }



#endregion









/*public Rect ToDisplay(Rect rect)
```

```
{
 return new Rect(rect.x * zoom + scroll.x, rect.y * zoom + scroll.y, rect.width * zoom, rect.height * zoom);
}

public Rect ToDisplay(float offsetX, float offsetY, float sizeX, float sizeY)
{
 return new Rect(offsetX * zoom + scroll.x, offsetY * zoom + scroll.y, sizeX * zoom, sizeY * zoom);
}

public Rect ToDisplay(Float2 pos, Float2 size)
{
 if (paddingBox==null) return new Rect(
  (int)( pos.x * zoom + scroll.x  + 0.5f),
  (int)( pos.y * zoom + scroll.y  + 0.5f),
  (int)( size.x * zoom  + 0.5f),
  (int)( size.y * zoom  + 0.5f) );

 else
 {
  Padding padding = (Padding)paddingBox;
  return new Rect(
  (int)( (pos.x + padding.left) * zoom + scroll.x  + 0.5f),
  (int)( (pos.y + padding.top) * zoom + scroll.y   + 0.5f),
  (int)( (size.x - (padding.left+padding.right)) * zoom  + 0.5f),
  (int)( (size.y - (padding.top+padding.bottom)) * zoom  + 0.5f));
 }
```

```
  }

  public Rect ToInternal(Rect rect)
  {
   return new Rect((rect.x - scroll.x) / zoom, (rect.y - scroll.y) / zoom, rect.width / zoom, rect.height / zoom);
  }

  public Vector2 ToInternal(Vector2 pos)
  {
   return (pos - scroll) / zoom;
  } //return new Vector2( (pos.x-scroll.x)/zoom, (pos.y-scroll.y)/zoom); }


  public void Focus(Cell cell, Vector2 pos)
  {
   throw new System.NotImplementedException();
  }*/
 }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;

namespace Den.Tools.GUI
{
 public class StylesCache
 {
  #if UNITY_2019_3_OR_NEWER
  public const int defaultFontSize = 12;
  #else
  public const int defaultFontSize = 11;
  #endif

  public const int defaultToolbarFontSize = 9;

  private bool initializedAsPro; //is currently initialized as pro skin
  public static bool isPro; //= UnityEditor.EditorGUIUtility.isProSkin

  public Color FontColor
  {get{
   return isPro ?  new Color(1, 1, 1, 0.7f) : Color.black;
  }}
```

```
public GUIStyle label;
public GUIStyle boldLabel;
public GUIStyle centerLabel;
public GUIStyle boldMiddleCenterLabel;
public GUIStyle middleLabel;
public GUIStyle middleBlackLabel;
public GUIStyle rightLabel;
public GUIStyle topLabel;
public GUIStyle middleCenterLabel;
public GUIStyle smallLabel;
public GUIStyle tinyLabel;
public GUIStyle bigLabel;
public GUIStyle blackLabel;
public GUIStyle whiteLabel;
public GUIStyle whiteMiddleLabel;
public GUIStyle url;
public GUIStyle foldout;
public GUIStyle foldoutBackground;
public GUIStyle foldoutOpaque;
public GUIStyle field;
public GUIStyle checkbox;
public GUIStyle button;
public GUIStyle enumClose;
public GUIStyle toolbar;
```

```
public GUIStyle toolbarButton;
public GUIStyle toolbarHoverButton;
public GUIStyle toolbarField;
public GUIStyle helpBox;
public GUIStyle progressBarBackground;

public static Texture2D blankTex;
public static Texture2D pencilTex;
public static Texture2D diamonTex;
public static Texture2D debugTex;
public static Texture2D progressBarFill;
public static Texture2D enumSign;
//public static Texture2D toggleTex;
public static Texture2D objectPickerTex;


public void CheckInit ()
{
 isPro = UnityEditor.EditorGUIUtility.isProSkin;

 if (label == null  ||  initializedAsPro != isPro  ||  bigLabel.font == null)
  Init();
}

public void Init ()
{
```

```
lastZoom = 1; //resetting zoom cache since all styles will be rescaled

label = new GUIStyle(UnityEditor.EditorStyles.label);
label.normal.textColor = label.focused.textColor = label.active.textColor = FontColor; //no focus
label.fontSize = defaultFontSize;

boldLabel = new GUIStyle(label);
boldLabel.fontStyle = FontStyle.Bold;

centerLabel = new GUIStyle(label);
centerLabel.alignment = TextAnchor.LowerCenter;

middleLabel = new GUIStyle(label);
middleLabel.alignment = TextAnchor.MiddleLeft;

middleBlackLabel = new GUIStyle(label);
middleBlackLabel.alignment = TextAnchor.MiddleLeft;
middleBlackLabel.active.textColor = middleBlackLabel.normal.textColor =  middleBlackLabel.focused.tex

rightLabel = new GUIStyle(label);
rightLabel.alignment = TextAnchor.MiddleRight;

topLabel = new GUIStyle(label);
topLabel.alignment = TextAnchor.UpperLeft;

middleCenterLabel = new GUIStyle(label);
```

```
middleCenterLabel.alignment = TextAnchor.MiddleCenter;

boldMiddleCenterLabel = new GUIStyle(boldLabel);
boldMiddleCenterLabel.alignment = TextAnchor.MiddleCenter;

smallLabel = new GUIStyle(label);
smallLabel.fontSize = 9;

tinyLabel = new GUIStyle(label);
tinyLabel.fontSize = 6;

bigLabel = new GUIStyle(label);
bigLabel.font = Font.CreateDynamicFontFromOSFont("Verdana", 16);
bigLabel.fontSize = 16;
//style.fontStyle = FontStyle.Bold;
bigLabel.alignment = TextAnchor.MiddleLeft;
bigLabel.contentOffset = new Vector2(0,-2);

blackLabel = new GUIStyle(label);
blackLabel.active.textColor = blackLabel.normal.textColor =  blackLabel.focused.textColor = Color.black;

whiteLabel = new GUIStyle(label);
whiteLabel.active.textColor = whiteLabel.normal.textColor =  whiteLabel.focused.textColor = Color.white;

whiteMiddleLabel = new GUIStyle(whiteLabel);
whiteMiddleLabel.alignment = TextAnchor.MiddleCenter;
```

```
url = new GUIStyle(label);
url.normal.textColor = new Color(0.3f, 0.5f, 1f);

foldout = new GUIStyle(UnityEditor.EditorStyles.foldout);
foldout.fontSize = defaultFontSize;
foldout.fontStyle = FontStyle.Bold;
foldout.focused.textColor = foldout.active.textColor = foldout.onActive.textColor = FontColor;

foldoutBackground = new GUIStyle();
Texture2D foldoutBackTex = TexturesCache.LoadTextureAtPath("DPUI/Backgrounds/Foldout");
foldoutBackground.normal.background = foldoutBackTex;
foldoutBackground.border = new RectOffset(4,4,4,4);

foldoutOpaque = new GUIStyle(foldoutBackground);
Texture2D foldoutOpaqueTex = TexturesCache.LoadTextureAtPath("DPUI/Backgrounds/FoldoutOpaque
foldoutOpaque.normal.background = foldoutOpaqueTex;

field = new GUIStyle(UnityEditor.EditorStyles.numberField);
field.fontSize = defaultFontSize;
Texture2D fieldTexture = TexturesCache.LoadTextureAtPath("DPUI/Backgrounds/Field");
#if !UNITY_2019_3_OR_NEWER
field.normal.background = field.active.background = field.focused.background = fieldTexture;
field.border = new RectOffset(4,4,4,4);
#endif
field.normal.textColor = field.focused.textColor = field.active.textColor = FontColor;
```

```
checkbox = new GUIStyle(UnityEditor.EditorStyles.numberField);
checkbox.fontSize = defaultFontSize;
Texture2D toggleCheckedTex = TexturesCache.LoadTextureAtPath("DPUI/Toggle/Checked");
Texture2D toggleCheckedActiveTex = TexturesCache.LoadTextureAtPath("DPUI/Toggle/CheckedActive
Texture2D toggleUncheckedTex = TexturesCache.LoadTextureAtPath("DPUI/Toggle/Unchecked");
Texture2D toggleUncheckedActiveTex = TexturesCache.LoadTextureAtPath("DPUI/Toggle/UncheckedA
checkbox.normal.background = checkbox.focused.background = toggleUncheckedTex;
checkbox.onNormal.background = checkbox.onFocused.background = toggleCheckedTex;
checkbox.onActive.background = toggleCheckedActiveTex;
checkbox.active.background = toggleUncheckedActiveTex;
checkbox.border = new RectOffset(1,1,1,1);
checkbox.overflow = new RectOffset(-1,-1,-1,-1);
checkbox.normal.textColor = checkbox.focused.textColor = checkbox.active.textColor = FontColor;

button = new GUIStyle("Button");
button.fontSize = defaultFontSize;
button.normal.textColor = button.focused.textColor = button.active.textColor = FontColor; //no focus
Texture2D buttonTexture = TexturesCache.LoadTextureAtPath("DPUI/Backgrounds/Button");
Texture2D buttonPressedTexture = TexturesCache.LoadTextureAtPath("DPUI/Backgrounds/ButtonPress
Texture2D buttonActiveTexture = TexturesCache.LoadTextureAtPath("DPUI/Backgrounds/ButtonActive"
button.normal.background = buttonTexture;
button.onNormal.background = buttonPressedTexture;
button.border = new RectOffset(2,2,2,2);
button.overflow.bottom = 0;
```

```
//#if !UNITY_2019_3_OR_NEWER
enumClose = new GUIStyle(button);
enumClose.alignment = TextAnchor.LowerLeft;
enumClose.fontSize = defaultFontSize; //defaultToolbarFontSize;
enumClose.overflow.bottom = 0;
//#else
//enumClose = new GUIStyle(UnityEditor.EditorStyles.popup);
//#endif

toolbar = new GUIStyle(UnityEditor.EditorStyles.toolbar);
toolbar.overflow = new RectOffset(0,0,0,0);
toolbar.padding = new RectOffset(0,0,0,0);
toolbar.fixedHeight = 0;

toolbarButton = new GUIStyle(UnityEditor.EditorStyles.toolbarButton);
toolbarButton.overflow = new RectOffset(0,0,1,0);
//style.padding = new RectOffset(0,0,10,0);
//style.margin = new RectOffset(0,0,10,0);
toolbarButton.fixedHeight = 0;
toolbarButton.fontSize = defaultToolbarFontSize;

toolbarHoverButton = new GUIStyle(button);
toolbarHoverButton.border = toolbarHoverButton.border;
toolbarHoverButton.normal.background = toolbarHoverButton.normal.background;
//style.active.background = style.normal.background;
toolbarHoverButton.normal.textColor = toolbarHoverButton.focused.textColor = toolbarHoverButton.activ
```

```
toolbarHoverButton.fontSize = defaultToolbarFontSize;

toolbarField = new GUIStyle(field);
toolbarField.alignment = TextAnchor.MiddleLeft;
//toolbarField.fontSize = (int)(defaultToolbarFontSize);

helpBox = new GUIStyle(UnityEditor.EditorStyles.helpBox);
helpBox.fontSize = defaultToolbarFontSize;

progressBarBackground = new GUIStyle(UnityEditor.EditorStyles.helpBox);
progressBarBackground.normal.background = TexturesCache.LoadTextureAtPath("DPUI/ProgressBar/B
progressBarBackground.border = new RectOffset(1,1,1,1);

blankTex = TexturesCache.LoadTextureAtPath("DPUI/Backgrounds/White");
pencilTex = TexturesCache.LoadTextureAtPath("DPUI/Icons/Pencil");
diamonTex = TexturesCache.LoadTextureAtPath("DPUI/Icons/Circle");
debugTex = TexturesCache.LoadTextureAtPath("DPUI/Backgrounds/Gradient");
progressBarFill = TexturesCache.LoadTextureAtPath("DPUI/ProgressBar/Fill");
enumSign = TexturesCache.LoadTextureAtPath("DPUI/Icons/Enum");
//toggleTex = TexturesCache.LoadTextureAtPath("DPUI/Icons/Toggle");
objectPickerTex = TexturesCache.LoadTextureAtPath("DPUI/Icons/ObjectPicker");

PopulateFontSizes();
initializedAsPro = isPro;
}
```

```
public static void Reload ()
{
 System.Runtime.CompilerServices.RuntimeHelpers.RunClassConstructor(typeof(StylesCache).TypeHan
}

private Dictionary<GUIStyle,float> fontsSizes = new Dictionary<GUIStyle, float>();
private void PopulateFontSizes ()
{
 FieldInfo[] fields = typeof(StylesCache).GetFields(BindingFlags.Instance | BindingFlags.Public | BindingF
 for (int i=0; i<fields.Length; i++)
 {
  if (fields[i].FieldType != typeof(GUIStyle)) continue;
  GUIStyle style = fields[i].GetValue(this) as GUIStyle;
  if (style.fontSize == 0) continue;
  fontsSizes.Add(style, style.fontSize);
 }
}

private float lastZoom = 1;
public void Resize (float zoom=-1)
{
 if (zoom == -1) zoom = lastZoom;
 if (zoom == lastZoom) return;
 lastZoom = zoom;
```

```
   foreach (var kvp in fontsSizes)
   {
    GUIStyle style = kvp.Key;


    if (fontsSizes.TryGetValue(style, out float fs))
     style.fontSize = Mathf.RoundToInt(fs * zoom);
   }
  }

  private Dictionary<string,GUIStyle> styles = new Dictionary<string,GUIStyle>();
  //private Dictionary<GUIStyle,float> customFontsSize = new Dictionary<GUIStyle, float>();
 }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;

namespace Den.Tools.GUI
{
 public class TexturesCache
 {
  private Dictionary<string,Texture2D> textures = new Dictionary<string,Texture2D>();
  private Dictionary<string, Dictionary<uint,Texture2D>> colorTextures = new Dictionary<string, Dictionary<
  private Dictionary<uint, Texture2D> blankTextures = new Dictionary<uint, Texture2D>();
  private Dictionary<Texture2D,GUIStyle> styles = new Dictionary<Texture2D,GUIStyle>();

  private Dictionary<string,Material> materials = new Dictionary<string, Material>();

  public bool forcePro = false;
  public bool forceLight = false;


  public Texture2D GetTexture (string textureName)
  {
   if (textures.TryGetValue(textureName, out Texture2D tex)) return tex;
   else
```

```
 {
  tex = LoadTextureAtPath(textureName);
  textures.Add(textureName, tex);
  return tex;
 }
}

public Texture2D GetTexture (string texturePath, string textureName, bool forceLight=false, bool forcePro
{
 if (textures.TryGetValue(textureName, out Texture2D tex)) return tex;
 else
 {
  tex = LoadTextureAtPath(texturePath, forceLight:forceLight, forcePro:forcePro);
  textures.Add(textureName, tex);
  return tex;
 }
}

private uint ColorHash (Color color)
/// Using manual color hash since color in dictionary (compare) creates garbage for some reason
{
 uint r = (uint)(color.r*255);
 uint g = (uint)(color.g*255);
 uint b = (uint)(color.b*255);
 uint a = (uint)(color.a*255);
```

```
 return r<<24 | g<<16 | b<<8 | a;
}

public Texture2D GetColorizedTexture (string textureName, Color color)
{
 if (!colorTextures.TryGetValue(textureName, out var dict))
 {
  dict = new Dictionary<uint, Texture2D>();
  colorTextures.Add(textureName, dict);
 }

 uint colorHash = ColorHash(color);

 if (dict.ContainsKey(colorHash)  &&  !(dict[colorHash]==null || dict[colorHash].Equals(null)))  //texture is re
  return dict[colorHash];

 //else
 {
  Texture2D source = GetTexture(textureName);
  Texture2D clone = source.Clone();
  clone.Multiply(color, multiplyAlpha:false);

  if (!dict.ContainsKey(colorHash)) dict.Add(colorHash, clone);
  else dict[colorHash] = clone;

  return clone;
```

```
 }



}



public Texture2D GetBlankTexture (float color)
 { return GetBlankTexture( new Color(color, color, color)); }



public Texture2D GetBlankTexture (Color color)
{
 uint colorHash = ColorHash(color);

 blankTextures.TryGetValue(colorHash, out Texture2D tex); //if(TryGet) //texture could be removed, tryGe
 if (tex != null) return tex;
 else
 {
  tex = new Texture2D(4,4);
  Color[] colors = tex.GetPixels();
  for (int i=0; i<colors.Length; i++) colors[i] = color; //new Color(0.0f, 0.0f, 0.0f, 1);
  tex.SetPixels(colors);
  tex.Apply(true, true);

  if (!blankTextures.ContainsKey(colorHash)) blankTextures.Add(colorHash, tex);
  else blankTextures[colorHash] = tex;
```

```
  return tex;
 }
}



public static Texture2D LoadTextureAtPath (string textureName, bool forceLight=false, bool forcePro=fals
/// Loads texture dealing with pro name and folders
{
 Texture2D texture;

 bool isPro = StylesCache.isPro;
 if (forceLight) isPro = false;
 if (forcePro) isPro = true;

 if (!isPro)
 {
  texture = Resources.Load(textureName) as Texture2D;
  //if (texture == null) texture = Resources.Load(proName) as Texture2D;
  //if (texture == null) texture = Resources.Load(proFolderName) as Texture2D;
 }
 else
 {
  string proName = textureName + "_pro";

  int sepIndex = textureName.LastIndexOf('/'); if (sepIndex < 0) sepIndex = 0;
  string folderName = textureName.Substring(0,sepIndex);
```

```
  string fileName = textureName.Substring(sepIndex+1);
  string proFolderName = folderName + "/pro/" + fileName;

  texture = Resources.Load(proName) as Texture2D;
  if (texture == null) texture = Resources.Load(proFolderName) as Texture2D;
  if (texture == null) texture = Resources.Load(textureName) as Texture2D;
 }

 #if UNITY_EDITOR
 if (texture == null && !UnityEditor.BuildPipeline.isBuildingPlayer) //hack since there are no resources duri
  throw new Exception("Could not find texture " + textureName);
 #else
 if (texture == null)
  throw new Exception("Could not find texture " + textureName);
 #endif

 return texture;
}


public GUIStyle GetElementStyle (string textureName, RectOffset borders=null, RectOffset overflow=null)
{
 Texture2D tex = GetTexture(textureName);
 return GetElementStyle(tex, null, borders, overflow);
}
```

```
public GUIStyle GetElementStyle (string textureName, string onTextureName, RectOffset borders=null, R
{
 Texture2D tex = GetTexture(textureName);
 Texture2D onTex = GetTexture(onTextureName);
 return GetElementStyle(tex, onTex, borders, overflow);
}

public GUIStyle GetElementStyle (Texture2D tex, Texture2D onTex=null, RectOffset borders=null, RectO
{
 if (styles.TryGetValue(tex, out GUIStyle style)) return style;
 else
 {
  style = new GUIStyle();
  style.normal.background = tex;
  style.active.background = style.onActive.background = style.onNormal.background = style.onFocused.b

  if (borders == null) style.border = new RectOffset(tex.width/2, tex.width/2, tex.height/2, tex.height/2);
  else style.border = borders;
  //if (borders != null) style.border = borders;

  if (overflow != null) style.overflow = overflow;

  styles.Add(tex, style);
  return style;
 }
}
```

```
public Material GetMaterial (string shaderName)
{
 if (materials.ContainsKey(shaderName))
 {
  Material mat = materials[shaderName];
  if (mat==null) //destroys material on scene close
  {
   mat = new Material( Shader.Find(shaderName) );
   materials[shaderName] = mat;
  }
  return mat;
 }

 else
 {
  Material mat = new Material( Shader.Find(shaderName) );
  materials.Add(shaderName,mat);
  return mat;
 }
}

public void SetMaterial (Material mat)
      {
 string shaderName = mat.shader.name;
 if (materials.ContainsKey(shaderName)) materials[shaderName] = mat;
```

```
    else materials.Add(shaderName, mat);
  }


 }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
using System.Runtime.InteropServices;
using UnityEngine.Profiling;

namespace Den.Tools.GUI
{
 public class UI
 {
  public static UI current;

  public Cell rootCell;
  public List<(Action action, int order)> afterLayouts = new List<(Action,int)>();
  public List<(Action action, int order)> afterDraws = new List<(Action,int)>();
  public bool layout;

  public ScrollZoom scrollZoom = null;
  public StylesCache styles = null;
  public TexturesCache textures = new TexturesCache();
  public CellObjs cellObjs = new CellObjs();
  public Undo undo = null;
```

```
public EditorWindow editorWindow;

public Rect subWindowRect;
public Vector2 viewRectMax = new Vector2(int.MaxValue, int.MaxValue); //window rect in internal coordin
public Vector2 viewRectMin = new Vector2(-int.MaxValue/2, -int.MaxValue/2); //min is 0 when no scrollzo
public Vector2 mousePos; //mouse position in internal coordinates
public Vector2 prevMousePos;
public int mouseButton = -1; //shortcut for Event.current.button (0-left, 1-right, 2-middle)
public float ViewRectHeight { get{ return viewRectMax.y-viewRectMin.y; } }

public bool optimizeEvents = false;
public bool optimizeElements = false; //skips cell if it has no child cells
public bool optimizeCells = false; //skips cell if it has child cells. Experimental!

public bool hardwareMouse = false;

public bool isInspector = false; //to draw foldout
public Vector2 scrollBarPos; //for windows with the scrollbar

public Cell delayedCell;  //to change values only on Enter
public float delayedFloat;
public int delayedInt;

public static bool FieldLostFocus =>  //no way to find out if field lost focus, so using cases where it can do
 //(UI.current != null  &&  UI.current.editorWindow != EditorWindow.focusedWindow)  ||  //will reset cell in U
 (Event.current.rawType != EventType.Layout && //somehow called from UI on layout
```

```
 (Event.current.keyCode == KeyCode.Return  ||  Event.current.keyCode == KeyCode.KeypadEnter  ||  Ev
 Event.current.rawType == EventType.MouseDown  ||  Event.current.rawType == EventType.MouseUp  ||
 Event.current.rawType == EventType.MouseDrag;



public float DpiScaleFactor
{get{
 if (editorWindow==null) return 1;

 float factor = Screen.width / editorWindow.position.width;
 return ((int)(float)(factor * 4f + 0.5f)) / 4f; //rounding to 0.25f
}}

public enum RectSelector { Standard, Padded, Full }

public static bool MouseUp
//Working synonym for EventType.MouseUp
{get{
 if (Event.current.rawType == EventType.MouseUp) return true;

 //MouseUp is not called when mouse leaved window
 //but MouseLeaveWindow does - when releasing mouse
 //docs say that MouseLeaveWindow is not fired when mouse pressed, but actually it's fired when mouse
 //wantsMouseEnterLeaveWindow should be set to true

 //#if !UNITY_EDITOR_OSX  //in older versions it worked fine in OSX, but now seems the same as Windo
```

```
 if (Event.current.rawType == EventType.MouseLeaveWindow) return true;
 //#endif

 return false;
}}


public static UI ScrolledUI (float maxZoom=1, float minZoom=0.4375f)
{
 return new UI {
  scrollZoom = new ScrollZoom() { allowScroll=true, allowZoom=true, maxZoom = maxZoom, minZoom =
  optimizeEvents = true,
  optimizeElements = true };
}


#region Draw

 public void DrawInSubWindow (Action drawAction, int id, Rect rect)
 /// Draws in unity's BeginWindows group
 {
  this.subWindowRect = rect;

  //rect.position = new Vector2(0,0);

  UnityEngine.GUI.WindowFunction drawFn = tmp => Draw(drawAction, inInspector:false);
```

```
 //hack. GUILayout.Window will not be called when mouse is not in window rect, but we have to release c
 if (editorWindow != null  &&  !editorWindow.wantsMouseEnterLeaveWindow)
  editorWindow.wantsMouseEnterLeaveWindow = true;

 if (MouseUp)
 {
  Event.current.mousePosition -= rect.position; //offseting release button mouse position since it counts w
  Draw(drawAction, inInspector:false);
 }

 else
 {
  //placing 2 rects in _this_ window if GraphGUI was not called
  UnityEditor.EditorGUILayout.GetControlRect(GUILayout.Height(0));
  UnityEditor.EditorGUILayout.GetControlRect(GUILayout.Height(0));
 }

 //window
 GUILayout.Window(id, rect, drawFn, new GUIContent(), GUIStyle.none,
  GUILayout.MaxHeight(rect.height), GUILayout.MinHeight(rect.height),
  GUILayout.MaxWidth(rect.width), GUILayout.MinWidth(rect.width) );
}


public void Draw (Action drawAction, bool inInspector, Rect customRect=new Rect())
```

```
/// If calling two Draw instances in one window one should have offsetAfterDraw enabled (otherwise will n
/// inInspector - starts the rect after inspector content and adds height after to draw further components
/// if not in inspector - then can draw in custom rect (instead of full window)
{
 Profiler.BeginSample("DrawUI");

 UI.current = this;

 afterLayouts.Add((drawAction, 0));
 afterDraws.Add((drawAction, 0));

 editorWindow = GetActiveWindow();
 float dpiScaleFactor = DpiScaleFactor;
 //Vector2 screenSize = new Vector2(Screen.width, Screen.height) / DpiScaleFactor;
  //screen size not in pixels but in Unity's understanding. It will be scaled back to pixels then

 //if in scrollable window - enabling wants mouse leave
 //GUILayout.Window will not be called when mouse is not in window rect, but we have to release drag sc
 if (!inInspector && scrollZoom != null && (mouseButton==2 || Event.current.alt))
  editorWindow.wantsMouseEnterLeaveWindow = true;

 //finding rect
 UnityEditor.EditorGUI.indentLevel = 0;
 Rect rect;
 if (inInspector) rect = GUILayoutUtility.GetRect(new GUIContent(), GUIStyle.none); //EditorStyles.helpBo
 else
```

```
{
 if (customRect.width < 0.1f  &&  customRect.height < 0.1f)
  rect = new Rect(0,0,Screen.width/dpiScaleFactor, Screen.height/dpiScaleFactor);
 else
  rect = customRect;
}
subWindowRect = rect;
this.isInspector = inInspector;

//scroll/zoom
if (scrollZoom != null)  //just because alt rotates 'scene view in graph'
{
 #if MM_DEBUG
 if (!Event.current.alt)
 #endif
 {
  scrollZoom.Scroll();
  scrollZoom.Zoom();
 }
}



//styles
if (styles == null)
 styles = new StylesCache();
```

```
styles.CheckInit();
if (scrollZoom != null)
 styles.Resize(scrollZoom.zoom);

//mouse button
if (Event.current.type == EventType.MouseDown)
 mouseButton = Event.current.button;
else
 mouseButton = -1;

//mouse pos
prevMousePos = mousePos;
#if UNITY_EDITOR_WIN
if (hardwareMouse)
{
 GetCursorPos(out Vector2Int intPos);
 mousePos = intPos - editorWindow.position.position;
}
else
#endif
 mousePos = Event.current.mousePosition;

//internal rect
if (scrollZoom != null)
{
 viewRectMin = scrollZoom.ToInternal( new Vector2(0,0) ) - Vector2.one;
```

```
 viewRectMax = scrollZoom.ToInternal( new Vector2(Screen.width, Screen.height) ) + Vector2.one;
 mousePos = scrollZoom.ToInternal(mousePos);
}
else
{
 viewRectMin = Vector2.zero;
 viewRectMax = new Vector2(Screen.width, Screen.height);
}

//root cell rect (hacky)
Rect rootCellRect = inInspector ?
 rect :
 new Rect(0,0, Screen.width/dpiScaleFactor, Screen.height/dpiScaleFactor);

//preparing shaders
Shader.SetGlobalVector("_ScreenRect", new Vector4(rect.x, rect.y, Screen.width, Screen.height) );
Shader.SetGlobalVector("_ScreenParams", new Vector4(Screen.width, Screen.height, 1f/Screen.width,
Shader.SetGlobalVector("_InternalRect", new Vector4(viewRectMin.x, viewRectMin.y, viewRectMax.x-vi

//clearing active cell stack in case previous gui was failed to finish (or color picker clicked)
if (Cell.activeStack.Count != 0)
{
 Cell.activeStack.Clear();
 //Debug.Log("Trying to start UI with non-empty active stack");
}
```

```
//drawing
if (!optimizeEvents || !SkipEvent())
//using (Timer.Start("Draw GUI"))
{
 layout = true;
 //using (Timer.Start("Draw pre-layout"))
  using (Cell.Root(ref rootCell, rootCellRect))
  {
   for (int i=0; i<afterLayouts.Count; i++) //count could be increased while iterating
    afterLayouts[i].action();
  }

 //using (Timer.Start("CalculateMinContentsSize"))
  rootCell.CalculateMinContentsSize();

 //using (Timer.Start("CalculateRootRects"))
  rootCell.CalculateRootRects();

 layout = false;
 //using (Timer.Start("Draw final"))
  using (Cell.Root(ref rootCell, rootCellRect))
  {
   for (int i=0; i<afterDraws.Count; i++) //count could be increased while iterating
    afterDraws[i].action();
  }
```

```
 UI.current = null;
}

DragDrop.ResetTempObjs();

//resetting afterdraw actions
afterLayouts.Clear();
afterDraws.Clear();

cellObjs.Clear();

//setting inspector/window rect
if (inInspector)
{
 float inspectorHeight = rootCell!=null ? (float)rootCell.finalSize.y : 0;
 inspectorHeight -= 20; //Unity leaves empty space for some reason
 Rect wholeRect = UnityEditor.EditorGUILayout.GetControlRect(GUILayout.Height(inspectorHeight));

 UnityEngine.GUI.Button(wholeRect, "", GUIStyle.none);
 //drawing any control on all the field, otherwise OnMouseUp won't be called when mouse left the windo
 //known issue: whole rect doesnt cover all for some reason
}

//clearing delayed cell on field lost focus
if (delayedCell!=null  &&  FieldLostFocus)
```

```
  delayedCell = null;

 if (Event.current.keyCode == KeyCode.Escape)
  delayedCell = null;

 //disabling field right-click (copy/paste) when opened right-click menu
 if (Event.current.isMouse &&
  Event.current.type == EventType.MouseUp  &&
  Event.current.button == 1  &&
  EditorWindow.focusedWindow != null &&
  EditorWindow.focusedWindow.GetType() == typeof(UnityEditor.PopupWindow))
   Event.current.Use();

 Profiler.EndSample();
}

public void DrawAfter (Action action, int layer=1)
{
 if (layout)
 {
  afterLayouts.Add( (action,layer) );
  afterLayouts.Sort( (a,b) => -a.order + b.order );
 }
 else
 {
```

```
   afterDraws.Add( (action,layer) );
   afterDraws.Sort( (a,b) => a.order - b.order );
  }

  //Debug.Log("Layouts add " + afterLayouts.Count);
  //Debug.Log("Draws add " + afterDraws.Count);
 }

 public void ClearDrawAfter ()
 {
  if (layout)
   afterLayouts.Clear();
  else
   afterDraws.Clear();
 }

#endregion



#region Helpers

 public static bool SkipEvent ()
 /// Should this event be skipped?
 {
  bool skipEvent = false;
```

```
 if (Event.current.type == EventType.Layout  ||  Event.current.type == EventType.Used) skipEvent = true;
 if (Event.current.type == EventType.MouseDrag) //skip all mouse drags (except when dragging text sele
 {
  if (!UnityEditor.EditorGUIUtility.editingTextField) skipEvent = true;
  if (UnityEngine.GUI.GetNameOfFocusedControl() == "Temp") skipEvent = true;
 }
 if (Event.current.rawType == EventType.MouseUp) skipEvent = false;

 return skipEvent;
}


public bool IsInWindow ()
/// Finding if cell within a window by it's rect
{
 Cell cell = Cell.current;

 float borders = 1;

 //Vector2 cellRectPos = cell.worldPosition;
 //Vector2 cellRectSize = cell.finalSize;

 float minX = cell.worldPosition.x;
 float maxX = cell.worldPosition.x + cell.finalSize.x;

 float minY = cell.worldPosition.y;
```

```
 float maxY = cell.worldPosition.y + cell.finalSize.y;

 if (maxX < viewRectMin.x - borders||
  maxY < viewRectMin.y - borders ||
  minX > viewRectMax.x + borders ||
  minY > viewRectMax.y + borders)
   return false;

 return true;
}

public bool IsInWindow (float minX, float maxX, float minY, float maxY)
/// Finding if cell within a window by it's rect
{
 Cell cell = Cell.current;

 float borders = 1;

 if (maxX < viewRectMin.x - borders||
  maxY < viewRectMin.y - borders ||
  minX > viewRectMax.x + borders ||
  minY > viewRectMax.y + borders)
   return false;

 return true;
```

```
}


public static void RemoveFocusOnControl ()
/// GUI.FocusControl(null) is not reliable, so creating a temporary control and focusing on it
{
 //UnityEngine.GUI.SetNextControlName("Temp");
 //UnityEditor.EditorGUI.FloatField(new Rect(-10,-10,0,0), 0);
 //UnityEngine.GUI.FocusControl("Temp");

 UnityEngine.GUI.FocusControl(null);
}


public static void RepaintAllWindows ()
/// Usually called on undo
{
 UnityEditor.EditorWindow[] windows = Resources.FindObjectsOfTypeAll<UnityEditor.EditorWindow>();
 foreach (UnityEditor.EditorWindow win in windows)
  win.Repaint();
}


public void MarkChanged (bool completeUndo=false)
/// Writes undo and cell change. Should be called BEFORE actual change since writes undo
{
```

```
 //write undo and dirty (got to know undo object to set it dirty)
 undo?.Record(completeUndo);

 //writing changed state in all active cells
 for (int i=Cell.activeStack.Count-1; i>=0; i--)
 {
  if (!Cell.activeStack[i].trackChange) break; //root cell should not recieve value change if non-tracked cell
  Cell.activeStack[i].valChanged = true;
 }

}

private static string[] GetPopupNames<T> (T[] objs, Func<T,string> nameFn, string none=null, string[] na
/// Generates names array for popups. Use 'none' to place it before other variants. Use 'names' to re-use
{
 int arrLength = objs.Length;
 if (none != null) arrLength++;

 if (names == null || names.Length != arrLength)
  names = new string[arrLength];

 int c = 0;
 for (int i=0; i<arrLength; i++)
 {
  if (i==0 && none!=null) { names[0] = none; continue; }
```

```
  names[i] = nameFn(objs[c]);
  c++;
 }

 return names;
}


public static Texture2D GetBlankTex ()
{
 Texture2D tex = new Texture2D(4,4);
 Color[] colors = tex.GetPixels();
 for (int i=0; i<colors.Length; i++) colors[i] = new Color(0,0,0,1);
 tex.SetPixels(colors);
 tex.Apply(true, true);
 return tex;
}


[DllImport("user32.dll")]
public static extern bool GetCursorPos(out Vector2Int lpPoint);

[DllImport("user32.dll")]
public static extern bool SetCursorPos(int x, int y);
```

```
  public static EditorWindow GetActiveWindow ()
  {
   //HostView hostView = GUIView.current as HostView;
   //return hostView.actualView;

   Type guiViewType = typeof(EditorWindow).Assembly.GetType("UnityEditor.GUIView");
   PropertyInfo currentGuiViewProp = guiViewType.GetProperty("current", BindingFlags.Static | BindingFla
   object currentGuiView = currentGuiViewProp.GetValue(guiViewType, null);
   if (currentGuiView == null) return null;

   Type hostViewType = currentGuiView.GetType(); //could be DockArea, which also has a actualView pro
   //Type hostViewType = typeof(EditorWindow).Assembly.GetType("UnityEditor.HostView");
   //if (currentGuiView.GetType() != hostViewType) return null;
   PropertyInfo actualViewProp = hostViewType.GetProperty("actualView", BindingFlags.Instance | Binding
   object activeView = actualViewProp.GetValue(currentGuiView);

   return activeView as EditorWindow;
  }

 #endregion
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
using System.Runtime.InteropServices;

namespace Den.Tools.GUI
{
 public class Undo
 {
     public UnityEngine.Object undoObject;
  public string undoName;
  public Action undoAction;
  public string lastUndoName;  //the last undo group name (kept to know it on UndoRedoPerformed)

  public Undo ()
  {
   UnityEditor.Undo.undoRedoPerformed -= OnUndoRedoPerformed;
   UnityEditor.Undo.undoRedoPerformed += OnUndoRedoPerformed;
  }


  public void OnUndoRedoPerformed ()
  {
   string currGroupName = UnityEditor.Undo.GetCurrentGroupName();
```

```
 if (currGroupName == undoName || currGroupName == lastUndoName)
 // a bit hacky here. On undoRedoPerformed there is already no current group in stack, and no way to get
 // so we store previous (before mm change) name and performing undo if this name is first in stack
 // TODO: use undo from MapMagicBrush with ids, it's more stable
 {
  EditorWindow.focusedWindow?.Repaint();
  undoAction?.Invoke();

  if (currGroupName == lastUndoName)
   lastUndoName = null;
 }
}

public void Record (bool completeUndo=false)
{
 if (undoObject==null) return;

 string currGroupName = UnityEditor.Undo.GetCurrentGroupName();
 if (currGroupName != undoName)
  lastUndoName = currGroupName;

 if (completeUndo) UnityEditor.Undo.RegisterCompleteObjectUndo(undoObject, undoName);
 else UnityEditor.Undo.RecordObject(undoObject, undoName);

 EditorUtility.SetDirty(undoObject);
```

```
  }
 }
}
```

```
ï»¿using System;
using System.Collections.Generic;
using UnityEngine;

namespace Den.Tools.GUI.Popup
{
 public class Item
 {
  public const int separatorHeight = 6;
  public const int lineHeight = 20;

  public string name;
  public bool clicked;
  public bool disabled;
  public int priority;
  public Texture2D icon;  //currently doing nothing, just store data for custom draw
  public Color color;  //same
  public Action<Item,Rect> onDraw;
  public int offset;  //aka tab symbol
  public float scroll; //for the scroll position of sub-menu

  public float width;
  public float height = lineHeight;
  public bool isSeparator;

  public List<Item> subItems = null;
```

```
public bool sortSubItems = true;

public Action onClick; //action called when subitem selected
public bool closeOnClick; //close menu after

public int Count { get{ return subItems==null ? 0 : subItems.Count; } }
public bool hasSubs { get{ return subItems!=null;} }

public Item () { }
public Item (string name, Action<Item,Rect> onDraw=null, Action onClick=null, bool disabled=false, int pri
{
 this.name=name;
 this.priority=priority;
 this.onClick=onClick;
 this.onDraw = onDraw;
 this.disabled=disabled;

 this.width = UnityEngine.GUI.skin.label.CalcSize( new GUIContent(name) ).x + 20;  //20 for chevron
}
public Item (string name, params Item[] items)
{
 this.name=name;
 subItems = new List<Item>();
 subItems.AddRange(items);

 this.width = UnityEngine.GUI.skin.label.CalcSize( new GUIContent(name) ).x + 20;  //20 for chevron
```

```
}

public static Item Separator (int priority=0) { return new Item() { isSeparator=true, height=separatorHeight,

public void SortSubItems ()
{
 if (subItems == null) return;
 subItems.Sort(Compare);
 foreach (Item subItem in subItems)
  subItem.SortSubItems();
}
public static void SortItems (List<Item> items) { items.Sort(Compare); }
public static int Compare (Item a, Item b)
{
 if (a.priority != b.priority) return b.priority - a.priority;

 if (a.name==null || b.name==null) return 0;
 if (a.name.Length==0) return -1; if (b.name.Length==0) return 1;
 if ((int)(a.name[0]) < (int)(b.name[0])) return -1;
 else if ((int)(a.name[0]) == (int)(b.name[0])) return 0;
 else return 1;
}

public IEnumerable<Item> All(bool inSubItems=true)
{
 int subItemsCount = subItems.Count;
```

```
 for (int i=0; i<subItems.Count; i++)
 {
  yield return subItems[i];

  if (subItems[i].subItems != null  && inSubItems)
   foreach(Item sub in subItems[i].All(true))
    yield return sub;
 }
}

public Item Find (string findName, bool inSubItems=true, bool contains=false)
{
 string findNameLower = null;
 if (contains)
  findNameLower = findName.ToLower();

 int subItemsCount = subItems.Count;
 for (int i=0; i<subItems.Count; i++)
 {
  if (subItems[i].name == null)
   continue;

  if (subItems[i].name == findName  ||  (contains && subItems[i].name.ToLower().Contains(findNameLowe
   return subItems[i];
 }
```

```
 if (inSubItems)
 for (int i=0; i<subItems.Count; i++)
  if (subItems[i].subItems != null)
  {
   Item subFound = subItems[i].Find(findName, true);
   if (subFound != null)
    return subFound;
  }

 return null;
}


public List<Item> FindAll (string findName, bool inSubItems=true, bool contains=false)
/// Finds all references with this or contained name
{
 List<Item> found = null;
 string findNameLower = null;
 if (contains)
  findNameLower = findName.ToLower();

 int subItemsCount = subItems.Count;
 for (int i=0; i<subItems.Count; i++)
 {
  if (subItems[i].name == null)
   continue;
```

```
    if (subItems[i].name == findName  ||  (contains && subItems[i].name.ToLower().Contains(findNameLowe
    {
     if (found == null) found = new List<Item>();
     found.Add(subItems[i]);
    }
   }

   if (inSubItems)
   for (int i=0; i<subItems.Count; i++)
    if (subItems[i].subItems != null)
    {
     List<Item> subFound = subItems[i].FindAll(findName, true, contains);
     if (subFound != null)
     {
      if (found == null) found = new List<Item>();
      found.AddRange(subFound);
     }
    }

   return found;
  }
 }

}
```

```
ï»¿using System;
using System.Collections.Generic;

using UnityEngine;
using UnityEditor;

namespace Den.Tools.GUI.Popup
{
 public class SingleWindow : PopupWindowContent
 {
  const bool debug = false;

  public bool sortItems = true;
  public int width = 170;
  public int height = 305;

  private UI ui = UI.ScrolledUI();

  public List<Item> openedItems = new List<Item>();
  public Item RootItem => openedItems[0];
  public Item TopItem => openedItems[deepness];
  public int deepness = 0; //at which path level we are now (instead path.Count-1, since we've got to have s

  public static readonly Color highlightColor = new Color(0.6f, 0.7f, 0.9f);
  public static readonly Color backgroundColor =  new Color(0.925f, 0.925f, 0.925f);
  public static readonly Color highlightColorPro = new Color(0.219f, 0.387f, 0.629f);
```

```
public static readonly Color backgroundColorPro =  new Color(0.33f, 0.33f, 0.33f);

public static readonly float scrollSpeed = 5f;
private static DateTime lastFrameTime = DateTime.Now;

public string search;

public SingleWindow (Item rootItem) => openedItems.Add(rootItem);

public override void OnGUI (Rect rect)  =>  ui.Draw(DrawGUI, inInspector:false);

private void DrawGUI ()
{
 ui.scrollZoom.allowScroll = false;
 ui.scrollZoom.allowZoom = false;

 ui.optimizeElements = false;

 Draw.Rect(StylesCache.isPro ? backgroundColorPro : backgroundColor);

 //smoothly scrolling
 float deltaTime = (float)(DateTime.Now-lastFrameTime).TotalSeconds;
 lastFrameTime = DateTime.Now;

 float targetScroll = -deepness * width;
```

```
if (!debug)
{
 if (ui.scrollZoom.scroll.x < targetScroll)
 {
  ui.scrollZoom.scroll.x += scrollSpeed * width * deltaTime;
  if (ui.scrollZoom.scroll.x > targetScroll) ui.scrollZoom.scroll.x = targetScroll;
 }
 if (ui.scrollZoom.scroll.x > targetScroll)
 {
  ui.scrollZoom.scroll.x -= scrollSpeed * width * deltaTime;
  if (ui.scrollZoom.scroll.x < targetScroll) ui.scrollZoom.scroll.x = targetScroll;
 }
}

//drawing search first - otherwise it will loose focus
using (Cell.Custom(-targetScroll, 25+2, width, 18))
{
 Cell.EmptyRowPx(6);
 using (Cell.Row)
 {
  Draw.SearchLabel(ref search, forceFocus:true);

  if (Cell.current.valChanged)
   PerformSearch(search, openedItems[deepness], deepness);
 }
 Cell.EmptyRowPx(6);
```

```
 }

 //drawing
 for (int p=0; p<openedItems.Count; p++)
  using (Cell.RowPx(width))
   DrawMenu(openedItems[p], p);

 //refreshing selection frame
 this.editorWindow.Repaint();
}


private void PerformSearch (string search, Item startingItem, int startingItemNum)
{
 //removing all other opened items
 openedItems.RemoveAfter(startingItemNum);

 //search erased - removing search tab
 if (search == null || search.Length == 0)
 {
  if (openedItems[openedItems.Count-1].name.StartsWith("Search "))
   openedItems.RemoveAfter(openedItems.Count-2);

  deepness = openedItems.Count-1;
 }
```

```
 //search entered/changed
 else
 {
  Item baseItem = null; //item we performing search at
  Item searchItem = null; //item with search results

  if (openedItems[openedItems.Count-1].name.StartsWith("Search ")) //search is already opened
  {
   searchItem = openedItems[openedItems.Count-1];
   baseItem = openedItems[openedItems.Count-2];
  }
  else //new search item
  {
   baseItem = openedItems[openedItems.Count-1];
   searchItem = new Item("Search " + baseItem.name);
   openedItems.Add(searchItem);
  }

  deepness = openedItems.Count-1;

  searchItem.subItems = baseItem.FindAll(search, inSubItems:true, contains:true);
 }
}


private void DrawMenu(Item item, int itemDeepness)
```

```
{
 //header
 using (Cell.LinePx(25))
 {
  Texture2D headerTex = UI.current.textures.GetTexture("DPUI/Backgrounds/Popup");
  Draw.ColorizedTexture(headerTex, item.color);

  if (itemDeepness != 0)
  {
   Texture2D shveronTex = UI.current.textures.GetTexture("DPUI/Chevrons/TickLeft");
   using (Cell.RowPx(20)) Draw.Icon(shveronTex);
  }

  Draw.Label(item.name, style:UI.current.styles.boldMiddleCenterLabel);

  bool clicked = Cell.current.Contains(ui.mousePos) && Event.current.rawType == EventType.MouseDow
  if (clicked && itemDeepness != 0)
  {
   deepness = itemDeepness-1;

   search = null;
   UnityEditor.EditorGUI.FocusTextInControl(null);
  }
 }

 //search (actually just phantoms, real serch is drawn before)
```

```
Cell.EmptyLinePx(2);
using (Cell.LinePx(18))
{
 if (itemDeepness!=deepness)
  search = Draw.SearchLabel(search);
}
Cell.EmptyLinePx(2);

//sub-items
if (item.subItems != null)
{
 bool coloredIcons = item.name.StartsWith("Search ");

 int itemsSpace = height-25-22;
 using (Cell.LinePx(itemsSpace))
  using (new Draw.ScrollGroup(ref item.scroll, enabled:itemsSpace<item.subItems.Count*22))
 {

  for (int n=0; n<item.subItems.Count; n++)
  {
   using (Cell.LinePx(0))
   {
    Item currItem = item.subItems[n];

    //drawing
    bool highlighted = Cell.current.Contains(ui.mousePos);
```

```
 if (!currItem.isSeparator)
  using (Cell.LinePx(22)) DefaultItemDraw(currItem, n, highlighted, coloredIcons);
 else
  using (Cell.LinePx(Item.separatorHeight)) DrawSeparator();

 //clicking
 bool clicked = highlighted &&
  !currItem.disabled &&
  Event.current.type == EventType.MouseDown &&
  Event.current.button == 0 &&
  !UI.current.layout;

 if (clicked && currItem.subItems != null)
 {
  if (openedItems.Count-1 > itemDeepness)
   openedItems.RemoveAfter(itemDeepness);
  openedItems.Add(currItem);
  deepness = itemDeepness + 1;
 }
 if (clicked && currItem.onClick != null)
  currItem.onClick();

 if (clicked && currItem.closeOnClick)
  editorWindow.Close();
}
```

```
   }
  }
 }
}


public void DefaultItemDraw (Item item, int num, bool selected, bool colored=false)
{
 if (selected && !item.disabled)
  Draw.Rect(StylesCache.isPro ? highlightColorPro : highlightColor);

 Cell.current.disabled = item.disabled;

 //icon
 using (Cell.RowPx(30))
 {
  if (colored)
   Draw.Rect(item.color);

  if (item.icon!=null)
   Draw.Icon(item.icon, scale:0.5f);
 }

 //label
 using (Cell.Row) Draw.Label(item.name);
```

```
 //chevron
 if (item.subItems != null)
 {
  Texture2D chevronTex = UI.current.textures.GetTexture("DPUI/Chevrons/TickRight");
  using (Cell.RowPx(20)) Draw.Icon(chevronTex);
 }
}



public void DrawSeparator ()
{
 Cell.EmptyRowPx(20);
 using (Cell.Row)
 {
  Cell.EmptyLine();
  using (Cell.LinePx(1)) Draw.Rect(Color.gray);
  Cell.EmptyLine();
 }
 Cell.EmptyRowPx(20);
}



public override Vector2 GetWindowSize()
{
 return new Vector2(width*(debug ? 5 : 1), height);
}
```

```
  public void Show (Vector2 pos)
  {
    RootItem.SortSubItems();
    PopupWindow.Show(new Rect(pos.x-width/2,pos.y-10,width,0), this);
  }
 }

}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using System.Reflection;
using UnityEngine;

namespace Den.Tools
{
 public static class Log
 {
  public class Entry : IDisposable
  {
   public string name;
   public string threadName;
   public long startTicks;
   public long disposeTicks;
   public (string,string)[] fieldValues;
   public List<Entry> subs;
   public bool guiExpanded;

   public void Dispose () => Log.DisposeGroup();

   public int Count
   {get{
    int count = 1;
```

```
  if (subs!=null)
   foreach (Entry sub in subs)
    count += sub.Count;
  return count;
 }}

 public IEnumerable<Entry> SubsRecursive ()
 {
  if (subs==null) yield break;
  foreach (Entry sub in subs)
  {
   yield return sub;

   if (sub.subs != null)
    foreach (Entry subSub in sub.SubsRecursive())
     yield return subSub;
  }
 }
}

public static bool enabled = false;

public static Entry root = new Entry() {name="Root"};
private static Entry activeGroup = root; //not among openedGroups  //TODO: make a dictionary thread->g
private static List<Entry> openedGroups = new List<Entry>();
```

```
private static Entry tempGroup = new Entry(); //to return when recording disabled

public const string defaultId = "Default";

public static void AddThreadId (string name) => Add(name, Thread.CurrentThread.ManagedThreadId.ToS

public static void Add (string name, string id=defaultId)
{
 if (!enabled) return;

 Entry entry = new Entry() {name=name, threadName=id};

 if (activeGroup.subs == null) activeGroup.subs = new List<Entry>();
 activeGroup.subs.Add(entry);
}

public static void Add (string name, string id, object obj)
{
 if (!enabled) return;

 Entry entry = new Entry() {name=name, threadName=id};
 entry.fieldValues = ReadValues(obj);

 if (activeGroup.subs == null) activeGroup.subs = new List<Entry>();
 activeGroup.subs.Add(entry);
```

```
}


public static Entry Group (string name, string id=defaultId)
{
 if (!enabled) return tempGroup;

 Entry entry = new Entry() {name=name, threadName=id};

 if (activeGroup.subs == null) activeGroup.subs = new List<Entry>();
 activeGroup.subs.Add(entry);

 openedGroups.Add(activeGroup);
 activeGroup = entry;

 long unityStartTime = System.Diagnostics.Process.GetCurrentProcess().StartTime.Ticks;
 long currentTime = DateTime.Now.Ticks; //todo: minimize operations after DateTime.Now
 entry.startTicks = currentTime - unityStartTime;

 return entry;
}


private static void DisposeGroup ()
{
```

```
 if (!enabled) return;

 long currentTime = DateTime.Now.Ticks;
 long unityStartTime = System.Diagnostics.Process.GetCurrentProcess().StartTime.Ticks;
 activeGroup.disposeTicks = currentTime - unityStartTime;

 activeGroup = openedGroups[openedGroups.Count-1];
 openedGroups.RemoveAt(openedGroups.Count-1);
}



private static (string,string)[] ReadValues (object obj)
{
 Type type = obj.GetType();
 FieldInfo[] fields = type.GetFields(BindingFlags.Instance | BindingFlags.Static | BindingFlags.Public | Bind

 (string,string)[] fieldValues = new (string,string)[fields.Length];

 for (int i=0; i<fields.Length; i++)
 {
  string name = fields[i].Name;
  string value = fields[i].GetValue(obj).ToString();

  fieldValues[i] = (name,value);
 }
```

```
    return fieldValues;
  }



  public static void Clear () => root.subs.Clear();


  public static int Count => root.Count;


  public static IEnumerable AllEntries ()  //all except root
  {
    foreach (Entry sub in root.SubsRecursive())
      yield return sub;
  }



  public static HashSet<string> UsedThreads ()
  {
    HashSet<string> usedIds = new HashSet<string>();


    foreach (Entry sub in AllEntries())
      usedIds.Add(sub.threadName);


    return usedIds;
  }
 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;

namespace Den.Tools
{
 public class LogWindow : EditorWindow
 {
  const int toolbarHeight = 18;
  const int scrollWidth = 15;
  const int lineHeight = 18;
  const int rowMinWidth = 100;
  const float namesWidthPercent = 0.4f;
  Vector2 scrollPosition;
  bool threadedView = false;
  //int selectedLine = 1;

  float timeWidth = 50;

  GUIStyle labelStyle = null;

  bool groupEnabled = false;
  bool prevEnabled;

  int prevLogCount = 0;
```

```
public void OnInspectorUpdate ()
{
 if (Log.Count != prevLogCount)
  Repaint();
 prevLogCount = Log.Count;
}



public void OnGUI ()
{
 DrawToolbar();

 Dictionary<string,int> threadToRow = GetIdsToRows();
 DrawHeader(threadToRow);
 DrawList(threadToRow);
}

public void DrawToolbar ()
{
 //toolbar/header
 if (Event.current.type == EventType.Repaint)
  EditorStyles.toolbar.Draw(new Rect(0,0,position.width, toolbarHeight), new GUIContent(), 0);

 //Record
 Log.enabled = EditorGUI.Toggle(new Rect(5,0,50, toolbarHeight), Log.enabled, style:EditorStyles.toolba
```

```
 EditorGUI.LabelField(new Rect(9,1,50,toolbarHeight), "Record", style:EditorStyles.miniBoldLabel);

 //Group (not used, just for future)
 bool newGroupEnabled = EditorGUI.Toggle(new Rect(55,0,50,toolbarHeight), groupEnabled, style:Editor
 if (newGroupEnabled && groupEnabled) //just pressed
 {
  groupEnabled = true;
 }
 if (!newGroupEnabled && groupEnabled)
 {
  groupEnabled = false;
 }
 EditorGUI.LabelField(new Rect(63,1,50,toolbarHeight), "Group", style:EditorStyles.miniBoldLabel);

 //Clear
 if (UnityEngine.GUI.Button(new Rect(105,0,50,toolbarHeight), "Clear", style:EditorStyles.toolbarButton))
  Log.Clear();

 //threaded view
 threadedView = UnityEngine.GUI.Toggle(new Rect(165,-8,100,35), threadedView, "Threaded view");
}

public void DrawHeader (Dictionary<string,int> threadToRow)
{
 float rowWidth = (position.width-scrollWidth) / threadToRow.Count;
 if (rowWidth < rowMinWidth) rowWidth = rowMinWidth;
```

```
UnityEngine.GUI.BeginScrollView(
 position:new Rect(0, toolbarHeight, position.width-18, lineHeight),
 scrollPosition:new Vector2(scrollPosition.x,0),
 viewRect:new Rect(0, 0, threadToRow.Count*rowWidth, lineHeight),
 alwaysShowHorizontal:false,
 alwaysShowVertical:false,
 horizontalScrollbar:GUIStyle.none,
 verticalScrollbar:GUIStyle.none);
{
 Rect rect = new Rect(0, 0, 0, lineHeight);
 GUIContent content = new GUIContent("", "");

 //timestamp
 rect.width = timeWidth;
 content.text = ""+'\u23F0'; content.tooltip = "Timestamp";
 EditorGUI.LabelField(rect, content, labelStyle);

 //name
 rect.x += rect.width;
 content = new GUIContent("Name", "Name");
 EditorGUI.LabelField(rect, content, labelStyle);

 /*foreach (string id in threadToRow.Keys)
 {
  int rowNum = threadToRow[id];
```

```
   Rect rect = new Rect(rowNum*rowWidth,1,rowWidth,lineHeight);


   EditorGUI.LabelField(rect, id.ToString(), style:EditorStyles.boldLabel);
  }*/
 }
 UnityEngine.GUI.EndScrollView();
}

public void DrawList (Dictionary<string,int> threadToRow)
// if idToRow is null using non-threaded view (1 row)
{
 if (Log.root.subs == null)
  return;

 int totalHeight = 0;
 foreach (Log.Entry entry in Log.root.subs)
  totalHeight += GetEntryHeight(entry, recursively:true);

 Rect valsInternalRect = new Rect();
 valsInternalRect.width = position.width - scrollWidth;
 valsInternalRect.height = totalHeight;

 scrollPosition = UnityEngine.GUI.BeginScrollView(
  position:new Rect(0, toolbarHeight, position.width, position.height-toolbarHeight),
  scrollPosition:scrollPosition,
  viewRect:valsInternalRect,
```

```
  alwaysShowHorizontal:true,
  alwaysShowVertical:true);
 {
  if (labelStyle == null)
   labelStyle = new GUIStyle(UnityEditor.EditorStyles.label);
  labelStyle.alignment = TextAnchor.UpperLeft;

  //background
  EditorGUI.DrawRect(valsInternalRect, new Color(0.9f, 0.9f, 0.9f));

  int lineNum = 0;
  foreach (Log.Entry entry in Log.root.subs)
   DrawEntry(valsInternalRect, ref lineNum, entry);
 }
 UnityEngine.GUI.EndScrollView();
}



public void DrawEntry (Rect listRect, ref int line, Log.Entry entry)
/// returns the number of lines actually drawn
{
 Rect rect = new Rect(listRect.x, listRect.y*lineHeight, 0, lineHeight);
 GUIContent content = new GUIContent("", "");

 //line separator
```

```
EditorGUI.DrawRect(new Rect(listRect.x, listRect.y*line, listRect.width, 1), new Color(0.6f,0.6f,0.6f));

//timestamp
rect.width = timeWidth;
string timeString = $" ({(entry.startTicks/(float)System.TimeSpan.TicksPerMillisecond).ToString("0.0")} ms
content.text = timeString; content.tooltip = timeString;
EditorGUI.LabelField(rect, content, labelStyle);

//name
rect.x += rect.width;
string name = entry.name;
content = new GUIContent(name, name);
EditorGUI.LabelField(rect, content, labelStyle);

//label/foldout
/*if (!recursively || entry.subs==null)
 EditorGUI.LabelField(new Rect(rect.x, rect.y, rect.width, lineHeight), content, labelStyle);
else
 entry.guiExpanded = EditorGUI.Foldout(new Rect(rect.x, rect.y, rect.width, lineHeight), entry.guiExpande

//subs
int counter = 1;
if (entry.guiExpanded && recursively && entry.subs != null)
{
 foreach (Log.Entry sub in entry.subs)
  counter += DrawEntry(new Rect(rect.x+20, rect.y+counter*lineHeight, rect.width-20, rect.y-lineHeight), s
```

```
 }*/

 line++;
}


public Dictionary<string,int> GetIdsToRows ()
/// Generates a lut of idName -> row number if thread view is enabled
{
 HashSet<string> usedIds = Log.UsedThreads();

 Dictionary<string,int> threadToRow = new Dictionary<string,int>(usedIds.Count);
 int counter = 0;

 if (usedIds.Contains(Log.defaultId))
  { threadToRow.Add(Log.defaultId, 0); counter++; }

 List<string> orderedIds = new List<string>();
 orderedIds.AddRange(usedIds);
 orderedIds.Sort();

 foreach (string id in orderedIds)
  if (!threadToRow.ContainsKey(id))
   { threadToRow.Add(id, counter); counter++; }

 return threadToRow;
```

```
}


public void DrawHeaders (Dictionary<string,int> threadToRow)
{
 float rowWidth = (position.width-scrollWidth) / threadToRow.Count;
 if (rowWidth < rowMinWidth) rowWidth = rowMinWidth;

 UnityEngine.GUI.BeginScrollView(
  position:new Rect(0, toolbarHeight, position.width-18, lineHeight),
  scrollPosition:new Vector2(scrollPosition.x,0),
  viewRect:new Rect(0, 0, threadToRow.Count*rowWidth, lineHeight),
  alwaysShowHorizontal:false,
  alwaysShowVertical:false,
  horizontalScrollbar:GUIStyle.none,
  verticalScrollbar:GUIStyle.none);

  foreach (string id in threadToRow.Keys)
  {
   int rowNum = threadToRow[id];
   Rect rect = new Rect(rowNum*rowWidth,1,rowWidth,lineHeight);

   EditorGUI.LabelField(rect, id.ToString(), style:EditorStyles.boldLabel);
  }

 UnityEngine.GUI.EndScrollView();
```

```
}



public void DrawThreadedValues (Dictionary<string,int> threadToRow)
// if idToRow is null using non-threaded view (1 row)
{
 float rowWidth = (position.width-scrollWidth) / threadToRow.Count;
 if (rowWidth < rowMinWidth) rowWidth = rowMinWidth;

 int totalHeight = 0;
 foreach (Log.Entry entry in Log.AllEntries())
  totalHeight += GetEntryHeight(entry);

 Rect valsInternalRect = new Rect();
 valsInternalRect.width = threadToRow.Count*rowWidth;
 valsInternalRect.height = totalHeight;

 scrollPosition = UnityEngine.GUI.BeginScrollView(
  position:new Rect(0, toolbarHeight+lineHeight, position.width, position.height-toolbarHeight-lineHeight),
  scrollPosition:scrollPosition,
  viewRect:valsInternalRect,
  alwaysShowHorizontal:true,
  alwaysShowVertical:true);

 if (labelStyle == null)
   labelStyle = new GUIStyle(UnityEditor.EditorStyles.label);
```

```
  labelStyle.alignment = TextAnchor.UpperLeft;

  //background
  EditorGUI.DrawRect(valsInternalRect, new Color(0.9f, 0.9f, 0.9f));

  //row separators
  for (int i=0; i<threadToRow.Count; i++)
   EditorGUI.DrawRect(new Rect(i*rowWidth, valsInternalRect.y, 1, valsInternalRect.size.y), new Color(0.6

  int currHeight = 0;
  foreach (Log.Entry entry in Log.AllEntries())
  {
   int rowNum = threadToRow[entry.threadName];
   int entryHeight = GetEntryHeight(entry);
   Rect rect = new Rect(rowNum*rowWidth, currHeight, rowWidth, entryHeight);
   DrawEntry(rect, entry);

   currHeight += entryHeight;
  }

 UnityEngine.GUI.EndScrollView();
}

public void DrawValues ()
// if idToRow is null using non-threaded view (1 row)
```

```
{
 if (Log.root.subs == null)
  return;

 int totalHeight = 0;
 foreach (Log.Entry entry in Log.root.subs)
  totalHeight += GetEntryHeight(entry, recursively:true);

 Rect valsInternalRect = new Rect();
 valsInternalRect.width = position.width - scrollWidth;
 valsInternalRect.height = totalHeight;

 scrollPosition = UnityEngine.GUI.BeginScrollView(
  position:new Rect(0, toolbarHeight, position.width, position.height-toolbarHeight),
  scrollPosition:scrollPosition,
  viewRect:valsInternalRect,
  alwaysShowHorizontal:true,
  alwaysShowVertical:true);

  if (labelStyle == null)
   labelStyle = new GUIStyle(UnityEditor.EditorStyles.label);
  labelStyle.alignment = TextAnchor.UpperLeft;

  //background
  EditorGUI.DrawRect(valsInternalRect, new Color(0.9f, 0.9f, 0.9f));
```

```
 int currHeight = 0;
 foreach (Log.Entry entry in Log.root.subs)
 {
  int entryHeight = GetEntryHeight(entry, recursively:true);
  Rect rect = new Rect(valsInternalRect.x, currHeight, valsInternalRect.width, lineHeight);
  DrawEntry(rect, entry, recursively:true);

  currHeight += entryHeight;
 }

 UnityEngine.GUI.EndScrollView();
}


public int DrawEntry (Rect rect, Log.Entry entry, bool recursively=false)
/// returns the number of entries actually drawn
{
 //line separator
 EditorGUI.DrawRect(new Rect(rect.x, rect.y, rect.width, 1), new Color(0.6f,0.6f,0.6f));

 //what's written
 string name = entry.name;
 if (entry.startTicks != 0) name += $" ({((entry.disposeTicks-entry.startTicks)/(float)System.TimeSpan.Ticks
 GUIContent content = new GUIContent(name, name);

 //label/foldout
```

```
 if (!recursively || entry.subs==null)
  EditorGUI.LabelField(new Rect(rect.x, rect.y, rect.width, lineHeight), content, labelStyle);
 else
  entry.guiExpanded = EditorGUI.Foldout(new Rect(rect.x, rect.y, rect.width, lineHeight), entry.guiExpande

 //subs
 int counter = 1;
 if (entry.guiExpanded && recursively && entry.subs != null)
 {
  foreach (Log.Entry sub in entry.subs)
   counter += DrawEntry(new Rect(rect.x+20, rect.y+counter*lineHeight, rect.width-20, rect.y-lineHeight), s
 }

 return counter;
}



public int GetEntryHeight (Log.Entry entry, bool recursively=false)
{
 int height = lineHeight;

 if (recursively && entry.subs != null  &&  entry.guiExpanded)
 {
  foreach (Log.Entry sub in entry.subs)
   height += GetEntryHeight(sub, true);
 }
```

```
    return height;
  }



  [MenuItem ("Window/Log")]
  public static void ShowEditor ()
  {
    EditorWindow.GetWindow<LogWindow>("Log");
  }
 }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;


namespace Den.Tools
{


 [System.Serializable]
 public class Edges<T>
 {
  public CoordRect rect;
  public T[] array;

  public Edges () { }

  public Edges (CoordRect rect)
  {
   this.rect = rect;
   this.array = new T[(rect.size.x+rect.size.z)*2];
  }

  public T this[int x, int z]
  {
```

```
  get
  {
   if (x<rect.offset.x || x>= rect.offset.x+rect.size.x) throw new Exception("Edges: x:" + x + " is out of range:"
   if (z<rect.offset.z || z>= rect.offset.z+rect.size.z) throw new Exception("Edges: z:" + z + " is out of range:"

   if (z==rect.offset.z) return array[x-rect.offset.x];
   if (z==rect.offset.z+rect.size.z-1) return array[rect.size.x + x-rect.offset.x];
   if (x==rect.offset.x) return array[rect.size.x*2 + z-rect.offset.z];
   if (x==rect.offset.x+rect.size.x-1) return array[rect.size.x*2 + rect.size.z + z-rect.offset.z];

   throw new Exception("Edges: improper x:" + x + " and z:" + z + " while rect is:" + rect);
  }
  set
  {
   if (x<rect.offset.x || x>= rect.offset.x+rect.size.x) throw new Exception("Edges: x:" + x + " is out of range:"
   if (z<rect.offset.z || z>= rect.offset.z+rect.size.z) throw new Exception("Edges: z:" + z + " is out of range:"

   if (z==rect.offset.z) { array[x-rect.offset.x] = value; return; }
   if (z==rect.offset.z+rect.size.z-1) { array[rect.size.x + x-rect.offset.x] = value; return; }
   if (x==rect.offset.x) { array[rect.size.x*2 + z-rect.offset.z] = value; return; }
   if (x==rect.offset.x+rect.size.x-1) { array[rect.size.x*2 + rect.size.z + z-rect.offset.z] = value; return; }

   throw new Exception("Edges: improper x:" + x + " and z:" + z + " while rect is:" + rect);
  }
 }
}
```

```
[System.Serializable]
public class FloatEdges : Edges<float>
{
 public FloatEdges () { }

 public FloatEdges (CoordRect rect)
 {
  this.rect = rect;
  this.array = new float[(rect.size.x+rect.size.z)*2];
 }

 public void ReadFloats2D (float[,] heights2D)
 {
  int sizeX = heights2D.GetLength(1);
  int sizeZ = heights2D.GetLength(0);
  if (rect.size.x > sizeX || rect.size.z > sizeZ)
   throw new Exception("Float[" + sizeX + "," + sizeZ +"] is smaller than the edges rect " + rect);

  for (int x=0; x<rect.size.x; x++)
  {
   //this[rect.offset.x+x, rect.offset.z] = heights2D[0, x];
   //this[rect.offset.x+x, rect.offset.z+sizeZ-1] = heights2D[sizeZ-1, x];
```

```
  array[x] = heights2D[0, x];
  array[rect.size.x + x] = heights2D[sizeZ-1, x];
 }

 for (int z=0; z<rect.size.z; z++)
 {
  //this[rect.offset.x, rect.offset.z+z] = heights2D[z, 0];
  //this[rect.offset.x+sizeX-1, rect.offset.z+z] = heights2D[z, sizeX-1];

  array[rect.size.x*2 + z] = heights2D[z, 0];
  array[rect.size.x*2 + rect.size.z + z] = heights2D[z, sizeX-1];
 }
}

public void ReadDelta2D (float[,] heights2D)
/// Reads not the edge values, but the delta between edges and 2nd pixel from edge
{
 int sizeX = heights2D.GetLength(1);
 int sizeZ = heights2D.GetLength(0);
 if (rect.size.x > sizeX || rect.size.z > sizeZ)
  throw new Exception("Float[" + sizeX + "," + sizeZ +"] is smaller than the edges rect " + rect);

 for (int x=0; x<rect.size.x; x++)
 {
  array[x] = heights2D[0,x] - heights2D[1,x];
  array[rect.size.x + x] = heights2D[sizeZ-1,x] - heights2D[sizeZ-2,x];
```

```
 }

 for (int z=0; z<rect.size.z; z++)
 {
  array[rect.size.x*2 + z] = heights2D[z,0] - heights2D[z,1];
  array[rect.size.x*2 + rect.size.z + z] = heights2D[z, sizeX-1] - heights2D[z, sizeX-2];
 }
}

public void ReadFloats3D (float[,,] splats2D, int ch)
{
 int sizeX = splats2D.GetLength(1);
 int sizeZ = splats2D.GetLength(0);
 if (rect.size.x > sizeX || rect.size.z > sizeZ)
  throw new Exception("Float[" + sizeX + "," + sizeZ +"] is smaller than the edges rect " + rect);

 for (int x=0; x<rect.size.x; x++)
 {
  array[x] = splats2D[0, x, ch];
  array[rect.size.x + x] = splats2D[sizeZ-1, x, ch];
 }

 for (int z=0; z<rect.size.z; z++)
 {
  array[rect.size.x*2 + z] = splats2D[z, 0, ch];
  array[rect.size.x*2 + rect.size.z + z] = splats2D[z, sizeX-1, ch];
```

```
 }
}

public void ReadCornersFloats2D (float[,] heights2D)
{
 int sizeX = heights2D.GetLength(1);
 int sizeZ = heights2D.GetLength(0);
 if (rect.size.x > sizeX || rect.size.z > sizeZ)
  throw new Exception("Float[" + sizeX + "," + sizeZ +"] is smaller than the edges rect " + rect);

 array[0] = heights2D[0, 0];
 array[rect.size.x] = heights2D[sizeZ-1, 0];
 array[rect.size.x-1] = heights2D[0, sizeX-1];
 array[rect.size.x*2-1] = heights2D[sizeZ-1, sizeX-1];
}




public static void Weld_z (FloatEdges edges, float[,] heights2D)
{
 for (int x=0; x<edges.rect.size.x; x++)
 {
  float edgeVal = edges.array[edges.rect.size.x + x];
  heights2D[0, x] = edgeVal;
 }
}
```

```
public static void Weld_Z (FloatEdges edges, float[,] heights2D)
{
 for (int x=0; x<edges.rect.size.x; x++)
 {
  float edgeVal = edges.array[x];
  heights2D[edges.rect.size.z-1, x] = edgeVal;
 }
}

public static void Weld_x (FloatEdges edges, float[,] heights2D)
{
 for (int z=0; z<edges.rect.size.z; z++)
 {
  float edgeVal = edges.array[edges.rect.size.x*2 + edges.rect.size.z + z];
  heights2D[z, 0] = edgeVal;
 }
}

public static void Weld_X (FloatEdges edges, float[,] heights2D)
{
 for (int z=0; z<edges.rect.size.z; z++)
 {
  float edgeVal = edges.array[edges.rect.size.x*2 + z];
  heights2D[z,edges.rect.size.x-1] = edgeVal;
 }
```

```
}

public static void Weld (FloatEdges edges, Coord direction, float[,] heights2D)
{
 if (direction.x == 1) Weld_X(edges, heights2D);
 else if (direction.x == -1) Weld_x(edges, heights2D);
 else if (direction.z == 1) Weld_Z(edges, heights2D);
 else if (direction.z == -1) Weld_z(edges, heights2D);
}


///Same weld using 3d array
public static void Weld_z (FloatEdges edges, float[,,] splats3D, int ch)
{
 for (int x=0; x<edges.rect.size.x; x++)
 {
  float edgeVal = edges.array[edges.rect.size.x + x];
  splats3D[0, x, ch] = edgeVal;
 }
}

public static void Weld_Z (FloatEdges edges, float[,,] splats3D, int ch)
{
 for (int x=0; x<edges.rect.size.x; x++)
 {
  float edgeVal = edges.array[x];
```

```
  splats3D[edges.rect.size.z-1, x, ch] = edgeVal;
 }
}

public static void Weld_x (FloatEdges edges, float[,,] splats3D, int ch)
{
 for (int z=0; z<edges.rect.size.z; z++)
 {
  float edgeVal = edges.array[edges.rect.size.x*2 + edges.rect.size.z + z];
  splats3D[z, 0, ch] = edgeVal;
 }
}

public static void Weld_X (FloatEdges edges, float[,,] splats3D, int ch)
{
 for (int z=0; z<edges.rect.size.z; z++)
 {
  float edgeVal = edges.array[edges.rect.size.x*2 + z];
  splats3D[z,edges.rect.size.x-1, ch] = edgeVal;
 }
}

public static void Weld (FloatEdges edges, Coord direction, float[,,] splats3D, int ch)
{
 if (direction.x == 1) Weld_X(edges, splats3D, ch);
 else if (direction.x == -1) Weld_x(edges, splats3D, ch);
```

```
 else if (direction.z == 1) Weld_Z(edges, splats3D, ch);
 else if (direction.z == -1) Weld_z(edges, splats3D, ch);
}



///Welds preserving normals and using margins
public static void Weld_z (FloatEdges edges, FloatEdges deltas, float[,] heights2D, int margins=10)
{
 for (int x=0; x<edges.rect.size.x; x++)
 {
  float edgeVal = edges.array[edges.rect.size.x + x];
  float deltaVal = deltas.array[edges.rect.size.x + x];
  heights2D[0, x] = edgeVal;
   float innerDelta = (edgeVal + deltaVal) - heights2D[1,x];
  heights2D[1,x] = edgeVal + deltaVal;

  //if (x>margins && x<edges.rect.size.x-margins-1)
   for (int i=0; i<margins; i++)
    heights2D[2+i,x] += innerDelta * (1 - 1f*i/margins);
 }
}

public static void Weld_Z (FloatEdges edges, FloatEdges deltas, float[,] heights2D, int margins=10)
{
 for (int x=0; x<edges.rect.size.x; x++)
```

```
 {
  float edgeVal = edges.array[x];
  float deltaVal = deltas.array[x];
  heights2D[edges.rect.size.z-1, x] = edgeVal;
   float innerDelta = (edgeVal + deltaVal) - heights2D[edges.rect.size.z-2, x];
  heights2D[edges.rect.size.z-2, x] = edgeVal + deltaVal;

  //if (x>margins && x<edges.rect.size.x-margins-1)
   for (int i=0; i<margins; i++)
    heights2D[edges.rect.size.z-3-i, x] += innerDelta * (1 - 1f*i/margins);
 }
}

public static void Weld_x (FloatEdges edges, FloatEdges deltas, float[,] heights2D, int margins=10)
{
 for (int z=0; z<edges.rect.size.z; z++)
 {
  float edgeVal = edges.array[edges.rect.size.x*2 + edges.rect.size.z + z];
  float deltaVal = deltas.array[edges.rect.size.x*2 + edges.rect.size.z + z];
  heights2D[z, 0] = edgeVal;
   float innerDelta = (edgeVal + deltaVal) - heights2D[z, 1];
  heights2D[z, 1] = edgeVal + deltaVal;

  //if (z>margins && z<edges.rect.size.z-margins-1)
   for (int i=0; i<margins; i++)
    heights2D[z, 2+i] += innerDelta * (1 - 1f*i/margins);
```

```
 }
}

public static void Weld_X (FloatEdges edges, FloatEdges deltas, float[,] heights2D, int margins=10)
{
 for (int z=0; z<edges.rect.size.z; z++)
 {
  float edgeVal = edges.array[edges.rect.size.x*2 + z];
  float deltaVal = deltas.array[edges.rect.size.x*2 + z];
  heights2D[z,edges.rect.size.x-1] = edgeVal;
   float innerDelta = (edgeVal + deltaVal) - heights2D[z,edges.rect.size.x-2];
  heights2D[z,edges.rect.size.x-2] = edgeVal + deltaVal;

  //if (z>margins && z<edges.rect.size.z-margins-1)
   for (int i=0; i<margins; i++)
    heights2D[z,edges.rect.size.x-3-i] += innerDelta * (1 - 1f*i/margins);
 }
}

public static void Weld (FloatEdges edges, FloatEdges deltas, Coord direction, float[,] heights2D, int marg
{
 if (direction.x == 1) Weld_X(edges, deltas, heights2D, margins);
 else if (direction.x == -1) Weld_x(edges, deltas, heights2D, margins);
 else if (direction.z == 1) Weld_Z(edges, deltas, heights2D, margins);
 else if (direction.z == -1) Weld_z(edges, deltas, heights2D, margins);
}
```

```
 }



}
```

```
ï»¿// Operations with "maps"
// Note that functions are not inlined in editor, so keeping all method unfolded

using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;

[assembly: System.Runtime.CompilerServices.InternalsVisibleTo("Tests")]

namespace Den.Tools.Matrices
{
 [Serializable, StructLayout (LayoutKind.Sequential)] //to pass to native
 public class Matrix // : Matrix2D<float>
 {
  public CoordRect rect; //never assign it's size manually, use ChangeRect
  public int count;
  public float[] arr;

  public const bool native = true;

  #region Constructors

   public Matrix () { arr = new float[0]; rect = new CoordRect(0,0,0,0); count=0;  } //for serializer
```

```
public Matrix (int offsetX, int offsetZ, int sizeX, int sizeZ, float[] array=null)
{
 rect = new CoordRect(offsetX, offsetZ, sizeX, sizeZ);
 count = rect.Count;
 DefineArray(array);
}

public Matrix (CoordRect rect, float[] array=null)
{
 this.rect = rect;
 count = rect.Count;
 DefineArray(array);
}

public Matrix (Coord offset, Coord size, float[] array=null)
{
 rect = new CoordRect(offset, size);
 count = rect.Count;
 DefineArray(array);
}

public Matrix (Matrix src, float[] array=null)
{
 rect = src.rect;
 count = rect.Count;
```

```
 DefineArray(array);
 Array.Copy(src.arr, arr, arr.Length);
}

public Matrix (Texture2D texture, int channel=-1)
{
 rect = new CoordRect(0,0, texture.width, texture.height);
 count = rect.Count;
 arr = new float[count];

 Color[] colors = texture.GetPixels();
 ImportColors(colors, texture.width, texture.height, channel);
}

protected void DefineArray (float[] array=null)
{
 int matrixCount = rect.size.x*rect.size.z;
 if (array != null)
 {
  if (array.Length < matrixCount)
   throw new Exception("Array length: " + array.Length + " is lower then matrix capacity: " + matrixCount);
  else arr = array;
 }
 else arr = new float[matrixCount];
}
```

```
#endregion



#region Get/Set


 public float this[int x, int z]

 {

  get { return arr[(z-rect.offset.z)*rect.size.x + x - rect.offset.x]; } //rect fn duplicated to increase performanc

  set { arr[(z-rect.offset.z)*rect.size.x + x - rect.offset.x] = value; }

 }



 public float this[Coord c]

 {

  get { return arr[(c.z-rect.offset.z)*rect.size.x + c.x - rect.offset.x]; }

  set { arr[(c.z-rect.offset.z)*rect.size.x + c.x - rect.offset.x] = value; }

 }



 public float GetFloored (float fx, float fz)

 {

  int ix = (int)(float)fx; if (fx<0) ix--;

  int iz = (int)(float)fz; if (fz<0) iz--;


  return arr[(iz-rect.offset.z)*rect.size.x + ix - rect.offset.x];

 }
```

```
public float GetInterpolated (float fx, float fz, bool roundToShort=false)
/// Has native duplicate, but since it's called per-pixel better use this directly
{
 int ix = (int)fx; if (fx<0) ix--; if (ix==rect.offset.x+rect.size.x) ix--;
 int iz = (int)fz; if (fz<0) iz--; if (iz==rect.offset.z+rect.size.z) iz--;

 float xPercent = fx-ix;
 float zPercent = fz-iz;

 if (ix<rect.offset.x) ix = rect.offset.x;
 if (iz<rect.offset.z) iz = rect.offset.z;
 int ix1 = ix+1;  if (ix1>=rect.offset.x+rect.size.x) ix1 = rect.offset.x+rect.size.x-1;
 int iz1 = iz+1;  if (iz1>=rect.offset.z+rect.size.z) iz1 = rect.offset.z+rect.size.z-1;

 float val1 = arr[(iz-rect.offset.z)*rect.size.x + ix - rect.offset.x]; //this[ix,iz];
 float val2 = arr[(iz-rect.offset.z)*rect.size.x + ix1 - rect.offset.x]; //this[ix1,iz];

 //if (roundToShort) { val1 = (float)(int)(val1*32767)/32767; val2 = (float)(int)(val2*32767)/32767; }
 float val3 = val1*(1-xPercent) + val2*xPercent;

 float val4 = arr[(iz1-rect.offset.z)*rect.size.x + ix - rect.offset.x]; //this[ix,iz1];
 float val5 = arr[(iz1-rect.offset.z)*rect.size.x + ix1 - rect.offset.x]; //this[ix1,iz1];

 //if (roundToShort) { val3 = (float)(int)(val3*32767)/32767; val4 = (float)(int)(val4*32767)/32767; }
```

```
 float val6 = val4*(1-xPercent) + val5*xPercent;


 return val3*(1-zPercent) + val6*zPercent;
}



public float GetRelative (float rx, float rz)
/// Where rx and rz are 0-1 in percent relative to matrix rect
{
 float fx = rx*rect.size.x + rect.offset.x;
 float fz = rz*rect.size.z + rect.offset.z;

 return GetInterpolated(fx,fz);
}



public float GetRelative (float rx, float rz, CoordRect customRect)
/// Same GetRelative, but 0-1 range is relative to custom rect (for example, active zone)
{
 float fx = rx*customRect.size.x + customRect.offset.x;
 float fz = rz*customRect.size.z + customRect.offset.z;

 return GetInterpolated(fx,fz);
}
```

```
public float GetRelativeWithRotation (float sx, float sz, Vector2D rotationDirection, Vector2D rotationPivot
/// Same GetRelative, but 0-1 range is relative to custom rect (for example, active zone)
/// rotationPivot range is 0-1 too
{
 float cx = sx - rotationPivot.x;  float cz = sz - rotationPivot.z;

 Vector2D vx = rotationDirection * cx;
 Vector2D vz = new Vector2D(-rotationDirection.z, rotationDirection.x) * cz;
 Vector2D v = vx+vz;

 v.x += rotationPivot.x;  v.z += rotationPivot.z;

 if (v.x < 0 || v.x > 1) return 0;
 if (v.z < 0 || v.z > 1) return 0;

 float fx = v.x*customRect.size.x + customRect.offset.x;
 float fz = v.z*customRect.size.z + customRect.offset.z;

 return GetInterpolated(fx,fz);
}

#endregion


#region Import
```

```
public void ImportTexture (Texture2D tex, int channel=-1) { ImportTexture(tex, rect.offset, channel); }
//texOffset == matrix.rect.offset by default, so it will read texture from 0,0
//when matrix bigger - reads whole texture as matrix part
//when texture bigger - reads only the matrix part form it

public void ImportTexture (Texture2D tex, Coord texOffset, int channel=-1)
/// Imports texture using color arrays.
/// If channel -1 reads avg of 3 colors
{
 Coord texSize = new Coord(tex.width, tex.height);
 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(texOffset,texSize)); //to get array s

 Color[] colors = tex.GetPixels(intersection.offset.x-texOffset.x, intersection.offset.z-texOffset.z, intersecti
 ImportColors(colors, intersection.offset, intersection.size, channel);
}


public void ImportTextureRaw (Texture2D tex, int channel=0) { ImportTextureRaw(tex, rect.offset, channe
public void ImportTextureRaw (Texture2D tex, Coord texOffset, int channel=0)
/// Uses raw texture bytes instead of color arrays. Faster, but requires specific non-compressed format
{
 Coord texSize = new Coord(tex.width, tex.height);

 TextureFormat format = tex.format;
 if (format!=TextureFormat.RGBA32 && format!=TextureFormat.ARGB32 && format!=TextureFormat.RG
  throw new Exception("Matrix export: raw texture format is not supported");
```

```
 byte[] bytes = tex.GetRawTextureData();

 switch(format)
 {
   case TextureFormat.RGBA32: ImportRawBytes(bytes, texOffset, texSize, channel, 4); break;
   case TextureFormat.ARGB32: channel++; if (channel == 5) channel = 0; ImportRawBytes(bytes, texOff
   case TextureFormat.RGB24: ImportRawBytes(bytes, texOffset, texSize, channel, 3); break;
   case TextureFormat.R8: ImportRawBytes(bytes, texOffset, texSize, 0, 1); break;
   case TextureFormat.R16: ImportRaw16(bytes, texOffset, texSize); break;
 }
}


public void ImportColors (Color[] colors, int width, int height, int channel=-1) { ImportColors(colors, rect.of
public void ImportColors (Color[] colors, Coord colorsSize, int channel=-1) { ImportColors(colors, rect.offs
public void ImportColors (Color[] colors, Coord colorsOffset, Coord colorsSize,  int channel=-1)
{
 //if (colors.Length != colorsSize.x*colorsSize.z)
 // throw new Exception("Array count does not match texture dimensions");

 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(colorsOffset, colorsSize));
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
```

```
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int colorsPos = (z-colorsOffset.z)*colorsSize.x + x - colorsOffset.x;

   float val;
   switch (channel)
   {
    case 0: val = colors[colorsPos].r; break;
    case 1: val = colors[colorsPos].g; break;
    case 2: val = colors[colorsPos].b; break;
    case 3: val = colors[colorsPos].a; break;
    default: val = (colors[colorsPos].r + colors[colorsPos].g + colors[colorsPos].b)/3; break;
   }

   arr[matrixPos] = val;
  }
}



public void ImportRawBytes (byte[] bytes, int width, int height, int start, int step) { ImportRawBytes(bytes,
public void ImportRawBytes (byte[] bytes, Coord bytesSize, int start, int step) { ImportRawBytes(bytes, re

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

 public void ImportRawBytes (byte[] bytes, Coord bytesOffset, Coord bytesSize, int start, int step)
 {
```

```
 if (bytes.Length != bytesSize.x*bytesSize.z*step &&
  (bytes.Length < bytesSize.x*bytesSize.z*step*1.3f || bytes.Length > bytesSize.x*bytesSize.z*step*1.36
   throw new Exception("Array count does not match texture dimensions");

 ImportRawBytes(this, bytes, bytes.Length, bytesOffset, bytesSize, start, step);
}

[DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Matrix_ImportRaw
public static extern void ImportRawBytes (Matrix thism, byte[] bytes, int bytesLength, Coord bytesOffset,

#else

public void ImportRawBytes (byte[] bytes, Coord bytesOffset, Coord bytesSize, int start, int step)
{
 if (bytes.Length != bytesSize.x*bytesSize.z*step &&
  (bytes.Length < bytesSize.x*bytesSize.z*step*1.3f || bytes.Length > bytesSize.x*bytesSize.z*step*1.36
   throw new Exception("Array count does not match texture dimensions");

 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(bytesOffset, bytesSize));
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int bytesPos = (z-bytesOffset.z)*bytesSize.x + x - bytesOffset.x;
```

```
    bytesPos = bytesPos * step + start;

    float val = bytes[bytesPos] / 255f;  //matrix has the range 0-1 _inclusive_, it could be 1, so using 255
    arr[matrixPos] = val;
   }
 }
#endif



public void ImportRaw16 (byte[] bytes, int width, int height) { ImportRaw16(bytes, rect.offset, new Coord(v
public void ImportRaw16 (byte[] bytes, Coord texSize) { ImportRaw16(bytes, rect.offset, texSize); }

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void ImportRaw16 (byte[] bytes, Coord texOffset, Coord texSize)
 {
  if (texSize.x*texSize.z*2 > bytes.Length) //extra bytes could be mipmaps
   throw new Exception("Array count does not match texture dimensions");

  ImportRaw16(this, bytes, bytes.Length, texOffset, texSize);
 }

 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Matrix_ImportRaw
 private static extern void ImportRaw16 (Matrix thism, byte[] bytes, int bytesLength, Coord texOffset, Coo

#else
```

```
 public void ImportRaw16 (byte[] bytes, Coord texOffset, Coord texSize)
 {
  if (texSize.x*texSize.z*2 > bytes.Length) //extra bytes could be mipmaps
   throw new Exception("Array count does not match texture dimensions");

  CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(texOffset, texSize));
  Coord min = intersection.Min; Coord max = intersection.Max;

  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
    int bytesPos = (z-texOffset.z)*texSize.x + x - texOffset.x;
    bytesPos *= 2;

    float val = (bytes[bytesPos+1]*256f + bytes[bytesPos]) / 65536f;  //65025f;
    arr[matrixPos] = val;
   }
 }
#endif


public void ImportRawFloat (byte[] bytes, int width, int height, float mult=1) { ImportRawFloat(bytes, new C
public void ImportRawFloat (byte[] bytes, Coord texSize, float mult=1) { ImportRawFloat(bytes, texSize, r
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

 public void ImportRawFloat (byte[] bytes, Coord texOffset, Coord texSize, float mult=1)
 {
  int numPixels = texSize.x*texSize.z;
  if (numPixels*4 > bytes.Length) //extra bytes could be mipmaps
   throw new Exception("Array count does not match texture dimensions");

  ImportRawFloat(this, bytes, bytes.Length, texOffset, texSize, mult);
 }

 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Matrix_ImportRaw
 public static extern void ImportRawFloat (Matrix thism, byte[] bytes, int bytesLength, Coord texOffset, Co

#else

 public void ImportRawFloat (byte[] bytes, Coord texOffset, Coord texSize, float mult=1)
 {
  int numPixels = texSize.x*texSize.z;
  if (numPixels*4 > bytes.Length) //extra bytes could be mipmaps
   throw new Exception("Array count does not match texture dimensions");

  CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(texOffset, texSize));
  Coord min = intersection.Min; Coord max = intersection.Max;

  FloatToBytes converter = new FloatToBytes();
```

```
  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
    int bytesPos = (z-texOffset.z)*texSize.x + x - texOffset.x;
    bytesPos *= 4;

    converter.b0 = bytes[bytesPos];
    converter.b1 = bytes[bytesPos+1];
    converter.b2 = bytes[bytesPos+2];
    converter.b3 = bytes[bytesPos+3];

    arr[matrixPos] = converter.f * mult;
   }
  }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

 public void ImportHeights (float[,] heights)
 {
  Coord heightsSize = new Coord(heights.GetLength(1), heights.GetLength(0));
  ImportHeights (this, heights, rect.offset, heightsSize);
 }
```

```
public void ImportHeights (float[,] heights, Coord heightsOffset)
{
 Coord heightsSize = new Coord(heights.GetLength(1), heights.GetLength(0));
 ImportHeights (this, heights, heightsOffset, heightsSize);
}

[DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Matrix_ImportHeig
private static extern void ImportHeights (Matrix thism, float[,] heights, Coord heightsOffset, Coord height

#else

public void ImportHeights (float[,] heights) { ImportHeights(heights, rect.offset); }
public void ImportHeights (float[,] heights, Coord heightsOffset)
{
 Coord heightsSize = new Coord(heights.GetLength(1), heights.GetLength(0));  //x and z swapped
 CoordRect heightsRect = new CoordRect(heightsOffset, heightsSize);

 CoordRect intersection = CoordRect.Intersected(rect, heightsRect);
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int heightsPosX = x - heightsRect.offset.x;
```

```
    int heightsPosZ = z - heightsRect.offset.z;


    arr[matrixPos] = heights[heightsPosZ, heightsPosX];
   }
 }
#endif

public void ImportHeightStrips (float[][,] heights) { ImportHeightStrips(heights, rect.offset); }
public void ImportHeightStrips (float[][,] heights, Coord heightsOffset)
{
 //TODO: offset doesnt work
 int offset = 0;
 for (int s=0; s<heights.Length; s++)
 {
  ImportHeights(heights[s], new Coord(0,offset));
  offset+=heights[s].GetLength(0);
 }
}


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

 public void ImportSplats (float[,,] splats, int channel)
 {
  Coord splatsSize = new Coord(splats.GetLength(1), splats.GetLength(0));
  ImportSplats (this, splats, rect.offset, splatsSize, splats.GetLength(2), channel);
```

```
}

public void ImportSplats (float[,,] splats, Coord heightsOffset, int channel)
{
 Coord splatsSize = new Coord(splats.GetLength(1), splats.GetLength(0));
 ImportSplats (this, splats, heightsOffset, splatsSize, splats.GetLength(2), channel);
}

[DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Matrix_ImportSpla
private static extern void ImportSplats (Matrix thism, float[,,] splats, Coord splatsOffset, Coord splatsSize

#else

public void ImportSplats (float[,,] splats, int channel) { ImportSplats(splats, rect.offset, channel); }
public void ImportSplats (float[,,] splats, Coord splatsOffset, int channel)
{
 Coord splatsSize = new Coord(splats.GetLength(1), splats.GetLength(0));  //x and z swapped
 CoordRect splatsRect = new CoordRect(splatsOffset, splatsSize);

 CoordRect intersection = CoordRect.Intersected(rect, splatsRect);
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
```

```
    int heightsPosX = x - splatsRect.offset.x;
    int heightsPosZ = z - splatsRect.offset.z;

    arr[matrixPos] = splats[heightsPosZ, heightsPosX, channel];
   }
  }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

 public void ImportDetail (int[,] detail, float density=1)
 {
  Coord detailSize = new Coord(detail.GetLength(1), detail.GetLength(0));
  ImportDetail (this, detail, rect.offset, detailSize, density);
 }

 public void ImportDetail (int[,] detail, Coord detailOffset, float density=1)
 {
  Coord detailSize = new Coord(detail.GetLength(1), detail.GetLength(0));
  ImportDetail (this, detail, detailOffset, detailSize, density);
 }

 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Matrix_ImportDeta
 private static extern void ImportDetail (Matrix thism,  int[,] detail, Coord detailOffset, Coord detailSize, flo
```

```
#else

 public void ImportDetail (int[,] detail, float density=1) => ImportDetail(detail, rect.offset);
 public void ImportDetail (int[,] detail, Coord detailOffset, float density=1) //aka Grass
 {
  Coord splatsSize = new Coord(detail.GetLength(1), detail.GetLength(0));  //x and z swapped
  CoordRect detailRect = new CoordRect(detailOffset, splatsSize);

  CoordRect intersection = CoordRect.Intersected(rect, detailRect);
  Coord min = intersection.Min; Coord max = intersection.Max;

  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
    int detailPosX = x - detailRect.offset.x;
    int detailPosZ = z - detailRect.offset.z;

    float val = detail[detailPosZ, detailPosX];
    arr[matrixPos] = val / density;
   }
 }
#endif


public void ImportData (TerrainData data, int channel=-1) { ImportData (data, rect.offset, channel=-1); }
```

```
public void ImportData (TerrainData data, Coord dataOffset, int channel=-1)
/// Partial terrain data (loading only the part intersecting with matrix). Do not work in thread!
/// If channel is -1 getting height
{
 int resolution = channel==-1 ?  data.heightmapResolution : data.alphamapResolution;

 Coord dataSize = new Coord(resolution, resolution);
 CoordRect dataIntersection = CoordRect.Intersected(rect, new CoordRect(dataOffset, dataSize));
 if (dataIntersection.size.x==0 || dataIntersection.size.z==0) return;

 if (channel == -1)
 {
  float[,] heights = data.GetHeights(dataIntersection.offset.x-dataOffset.x, dataIntersection.offset.z-dataOf
  ImportHeights(heights, dataIntersection.offset);
 }

 else
 {
  float[,,] splats = data.GetAlphamaps(dataIntersection.offset.x-dataOffset.x, dataIntersection.offset.z-data
  ImportSplats(splats, dataIntersection.offset, channel);
 }
}

#endregion
```

```
#region Export

 public void ExportTexture (Texture2D tex, int channel=-1) { ExportTexture(tex, rect.offset, channel); }
 //texOffset == matrix.rect.offset by default, so it will write texture from matrix's start
 //when matrix bigger - writes whole texture, disregards unnecessary data
 //when texture bigger - writes only the matrix part to it

 public void ExportTexture (Texture2D tex, Coord texOffset, int channel=-1)
 {
  Coord texSize = new Coord(tex.width, tex.height);

  CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(texOffset,texSize)); //to get array s

  Color[] colors;
  if (channel < 0) //will re-create all colors anyways
   colors = new Color[intersection.size.x * intersection.size.z];
  else
   colors = tex.GetPixels(intersection.offset.x-texOffset.x, intersection.offset.z-texOffset.z, intersection.size

  ExportColors(colors, intersection.offset, intersection.size, channel);
  tex.SetPixels(intersection.offset.x-texOffset.x, intersection.offset.z-texOffset.z, intersection.size.x, interse

  tex.Apply();
 }
```

```
public void ExportTextureRaw (Texture2D tex)
/// Will overwrite all data (not partial)
/// More of a snippet since could be mostly done in thread
{
 if (tex.width!=rect.size.x || tex.height!=rect.size.z)
  throw new Exception("Matrix export: matrix size and texture resolution mismatch (tex:" + tex.width +"*" +

 byte[] bytes;

 switch (tex.format)
 {
  case TextureFormat.RGBA32:
   bytes = new byte[rect.Count*4];
   ExportRawBytes(bytes, rect.offset, rect.size, 0, 4);
   ExportRawBytes(bytes, rect.offset, rect.size, 1, 4);
   ExportRawBytes(bytes, rect.offset, rect.size, 2, 4);
   break;

  case TextureFormat.ARGB32:
   bytes = new byte[rect.Count*4];
   ExportRawBytes(bytes, rect.offset, rect.size, 1, 4);
   ExportRawBytes(bytes, rect.offset, rect.size, 2, 4);
   ExportRawBytes(bytes, rect.offset, rect.size, 3, 4);
   break;

  case TextureFormat.RGB24:
```

```
  bytes = new byte[rect.Count*3];
  ExportRawBytes(bytes, rect.offset, rect.size, 0, 3);
  ExportRawBytes(bytes, rect.offset, rect.size, 1, 3);
  ExportRawBytes(bytes, rect.offset, rect.size, 2, 3);
  break;

 case TextureFormat.R8:
  bytes = new byte[rect.Count];
  ExportRawBytes(bytes, rect.offset, rect.size, 0, 1);
  break;

 case TextureFormat.R16:
  bytes = new byte[rect.Count*2];
  ExportRaw16(bytes, rect.offset, rect.size);
  break;

 case TextureFormat.RFloat:
  bytes = new byte[rect.Count*4];
  ExportRawFloat(bytes, rect.offset, rect.size);
  break;

 default:
  throw new Exception("Matrix export: raw texture format is not supported (" + tex.format + ")");
}

tex.LoadRawTextureData(bytes);
```

```
 tex.Apply();
}



public void ExportTextureRaw (Texture2D tex, Coord texOffset, int channel=-1)
{
 TextureFormat format = tex.format;
 if (format!=TextureFormat.RGBA32 && format!=TextureFormat.ARGB32 && format!=TextureFormat.RG
  throw new Exception("Matrix export: raw texture format is not supported");

 Coord texSize = new Coord(tex.width, tex.height);

 byte[] bytes = tex.GetRawTextureData();  //to use part of the texture and to apply only one channel

 switch(format)
 {
   case TextureFormat.RGBA32: ExportRawBytes(bytes, texOffset, texSize, channel, 4); break;
   case TextureFormat.ARGB32: channel++; if (channel == 5) channel = 0; ExportRawBytes(bytes, texOf
   case TextureFormat.RGB24: ExportRawBytes(bytes, texOffset, texSize, channel, 3); break;
   case TextureFormat.R8: ExportRawBytes(bytes, texOffset, texSize, 0, 1); break;
   case TextureFormat.R16: ExportRaw16(bytes, texOffset, texSize); break;
   case TextureFormat.RFloat: ExportRawFloat(bytes, texOffset, texSize); break;
 }

 tex.LoadRawTextureData(bytes);
 tex.Apply();
```

```
}


public void ExportColors (Color[] colors, int width, int height, int channel=-1, bool markOutrange=true, Ma
public void ExportColors (Color[] colors, Coord colorsSize, int channel=-1, bool markOutrange=true, Matr
public void ExportColors (Color[] colors, Coord colorsOffset, Coord colorsSize,  int channel=-1, bool mark
{
 if (colors.Length != colorsSize.x*colorsSize.z)
  throw new Exception("Array count does not match texture dimensions");

 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(colorsOffset, colorsSize));
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int colorsPos = (z-colorsOffset.z)*colorsSize.x + x - colorsOffset.x;

   float val = arr[matrixPos];
   if (mask != null)
    val *= mask.arr[matrixPos];

   //if (float.IsNaN(val))  colors[colorsPos] = new Color(0,0,1,0);
   if (val > 1  &&  markOutrange) colors[colorsPos] = new Color(0,1,0,0);
   else if (val < 0  &&  markOutrange) colors[colorsPos] = new Color(1,0,0,0);
```

```
   else switch (channel)
   {
    case 0: colors[colorsPos].r = val; break;
    case 1: colors[colorsPos].g = val; break;
    case 2: colors[colorsPos].b = val; break;
    case 3: colors[colorsPos].a = val; break;
    default: colors[colorsPos].r = val; colors[colorsPos].g =val; colors[colorsPos].b = val; colors[colorsPos].
   }
  }
}



public void ExportRawBytes (byte[] bytes, int width, int height, int start, int step) { ExportRawBytes(bytes,
public void ExportRawBytes (byte[] bytes, Coord bytesSize, int start, int step) { ExportRawBytes(bytes, by

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

 public void ExportRawBytes (byte[] bytes, Coord bytesOffset, Coord bytesSize, int start, int step)
 {
  if (bytes.Length != bytesSize.x*bytesSize.z*step &&
   (bytes.Length < bytesSize.x*bytesSize.z*step*1.3f || bytes.Length > bytesSize.x*bytesSize.z*step*1.36
    throw new Exception("Array count does not match texture dimensions");

  ExportRawBytes(this, bytes, bytes.Length, bytesOffset, bytesSize, start, step);
 }
```

```
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Matrix_ExportRaw
 public static extern void ExportRawBytes (Matrix thism, byte[] bytes, int bytesLength, Coord bytesOffset,

#else

 public void ExportRawBytes (byte[] bytes, Coord bytesOffset, Coord bytesSize, int start, int step)
 {
  if (bytes.Length != bytesSize.x*bytesSize.z*step &&
   (bytes.Length < bytesSize.x*bytesSize.z*step*1.3f || bytes.Length > bytesSize.x*bytesSize.z*step*1.36
    throw new Exception("Array count does not match texture dimensions");

  CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(bytesOffset, bytesSize));
  Coord min = intersection.Min; Coord max = intersection.Max;

  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
    int bytesPos = (z-bytesOffset.z)*bytesSize.x + x - bytesOffset.x;
    bytesPos = bytesPos * step + start;

    float val = arr[matrixPos];
    bytes[bytesPos] = (byte)(val * 255f); //matrix has the range 0-1 _inclusive_, it could be 1
   }
 }
#endif
```

```
public void ExportRaw16 (byte[] bytes, int width, int height) { ExportRaw16(bytes, new Coord(width,heigh
public void ExportRaw16 (byte[] bytes, Coord texSize) { ExportRaw16(bytes, texSize, rect.offset); }

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

 public void ExportRaw16 (byte[] bytes, Coord texOffset, Coord texSize)
 {
  if (texSize.x*texSize.z*2 != bytes.Length)
   throw new Exception("Array count does not match texture dimensions");
  ExportRaw16(this, bytes, bytes.Length, texOffset, texSize);
 }

 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Matrix_ExportRaw
 private static extern void ExportRaw16 (Matrix thism, byte[] bytes, int bytesLength, Coord texOffset, Coo

#else

 public void ExportRaw16 (byte[] bytes, Coord texOffset, Coord texSize)
 {
  if (texSize.x*texSize.z*2 != bytes.Length)
   throw new Exception("Array count does not match texture dimensions");

  CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(texOffset, texSize));
  Coord min = intersection.Min; Coord max = intersection.Max;
```

```
  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
    int bytesPos = (z-texOffset.z)*texSize.x + x - texOffset.x;
    bytesPos *= 2;

    float val = arr[matrixPos]; //this[x+regionRect.offset.x, z+regionRect.offset.z];

    int intVal = (int)(val*65536);
    if (intVal>=65536) intVal = 65535;
    bytes[bytesPos] = (byte)(intVal & 0xFF);
    bytes[bytesPos+1] = (byte)(intVal>>8);
   }
 }
#endif


public void ExportRawFloat (byte[] bytes, int width, int height, float mult=1) { ExportRawFloat(bytes, new
public void ExportRawFloat (byte[] bytes, Coord texSize, float mult=1) { ExportRawFloat(bytes, texSize, r

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

 public void ExportRawFloat (byte[] bytes, Coord texOffset, Coord texSize, float mult=1)
 {
```

```
 int numPixels = texSize.x*texSize.z;
 if (numPixels*4 != bytes.Length)
  throw new Exception("Array count does not match texture dimensions");

 ExportRawFloat(this, bytes, bytes.Length, texOffset, texSize, mult);
}

[DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Matrix_ExportRaw
private static extern int ExportRawFloat (Matrix thism, byte[] bytes, int bytesLength, Coord texOffset, Co

#else

public void ExportRawFloat (byte[] bytes, Coord texOffset, Coord texSize, float mult=1)
{
 int numPixels = texSize.x*texSize.z;
 if (numPixels*4 != bytes.Length)
  throw new Exception("Array count does not match texture dimensions");

 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(texOffset, texSize));
 Coord min = intersection.Min; Coord max = intersection.Max;

 FloatToBytes converter = new FloatToBytes();

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
```

```
    int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
    int bytesPos = (z-texOffset.z)*texSize.x + x - texOffset.x;
    bytesPos *= 4;

    if (bytesPos>=bytes.Length || bytesPos<0) Debug.Log("Test");

    converter.f = arr[matrixPos] * mult;
    bytes[bytesPos] = converter.b0;
    bytes[bytesPos+1] = converter.b1;
    bytes[bytesPos+2] = converter.b2;
    bytes[bytesPos+3] = converter.b3;
   }
 }
#endif

[StructLayout(LayoutKind.Explicit)]
public class FloatToBytes
{
 [FieldOffset(0)] public float f;
 [FieldOffset(0)] public byte b0;
 [FieldOffset(1)] public byte b1;
 [FieldOffset(2)] public byte b2;
 [FieldOffset(3)] public byte b3;
}
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

 public void ExportHeights (float[,] heights)
 {
  Coord heightsSize = new Coord(heights.GetLength(1), heights.GetLength(0));
  ExportHeights (this, heights, rect.offset, heightsSize);
 }

 public void ExportHeights (float[,] heights, Coord heightsOffset)
 {
  Coord heightsSize = new Coord(heights.GetLength(1), heights.GetLength(0));
  ExportHeights (this, heights, heightsOffset, heightsSize);
 }

 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Matrix_ExportHeig
 private static extern void ExportHeights (Matrix thism, float[,] heights, Coord heightsOffset, Coord height

#else

 public void ExportHeights (float[,] heights) => ExportHeights(heights, rect.offset);
 public void ExportHeights (float[,] heights, Coord heightsOffset)
 {
  Coord heightsSize = new Coord(heights.GetLength(1), heights.GetLength(0));  //x and z swapped
  CoordRect heightsRect = new CoordRect(heightsOffset, heightsSize);

  CoordRect intersection = CoordRect.Intersected(rect, heightsRect);
```

```
  Coord min = intersection.Min; Coord max = intersection.Max;


  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
    int heightsPosX = x - heightsRect.offset.x;
    int heightsPosZ = z - heightsRect.offset.z;


    float val = arr[matrixPos];
    heights[heightsPosZ, heightsPosX] = val;
   }
 }
#endif


public void ExportHeightStrips (float[][,] heights) { ExportHeightStrips(heights, rect.offset); }
public void ExportHeightStrips (float[][,] heights, Coord heightsOffset)
{
 //TODO: offset doesnt work
 int offset = 0;
 for (int s=0; s<heights.Length; s++)
 {
  ExportHeights(heights[s], new Coord(0,offset));
  offset+=heights[s].GetLength(0);
 }
}
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

 public void ExportSplats (float[,,] splats, int channel)
 {
  Coord splatsSize = new Coord(splats.GetLength(1), splats.GetLength(0));
  ExportSplats (this, splats, rect.offset, splatsSize, splats.GetLength(2), channel);
 }

 public void ExportSplats (float[,,] splats, Coord heightsOffset, int channel)
 {
  Coord splatsSize = new Coord(splats.GetLength(1), splats.GetLength(0));
  ExportSplats (this, splats, heightsOffset, splatsSize, splats.GetLength(2), channel);
 }

 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Matrix_ExportSpla
 private static extern void ExportSplats (Matrix thism, float[,,] splats, Coord splatsOffset, Coord splatsSize

#else

 public void ExportSplats (float[,,] splats, int channel) { ExportSplats(splats, rect.offset, channel); }
 public void ExportSplats (float[,,] splats, Coord splatsOffset, int channel)
 {
  Coord splatsSize = new Coord(splats.GetLength(1), splats.GetLength(0));  //x and z swapped
  CoordRect splatsRect = new CoordRect(splatsOffset, splatsSize);
```

```
  CoordRect intersection = CoordRect.Intersected(rect, splatsRect);
  Coord min = intersection.Min; Coord max = intersection.Max;

  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
    int heightsPosX = x - splatsRect.offset.x;
    int heightsPosZ = z - splatsRect.offset.z;

    float val = arr[matrixPos];
    splats[heightsPosZ, heightsPosX, channel] = val;
   }
  }
 #endif


/* #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

  public void ExportDetail (int[,] detail, int channel, Noise random, float density=1)
  {
   Coord detailSize = new Coord(detail.GetLength(1), detail.GetLength(0));
   ExportDetail (this, detail, rect.offset, detailSize, channel, random, density);
  }
```

```
 public void ExportDetail (int[,] detail, Coord detailOffset, int channel, Noise random, float density=1)
 {
  Coord detailSize = new Coord(detail.GetLength(1), detail.GetLength(0));
  ExportDetail (this, detail, detailOffset, detailSize, channel, random, density);
 }

 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Matrix_ExportDeta
 private static extern void ExportDetail (Matrix thism, int[,] detail, Coord detailOffset, Coord detailSize, int

#else */

 public void ExportDetail (int[,] detail, int channel, Noise random, float density=1) => ExportDetail(detail, r
 public void ExportDetail (int[,] detail, Coord detailOffset, int channel, Noise random, float density=1) //aka
 {
  Coord splatsSize = new Coord(detail.GetLength(1), detail.GetLength(0));  //x and z swapped
  CoordRect splatsRect = new CoordRect(detailOffset, splatsSize);

  CoordRect intersection = CoordRect.Intersected(rect, splatsRect);
  Coord min = intersection.Min; Coord max = intersection.Max;

  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
    int detailPosX = x - splatsRect.offset.x;
    int detailPosZ = z - splatsRect.offset.z;
```

```
float val = arr[matrixPos];

//interpolating value since detail resolution is 512, while height is 513
//val += matrix.arr[pos+1] + matrix.arr[pos+fullSize] + matrix.arr[pos+fullSize+1]; //margins should prev
//val /= 4;

//or using minimal interpolation (creates better visual effect - grass isn't growing where it should not)
float val1 = x<max.x-1 ? arr[matrixPos+1] : val;
float val2 = z<max.z-1 ? arr[matrixPos+rect.size.x] : val;
float val3 = x<max.x-1 && z<max.z-1 ? arr[matrixPos+rect.size.x+1] : val;
//if (val1<val) val=val1;
//if (val2<val) val=val2;
//if (val3<val) val=val3;

//multiply with biome
//if (biomeMask != null) //no empty biomes in list (so no mask == root biome)
// val *= biomeMask.arr[pos]; //if mask is not assigned biome was ignored, so only main outs with mask

// if (val < 0) val = 0; if (val > 1) val = 1;

//the number of bushes in pixel
val *= density;//*pixelSize;

//random
float rnd = random.Random(channel, x,z);
```

```
    //converting to integer with random
    int intVal = (int)val;
    float remain = val - intVal;
    if (remain>rnd) intVal++;

    intVal = val>0.001f ? 1 : 0;

    detail[detailPosZ, detailPosX] = (int)(val+0.5f);
   }
 }

//#endif

//partial terrain data (loading only the part intersecting with matrix). Do not work in thread!
public void ExportTerrainData (TerrainData data) { ExportTerrainData (data, rect.offset, -1); }
public void ExportTerrainData (TerrainData data, int channel) { ExportTerrainData (data, rect.offset, chan
public void ExportTerrainData (TerrainData data, Coord dataOffset, int channel) //if channel is -1 getting l
{
 int resolution = channel==-1 ?  data.heightmapResolution : data.alphamapResolution;

 Coord dataSize = new Coord(resolution, resolution);
 CoordRect dataIntersection = CoordRect.Intersected(rect, new CoordRect(dataOffset, dataSize));
 if (dataIntersection.size.x==0 || dataIntersection.size.z==0) return;

 if (channel == -1)
```

```
 {
  float[,] heights = new float[dataIntersection.size.z, dataIntersection.size.x];  //x and z swapped
  ExportHeights(heights, dataIntersection.offset);
  data.SetHeights(dataIntersection.offset.x-dataOffset.x, dataIntersection.offset.z-dataOffset.z, heights);
 }

 else
 {
  float[,,] splats = data.GetAlphamaps(dataIntersection.offset.x-dataOffset.x, dataIntersection.offset.z-data
  ExportSplats(splats, dataIntersection.offset, channel);
  data.SetAlphamaps(dataIntersection.offset.x-dataOffset.x, dataIntersection.offset.z-dataOffset.z, splats)
 }
}


#endregion


#region Per-pixel Arithmetic

 //MMNATIVE directive is more convenient to use per-call, not in a separate area
 //it can help to find what function does faster
 //easier to comment directive out for debug
 //and help avoid creating native-nonnative garbage
 //directive areas were used initially (like MatrixNativeExtensions.cs), but I've changed my mind to this sys
 //ALTERNATIVE: use native call inside fn body (+script with native calls). See Fill as examples
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixFillVal")]
 private static extern void Fill (Matrix thisMatrix, float val);
#endif

public void Fill (float val)
{
 #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
  Fill(this, val);
 #else
  for (int i=0; i<count; i++)
   arr[i] = val;
 #endif
}

public void Fill (Matrix m)
 /// copies matrix of the same size. For different matrices use Copy
 #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
  => Fill(this, m);

  [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixFill")]
  private static extern void Fill (Matrix thisMatrix, Matrix m);
 #else
  {
```

```
   for (int i=0; i<count; i++)
    arr[i] = m.arr[i];
  }
 #endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Fill (float val, float opacity) => Fill(this, val, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixFillOpacity")]
 private static extern void Fill (Matrix thisMatrix, float val, float opacity);
#else
 public void Fill (float val, float opacity)
 {
  for (int i=0; i<count; i++)
   arr[i] = arr[i]*(1-opacity) + val*opacity;
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Mix (Matrix m, float opacity=1) => MixEx(this, m, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixMix")]
 private static extern void MixEx (Matrix thisMatrix, Matrix m, float opacity=1);
#else
 public void Mix (Matrix m, float opacity=1)
 {
```

```
  float invOpacity = 1-opacity;
  for (int i=0; i<count; i++)
   arr[i] = m.arr[i]*opacity + arr[i]*invOpacity;
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Mix (Matrix m, Matrix mask) => MixEx(this, m, mask);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixMixMask")]
 private static extern void MixEx (Matrix thisMatrix, Matrix m, Matrix mask);
#else
 public void Mix (Matrix m, Matrix mask)
 {
  for (int i=0; i<count; i++)
   arr[i] = arr[i]*(1-mask.arr[i]) + m.arr[i]*mask.arr[i];
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Mix (Matrix m, Matrix mask, float opacity) => MixEx(this, m, mask, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixMixMaskOp
 private static extern void MixEx (Matrix thisMatrix, Matrix m, Matrix mask, float opacity);
#else
 public void Mix (Matrix m, Matrix mask, float opacity)
```

```
 {
  float invOpacity = 1-opacity;
  for (int i=0; i<count; i++)
   arr[i] = arr[i]*invOpacity*(1-mask.arr[i]) + m.arr[i]*opacity*mask.arr[i];
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Mix (Matrix m, Matrix mask, float maskMin, float maskMax, bool maskInvert, bool fallof, float
  Mix (this, m, mask, maskMin, maskMax, maskInvert, fallof, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixMixComplex
 private static extern void Mix (Matrix thisMatrix, Matrix m, Matrix mask, float maskMin, float maskMax, bo
#else
 public void Mix (Matrix m, Matrix mask, float maskMin, float maskMax, bool maskInvert, bool fallof, float
 //used, check native
 {
  for (int i=0; i<count; i++)
  {
   float percent = mask.arr[i];

   //percent = (percent-maskMin)/(maskMax-maskMin);
   //was a comment "maskMax-maskMin could be 0". Instead using (can test no change, but for some pur

   if (maskMax-maskMin!=0) percent = (percent-maskMin)/(maskMax-maskMin);
   else percent = 0;
```

```
   if (percent<0) percent = 0; if (percent>1) percent = 1;
   //if (fallof) percent = 3*percent*percent - 2*percent*percent*percent;
   percent *= opacity;
   if (maskInvert) percent = 1-percent;

   arr[i] = arr[i]*(1-percent) + m.arr[i]*percent;
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void InvMix (Matrix m, Matrix invMask, float opacity=1) => InvMix(this, m, invMask, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixInvMix")]
 private static extern void InvMix (Matrix thisMatrix, Matrix m, Matrix invMask, float opacity=1);
#else
 public void InvMix (Matrix m, Matrix invMask, float opacity=1)
 //using inverted mask: mask1 will leave original value, mask0 will use m
 {
  float invOpacity = 1-opacity;
  for (int i=0; i<count; i++)
   arr[i] = arr[i]*invOpacity*invMask.arr[i] + m.arr[i]*opacity*(1-invMask.arr[i]);
 }
#endif
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Add (Matrix add, float opacity=1) => AddEx(this, add, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixAdd")]
 private static extern void AddEx (Matrix thisMatrix, Matrix add, float opacity=1);
#else
 public void Add (Matrix add, float opacity=1)
 {
  for (int i=0; i<count; i++)
   arr[i] += add.arr[i] * opacity;
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Add (Matrix add, Matrix mask, float opcaity=1) => AddEx(this, add, mask, opcaity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixAddMask")]
 private static extern void AddEx (Matrix thisMatrix, Matrix add, Matrix mask, float opcaity=1);
#else
 public void Add (Matrix add, Matrix mask, float opcaity=1)
 {
  for (int i=0; i<count; i++)
   arr[i] += add.arr[i] * mask.arr[i] * opcaity;
 }
#endif
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Add (float add) => AddEx(this,add);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixAddVal")]
 private static extern void AddEx (Matrix thisMatrix, float add);
#else
 public void Add (float add)
 {
  for (int i=0; i<count; i++)
   arr[i] += add;
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Blend (Matrix matrix, Matrix mask, Matrix add, Matrix addMask) => Blend(this,matrix, mask, a
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixBlend")]
 private static extern void Blend (Matrix thisMatrix, Matrix matrix, Matrix mask, Matrix add, Matrix addMas
#else
 public void Blend (Matrix matrix, Matrix mask, Matrix add, Matrix addMask)
 {
  for (int i=0; i<count; i++)
  {
   float sum = mask.arr[i] + addMask.arr[i];
    arr[i] = sum != 0 ?
    (matrix.arr[i]*mask.arr[i] + add.arr[i]*addMask.arr[i]) / sum :
```

```
    (matrix.arr[i] + add.arr[i])/2;
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Max (Matrix matrix, Matrix mask, Matrix add, Matrix addMask) => Max(this, matrix, mask, add
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixMaxComple
 private static extern void Max (Matrix thisMatrix, Matrix matrix, Matrix mask, Matrix add, Matrix addMask
#else
 public void Max (Matrix matrix, Matrix mask, Matrix add, Matrix addMask)
 {
  for (int i=0; i<count; i++)
    arr[i] = mask.arr[i] > addMask.arr[i]  ?  matrix.arr[i]  :  add.arr[i];
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Step (float mid=0.5f) => Step(this, mid);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixStep")]
 private static extern void Step (Matrix thisMatrix, float mid=0.5f);
#else
 public void Step (float mid=0.5f)
 {
```

```
  for (int i=0; i<count; i++)
    arr[i] = arr[i]>mid ? 1 : 0;
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Subtract (Matrix m, float opacity=1) => Subtract(this, m, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixSubtract")]
 private static extern void Subtract (Matrix thisMatrix, Matrix m, float opacity=1);
#else
 public void Subtract (Matrix m, float opacity=1)
 {
  for (int i=0; i<count; i++)
    arr[i] -= m.arr[i] * opacity;
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void InvSubtract (Matrix m, float opacity=1) => InvSubtract(this, m, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixInvSubtract"
 private static extern void InvSubtract (Matrix thisMatrix, Matrix m, float opacity=1);
 /// subtracting this matrix from m
#else
 public void InvSubtract (Matrix m, float opacity=1)
```

```
 /// subtracting this matrix from m
 {
  for (int i=0; i<count; i++)
   arr[i] = m.arr[i]*opacity - arr[i];
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Multiply (Matrix m, float opacity=1) => MultiplyEx(this, m, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixMultiply")]
 private static extern void MultiplyEx (Matrix thisMatrix, Matrix m, float opacity=1);
#else
 public void Multiply (Matrix m, float opacity=1)
 {
  float invOpacity = 1-opacity;
  for (int i=0; i<count; i++)
   arr[i] *= m.arr[i]*opacity + invOpacity;
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Multiply (float m) => MultiplyEx(this,m);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixMultiplyVal"
 private static extern void MultiplyEx (Matrix thisMatrix, float m);
```

```
#else
 public void Multiply (float m)
 {
  for (int i=0; i<count; i++)
   arr[i] *= m;
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void MultiplyInv (Matrix invFactor) => MultiplyInvEx(this, invFactor);
 /// Multiplies with inversed (1-val) value of other matrix
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixMultiplyInv")
 private static extern void MultiplyInvEx (Matrix thisMatrix, Matrix invFactor);
#else
 public void MultiplyInv (Matrix invFactor)
 /// Multiplies with inversed (1-val) value of other matrix
 {
  for (int i=0; i<count; i++)
   arr[i] *= 1-invFactor.arr[i];
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Contrast (float m) => Contrast(this, m);
```

```
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixContrast")]
 private static extern void Contrast (Matrix thisMatrix, float m);
 /// Leaving 0.5 values untouched, and increasing/shrinking 1-0 range
#else
 public void Contrast (float m)
 /// Leaving 0.5 values untouched, and increasing/shrinking 1-0 range
 {
  for (int i=0; i<count; i++)
  {
   float val = arr[i] - 0.5f;
   val *= m;
   arr[i] = val + 0.5f;
  }
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Divide (Matrix m, float opacity=1) => Divide(this, m, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixDivide")]
 private static extern void Divide (Matrix thisMatrix, Matrix m, float opacity=1);
#else
 public void Divide (Matrix m, float opacity=1)
 {
  float invOpacity = 1-opacity;
  for (int i=0; i<count; i++)
```

```
   arr[i] *= opacity/m.arr[i] + invOpacity;
  }
 #endif


 #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
  public void Difference (Matrix m, float opacity=1) => Difference(this, m, opacity);
  [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixDifference")
  private static extern void Difference (Matrix thisMatrix, Matrix m, float opacity=1);
 #else
  public void Difference (Matrix m, float opacity=1)
  {
   for (int i=0; i<count; i++)
   {
    float val = arr[i] - m.arr[i]*opacity;
    if (val < 0) val = -val;
    arr[i] = val;
   }
  }
 #endif


 #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
  public void Overlay (Matrix m, float opacity=1) => Overlay(this, m, opacity);
  [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixOverlay")]
  private static extern void Overlay (Matrix thisMatrix, Matrix m, float opacity=1);
```

```
#else
 public void Overlay (Matrix m, float opacity=1)
 {
  for (int i=0; i<count; i++)
  {
   float a = arr[i];
   float b = m.arr[i];

   b = b*opacity + (0.5f - opacity/2); //enhancing contrast via levels

   if (a > 0.5f) b = 1 - 2*(1-a)*(1-b);
   else b = 2*a*b;

   arr[i] = b;// b*opacity + a*(1-opacity); //the same
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void HardLight (Matrix m, float opacity=1) => HardLight(this, m, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixHardLight")]
 private static extern void HardLight (Matrix thisMatrix, Matrix m, float opacity=1);
 /// Same as overlay but estimating b>0.5
#else
 public void HardLight (Matrix m, float opacity=1)
```

```
 /// Same as overlay but estimating b>0.5
 {
  for (int i=0; i<count; i++)
  {
   float a = arr[i];
   float b = m.arr[i];

   if (b > 0.5f) b = 1 - 2*(1-a)*(1-b);
   else b = 2*a*b;

   arr[i] = b*opacity + a*(1-opacity);
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void SoftLight (Matrix m, float opacity=1) => SoftLight(this, m, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixSoftLight")]
 private static extern void SoftLight (Matrix thisMatrix, Matrix m, float opacity=1);
#else
 public void SoftLight (Matrix m, float opacity=1)
 {
  for (int i=0; i<count; i++)
  {
   float a = arr[i];
```

```
   float b = m.arr[i];
   b = (1-2*b)*a*a + 2*b*a;
   arr[i] = b*opacity + a*(1-opacity);
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Max (Matrix m, float opacity=1) => Max(this, m, opacity);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixMax")]
 private static extern void Max (Matrix thisMatrix, Matrix m, float opacity=1);
#else
 public void Max (Matrix m, float opacity=1)
 {
  for (int i=0; i<count; i++)
  {
   float val = m.arr[i]>arr[i] ? m.arr[i] : arr[i];
   arr[i] = val*opacity + arr[i]*(1-opacity);
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Min (Matrix m, float opacity=1) => Min(this, m, opacity);
```

```
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixMin")]
 private static extern void Min (Matrix thisMatrix, Matrix m, float opacity=1);
#else
 public void Min (Matrix m, float opacity=1)
 {
  for (int i=0; i<count; i++)
  {
   float val = m.arr[i]<arr[i] ? m.arr[i] : arr[i];
   arr[i] = val*opacity + arr[i]*(1-opacity);
  }
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Select (float level) => Select(this, level);
 /// Fills values below level with 0, above level with 1
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixSelect")]
 private static extern void Select (Matrix thisMatrix, float level);
#else
 public void Select (float level)
 {
  for (int i=0; i<count; i++)
   arr[i] = arr[i]>level ? 1 : 0;
 }
#endif
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Invert() => Invert(this);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixInvert")]
 private static extern void Invert(Matrix thisMatrix);
#else
 public void Invert()
 {
  for (int i=0; i<count; i++)
   arr[i] = -arr[i];
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void InvertOne() => InvertOne(this);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixInvertOne")]
 private static extern void InvertOne(Matrix thisMatrix);
#else
 public void InvertOne()
 {
  for (int i=0; i<count; i++)
   arr[i] = 1-arr[i];
 }
#endif
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void SelectRange (float minFrom, float minTo, float maxFrom, float maxTo) =>
  SelectRange(this, minFrom, minTo, maxFrom, maxTo);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixSelectRang
 private static extern void SelectRange (Matrix thisMatrix, float minFrom, float minTo, float maxFrom, floa
 /// Fill all values within min1-max0 with 1, while min0-1 and max0-1 are filled with blended
#else
 public void SelectRange (float minFrom, float minTo, float maxFrom, float maxTo)
 /// Fill all values within min1-max0 with 1, while min0-1 and max0-1 are filled with blended
 {
  for (int i=0; i<count; i++)
  {
   float src = arr[i];
   float dst;

   if (src<minFrom || src>maxTo) dst = 0;
   else if (src>minTo && src<maxFrom) dst = 1;
   else
   {
    float minVal = (src-minFrom)/(minTo-minFrom);
    float maxVal = 1-(src-maxFrom)/(maxTo-maxFrom);
    dst = minVal>maxVal? maxVal : minVal;
    if (dst<0) dst=0; if (dst>1) dst=1;
   }
```

```
   arr[i] = dst;
  }
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void ChangeRange (float fromMin, float fromMax, float toMin, float toMax) =>
  ChangeRange(this, fromMin, fromMax, toMin, toMax);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixChangeRan
 private static extern void ChangeRange (Matrix thisMatrix, float fromMin, float fromMax, float toMin, float
 /// Used to convert matrix from -1 1 range to 0 1 or vice versa
#else
 public void ChangeRange (float fromMin, float fromMax, float toMin, float toMax)
 /// Used to convert matrix from -1 1 range to 0 1 or vice versa
 {
  float fromRange = fromMax - fromMin;
  float toRange = toMax - toMin;

  for (int i=0; i<count; i++)
  {
   float val = (arr[i]-fromMin) / fromRange;  //converting to 0 1
   arr[i] = val*toRange + toMin;  //converting to to
  }
 }
```

```
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Clamp01 () => Clamp01(this);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixClamp01")]
 private static extern void Clamp01 (Matrix thisMatrix);
#else
 public void Clamp01 ()
 {
  for (int i=0; i<count; i++)
  {
   float val = arr[i];
   if (val > 1) arr[i] = 1;
   else if (val < 0) arr[i] = 0;
  }
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public float MaxValue () => MaxValue(this);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixMaxValue")]
 private static extern float MaxValue (Matrix thisMatrix);
#else
 public float MaxValue ()
```

```
 {
  float max=float.MinValue;
  for (int i=0; i<count; i++)
  {
   float val = arr[i];
   if (val > max) max = val;
  }
  return max;
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public float MinValue () => MinValue(this);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixMinValue")]
 private static extern float MinValue (Matrix thisMatrix);
#else
 public float MinValue ()
 {
  float min=float.MaxValue;
  for (int i=0; i<count; i++)
  {
   float val = arr[i];
   if (val < min) min = val;
  }
  return min;
```

```
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public float Average () => Average(this);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="Average")]
 private static extern float Average (Matrix thisMatrix);
#else
 public float Average ()
 {
  float avg = 0;
  for (int i=0; i<count; i++)
   avg += arr[i] / count;
  return avg;
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public virtual bool IsEmpty () => IsEmpty(this);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixIsEmpty")]
 private static extern bool IsEmpty (Matrix thisMatrix);
 /// Better than MinValue since it can quit if matrix is not empty
#else
 public virtual bool IsEmpty ()
```

```
 /// Better than MinValue since it can quit if matrix is not empty
 {
  for (int i=0; i<count; i++)
   if (arr[i] > 0.0001f) return false;
  return true;
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public virtual bool IsEmpty (float delta) => IsEmpty(this, delta);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixIsEmptyDel
 private static extern bool IsEmpty (Matrix thisMatrix, float delta);
#else
 public virtual bool IsEmpty (float delta)
 {
  for (int i=0; i<count; i++)
   if (arr[i] > delta) return false;
  return true;
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void BlackWhite (float mid) => BlackWhite(this, mid);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixBlackWhite"
```

```
 private static extern void BlackWhite (Matrix thisMatrix, float mid);
 /// Sets all values bigger than mid to white (1), and those lower to black (0)
#else
 public void BlackWhite (float mid)
 /// Sets all values bigger than mid to white (1), and those lower to black (0)
 {
  for (int i=0; i<count; i++)
  {
   float val = arr[i];
   if (val > mid) arr[i] = 1;
   else arr[i] = 0;
  }
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void BrighnesContrast (float brightness, float contrast) => BrighnesContrast(this, brightness, contr
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixBrightnessC
 private static extern void BrighnesContrast (Matrix thisMatrix, float brightness, float contrast);
#else
 public void BrighnesContrast (float brightness, float contrast)
 {
  for (int i=0; i<count; i++)
  {
   float val = arr[i];
```

```
   val = ((val-0.5f)*contrast) + 0.5f;  //contrast
   val += brightness/2 * (contrast<1 ? 1 : contrast); //brightness  //this way brightness works in range -1 to
   //val += brightness/2 * (1+contrast);  //alt brightness

   if (val<0) val = 0; if (val>1) val=1;
   arr[i] = val;
  }
 }
#endif

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Terrace (float[] terraces, float steepness) =>
  Terrace(this, terraces, terraces.Length, steepness); //+calls extern with array length
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixTerrace")]
 private static extern void Terrace (Matrix thisMatrix, float[] terraces, int terraceCount, float steepness);
#else
 public void Terrace (float[] terraces, float steepness)
 {
  float intensity = Mathf.Sqrt(steepness);

  for (int i=0; i<count; i++)
  {
   float val = arr[i];
   if (val > 0.9999f) continue; //do nothing with values that are out of range
```

```
  int terrNum = 0;
  for (int t=0; t<terraces.Length-1; t++)
  {
   if (terraces[terrNum+1] > val || terrNum+1 == terraces.Length) break;
   terrNum++;
  }

  //kinda curve evaluation
  float delta = terraces[terrNum+1] - terraces[terrNum];
  float relativePos = (val - terraces[terrNum]) / delta;

  float percent = 3*relativePos*relativePos - 2*relativePos*relativePos*relativePos;

  percent = (percent-0.5f)*2;
  bool minus = percent<0; percent = Mathf.Abs(percent);

  percent = Mathf.Pow(percent,1f-steepness);

  if (minus) percent = -percent;
  percent = percent/2 + 0.5f;

  float dstVal = terraces[terrNum]*(1-percent) + terraces[terrNum+1]*percent;
  arr[i] = dstVal*intensity + val*(1-intensity);
 }
}
```

```
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Levels (float inMin, float inMax, float gamma, float outMin, float outMax) =>
  Levels(this, inMin, inMax, gamma, outMin, outMax);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixLevels")]
 private static extern void Levels (Matrix thisMatrix, float inMin, float inMax, float gamma, float outMin, floa
#else
 public void Levels (float inMin, float inMax, float gamma, float outMin, float outMax)
 {
  float inDelta = inMax - inMin;
  float outDelta = outMax - outMin;

  for (int i=0; i<count; i++)
  {
   float val = arr[i];

   //preliminary clamping
   if (val < inMin) { arr[i] = outMin; continue; }
   if (val > inMax) { arr[i] = outMax; continue; }

   //input
   if (inDelta != 0)
    val = (val-inMin) / inDelta;
   else
```

```
    val = inMin;

   //gamma
   if (gamma>1.00001f || gamma<0.9999f)  // gamma != 1
   {
    if (gamma<1) val = Mathf.Pow(val, gamma);
    else val = Mathf.Pow(val, 1/(2-gamma));
   }

   //output
   if (outDelta != 0)
    val = outMin + val * outDelta;
   else
    val = outMin;

   arr[i] = val;
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void UniformCurve (float[] lut) =>
  UniformCurve(this, lut, lut.Length);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixUniformCur
 private static extern void UniformCurve (Matrix thisMatrix, float[] lut, int lutCount);
```

```
#else
 public void UniformCurve (float[] lut)
 /// Applies curve that got curve lut with uniformly placed times
 /// A copy of curve's EvaluateLuted
 {
  float step = 1f / (lut.Length-1);

  for (int i=0; i<count; i++)
  {
   float val = arr[i];

   int prevNum = (int)(val/step);
   int nextNum = prevNum+1;

   if (prevNum<0) prevNum = 0; if (prevNum>=lut.Length) prevNum=lut.Length-1;
   if (nextNum<0) nextNum = 0; if (nextNum>=lut.Length) nextNum=lut.Length-1;

   float prevX = prevNum * step;
   float percent = (val-prevX) / step;

   float prevY = lut[prevNum];
   float nextY = lut[nextNum];

   arr[i] = prevY*(1-percent) + nextY*percent;
  }
 }
```

```
#endif

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Pow (float powVal) => Pow(this, powVal);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixPow")]
 private static extern void Pow (Matrix thisMatrix, float powVal);
#else
 public void Pow (float pow)
 {
  for (int i=0; i<count; i++)
   arr[i] = Mathf.Pow(arr[i], pow);
 }
#endif



//#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
// public void Parallax (Vector2D offset, Matrix src, Matrix maskX, Matrix maskZ, int interpolation=1 /*0-no
//  Parallax(this, offset.x, offset.z, src, maskX, maskZ, maskX!=null, maskZ!=null, interpolation);

// [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixParallax")]
// private static extern void Parallax (Matrix thisMatrix, float directionX, float directionZ, Matrix src, Matrix
//#else
 public void Parallax (Vector2D offset, Matrix src, Matrix maskX, Matrix maskZ, int interpolation=1 /*0-no
 ///For each pixel gives the value of coord + direction*intensityMask
 ///interpolateTransitions reads interpolated coordinates if mask value more 0 and less 1. interpolateFull r
 {
```

```
Coord min = rect.Min; Coord max = rect.Max;
for (int x=min.x; x<max.x; x++)
 for (int z=min.z; z<max.z; z++)
 {
  int pos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;

  float intensityX = maskX!=null ? maskX.arr[pos] : 1;
  float intensityZ = maskZ!=null ? maskZ.arr[pos] : 1; //or use intensityX?

  if (intensityX < 0.0001f  &&  intensityZ < 0.0001f)
   { arr[pos] = src.arr[pos]; continue; }

  float px = x - offset.x*intensityX; //negative intensity (ie direction) to READ pixels will create an effect o
  float pz = z - offset.z*intensityZ;

  if (px < min.x) px = min.x; if (px>max.x-1) px = max.x-1;
  if (pz < min.z) pz = min.z; if (pz>max.z-1) pz = max.z-1;

  float pVal;

  if (interpolation == 0)
   pVal = src.GetFloored(px+0.5f, pz+0.5f);

  else if (interpolation == 1)
   pVal = src.GetInterpolated(px, pz);
```

```
    else
    {
     if (intensityX > 0.999f  &&  intensityZ > 0.999f) //full intensity - no interpolation
      pVal = src.GetFloored(px+0.5f, pz+0.5f);
     else
      pVal = src.GetInterpolated(px, pz);
    }

    arr[pos] = pVal;
   }
  }
 }
//#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void Stroke (Vector2D pos, float radius, float hardness, bool smoothTransition=true, float bckgVal
  Stroke(this, pos.x, pos.z, radius, hardness, smoothTransition, bckgVal, strokeVal);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixStroke")]
 private static extern void Stroke (Matrix thisMatrix, float posX, float posZ, float radius, float hardness, boo
#else
 public void Stroke (Vector2D pos, float radius, float hardness, bool smoothTransition=true, float bckgVal
 /// Creates a circular dot on a matrix using radius and hardness
 /// All values are in pixels and using matrix offset
 {
  Coord min = rect.Min; Coord max = rect.Max;
  for (int x=min.x; x<max.x; x++)
```

```
   for (int z=min.z; z<max.z; z++)
   {
    int p = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;

    float dist = Mathf.Sqrt((x-pos.x)*(x-pos.x) + (z-pos.z)*(z-pos.z));
    if (dist > radius) { arr[p] = bckgVal; continue; }
    if (dist < radius*hardness) { arr[p] = strokeVal; continue; }

    float fallof = 1 - (dist-radius*hardness) / (radius*(1-hardness));
    if (fallof>1) fallof = 1; if (fallof<0) fallof = 0;
    if (smoothTransition) fallof = 3*fallof*fallof - 2*fallof*fallof*fallof;

    arr[p] = bckgVal*(1-fallof) + strokeVal*fallof;
   }
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Res
 private static extern void Fallof (Matrix srcBackground, Matrix srcBrush, Matrix dst, Vector2D pos, float r
 /// Blends two matrices in a smooth circular way using radius and hardness
 /// All values are in pixels and using matrix offset
 /// All matrix rects could be different - will process only their intersection
#else
```

```
public static void Fallof (Matrix srcBackground, Matrix srcBrush, Matrix dst, Vector2D pos, float radius, fl
/// Blends two matrices in a smooth circular way using radius and hardness
/// All values are in pixels and using matrix offset
/// All matrix rects could be different - will process only their intersection
{
 CoordRect intersection = CoordRect.Intersected(srcBackground.rect, srcBrush.rect);
 intersection = CoordRect.Intersected(dst.rect, intersection);
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int dstPos = (z-dst.rect.offset.z)*dst.rect.size.x + x - dst.rect.offset.x;
   int brsPos = (z-srcBrush.rect.offset.z)*srcBrush.rect.size.x + x - srcBrush.rect.offset.x;
   int bkgPos = (z-srcBackground.rect.offset.z)*srcBackground.rect.size.x + x - srcBackground.rect.offset

   float dist = Mathf.Sqrt((x-pos.x)*(x-pos.x) + (z-pos.z)*(z-pos.z));
   if (dist > radius) { dst.arr[dstPos] = srcBackground.arr[bkgPos]; continue; }
   if (dist < radius*hardness) { dst.arr[dstPos] = srcBrush.arr[brsPos]; continue; }

   float fallof = 1 - (dist-radius*hardness) / (radius*(1-hardness));
   if (fallof>1) fallof = 1; if (fallof<0) fallof = 0;
   if (smoothTransition) fallof = 3*fallof*fallof - 2*fallof*fallof*fallof;

   dst.arr[dstPos] = srcBackground.arr[bkgPos]*(1-fallof) + srcBrush.arr[brsPos]*fallof;
  }
```

```
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 public void BlendStamped (Matrix src, Matrix stamp, float centerX, float centerZ, float radius, float transit
  BlendStamped(this, src, stamp, centerX, centerZ, radius, transition, smoothFallof);
 [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixBlendStamp
 private static extern void BlendStamped (Matrix thisMatrix, Matrix src, Matrix stamp, float centerX, float c
#else
 public void BlendStamped (Matrix src, Matrix stamp, float centerX, float centerZ, float radius, float transit
 /// Blends two matrices in a smooth circular way using radius and hardness
 /// Fill all values outside radius+transition with SRC, inside radius with STAMP. In most cases src is this.
 /// All values are in pixels and using matrix offset
 /// Center does not need to be the real center, it's just used to calculate fallof
 /// Hardness is the percent (0-1) of the stamp that has 100% fallof
 /// Used in Locks (seems only)
 /// TODO: switch to Fallof
 {
  CoordRect intersection = CoordRect.Intersected(rect, stamp.rect);
  Coord min = intersection.Min; Coord max = intersection.Max;

  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    float dist = Mathf.Sqrt((x-centerX)*(x-centerX) + (z-centerZ)*(z-centerZ));
```

```
    int pos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;

    int stampPos = (z-stamp.rect.offset.z)*stamp.rect.size.x + x - stamp.rect.offset.x;
    //if (dist < radius) { arr[pos] = stamp.arr[stampPos]; continue; } //not radius, but radius/transition

    //int srcPos = (z-src.rect.offset.z)*src.rect.size.x + x - src.rect.offset.x; //not used
    //if (dist > radius+transition) { arr[pos] = src.arr[srcPos]; continue; }

    float fallof;
    if (transition == 0)
     fallof = dist>radius ? 0 : 1;
    else
    {
     fallof = 1 - (dist-radius) / transition;
     if (fallof>1) fallof = 1; if (fallof<0) fallof = 0;
     if (smoothFallof) fallof = 3*fallof*fallof - 2*fallof*fallof*fallof;
    }

    arr[pos] = src.arr[pos]*(1-fallof) + stamp.arr[stampPos]*fallof;
   }
  }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
```

```
 public static void ReadMatrix (Matrix src, Matrix dst, CoordRect.TileMode tileMode = CoordRect.TileMod
   ReadMatrix (src, dst, (int)tileMode);
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "ReadMatrixTile
 private static extern void ReadMatrix (Matrix src, Matrix dst, int tileMode);
 /// Fills dst with values that are at the same position in src
 /// if coordinate is out of src bounds - reading using specified tiling
#else
 static public void ReadMatrix (Matrix src, Matrix dst, CoordRect.TileMode tileMode = CoordRect.TileMod
 /// Fills dst with values that are at the same position in src
 /// if coordinate is out of src bounds - reading using specified tiling
 {
  Coord tmp = new Coord(0,0);
  Coord min = dst.rect.Min; Coord max = dst.rect.Max;
  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    tmp.x = x; tmp.z=z;
    src.rect.Tile(ref tmp, tileMode);
    dst[x,z] = src[tmp];
   }
 }
#endif



public static void CopyIntersected (Matrix src, Matrix dst) => ReadMatrix(src, dst); //just name dup to find
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
```

```
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "ReadMatrixFull
 public static extern void ReadMatrix (Matrix src, Matrix dst);
 /// Fills dst with values that are at the same position in src
 /// if coordinate is out of src bounds - just skipping it
#else
 static public void ReadMatrix (Matrix src, Matrix dst)
 /// Fills dst with values that are at the same position in src
 /// if coordinate is out of src bounds - just skipping it
 {
  CoordRect intersection = CoordRect.Intersected(src.rect, dst.rect);
  if (intersection.size.x<=0 || intersection.size.z<=0) return;
  Coord min = intersection.Min; Coord max = intersection.Max;

  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    int srcPos = (z-src.rect.offset.z)*src.rect.size.x + x - src.rect.offset.x;
    int dstPos = (z-dst.rect.offset.z)*dst.rect.size.x + x - dst.rect.offset.x;

    dst.arr[dstPos] = src.arr[srcPos];
   }
 }
#endif

public static void CopyRect (Matrix src, Matrix dst, CoordRect rect) => ReadMatrix(src, dst, rect);
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
```

```
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "ReadMatrixRec
 public static extern void ReadMatrix (Matrix src, Matrix dst, CoordRect rect);
 /// The same, but only within given rect
#else
 static public void ReadMatrix (Matrix src, Matrix dst, CoordRect rect)
 /// The same, but only within given rect
 {
  CoordRect intersection = CoordRect.Intersected(rect, dst.rect);
  intersection = CoordRect.Intersected(src.rect, intersection);
  Coord min = intersection.Min; Coord max = intersection.Max;

  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    int srcPos = (z-src.rect.offset.z)*src.rect.size.x + x - src.rect.offset.x;
    int dstPos = (z-dst.rect.offset.z)*dst.rect.size.x + x - dst.rect.offset.x;

    dst.arr[dstPos] = src.arr[srcPos];
   }
 }
#endif

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixCopyRes
 public static extern void CopyResized (Matrix src, Matrix dst,
  Vector2D srcRectPos, Vector2D srcRectSize,
```

```
 Coord dstRectPos, Coord dstRectSize);
/// Reads matrix if with NN interpolation - srcRect and dstRect are actually onerect - on src and dst
/// Can provide custom size ratio (src/dst) as well for better pixel positioning
#else
static public void CopyResized  (Matrix src, Matrix dst,
 Vector2D srcRectPos, Vector2D srcRectSize,
 Coord dstRectPos, Coord dstRectSize)
{
 CoordRect dstIntersection = CoordRect.Intersected(new CoordRect(dstRectPos,dstRectSize), dst.rect);
 Coord min = dstIntersection.Min; Coord max = dstIntersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   float pX = (float)(x-dstRectPos.x) / dstRectSize.x; //-1?
   float pZ = (float)(z-dstRectPos.z) / dstRectSize.z;

   int srcX = (int)(pX*srcRectSize.x + srcRectPos.x + 0.5f);
   int srcZ = (int)(pZ*srcRectSize.z + srcRectPos.z + 0.5f);
   if (srcX < src.rect.offset.x  ||  srcX >= src.rect.offset.x+src.rect.size.x  ||
    srcZ < src.rect.offset.z  ||  srcZ >= src.rect.offset.z+src.rect.size.z)
     continue;

   int srcPos = (srcZ-src.rect.offset.z)*src.rect.size.x + srcX - src.rect.offset.x;
   int dstPos = (z-dst.rect.offset.z)*dst.rect.size.x + x - dst.rect.offset.x;
```

```
    dst.arr[dstPos] = src.arr[srcPos];
   }
  }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixResize")]
 public static extern void Resize (Matrix src, Matrix dst);
 /// Uses NN interpolation to copy pixels from src to dst
#else
 public static void Resize (Matrix src, Matrix dst)
 /// Uses NN interpolation to copy pixels from src to dst
 {
  float dstToSrcX = (float)src.rect.size.x / (float)dst.rect.size.x; //src/dst - isn't it an error?
  float dstToSrcZ = (float)src.rect.size.z / (float)dst.rect.size.z;

  for (int x=0; x<dst.rect.size.x; x++)
   for (int z=0; z<dst.rect.size.z; z++)
   {
    int dstPos = z*dst.rect.size.x + x;

    int srcX = (int)((x+0.5f)*dstToSrcX);
    int srcZ = (int)((z+0.5f)*dstToSrcZ);
    int srcPos = srcZ*src.rect.size.x + srcX;
```

```
    dst.arr[dstPos] = src.arr[srcPos];
   }
 }
#endif



static public void BlendLayers (Matrix[] matrices, float[] opacity=null)
/// Changes splatmaps in photoshop layered style so their summary value does not exceed 1
{
 CoordRect? rect = matrices.Any()?.rect;
 if (rect == null) return;

 int rectCount = rect.Value.Count;
 for (int pos=0; pos<rectCount; pos++)
 {
  float left = 1;
  for (int i=matrices.Length-1; i>=0; i--) //layer 0 is background, layer Length-1 is the top one
  {
   if (matrices[i] == null) continue;

   float val = matrices[i].arr[pos];

   if (opacity != null) val *= opacity[i];

   val = val * left;
   matrices[i].arr[pos] = val;
```

```
   left -= val;


   if (left < 0) break;
  }
 }
}



static public void NormalizeLayers (Matrix[] matrices, bool allowBelowOne=false)
/// Just changes splatmaps so their summary value is always 1 (or never more than 1 if allowBelowOne d
{
 CoordRect? rect = matrices.Any()?.rect;
 if (rect == null) return;

 int rectCount = rect.Value.Count;
 for (int pos=0; pos<rectCount; pos++)
 {
  float sum = 0;
  for (int i=0; i<matrices.Length; i++) sum += matrices[i].arr[pos];
  if (sum > 1f || !allowBelowOne) for (int i=0; i<matrices.Length; i++) matrices[i].arr[pos] /= sum;
 }
}



static public void NormalizeLayers (Matrix[] matrices, float[] opacities, bool allowBelowOne=false)
/// Just changes splatmaps so their summary value is always 1 (or never more than 1 if allowBelowOne d
```

```
{
 CoordRect? rect = matrices.Any()?.rect;
 if (rect == null) return;

 int rectCount = rect.Value.Count;
 for (int pos=0; pos<rectCount; pos++)
 {
  float sum = 0;

  for (int i=0; i<matrices.Length; i++)
   sum += matrices[i].arr[pos] * opacities[i];

  if (sum > 1f || !allowBelowOne)
   for (int i=0; i<matrices.Length; i++)
    matrices[i].arr[pos] = (matrices[i].arr[pos]*opacities[i]) / sum;
 }
}



static public void NormalizeLayers (Matrix[] matrices, Matrix[] masks, bool allowBelowOne=false)
/// Just changes splatmaps so their summary value always 1
{
 CoordRect? rect = matrices.Any()?.rect;
 if (rect == null) return;

 int rectCount = rect.Value.Count;
```

```
 for (int pos=0; pos<rectCount; pos++)
 {
  for (int i=0; i<matrices.Length; i++)
   matrices[i].arr[pos] *= masks[i].arr[pos];

  float sum = 0;
  for (int i=0; i<matrices.Length; i++) sum += matrices[i].arr[pos];
  if (sum > 1f || !allowBelowOne)
   for (int i=0; i<matrices.Length; i++)
    matrices[i].arr[pos] /= sum;
 }
}


 public bool ContainsNaN ()
 {
  for (int i=0; i<arr.Length; i++)
   if (float.IsNaN(arr[i])) return true;
  return false;
 }


#endregion


#region Per-Rect Arithmetics
```

```
public void MultiplyRect (Matrix m)
{
 //if (rect == m.rect)
 // { Multiply(m); return; }

 CoordRect intersection = CoordRect.Intersected(rect, m.rect);
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int z=min.z; z<max.z; z++) //more sequential if z goes first
  for (int x=min.x; x<max.x; x++)
  {
   int p = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int mp = (z-m.rect.offset.z)*m.rect.size.x + x - m.rect.offset.x;

   arr[p] *= arr[mp];
  }
}


public void FillRect (float val, CoordRect fillRect)
/// Fills everything inside intersecting rect with val
{
 CoordRect intersection = CoordRect.Intersected(rect, fillRect);
 Coord min = intersection.Min; Coord max = intersection.Max;
```

```
 for (int z=min.z; z<max.z; z++)
  for (int x=min.x; x<max.x; x++)
  {
   int p = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   arr[p] = val;
  }
}


public void FillOuterRect (float val, CoordRect fillRect)
/// Opposite to FillRect - fills everything outside intersecting rect with val
{
 Coord fillMin = fillRect.Min; Coord fillMax = fillRect.Max;
 Coord min = rect.Min; Coord max = rect.Max;

 for (int z=min.z; z<max.z; z++)
  for (int x=min.x; x<max.x; x++)
  {
   if (x>=fillMin.x  &&  x<=fillMax.x  &&
    z>=fillMin.z  &&  z<=fillMax.z)
     continue;

   int p = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   arr[p] = val;
  }
}
```

```
public void Crop (CoordRect newRect)
/// Copies part of the matrix to new rect, leaving 0 at non-intersecting positions
{
 Coord newMin = newRect.Min; Coord newMax = newRect.Max;
 Coord min = rect.Min; Coord max = rect.Max;

 Matrix newMatrix = new Matrix(newRect);

 for (int z=min.z; z<max.z; z++)
  for (int x=min.x; x<max.x; x++)
  {
   if (x<newMin.x  ||  x>=newMax.x  ||
    z<newMin.z  ||  z>=newMax.z)
     continue;

   int p = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int newp = (z-newRect.offset.z)*newRect.size.x + x - newRect.offset.x;
   newMatrix.arr[newp] = arr[p];
  }

 arr = newMatrix.arr;
 rect = newMatrix.rect;
 count = newMatrix.count;
}
```

```
#endregion



#region Histogram


 public float[] Histogram (int resolution, float max=1, bool normalize=true)

 /// Evaluates all pixels in matrix and returns an array of pixels count by their value.

 {

  float[] quants = new float[resolution];


  for (int i=0; i<count; i++)

  {

   float val = arr[i];

   if (val<0 || val>max) continue; //out of histogram range


   float percent = val / max;

   int num = (int)(percent*resolution);

   if (num==resolution) num--; //this could happen when val==max


   quants[num]++;

  }


  if (normalize)

  {

   float maxQuant = 0;
```

```
  for (int i=0; i<resolution; i++)
    if (quants[i] > maxQuant) maxQuant = quants[i];

  for (int i=0; i<resolution; i++)
    quants[i] /= maxQuant;
 }

 return quants;
}

public byte[] HistogramBytes (int resolution, float max=1, bool normalize=true)
/// Just as Histogram, but returns bytes array ready to convert to texture
{
 float[] quants = Histogram(resolution, max, normalize);
 byte[] bytes = new byte[quants.Length];

 for (int i=0; i<quants.Length; i++)
   bytes[i] = (byte)(quants[i]*255);

 return bytes;
}

public static byte[] HistogramToTextureBytes (float[] quants, int height, byte empty=0, byte top=255, byte
```

```
/// Converts an array from Histogram to texture bytes
{
 int width = quants.Length;
 byte[] bytes = new byte[width * height];

 for (int x=0; x<width; x++)
 {
  int max = (int)(quants[x] * height);
  if (max==height) max--;

  for (int z=0; z<height; z++)
  {
   byte val = empty;
   if (z==max) val=top;
   else if (z<max) val=filled;

   bytes[z*width + x] = val;
  }
 }

 return bytes;
}

public static Texture2D HistogramToTextureR8 (float[] quants, int height, byte empty=0, byte top=255, by
/// Converts an array from Histogram to texture (format R8)
{
```

```
 byte[] bytes = HistogramToTextureBytes(quants, height, empty, top, filled);

 Texture2D tex = new Texture2D(quants.Length, height, TextureFormat.R8, false, linear:true);
 tex.LoadRawTextureData(bytes);
 tex.Apply(updateMipmaps:false);

 return tex;
}

public byte[] HistogramTextureBytes (int width, int height, byte empty=0, byte top=255, byte filled=128)
{
 float[] quants = Histogram(width);
 byte[] bytes = HistogramToTextureBytes(quants, height, empty, top, filled);
 return bytes;
}

#endregion


#region Editor Events

public static Action<Matrix,string> onPreview;  //manually called to announce MatrixWindow and other ed

public void ToWindow (string name, bool useCopy=false, bool mainThread=false)
{
 Matrix matrix = useCopy ? new Matrix(this) : this;
```

```
 if (!mainThread)
  onPreview?.Invoke(matrix,name);
 else
  Den.Tools.Tasks.CoroutineManager.Enqueue(() => onPreview?.Invoke(matrix,name) );
}

#endregion



#region Other

 #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
  public void Line (Vector2D start, Vector2D end, float valStart=1, float valEnd=1, bool antialised=false, bo
   Line(this, start.x, start.z, end.x, end.z, valStart, valEnd, antialised, paddedOnePixel, endInclusive);

  [DllImport ("NativePlugins", CallingConvention=CallingConvention.Cdecl, EntryPoint="MatrixLine")]
  private static extern void Line (Matrix thism, float startX, float startZ, float endX, float endZ, float valStart,
 #else
  public void Line (Vector2D start, Vector2D end, float valStart=1, float valEnd=1, bool antialised=false, bo
  /// Strokes the line from start (inclusive) to end (non-inclusive), gradientally filling it with valStart to valEnd
  /// PaddedOnePixel works similarly to AA, but fills border pixels with full value (to create main tex for the
  {
   start.x += 0.5f; start.z += 0.5f;  //way more convenient rounding in code below
   end.x += 0.5f; end.z += 0.5f;
```

```
int numSteps = Mathf.Max(
 Mathf.Abs( Mathf.FloorToInt(end.x) - Mathf.FloorToInt(start.x) ),  //NOT (int)(end-start)!
 Mathf.Abs( Mathf.FloorToInt(end.z) - Mathf.FloorToInt(start.z) )  );

float stepX = (end.x-start.x) / numSteps;
float stepZ = (end.z-start.z) / numSteps;

bool isVertical = Mathf.Abs(stepZ) > Mathf.Abs(stepX); //for antializasing

int ei = endInclusive ? 1 : 0;
for (int s=0; s<numSteps+ei; s++)
{
 float fx = start.x + stepX*s - rect.offset.x;  //neglecting matrix rect, should be from 0 to matrix.rect.size
 float fz = start.z + stepZ*s - rect.offset.z;

 int ix = (int)(float)fx;  //if (fx<0) ix--;
 int iz = (int)(float)fz;  //if (fz<0) iz--;

 if (ix<0 || ix>rect.size.x-1 ||
  iz<0 || iz>rect.size.z-1 )
   continue;

 int pos = iz*rect.size.x + ix;

 float valPercent = 1f*s/numSteps;
 float val = valStart*(1-valPercent) + valEnd*valPercent;
```

```
arr[pos] = val;


if (antialised)
{
 if (!isVertical)
 {
  if (iz<1 || iz>rect.size.z-2) continue;
  float p = fz-iz;
  arr[pos-rect.size.x] = arr[pos-rect.size.x]*p + val*(1-p); //blending commented out since it create sema
  arr[pos+rect.size.x] = arr[pos+rect.size.x]*(1-p) + val*p;
 }
 else
 {
  if (ix<1 || ix>rect.size.x-2) continue;
  float p = fx-ix;
  arr[pos-1] = arr[pos-1]*p + val*(1-p);
  arr[pos+1] = arr[pos+1]*(1-p) + val*p;
 }
}


if (paddedOnePixel)
{
 if (!isVertical)
 {
  if (iz<1 || iz>rect.size.z-2) continue;
```

```
        arr[pos-rect.size.x] = val;
        arr[pos+rect.size.x] = val;
       }
       else
       {
        if (ix<1 || ix>rect.size.x-2) continue;
        arr[pos-1] = val;
        arr[pos+1] = val;
       }
      }
     }
    }
   #endif

  #endregion
 }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;

namespace Den.Tools
{
 [Serializable, StructLayout (LayoutKind.Sequential)]
 public class Matrix2D<T>// : ICloneable
 {
  public CoordRect rect; //never assign it's size manually, use ChangeRect
  public int count;
  public int pos;
  public T[] arr;

  public const bool native = true;
   //rect.size.x*rect.size.z, not a property for faster access

  #region Creation

   public Matrix2D () {}

   public Matrix2D (int x, int z, T[] array=null)
   {
    rect = new CoordRect(0,0,x,z);
```

```
 count = x*z;
 if (array != null && array.Length<count) Debug.LogError("Array length: " + array.Length + " is lower then
 if (array != null && array.Length>=count) this.arr = array;
 else this.arr = new T[count];
}

public Matrix2D (CoordRect rect, T[] array=null)
{
 this.rect = rect;
 count = rect.size.x*rect.size.z;
 if (array != null && array.Length<count) Debug.Log("Array length: " + array.Length + " is lower then matri
 if (array != null && array.Length>=count) this.arr = array;
 else this.arr = new T[count];
}

public Matrix2D (Coord offset, Coord size, T[] array=null)
{
 rect = new CoordRect(offset, size);
 count = rect.size.x*rect.size.z;
 if (array != null && array.Length<count) Debug.Log("Array length: " + array.Length + " is lower then matri
 if (array != null && array.Length>=count) this.arr = array;
 else this.arr = new T[count];
}

public Matrix2D (Matrix2D<T> src)
{
```

```
  rect = src.rect;
  count = src.count;
  arr = new T[count];
  for (int i=0; i<arr.Length; i++)
   arr[i] = src.arr[i];
 }

#endregion

public T this[int x, int z]
{
 get { return arr[(z-rect.offset.z)*rect.size.x + x - rect.offset.x]; } //rect fn duplicated to increase performanc
 set { arr[(z-rect.offset.z)*rect.size.x + x - rect.offset.x] = value; }
}

public T this[float x, float z]
///Floors coordinates and gets value. To get interpolated value use GetInterpolated
{
 get{
  int ix = (int)(x); if (x<1) ix--;
  int iz = (int)(z); if (z<1) iz--;
  return arr[(iz-rect.offset.z)*rect.size.x + ix - rect.offset.x];
 }
 set{
  int ix = (int)(x); if (x<1) ix--;
  int iz = (int)(z); if (z<1) iz--;
```

```
  arr[(iz-rect.offset.z)*rect.size.x + ix - rect.offset.x] = value;
 }
}

public T this[Coord c]
{
 get { return arr[(c.z-rect.offset.z)*rect.size.x + c.x - rect.offset.x]; }
 set { arr[(c.z-rect.offset.z)*rect.size.x + c.x - rect.offset.x] = value; }
}

public T CheckGet (int x, int z)
{
 if (x>=rect.offset.x && x<rect.offset.x+rect.size.x && z>=rect.offset.z && z<rect.offset.z+rect.size.z)
  return arr[(z-rect.offset.z)*rect.size.x + x - rect.offset.x];
 else return default(T);
}

/*public T this[Vector3 pos]
{
 get { return array[((int)pos.z-rect.offset.z)*rect.size.x + (int)pos.x - rect.offset.x]; }
 set { array[((int)pos.z-rect.offset.z)*rect.size.x + (int)pos.x - rect.offset.x] = value; }
}*/

public T this[Vector2 pos]
{
 get{
```

```
  int posX = (int)(pos.x + 0.5f); if (pos.x < 0) posX--;
  int posZ = (int)(pos.y + 0.5f); if (pos.y < 0) posZ--;
  return arr[(posZ-rect.offset.z)*rect.size.x + posX - rect.offset.x];
 }
 set{
  int posX = (int)(pos.x + 0.5f); if (pos.x < 0) posX--;
  int posZ = (int)(pos.y + 0.5f); if (pos.y < 0) posZ--;
  arr[(posZ-rect.offset.z)*rect.size.x + posX - rect.offset.x] = value;
 }
}

public int Pos (Coord coord) { return (coord.z-rect.offset.z)*rect.size.x + coord.x - rect.offset.x; }
public int Pos (int posX, int posZ) { return (posZ-rect.offset.z)*rect.size.x + posX - rect.offset.x; }

public Coord CoordByNum (int num)
{
 int z = num / rect.size.x;
 int x = num - z*rect.size.x;
 return new Coord(x+rect.offset.x, z+rect.offset.z);
}

public T GetRaw(int x, int z)
/// Gets value without using an offset
{
 return arr[z*rect.size.x + x]; //rect fn duplicated to increase performance
}
```

```
public void SetRaw (int x, int z, T value)
/// Sets value without offset
{
 arr[z*rect.size.x + x] = value;
}

public void Clear () { for (int i=0; i<arr.Length; i++) arr[i] = default(T); }

public void ChangeRect (CoordRect newRect, bool forceNewArray=false) //will re-create array only if capa
{
 rect = newRect;
 count = newRect.size.x*newRect.size.z;

 if (arr.Length!=count || forceNewArray) arr = new T[count];
}

public virtual object Clone () { return Clone(null); } //separate fn for IClonable
public Matrix2D<T> Clone (Matrix2D<T> result)
{
 if (result==null) result = new Matrix2D<T>(rect);

 //copy params
 result.rect = rect;
 result.pos = pos;
 result.count = count;
```

```
 //copy array
 //result.array = (float[])array.Clone(); //no need to create it any time
 if (result.arr.Length != arr.Length) result.arr = new T[arr.Length];
 for (int i=0; i<arr.Length; i++)
  result.arr[i] = arr[i];

 return result;
}

public void Fill (T v) { for (int i=0; i<count; i++) arr[i] = v; }

public void Fill (Matrix2D<T> m, bool removeBorders=false)
{
 CoordRect intersection = CoordRect.Intersected(rect, m.rect);
 Coord min = intersection.Min; Coord max = intersection.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
   this[x,z] = m[x,z];
 if (removeBorders) RemoveBorders(intersection);
}

#region Quick Pos

 public void SetPos(int x, int z) { pos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x; }
 public void SetPos(int x, int z, int s) { pos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x  +  s*rect.size.x*r
```

```
 public void MoveX() { pos++; }
 public void MoveZ() { pos += rect.size.x; }
 public void MovePrevX() { pos--; }
 public void MovePrevZ() { pos -= rect.size.x; }

 //public float current { get { return array[pos]; } set { array[pos] = value; } }
 /*public T nextX { get { return array[pos+1]; } set { array[pos+1] = value; } }
 public T prevX { get { return array[pos-1]; } set { array[pos-1] = value; } }
 public T nextZ { get { return array[pos+rect.size.x]; } set { array[pos+rect.size.x] = value; } }
 public T prevZ { get { return array[pos-rect.size.x]; } set { array[pos-rect.size.x] = value; } }
 public T nextXnextZ { get { return array[pos+rect.size.x+1]; } set { array[pos+rect.size.x+1] = value; } }
 public T prevXnextZ { get { return array[pos+rect.size.x-1]; } set { array[pos+rect.size.x-1] = value; } }
 public T nextXprevZ { get { return array[pos-rect.size.x+1]; } set { array[pos-rect.size.x+1] = value; } }
 public T prevXprevZ { get { return array[pos-rect.size.x-1]; } set { array[pos-rect.size.x-1] = value; } }*/

#endregion

#region OrderedFromCenter

 /*public Coord[] GetOrderedFromCenterCoords ()
 {
  Coord[] sortedByDistance = new Coord[array.Length];
  int i=0;
  Coord min = rect.Min; Coord max = rect.Max;
  for (int x=min.x; x<max.x; x++)
```

```
  for (int z=min.z; z<max.z; z++)
   { sortedByDistance[i] = new Coord(x,z); i++; }

 float[] distances = new float[array.Length];
 for (int z=0; z<rect.size.z; z++)
  for (int x=0; x<rect.size.x; x++)
   distances[z*rect.size.x + x] = (x-rect.size.x/2)*(x-rect.size.x/2) + (z-rect.size.z/2)*(z-rect.size.z/2); //Math

 Extensions.ArrayQSort(sortedByDistance, distances);
 return sortedByDistance;
}

public IEnumerable<Coord> OrderedFromCenterCoord ()
{
 Coord[] sortedByDistance = GetOrderedFromCenterCoords();
 for (int i=0; i<sortedByDistance.Length; i++)
  yield return sortedByDistance[i];
}

public IEnumerable<T> OrderedFromCenter ()
{
 Coord[] sortedByDistance = GetOrderedFromCenterCoords();
 for (int i=0; i<sortedByDistance.Length; i++)
  yield return this[sortedByDistance[i]];
}*/
```

```
#endregion

#region Borders

 public void RemoveBorders ()
 {
  Coord min = rect.Min; Coord last = rect.Max - 1;

  for (int x=min.x; x<=last.x; x++)
   { SetPos(x,min.z); arr[pos] = arr[pos+rect.size.x]; }

  for (int x=min.x; x<=last.x; x++)
   { SetPos(x,last.z); arr[pos] = arr[pos-rect.size.x]; }

  for (int z=min.z; z<=last.z; z++)
   { SetPos(min.x,z); arr[pos] = arr[pos+1]; }

  for (int z=min.z; z<=last.z; z++)
   { SetPos(last.x,z); arr[pos] = arr[pos-1]; }
 }

 public void RemoveBorders (int borderMinX, int borderMinZ, int borderMaxX, int borderMaxZ)
 {
  Coord min = rect.Min; Coord max = rect.Max;

  if (borderMinZ != 0)
```

```
for (int x=min.x; x<max.x; x++)
{
 T val = this[x, min.z+borderMinZ];
 for (int z=min.z; z<min.z+borderMinZ; z++) this[x,z] = val;
}

if (borderMaxZ != 0)
for (int x=min.x; x<max.x; x++)
{
 T val = this[x, max.z-borderMaxZ];
 for (int z=max.z-borderMaxZ; z<max.z; z++) this[x,z] = val;
}

if (borderMinX != 0)
for (int z=min.z; z<max.z; z++)
{
 T val = this[min.x+borderMinX, z];
 for (int x=min.x; x<min.x+borderMinX; x++) this[x,z] = val;
}

if (borderMaxX != 0)
for (int z=min.z; z<max.z; z++)
{
 T val = this[max.x-borderMaxX, z];
 for (int x=max.x-borderMaxX; x<max.x; x++) this[x,z] = val;
}
```

```
  }

  public void RemoveBorders (CoordRect centerRect)
  {
   RemoveBorders(
    Mathf.Max(0,centerRect.offset.x-rect.offset.x),
    Mathf.Max(0,centerRect.offset.z-rect.offset.z),
    Mathf.Max(0,rect.Max.x-centerRect.Max.x+1),
    Mathf.Max(0,rect.Max.z-centerRect.Max.z+1) );
  }

 #endregion
}

}
```

```
ï»¿using UnityEngine;
using System.Collections;
using System.Collections.Generic;

namespace Den.Tools
{
 [System.Serializable]
 public struct CoordDir
 {
  public int x; public int y; public int z;
  public byte dir; //0-5 sides, 6 is itself, 7 is non-existing CoordDir

  static readonly public byte[] oppositeDir = {1,0,3,2,5,4};

  static readonly public int[] dirToPosX = {0, 0, 1,-1, 0, 0};
  static readonly public int[] dirToPosY = {1,-1, 0, 0, 0, 0};
  static readonly public int[] dirToPosZ = {0, 0, 0, 0, 1,-1};


  public CoordDir opposite
   {get{ return new CoordDir(x+dirToPosX[dir], y+CoordDir.dirToPosY[dir], z+CoordDir.dirToPosZ[dir], oppo

  public CoordDir (bool empty) {this.x=0; this.y=0; this.z=0; this.dir=0; if (!empty) dir=7; }
  public static CoordDir empty {get{ return new CoordDir(0,0,0,7); }}
  public CoordDir (int x, int y, int z) {this.x=x; this.y=y; this.z=z; this.dir=7; }
  public CoordDir (int x, int y, int z, byte d) {this.x=x; this.y=y; this.z=z; this.dir=d; }
```

```
public CoordDir (CoordDir c, byte d) {this.x=c.x; this.y=c.y; this.z=c.z; this.dir=d; }

public bool exists {get{return dir!=7;}}
public Vector3 center { get{ return new Vector3(x+0.5f,y+0.5f,z+0.5f); } }
public Vector3 pos { get{ return new Vector3(x,y,z); } }

public static CoordDir operator + (CoordDir a, CoordDir b) { return new CoordDir(a.x+b.x, a.y+b.y, a.z+b.z
public static CoordDir operator + (CoordDir a, int i) { return new CoordDir(a.x+i, a.y+i, a.z+i); }
public static CoordDir operator - (CoordDir a, CoordDir b) { return new CoordDir(a.x-b.x, a.y-b.y, a.z-b.z);
public static CoordDir operator - (CoordDir a, int i) { return new CoordDir(a.x-i, a.y-i, a.z-i); }
public static CoordDir operator ++ (CoordDir a) { return new CoordDir(a.x+1, a.y+1, a.z+1); }
public static CoordDir operator * (CoordDir a, int s) { return  new CoordDir(a.x*s, a.y*s, a.z*s); }
public static CoordDir operator * (CoordDir a, CoordDir b) { return  new CoordDir(a.x*b.x, a.y*b.y, a.z*b.z)
public static CoordDir operator / (CoordDir a, int s) { return  new CoordDir(a.x/s, a.y/s, a.z/s); }
public static CoordDir operator / (CoordDir a, CoordDir b) { return  new CoordDir(a.x/b.x, a.y/b.y, a.z/b.z);

public static bool operator == (CoordDir a, CoordDir b) { return (a.x==b.x && a.y==b.y && a.z==b.z && a.d
public static bool operator != (CoordDir a, CoordDir b) { return !(a.x==b.x && a.y==b.y && a.z==b.z && a.d
public override bool Equals(object obj) { return base.Equals(obj); }
public override int GetHashCode() {return ((y & 0xFFFFFFF) << 32)  |  ((x & 0xFFFF) << 16)  |  (z & 0xFFF

public static bool operator > (CoordDir a, CoordDir b) { return (a.x>b.x || a.z>b.z); } //do not compare y's
public static bool operator >= (CoordDir a, CoordDir b) { return (a.x>=b.x || a.z>=b.z); }
public static bool operator < (CoordDir a, CoordDir b) { return (a.x<b.x || a.z<b.z); }
public static bool operator <= (CoordDir a, CoordDir b) { return (a.x<=b.x || a.z<=b.z); }
```

```
public static CoordDir Min (CoordDir[] coords)
{
 CoordDir min = new CoordDir(int.MaxValue, int.MaxValue, int.MaxValue,7);
 for (int i=0; i<coords.Length; i++)
 {
  if (coords[i].x < min.x) min.x = coords[i].x;
  if (coords[i].y < min.y) min.y = coords[i].y;
  if (coords[i].z < min.z) min.z = coords[i].z;
 }
 return min;
}

public static CoordDir Max (CoordDir[] coords)
{
 CoordDir max = new CoordDir(int.MinValue, int.MinValue, int.MinValue,7);
 for (int i=0; i<coords.Length; i++)
 {
  if (coords[i].x > max.x) max.x = coords[i].x;
  if (coords[i].y > max.y) max.y = coords[i].y;
  if (coords[i].z > max.z) max.z = coords[i].z;
 }
 return max;
}

public CoordDir GetChunkCoord (CoordDir worldCoord, int chunkSize) //gets chunk CoordDirinates using
```

```
{
 return new CoordDir
  (
   worldCoord.x>=0 ? (int)(worldCoord.x/chunkSize) : (int)((worldCoord.x+1)/chunkSize)-1,
   0,
   worldCoord.z>=0 ? (int)(worldCoord.z/chunkSize) : (int)((worldCoord.z+1)/chunkSize)-1
   );
}

public int BlockMagnitude2 { get { return Mathf.Abs(x)+Mathf.Abs(z); } }
public int BlockMagnitude3 { get { return Mathf.Abs(x)+Mathf.Abs(y)+Mathf.Abs(z); } }

public override string ToString() { return "x:"+x+" y:"+y+" z:"+z+" dir:"+dir; }

public Vector3 vector3 { get { return new Vector3(x,y,z); }}
public Vector3 vector3centered { get { return new Vector3(x+0.5f,y+0.5f,z+0.5f); }}

public Coord coord { get{ return new Coord(x,z); }}

public static byte NormalToDir (Vector3 normal)
{
 float absX = normal.x>0? normal.x : -normal.x;
 float absY = normal.y>0? normal.y : -normal.y;
 float absZ = normal.z>0? normal.z : -normal.z;

 if (absY>absX && absY>absZ)
```

```
 {
  if (normal.y>0) return 0;
  else return 1;
 }

 else if (absX>absY && absX>absZ)
 {
  if (normal.x>0) return 2;
  else return 3;
 }

 else if (absZ>absY && absZ>absX)
 {
  if (normal.z>0) return 4;
  else return 5;
 }

 else return 7; //not exists
}

#region Neighbours

 public static readonly CoordDir[] neigsLut = new CoordDir[] {
  //planar block      //this block  //concave corner
  //dir0
  new CoordDir(0, 0, 1, 0), new CoordDir(0,0,0,4),  new CoordDir(0, 1, 1, 5),
```

```
new CoordDir(1, 0, 0, 0), new CoordDir(0,0,0,2),  new CoordDir(1, 1, 0, 3),
new CoordDir(0, 0,-1, 0), new CoordDir(0,0,0,5),  new CoordDir(0, 1,-1, 4),
new CoordDir(-1,0, 0, 0), new CoordDir(0,0,0,3),  new CoordDir(-1,1, 0, 2),
//dir1
new CoordDir(0, 0, 1, 1), new CoordDir(0,0,0,4),  new CoordDir(0,-1, 1, 5),
new CoordDir(-1,0, 0, 1), new CoordDir(0,0,0,3),  new CoordDir(-1,-1,0, 2),
new CoordDir(0, 0,-1, 1), new CoordDir(0,0,0,5),  new CoordDir(0,-1,-1, 4),
new CoordDir(1, 0, 0, 1), new CoordDir(0,0,0,2),  new CoordDir(1,-1, 0, 3),
//dir2
new CoordDir(0, 1, 0, 2), new CoordDir(0,0,0,0),  new CoordDir(1, 1, 0, 1),
new CoordDir(0, 0, 1, 2), new CoordDir(0,0,0,4),  new CoordDir(1, 0, 1, 5),
new CoordDir(0,-1, 0, 2), new CoordDir(0,0,0,1),  new CoordDir(1,-1, 0, 0),
new CoordDir(0, 0,-1, 2), new CoordDir(0,0,0,5),  new CoordDir(1, 0,-1, 4),
//dir3
new CoordDir(0, 1, 0, 3), new CoordDir(0,0,0,0),  new CoordDir(-1, 1, 0, 1),
new CoordDir(0, 0,-1, 3), new CoordDir(0,0,0,5),  new CoordDir(-1, 0,-1, 4),
new CoordDir(0,-1, 0, 3), new CoordDir(0,0,0,1),  new CoordDir(-1,-1, 0, 0),
new CoordDir(0, 0, 1, 3), new CoordDir(0,0,0,4),  new CoordDir(-1, 0, 1, 5),
//dir4
new CoordDir(1, 0, 0, 4), new CoordDir(0,0,0,2),  new CoordDir(1, 0, 1, 3),
new CoordDir(0, 1, 0, 4), new CoordDir(0,0,0,0),  new CoordDir(0, 1, 1, 1),
new CoordDir(-1,0, 0, 4), new CoordDir(0,0,0,3),  new CoordDir(-1,0, 1, 2),
new CoordDir(0,-1, 0, 4), new CoordDir(0,0,0,1),  new CoordDir(0,-1, 1, 0),
//dir5
new CoordDir(1, 0, 0, 5), new CoordDir(0,0,0,2),  new CoordDir(1, 0,-1, 3),
new CoordDir(0,-1, 0, 5), new CoordDir(0,0,0,1),  new CoordDir(0,-1,-1, 0),
```

```
    new CoordDir(-1,0, 0, 5), new CoordDir(0,0,0,3),  new CoordDir(-1,0,-1, 2),
    new CoordDir(0, 1, 0, 5), new CoordDir(0,0,0,0),  new CoordDir(0, 1,-1, 1)
   };

 #endregion

}

[System.Serializable]
public struct CoordCube
{
 public CoordDir offset;
 public CoordDir size;

 public CoordCube (CoordDir offset, CoordDir size) { this.offset = offset; this.size = size; }
 public CoordCube (int offsetX, int offsetY, int offsetZ, int sizeX, int sizeY, int sizeZ) { this.offset = new Coo

 public CoordDir Max { get { return offset+size; } set { offset = value-size; } }
 public CoordDir Min { get { return offset; } set { offset = value; } }
 public CoordDir Center { get { return offset + size/2; } }

 public static bool operator == (CoordCube c1, CoordCube c2) { return c1.offset.x==c2.offset.x && c1.offse
 public static bool operator != (CoordCube c1, CoordCube c2) { return c1.offset.x!=c2.offset.x || c1.offset.y
 public override bool Equals(object obj) { return base.Equals(obj); }
 public override int GetHashCode() {return offset.GetHashCode()^size.GetHashCode(); }
```

```
public static CoordCube operator * (CoordCube c, int s) { return  new CoordCube(c.offset*s, c.size*s); }
public static CoordCube operator / (CoordCube c, int s) { return  new CoordCube(c.offset/s, c.size/s); }

public int this[int x, int y, int z] { get { return (z-offset.z)*size.x*size.y + (y-offset.y)*size.x + x - offset.x; }}
public int this[CoordDir c] { get { return (c.z-offset.z)*size.x*size.y + (c.y-offset.y)*size.x + c.x - offset.x; }}

public bool Contains (CoordDir c)
{
 return  c.x>=offset.x && c.x<offset.x+size.x &&
   c.y>=offset.y && c.y<offset.y+size.y &&
   c.z>=offset.z && c.z<offset.z+size.z ;
}

public bool Contains (int x, int y, int z)
{
 return  x>=offset.x && x<offset.x+size.x &&
   y>=offset.y && y<offset.y+size.y &&
   z>=offset.z && z<offset.z+size.z ;
}

public int GetPos (int x, int y, int z)
{
 #if WDEBUG
 if (x<offset.x || x>=offset.x+size.x) throw new System.ArgumentOutOfRangeException("x", "Index Out Of
 if (y<offset.y || y>=offset.y+size.y) throw new System.ArgumentOutOfRangeException("y", "Index Out Of
 if (z<offset.z || z>=offset.z+size.z) throw new System.ArgumentOutOfRangeException("z", "Index Out Of
```

```
 #endif

 return (z-offset.z)*size.x*size.y + (y-offset.y)*size.x + x - offset.x;
}

public int GetPos (CoordDir c)
{
 #if WDEBUG
 if (c.x<offset.x || c.x>=offset.x+size.x) throw new System.ArgumentOutOfRangeException("x", "Index Out
 if (c.y<offset.y || c.y>=offset.y+size.y) throw new System.ArgumentOutOfRangeException("y", "Index Out
 if (c.z<offset.z || c.z>=offset.z+size.z) throw new System.ArgumentOutOfRangeException("z", "Index Out
 #endif

 return (c.z-offset.z)*size.x*size.y + (c.y-offset.y)*size.x + c.x - offset.x;
}

public void Encapsulate (CoordDir coord)
/// Resizes this rect so that coord is included
{
 if (coord.x < offset.x) { size.x += offset.x-coord.x; offset.x = coord.x; }
 if (coord.x >= offset.x+size.x) { size.x = coord.x-offset.x+1; }

 if (coord.y < offset.y) { size.y += offset.y-coord.y; offset.y = coord.y; }
 if (coord.y >= offset.y+size.y) { size.y = coord.y-offset.y+1; }

 if (coord.z < offset.z) { size.z += offset.z-coord.z; offset.z = coord.z; }
```

```
  if (coord.z >= offset.z+size.z) { size.z = coord.z-offset.z+1; }
 }

 public CoordRect rect {get{ return new CoordRect(offset.coord, size.coord); }}
}

[System.Serializable]
public class Matrix3D<T>
{
 public CoordCube cube;
 public T[] array;
 public int count;

 /*public int sizeX;
 public int sizeY;
 public int sizeZ;

 public int offsetX;
 public int offsetY;
 public int offsetZ;*/

 //public int pos;

 public T this[int x, int y, int z]
 {
  get {
```

```
  #if WDEBUG
  if (x<cube.offset.x || x>=cube.offset.x+cube.size.x) throw new System.ArgumentOutOfRangeException("
  if (y<cube.offset.y || y>=cube.offset.y+cube.size.y) throw new System.ArgumentOutOfRangeException("
  if (z<cube.offset.z || z>=cube.offset.z+cube.size.z) throw new System.ArgumentOutOfRangeException("
  #endif

  return array[(z-cube.offset.z)*cube.size.x*cube.size.y + (y-cube.offset.y)*cube.size.x + x - cube.offset.x];

 set {
  #if WDEBUG
  if (x<cube.offset.x || x>=cube.offset.x+cube.size.x) throw new System.ArgumentOutOfRangeException("
  if (y<cube.offset.y || y>=cube.offset.y+cube.size.y) throw new System.ArgumentOutOfRangeException("
  if (z<cube.offset.z || z>=cube.offset.z+cube.size.z) throw new System.ArgumentOutOfRangeException("
  #endif

  array[(z-cube.offset.z)*cube.size.x*cube.size.y + (y-cube.offset.y)*cube.size.x + x - cube.offset.x] = value
}

public T this[CoordDir c]
{
 get {
  #if WDEBUG
  if (c.x<cube.offset.x || c.x>=cube.offset.x+cube.size.x) throw new System.ArgumentOutOfRangeExceptio
  if (c.y<cube.offset.y || c.y>=cube.offset.y+cube.size.y) throw new System.ArgumentOutOfRangeExceptio
  if (c.z<cube.offset.z || c.z>=cube.offset.z+cube.size.z) throw new System.ArgumentOutOfRangeExceptio
  #endif
```

```
 return array[(c.z-cube.offset.z)*cube.size.x*cube.size.y + (c.y-cube.offset.y)*cube.size.x + c.x - cube.offs


set {
 #if WDEBUG
 if (c.x<cube.offset.x || c.x>=cube.offset.x+cube.size.x) throw new System.ArgumentOutOfRangeExceptio
 if (c.y<cube.offset.y || c.y>=cube.offset.y+cube.size.y) throw new System.ArgumentOutOfRangeExceptio
 if (c.z<cube.offset.z || c.z>=cube.offset.z+cube.size.z) throw new System.ArgumentOutOfRangeExceptio
 #endif


 array[(c.z-cube.offset.z)*cube.size.x*cube.size.y + (c.y-cube.offset.y)*cube.size.x + c.x - cube.offset.x] =
}
 /*
 try {array[(z-offsetZ)*sizeX*sizeY + (y-offsetY)*sizeX + x - offsetX] = value; }
 catch(System.Exception ex) { Debug.Log("offsetX:" + offsetX + " sizeX:" + sizeX +
  "offsetY:" + offsetY + " sizeY:" + sizeY +
  "offsetZ:" + offsetZ + " sizeZ:" + sizeZ +
  "   coords:" + x + ", " + y + ", " + z); throw; }
 }
 /*
 get
 {
 if (x-offsetX < 0) Debug.LogError("Value of x (" + x.ToString() + ") is less then offset (" + offsetX.ToString
 if (y-offsetY < 0) Debug.LogError("Value of y (" + y.ToString() + ") is less then offset (" + offsetY.ToString
 if (z-offsetZ < 0) Debug.LogError("Value of z (" + z.ToString() + ") is less then offset (" + offsetZ.ToString
 if (x-offsetX >= sizeX) Debug.LogError("Value of x (" + x.ToString() + ") is equal or more then size (" + si
```

```
  if (y-offsetY >= sizeY) Debug.LogError("Value of y (" + y.ToString() + ") is equal or more then size (" + si
  if (z-offsetZ >= sizeZ) Debug.LogError("Value of z (" + z.ToString() + ") is equal or more then size (" + si
  return array[(z-offsetZ)*sizeX*sizeY + (y-offsetY)*sizeX + x - offsetX];
 }
 */

/*public int GetPos (int x,int y,int z) { return (z-offsetZ)*sizeX*sizeY + (y-offsetY)*sizeX + x - offsetX; }
public void SetPos (int x, int y, int z) { pos = (z-offsetZ)*sizeX*sizeY + (y-offsetY)*sizeX + x - offsetX; }
 //if (pos>=array.Length) Debug.Log("pos:" + x + " " + y + " " + z + " offset:" + offsetX + " " + offsetY + " " +
public void MovePos (int x, int y, int z) { pos += z*sizeX*sizeY + y*sizeX + x; }
public void MovePosNextY () { pos += sizeX; }
public void MovePosPrevY () { pos -= sizeX; }

public T current { get { return array[pos]; }    set { array[pos] = value; } }
public T nextX { get { return array[pos+1]; }    set { array[pos+1] = value; }  }
public T prevX { get { return array[pos-1]; }    set { array[pos-1] = value; }  }
public T nextY { get { return array[pos+sizeX]; }  set { array[pos+sizeX] = value; }  }
public T prevY { get { return array[pos-sizeX]; }   set { array[pos-sizeX] = value; }  }
public T nextZ { get { return array[pos+sizeX*sizeY]; } set { array[pos+sizeX*sizeY] = value; }  }
public T prevZ { get { return array[pos-sizeX*sizeY]; } set { array[pos-sizeX*sizeY] = value; }  }
public T nextXnextY { get { return array[pos+1+sizeX]; } set { array[pos+1+sizeX] = value; } }
public T prevXnextY { get { return array[pos-1+sizeX]; } set { array[pos-1+sizeX] = value; } }
public T nextZnextY { get { return array[pos+sizeX*sizeY+sizeX]; } set { array[pos+sizeX*sizeY+sizeX] = v
public T prevZnextY { get { return array[pos-sizeX*sizeY+sizeX]; } set { array[pos-sizeX*sizeY+sizeX] = va
public T nextXprevY { get { return array[pos+1-sizeX]; } set { array[pos+1-sizeX] = value; } }
public T prevXprevY { get { return array[pos-1-sizeX]; } set { array[pos-1-sizeX] = value; } }
```

```
public T nextZprevY { get { return array[pos+sizeX*sizeY-sizeX]; } set { array[pos+sizeX*sizeY-sizeX] = va
public T prevZprevY { get { return array[pos-sizeX*sizeY-sizeX]; } set { array[pos-sizeX*sizeY-sizeX] = val

/*
public T SafeGet (int x, int y, int z)
{
 x = Mathf.Clamp(x, 0,sizeX);
 y = Mathf.Clamp(y, 0,sizeY);
 z = Mathf.Clamp(z, 0,sizeZ);
 return array[z*sizeX*sizeY + y*sizeX + x];
}
*/

public Matrix3D () { }

public Matrix3D (CoordCube cube, T[] array=null)
{
 this.cube = cube;
 count = cube.size.x * cube.size.y * cube.size.z;

 if (array != null && array.Length<count) Debug.LogError("Array length: " + array.Length + " is lower then r
 if (array != null && array.Length>=count) this.array = array;
 else this.array = new T[count];
}

public Matrix3D (CoordDir offset, CoordDir size, T[] array=null)
```

```
{
 cube = new CoordCube(offset,size);
 count = cube.size.x * cube.size.y * cube.size.z;

 if (array != null && array.Length<count) Debug.LogError("Array length: " + array.Length + " is lower then r
 if (array != null && array.Length>=count) this.array = array;
 else this.array = new T[count];
}

public Matrix3D (int x, int y, int z, T[] array=null)
{
 this.cube = new CoordCube(0,0,0,x,y,z);
 count = cube.size.x * cube.size.y * cube.size.z;

 if (array != null && array.Length<count) Debug.LogError("Array length: " + array.Length + " is lower then r
 if (array != null && array.Length>=count) this.array = array;
 else this.array = new T[count];
}

public Matrix3D (int ox, int oy, int oz, int sx, int sy, int sz, T[] array=null)
{
 this.cube = new CoordCube(ox,oy,oz,sx,sy,sz);
 count = cube.size.x * cube.size.y * cube.size.z;

 if (array != null && array.Length<count) Debug.LogError("Array length: " + array.Length + " is lower then r
 if (array != null && array.Length>=count) this.array = array;
```

```
 else this.array = new T[count];
}

public void Fill(T def)
{
 for(int i=0;i<array.Length;i++) array[i] = def;
}

public Matrix3D<T> Copy ()
{
 Matrix3D<T> newMatrix = new Matrix3D<T>(cube);
 for (int i=0; i<array.Length; i++) newMatrix.array[i] = array[i];
 return newMatrix;
}

public Matrix2D<T> Matrix2
{get{
 #if WDEBUG
 if (cube.size.y!=1) Debug.LogError("Trying to convert non-flat Matrix3 to Matrix2");
 #endif
 return new Matrix2D<T>( new CoordRect(cube.offset.x, cube.offset.z, cube.size.x, cube.size.z), array);
}}

public static void CopyData (Matrix3D<T> src, Matrix3D<T> dst)
{
 for (int x=dst.cube.offset.x; x<dst.cube.offset.x+dst.cube.size.x; x++)
```

```
  for (int y=dst.cube.offset.y; y<dst.cube.offset.y+dst.cube.size.y; y++)
   for (int z=dst.cube.offset.z; z<dst.cube.offset.z+dst.cube.size.z; z++)
   {
    if (src.cube.Contains(x,y,z))
     dst[x,y,z] = src[x,y,z];
   }
}

/*public void InsertLayer (int x, int z, int start, int depth, T val)
{
 int min = offsetY; if (start>min) min=start;
 int max = offsetY+sizeY; if (start+depth<max) max=start+depth;

 int i = (z-offsetZ)*sizeX*sizeY + (min-offsetY)*sizeX + x - offsetX;

 for (int y=min; y<max; y++)
 {
  array[i] = val;
  i += sizeX;
 }
}

public static Matrix3<T> Rescale (Matrix3<T> src, int offsetX, int offsetY, int offsetZ, int sizeX, int sizeY, in
{
 Matrix3<T> dst = new Matrix3<T>(sizeX, sizeY, sizeZ);
 dst.offsetX = offsetX; dst.offsetY = offsetY; dst.offsetZ = offsetZ;
```

```
  for (int x=dst.offsetX; x<dst.offsetX+dst.sizeX; x++)
  {
   if (x<src.offsetX || x>=src.offsetX+src.sizeX) continue;

   for (int y=dst.offsetY; y<dst.offsetY+dst.sizeY; y++)
   {
    if (y<src.offsetY || y>=src.offsetY+src.sizeY) continue;

    for (int z=dst.offsetZ; z<dst.offsetZ+dst.sizeZ; z++)
    {
     if (z<src.offsetZ || z>=src.offsetZ+src.sizeZ) continue;

     //dst[x,y,z] = src[x,y,z];
     dst.array[(z-dst.offsetZ)*dst.sizeX*dst.sizeY + (y-dst.offsetY)*dst.sizeX + x - dst.offsetX] = src.array[(z-s
    }
   }
  }
  return dst;
 }*/
}

/* [System.Serializable]
 public struct Matrix4<T>
 {
  public T[] array; //must be private
```

```
public int sizeX;
public int sizeY;
public int sizeZ;
public int sizeW;

public int stepZ;
public int stepW;

public int offsetX;
public int offsetY;
public int offsetZ;
public int offsetW;

public int pos;

public T this[int x, int y, int z, int w]
{
 get { return array[(w-offsetW)*stepW + (z-offsetZ)*stepZ + (y-offsetY)*sizeX + x - offsetX]; }
 set { array[(w-offsetW)*stepW + (z-offsetZ)*stepZ + (y-offsetY)*sizeX + x - offsetX] = value; }
}

public bool CheckInRange (int x, int y, int z, int w)
{
 return (x-offsetX >= 0 && x-offsetX < sizeX &&
         y-offsetY >= 0 && y-offsetY < sizeY &&
```

```
        z-offsetZ >= 0 && z-offsetZ < sizeZ &&
        w-offsetW >= 0 && w-offsetW < sizeW);
}

public bool CheckInRange (int x, int y, int z)
{
 return (x-offsetX >= 0 && x-offsetX < sizeX &&
        y-offsetY >= 0 && y-offsetY < sizeY &&
        z-offsetZ >= 0 && z-offsetZ < sizeZ);
}

public void SetPos (int x, int y, int z, int w) { pos = (w-offsetW)*stepW + (z-offsetZ)*stepZ + (y-offsetY)*siz
public void MovePos (int x, int y, int z, int w) { pos += stepW + z*stepZ + y*sizeX + x; }
public void MovePosNextY ()  { pos += sizeX; }

public T current { get { return array[pos]; }    set { array[pos] = value; } }
public T nextX { get { return array[pos+1]; }    set { array[pos+1] = value; }  }
public T prevX { get { return array[pos-1]; }    set { array[pos-1] = value; }  }
public T nextY { get { return array[pos+sizeX]; }  set { array[pos+sizeX] = value; }  }
public T prevY { get { return array[pos-sizeX]; }   set { array[pos-sizeX] = value; }  }
public T nextZ { get { return array[pos+stepZ]; } set { array[pos+stepZ] = value; }  }
public T prevZ { get { return array[pos-stepZ]; } set { array[pos-stepZ] = value; }  }
public T nextW { get { return array[pos+stepZ]; } set { array[pos+stepW] = value; }  }
public T prevW { get { return array[pos-stepZ]; } set { array[pos-stepW] = value; }  }
public T nextXnextY { get { return array[pos+1+sizeX]; } set { array[pos+1+sizeX] = value; } }
public T prevXnextY { get { return array[pos-1+sizeX]; } set { array[pos-1+sizeX] = value; } }
```

```
 public T nextZnextY { get { return array[pos+stepZ+sizeX]; } set { array[pos+stepZ+sizeX] = value; } }
 public T prevZnextY { get { return array[pos-stepZ+sizeX]; } set { array[pos-stepZ+sizeX] = value; } }

 public Matrix4 (int x, int y, int z, int w)
 {
  array = new T[w*x*y*z];
  sizeX = x;
  sizeY = y;
  sizeZ = z;
  sizeW = w;
  offsetX = 0;
  offsetY = 0;
  offsetZ = 0;
  offsetW = 0;
  pos = 0;
  stepW = sizeX*sizeY*sizeZ;
  stepZ = sizeX*sizeY;
 }

 public void Reset (T def)
 {
  if (array == null) array = new T[0];
  for (int i=0; i<array.Length; i++) array[i] = def;
 }
}*/
```

}

```
using UnityEngine;
using System.Collections;
using System;
using System.IO;

using Den.Tools.Matrices;

namespace Den.Tools
{
 [HelpURL("https://gitlab.com/denispahunov/mapmagic/wikis/home")]
 [CreateAssetMenu(menuName = "MapMagic/Imported Map", fileName = "Imported Map.asset", order = 11
 [Serializable, PreferBinarySerialization]
 public class MatrixAsset : ScriptableObject
 {
  public Matrix matrix = new Matrix( new CoordRect(0,0,1,1) );
  public Texture2D preview;

  public enum Source { Raw, Texture, New }
  public Source source;

  //values to reload
  public string rawPath;
  public Texture2D textureSource;
  public enum Channel { Average, Grayscale, Red, Green, Blue, Alpha };
  public Channel channelSource;
```

```
//public static WeakEvent<MatrixAsset,Matrix> OnReloaded = new WeakEvent<MatrixAsset,Matrix>();
//public static WeakEvent<Texture2D> OnTextureImported = new WeakEvent<Texture2D>();

public static Action<MatrixAsset> OnReloaded;
public static Action<Texture2D> OnTextureImported; //called in AssetPostprocessor

public MatrixAsset() : base()
{
 OnTextureImported +=
  tex => { if (source == Source.Texture && textureSource == tex) Reload(); };
}


public static void ImportRaw (ref Matrix matrix, string path=null)
{
 //reading file
 FileInfo fileInfo = new FileInfo(path);
 FileStream stream = fileInfo.Open(FileMode.Open, FileAccess.Read);

 int size = (int)Mathf.Sqrt(stream.Length/2);
 byte[] vals = new byte[size*size*2];

 stream.Read(vals,0,vals.Length);
 stream.Close();

 //setting matrix
```

```
 if (matrix == null  ||  matrix.rect.size.x != size  ||  matrix.rect.size.z != size)
  matrix = new Matrix( new CoordRect(0,0,size,size) );
 matrix.ImportRaw16(vals, size, size);

 //flipping vertically
 Matrix flipped = new Matrix(matrix.rect);
 MatrixOps.FlipVertical(matrix, flipped);
 matrix = flipped;
}


public static void ImportTexture (ref Matrix matrix, Texture2D texture, Channel channel = Channel.Averag
{
 if (!texture.IsReadable())
  texture = texture.ReadableClone();

 Color[] colors = texture.GetPixels();

 //setting matrix
 if (matrix == null  ||  matrix.rect.size.x != texture.width  ||  matrix.rect.size.z != texture.height)
  matrix = new Matrix( new CoordRect(0,0,texture.width,texture.height) );
 int i = 0;
 Coord min = matrix.rect.Min; Coord max = matrix.rect.Max;
 //for (int z=max.z-1; z>=min.z; z--)
 for (int z=min.z; z<max.z; z++)
  for (int x=min.x; x<max.x; x++)
```

```
 {
  float val;

  switch (channel)
  {
   case Channel.Average: default: val = (colors[i].r + colors[i].g + colors[i].b) / 3f; break;
   case Channel.Grayscale: val = 0.21f*colors[i].r + 0.72f*colors[i].g + 0.07f*colors[i].b; break;
   case Channel.Red: val = colors[i].r; break;
   case Channel.Green: val = colors[i].g; break;
   case Channel.Blue: val = colors[i].b; break;
   case Channel.Alpha: val = colors[i].a; break;
  }

  matrix[x,z] = val;
  i++;
 }
}



public static void ImportArray (ref Matrix matrix, float[,] heights)
{
 //creating ref matrix
 if (matrix == null  ||  matrix.rect.size.x != heights.GetLength(1)  ||  matrix.rect.size.z != heights.GetLength(
  matrix = new Matrix( new CoordRect(0, 0, heights.GetLength(1), heights.GetLength(0)) );

 //importing
```

```
 matrix.ImportHeights(heights);
}



public void RefreshPreview (int size=128)
{
 if (matrix != null)
 {
  Matrix previewMatrix = matrix;// new Matrix( new CoordRect(0,0,size,size) );
  //MatrixOps.Resize(matrix, previewMatrix);
  preview = new Texture2D(previewMatrix.rect.size.x, previewMatrix.rect.size.z);
  previewMatrix.ExportTexture(preview, -1);
 }
 else preview = TextureExtensions.ColorTexture(2,2,Color.black);
}



public void Reload ()
{
 if (source == Source.Raw)
 {
  if (rawPath != null)
   ImportRaw(ref matrix, rawPath);
 }
 else
 {
```

```
    if (textureSource != null)
      ImportTexture(ref matrix, textureSource, channelSource);
    }

    RefreshPreview(256);

    OnReloaded?.Invoke(this);
  }
 }
}
```

```
ï»¿using UnityEngine;
using System;
using System.Collections.Generic;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.GUI;



namespace Den.Tools.Matrices
{
 public class MatrixTextureGizmo : DebugGizmos.IGizmo
 {
  private Mesh mesh;
  private Texture2D texture;
  private byte[] bytes;
  private Material material;
  private Matrix4x4 tfm;

  private static readonly Vector3[] verts = new Vector3[] { new Vector3(0,0,0), new Vector3(0,0,1), new Vec
  private static readonly Vector2[] uvs = new Vector2[] { new Vector3(0,0), new Vector3(0,1), new Vector3(1
  private static readonly int[] tris = new int[] { 1, 2, 0,  2, 1, 3 };

  public void SetMatrix (Matrix matrix, bool centerCell=false, FilterMode filterMode=FilterMode.Point)
  {
```

```
//generating preview texture
if (texture == null || texture.width != matrix.rect.size.x || texture.height != matrix.rect.size.z)
 texture = new Texture2D(matrix.rect.size.x, matrix.rect.size.z, TextureFormat.RFloat, false, true);
texture.filterMode = filterMode;

matrix.ExportTextureRaw(texture);

if (mesh==null) mesh = new Mesh();
mesh.Clear();

if (centerCell)
{
 Vector2 pixelSize = new Vector2(1f/matrix.rect.size.x, 1f/matrix.rect.size.z);

 mesh.vertices = verts;

 Vector2[] muvs = uvs.Copy();
 for (int i=0; i<muvs.Length; i++)
  muvs[i] -= muvs[i]*pixelSize - pixelSize/2;
 mesh.uv = muvs;

 mesh.triangles = tris;
}
else
{
 mesh.vertices = verts;
```

```
  mesh.uv = uvs;
  mesh.triangles = tris;
 }

 //material
 if (material==null) material = new Material(Shader.Find("Hidden/MapMagic/TexturePreview"));
 material.SetTexture("_MainTex", texture);
 material.SetInt("_Margins", 0);
}



public void SetOffsetSize (Vector2D worldOffset, Vector2D worldSize)
{
 tfm = Matrix4x4.TRS((Vector3)worldOffset, Quaternion.identity, (Vector3)worldSize);
}



public void SetMatrixWorld (MatrixWorld matrix, bool centerCell=false, FilterMode filterMode=FilterMode.P
{
 SetMatrix(matrix, centerCell:centerCell, filterMode:filterMode);
 SetOffsetSize((Vector2D)matrix.worldPos, (Vector2D)matrix.worldSize);
}



public void Draw () => Draw(colorize:false, relief:false, min:0, max:1, parent:null);
public void Draw (
```

```
 bool colorize=false, bool relief=false,
 float min=0, float max=1,
 Transform parent=null)
{
 if (material==null || mesh==null) return;

 material.SetFloat("_Colorize", colorize ? 1 : 0);
 material.SetFloat("_Relief", relief ? 1 : 0);
 material.SetFloat("_MinValue", min);
 material.SetFloat("_MaxValue", max);

 material.SetPass(0);
 Graphics.DrawMeshNow(mesh, tfm);
}


public static void DrawNow (Matrix matrix, Vector2D worldOffset, Vector2D worldSize)
{
 MatrixTextureGizmo gizmo = new MatrixTextureGizmo();
 gizmo.SetMatrix(matrix);
 gizmo.SetOffsetSize(worldOffset, worldSize);
 gizmo.Draw();
}

public Color Color { get{return Color.white;} set{} }
```

```
}



public class MatrixHeightGizmo
{
 private Mesh mesh;
 //private Vector3[] vertices;
 //private Vector2[] uv;
 //private int[] tris;
 private Material material;
 private Matrix4x4 tfm;

 public enum ZMode { Occluded, Overlay, Both };



 public void SetMatrix (Matrix matrix, bool faceted=false)// bool centerCell=true)
 //centerCell: offsetting half-pixel to make verts be placed at grid cells centers
 {
  (Vector3[] verts, Vector2[] uvs, int[] tris) = MakePlane(matrix.rect.size.x-1);
   //MakePlane takes resolution as a number of planes, not vertices

  for (int x=0; x<matrix.rect.size.x; x++)
   for (int z=0; z<matrix.rect.size.z; z++)
   {
    int pos = z*matrix.rect.size.x + x;
    verts[pos].y = matrix.arr[pos];
```

```
    }



   /*if (centerCell)
   {
    OffsetScale(verts,
     new Vector2D(1f/matrix.rect.size.x * 0.5f, 1f/matrix.rect.size.z * 0.5f),
     new Vector2D((float)(matrix.rect.size.x-1)/matrix.rect.size.x, (float)(matrix.rect.size.z-1)/matrix.rect.size.z
   }*/

   if (faceted)
    SplitFaceted(ref verts, ref tris);

   //mesh
   if (mesh == null) { mesh = new Mesh(); mesh.MarkDynamic(); }

   mesh.vertices = verts;
//   mesh.uv = uv;
   mesh.triangles = tris;
   mesh.RecalculateNormals();

   //material
   if (material==null)
   {
    material = new Material(Shader.Find("Standard"));
    material.SetColor("_Color", Color.gray);
```

```
 }
}



public void SetMatrixWorld (MatrixWorld matrix, bool faceted=false)//, bool centerCell=true)
{
 SetMatrix(matrix, faceted);//, centerCell);
 SetOffsetSize(matrix.worldPos, matrix.worldSize);
}



private static (Vector3[] verts, Vector2[] uvs, int[] tris) MakePlane (int resolution)
/// Fills verts,tris,uv with a plane data within 0-1 coordinate
{
 float step = 1f / resolution;

 Vector3[] verts = new Vector3[(resolution+1)*(resolution+1)];
 Vector2[] uv = new Vector2[verts.Length];
 int[] tris = new int[resolution*resolution*2*3];

 int vertCounter = 0;
 int triCounter = 0;
 for (float x=0; x<1.00001f; x+=step) //including max
  for (float z=0; z<1.00001f; z+=step)
 {
  verts[vertCounter] = new Vector3(z, 0, x);
```

```
  uv[vertCounter] = new Vector2(z, x);

  if (x>0.00001f && z>0.00001f)
  {
   tris[triCounter] = vertCounter-(resolution+1);  tris[triCounter+1] = vertCounter-resolution-2; tris[triCounte
   tris[triCounter+3] = vertCounter-1;     tris[triCounter+4] = vertCounter;    tris[triCounter+5] = vertCounter-
   triCounter += 6;
  }

  vertCounter++;
 }

 return (verts, uv, tris);
}


private void OffsetScale (Vector3[] verts, Vector2D offset, Vector2D scale)
{
 for (int i=0; i<verts.Length; i++)
  verts[i] = verts[i]*scale + offset;
}


private void SplitFaceted (ref Vector3[] verts, ref int[] tris)
/// tris count won't change, marking it as ref for company
{
```

```
 Vector3[] newVerts = new Vector3[(tris.Length/6)*4];


 int vertCounter = 0;
 for (int t=0; t<tris.Length; t+=6)
 {
  newVerts[vertCounter] = verts[tris[t]];
  newVerts[vertCounter+1] = verts[tris[t+1]];
  newVerts[vertCounter+2] = verts[tris[t+2]];
  newVerts[vertCounter+3] = verts[tris[t+4]];

  tris[t] = vertCounter;  tris[t+1] = vertCounter+1;  tris[t+2] = vertCounter+2;
  tris[t+3] = vertCounter+2; tris[t+4] = vertCounter+3;  tris[t+5] = vertCounter;

  vertCounter+=4;
 }

 verts = newVerts;
}



public void SetOffsetSize (Vector3 worldOffset, Vector3 worldSize)
{
 tfm = Matrix4x4.TRS(worldOffset, Quaternion.identity, worldSize);
}
```

```
  public void Draw (
   Material material=null,
   Transform parent=null)
  {
   Material currMat = material ?? this.material;
   currMat.SetPass(0);
   Graphics.DrawMeshNow(mesh, tfm);
  }


  public static void DrawNow (Matrix matrix, Vector3 worldOffset, Vector3 worldSize)
  {
   MatrixHeightGizmo gizmo = new MatrixHeightGizmo();
   gizmo.SetMatrix(matrix);
   gizmo.SetOffsetSize(worldOffset, worldSize);
   gizmo.Draw();
  }
 }
}
```

```
ï»¿using UnityEngine;
using System.Collections;
using System;
using System.IO;

using Den.Tools;

namespace Den.Tools.Matrices
{
 public class MatrixObject : MonoBehaviour
 /// Same as MatrixAsset, but saved in scene instead of assets
 /// Plus preview gizmos
 /// Mostly for test purpose
 {
  [NonSerialized] public Matrix matrix = new Matrix( new CoordRect(0,0,1,1) );
  public MatrixWorld MatrixWorld => new MatrixWorld(matrix, (Vector3)worldPosition, new Vector3(worldSi

  public Vector2D worldPosition;
  public Vector2D worldSize = new Vector2D(1000, 1000);
  public float worldHeight = 250;

  [NonSerialized] public Texture2D preview;

  public MatrixAsset.Source source;

  //values to reload
```

```
public string rawPath;
public Texture2D textureSource;
public MatrixAsset.Channel channelSource;
public int newRes = 128;
public Coord newOffset;

//gizmos
public enum DisplayGizmo { None, Texture, Height, FacetedHeight }
public DisplayGizmo displayGizmo;
public bool centerCell = true;
public FilterMode filterMode;
[NonSerialized] public MatrixHeightGizmo heightGizmo;
[NonSerialized] public MatrixTextureGizmo textureGizmo;



public void RefreshPreview (int size=128)
{
 if (matrix != null)
 {
  Matrix previewMatrix = matrix;// new Matrix( new CoordRect(0,0,size,size) );
  //MatrixOps.Resize(matrix, previewMatrix);
  preview = new Texture2D(previewMatrix.rect.size.x, previewMatrix.rect.size.z);
  previewMatrix.ExportTexture(preview, -1);
 }
 else preview = TextureExtensions.ColorTexture(2,2,Color.black);
}
```

```
public void RefreshGizmos ()
{
 switch (displayGizmo)
 {
  case DisplayGizmo.Texture:
   if (textureGizmo == null) textureGizmo = new MatrixTextureGizmo();
   textureGizmo.SetMatrix(matrix, centerCell:centerCell, filterMode:filterMode);
   break;

  case DisplayGizmo.Height:
   if (heightGizmo == null) heightGizmo = new MatrixHeightGizmo();
   heightGizmo.SetMatrix(matrix);
   break;

  case DisplayGizmo.FacetedHeight:
   if (heightGizmo == null) heightGizmo = new MatrixHeightGizmo();
   heightGizmo.SetMatrix(matrix, faceted:true);
   break;
 }
}


public void Reload ()
{
```

```
  switch (source)
  {
   case MatrixAsset.Source.Raw:
    if (rawPath != null)
     MatrixAsset.ImportRaw(ref matrix, rawPath);
    break;

   case MatrixAsset.Source.Texture:
    if (textureSource != null)
     MatrixAsset.ImportTexture(ref matrix, textureSource, channelSource);
    break;

   case MatrixAsset.Source.New:
    if (matrix == null  ||  matrix.rect.size.x !=newRes  ||  matrix.rect.size.z != newRes)
     matrix = new Matrix( new CoordRect(0,0,newRes,newRes) );
    else matrix.Fill(0);
    matrix.rect.offset = newOffset;
    break;
  }

  RefreshPreview();
  RefreshGizmos();
 }


}
```

```
}
```

```
ï»¿using System;
/// Matrix Stripe Operations

using System.Runtime.InteropServices;
using UnityEngine;

[assembly: System.Runtime.CompilerServices.InternalsVisibleTo("Tests")]

namespace Den.Tools.Matrices {
 public static class MatrixOps
 /// All the operations that work with neighbor pixels
 /// The ones that require Stripes
 {
  [Serializable, StructLayout (LayoutKind.Sequential)] //to pass to native
  public class Stripe
  {
   public int length; //in case array is longer (for re-use)
   public float[] arr;

   public Stripe (float[] arr) { this.arr = arr; length = arr.Length; }
   public Stripe (int length) { this.length = length; arr = new float[length]; }
   public Stripe (Stripe stripe) { this.arr = new float[stripe.arr.Length]; Array.Copy(stripe.arr, arr, stripe.arr.Le

   public void Expand (int newCount) { length = newCount; Array.Resize(ref arr, length); }
   public void Fill (float val) { for (int i=0; i<arr.Length; i++) arr[i] = val; }
```

```
 public static void Copy (Stripe src, Stripe dst) { Array.Copy(src.arr, dst.arr, src.length); dst.length = src.le
 public static void Swap (Stripe s1, Stripe s2) { float[] t=s1.arr; s1.arr=s2.arr; s2.arr=t; }
}


#region Stripe Readings/Writings

 public static void ReadLine (Stripe stripe, Matrix matrix, int x, int z)
 {
  int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
  for (int ix=0; ix<stripe.length; ix++)
   stripe.arr[ix] = matrix.arr[start+ix];
  //Array.Copy(this.arr, start, stripe.arr, 0, stripe.length);
 }

 public static void WriteLine (Stripe stripe, Matrix matrix, int x, int z)
 {
  int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
  for (int ix=0; ix<stripe.length; ix++)
   matrix.arr[start+ix] = stripe.arr[ix];
  //Array.Copy(stripe.arr, 0, this.arr, start, stripe.length);
 }

 public static void MaxLine (Stripe stripe, Matrix matrix, int x, int z)
 {
  int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
```

```
 for (int ix=0; ix<stripe.length; ix++)
 {
  float matrixVal = matrix.arr[start+ix];
  float lineVal = stripe.arr[ix];
  if (lineVal>matrixVal) matrix.arr[start+ix] = lineVal;
 }
}

public static void MinLine (Stripe stripe, Matrix matrix, int x, int z)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
 for (int ix=0; ix<stripe.length; ix++)
 {
  float matrixVal = matrix.arr[start+ix];
  float lineVal = stripe.arr[ix];
  if (lineVal<matrixVal) matrix.arr[start+ix] = lineVal;
 }
}

public static void AddLine (Stripe stripe, Matrix matrix, int x, int z, float opacity = 1)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
 for (int ix=0; ix<stripe.length; ix++)
  matrix.arr[start+ix] += stripe.arr[ix]*opacity;
}
```

```
public static void ReadRow (Stripe stripe, Matrix matrix, int x, int z)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
 for (int iz=0; iz<stripe.length; iz++)
  stripe.arr[iz] = matrix.arr[start + iz*matrix.rect.size.x];
}

public static void WriteRow (Stripe stripe, Matrix matrix, int x, int z)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
 for (int iz=0; iz<stripe.length; iz++)
  matrix.arr[start + iz*matrix.rect.size.x] = stripe.arr[iz];
}

public static void WriteRow (Stripe stripe, Matrix matrix, int x, int z, float[] mask)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
 for (int iz=0; iz<stripe.length; iz++)
  matrix.arr[start + iz*matrix.rect.size.x] = stripe.arr[iz] * mask[iz];
}

public static void MaxRow (Stripe stripe, Matrix matrix, int x, int z)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
 for (int iz=0; iz<stripe.length; iz++)
 {
```

```
  float matrixVal = matrix.arr[start+iz*matrix.rect.size.x];
  float lineVal = stripe.arr[iz];
  if (lineVal>matrixVal) matrix.arr[start+iz*matrix.rect.size.x] = lineVal;
 }
}

public static void MinRow (Stripe stripe, Matrix matrix, int x, int z)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
 for (int iz=0; iz<stripe.length; iz++)
 {
  float matrixVal = matrix.arr[start+iz*matrix.rect.size.x];
  float lineVal = stripe.arr[iz];
  if (lineVal<matrixVal) matrix.arr[start+iz*matrix.rect.size.x] = lineVal;
 }
}

public static void AddRow (Stripe stripe, Matrix matrix, int x, int z, float opacity = 1)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
 for (int iz=0; iz<stripe.length; iz++)
  matrix.arr[start + iz*matrix.rect.size.x] += stripe.arr[iz] * opacity;
}

public static void OverlayRow (Stripe stripe, Matrix matrix, int x, int z, float opacity = 1)
//Uses stripe values with range -1 - 1
```

```
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
 for (int iz=0; iz<stripe.length; iz++)
 {
  float valA = matrix.arr[start + iz*matrix.rect.size.x];
  float valB = stripe.arr[iz];

  valB *= opacity;

  matrix.arr[start + iz*matrix.rect.size.x] = 2*valA*valB + valA + valB;

  //for range 0-1 just in case
  //if (a > 0.5f) b = 1 - 2*(1-a)*(1-b);
  //else b = 2*a*b;
 }
}

public static void MixRow (Matrix dst, Matrix matrix, Matrix matrixMask, Stripe stripe, Stripe stripeMask, i
/// Commutative operation, doesn't matter if row applied to mask or vice versa
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
 for (int iz=0; iz<stripe.length; iz++)
 {
  int pos = start + iz*matrix.rect.size.x;
  float sum = matrixMask.arr[pos] + stripeMask.arr[iz];
```

```
  dst.arr[pos] = sum>0 ?
   (matrix.arr[pos]*matrixMask.arr[pos] + stripe.arr[iz]*stripeMask.arr[iz]) / sum :
   matrix.arr[pos] + stripe.arr[iz];
 }
}

public static void MaxRow (Matrix dst, Matrix matrix, Matrix matrixMask, Stripe stripe, Stripe stripeMask,
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;
 for (int iz=0; iz<stripe.length; iz++)
 {
  int pos = start + iz*matrix.rect.size.x;
  dst.arr[pos] = matrixMask.arr[pos] > stripeMask.arr[iz] ?
   matrix.arr[pos] :
   stripe.arr[iz];
 }
}

public static void ReadDiagonal (Stripe stripe, Matrix matrix, int x, int z, float stepX = 1, float stepZ = 1)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;

 //clamping stripe length to matrix borders
 float xLength = matrix.rect.size.x + matrix.rect.size.z;
 if (stepX > 0.001f) xLength = (matrix.rect.offset.x+matrix.rect.size.x - x) / stepX;
 if (stepX < -0.001f) xLength =  (x - matrix.rect.offset.x) / (-stepX);
```

```
 float zLength = matrix.rect.size.x + matrix.rect.size.z;
 if (stepZ > 0.001f) zLength = (matrix.rect.offset.z+matrix.rect.size.z - z) / stepZ;
 if (stepZ < -0.001f) zLength = (z - matrix.rect.offset.z) / (-stepZ);

 stripe.length = (int)((xLength<zLength ? xLength : zLength) - 0.5f); //if no 0.5 will exceed the matrix borde

 //reading
 for (int i=0; i<stripe.length; i++)
 {
  int posX = (int)(i*stepX + 0.5f); // if (posX+x<matrix.rect.offset.x  || posX+x>=matrix.rect.offset.x+matrix.r
  int posZ = (int)(i*stepZ + 0.5f); // if (posZ+z<matrix.rect.offset.z  || posZ+z>=matrix.rect.offset.z+matrix.re

  stripe.arr[i] = matrix.arr[start + posX + posZ*matrix.rect.size.x];
 }
}

public static void WriteDiagonal (Stripe stripe, Matrix matrix, int x, int z, float stepX = 1, float stepZ = 1)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;

 for (int i=0; i<stripe.length; i++)
 {
  int posX = (int)(i*stepX + 0.5f);  //if (posX+x<matrix.rect.offset.x  || posX+x>=matrix.rect.offset.x+matrix.r
  int posZ = (int)(i*stepZ + 0.5f);  //if (posZ+z<matrix.rect.offset.z  || posZ+z>=matrix.rect.offset.z+matrix.re
```

```
  matrix.arr[start + posX + posZ*matrix.rect.size.x] = stripe.arr[i];
 }
}

public static void MaxDiagonal (Stripe stripe, Matrix matrix, int x, int z, float stepX = 1, float stepZ = 1)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  x;

 for (int i=0; i<stripe.length; i++)
 {
  int posX = (int)(i*stepX + 0.5f);  //if (posX+x<matrix.rect.offset.x  || posX+x>=matrix.rect.offset.x+matrix.r
  int posZ = (int)(i*stepZ + 0.5f); // if (posZ+z<matrix.rect.offset.z  || posZ+z>=matrix.rect.offset.z+matrix.re

  float matrixVal = matrix.arr[start + posX + posZ*matrix.rect.size.x];
  float lineVal = stripe.arr[i];

  if (lineVal>matrixVal)
   matrix.arr[start + posX + posZ*matrix.rect.size.x] = lineVal;
 }
}

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
[DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Read
public static extern void ReadInclined(Stripe stripe, Matrix matrix, Vector2 start, Vector2 step);
#else
```

```
public static void ReadInclined (Stripe stripe, Matrix matrix, Vector2 start, Vector2 step)
{
 for (int i=0; i<stripe.length; i++)
 {
  float x = start.x + step.x*i;
  float z = start.y + step.y*i;

  if (x<0) x--;  int ix = (int)(float)(x + 0.5f);
  if (z<0) z--;  int iz = (int)(float)(z + 0.5f);

  if (ix<matrix.rect.offset.x || ix>=matrix.rect.offset.x+matrix.rect.size.x ||
   iz<matrix.rect.offset.z || iz>=matrix.rect.offset.z+matrix.rect.size.z )
    stripe.arr[i] = -0.00001f; //-Mathf.Epsilon;
  else
  {
   int pos = (iz-matrix.rect.offset.z)*matrix.rect.size.x + ix - matrix.rect.offset.x;
   stripe.arr[i] = matrix.arr[pos];
  }
 }
}
#endif

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
[DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Write
public static extern void WriteInclined(Stripe stripe, Matrix matrix, Vector2 start, Vector2 step);
#else
```

```
public static void WriteInclined (Stripe stripe, Matrix matrix, Vector2 start, Vector2 step)
{
 for (int i=0; i<stripe.length; i++)
 {
  float x = start.x + step.x*i;
  float z = start.y + step.y*i;

  if (x<0) x--;  int ix = (int)(float)(x + 0.5f);
  if (z<0) z--;  int iz = (int)(float)(z + 0.5f);

  if (ix<matrix.rect.offset.x || ix>=matrix.rect.offset.x+matrix.rect.size.x ||
   iz<matrix.rect.offset.z || iz>=matrix.rect.offset.z+matrix.rect.size.z )
    continue;
  else
  {
   int pos = (iz-matrix.rect.offset.z)*matrix.rect.size.x + ix - matrix.rect.offset.x;
   matrix.arr[pos] = stripe.arr[i];
  }
 }
}
#endif

public static void ReadStrip (Stripe stripe, Matrix matrix, Vector2 start, Vector2 end)
{
 Vector2 delta = end-start;
```

```
 Vector2 direction = delta.normalized;

 //making any of the step components equal to 1
 Vector2 posDir = new Vector2 (
  (direction.x>0 ? direction.x : -direction.x),
  (direction.y>0 ? direction.y : -direction.y) );
 float max = posDir.x>posDir.y ? posDir.x : posDir.y;
 Vector2 step = direction / max;

 Vector2 absDelta = new Vector2(delta.x>0 ? delta.x : -delta.x, delta.y>0 ? delta.y : -delta.y);
 int numSteps = (int)(absDelta.x>absDelta.y ? absDelta.x : absDelta.y) + 1;
 stripe.length = numSteps < stripe.length ? numSteps : stripe.length;

 ReadInclined(stripe, matrix, start, step);
}


public static void WriteStrip (Stripe stripe, Matrix matrix, Vector2 start, Vector2 end)
{
 Vector2 delta = end-start;
 Vector2 direction = delta.normalized;

 //making any of the step components equal to 1
 Vector2 posDir = new Vector2 (
  (direction.x>0 ? direction.x : -direction.x),
  (direction.y>0 ? direction.y : -direction.y) );
```

```
 float max = posDir.x>posDir.y ? posDir.x : posDir.y;
 Vector2 step = direction / max;

 int numSteps = (int)(delta.x>delta.y ? delta.x : delta.y) + 1;
 stripe.length = numSteps < stripe.length ? numSteps : stripe.length;

 WriteInclined(stripe, matrix, start, step);
}


public static void ReadSquare (Stripe stripe, Matrix matrix, Coord center, int radius)
/// Same as circular, but in form of square. 4 lines one-by-one. Useful for blurs and spreads
{
 int side = radius*2 + 1;
 stripe.length = side*4;

 //resetting line
 for (int i=0; i<side*4; i++)
  stripe.arr[i] = - Mathf.Epsilon;

 Coord min = center-radius;
 Coord max = center+radius;

 Coord rectMin = matrix.rect.offset;
 Coord rectMax = matrix.rect.offset + matrix.rect.size;
```

```
int start = (min.z-matrix.rect.offset.z-1)*matrix.rect.size.x - matrix.rect.offset.x  +  min.x; //matrix[min.x, mi
if (min.z-1 >= rectMin.z  &&  min.z-1 < rectMax.z)
 for (int x=0; x<side; x++)
  if (x+min.x >= rectMin.x  &&  x+min.x < rectMax.x)
   stripe.arr[x] = matrix.arr[start+x];

start = (min.z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  max.x; //matrix[max.x, min.z]
if (max.x >= rectMin.x  &&  max.x < rectMax.x)
 for (int z=0; z<side; z++)
  if (z+min.z >= rectMin.z  &&  z+min.z < rectMax.z)
   stripe.arr[z+side] = matrix.arr[start + z*matrix.rect.size.x];

start = (max.z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  max.x; //matrix[max.x-1, ma
if (max.z >= rectMin.z  &&  max.z < rectMax.z)
for (int x=0; x<side; x++)
 if (max.x-x >= rectMin.x  &&  max.x-x < rectMax.x)
  stripe.arr[x+side*2] = matrix.arr[start -x];

start = (max.z-matrix.rect.offset.z-1)*matrix.rect.size.x - matrix.rect.offset.x  +  min.x; //matrix[min.x-1, ma
if (min.x-1 >= rectMin.x  &&  min.x < rectMax.x)
for (int z=0; z<side; z++)
 if (max.z-1-z >= rectMin.z  &&  max.z-1-z < rectMax.z)
  stripe.arr[z+side*3] = matrix.arr[start - z*matrix.rect.size.x]; //matrix[min.x, max.z-1-z]

//closing
stripe.arr[0] = stripe.arr[stripe.length-1];
```

```
}



public static void WriteSquare (Stripe stripe, Matrix matrix, Coord center, int radius)
/// Same as circular, but in form of square. 4 lines one-by-one. Useful for blurs and spreads
/// Line length should be radius*8 + 4 corners
{
 int side = radius*2 + 1;
 stripe.length = side*4;

 Coord min = center-radius;
 Coord max = center+radius;

 Coord rectMin = matrix.rect.offset;
 Coord rectMax = matrix.rect.offset + matrix.rect.size;

 int start = (min.z-matrix.rect.offset.z-1)*matrix.rect.size.x - matrix.rect.offset.x  +  min.x; //matrix[min.x, mi
 if (min.z-1 >= rectMin.z  &&  min.z-1 < rectMax.z)
  for (int x=0; x<side; x++)
   if (x+min.x >= rectMin.x  &&  x+min.x < rectMax.x)
    matrix.arr[start+x] = stripe.arr[x];

 start = (min.z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  max.x; //matrix[max.x, min.z]
 if (max.x >= rectMin.x  &&  max.x < rectMax.x)
  for (int z=0; z<side; z++)
   if (z+min.z >= rectMin.z  &&  z+min.z < rectMax.z)
```

```
    matrix.arr[start + z*matrix.rect.size.x] = stripe.arr[z+side];


 start = (max.z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  max.x; //matrix[max.x-1, ma
 if (max.z >= rectMin.z  &&  max.z < rectMax.z)
 for (int x=0; x<side; x++)
  if (max.x-x >= rectMin.x  &&  max.x-x < rectMax.x)
   matrix.arr[start -x] = stripe.arr[x+side*2];


 start = (max.z-matrix.rect.offset.z-1)*matrix.rect.size.x - matrix.rect.offset.x  +  min.x; //matrix[min.x-1, ma
 if (min.x >= rectMin.x  &&  min.x < rectMax.x)
 for (int z=0; z<side; z++)
  if (max.z-1-z >= rectMin.z  &&  max.z-1-z < rectMax.z)
   matrix.arr[start - z*matrix.rect.size.x] = stripe.arr[z+side*3]; //matrix[min.x, max.z-1-z]
}



public static void FlipVertical (Matrix src, Matrix dst)
{
 Coord min = src.rect.Min; Coord max = src.rect.Max;
 Stripe stripe = new Stripe(src.rect.size.x);
 for (int z=min.z; z<max.z; z++)
 {
  ReadLine(stripe, src, src.rect.offset.x, z);
  WriteLine(stripe, dst, dst.rect.offset.x, max.z-(z-min.z)-1);
 }
}
```

```
#endregion



#region Free Resize


 public static void Resize (Matrix src, CoordRect dstRect)
 {
  Matrix dst = new Matrix(dstRect);
  Resize(src, dst);
  src.rect=dst.rect; src.arr=dst.arr;
 }



 public static void Resize (Matrix src, Matrix dst, Matrix tmp=null)
 /// Writing dst with new sized src contents
 /// Using cubic filtration for upsizing and linear for downsizing
 /// Using temp array for downsizing both dimensions (tmp should be src size at least vertically)
 {
  Coord savedDstOffset = dst.rect.offset;
  dst.rect.offset = src.rect.offset;


  Coord srcSize = src.rect.size;
  Coord dstSize = dst.rect.size;


  Stripe srcStripe = new Stripe( Mathf.Max(srcSize.x, srcSize.z, dst.rect.size.x, dst.rect.size.z) ); //just the
```

```
Stripe dstStripe = new Stripe( srcStripe.length );

Coord min = src.rect.Min; Coord srcMax = src.rect.Max; Coord dstMax = dst.rect.Max;

//shrinking both dimensions (using temporary array)
if (dstSize.x < srcSize.x  &&  dstSize.z < srcSize.z)
{
 if (tmp==null)
  tmp = new Matrix(src.rect.offset.x, src.rect.offset.z, dstSize.x, srcSize.z);

 DownsizeHorizontally(src, tmp, srcStripe, dstStripe);
 DownsizeVertically(tmp, dst, srcStripe, dstStripe);
}

//expanding both dimensions
else if (dstSize.x > srcSize.x  &&  dstSize.z > srcSize.z)
{
 Coord tmpSize = new Coord(dstSize.x, srcSize.z); //dst horizontally, src vertically
 tmp = new Matrix(src.rect.offset, tmpSize, dst.arr); //using dst array!

 UpsizeHorizontally(src, tmp, srcStripe, dstStripe);
 UpsizeVertically(tmp, dst, srcStripe, dstStripe); //note tmp and dst arrays are same
}

//expanding horizontally, shrinking vertically
else if (dstSize.x > srcSize.x  &&  dstSize.z < srcSize.z)
```

```
{
 Coord tmpSize = new Coord(srcSize.x, dstSize.z);
 tmp = new Matrix(src.rect.offset, tmpSize, dst.arr); //using dst array!

 DownsizeVertically(src, tmp, srcStripe, dstStripe);
 UpsizeHorizontally(tmp, dst, srcStripe, dstStripe); //note tmp and dst arrays are same
}

//expanding vertically, shrinking horizontally
else if (dstSize.x < srcSize.x  &&  dstSize.z > srcSize.z)
{
 Coord tmpSize = new Coord(dstSize.x, srcSize.z);
 tmp = new Matrix(src.rect.offset, tmpSize, dst.arr); //using dst array!

 DownsizeHorizontally(src, tmp, srcStripe, dstStripe);
 UpsizeVertically(tmp, dst, srcStripe, dstStripe); //note tmp and dst arrays are same
}

else if (dstSize.x > srcSize.x  &&  dstSize.z == srcSize.z)
 UpsizeHorizontally(src, dst, srcStripe, dstStripe);

else if (dstSize.x < srcSize.x  &&  dstSize.z == srcSize.z)
 DownsizeHorizontally(src, dst, srcStripe, dstStripe);

else if (dstSize.x == srcSize.x  &&  dstSize.z > srcSize.z)
 UpsizeVertically(src, dst, srcStripe, dstStripe);
```

```
 else if (dstSize.x == srcSize.x  &&  dstSize.z < srcSize.z)
  DownsizeVertically(src, dst, srcStripe, dstStripe);

 else
  dst.Fill(src); //if size match just copying array

 dst.rect.offset = savedDstOffset;
}



public static void Upsize (Matrix src, Matrix dst)
/// Expanding both dimensions
{
 if (dst.rect.size.x < src.rect.size.x  ||  dst.rect.size.z < src.rect.size.z)
  throw new Exception($"Couldn't upsize: src {src.rect.ToString()} is less than dst {dst.rect.ToString()}");

 Stripe srcStripe = new Stripe( Mathf.Max(src.rect.size.x, src.rect.size.z, dst.rect.size.x, dst.rect.size.z) );
 Stripe dstStripe = new Stripe( srcStripe.length );

 Coord tmpSize = new Coord(dst.rect.size.x, src.rect.size.z); //dst horizontally, src vertically
 Matrix tmp = new Matrix(src.rect.offset, tmpSize, dst.arr); //using dst array!

 UpsizeHorizontally(src, tmp, srcStripe, dstStripe, 0, src.rect.size.x);
 UpsizeVertically(tmp, dst, srcStripe, dstStripe, 0, src.rect.size.z); //note tmp and dst arrays are same
```

```
}


public static void Upsize (Matrix src, Vector2D srcOffset, Vector2D srcSize, Matrix dst)
/// Custom float-rect for src matrix to begin/end reading on half-pixel
/// For Import node rescale
{
 if (dst.rect.size.x < src.rect.size.x  ||  dst.rect.size.z < src.rect.size.z)
  throw new Exception($"Couldn't upsize: src {src.rect.ToString()} is less than dst {dst.rect.ToString()}");

 Stripe srcStripe = new Stripe( Mathf.Max(src.rect.size.x, src.rect.size.z, dst.rect.size.x, dst.rect.size.z) );
 Stripe dstStripe = new Stripe( srcStripe.length );

 Coord tmpSize = new Coord(dst.rect.size.x, src.rect.size.z); //dst horizontally, src vertically
 Matrix tmp = new Matrix(src.rect.offset, tmpSize, dst.arr); //using dst array!

 UpsizeHorizontally(src, tmp, srcStripe, dstStripe, srcOffset.x-src.rect.offset.x, srcSize.x);
 UpsizeVertically(tmp, dst, srcStripe, dstStripe, srcOffset.z-src.rect.offset.z, srcSize.z); //note tmp and dst
}


public static void Downsize (Matrix src, Matrix dst, Matrix tmp=null)
// Shrinking both dimensions (using temporary array)
{
 if (dst.rect.size.x > src.rect.size.x  ||  dst.rect.size.z > src.rect.size.z)
  throw new Exception($"Couldn't downsize: src {src.rect.ToString()} is more than dst {dst.rect.ToString()}
```

```
 Stripe srcStripe = new Stripe( Mathf.Max(src.rect.size.x, src.rect.size.z, dst.rect.size.x, dst.rect.size.z) );
 Stripe dstStripe = new Stripe( srcStripe.length );

 if (tmp==null)
  tmp = new Matrix(src.rect.offset.x, src.rect.offset.z, dst.rect.size.x, src.rect.size.z);

 DownsizeHorizontally(src, tmp, srcStripe, dstStripe);
 DownsizeVertically(tmp, dst, srcStripe, dstStripe);
}


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Dow
 private static extern void DownsizeHorizontally (Matrix src, Matrix dst, Stripe srcStripe, Stripe dstStripe);
#else
 private static void DownsizeHorizontally (Matrix src, Matrix dst, Stripe srcStripe, Stripe dstStripe)
 {
  srcStripe.length = src.rect.size.x;
  dstStripe.length = dst.rect.size.x;

  for (int z=0; z<src.rect.size.z; z++) //should match dst.rect.size.z
  {
   ReadLine(srcStripe, src, src.rect.offset.x, z+src.rect.offset.z);
   ResampleStripeLinear(srcStripe, dstStripe);
   //ResampleStripeCubic(srcStripe, dstStripe);
```

```
   WriteLine(dstStripe, dst, dst.rect.offset.x, z+dst.rect.offset.z);
  }
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Dow
 private static extern void DownsizeVertically (Matrix src, Matrix dst, Stripe srcStripe, Stripe dstStripe);
#else
 private static void DownsizeVertically (Matrix src, Matrix dst, Stripe srcStripe, Stripe dstStripe)
 {
  srcStripe.length = src.rect.size.z;
  dstStripe.length = dst.rect.size.z;

  for (int x=0; x<dst.rect.size.x; x++) //should match src.rect.size.x
  {
   ReadRow(srcStripe, src, x+src.rect.offset.x, src.rect.offset.z);
   ResampleStripeLinear(srcStripe, dstStripe);
   //ResampleStripeCubic(srcStripe, dstStripe);
   WriteRow(dstStripe, dst, x+dst.rect.offset.x, dst.rect.offset.z);
  }
 }
#endif
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Ups
 private static extern void UpsizeHorizontally (Matrix src, Matrix dst, Stripe srcStripe, Stripe dstStripe, floa
#else
 private static void UpsizeHorizontally (Matrix src, Matrix dst, Stripe srcStripe, Stripe dstStripe, float srcOf
 /// srcOffset is zero-based (pixels offset from start of the stripe)
 {
  srcStripe.length = src.rect.size.x;
  dstStripe.length = dst.rect.size.x;

  for (int z=src.rect.size.z-1; z>=0; z--) //from end to start since in some cases we would be using the sam
  {
   ReadLine(srcStripe, src, src.rect.offset.x, z+src.rect.offset.z);
   ResampleStripeCubic(srcStripe, dstStripe, srcOffset, srcLength);
   WriteLine(dstStripe, dst, dst.rect.offset.x, z+dst.rect.offset.z);
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Ups
 private static extern void UpsizeVertically (Matrix src, Matrix dst, Stripe srcStripe, Stripe dstStripe, float s
#else
 private static void UpsizeVertically (Matrix src, Matrix dst, Stripe srcStripe, Stripe dstStripe, float srcOffse
 /// srcOffset is zero-based (pixels offset from start of the stripe)
```

```
 {
  srcStripe.length = src.rect.size.z;
  dstStripe.length = dst.rect.size.z;

  for (int x=src.rect.size.x-1; x>=0; x--)
  {
   ReadRow(srcStripe, src, x+src.rect.offset.x, src.rect.offset.z);
   ResampleStripeCubic(srcStripe, dstStripe, srcOffset, srcLength);
   WriteRow(dstStripe, dst, x+dst.rect.offset.x, dst.rect.offset.z);
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Res
 private static extern void ResampleStripeCubic (Stripe src, Stripe dst, float srcOffset=0, float srcLength=
 /// Scales the stripe filling dst with interpolated values. Cubic for upscale
#else
 private static void ResampleStripeCubic (Stripe src, Stripe dst, float srcOffset=0, float srcLength=0)
 /// Scales the stripe filling dst with interpolated values. Cubic for upscale
 {
  if (srcLength<1) srcLength = src.length;

  for (int dstX=0; dstX<dst.length; dstX++)
  {
```

```
float srcX = (float)dstX * srcLength / dst.length  + srcOffset;


int px = (int)srcX; if (px<0) px=0;

int nx = px+1;  if (nx>src.length-1) nx = src.length-1;

int ppx = px-1;  if (ppx<0) ppx = 0;

int nnx = nx+1;  if (nnx>src.length-1) nnx = src.length-1;

int pppx = ppx-1; if (pppx<0) pppx = 0;

int nnnx = nnx+1; if (nnnx>src.length-1) nnnx = src.length-1;



float vp = src.arr[px]; float vpp = src.arr[ppx]; float vppp = src.arr[pppx];

float vn = src.arr[nx]; float vnn = src.arr[nnx]; float vnnn = src.arr[nnnx];


float p = srcX-px;

float ip = 1f-p;


float dpp = vpp - (vppp+vp-vpp*2) * 0.25f;

float dp = vp - (vpp+vn-vp*2) * 0.25f;

float dn = vn - (vnn+vp-vn*2) * 0.25f;

float dnn = vnn - (vnnn+vn-vnn*2) * 0.25f;


float tp = (dn-dpp)*0.5f;

float tn = (dp-dnn)*0.5f;


float l = vp*ip + vn*p; //linear filtration
```

```
   float cp =
    (vp + tp*p) * ip +
    l * p;

   float cn = l * ip +
    (vn + tn*ip) * p;

   dst.arr[dstX] =  cp*ip + cn*p;

   //dst.arr[dstX] = vp + 0.5f * p * (vn - vpp + p*(2.0f*vpp - 5.0f*vp + 4.0f*vn - vnn + p*(3.0f*(vp - vn) + vnn
    //standard quadratic filtration (here for test purpose)

   if (dst.arr[dstX] > 1) dst.arr[dstX] = 1;
   if (dst.arr[dstX] < 0) dst.arr[dstX] = 0;
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Res
 private static extern void ResampleStripeQuadratic (Stripe src, Stripe dst);
 /// Scales the stripe filling dst with interpolated values. Faster than cubic, but can result in "grid" look bec
#else
 private static void ResampleStripeQuadratic (Stripe src, Stripe dst)
 /// Scales the stripe filling dst with interpolated values. Faster than cubic, but can result in "grid" look bec
```

```
{
 for (int dstX=0; dstX<dst.length; dstX++)
 {
  float srcX = (1.0f*dstX / dst.length) * src.length;

  int px = (int)srcX; if (px<0) px=0;
  int nx = px+1;  if (nx>src.length-1) nx = src.length-1;
  int ppx = px-1;  if (ppx<0) ppx = 0;
  int nnx = nx+1;  if (nnx>src.length-1) nnx = src.length-1;

  float vp = src.arr[px]; float vpp = src.arr[ppx];
  float vn = src.arr[nx]; float vnn = src.arr[nnx];

  float p = srcX-px;
  float ip = 1-p;

  float tp = (vn-vpp)*0.5f;
  float tn = (vp-vnn)*0.5f;

  float l = vp*ip + vn*p; //linear filtration

  float cp =
   (vp + tp*p) * ip +
   l * p;

  float cn = l * ip +
```

```
    (vn + tn*ip) * p;

   dst.arr[dstX] =  cp*ip + cn*p;

   //dst.arr[dstX] = vp + 0.5f * p * (vn - vpp + p*(2.0f*vpp - 5.0f*vp + 4.0f*vn - vnn + p*(3.0f*(vp - vn) + vnn
    //standard quadratic filtration (here for test purpose)

   if (dst.arr[dstX] > 1) dst.arr[dstX] = 1;
   if (dst.arr[dstX] < 0) dst.arr[dstX] = 0;
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Res
 private static extern void ResampleStripeLinear (Stripe src, Stripe dst);
 /// Scales the line filling dst with ALL src interpolated values. Linear for downscale
#else
 private static void ResampleStripeLinear (Stripe src, Stripe dst)
 /// Scales the line filling dst with interpolated values. Linear for downscale
 {
  float dstToSrc = (float)src.length / (float)dst.length;

  for (int dstX=0; dstX<dst.length; dstX++)
  {
```

```
  int srcStartX = (int)((dstX-1) * dstToSrc - 1);
  if (srcStartX < 0) srcStartX = 0;

  int srcEndX = (int)((dstX+1) * dstToSrc + 2);
  if (srcEndX >= src.length) srcEndX = src.length-1;

  float val = 0;
  float sum = 0;

  for (int srcX = srcStartX; srcX <= srcEndX; srcX++)
  {
   float refX = srcX / dstToSrc; //converting back to dst gauge

   float percent = (refX - dstX) * dstToSrc;
   if (percent > 1) percent = 1;
   if (percent < -1) percent = -1;
   if (percent < 0) percent = -percent;
   percent = 1-percent;

   val += src.arr[srcX] * percent;
   sum += percent;
  }

  dst.arr[dstX] = sum!=0 ? val/sum : 0;
 }
}
```

```
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Res
 public static extern void ResizeNearestNeighbor (Matrix src, Matrix dst);
#else
 public static void ResizeNearestNeighbor (Matrix src, Matrix dst)
 /// For upsizing and downsizing. Does not actually use Stripe, so might be moved to Matrix
 {
  float dstToSrcX = (float)src.rect.size.x / (float)dst.rect.size.x;
  float dstToSrcZ = (float)src.rect.size.z / (float)dst.rect.size.z;

  for (int x=0; x<dst.rect.size.x; x++)
   for (int z=0; z<dst.rect.size.z; z++)
   {
    int dstPos = z*dst.rect.size.x + x;

    int srcX = (int)((x+0.5f)*dstToSrcX);
    int srcZ = (int)((z+0.5f)*dstToSrcZ);
    int srcPos = srcZ*src.rect.size.x + srcX;

    dst.arr[dstPos] = src.arr[srcPos];
   }
 }
#endif
```

```
#endregion



#region ResizeFast


 public static void ResizeFast (Matrix src, Matrix dst)
 /// When dstRect is 2x, 4x, etc larger or smaller
 /// Using ResampleStripeDownFast instead of ResampleStripeCubic/Linear
 {
  if (dst.rect.size.x == src.rect.size.x*2 || dst.rect.size.z == src.rect.size.z*2)
   UpscaleFast(src, dst);


  else if (dst.rect.size.x*2 == src.rect.size.x || dst.rect.size.z*2 == src.rect.size.z)
   DownscaleFast(src, dst);


  else
   throw new Exception("Matrix ResizeFast: rect size mismatch: src:" + src.rect.size.ToString() + " dst:" + d
 }



 public static Matrix[] GenerateMips (Matrix src, int count=-1, float blur=0, float multiply=1)
 //
 {
  if (count < 0)
   count = (int)Mathf.Log(src.rect.size.x, 2) - 1;
```

```
CoordRect rect = src.rect;
Matrix[] mips = new Matrix[count];
Matrix mat = src;
Matrix mip;

Matrix tmp = new Matrix( new CoordRect(rect.offset.x, rect.offset.z, rect.size.x/2, rect.size.z) );

Stripe srcStripe = new Stripe( Mathf.Max(rect.size.x, rect.size.z) ); //just the longest dimension
Stripe dstStripe = new Stripe( srcStripe.length );

for (int m=0; m<count; m++)
{
 mip = new Matrix( new CoordRect(mat.rect.offset.x, mat.rect.offset.z, mat.rect.size.x/2, mat.rect.size.z/2

 DownscaleFast(mat, mip, tmp, srcStripe, dstStripe);

 if (blur > 0.0001f)
  GaussianBlur(mip, blur);

 if (multiply != 1)
  mip.Multiply(multiply);

 mips[m] = mip;
 mat = mip;
}
```

```
 return mips;
}


public static Matrix TestMips (Matrix[] mips)
/// Blends all mipmaps in one matrix
{
 int width = 0;
 for (int m=0; m<mips.Length; m++)
  width += mips[m].rect.size.x;

 Matrix matrix = new Matrix( new CoordRect(0,0,width, mips[0].rect.size.x) );
 matrix.Fill(-1);

 width = 0;
 for (int m=0; m<mips.Length; m++)
 {
  Matrix mip = mips[m];
  CoordRect mipRect = mip.rect;

  for (int x=0; x<mipRect.size.x; x++)
   for (int z=0; z<mipRect.size.z; z++)
   {
    int mipPos = z*mipRect.size.x + x;
    int matPos = z*matrix.rect.size.x + x + width;
```

```
    matrix.arr[matPos] = mip.arr[mipPos];
   }

  width += mipRect.size.x;
 }

 return matrix;
}


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Ups
 public static extern void UpscaleFast (Matrix src, Matrix dst);
#else
 public static void UpscaleFast (Matrix src, Matrix dst)
 /// Only for cases when dst rect size is exactly twice bigger than src
 /// Does not require temp matrix
 {
  if (dst.rect.size.x != src.rect.size.x*2 || dst.rect.size.z != src.rect.size.z*2)
   throw new Exception("Matrix Upscale Fast: rect size mismatch: src:" + src.rect.size.ToString() + " dst:"

  Stripe srcStripe = new Stripe( dst.rect.size.Maximal );
  Stripe dstStripe = new Stripe( srcStripe.length );

  srcStripe.length = src.rect.size.x;
```

```
  dstStripe.length = dst.rect.size.x;
  for (int z=0; z<src.rect.size.z; z++)
  {
   ReadLine(srcStripe, src, src.rect.offset.x, z + src.rect.offset.z);
   ResampleStripeUpFast(srcStripe, dstStripe);
   WriteLine(dstStripe, dst, dst.rect.offset.x, z + dst.rect.offset.z);
  }

  srcStripe.length = src.rect.size.z;
  dstStripe.length = dst.rect.size.z;
  for (int x=dst.rect.size.x-1; x>=0; x--) //inverse order to re-use the same array
  {
   ReadRow(srcStripe, dst, x + dst.rect.offset.x, dst.rect.offset.z);
   ResampleStripeUpFast(srcStripe, dstStripe);
   WriteRow(dstStripe, dst, x + dst.rect.offset.x, dst.rect.offset.z);
  }
 }
#endif


public static void DownscaleFast (Matrix src, Matrix dst) =>  DownscaleFast (src, dst, null, null, null);

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

 public static void DownscaleFast (Matrix src, Matrix dst, Matrix tmp=null, Stripe srcStripe=null, Stripe dst
 {
```

```
  if (tmp == null)
   tmp = new Matrix( new CoordRect(src.rect.offset.x, src.rect.offset.z, src.rect.size.x/2, src.rect.size.z) );

  if (srcStripe==null) srcStripe = new Stripe( src.rect.size.Maximal );
  if (dstStripe==null) dstStripe = new Stripe( srcStripe.length );

  DownscaleFastTmp(src, dst, tmp, srcStripe, dstStripe);
 }

 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Dow
 private static extern void DownscaleFastTmp (Matrix src, Matrix dst, Matrix tmp, Stripe srcStripe, Stripe
#else
 public static void DownscaleFast (Matrix src, Matrix dst, Matrix tmp=null, Stripe srcStripe=null, Stripe dst
 /// Only for cases when dst rect size is exactly twice smaller than src
 {
  //if (dst.rect.size.x*2 != src.rect.size.x || dst.rect.size.z*2 != src.rect.size.z)
  // throw new Exception("Matrix Downscale Fast: rect size mismatch: src:" + src.rect.size.ToString() + " d

  if (tmp == null)
   tmp = new Matrix( new CoordRect(src.rect.offset.x, src.rect.offset.z, src.rect.size.x/2, src.rect.size.z) );

  if (srcStripe==null) srcStripe = new Stripe( src.rect.size.Maximal );
  if (dstStripe==null) dstStripe = new Stripe( srcStripe.length );

  srcStripe.length = src.rect.size.x;
  dstStripe.length = dst.rect.size.x;
```

```
  for (int z=0; z<src.rect.size.z; z++)
  {
   ReadLine(srcStripe, src, src.rect.offset.x, z + src.rect.offset.z);
   ResampleStripeDownFast(srcStripe, dstStripe);
   WriteLine(dstStripe, tmp, tmp.rect.offset.x, z + tmp.rect.offset.z);
  }

  srcStripe.length = src.rect.size.z;
  dstStripe.length = dst.rect.size.z;
  for (int x=0; x<dst.rect.size.x; x++)
  {
   ReadRow(srcStripe, tmp, x + tmp.rect.offset.x, tmp.rect.offset.z);
   ResampleStripeDownFast(srcStripe, dstStripe);
   WriteRow(dstStripe, dst, x + dst.rect.offset.x, dst.rect.offset.z);
  }
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Res
 private static extern void ResampleStripeDownFast (Stripe src, Stripe dst);
#else
 private static void ResampleStripeDownFast (Stripe src, Stripe dst)
 /// Like linear, works faster, but requires dst.size = src.size/2
 {
```

```
  for (int dstX=1; dstX<dst.length-1; dstX++)
  {
   //dst.arr[dstX] = src.arr[dstX*2]*0.5f + src.arr[dstX*2-1]*0.25f + src.arr[dstX*2+1]*0.25f;
   dst.arr[dstX] = src.arr[dstX*2]*0.5f + src.arr[dstX*2+1]*0.5f;
    //surprizingly this way it generates no offset
  }

  dst.arr[0] = src.arr[0]*0.75f + src.arr[1]*0.25f;
  dst.arr[dst.length-1] = src.arr[src.length-1]*0.75f + src.arr[src.length-2]*0.25f;
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Res
 private static extern void ResampleStripeUpFast (Stripe src, Stripe dst);
#else
 private static void ResampleStripeUpFast (Stripe src, Stripe dst)
 /// Like cubic, works faster, but requires dst.size = src.size*2
 {
  for (int srcX=0; srcX<src.length-1; srcX++)
  {
   dst.arr[srcX*2] = src.arr[srcX]*0.5f + src.arr[srcX+1]*0.5f;
   dst.arr[srcX*2+1] = src.arr[srcX+1];  //placing original value to pixel+1, this will downscale lossless with
  }
```

```
   if (src.length*2 < dst.length) //duplicating the last pixel if dst is 1-pixel larger than src*2
    dst.arr[dst.length-1] = src.arr[src.length-1];

   dst.arr[src.length*2-1] = src.arr[src.length-1]; //*0.5f + src.arr[src.length-2]*0.5f;
   dst.arr[src.length*2-2] = src.arr[src.length-1]*0.5f + src.arr[src.length-2]*0.5f;
  }
 #endif

#endregion



#region GaussianBlur

 public static void GaussianBlur (Matrix matrix, float blur)
  { GaussianBlur(matrix, matrix, blur); }

 #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
  [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Gau
  public static extern void GaussianBlur (Matrix src, Matrix dst, float blur);
 #else
  public static void GaussianBlur (Matrix src, Matrix dst, float blur)
  /// Blur value is the number of iterations
  {
   CoordRect rect = src.rect;
   Coord min = rect.Min; Coord max = rect.Max;
```

```
  Stripe stripe = new Stripe( Mathf.Max(rect.size.x, rect.size.z) );

  stripe.length = rect.size.x;
  for (int z=min.z; z<max.z; z++)
  {
   ReadLine(stripe, src, rect.offset.x, z);
   BlurStripe(stripe, blur);
   WriteLine(stripe, dst, rect.offset.x, z);
  }

  stripe.length = rect.size.z;
  for (int x=min.x; x<max.x; x++)
  {
   ReadRow(stripe, dst, x, rect.offset.z);
   BlurStripe(stripe, blur);
   WriteRow(stripe, dst, x, rect.offset.z);
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Blur
 public static extern void BlurStripe (Stripe src, float blur);
#else
 internal static void BlurStripe (Stripe src, float blur)
```

```
 {
  int iterations = (int)blur;

  //iteration blur
  for (int i=0; i<iterations; i++)
   BlurIteration(src, 1);

  //last iteration - percentage
  float percent = blur - iterations;
  if (percent > 0.0001f)
   BlurIteration(src, percent);
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Blur
 public static extern void BlurIteration (Stripe src, float blur);
#else
 internal static void BlurIteration (Stripe src, float blur)
 {
  float qBlur = blur*0.25f;
  float hBlur = 1 - qBlur*2;

  float vp = src.arr[0];
  float vx = src.arr[1];
```

```
  float vn;

  for (int x=1; x<src.length-1; x++)
  {
   vn = src.arr[x+1];

   src.arr[x] = vp*qBlur + vx*hBlur + vn*qBlur;

   vp = vx;
   vx = vn;
  }

  src.arr[0] = src.arr[0]*hBlur + src.arr[1]*qBlur*2;
  src.arr[src.length-1] = src.arr[src.length-1]*hBlur + src.arr[src.length-2]*qBlur*2;
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Blur
 public static extern void BlurIteration (Stripe src);
#else
 internal static void BlurIteration (Stripe src)
 /// Surprisingly worse performance rather than read array. I expected less bounds check, but...
 {
  float vp = src.arr[0];
```

```
  float vx = src.arr[1];
  float vn;

  for (int x=1; x<src.length-1; x++)
  {
   vn = src.arr[x+1];

   src.arr[x] = (vp + vx*2 + vn - 0.000000000000001f)*0.25f;

   vp = vx;
   vx = vn;
  }

  src.arr[0] = (src.arr[1] + src.arr[0] + src.arr[0] -0.000000000000001f) / 3;
  src.arr[src.length-1] = (src.arr[src.length-2] + src.arr[src.length-1] + src.arr[src.length-1] -0.00000000000
 }
#endif


/*internal static unsafe void BlurIteration_Unsafe (Stripe src)
/// Same performance as the standard one. Expected array bounds check, but...
{
 float vp = src.arr[0];
 float vx = src.arr[1];
 float vn;
```

```
 fixed (float* arrPtr = src.arr)
  for (int x=1; x<src.arr.Length-1; x++)
 {
  vn = arrPtr[x+1];


  arrPtr[x] = (vp + vx*2 + vn - 0.000000000000001f)*0.25f;


  vp = vx;
  vx = vn;
 }


 src.arr[0] = (src.arr[1] + src.arr[0] + src.arr[0]) / 3;
 src.arr[src.arr.Length-1] = (src.arr[src.arr.Length-2] + src.arr[src.arr.Length-1] + src.arr[src.arr.Length-1])
}*/



/*public static void BlurIteration_Wrong (Stripe src) // Reads the changed array
{
 for (int x=1; x<src.arr.Length-1; x++)
 {
  src.arr[x] = (
   src.arr[x-1] +
   src.arr[x]*2 +
   src.arr[x+1] -0.000000000000001f)*0.25f;
 }
```

```
 src.arr[0] = (src.arr[1] + src.arr[0] + src.arr[0]) / 3;
 src.arr[src.arr.Length-1] = (src.arr[src.arr.Length-2] + src.arr[src.arr.Length-1] + src.arr[src.arr.Length-1])
}*/

#endregion



#region Downsample Blur

 public static void DownsampleBlur (Matrix matrix, int downsample, float blur)
 { DownsampleBlur(matrix, matrix, downsample, blur); }

 #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Dow
 public static extern void DownsampleBlur(Matrix src, Matrix dst, int downsample, float blur);
 #else
 public static void DownsampleBlur (Matrix src, Matrix dst, int downsample, float blur)
 /// Blurs matrix by downscaling each line and then upscaling it back
 /// Downsample 1 means no re-scaling (however resample stripe cubic produces artifacts in this case, so
 {
  int downsamplePot = (int)Mathf.Pow(2,downsample-1);
  CoordRect rect = src.rect;
  Coord min = rect.Min; Coord max = rect.Max;

  Stripe hiStripe = new Stripe( Mathf.Max(rect.size.x, rect.size.z) );
  Stripe loStripe = new Stripe( hiStripe.length / downsample);
```

```
hiStripe.length = rect.size.x;
loStripe.length = hiStripe.length / downsample;
for (int z=min.z; z<max.z; z++)
{
 ReadLine(hiStripe, src, rect.offset.x, z);

 ResampleStripeLinear(hiStripe, loStripe);
 BlurStripe(loStripe, blur);
 ResampleStripeCubic(loStripe, hiStripe);

 WriteLine(hiStripe, dst, rect.offset.x, z);
}

hiStripe.length = rect.size.z;
loStripe.length = hiStripe.length / downsample;
for (int x=min.x; x<max.x; x++)
{
 ReadRow(hiStripe, dst, x, rect.offset.z);

 ResampleStripeLinear(hiStripe, loStripe);
 BlurStripe(loStripe, blur);
 ResampleStripeCubic(loStripe, hiStripe);

 WriteRow(hiStripe, dst, x, rect.offset.z);
}
```

```
 }
 #endif

#endregion



#region Overblur Mipped

 public static void OverblurMipped (Matrix matrix, float downsample, float escalate=4, float blur=1)
  { OverblurMipped(matrix, matrix, downsample, escalate, blur); }

 public static void OverblurMipped (Matrix src, Matrix dst, float downsample, float escalate=4, float blur=1)
 /// Takes the zero-centered (-1 - +1) map and overblurs it maintaining the middle (0) value
 /// Useful to blur cavity maps
 /// Escalate increases the contrast of each next mipmap
 {
  int iDownsample = (int)downsample + 1; //not ceiltoint (in case of 3.0 ceil returns 3)!
  Matrix[] mips = GenerateMips(src, iDownsample, blur:1, multiply:2f);

  if (dst!=src) dst.Fill(src);
  ArrayTools.Insert(ref mips, 0, dst);

  //lowering last mip contrast to gradually switch between downsamples
  float lastMipFactor = downsample - (int)downsample;
  mips[mips.Length-1].Multiply(lastMipFactor);
```

```
 Matrix tmp = new Matrix( new CoordRect(src.rect.offset.x, src.rect.offset.z, src.rect.size.x, src.rect.size.z
 Stripe srcStripe = new Stripe( Mathf.Max(src.rect.size.x, src.rect.size.z) ); //just the longest dimension
 Stripe dstStripe = new Stripe( srcStripe.length );

 for (int i=mips.Length-2; i>=0; i--)
  OverblurMippedIteration(mips[i+1], mips[i], tmp, srcStripe, dstStripe, escalate);
}

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Ove
 public static extern void OverblurMippedIteration (Matrix mip, Matrix mat, Matrix tmp, Stripe srcStripe, St
#else
 private static void OverblurMippedIteration (Matrix mip, Matrix mat, Matrix tmp, Stripe srcStripe, Stripe ds
 {
  srcStripe.length = mip.rect.size.x;
  dstStripe.length = mat.rect.size.x;
  for (int z=mip.rect.offset.z; z<mip.rect.offset.z+mip.rect.size.z; z++)
  {
   ReadLine(srcStripe, mip, mip.rect.offset.x, z);
   ResampleStripeUpFast(srcStripe, dstStripe);
   WriteLine(dstStripe, tmp, mat.rect.offset.x, z);
  }

  srcStripe.length = mip.rect.size.z;
  dstStripe.length = mat.rect.size.z;
  for (int x=mat.rect.offset.x; x<mat.rect.offset.x+mat.rect.size.x; x++)
```

```
  {
   ReadRow(srcStripe, tmp, x, mat.rect.offset.z);
   ResampleStripeUpFast(srcStripe, dstStripe);
   OverlayRow(dstStripe, mat, x, mat.rect.offset.z, escalate);
  }
 }
#endif

#endregion



#region Normals/Delta

 //normals formula:
 //new Vector3(
 // (prevXHeight-nextXHeight)*height,
 // pixelSize*2,
 // (prevZHeight-nextZHeight)*height).
 //normalized;

 public static (Matrix r, Matrix g, Matrix b) NormalsSet (Matrix src, float pixelSize, float height)
 /// Generates 3 channels normal map from heightmap
 /// This one creates new matrices
 {
  Matrix r = new Matrix(src.rect);
  Matrix g = new Matrix(src.rect);
```

```
 Matrix b = new Matrix(src.rect);

 NormalsSet(src, r, g, b, pixelSize, height);

 return (r,g,b);
}

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Nor
 public static extern void NormalsSet (Matrix src, Matrix normX, Matrix normZ, Matrix normBlue, float pixe
 /// Generates full set of normal map
 /// This one uses prepared matrices
#else
 public static void NormalsSet (Matrix src, Matrix normX, Matrix normZ, Matrix normBlue, float pixelSize,
 /// Generates full set of normal map
 /// This one uses prepared matrices
 {
  Coord min = src.rect.Min; Coord max = src.rect.Max;

  Stripe stripe = new Stripe(src.rect.size.Maximal);

  stripe.length = src.rect.size.x;
  for (int z=min.z; z<max.z; z++)
  {
   ReadLine(stripe, src, src.rect.offset.x, z);
```

```
  NormalsStripe(stripe, height);
  WriteLine(stripe, normX, src.rect.offset.x, z);
 }

 stripe.length = src.rect.size.z;
 for (int x=min.x; x<max.x; x++)
 {
  ReadRow(stripe, src, x, src.rect.offset.z);
  NormalsStripe(stripe, height);
  WriteRow(stripe, normZ, x, src.rect.offset.z);
 }

 NormalizeSet(normX, normZ, normBlue, pixelSize);
}

private static void NormalizeSet (Matrix normX, Matrix normZ, Matrix normBlue, float pixelSize)
{
 Coord min = normX.rect.Min; Coord max = normX.rect.Max;

 for (int i=0; i<normX.count; i++)
 {
  float nx = normX.arr[i];
  float nz = normZ.arr[i];
  float blue = pixelSize*2;

  float length = Mathf.Sqrt(nx*nx + blue*blue + nz*nz);
```

```
  normX.arr[i] = (nx/length + 1)/2;
  normZ.arr[i] = (nz/length + 1)/2;
  //if (normBlue != null) //can't observe it in c++
    normBlue.arr[i] = blue/length;
 }
}

private static void NormalsStripe (Stripe stripe, float height)
{
 if (stripe.length < 3) return;

 float prevHeight = stripe.arr[0];
 float currHeight = stripe.arr[1];
 for (int x=1; x<stripe.length-2; x++)
 {
  float nextHeight = stripe.arr[x+1];

  stripe.arr[x] = (prevHeight-nextHeight)*height;

  prevHeight = currHeight;
  currHeight = nextHeight;
 }

 stripe.arr[0] = stripe.arr[1];
 stripe.arr[stripe.length-1] = stripe.arr[stripe.length-2];
```

```
 }
#endif



public static void NormalsDir (Matrix src, Matrix dst, Vector3 dir, float pixelSize, float height, float intensity
/// Generates dot product (lightened) image from heightmap
/// Generates 3-channel normals and applies light to them
{
 wrapping /= 2; //maximum wrapping is reached at level 0.5

 Matrix normX = new Matrix(src.rect);
 Matrix normZ = new Matrix(src.rect);
 Matrix normBlue = dst;
 normBlue.Fill(0);

 NormalsSet(src, normX, normZ, normBlue, pixelSize, height);

 SetToDir(normX, normZ, normBlue, dst, dir.x, dir.y, dir.z, intensity, wrapping);
}

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_SetT
 public static extern void  SetToDir (Matrix normX, Matrix normZ, Matrix normBlue, Matrix dst, float dirX, f
 /// Dot-lighting 3-channel normals set depending on direction
#else
 private static void SetToDir (Matrix normX, Matrix normZ, Matrix normBlue, Matrix dst, float dirX, float dir
```

```
 /// Dot-lighting 3-channel normals set depending on direction
{
 Vector3 dir = new Vector3(dirX, dirY, dirZ);
 Vector3 normal = new Vector3();

 for (int i=0; i<dst.count; i++)
 {
  normal.x = normX.arr[i]*2 - 1;
  normal.y = normBlue.arr[i]*2 - 1;
  normal.z = normZ.arr[i]*2 - 1;

  //float val = Vector3.Dot(dir, normal);
  float val = dir.x*normal.x + dir.y*normal.y + dir.z*normal.z; //to use the same as c++ code

  val = val*(1-wrapping) + wrapping/2;
  val *= intensity;

  if (val < 0) val = 0;
  if (val > 1) val = 1;
  dst.arr[i] = val;
 }
}
#endif

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
```

```
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Delt
 public static extern void Delta(Matrix src, Matrix dst);
#else
 public static void Delta (Matrix src, Matrix dst)
 /// Finds the maximum delta (both pos and neg) for the neighbor pixel
 {
  Coord min = src.rect.Min; Coord max = src.rect.Max;

  Stripe stripe = new Stripe( Mathf.Max(src.rect.size.x, src.rect.size.z) );

  stripe.length = src.rect.size.x;
  for (int z=min.z; z<max.z; z++)
  {
   ReadLine(stripe, src, src.rect.offset.x, z);
   DeltaStripe(stripe);
   WriteLine(stripe, dst, src.rect.offset.x, z);
  }

  stripe.length = src.rect.size.z;
  for (int x=min.x; x<max.x; x++)
  {
   ReadRow(stripe, src, x, src.rect.offset.z);
   DeltaStripe(stripe);
   MaxRow(stripe, dst, x, src.rect.offset.z);
  }
 }
```

```
 private static void DeltaStripe (Stripe stripe)
 {
  float prev = stripe.arr[0];
  float curr = stripe.arr[1];

  for (int x=1; x<stripe.length-1; x++)
  {
   //float prev = arr[x-1];
   //float curr = arr[x];
   float next = stripe.arr[x+1];

   float prevDelta = prev-curr; if (prevDelta < 0) prevDelta = -prevDelta;
   float nextDelta = next-curr; if (nextDelta < 0) nextDelta = -nextDelta;
   float delta = prevDelta>nextDelta? prevDelta : nextDelta;

   stripe.arr[x] = delta;

   prev = curr;
   curr = next;
  }

  stripe.arr[0] = stripe.arr[1];
  stripe.arr[stripe.length-1] = stripe.arr[stripe.length-2];
 }
#endif
```

```
#endregion



#region Select

 #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
  [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Silh
  public static extern void Silhouette (Matrix src, Matrix dst, float level, bool antialiasing=true);
 #else
  public static void Silhouette (Matrix src, Matrix dst, float level, bool antialiasing=true)
  /// Makes the pixels crossing level 1, others - 0
  {
   CoordRect rect = src.rect;
   Coord min = rect.Min; Coord max = rect.Max;

   Stripe stripe = new Stripe( Mathf.Max(src.rect.size.x, src.rect.size.z) );

   stripe.length = rect.size.x;
   for (int z=min.z; z<max.z; z++)
   {
    ReadLine(stripe, src, rect.offset.x, z);
    SilhouetteStripe(stripe, level, antialiasing);
    WriteLine(stripe, dst, rect.offset.x, z);
   }
```

```
  stripe.length = rect.size.z;
  for (int x=min.x; x<max.x; x++)
  {
   ReadRow(stripe, src, x, rect.offset.z);
   SilhouetteStripe(stripe, level, antialiasing);
   MaxRow(stripe, dst, x, rect.offset.z);
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Silh
 public static extern void SilhouetteStripe (Stripe stripe, float level, bool antiAliasing);
#else
 private static void SilhouetteStripe (Stripe stripe, float level, bool antiAliasing)
 {
  float prevVal = stripe.arr[0];
  for (int x=0; x<stripe.length-1; x++)
  {
   float nextVal = stripe.arr[x+1];

   float prevSelect = 0;
   float nextSelect = 0;
   float percent = -1;
```

```
if (prevVal < level  &&  nextVal >= level) //growing
{
 float prevAbs = -(prevVal - level);
 float nextAbs = nextVal - level;
 percent = prevAbs / (prevAbs+nextAbs);
}

if (prevVal > level  &&  nextVal <= level) //lowering
{
 float prevAbs = prevVal - level;
 float nextAbs = -(nextVal - level);
 percent = prevAbs / (prevAbs+nextAbs);
}

if (percent >= 0) //if growing or lowering
{
 //prevSelect = 1-percent;
 //nextSelect = percent;
 //making closest pixel always 1, the other one 0-1

 if (percent < 0.5f)
 {
  prevSelect = 1;
  if (antiAliasing) nextSelect = percent*2;
 }
```

```
     else
     {
      nextSelect = 1;
      if (antiAliasing) prevSelect = (1-percent)*2;
     }
    }

    if (prevSelect > stripe.arr[x]) stripe.arr[x] = prevSelect;
    stripe.arr[x+1] = nextSelect;

    prevVal = nextVal;
   }


  }
 #endif

#endregion


#region Cavity

 #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
  [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Cav
  public static extern void Cavity(Matrix src, Matrix dst);
```

```
#else
 public static void Cavity (Matrix src, Matrix dst)
 /// Creates a map of curvatures - white for concave pixels and black for convex
 {
  CoordRect rect = src.rect;
  Coord min = rect.Min; Coord max = rect.Max;

  Stripe stripe = new Stripe( Mathf.Max(rect.size.x, rect.size.z) );

  stripe.length = rect.size.x;
  for (int z=min.z; z<max.z; z++)
  {
   ReadLine(stripe, src, rect.offset.x, z);
   CavityStripe(stripe);
   AddLine(stripe, dst, rect.offset.x, z, 0.5f);
  }

  stripe.length = rect.size.z;
  for (int x=min.x; x<max.x; x++)
  {
   ReadRow(stripe, src, x, rect.offset.z);
   CavityStripe(stripe);
   AddRow(stripe, dst, x, rect.offset.z, 0.5f); //apply row additively (with mid-point 0.5)
  }
 }
```

```
 internal static void CavityStripe (Stripe stripe)
 {
  float prev = stripe.arr[0];
  float curr = stripe.arr[1];

  for (int x=1; x<stripe.length-1; x++)
  {
   float next = stripe.arr[x+1];

   //float val = curr - (next + prev)/2;
   //float sign = val>0 ? 1 : -1;
   //val = (val*val*sign)*intensity*1000;
   //val = (val+1) / 2;
   float avg = (next + prev)/2;
   float val = avg - curr;

   stripe.arr[x] = val;

   prev = curr;
   curr = next;
  }
  stripe.arr[0] = stripe.arr[1];
  stripe.arr[stripe.length-1] = stripe.arr[stripe.length-2];
 }
#endif
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Ove
 public static extern void OverSpread(Matrix src, Matrix dst, float multiply);
#else
 public static void OverSpread (Matrix src, Matrix dst, float multiply)
 /// Doesn't work yet
 {
  CoordRect rect = src.rect;
  Coord min = rect.Min; Coord max = rect.Max;

  Stripe stripe = new Stripe( Mathf.Max(rect.size.x, rect.size.z) );

  Matrix pos = new Matrix(src);
  for (int i=0; i<pos.arr.Length; i++)
   pos.arr[i] = (pos.arr[i] - 0.5f)*2;

  for (int i=0; i<5; i++)
  {
   stripe.length = rect.size.x;
   for (int z=min.z; z<max.z; z++)
   {
    ReadLine(stripe, pos, rect.offset.x, z);
    SpreadMultiply(stripe, multiply);
    WriteLine(stripe, pos, rect.offset.x, z);
   }
```

```
  stripe.length = rect.size.z;
  for (int x=min.x; x<max.x; x++)
  {
   ReadRow(stripe, pos, x, rect.offset.z);
   SpreadMultiply(stripe, multiply);
   WriteRow(stripe, pos, x, rect.offset.z); //apply row additively (with mid-point 0.5)
  }
 }

 dst.arr = pos.arr;
}


public static Matrix OverSpread (this Matrix src, float multiply)
{
 Matrix dst = new Matrix(src.rect);
 OverSpread(src, dst, multiply);
 return dst;
}


private static void Invert (Stripe src, Stripe dst)
{
 for (int i=0; i<src.arr.Length; i++)
  dst.arr[i] = -src.arr[i];
```

```
 }
#endif


#endregion


#region Spread

 #if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)  //when native checke
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Spre
 public static extern void SpreadLinear (Matrix src, Matrix dst, float subtract=0.01f, bool diagonals=false,
#else

 public static void SpreadLinear (Matrix src, Matrix dst, float subtract=0.01f, bool diagonals=false, bool qu
 /// Smears all white values all over the matrix
 /// Each new pixel is multiplied with multiply and get subtract subtracted
 /// Will overwrite the gray values with the bigger spread values, so couldn't be used as padding
 {
  if (src==dst)
   throw new Exception("MatrixOps: same matrix is used as src and dst at the same time");

 // if (bulb) subtract *= 2; //bulb is bigger than original, so increasing subtract

  Coord min = src.rect.Min; Coord max = src.rect.Max;
```

```
Stripe stripe = new Stripe(src.rect.size.x + src.rect.size.z); //+ to process diagonals

//spreading two dimensions independently
stripe.length = src.rect.size.x;
for (int z=min.z; z<max.z; z++)
{
 ReadLine(stripe, src, src.rect.offset.x, z);
 SpreadLinearLeft(stripe, subtract, regardSmallerValues:1);
 SpreadLinearRight(stripe, subtract, regardSmallerValues:1);
 MaxLine(stripe, dst, src.rect.offset.x, z);
}

stripe.length = src.rect.size.z;
for (int x=min.x; x<max.x; x++)
{
 ReadRow(stripe, src, x, src.rect.offset.z);
 SpreadLinearLeft(stripe, subtract, regardSmallerValues:1);
 SpreadLinearRight(stripe, subtract, regardSmallerValues:1);
 MaxRow(stripe, dst, x, dst.rect.offset.z);
}

//connecting crosses lines
if (!diagonals)
{
 stripe.length = src.rect.size.x;
 for (int z=min.z; z<max.z; z++)
```

```
 {
  ReadLine(stripe, dst, src.rect.offset.x, z);
  SpreadLinearLeft(stripe, subtract, regardSmallerValues:0);
  SpreadLinearRight(stripe, subtract, regardSmallerValues:0);
  MaxLine(stripe, dst, src.rect.offset.x, z);
 }

 stripe.length = src.rect.size.z;
 for (int x=min.x; x<max.x; x++)
 {
  ReadRow(stripe, dst, x, src.rect.offset.z);
  SpreadLinearLeft(stripe, subtract, regardSmallerValues:0);
  SpreadLinearRight(stripe, subtract, regardSmallerValues:0);
  MaxRow(stripe, dst, x, dst.rect.offset.z);
 }
}

else //diagonals
{
 if (quarters)
 {
  float step = 0.4142f;  // sin(radians(22.5)) / cos(radians(22.5))
  float factor = 1.082387f;  // (1,step).magnitude;

  for (int z=max.z-1; z>=min.z; z-=3)
  {
```

```
 ReadDiagonal(stripe, dst, min.x, z, step, 1);
 SpreadLinearLeft(stripe, subtract*factor, regardSmallerValues:0);
 SpreadLinearRight(stripe, subtract*factor, regardSmallerValues:0);
 MaxDiagonal(stripe, dst, min.x, z, step, 1);
}
for (int x=min.x; x<max.x; x++)
{
 ReadDiagonal(stripe, dst, x, min.z, step, 1);
 SpreadLinearLeft(stripe, subtract*factor, regardSmallerValues:0);
 SpreadLinearRight(stripe, subtract*factor, regardSmallerValues:0);
 MaxDiagonal(stripe, dst, x, min.z, step, 1);
}

for (int x=min.x; x<max.x; x+=3)
{
 ReadDiagonal(stripe, dst, x, min.z, -1, step);
 SpreadLinearLeft(stripe, subtract*factor, regardSmallerValues:0);
 SpreadLinearRight(stripe, subtract*factor, regardSmallerValues:0);
 MaxDiagonal(stripe, dst, x, min.z, -1, step);
}
for (int z=min.z; z<max.z; z++)
{
 ReadDiagonal(stripe, dst, max.x-1, z, -1, step);
 SpreadLinearLeft(stripe, subtract*factor, regardSmallerValues:0);
 SpreadLinearRight(stripe, subtract*factor, regardSmallerValues:0);
 MaxDiagonal(stripe, dst, max.x-1, z, -1, step);
```

```
}

for (int x=max.x-1; x>=min.x; x--)
{
 ReadDiagonal(stripe, dst, x, max.z-1, step, -1);
 SpreadLinearRight(stripe, subtract*factor, regardSmallerValues:0);
 SpreadLinearLeft(stripe, subtract*factor, regardSmallerValues:0);
 MaxDiagonal(stripe, dst, x, max.z-1, step, -1);
}
for (int z=max.z-1; z>=min.z; z-=3)
{
 ReadDiagonal(stripe, dst, min.x, z, step, -1);
 SpreadLinearLeft(stripe, subtract*factor, regardSmallerValues:0);
 SpreadLinearRight(stripe, subtract*factor, regardSmallerValues:0);
 MaxDiagonal(stripe, dst, min.x, z, step, -1);
}

for (int z=min.z; z<max.z; z++)
{
 ReadDiagonal(stripe, dst, max.x-1, z, -1, -step);
 SpreadLinearLeft(stripe, subtract*factor, regardSmallerValues:0);
 SpreadLinearRight(stripe, subtract*factor, regardSmallerValues:0);
 MaxDiagonal(stripe, dst, max.x-1, z, -1, -step);
}
for (int x=max.x-1; x>=min.x; x-=3)
{
```

```
  ReadDiagonal(stripe, dst, x, max.z-1, -1, -step);
  SpreadLinearLeft(stripe, subtract*factor, regardSmallerValues:0);
  SpreadLinearRight(stripe, subtract*factor, regardSmallerValues:0);
  MaxDiagonal(stripe, dst, x, max.z-1, -1, -step);
 }
}


for (int z=max.z-1; z>=min.z; z--)  //quarters leave empty pixels since they apply 1to3, so using diagona
{
 int maxX = z==min.z ? max.x : min.x+1;
 for (int x=min.x; x<maxX; x++)
 {
  ReadDiagonal(stripe, dst, x, z, 1, 1);
  SpreadLinearLeft(stripe, subtract*1.4142f, regardSmallerValues:0);  //sqrt(2)
  SpreadLinearRight(stripe, subtract*1.4142f, regardSmallerValues:0);
  MaxDiagonal(stripe, dst, x, z, 1, 1);
 }
}

for (int x=min.x; x<max.x; x++)
{
 int maxZ = x==max.x-1 ? max.z : min.z+1;
 for (int z=min.z; z<maxZ; z++)
 {
  ReadDiagonal(stripe, dst, x, z, -1, 1);
```

```
   SpreadLinearLeft(stripe, subtract*1.41421f, regardSmallerValues:0);
   SpreadLinearRight(stripe, subtract*1.41421f, regardSmallerValues:0);
   MaxDiagonal(stripe, dst, x, z, -1, 1);
  }
 }
}

//bulb
//simulates pseudo-round behaviour on edges
if (bulb)
{
 //applying bulb function to enter bulb mode
 for (int i=0; i<dst.count; i++)
  dst.arr[i] = (float)(Math.Sqrt(2*dst.arr[i] - dst.arr[i]*dst.arr[i])); //*0.5f + dst.arr[i]*0.5f;

 //spreading two dimensions
 stripe.length = src.rect.size.z;
 for (int x=min.x; x<max.x; x++)
 {
  ReadRow(stripe, dst, x, src.rect.offset.z);
  SpreadLinearLeft(stripe, subtract);
  SpreadLinearRight(stripe, subtract);
  MaxRow(stripe, dst, x, dst.rect.offset.z);
 }
```

```
   stripe.length = src.rect.size.x;
   for (int z=min.z; z<max.z; z++)
   {
    ReadLine(stripe, dst, src.rect.offset.x, z);
    SpreadLinearLeft(stripe, subtract);
    SpreadLinearRight(stripe, subtract);
    MaxLine(stripe, dst, src.rect.offset.x, z);
   }

   //applying bulb inverse and some magic to 'linearize' bulb
   for (int i=0; i<dst.count; i++)
   {
    float val = dst.arr[i];
    val = (float)(1 - Math.Sqrt(1-val*val));
    val = (float)(1 - Math.Sqrt(1-val*val));

    val = (val-0.5f)*2;
    if (val<0) val = 0;

    dst.arr[i] = val;
   }
  }
 }
#endif
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Spr
 public static extern void SpreadLinearRight(Stripe stripe, float subtract = 0.01f, float regardSmallerValue
#else

 private static void SpreadLinearRight (Stripe stripe, float subtract=0.01f, float regardSmallerValues=1)
 {
  if (stripe.length == 0) return; //diagonals

  float prevVal = stripe.arr[0];
  for (int x=1; x<stripe.length; x++)
  {
   float currVal = stripe.arr[x];

   if (prevVal > currVal)
   {
    float newVal = currVal;

    newVal = prevVal - subtract + (currVal/prevVal)*subtract*regardSmallerValues;
    if (newVal<0) newVal =0;

    if (newVal > currVal)
    {
     stripe.arr[x] = newVal;
     currVal = newVal;
    }
```

```
   }

    prevVal = currVal;
  }
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Spr
 public static extern void SpreadLinearLeft(Stripe stripe, float subtract = 0.01f, float regardSmallerValues
#else

 private static void SpreadLinearLeft (Stripe stripe, float subtract=0.01f, float regardSmallerValues=1)
 {
  if (stripe.length < 3) return; //diagonals

  float prevVal = stripe.arr[stripe.length-1];
  for (int x=stripe.length-2; x>=0; x--)
  {
   float currVal = stripe.arr[x];

   if (prevVal > currVal)
   {
    float newVal = currVal;
```

```
    newVal = prevVal - subtract + (currVal/prevVal)*subtract*regardSmallerValues;
    if (newVal<0) newVal =0;

    if (newVal > currVal)
    {
     stripe.arr[x] = newVal;
     currVal = newVal;
    }
   }

   prevVal = currVal;
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Spr
 public static extern void SpreadMultiply(Stripe stripe, float multiply = 1.0f, float regardSmallerValues = 1)
#else

private static void SpreadMultiply (Stripe stripe, float multiply=1f, float regardSmallerValues=1)
{
 SpreadMultiplyRight(stripe, multiply, regardSmallerValues);
 SpreadMultiplyLeft(stripe, multiply, regardSmallerValues);
}
```

```
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Spr
 public static extern void SpreadMultiplyLeft(Stripe stripe, float multiply = 1.0f, float regardSmallerValues
#else

private static void SpreadMultiplyLeft (Stripe stripe, float multiply=1f, float regardSmallerValues=1)
{
 float prevVal = stripe.arr[stripe.length-1];
 for (int x=stripe.length-2; x>=0; x--)
 {
  float currVal = stripe.arr[x];

  if (prevVal > currVal)
  {
   currVal = prevVal*multiply + currVal*(1-multiply)*regardSmallerValues;
   stripe.arr[x] = currVal;
  }

  prevVal = currVal;
 }
}
#endif
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Spr
 public static extern void SpreadMultiplyRight(Stripe stripe, float multiply = 1.0f, float regardSmallerValue
#else

 private static void SpreadMultiplyRight (Stripe stripe, float multiply=1f, float regardSmallerValues=1)
 {
  float prevVal = stripe.arr[0];
  for (int x=1; x<stripe.length-1; x++)
  {
   float currVal = stripe.arr[x];

   if (prevVal > currVal)
   {
    currVal = prevVal*multiply + currVal*(1-multiply)*regardSmallerValues;
    stripe.arr[x] = currVal;
   }

   prevVal = currVal;
  }
 }
#endif

#endregion
```

```
#region Padding

 public static void PaddingMipped (Matrix src, Matrix mask, Matrix dst, int mipsCount=-1, float mipContras
 /// Fills all of the unmasked areas with extended src values
 /// Supports mask AA (note that transparent mask values should be fully filled in src, like the opaque ones
 {
  if (src==dst)
   throw new Exception("MatrixOps: same matrix is used as src and dst at the same time");

  //downscaling
  //pretty similar to GenerateMips
  if (mipsCount < 0)
   mipsCount = (int)Mathf.Log(src.rect.size.x, 2) - 1;

  Matrix[] mips = new Matrix[mipsCount];
  Matrix[] maskMips = new Matrix[mipsCount];

  Matrix mat = src; Matrix matMask = mask;
  Matrix mip; Matrix mipMask;

  CoordRect rect = src.rect;
  Matrix tmp = new Matrix( new CoordRect(rect.offset.x, rect.offset.z, rect.size.x/2, rect.size.z) );
  Matrix tmpMask = new Matrix( new CoordRect(rect.offset.x, rect.offset.z, rect.size.x/2, rect.size.z) );

  for (int i=0; i<mipsCount; i++)
```

```
{
 mip = new Matrix( new CoordRect(mat.rect.offset.x, mat.rect.offset.z, mat.rect.size.x/2, mat.rect.size.z/2
 mipMask = new Matrix(mip.rect);

 DownscaleMaskedFast(mat, matMask, mip, mipMask, tmp, tmpMask);
 mipMask.Multiply(mipContrast);
 mipMask.Clamp01();
 PaddingOnePixel(mip, mipMask, mipOnePxPadding*(1-1f*i/mipsCount));

 mips[i] = mip; maskMips[i] = mipMask;
 mat = mip;  matMask = mipMask;
}

ArrayTools.Insert(ref mips, 0, src);
ArrayTools.Insert(ref maskMips, 0, mask);

//Matrix maskMipsTest = TestMips(maskMips);
//maskMipsTest.ToWindow("Mask Mips");

//Matrix mipsTest = TestMips(mips);
//mipsTest.ToWindow("Mips Before");

//upscaling
for (int m=mips.Length-2; m>=0; m--)
{
 Matrix prevMip = mips[m+1];
```

```
  tmp = m!=0 ? new Matrix(mips[m].rect) : dst; //last iteration mixing to dst
  tmpMask = new Matrix(mips[m].rect);

  GaussianBlur(prevMip, 0.5f);
  GaussianBlur(prevMip, 0.5f);
  MatrixOps.UpscaleMaskedFast(prevMip, maskMips[m+1], tmp, tmpMask);
  tmp.Mix(mips[m], maskMips[m]);

  mips[m] = tmp;
  maskMips[m] = tmpMask;
 }
}

public static void PaddingOnePixel (Matrix matrix, Matrix mask, float intensity=1)
 { PaddingOnePixel(matrix, mask, matrix, mask, intensity); }

#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Pad
 public static extern void PaddingOnePixel(Matrix src, Matrix srcMask, Matrix dst, Matrix dstMask, float in
#else
 public static void PaddingOnePixel (Matrix src, Matrix srcMask, Matrix dst, Matrix dstMask, float intensity
 /// Pads one pixel only in all directions
 /// Supports AA
 {
```

```
Stripe maskStripe = new Stripe(Mathf.Max(src.rect.size.x, src.rect.size.z));
Stripe stripe = new Stripe(maskStripe.length);

Coord min = src.rect.Min; Coord max = src.rect.Max;

maskStripe.length = maskStripe.length =  src.rect.size.x;
for (int z=min.z; z<max.z; z++)
{
 ReadLine(stripe, src, src.rect.offset.x, z);
 ReadLine(maskStripe, srcMask, srcMask.rect.offset.x, z);

 PadStripeOnePixel(stripe, maskStripe, intensity, toLeft:false);
 PadStripeOnePixel(stripe, maskStripe, intensity, toLeft:true);

 WriteLine(stripe, dst, src.rect.offset.x, z);
 WriteLine(maskStripe, dstMask, srcMask.rect.offset.x, z);
}

maskStripe.length =  maskStripe.length = src.rect.size.x;
for (int x=min.x; x<max.x; x++)
{
 ReadRow(stripe, dst, x, src.rect.offset.z);
 ReadRow(maskStripe, dstMask, x, srcMask.rect.offset.z);

 PadStripeOnePixel(stripe, maskStripe, intensity, toLeft:false);
 PadStripeOnePixel(stripe, maskStripe, intensity, toLeft:true);
```

```
   WriteRow(stripe, dst, x, src.rect.offset.z);
   WriteRow(maskStripe, dstMask, x, srcMask.rect.offset.z);
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Pad
 public static extern void PadStripeOnePixel(Stripe stripe, Stripe mask, float intensity = 1, bool toLeft = fa
#else
 private static void PadStripeOnePixel (Stripe stripe, Stripe mask, float intensity=1, bool toLeft=false)
 /// Moves image one pixel left, blending it with the previous one the way it's done in photoshop
 {
  float prev = stripe.arr[ !toLeft ? 0 : stripe.length-1 ];
  float prevMask = mask.arr[ !toLeft ? 0 : stripe.length-1 ];

  for (int x= !toLeft ? 0 : stripe.length-1;
   !toLeft ? x<stripe.length : x>=0;
   x += !toLeft ? +1 : -1)
  {
   float val = stripe.arr[x];
   float maskVal = mask.arr[x];

   float maskSum = maskVal+prevMask; if (maskSum==0) maskSum = 1;
```

```
   float modifiedVal = (val*maskVal + prev*prevMask) /maskSum;
   stripe.arr[x] = val*maskVal +  //should maintain original value with original mask anyways
         modifiedVal*(1-maskVal);

   float newMaskVal = maskVal + prevMask*(1-maskVal);
   mask.arr[x] = maskVal*(1-intensity) + newMaskVal*intensity; //intensity is applied only to mask

   prev = val;
   prevMask = maskVal;
  }
 }
#endif


public static (Matrix[],Matrix[]) GenerateMaskedMips (Matrix src, Matrix mask, int count=-1, float blur=0)
/// Like GenerateMips, but ignores black (lower than minVal) values
{
 if (count < 0)
  count = (int)Mathf.Log(src.rect.size.x, 2) - 1;

 Matrix[] mips = new Matrix[count]; Matrix[] mipsMask = new Matrix[count];
 Matrix mat = src; Matrix matMask = mask;
 Matrix mip; Matrix mipMask;

 CoordRect rect = src.rect;
 Matrix tmp = new Matrix( new CoordRect(rect.offset.x, rect.offset.z, rect.size.x/2, rect.size.z) );
```

```
 Matrix tmpMask = new Matrix( new CoordRect(rect.offset.x, rect.offset.z, rect.size.x/2, rect.size.z) );

 for (int i=0; i<count; i++)
 {
  mip = new Matrix( new CoordRect(mat.rect.offset.x, mat.rect.offset.z, mat.rect.size.x/2, mat.rect.size.z/2
  mipMask = new Matrix(mip.rect);

  DownscaleMaskedFast(mat, matMask, mip, mipMask, tmp, tmpMask);

  mips[i] = mip; mipsMask[i] = mipMask;
  mat = mip;  matMask = mipMask;
 }

 return (mips,mipsMask);
}


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Dow
 public static extern void DownscaleMaskedFast(Matrix src, Matrix srcMask, Matrix dst, Matrix dstMask, M
#else
 private static void DownscaleMaskedFast (Matrix src, Matrix srcMask, Matrix dst, Matrix dstMask, Matrix
 {
  CoordRect rect = src.rect;

  Stripe srcStripe = new Stripe( Mathf.Max(rect.size.x, rect.size.z) ); //just the longest dimension
```

```
Stripe dstStripe = new Stripe( srcStripe.length );
Stripe srcMaskStripe = new Stripe( srcStripe.length );
Stripe dstMaskStripe = new Stripe( srcStripe.length );

srcStripe.length = srcMaskStripe.length = src.rect.size.x;
dstStripe.length = dstMaskStripe.length = dst.rect.size.x;
for (int z=src.rect.offset.z; z<src.rect.offset.z+src.rect.size.z; z++)
{
 ReadLine(srcStripe, src, src.rect.offset.x, z);
 ReadLine(srcMaskStripe, srcMask, src.rect.offset.x, z);

 ResampleMaskedStripeDownFast(srcStripe, srcMaskStripe, dstStripe);
 ResampleStripeDownFast(srcMaskStripe, dstMaskStripe);

 WriteLine(dstStripe, tmp, src.rect.offset.x, z);
 WriteLine(dstMaskStripe, tmpMask, src.rect.offset.x, z);
}

srcStripe.length = srcMaskStripe.length = src.rect.size.z;
dstStripe.length = dstMaskStripe.length = dst.rect.size.z;
for (int x=dst.rect.offset.x; x<dst.rect.offset.x+dst.rect.size.x; x++)
{
 ReadRow(srcStripe, tmp, x, src.rect.offset.z);
 ReadRow(srcMaskStripe, tmpMask, x, src.rect.offset.z);

 ResampleMaskedStripeDownFast(srcStripe, srcMaskStripe, dstStripe);
```

```
   ResampleStripeDownFast(srcMaskStripe, dstMaskStripe);


   WriteRow(dstStripe, dst, x, src.rect.offset.z);
   WriteRow(dstMaskStripe, dstMask, x, src.rect.offset.z);
  }
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Res
 public static extern void ResampleAutomaskedStripeDownFast(Stripe src, Stripe dst, float minVal);
#else
 private static void ResampleAutomaskedStripeDownFast (Stripe src, Stripe dst, float minVal)
 /// Like ResampleStripeDownFast, but ignores black (lower than minVal) values
 {
  float sum; int num;


  for (int dstX=1; dstX<dst.length-1; dstX++)
  {
   sum = 0; num = 0;


   if (src.arr[dstX*2] > minVal) { sum += src.arr[dstX*2] * 2; num += 2; } //TODO: compare in longs
   if (src.arr[dstX*2-1] > minVal) { sum += src.arr[dstX*2-1]; num ++; }
   if (src.arr[dstX*2+1] > minVal) { sum += src.arr[dstX*2+1]; num ++; }
```

```
   dst.arr[dstX] = num>0 ? sum/num : 0;
  }

  sum = 0; num = 0;
  if (src.arr[0] > minVal) { sum += src.arr[0]*3; num += 3; }
  if (src.arr[1] > minVal) { sum += src.arr[1]; num ++; }
  dst.arr[0] = num>0 ? sum/num : 0;

  sum = 0; num = 0;
  if (src.arr[src.length-1] > minVal) { sum += src.arr[src.length-1]*3; num += 3; }
  if (src.arr[src.length-2] > minVal) { sum += src.arr[src.length-2]; num ++; }
  dst.arr[dst.length-1] = num>0 ? sum/num : 0;
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Res
 public static extern void ResampleMaskedStripeDownFast(Stripe src, Stripe mask, Stripe dst);
#else
 private static void ResampleMaskedStripeDownFast (Stripe src, Stripe mask, Stripe dst)
 /// Like ResampleAutomaskedStripeDownFast, but uses mask instead minVal
 {
  float sum; float num;

  for (int dstX=1; dstX<dst.length-1; dstX++)
```

```
  {
   sum = 0; num = 0;

   sum += src.arr[dstX*2] * mask.arr[dstX*2] * 2;  num += mask.arr[dstX*2] * 2;
   sum += src.arr[dstX*2-1] * mask.arr[dstX*2-1];  num += mask.arr[dstX*2-1];
   sum += src.arr[dstX*2+1] * mask.arr[dstX*2+1];  num += mask.arr[dstX*2+1];

   dst.arr[dstX] = num>0 ? sum/num : 0;
  }

  sum = 0; num = 0;
  sum += src.arr[0]*mask.arr[0]*3;  num += mask.arr[0]*3;
  sum += src.arr[1]*mask.arr[1];  num += mask.arr[1];
  dst.arr[0] = num>0 ? sum/num : 0;

  sum = 0; num = 0;
  sum += src.arr[src.length-1]*mask.arr[src.length-1]*3; num += mask.arr[src.length-1]*3;
  sum += src.arr[src.length-2]*mask.arr[src.length-2];   num += mask.arr[src.length-2];
  dst.arr[dst.length-1] = num>0 ? sum/num : 0;
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Ups
 public static extern void UpscaleMaskedFast(Matrix src, Matrix srcMask, Matrix dst, Matrix dstMask);
```

```
#else
 public static void UpscaleMaskedFast (Matrix src, Matrix srcMask, Matrix dst, Matrix dstMask)
 {
  //if (dst.rect.size.x/2 != src.rect.size.x || dst.rect.size.z/2 != src.rect.size.z)
  // throw new Exception("Matrix Upscale Fast: rect size mismatch: src:" + src.rect.size.ToString() + " dst:"

  Stripe srcStripe = new Stripe( dst.rect.size.Maximal );
  Stripe srcMaskStripe = new Stripe(srcStripe.length);
  Stripe dstStripe = new Stripe( srcStripe.length );
  Stripe dstMaskStripe = new Stripe(srcStripe.length);

  srcStripe.length = srcMaskStripe.length = src.rect.size.x;
  dstStripe.length = dstMaskStripe.length = dst.rect.size.x;
  for (int z=src.rect.offset.z; z<src.rect.offset.z+src.rect.size.z; z++)
  {
   ReadLine(srcStripe, src, src.rect.offset.x, z);
   ReadLine(srcMaskStripe, srcMask, srcMask.rect.offset.x, z);

   ResampleMaskedStripeUpFast(srcStripe, srcMaskStripe, dstStripe);
   ResampleStripeUpFast(srcMaskStripe, dstMaskStripe);

   WriteLine(dstStripe, dst, dst.rect.offset.x, z);
   WriteLine(dstMaskStripe, dstMask, dst.rect.offset.x, z);
  }

  srcStripe.length = srcMaskStripe.length = src.rect.size.z;
```

```
  dstStripe.length = dstMaskStripe.length = dst.rect.size.z;
  for (int x=dst.rect.offset.x; x<dst.rect.offset.x+dst.rect.size.x; x++)
  {
   ReadRow(srcStripe, dst, x, dst.rect.offset.z);
   ReadRow(srcMaskStripe, dstMask, x, dst.rect.offset.z);

   ResampleMaskedStripeUpFast(srcStripe, srcMaskStripe, dstStripe);
   ResampleStripeUpFast(srcMaskStripe, dstMaskStripe);

   WriteRow(dstStripe, dst, x, src.rect.offset.z);
   WriteRow(dstMaskStripe, dstMask, x, src.rect.offset.z);
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Res
 public static extern void ResampleAutomaskedStripeUpFast(Stripe src, Stripe dst, float minVal);
#else
 private static void ResampleAutomaskedStripeUpFast (Stripe src, Stripe dst, float minVal)
 {
  float sum = 0; int num = 0;

  for (int srcX=0; srcX<src.length-1; srcX++)
  {
```

```
   dst.arr[srcX*2] = src.arr[srcX];


   sum = 0; num = 0;
   if (src.arr[srcX] > minVal) { sum += src.arr[srcX]; num += 1; }
   if (src.arr[srcX+1] > minVal) { sum += src.arr[srcX+1]; num += 1; }
   dst.arr[srcX*2+1] = num>0 ? sum/num : 0;
  }


  dst.arr[dst.length-1] = src.arr[src.length-1];


  sum = 0; num = 0;
  if (src.arr[src.length-1] > minVal) { sum += src.arr[src.length-1]; num += 1; }
  if (src.arr[src.length-2] > minVal) { sum += src.arr[src.length-2]; num += 1; }
  dst.arr[dst.length-2] = num>0 ? sum/num : 0;
 }
#endif



#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Res
 public static extern void ResampleMaskedStripeUpFast(Stripe src, Stripe mask, Stripe dst);
#else
 private static void ResampleMaskedStripeUpFast (Stripe src, Stripe mask, Stripe dst)
 {
  float sum = 0; float num = 0;
```

```
 for (int srcX=0; srcX<src.length-1; srcX++)
 {
  dst.arr[srcX*2] = src.arr[srcX];

  sum = 0; num = 0;
  sum += src.arr[srcX]*mask.arr[srcX]; num += mask.arr[srcX];
  sum += src.arr[srcX+1]*mask.arr[srcX+1]; num += mask.arr[srcX+1];
  dst.arr[srcX*2+1] = num>0 ? sum/num : 0;
 }

 if (src.length*2 < dst.length) //duplicating the last pixel if dst is 1-pixel larger than src*2
  dst.arr[dst.length-1] = src.arr[src.length-1];

 dst.arr[src.length*2-1] = src.arr[src.length-1];

 sum = 0; num = 0;
 sum += src.arr[src.length-1]*mask.arr[src.length-1]; num += mask.arr[src.length-1];
 sum += src.arr[src.length-2]*mask.arr[src.length-2]; num += mask.arr[src.length-2];

 if (num==0)
  num=0;

 dst.arr[src.length*2-2] = num>0 ? sum/num : 0;
}
#endif
```

```
#endregion



#region Outdated Padding


 public static void PaddingSpread (Matrix srcDst, Matrix mask)
  { PaddingSpread(srcDst, srcDst, mask, mask); }


 public static void PaddingSpread (Matrix src, Matrix dst, Matrix srcMask, Matrix dstMask, float horFade=0
 /// Previous experiments with padding. Slow and do not give an effect of PaddingMips.
 {
  Stripe leftStripe = new Stripe(Mathf.Max(src.rect.size.x, src.rect.size.z));
  Stripe leftMask = new Stripe(Mathf.Max(src.rect.size.x, src.rect.size.z));


  Coord min = src.rect.Min; Coord max = src.rect.Max;



  leftStripe.length = leftMask.length = src.rect.size.x;
  for (int z=min.z; z<max.z; z++)
  {
   ReadLine(leftStripe, src, src.rect.offset.x, z);  //Stripe.Copy(leftStripe, rightStripe);
   ReadLine(leftMask, srcMask, srcMask.rect.offset.x, z);


   PadStripe(leftStripe, leftMask, fade:vertFade, toLeft:false);
   PadStripe(leftStripe, leftMask, fade:vertFade, toLeft:true);
```

```
  WriteLine(leftStripe, dst, dst.rect.offset.x, z);
  WriteLine(leftMask, dstMask, dstMask.rect.offset.x, z);
 }

 leftStripe.length = leftMask.length = src.rect.size.z;
 for (int x=min.x; x<max.x; x++)
 {
  ReadRow(leftStripe, dst, x, src.rect.offset.z);
  ReadRow(leftMask, dstMask, x, src.rect.offset.z);

  for (int i=0; i<leftMask.length; i++)
   leftMask.arr[i] *= leftMask.arr[i];

  PadStripe(leftStripe, leftMask, fade:vertFade, toLeft:false);
  PadStripe(leftStripe, leftMask, fade:vertFade, toLeft:true);

  WriteRow(leftStripe, dst, x, dst.rect.offset.z);
  WriteRow(leftMask, dstMask, x, dstMask.rect.offset.z);
 }
}


private static void PadStripe (Stripe stripe, Stripe mask, float fade=0.9f, bool toLeft=false)
/// For mask: spreading with linear lowering
/// For stripe: if new (spreaded) mask value > curr mask value - setting prev
/// prev = stripe*mask + prev*(1-mask)
```

```
{
 float prev = stripe.arr[ !toLeft ? 0 : stripe.arr.Length-1 ];
 float prevMask = mask.arr[ !toLeft ? 0 : stripe.arr.Length-1 ];

 for (int x= !toLeft ? 0 : stripe.arr.Length-1;
  !toLeft ? x<stripe.arr.Length : x>=0;
  x += !toLeft ? +1 : -1)
 {
  float val = stripe.arr[x];
  float maskVal = mask.arr[x];

  float modifiedVal = (val*maskVal + prev*prevMask) / ((maskVal+prevMask != 0) ? maskVal+prevMask :
  val = val*maskVal + modifiedVal*(1-maskVal);

  //maskVal = maskVal > prevMask ? maskVal : prevMask;
  maskVal = (long)(maskVal * 0x3FFFFFFFFFFFFFFE) > (long)(prevMask * 0x3FFFFFFFFFFFFFFE) ? m

  prev = val;
  prevMask = maskVal*fade;

  stripe.arr[x] = val;
  mask.arr[x] = maskVal;
 }
}
```

```
private static void PadStripeLeftWithBlur (Stripe stripe, float minVal=0, int blur=30)
/// Not used, stored just in case
/// Continues last value if current is less minVal, in 'blur' pixels blending it to blurred (average) value
{
 float prevDefinedVal = stripe.arr[stripe.length-1];
 float prevDefinedValBlurred = stripe.arr[stripe.length-1];
 int undefinedCount = 0;

 float invBlurStrength = 1f / blur;
 float blurStrength = 1 - invBlurStrength;

 for (int x=stripe.length-2; x>=0; x--)
 {
  float val = stripe.arr[x];
  if (val > minVal)
  {
   prevDefinedVal = val;

   prevDefinedValBlurred = undefinedCount == 0 ?
    prevDefinedValBlurred*blurStrength + val*invBlurStrength :
    prevDefinedVal; //resetting blurred on new defined pixel

   undefinedCount = 0;
  }
  else
  {
```

```
   if (undefinedCount > blur)
    val = prevDefinedValBlurred;
   else
   {
    float blurPercent = 1f*undefinedCount / blur;
    val = prevDefinedVal*(1-blurPercent) + prevDefinedValBlurred*blurPercent;
   }

   stripe.arr[x] = val;

   undefinedCount++;
  }
 }
}


public static void StripeToMask (Stripe stripe, Stripe mask, float minVal=0)
/// Sets mask values to 1 if stripe > minVal, and to 0 if stripe less minVal
/// Reads stripe, writes mask
{
 for (int x=0; x<stripe.length; x++)
  mask.arr[x] = stripe.arr[x] > minVal ? 1 : 0;
}


public static void BlendStripes (Stripe leftStripe, Stripe rightStripe, Stripe leftMask, Stripe rightMask)
```

```
/// Blending two stripes together using their mask values (more mask - more weight)
/// Writing result to leftStripe.
/// Btw merging masks and writing in leftMask
{
 for (int x=0; x<leftStripe.length; x++)
 {
  float leftMaskVal =  leftMask.arr[x];
  float rightMaskVal =  rightMask.arr[x];

  float sum = leftMaskVal + rightMaskVal;
  leftStripe.arr[x] =
   sum > 0 ?
   (leftStripe.arr[x] * leftMaskVal  +  rightStripe.arr[x] * rightMaskVal) / sum :
   0;
  leftMask.arr[x] = sum / 2;
 }
}

public static void MaxStripes (Stripe leftStripe, Stripe rightStripe, Stripe leftMask, Stripe rightMask)
/// Blending two stripes together using their mask values (more mask - more weight)
/// Writing result to leftStripe.
/// Btw merging masks and writing in leftMask
{
 for (int x=0; x<leftStripe.length; x++)
 {
```

```
  float leftMaskVal =  leftMask.arr[x];
  float rightMaskVal =  rightMask.arr[x];

  float sum = leftMaskVal + rightMaskVal;
  leftStripe.arr[x] = leftMask.arr[x] > rightMask.arr[x] ?
   leftStripe.arr[x] :
   rightStripe.arr[x];
  leftMask.arr[x] = (leftMask.arr[x]+rightMask.arr[x]) / 2;
 }
}


public static void PaddingBac (Matrix matrix, Matrix mask, float sharpness=0.25f, int iterations=2)
/// Fills non-masked areas with edge color
/// Does not support initial mask antialiasing
{
 float multiply = 1-sharpness; //multiply controls sharpness

 Stripe leftMask = new Stripe(Mathf.Max(matrix.rect.size.x, matrix.rect.size.z));
 Stripe rightMask = new Stripe(leftMask.length);
 Stripe leftStripe = new Stripe(leftMask.length);
 Stripe rightStripe = new Stripe(leftMask.length);

 Coord min = matrix.rect.Min; Coord max = matrix.rect.Max;

 for (int i=0; i<iterations; i++)
```

```
{


 leftMask.length = matrix.rect.size.x;
 rightMask.length = leftMask.length;
 leftStripe.length = leftMask.length;
 rightStripe.length = leftMask.length;
 for (int z=min.z; z<max.z; z++)
 {
  //reading lines
  ReadLine(leftStripe, matrix, matrix.rect.offset.x, z);
  ReadLine(leftMask, mask, matrix.rect.offset.x, z);
  Stripe.Copy(leftStripe, rightStripe);
  Stripe.Copy(leftMask, rightMask);

  //spreading
  PadLeft(leftStripe, leftMask, multiply);
  PadRight(rightStripe, rightMask, multiply);

  //assembling back to leftStripe
  for (int x=0; x<leftStripe.length; x++)
  {
   float sum = leftMask.arr[x] + rightMask.arr[x];
   leftStripe.arr[x] =
    sum > 0 ?
    (leftStripe.arr[x] * leftMask.arr[x]  +  rightStripe.arr[x] * rightMask.arr[x]) / sum :
    0;
```

```
  leftMask.arr[x] = sum / 2;
 }

 WriteLine(leftStripe, matrix, matrix.rect.offset.x, z);
 WriteLine(leftMask, mask, mask.rect.offset.x, z);
}

leftMask.length = matrix.rect.size.z;
rightMask.length = leftMask.length;
leftStripe.length = leftMask.length;
rightStripe.length = leftMask.length;

for (int x=min.x; x<max.x; x++)
{
 //reading lines
 ReadRow(leftStripe, matrix, x, matrix.rect.offset.z);
 ReadRow(leftMask, mask, x, matrix.rect.offset.z);
 Stripe.Copy(leftStripe, rightStripe);
 Stripe.Copy(leftMask, rightMask);

 //spreading
 PadLeft(leftStripe, leftMask, multiply);
 PadRight(rightStripe, rightMask, multiply);

 //applying to dst
 for (int z=0; z<leftStripe.length; z++)
```

```
   {
    float sum = leftMask.arr[z] + rightMask.arr[z];
    leftStripe.arr[z] =
     sum > 0 ?
     (leftStripe.arr[z] * leftMask.arr[z]  +  rightStripe.arr[z] * rightMask.arr[z]) / sum :
     0;

    leftMask.arr[z] = sum / 2;
   }

   WriteRow(leftStripe, matrix, x, matrix.rect.offset.z);
   WriteRow(leftMask, mask, x, mask.rect.offset.z);
  }

 }
}


private static void PadLeft (Stripe stripe, Stripe mask, float multiply=0.9f)
{
 float prevVal = stripe.arr[stripe.length-1];
 float prevMask = mask.arr[stripe.length-1];

 for (int x=stripe.length-2; x>=0; x--)
 {
  float currMask = mask.arr[x];
```

```
  float currVal = stripe.arr[x];

  float maskSum = currMask+prevMask;
  currVal = maskSum > 0 ?
   (currVal*currMask + prevVal*prevMask) / maskSum :
   0;
  stripe.arr[x] = currVal;

  if (prevMask > currMask)
  {
   currMask = prevMask*multiply; // + currVal*(1-multiply)*regardSmallerValues;
   mask.arr[x] = currMask;
  }

  prevVal = currVal;
  prevMask = currMask;
 }
}


private static void PadRight (Stripe stripe, Stripe mask, float multiply=0.9f)
{
 float prevVal = stripe.arr[0];
 float prevMask = mask.arr[0];

 for (int x=1; x<stripe.length-1; x++)
```

```
  {
   float currMask = mask.arr[x];
   float currVal = stripe.arr[x];

   float maskSum = currMask+prevMask;
   currVal = maskSum > 0 ?
    (currVal*currMask + prevVal*prevMask) / maskSum :
    0;
   stripe.arr[x] = currVal;

   if (prevMask > currMask)
   {
    currMask = prevMask*multiply; // + currVal*(1-multiply)*regardSmallerValues;
    mask.arr[x] = currMask;
   }

   prevVal = currVal;
   prevMask = currMask;
  }
 }

#endregion


#region Outdated Padding (Predict)
```

```
#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)

 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Pre
 public static extern void PredictPadding(Matrix src, Matrix dst, float expandEdge = 0.1f, int expandPixels

 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Pad
 public static extern void PadStripe(Stripe leftStripe, Stripe rightStripe, Stripe leftMask, Stripe rightMask, f

 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Pre
 public static extern void PredictPadStripeLeft(Stripe stripe, float expandEdge = 0.1f, int expandPixels = 5

 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Pre
 public static extern void PredictPadStripeRight(Stripe stripe, float expandEdge = 0.1f, int expandPixels =

 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Pad
 public static extern void PadStripeLeft(Stripe stripe);

 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Pad
 public static extern void PadStripeRight(Stripe stripe);

#else

public static void PredictPadding (Matrix src, Matrix dst, float expandEdge=0.1f, int expandPixels=50)
/// Fills gaps in matrix. Uses negative values as a mask
{
 CoordRect rect = src.rect;
```

```
Coord min = rect.Min; Coord max = rect.Max;

Stripe leftStripe = new Stripe( Mathf.Max(rect.size.x, rect.size.z) );
Stripe rightStripe = new Stripe(leftStripe.length);
Stripe leftMask = new Stripe(leftStripe.length);
Stripe rightMask = new Stripe(leftStripe.length);

for (int i=0; i<5; i++)
{
 leftStripe.length = rect.size.x;  rightStripe.length = rect.size.x;
 leftMask.length = rect.size.x;   rightMask.length = rect.size.x;

 for (int z=min.z; z<max.z; z++)
 {
  ReadLine(leftStripe, src, rect.offset.x, z);
  PadStripe(leftStripe, rightStripe, leftMask, rightMask, expandEdge, expandPixels);
  WriteLine(leftStripe, dst, rect.offset.x, z);
 }

 /*stripe.length = rect.size.z;
 maskStripe.length = rect.size.z;
 for (int x=min.x; x<max.x; x++)
 {
  ReadRow(src, stripe, x, rect.offset.z);
  mask.ReadRow(maskStripe, x, rect.offset.z);
  PadStripe(stripe, maskStripe);
```

```
   dst.WriteRow(stripe, x, rect.offset.z); //apply row additively (with mid-point 0.5)
  }*/
 }
}



internal static void PadStripe (Stripe leftStripe, Stripe rightStripe, Stripe leftMask, Stripe rightMask, float e
/// Processed values are stored in leftStripe
{
 //reading left stripe
 for (int x=0; x<leftStripe.length; x++)
 {
  leftMask.arr[x] = rightMask.arr[x] = leftStripe.arr[x] > 0 ? 1 : 0;
  rightStripe.arr[x] = leftStripe.arr[x];
 }

 //spreading masks
 SpreadMultiplyLeft(leftMask, multiply:0.9f);
 SpreadMultiplyRight(rightMask, multiply:0.9f);

 //padding stripes
 PadStripeLeft(leftStripe);
 PadStripeRight(rightStripe);

 //applying masks
 for (int x=0; x<leftStripe.length; x++)
```

```
    {
     float lm = leftMask.arr[x];
     float rm = rightMask.arr[x];
     float sum = lm+rm;
     if (sum != 0) { lm/=sum; rm/=sum; }

     float p;
     if (lm > rm)
     {
      p = lm;
      leftStripe.arr[x] = leftStripe.arr[x]*(1-p) + rightStripe.arr[x]*p;
     }
     else
     {
      p = rm;
      leftStripe.arr[x] = leftStripe.arr[x]*p + rightStripe.arr[x]*(1-p);
     }

//     leftStripe.arr[x] = leftMask.arr[x];
    }
   }


   internal static void PredictPadStripeLeft (Stripe stripe, float expandEdge=0.1f, int expandPixels=50)
   {
    //finding stripe start
```

```
int s;
for (s=0; s<stripe.length; s++)
 if (stripe.arr[s] >= 0) break;
if (s>=stripe.length-1) return;

float pv = stripe.arr[s];

int empty = 0;
//float avg = stripe.arr[0];
float vector = 0;

for (int x=s; x<stripe.length; x++)
{
 float v = stripe.arr[x];
 //float m = maskStripe.arr[x];

 if (v < 0)
 {
  empty++;

  if (empty < expandPixels)
   pv += vector*(1f-empty/expandPixels);

  stripe.arr[x] = pv;
 }
 else
```

```
  {
   empty = 0;
   vector = vector*(1-expandEdge) + (v-pv)*expandEdge;
   //avg = avg*0.9f + v*0.1f;

   //stripe.arr[x] = v;
   pv = v;
  }
 }
}

internal static void PredictPadStripeRight (Stripe stripe, float expandEdge=0.1f, int expandPixels=50)
{
 float pv = stripe.arr[stripe.length-1];

 int empty = 0;
 float vector = 0;

 for (int x=stripe.length-1; x>0; x--)
 {
  float v = stripe.arr[x];

  if (v < 0)
  {
   empty++;
```

```
   if (empty < expandPixels)
    pv += vector*(1f-empty/expandPixels);

   stripe.arr[x] = pv;
  }
  else
  {
   empty = 0;
   vector = vector*(1-expandEdge) + (v-pv)*expandEdge;
   //avg = avg*0.9f + v*0.1f;

   stripe.arr[x] = v;
   pv = v;
  }
 }
}

internal static void PadStripeLeft (Stripe stripe)
{
 float pv = stripe.arr[0];

 for (int x=0; x<stripe.length; x++)
 {
  float v = stripe.arr[x];

  if (v < 0)
```

```
    stripe.arr[x] = pv;
   else
    pv = v;
  }
 }

 internal static void PadStripeRight (Stripe stripe)
 {
  float pv = stripe.arr[0];

  for (int x=stripe.length-1; x>0; x--)
  {
   float v = stripe.arr[x];

   if (v < 0)
    stripe.arr[x] = pv;
   else
    pv = v;
  }
 }

 #endif

#endregion
```

```
#region Lock Padding

 public static void ExtendCircular (this Matrix matrix, Coord center, int radius, int extendRange, int expand
 {
  //resetting area out of radius
  RemoveOuter(matrix, center, radius);

  //creating radial lines
  int numLines = Mathf.CeilToInt( Mathf.PI * radius ); //using only the half of the needed lines
  float angleStep = Mathf.PI * 2 / (numLines-1); //in radians

  Stripe stripe = new Stripe(extendRange*2);
  Stripe maskStripe = new Stripe(extendRange*2);
  for (int i=0; i<extendRange; i++)
   maskStripe.arr[i] = 1;

  for (int i=0; i<numLines; i++)
  {
   float angle = i*angleStep;
   angle -= Mathf.PI*3f / 4f;
   Vector2 direction = new Vector2( Mathf.Sin(angle), Mathf.Cos(angle) ).normalized;

   //making any of the step components equal to 1
   Vector2 posDir = new Vector2 (
    (direction.x>0 ? direction.x : -direction.x),
    (direction.y>0 ? direction.y : -direction.y) );
```

```
 float max = posDir.x>posDir.y ? posDir.x : posDir.y;
 Vector2 step = direction / max;
 int predictStart = (int)(extendRange * max);

 //finding proper start so that stripe middle be on the edge
 Vector2 start = center.vector2 + direction*radius;
 start -= step*extendRange;

 ReadInclined(stripe, matrix, start, step);
 PredictPadStripeLeft(stripe, expandPixels:expandPixels);
 WriteInclined(stripe, matrix, start, step);
}

//creating a diagonal stripe for squares
{
 Vector2 dir =  new Vector2(-1,-1);
 Vector2 start = center.vector2 + dir.normalized*radius;
 stripe = new Stripe(extendRange*2); //new Stripe( Mathf.Max(matrix.rect.size.x, matrix.rect.size.z) / 2);

 ReadInclined(stripe, matrix, start, dir);
 PredictPadStripeLeft(stripe, expandPixels:expandPixels);
 WriteInclined(stripe, matrix, start, dir);
}

//filling gaps between lines
Stripe leftStripe = new Stripe((radius+extendRange+1)*2*4 + 1);  leftStripe.Fill(-1); //otherwise will partly
```

```
 Stripe rightStripe = new Stripe(leftStripe.length);     rightStripe.Fill(-1);
 Stripe leftMask = new Stripe(leftStripe.length);
 Stripe rightMask = new Stripe(leftStripe.length);

 for (int i=(int)(radius*0.7f); i<radius+extendRange; i++)
 {
  ReadSquare(leftStripe, matrix, center, i);
  PadStripe(leftStripe, rightStripe, leftMask, rightMask, expandEdge:0, expandPixels:0);
  WriteSquare(leftStripe, matrix, center, i);
 }

 //blurring
 BlurCircular(matrix, center, radius + extendRange, extendRange);
 DownsampleBlurCircular(matrix, center, radius + (int)(extendRange*0.05f), (int)(extendRange*0.95f+1),
}


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Rem
 public static extern void RemoveOuter(Matrix matrix, Coord coord, int radius);
#else
 private static void RemoveOuter (this Matrix matrix, Coord coord, int radius)
 /// Fill the outer part of the matrix with negative values so that FillGaps could be used
 {
  Coord min = matrix.rect.Min; Coord max = matrix.rect.Max;
  int radiusSq = radius*radius;
```

```
  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    int pos = (z-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;

    float dist = (x-coord.x)*(x-coord.x) + (z-coord.z)*(z-coord.z);
    if (dist > radiusSq) { matrix.arr[pos] = -1; continue; }
   }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Blur
 public static extern void BlurCircular(Matrix matrix, Coord coord, int radius, int extendRange);
#else
 private static void BlurCircular (this Matrix matrix, Coord coord, int radius, int extendRange)
 /// Maybe blur standard and then mask circular?
 {
  CoordRect rect = matrix.rect;
  Coord min = rect.Min; Coord max = rect.Max;
  int radiusSq = radius*radius;
  int extendRangeSq = extendRange*extendRange;

  Stripe stripe = new Stripe( Mathf.Max(rect.size.x, rect.size.z) );
```

```
stripe.length = rect.size.x;
for (int z=min.z; z<max.z; z++)
{
 ReadLine(stripe, matrix, rect.offset.x, z);
 BlurIteration(stripe, 1.5f);
 //BlurIteration(stripe, 1.5f);
 for (int x=min.x; x<max.x; x++)
 {
  int stripePos = x-matrix.rect.offset.x;
  int matrixPos = (z-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;

  float distSq = (x-coord.x)*(x-coord.x) + (z-coord.z)*(z-coord.z);
  if (distSq > radiusSq)
   matrix.arr[matrixPos] = stripe.arr[stripePos];
 }
}

stripe.length = rect.size.z;
for (int x=min.x; x<max.x; x++)
{
 ReadRow(stripe, matrix, x, rect.offset.z);
 BlurIteration(stripe, 1.5f);
 //BlurIteration(stripe, 1.5f);
 for (int z=min.z; z<max.z; z++)
 {
```

```
    int stripePos = z-matrix.rect.offset.z;
    int matrixPos = (z-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;

    float distSq = (x-coord.x)*(x-coord.x) + (z-coord.z)*(z-coord.z);
    if (distSq > radiusSq)
     matrix.arr[matrixPos] = stripe.arr[stripePos];
   }
  }
 }
#endif


#if MM_NATIVE && (UNITY_EDITOR || !UNITY_ANDROID && !ENABLE_IL2CPP)
 [DllImport ("NativePlugins", CallingConvention = CallingConvention.Cdecl, EntryPoint = "MatrixOps_Dow
 public static extern void DownsampleBlurCircular(Matrix matrix, Coord coord, int radius, int extendRange
#else
 private static void DownsampleBlurCircular (this Matrix matrix, Coord coord, int radius, int extendRange,
 {
  int maxDownsample = (int)Mathf.Log(matrix.rect.size.x, 2) - 1;
  if (downsample > maxDownsample) downsample = maxDownsample;
  if (downsample == 0) return;

  CoordRect rect = matrix.rect;
  Coord min = rect.Min; Coord max = rect.Max;
  int radiusSq = radius*radius;
  int extendRangeSq = extendRange*extendRange;
```

```
Stripe hiStripe = new Stripe( Mathf.Max(rect.size.x, rect.size.z) );
Stripe srcStripe = new Stripe(hiStripe.length);
Stripe loStripe = new Stripe( hiStripe.length / 2);

hiStripe.length = rect.size.x;
loStripe.length = hiStripe.length / 2;
for (int z=min.z; z<max.z; z++)
{
 ReadLine(hiStripe, matrix, rect.offset.x, z);

 ResampleStripeDownFast(hiStripe, loStripe);
 BlurStripe(loStripe, blur);
 ResampleStripeUpFast(loStripe, hiStripe);

 for (int x=min.x; x<max.x; x++)
 {
  int stripePos = x-matrix.rect.offset.x;
  int matrixPos = (z-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;

  float distSq = (x-coord.x)*(x-coord.x) + (z-coord.z)*(z-coord.z);
  if (distSq > radiusSq)
  {
   float p = 1 - (distSq-radiusSq) / extendRangeSq;
   if (p>1) p = 1; if (p<0) p = 0;
   p *= p;
```

```
   matrix.arr[matrixPos] = matrix.arr[matrixPos]*p + hiStripe.arr[stripePos]*(1-p);
  }
 }
}


hiStripe.length = rect.size.z;
loStripe.length = hiStripe.length / 2;
for (int x=min.x; x<max.x; x++)
{
 ReadRow(hiStripe, matrix, x, rect.offset.z);

 ResampleStripeDownFast(hiStripe, loStripe);
 BlurStripe(loStripe, blur);
 ResampleStripeUpFast(loStripe, hiStripe);

 for (int z=min.z; z<max.z; z++)
 {
  int stripePos = z-matrix.rect.offset.z;
  int matrixPos = (z-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;

  float distSq = (x-coord.x)*(x-coord.x) + (z-coord.z)*(z-coord.z);
  if (distSq > radiusSq)
  {
   float p = 1 - (distSq-radiusSq) / extendRangeSq;
   if (p>1) p = 1; if (p<0) p = 0;
```

```
        p *= p;
        matrix.arr[matrixPos] = matrix.arr[matrixPos]*p + hiStripe.arr[stripePos]*(1-p);
       }
      }
     }

     /*hiStripe.length = rect.size.z;
     loStripe.length = hiStripe.length / downsample;
     for (int x=min.x; x<max.x; x++)
     {
      matrix.ReadRow(hiStripe, x, rect.offset.z);

      ResampleStripeLinear(hiStripe, loStripe);
      BlurStripe(loStripe, blur);
      ResampleStripeCubic(loStripe, hiStripe);

      matrix.WriteRow(hiStripe, x, rect.offset.z);
     }*/
    }
   #endif

  #endregion
 }
}
```

```
ï»¿//using System; //conflicts with UnityEngine.Object
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace Den.Tools.Matrices
{
 //[System.Serializable]
 public partial class MatrixSet
 {
  public CoordRect rect;  //storing for 0 length

  public Vector3 worldPos;
  public Vector3 worldSize;
  public Vector3 WorldMax => worldPos+worldSize;
  public Vector2D PixelSize => new Vector2D(worldSize.x/(rect.size.x-1), worldSize.z/(rect.size.z-1));

  private DictionaryOrdered<Prototype,Matrix> prototypesMatrices = new DictionaryOrdered<Prototype,Ma

  public enum PrototypeType { Texture, Prefab, TerrainLayer };

  public MatrixSet (CoordRect rect, Vector3 worldPos, Vector3 worldSize)
  {
   this.rect = rect;
   this.worldPos = worldPos;
   this.worldSize = worldSize;
```

```
}

public MatrixSet (CoordRect rect, Vector2D worldPos, Vector2D worldSize, float height)
{
 this.rect = rect;
 this.worldPos = new Vector3(worldPos.x, 0, worldPos.z);
 this.worldSize = new Vector3(worldSize.x, height, worldSize.z);
}

public MatrixSet (MatrixSet src)
{
 rect = src.rect;
 worldPos = src.worldPos;
 worldSize = src.worldSize;

 foreach (var kvp in src.prototypesMatrices)
  prototypesMatrices.Add(kvp.Key, new Matrix(kvp.Value));
}

public MatrixSet (CoordRect rect, Vector3 worldPos, Vector3 worldSize, ICollection<Prototype> prototype
{
 this.rect = rect;
 this.worldPos = worldPos;
 this.worldSize = worldSize;

 foreach (Prototype prototype in prototypes)
```

```
  prototypesMatrices.Add(prototype, new Matrix(rect));
}



public Matrix this[Prototype prototype]
{
 get{
  if (prototypesMatrices.TryGetValue(prototype, out Matrix matrix))
   return matrix;
  else
   return null;
 }

 set{
  if (value.rect != rect)
   throw new System.Exception("Trying to add a matrix of different resolution");

  if (prototypesMatrices.ContainsKey(prototype))
   prototypesMatrices[prototype] = value;
  else
   prototypesMatrices.Add(prototype, value);
 }
}

public Matrix this[int num]
{
```

```
 get => prototypesMatrices[num];
 set => prototypesMatrices[num] = value;
}

public Matrix GetMatrixByNum (int num) => prototypesMatrices[num];
public void SetMatrixByNum (int num, Matrix value) => prototypesMatrices[num] = value;

public Prototype GetPrototypeByNum (int num) => prototypesMatrices.GetKeyByNum(num);

public int Count => prototypesMatrices.Count;

public ICollection<Prototype> Prototypes => prototypesMatrices.Keys;
public ICollection<Matrix> Matrices => prototypesMatrices.Values;

public IEnumerable<T> PrototypesOfType<T> () where T: class
{
 foreach (object obj in prototypesMatrices.Keys)
  if (obj is T tobj) yield return tobj;
}

public bool TryGetValue (Prototype prototype, out Matrix matrix) => prototypesMatrices.TryGetValue(prot
public bool TryGetValue (System.Predicate<object> predicate, out Matrix matrix)
{
 foreach (var kvp in prototypesMatrices)
```

```
  if (predicate(kvp.Key))
  {
   matrix = kvp.Value;
   return true;
  }

 matrix = null;
 return false;
}

public void SetOffset (Coord newOffset)
{
 rect.offset = newOffset;
 foreach (Matrix im in prototypesMatrices.Values)
  im.rect.offset = newOffset;
}

public void Append (Matrix matrix, Prototype prototype, bool normalized=true)
///Adds new channel (if no prototype in dict) or appends existing one (if protoptye already added)
///InvMultiplies all other channels making result normalized
{
 if (normalized)
 {
  foreach (Matrix im in prototypesMatrices.Values)
   im.MultiplyInv(matrix);
 }
```

```
 if (prototypesMatrices.TryGetValue(prototype, out Matrix m))
  m.Add(matrix);
 else
 {
  m = new Matrix(matrix);
  prototypesMatrices.Add(prototype,m);
 }
}


public void SyncPrototypes (ICollection<Prototype> prototypes)
///Adds prototypes from list if they are not added and creates empty matrices for them
{
 foreach (Prototype prototype in prototypes)
  if (!prototypesMatrices.Contains(prototype))
  {
   Matrix matrix = new Matrix(rect);
   prototypesMatrices.Add(prototype, matrix);
  }
}


public void Resize (CoordRect dstRect, bool interpolate=false)
///Resizes all the matrices and changes rect size
///Trying to use fast resize when possible
```

```
///Mainly for splats import/export
{
 DictionaryOrdered<Prototype,Matrix> resizedProtMatrices = new DictionaryOrdered<Prototype,Matrix>()

 foreach (var kvp in prototypesMatrices)
 {
  Matrix src = kvp.Value;
  Matrix dst = new Matrix(dstRect);

  ResizeMatrix(src, dst, interpolate);

  resizedProtMatrices.Add(kvp.Key, dst);
 }

 prototypesMatrices = resizedProtMatrices;
 rect = dstRect;
}

public static void Resize (MatrixSet src, MatrixSet dst, bool interpolate=false)
/// instant resise doesn't actually changes src
{
 foreach (var kvp in src.prototypesMatrices)
 {
  Matrix srcMatrix = kvp.Value;

  Matrix dstMatrix;
```

```
  if (!dst.TryGetValue(kvp.Key, out dstMatrix)) //adding matrix if it's not in dst
  {
   dstMatrix = new Matrix(dst.rect);
   dst[kvp.Key] = dstMatrix;
  }

  ResizeMatrix(srcMatrix, dstMatrix, interpolate);
 }
}

private static void ResizeMatrix (Matrix src, Matrix dst, bool interpolate=false)
{
 if (interpolate)
 {
  if (src.rect.size.x*2 == dst.rect.size.x)
   MatrixOps.UpscaleFast(src, dst);
  else if (src.rect.size.x == dst.rect.size.x*2)
   MatrixOps.DownscaleFast(src, dst);
  else
   MatrixOps.Resize(src, dst);
 }
 else
  MatrixOps.ResizeNearestNeighbor(src, dst);
}
```

```
public static void CopyIntersected (MatrixSet src, MatrixSet dst)
/// Analog of Matrix.Copy / Matrix.Fill, but only works with a set
/// This will also add new src matrix to dst
{
 foreach (var kvp in src.prototypesMatrices)
 {
  Prototype prototype = kvp.Key;

  Matrix srcMatrix = kvp.Value;
  Matrix dstMatrix;

  if (!dst.prototypesMatrices.TryGetValue(prototype, out dstMatrix))
  {
   dstMatrix = new Matrix(dst.rect);
   dst.prototypesMatrices.Add(prototype, dstMatrix);
  }

  Matrix.CopyIntersected(srcMatrix, dstMatrix);
 }
}

public static void CopyResized (MatrixSet src, MatrixSet dst,
 Vector2D srcRectPos, Vector2D srcRectSize, Coord dstRectPos, Coord dstRectSize)
///Resizes all the matrices and changes rect size
///Trying to use fast resize when possible
///Mainly for splats import/export
```

```
  {
   foreach (var kvp in src.prototypesMatrices)
   {
    Prototype prototype = kvp.Key;

    Matrix srcMatrix = kvp.Value;
    Matrix dstMatrix;

    if (!dst.prototypesMatrices.TryGetValue(prototype, out dstMatrix))
    {
     dstMatrix = new Matrix(dst.rect);
     dst.prototypesMatrices.Add(prototype, dstMatrix);
    }

    Matrix.CopyResized(srcMatrix, dstMatrix, srcRectPos, srcRectSize, dstRectPos, dstRectSize);

   }
  }
 }

}
```

```
ï»¿//using System; //conflicts with UnityEngine.Object
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace Den.Tools.Matrices
{
 //[System.Serializable]
 public partial class MatrixSet
 {
  public struct Prototype
  {
   public TerrainLayer layer;
   public Texture2D tex;
   public GameObject prefab;
   //public TerrainLayer tlayer; //not complicating things yet
   public int instanceNum; //if this texture is used in array more than once. Starts with 0

   public Object Object
   {get{
    if (layer != null) return layer;
    else if (tex != null) return tex;
    else return prefab;
   }}

   public Prototype (TerrainLayer layer) { this.layer=layer; this.tex = null; this.prefab = null; instanceNum = 0
```

```
public Prototype (Texture2D texture) { this.layer=null; this.tex = texture; this.prefab = null; instanceNum =

public Prototype (GameObject prefab) { this.layer=null; this.tex = null; this.prefab = prefab; instanceNum

public Prototype (TerrainLayer layer, int instanceNum) { this.layer=layer; this.tex = null; this.prefab = null;

public Prototype (Texture2D texture, int instanceNum) { this.layer=null; this.tex = texture; this.prefab = nu

public Prototype (GameObject prefab, int instanceNum) { this.layer=null; this.tex = null; this.prefab = pref


public Prototype (System.Array layers, int num)
/// Creates a prototype and assigns proper texture/object and instance num
/// No need to make generic
{
 object layer = layers.GetValue(num);
 Object obj = GetLayerObject(layer);
 if (obj == null)
  {this.layer = null; this.tex = null; this.prefab = null; instanceNum = 0; return; }


 instanceNum = 0;
 for (int i=0; i<num; i++)
 {
  Object prevObj =  GetLayerObject( layers.GetValue(i) );
  if (prevObj == null)
   continue;


  if (prevObj == obj)
   instanceNum++;
 }
```

```
 if (obj is TerrainLayer layerObj) this.layer = layerObj;
 else this.layer = null;

 if (obj is Texture2D texObj) this.tex = texObj;
 else this.tex = null;

 if (obj is GameObject goObj) this.prefab = goObj;
 else this.prefab = null;
}

public T[] CheckAppendLayers<T> (T[] layers) where T: class
/// If there is no such prototype in layers - appending layers with newly created prototype
/// Returns layers themselves if no need to add
{
 int currInstNum = 0;
 Object thisObj = Object;
 bool foundInLayers = false;

 for (int i=0; i<layers.Length; i++)
 {
  Object layerObj = GetLayerObject( layers.GetValue(i) );
  if (layerObj == null)
   continue;

  if (layerObj==thisObj)
```

```
  {
   if (currInstNum==instanceNum)
    { foundInLayers=true; break; }

   currInstNum++;
  }
 }

 if (!foundInLayers)
  ArrayTools.Add(ref layers, NewLayer<T>(this));

 return layers;
}

private static Object GetLayerObject (object layer)
/// Gets terrainLayer/detailPrototype main object (texture or prefab)
{
 switch (layer)
 {
  case TerrainLayer tlayer: return tlayer;
  case DetailPrototype dprot: return  dprot.Object();
  case Texture2D tex: return tex;
  default: return null;
 }
}
```

```
public static T NewLayer<T> (Prototype prot) where T: class
/// Converts prototype to new terrainLayer or detailPrototype - depends on T type
{
 if (typeof(T) == typeof(Texture2D))
 {
  if (prot.tex!=null) return (T)(object)prot.tex;
  if (prot.prefab!=null) return (T)(object)prot.prefab.GetMainTexture();
 }

 if (typeof(T) == typeof(TerrainLayer))
 {
  if (prot.layer!=null) return (T)(object)prot.layer;
  if (prot.tex!=null)
  {
   TerrainLayer tlayer = new TerrainLayer() {
    diffuseTexture = prot.tex,
    normalMapTexture = prot.tex.GetNormalTexture(),
    tileSize = new Vector2(20,20) };
   #if UNITY_EDITOR
   UnityEditor.AssetDatabase.CreateAsset(tlayer, "Assets/" + prot.tex.name + "_TerrainLayer.terrainlayer
   #endif
   return tlayer as T;
  }
 }
```

```
    if (typeof(T) == typeof(DetailPrototype))
    {
     if (prot.tex!=null) return new DetailPrototype() {
      renderMode = DetailRenderMode.Grass,
      dryColor = new Color(0.95f, 1f, 0.65f),
      healthyColor = new Color(0.5f, 0.65f, 0.35f),
      prototypeTexture = prot.tex } as T;
     //if (prot.tlayer!=null) return new DetailPrototype() {
     // renderMode = DetailRenderMode.Grass,
     // prototypeTexture = prot.tlayer.diffuseTexture } as T;
     else if (prot.prefab!=null) return new DetailPrototype() {
      renderMode = DetailRenderMode.VertexLit,
      usePrototypeMesh = true,
      prototype = prot.prefab } as T;
    }

    return null;
   }
  }
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace Den.Tools.Matrices
{
 [System.Serializable]
 public class MatrixWorld : Matrix, ICloneable
 {
  //public CoordRect rect; //in base
  public Vector3 worldPos;
  public Vector3 worldSize;
  public Vector3 WorldMax => worldPos+worldSize;

  public Vector2D PixelSize => new Vector2D(worldSize.x/(rect.size.x-1), worldSize.z/(rect.size.z-1));


  #region Constructors

   public MatrixWorld () { arr = new float[0]; rect = new CoordRect(0,0,0,0); } //for serializer

   public MatrixWorld (CoordRect rect, Vector2D worldPos, Vector2D worldSize, float worldHeight, float[] ar
   {
    this.rect = rect;
    this.count = rect.Count;
```

```
 DefineArray(array);

 this.worldPos = (Vector3)worldPos;
 this.worldSize = (Vector3)worldSize;
 this.worldSize.y = worldHeight;
}

public MatrixWorld (CoordRect rect, Vector3 worldPos, Vector3 worldSize)
{
 this.rect = rect;
 this.count = rect.Count;
 arr = new float[rect.size.x*rect.size.z];
 this.worldPos = worldPos;
 this.worldSize = worldSize;
}

public MatrixWorld (Coord offset, Coord size, Vector3 worldPos, Vector3 worldSize)
{
 this.rect = new CoordRect(offset,size);
 this.count = rect.Count;
 arr = new float[rect.size.x*rect.size.z];
 this.worldPos = worldPos;
 this.worldSize = worldSize;
}

public MatrixWorld (Matrix matrix, Vector3 worldPos, Vector3 worldSize)
```

```
{
 this.rect = matrix.rect;
 this.count = rect.Count;
 arr = matrix.arr;
 this.worldPos = worldPos;
 this.worldSize = worldSize;
}

public MatrixWorld (Matrix matrix, Vector2D worldPos, Vector2D worldSize, float worldHeight)
{
 this.rect = matrix.rect;
 this.count = rect.Count;
 arr = matrix.arr;
 this.worldPos = (Vector3)worldPos;
 this.worldSize = new Vector3(worldSize.x, worldHeight, worldSize.x);
}

public MatrixWorld (MatrixWorld mw)
{
 this.rect = mw.rect;
 this.count = rect.Count;
 arr = new float[mw.arr.Length];
 Array.Copy(mw.arr, arr, mw.arr.Length);
 worldPos = mw.worldPos;
 worldSize = mw.worldSize;
}
```

```
 public object Clone () { return new MatrixWorld(this); }

 //to create from simple Matrix use new MatrixWorld(matrix.rect, wp, ws, matrix.arr);

#endregion


#region World to Pixel / Pixel to World

 public Coord WorldToPixel (float x, float z)
 {
  //finding relative percent
  float percentX = (x - worldPos.x) / worldSize.x;
  float percentZ = (z - worldPos.z) / worldSize.z;

  //get map coordinates
  float mapX = percentX*(rect.size.x-1) + rect.offset.x;
  float mapZ = percentZ*(rect.size.z-1) + rect.offset.z;

  mapX += 0.5f; mapZ += 0.5f;

  //flooring map values
  int ix = (int)mapX; if (mapX<0) ix--; if (ix==rect.offset.x+rect.size.x) ix--;
  int iz = (int)mapZ; if (mapZ<0) iz--; if (iz==rect.offset.z+rect.size.z) iz--;
```

```
 return new Coord(ix, iz);
}



public Vector3 WorldToPixelInterpolated (float x, float z)
{
 //finding relative percent
 float percentX = (x - worldPos.x) / worldSize.x;
 float percentZ = (z - worldPos.z) / worldSize.z;

 //get map coordinates
 float mapX = percentX*(rect.size.x-1) + rect.offset.x;
 float mapZ = percentZ*(rect.size.z-1) + rect.offset.z;

 //mapX = (x-worldPos.x) * rect.size.x / worldSize.x  +  rect.offset.x  -  0.5f;
 //mapZ = (z-worldPos.z) * rect.size.z / worldSize.z  +  rect.offset.z  -  0.5f;

 //float pixelSizeX = worldSize.x / (rect.size.x);
 //mapX = x / pixelSizeX;

 //float halfPixelX = (worldSize.x/rect.size.x) / 2;
 //percentX = (x-halfPixelX - worldPos.x) / worldSize.x;  //finding relative percent
 //mapX = percentX*rect.size.x + rect.offset.x;

 return new Vector3(mapX, 0, mapZ);
}
```

```
public int WorldDistToPixel (float worldX)
{
 float percentX = worldX / worldSize.x;
 float mapX = percentX*rect.size.x;// + rect.offset.x;
 int ix = (int)(mapX); if (mapX<0) ix--; if (ix==rect.offset.x+rect.size.x) ix--;
 return ix;
}

public float WorldDistToPixelInterpolated (float worldX)
{
 float percentX = worldX / worldSize.x;
 return percentX*rect.size.x;// + rect.offset.x;
}



public Vector3 PixelToWorld (float x, float z)
/// Finding pixel center if Center enabled
/// Can take interpolated values too
{
 //finding relative percent
 float percentX = 1f * (x - rect.offset.x) / (rect.size.x-1);
 float percentZ = 1f * (z - rect.offset.z) / (rect.size.z-1);

 //get map coordinates
```

```
 float worldX = percentX*worldSize.x + worldPos.x;
 float worldZ = percentZ*worldSize.z + worldPos.z;

 return new Vector3(worldX, 0, worldZ);
}


public float PixelDistToWorld (int mapX)
{
 float percentX = (mapX+0.5f - rect.offset.x) / rect.size.x;  //taking the center of the pixel
 float worldX = percentX*worldSize.x;// + worldPos.x;
 return worldX;
}

public CoordRect WorldRectToPixels (Vector2D wOffset, Vector2D wSize)
/// Creates pixel CoordRect that includes world rect (size is always ceiled)
{
 Vector2D wMax = wOffset + wSize;

 Coord pOffset = WorldToPixel(wOffset.x, wOffset.z);
 Coord pMax = WorldToPixel(wMax.x, wMax.z);

 return new CoordRect(pOffset, pMax-pOffset);
}

#endregion
```

```
#region Get/Set

 public bool ContainsWorldValue (float x, float z)
 {
  return  x > worldPos.x  &&   x < worldPos.x+worldSize.x  &&
    z > worldPos.z  &&   z < worldPos.z+worldSize.z;
 }

 public virtual float GetWorldValue (float x, float z)
 {
  Coord coord = WorldToPixel(x,z);
  return this[coord];

  /*//finding relative percent
  float percentX = (x - worldPos.x) / worldSize.x;
  float percentZ = (z - worldPos.z) / worldSize.z;

  //get map coordinates
  float mapX = percentX*rect.size.x + rect.offset.x;
  float mapZ = percentZ*rect.size.z + rect.offset.z;

  //flooring map values (values should be floored, not rounded since height pixel on terrain has it's own dim
  int ix = (int)mapX; if (mapX<0) ix--; if (ix==rect.offset.x+rect.size.x) ix--;
  int iz = (int)mapZ; if (mapZ<0) iz--; if (iz==rect.offset.z+rect.size.z) iz--;
```

```
 UnityEngine.Assertions.Assert.IsTrue(ix>=rect.offset.x && iz>=rect.offset.z && ix<rect.offset.x+rect.size.

 return arr[(iz-rect.offset.z)*rect.size.x + ix - rect.offset.x];*/
}

public void SetWorldValue (float x, float z, float val)
{
 //finding relative percent
 float percentX = (x - worldPos.x) / worldSize.x;
 float percentZ = (z - worldPos.z) / worldSize.z;

 //get map coordinates
 float mapX = percentX*rect.size.x + rect.offset.x;
 float mapZ = percentZ*rect.size.z + rect.offset.z;

 //flooring map values (values should be floored, not rounded since height pixel on terrain has it's own dir
 int ix = (int)mapX; if (mapX<0) ix--; if (ix==rect.offset.x+rect.size.x) ix--;
 int iz = (int)mapZ; if (mapZ<0) iz--; if (iz==rect.offset.z+rect.size.z) iz--;

 UnityEngine.Assertions.Assert.IsTrue(ix>=rect.offset.x && iz>=rect.offset.z && ix<rect.offset.x+rect.size.

 arr[(iz-rect.offset.z)*rect.size.x + ix - rect.offset.x] = val;
}

public virtual float GetWorldInterpolatedValue (float x, float z, bool roundToShort=false)
```

```
{
 //float halfPixelX = (worldSize.x/rect.size.x) / 2;
 //float percentX = (x-halfPixelX - worldPos.x) / worldSize.x;  //finding relative percent
 //float mapX = percentX*rect.size.x + rect.offset.x;

 //float mapX = (x-worldPos.x) * rect.size.x / worldSize.x  +  rect.offset.x  -  0.5f;   //optimized
 //float mapZ = (z-worldPos.z) * rect.size.z / worldSize.z  +  rect.offset.z  -  0.5f;

 //finding relative percent
 float percentX = (x - worldPos.x) / worldSize.x;
 float percentZ = (z - worldPos.z) / worldSize.z;

 //clamping
 if (percentX > 1) percentX = 1;
 if (percentZ > 1) percentZ = 1;

 //get map coordinates
 float mapX = percentX*(rect.size.x-1) + rect.offset.x;
 float mapZ = percentZ*(rect.size.z-1) + rect.offset.z;

 return GetInterpolated(mapX, mapZ, roundToShort); //copy
 //return GetFloored(mapX,mapZ);

 //float val = GetFloored(mapX,mapZ);
 //ushort sh = (ushort)(val*65535);
 //val = (float)sh / (32767*2);
```

```
    //return val;
   }


  #endregion
 }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Den.Tools.Matrices;

namespace Den.Tools
{
 [System.Serializable]
 public class PositionMatrix : Matrix2D<Vector3>
 /// A Matrix of Vector3 that always has an origin at the center of coordinates
 /// WorldRect just intersects the real rect. Rect never starts at worldPos, except 0 coordinate
 {
  public Vector3 worldPos;
  public Vector3 worldSize;
  public float cellSize;
  public int margins; //number of cells rect expanded, bounds arount world rect

  public PositionMatrix (CoordRect rect, Vector3 worldPos, Vector3 worldSize)
  {
   this.rect = rect;
   this.worldPos = worldPos;
   this.worldSize = worldSize;
   this.cellSize = worldSize.x / rect.size.x;

   count = rect.size.x*rect.size.z;
```

```
 arr = new Vector3[count];
}



public Coord GetCoord (Vector3 worldPos)
/// Returns the cell that contains worldPos
{
 int x = (int)(float)(worldPos.x/cellSize); if (worldPos.x<0) x--;
 int z = (int)(float)(worldPos.z/cellSize); if (worldPos.z<0) z--;
 return new Coord(x,z);
}

public void SetPosition (Vector3 worldPos)
{
 //GetCoord(worldPos)
 int x = (int)(float)(worldPos.x/cellSize); if (worldPos.x<0) x--;
 int z = (int)(float)(worldPos.z/cellSize); if (worldPos.z<0) z--;

 //return this[x,z];
 arr[(z-rect.offset.z)*rect.size.x + x - rect.offset.x] = worldPos;
}



public void SetHeight (int x, int z, float height)
{
```

```
 //this[x,z].y = height;
 arr[(z-rect.offset.z)*rect.size.x + x - rect.offset.x].y = height;
}



public float GetHeight (int x, int z)
{
 //return this[x,z].y;
 return arr[(z-rect.offset.z)*rect.size.x + x - rect.offset.x].y;
}



public void Scatter (float uniformity, Noise rnd, float maxHeight=1)
/// use maxHeight = 0 if want to ignore scattering y
{
 Coord min = rect.Min; Coord max = rect.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   Vector3 pos = new Vector3(x*cellSize + cellSize/2, 0, z*cellSize + cellSize/2); //cell center
   if (uniformity < 1)
   {
    Vector3 rndPos = new Vector3(
     x*cellSize  +  rnd.Random(x,z,0)*cellSize,
     rnd.Random(x,z,2) * maxHeight,
     z*cellSize  +  rnd.Random(x,z,1)*cellSize);
```

```
    pos = pos*uniformity + rndPos*(1-uniformity);
   }


   this[x,z] = pos;
  }
}



public PositionMatrix Relaxed (float strength=1)
/// Trying to increase the distance if two objects are too close to each other
{
 float relStrength = strength*cellSize;
 PositionMatrix newMatrix = new PositionMatrix(rect, worldPos, worldSize);

 Coord min = rect.Min; Coord max = rect.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
 {
  Vector3 pos = this[x,z];
  Vector3 relaxVec = new Vector3();

  for (int ix=-1; ix<=1; ix++)
   for (int iz=-1; iz<=1; iz++)
  {
   if (ix==0 && iz==0) continue;
   int nx = x+ix; int nz=z+iz;
```

```
    if (nx<min.x || nx>=max.x || nz<min.z || nz>=max.z) continue;

    Vector3 npos = arr[(nz-rect.offset.z)*rect.size.x + nx - rect.offset.x]; //this[nx,nz];

    Vector3 relaxDir = pos - npos;
    relaxVec += relaxDir.normalized * (1/relaxDir.sqrMagnitude);
   }

   pos += relaxVec*relStrength;

   //clamping within cell
   if (pos.x < x*cellSize) pos.x = x*cellSize;
   if (pos.x > (x+1)*cellSize) pos.x = (x+1)*cellSize;
   if (pos.z < z*cellSize) pos.z = z*cellSize;
   if (pos.z > (z+1)*cellSize) pos.z = (z+1)*cellSize;

   newMatrix[x,z] = pos;
  }

  return newMatrix;
 }

 public void CleanUp (Matrix probMatrix, Noise rnd)
 /// Erasing objects according to probability matrix
 /// Erasing means setting Y coord to negative infinity
```

```
/// This and probMatrix rects are combined (projected onto each other)
{
 Coord min = rect.Min; Coord max = rect.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
 {
  int i = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;

  Vector3 pos = arr[i];
  float relX = (pos.x-worldPos.x) / worldSize.x;
  float relZ = (pos.z-worldPos.z) / worldSize.z;

  float probX = (relX * probMatrix.rect.size.x) + probMatrix.rect.offset.x;
  float probZ = (relZ * probMatrix.rect.size.z) + probMatrix.rect.offset.z;

  if (probX < probMatrix.rect.offset.x) probX = probMatrix.rect.offset.x;
  if (probZ < probMatrix.rect.offset.z) probZ = probMatrix.rect.offset.z;
  if (probX >= probMatrix.rect.offset.x+probMatrix.rect.size.x-1) probX = probMatrix.rect.offset.x+probMatri
  if (probZ >= probMatrix.rect.offset.z+probMatrix.rect.size.z-1) probZ = probMatrix.rect.offset.z+probMatri

  //float probVal = probMatrix.GetInterpolated(probX, probZ); //has a 0.5-pixel offset (oddly enough)
  float probVal = probMatrix[(int)probX, (int)probZ];
  float rndVal = rnd.Random(x,z,0);
  if (probVal < rndVal) arr[i].y = Mathf.NegativeInfinity;
 }
}
```

```
public void GetTwoClosest (Vector3 worldPos, out Vector3 closest, out Vector3 secondClosest, out float m
{
 //GetCoord(worldPos)
 int x = (int)(float)(worldPos.x/cellSize); if (worldPos.x<0) x--;
 int z = (int)(float)(worldPos.z/cellSize); if (worldPos.z<0) z--;

 Vector3 point = arr[(z-rect.offset.z)*rect.size.x + x - rect.offset.x]; //this[x,z];

 closest = secondClosest = point; //to avoid using unassigned
 minDist = secondMinDist = 200000000;

 for (int ix=-1; ix<=1; ix++)
  for (int iz=-1; iz<=1; iz++)
 {
  //if (ix==0 && iz==0) continue;

  int nx = x+ix; int nz=z+iz;

  //if (!rect.CheckInRange(nx,nz)) continue;
  if (nx<rect.offset.x || nx>=rect.offset.x+rect.size.x ||
   nz<rect.offset.z || nz>=rect.offset.z+rect.size.z) continue;

  Vector3 nPoint = arr[(nz-rect.offset.z)*rect.size.x + nx - rect.offset.x]; //this[nx, nz];
```

```
  float dist = (worldPos.x-nPoint.x)*(worldPos.x-nPoint.x) + (worldPos.z-nPoint.z)*(worldPos.z-nPoint.z);
  if (dist<minDist)
  {
   secondMinDist = minDist; minDist = dist;
   secondClosest = closest; closest = nPoint;
  }
  else if (dist<secondMinDist)
  {
   secondMinDist = dist;
   secondClosest = nPoint;
  }
 }
}



public void FillPosTab (PosTab posTab, float minHeight=-200000000)
/// Copy all of the positions to posTab, skipping objects that out of range and those who have height belov
{
 Coord min = rect.Min; Coord max = rect.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   Vector3 pos = this[x,z];

   //skipping out of cell
   if (pos.x < x*cellSize || pos.x > (x+1)*cellSize ||
```

```
    pos.z < z*cellSize || pos.z > (z+1)*cellSize) continue;


   //skipping out of range
   if (pos.x < posTab.pos.x || pos.x > posTab.pos.x+posTab.size.x ||
    pos.z < posTab.pos.z || pos.z > posTab.pos.z+posTab.size.z) continue;


   //skipping height
   if (pos.y<minHeight) continue;


   Transition trs = new Transition(pos.x, pos.z);
   trs.hash = x*2000 + z; //to make hash independent from grid size
   posTab.Add(trs);
  }
}



public Vector3[] ToArray ()
{
 Vector3[] arr = new Vector3[rect.size.x * rect.size.z];


 int t = 0;
 Coord min = rect.Min; Coord max = rect.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   arr[t] = this[x,z];
```

```
   t++;
  }

 return arr;
}



public void AddTransitionsList (TransitionsList trns)
{
 for (int t=0; t<trns.count; t++)
  SetPosition(trns.arr[t].pos);
}

public void AddTransitionsList (TransitionsList trns, float customHeight)
{
 for (int t=0; t<trns.count; t++)
 {
  if (trns.arr[t].pos.x < worldPos.x || trns.arr[t].pos.x > worldPos.x+worldSize.x ||
   trns.arr[t].pos.z < worldPos.z || trns.arr[t].pos.z > worldPos.z+worldSize.z)
    continue;

  SetPosition( new Vector3(trns.arr[t].pos.x, customHeight, trns.arr[t].pos.z) );
 }
}
```

```
  public TransitionsList ToTransitionsList ()
  {
   TransitionsList list = new TransitionsList(); //capacity rect.size.x * rect.size.z

   Coord min = rect.Min; Coord max = rect.Max;
   for (int x=min.x; x<max.x; x++)
    for (int z=min.z; z<max.z; z++)
    {
     Vector3 pos = this[x,z];
     //if (pos.y < minHeight) continue;
     Transition trs = new Transition(pos.x, pos.z);
     trs.hash = x*2000 + z; //to make hash independent from grid size
     list.Add(trs);
    }

   return list;
  }
 }
}
```

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Den.Tools.Matrices;

[assembly: System.Runtime.CompilerServices.InternalsVisibleTo("Tools.Tests.Editor")]

namespace Den.Tools
{
 [System.Serializable]
 public class StackMatrix<T>
 /// Matrix-like array with limited length, that holds only N elements
 /// Adds new ones with Set, and erases old ones when stack is full
 {
  [SerializeField] internal Dictionary<Coord, (uint,T)> stack = new Dictionary<Coord, (uint, T)>();  //coord to
  [SerializeField] private int count;
  [SerializeField] private int capacity = 10;
  [SerializeField] private uint lastId = 0;

  public StackMatrix() {}
  public StackMatrix (int capacity) {this.capacity=capacity;}

  public T this[Coord c]
  {
   get
```

```
{
 stack.TryGetValue(c, out (uint,T) numT);
 return numT.Item2;
}
set
{
 lastId++;

 //coord already in stack - just replacing with new, and refreshing id
 if (stack.ContainsKey(c))
  stack[c] = (lastId, value);

 else
 {
  //capacity isn't reached - adding to stack
  if (count < capacity)
  {
   stack.Add(c, (lastId,value));
   count++;
  }

  //capacity reached - common case - removing old, adding new
  else
  {
   Coord lowerestCoord = GetCoordWithLowerestId();
   stack.Remove(lowerestCoord);
```

```
    stack.Add(c, (lastId,value));
   }
  }

  if (lastId >= 4294967294)
  {
   SetMaxId(2147483647);
   lastId = 2147483648;
  }
 }
}

public T this[int x, int z]
{
 get => this[new Coord(x,z)];
 set => this[new Coord(x,z)] = value;
}

public int Count { get => count; }

public int Capacity
{
 get {return capacity;}
```

```
 set
 {
  if (value < count)
   Limit(value);
  capacity = value;
 }
}


public bool Contains (Coord c) => stack.ContainsKey(c);
public bool Contains (int x, int z) => stack.ContainsKey(new Coord(x,z));


private void Limit (int num)
///Leaves only num or less elements in stack
{
 Coord[] sorted = SortByIds();

 Dictionary<Coord, (uint,T)> srcStack = new Dictionary<Coord, (uint, T)>(stack);
 stack.Clear();

 for (int i=sorted.Length-1; i>=sorted.Length-num; i--)
 {
  Coord c = sorted[i];
  (uint,T) el = srcStack[c];
  stack.Add(c, el);
```

```
 }

 count = num;
}

private Coord[] SortByIds ()
///Returns all of the stack coordinates sorted by id nums from min (oldest) to max (newest)
{
 Coord[] coords = new Coord[stack.Count];
 uint[] ids = new uint[stack.Count];

 int i=0;
 foreach (var kvp in stack)
 {
  coords[i] = kvp.Key;
  ids[i] = kvp.Value.Item1;
  i++;
 }

 Array.Sort(ids, coords);

 return coords;
}
```

```
private Coord GetCoordWithLowerestId ()
{
 uint lowerestId = uint.MaxValue;
 Coord lowerestCoord = new Coord();

 foreach (var kvp in stack)
  if (kvp.Value.Item1 <= lowerestId)
  {
   lowerestId = kvp.Value.Item1;
   lowerestCoord = kvp.Key;
  }

 return lowerestCoord;
}


internal void SetMaxId (uint maxId)
///In case reached uint limitation
{
 Dictionary<Coord, (uint,T)> srcStack = new Dictionary<Coord, (uint, T)>(stack);
 stack.Clear();

 foreach (var kvp in srcStack)
 {
  uint id = kvp.Value.Item1;
  if (id>maxId) id -= maxId;
```

```
    else id = 0;
    stack.Add(kvp.Key, (id, kvp.Value.Item2));
   }
  }
 }
}
```

```
ï»¿
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;



//namespace MapMagic.Core
namespace Den.Tools
{
 [CustomEditor(typeof(MatrixAsset))]
 public class MatrixAssetInspector : Editor
 {
  MatrixAsset matrixAsset;
  UI ui = new UI();

  bool colorize = false;
  bool relief = false;



  public override void OnInspectorGUI ()
  {
   ui.Draw(DrawGUI, inInspector:true);
```

```
}



private void DrawGUI ()
{
 matrixAsset = (MatrixAsset)target;

 using (Cell.LinePx(32))
  Draw.Label("WARNING: Serializing this asset when selected in \nInspector can slow down editor GUI pe

 Cell.EmptyLinePx(5);

 if (matrixAsset.matrix != null  &&  matrixAsset.preview == null)
  matrixAsset.RefreshPreview();

 if (matrixAsset.preview != null)
 {
  using (Cell.LinePx(256))
  {
   Cell.EmptyRowRel(1);
   using (Cell.RowPx(256))
   {
    //Draw.Texture(matrixAsset.preview);
    Draw.MatrixPreviewTexture(matrixAsset.preview, colorize:colorize, relief:relief, min:0, max:1);
    Draw.MatrixPreviewReliefSwitch(ref colorize, ref relief);
   }
```

```
  Cell.EmptyRowRel(1);
 }

 if (matrixAsset.rawPath != null)
  using (Cell.LineStd) Draw.Label(matrixAsset.rawPath);

 if (matrixAsset.matrix != null)
  using (Cell.LineStd) Draw.Label(matrixAsset.matrix.rect.size.x + ", " + matrixAsset.matrix.rect.size.z);
}

Cell.EmptyLinePx(5);

using (Cell.LineStd) Draw.Field(ref matrixAsset.source, "Map Source");

if (matrixAsset.source == MatrixAsset.Source.Raw)
{
 using (Cell.LinePx(22)) Draw.Label("Square gray 16bit RAW, PC byte order", style:UI.current.styles.help

 using (Cell.LineStd) if (Draw.Button("Load RAW"))
 {
  string newPath = EditorUtility.OpenFilePanel("Import Texture File", "", "raw,r16");

  if (newPath!=null && newPath.Length!=0)
  {
   UnityEditor.Undo.RecordObject(this, "Import RAW");
   matrixAsset.rawPath = newPath;
```

```
    matrixAsset.Reload();

    EditorUtility.SetDirty(matrixAsset);
   }
  }
 }

 else //texture
  using (Cell.LinePx(0))
  {
   using (Cell.LineStd) Draw.ObjectField(ref matrixAsset.textureSource, "Texture"); //
   using (Cell.LineStd) Draw.Field(ref matrixAsset.channelSource, "Channel"); //

   if (Cell.current.valChanged)
    matrixAsset.Reload();
  }

 using (Cell.LineStd)
 {
  Cell.current.disabled =
   (matrixAsset.source == MatrixAsset.Source.Raw && matrixAsset.rawPath == null) ||
   (matrixAsset.source == MatrixAsset.Source.Texture && matrixAsset.textureSource == null);
  if (Draw.Button("Reload"))
  {
   matrixAsset.Reload();
```

```
    EditorUtility.SetDirty(matrixAsset);
   }
  }
 }
}

class MatrixAssetTexturePostprocessor : AssetPostprocessor
{
 //public static WeakEvent<Texture2D> OnTextureImported = new WeakEvent<Texture2D>();
 //In MatrixAsset since it should work in non-editors

 //public void OnPostprocessTexture(Texture2D tex)  //using OnPostprocessAllAssets because tex here is

 static void OnPostprocessAllAssets(string[] importedAssets, string[] deletedAssets, string[] movedAssets,
 {
  for (int a=0; a<importedAssets.Length; a++)
  {
   if (AssetDatabase.GetMainAssetTypeAtPath(importedAssets[a]) != typeof(Texture2D)) continue;
   Texture2D tex = AssetDatabase.LoadAssetAtPath<Texture2D>(importedAssets[a]);

   if (MatrixAsset.OnTextureImported != null)
    MatrixAsset.OnTextureImported(tex);
  }
 }
}
```

```
ï»¿
using UnityEngine;
using UnityEditor;
using System.Collections;
using System.Collections.Generic;

using Den.Tools;
using Den.Tools.GUI;



namespace Den.Tools.Matrices
{
 [CustomEditor(typeof(MatrixObject))]
 public class MatrixObjectInspector : Editor
 {
  MatrixObject matrixObject;
  UI ui = new UI();

  bool colorize = false;
  bool relief = false;

  public override void OnInspectorGUI () => ui.Draw(DrawGUI, inInspector:true);
  private void DrawGUI ()
  /// Copy of MatrixAssetInspector
  /// TODO: reuse code
  {
```

```
matrixObject = (MatrixObject)target;

if (matrixObject.matrix != null  &&  matrixObject.preview == null)
 matrixObject.RefreshPreview();

if (matrixObject.preview != null)
{
 using (Cell.LinePx(256))
 {
  Cell.EmptyRowRel(1);
  using (Cell.RowPx(256))
  {
   //Draw.Texture(matrixAsset.preview);
   Draw.MatrixPreviewTexture(matrixObject.preview, colorize:colorize, relief:relief, min:0, max:1);
   Draw.MatrixPreviewReliefSwitch(ref colorize, ref relief);
  }
  Cell.EmptyRowRel(1);
 }

 if (matrixObject.rawPath != null)
  using (Cell.LineStd) Draw.Label(matrixObject.rawPath);

 if (matrixObject.matrix != null)
  using (Cell.LineStd) Draw.Label(matrixObject.matrix.rect.size.x + ", " + matrixObject.matrix.rect.size.z);
}
```

```
Cell.EmptyLinePx(5);

using (Cell.LineStd) Draw.Field(ref matrixObject.source, "Map Source");

if (matrixObject.source == MatrixAsset.Source.Raw)
{
 using (Cell.LinePx(22)) Draw.Label("Square gray 16bit RAW, PC byte order", style:UI.current.styles.help

 using (Cell.LineStd) if (Draw.Button("Load RAW"))
 {
  string newPath = EditorUtility.OpenFilePanel("Import Texture File", "", "raw,r16");

  if (newPath!=null && newPath.Length==0)
  {
   UnityEditor.Undo.RecordObject(this, "Import RAW");
   matrixObject.rawPath = newPath;

   matrixObject.Reload();

   EditorUtility.SetDirty(matrixObject);
  }
 }
}

else if (matrixObject.source == MatrixAsset.Source.Texture)
 using (Cell.LinePx(0))
```

```
 {
  using (Cell.LineStd) Draw.ObjectField(ref matrixObject.textureSource, "Texture"); //
  using (Cell.LineStd) Draw.Field(ref matrixObject.channelSource, "Channel"); //

  if (Cell.current.valChanged)
   matrixObject.Reload();
 }

else if (matrixObject.source == MatrixAsset.Source.New)
{
 using (Cell.LinePx(0))
 {
  using (Cell.LineStd) Draw.Field(ref matrixObject.newRes, "Resolution");
  using (Cell.LineStd) Draw.Field(ref matrixObject.newOffset, "Offset");

  if (Cell.current.valChanged)
   matrixObject.Reload();
 }
}

using (Cell.LineStd)
{
 Cell.current.disabled =
  (matrixObject.source == MatrixAsset.Source.Raw && matrixObject.rawPath == null) ||
  (matrixObject.source == MatrixAsset.Source.Texture && matrixObject.textureSource == null);
 if (Draw.Button("Reload"))
```

```
 {
  matrixObject.Reload();
  EditorUtility.SetDirty(matrixObject);
 }
}

Cell.EmptyLinePx(5);
using (Cell.LineStd) Draw.Field(ref matrixObject.worldPosition, "World Pos");
using (Cell.LineStd) Draw.Field(ref matrixObject.worldSize, "World Size");
using (Cell.LineStd) Draw.Field(ref matrixObject.worldHeight, "World Height");

Cell.EmptyLinePx(5);
using (Cell.LineStd) Draw.Field(ref matrixObject.displayGizmo, "Gizmo");
using (Cell.LineStd) Draw.ToggleLeft(ref matrixObject.centerCell, "Center Cell");
using (Cell.LineStd) Draw.Field(ref matrixObject.filterMode, "Filter Mode");
}



public virtual void OnSceneGUI()
{
 if (matrixObject.displayGizmo == MatrixObject.DisplayGizmo.Texture)
 {
  if (matrixObject.textureGizmo == null) matrixObject.Reload();
  matrixObject.textureGizmo.SetOffsetSize(matrixObject.worldPosition, matrixObject.worldSize);
  matrixObject.textureGizmo.Draw();
 }
```

```
   if (matrixObject.displayGizmo == MatrixObject.DisplayGizmo.Height  ||  matrixObject.displayGizmo == Ma
   {
    if (matrixObject.heightGizmo == null) matrixObject.Reload();
    matrixObject.heightGizmo.SetOffsetSize(
     (Vector3)matrixObject.worldPosition,
     new Vector3(matrixObject.worldSize.x, matrixObject.worldHeight, matrixObject.worldSize.z));
    matrixObject.heightGizmo.Draw();
   }
  }
 }
}
```

```
ï»¿using UnityEngine;
using UnityEditor;
using System;
using System.Collections.Generic;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.GUI;



namespace Den.Tools.Matrices.Window
{
 [System.Serializable]
 public class MatrixWindow : BaseMatrixWindow
 {
  [SerializeField] Matrix matrix;
  public override Matrix Matrix => matrix;

  Texture2D previewTexture;
  public override Texture2D PreviewTexture => previewTexture;

  [RuntimeInitializeOnLoadMethod, UnityEditor.InitializeOnLoadMethod]
  public static void Subscribe ()
  {
   DebugGizmos.onPreviewMatrix += (m,n) => OpenUpdate(m,n);
```

```
 Matrix.onPreview += (m,n) => OpenUpdate(m,n);
}

public static MatrixWindow GetWindow (Matrix matrix, string name=null)
/// Tries to find opened window by matrix (and name if provided)
{
 MatrixWindow[] windows = Resources.FindObjectsOfTypeAll<MatrixWindow>();

 for (int i=0; i<windows.Length; i++)
  if (windows[i].matrix == matrix || (name!=null && windows[i].name==name))
   return windows[i];

 return null;
}


public static MatrixWindow GetCreateWindow (Matrix matrix, string name=null)
/// Tries to find opened window by matrix/name and updates it, and opens new window if non is found
{
 MatrixWindow window = GetWindow(matrix, name);

 if (window == null)
 {
  window = CreateInstance<MatrixWindow>();
  window.ShowTab();
  window.SetMatrix(matrix, name);
```

```
 }

 window.Show();

 return window;
}

public void SetMatrix (Matrix matrix, string name=null)
{
 //generating preview texture
 if (previewTexture == null || previewTexture.width != matrix.rect.size.x || previewTexture.height != matrix.r
 {
  previewTexture = new Texture2D(matrix.rect.size.x, matrix.rect.size.z, TextureFormat.RFloat, false, true
  previewTexture.filterMode = FilterMode.Point;
 }

 matrix.ExportTextureRaw(previewTexture);

 SetMatrix(matrix, previewTexture);

 if (name != null) this.name = name;
}

public void SetMatrix (Matrix matrix, Texture2D previewTexture)
```

```
/// In case we already have preview texture (generated by MapMagic for instance)
{
 this.matrix = matrix;
 this.previewTexture = previewTexture;

 foreach (IPlugin plugin in plugins)
  plugin.Setup(matrix);

 Repaint();
}



public static void OpenUpdate (Matrix matrix, string name=null)
{
 MatrixWindow window = GetCreateWindow(matrix, name);
 if (name != null) window.name = name;
 else window.name = "Matrix Preview";
 window.SetMatrix(matrix);
}

public static void OpenUpdate (Matrix matrix, Texture2D tex, string name=null)
{
 MatrixWindow window = GetCreateWindow(matrix, name);
 if (name != null) window.name = name;
 else window.name = "Matrix Preview";
 window.SetMatrix(matrix, tex);
```

```
}



//[MenuItem ("Window/Test/Matrix Preview")]
public static void ShowWindow ()
{
 MatrixWindow window = (MatrixWindow)GetWindow(typeof (MatrixWindow));
 window.position = new Rect(100,100,300,800);
}

[MenuItem ("Assets/To Matrix Preview")]
private static void LoadToMatrixPreviewWindow()
{
 Texture2D tex = Selection.activeObject as Texture2D;
 if (tex == null) return;

 Matrix matrix = new Matrix( new CoordRect(0,0,tex.width, tex.height) );
 matrix.ImportTexture(tex, channel:0);

 GetCreateWindow(matrix, tex.name);
}

[MenuItem ("Assets/To Matrix Preview", true)]
private static bool LoadToMatrixPreviewWindowValidation()
{
 return Selection.activeObject is Texture2D;
```

```
 }
}

[System.Serializable]
public abstract class BaseMatrixWindow : EditorWindow
{
 public abstract Matrix Matrix { get; }
 public abstract Texture2D PreviewTexture { get; }
 private Material textureRawMat;

 public bool colorize;
 public bool relief;
 public float min = 0;
 public float max = 1;

 UI toolbarUI = new UI();
 UI previewUI = UI.ScrolledUI(maxZoom:16);
 Vector2 serializedScroll;
 bool serializedScrollStored = false;
 float serializedZoom = 1;

 const int toolbarWidth = 260; //128+4

 public IPlugin[] plugins = new IPlugin[] {
  new StatsPlugin(),
  new ViewPlugin(),
```

```
 new PixelPlugin(),
 new SlicePlugin(),
 new ProcessPlugin(),
 new ImportPlugin(),
 new ExportPlugin() };



#region Scroll/Zoom

 public Vector2 Scroll
 {
  get{ return previewUI.scrollZoom.scroll;}
  set{ previewUI.scrollZoom.scroll=value; serializedScroll=value; }
 }

 public float Zoom
 {
  get{ return previewUI.scrollZoom.zoom;}
  set{previewUI.scrollZoom.zoom=value; serializedZoom=value;}
 }

 public Vector2 ScrollZero => new Vector2(0,0);

 public Vector2 ScrollCenter
 {get{
```

```
 if (Matrix==null) return Vector2.zero;
 else return new Vector2(
 (-Matrix.rect.offset.x-Matrix.rect.size.x/2) * previewUI.scrollZoom.zoom - toolbarWidth/2 + Screen.width/2
 (Matrix.rect.offset.z+Matrix.rect.size.z/2) * previewUI.scrollZoom.zoom + Screen.height/2 );
}}

 public Rect MatrixRect
 {get{
 if (Matrix == null) return new Rect(0,0,0,0);
 else return new Rect(Matrix.rect.offset.x, -Matrix.rect.offset.z-Matrix.rect.size.z, Matrix.rect.size.x, Matrix
 }}

 public Rect ToMatrixRect (CoordRect srcRect) =>
   new Rect(srcRect.offset.x, -srcRect.offset.z-srcRect.size.z, srcRect.size.x, srcRect.size.z);


#endregion


#region GUI

 private void OnGUI ()
 {
 titleContent = new GUIContent(name);

 if (serializedScrollStored  ||  Matrix==null) previewUI.scrollZoom.scroll = serializedScroll;
```

```
 else previewUI.scrollZoom.scroll = ScrollCenter;
 previewUI.scrollZoom.zoom = serializedZoom;

 previewUI.Draw(DrawPreview, inInspector:false);
 toolbarUI.Draw(DrawToolbar, inInspector:false);

 serializedScroll = previewUI.scrollZoom.scroll;
 serializedScrollStored = true;
 serializedZoom = previewUI.scrollZoom.zoom;
}


protected virtual void DrawPreview ()
{
 //background
 Rect displayRect = new Rect(0, 0, Screen.width, Screen.height);
 float gridBackgroundColor = !StylesCache.isPro ? 0.45f : 0.2f;
 float gridColor = !StylesCache.isPro ? 0.5f : 0.23f;
 Draw.StaticGrid(
  displayRect: displayRect,
  cellSize:128,
  color:new Color(gridColor,gridColor,gridColor),
  background:new Color(gridBackgroundColor,gridBackgroundColor,gridBackgroundColor),
  fadeWithZoom:true);

 using (Cell.Custom(MatrixRect))
```

```
  Draw.MatrixPreviewTexture(PreviewTexture, colorize:colorize, relief:relief, min:min, max:max);


 Draw.StaticAxis(displayRect, 0, false, Color.red);
 Draw.StaticAxis(displayRect, 0, true, Color.blue);


 foreach (IPlugin plugin in plugins)
  plugin.DrawWindow(Matrix, this);
}



protected virtual void DrawToolbar ()
{
 Cell.EmptyRow();
 using (Cell.RowPx(toolbarWidth))
 {
  foreach (IPlugin plugin in plugins)
  {
   Cell.EmptyLinePx(6);
   using (Cell.LineStd)
   {
    bool enabled = plugin.Enabled;


    using (new Draw.FoldoutGroup(ref enabled, plugin.Name, isLeft:false, style:UI.current.styles.foldoutOp
     if (enabled)
      plugin.DrawInspector(Matrix, this);
```

```
        plugin.Enabled = enabled;
      }
     }
    }

    Cell.EmptyRowPx(8);
   }

  #endregion

 }
}
```

```
ï»¿using UnityEngine;
using UnityEditor;
using System;
using System.Collections.Generic;
using UnityEngine.Profiling;

using Den.Tools;
using Den.Tools.Matrices;
using Den.Tools.GUI;



namespace Den.Tools.Matrices.Window
{
 public interface IPlugin
 {
  bool Enabled { get; set; }
  string Name { get; }

  void Setup (Matrix matrix);  //aka OnEnable, runs once on matrix assigned/changed
  void DrawWindow (Matrix matrix, BaseMatrixWindow window); //draws something in window
  void DrawInspector (Matrix matrix, BaseMatrixWindow window); //draws matrix toolbar
 }

 public class StatsPlugin : IPlugin
 {
  private bool enabled = true;
```

```
public bool Enabled { get{return enabled;} set{enabled=value;} }

public string Name => "Stats";


private float min;

private float max;

private float[] histogram;


public void Setup (Matrix matrix)

{

 min = matrix.MinValue();

 max = matrix.MaxValue();


 histogram = matrix.Histogram(256, max:1, normalize:true);

}



public void DrawWindow (Matrix matrix, BaseMatrixWindow window) { }



public void DrawInspector (Matrix matrix, BaseMatrixWindow window)

{

 if (matrix==null) return;


 using (Cell.LineStd) Draw.Field(matrix.rect, "Rect");

 using (Cell.LineStd) Draw.Field(min, "Min Value");

 using (Cell.LineStd) Draw.Field(max, "Max Value");
```

```
  Cell.EmptyLinePx(4);
  using (Cell.LinePx(64))
  {
   if (histogram != null && histogram.Length!=0)
    Draw.Histogram(histogram, new Vector4(0,0,0,0.25f), new Vector4(0,0,0,0));
   Draw.Grid(new Color(0,0,0,0.4f)); //background grid
  }
 }
}


public class ViewPlugin : IPlugin
{
 private bool enabled = true;
 public bool Enabled { get{return enabled;} set{enabled=value;} }
 public string Name => "View";

 //public bool colorize;
 //public bool relief;
 //public float min = 0;
 //public float max = 1;

 private float matrixMin;
 private float matrixMax;
```

```
private static bool lockScrollZoom = false;



public void Setup (Matrix matrix)
{
 matrixMin = matrix.MinValue();
 matrixMax = matrix.MaxValue();
}



public void DrawWindow (Matrix matrix, BaseMatrixWindow window) { }



public void DrawInspector (Matrix matrix, BaseMatrixWindow window)
{
 using (Cell.LineStd) Draw.ToggleLeft(ref window.colorize, "Colorize");
 using (Cell.LineStd) Draw.ToggleLeft(ref window.relief, "Relief");

 using (Cell.LinePx(0))
 {
  using (Cell.Row)
  {
   using (Cell.LineStd) Draw.Field(ref window.min, "Min");
   using (Cell.LineStd) Draw.Field(ref window.max, "Max");
  }
  using (Cell.RowPx(50))
```

```
  if (Draw.Button("Full")) { window.min = matrixMin; window.max = matrixMax; }

 using (Cell.RowPx(50))
  if (Draw.Button("0-1")) { window.min = 0; window.max = 1; }
}

Cell.EmptyLinePx(4);
using (Cell.LineStd)
{
 using (Cell.Row) Draw.Label("Zoom");
 using (Cell.RowPx(40))
  if (Draw.Button("0.25")) window.Zoom = 0.25f;
 using (Cell.RowPx(40))
  if (Draw.Button("0.50")) window.Zoom = 0.5f;
 using (Cell.RowPx(40))
  if (Draw.Button("1")) window.Zoom = 1f;
 using (Cell.RowPx(40))
  if (Draw.Button("2")) window.Zoom = 2f;
 using (Cell.RowPx(40))
  if (Draw.Button("4")) window.Zoom = 4f;

 if (Cell.current.valChanged)
  UI.current.editorWindow.Repaint();
}

using (Cell.LineStd)
```

```
  {
   using (Cell.Row) Draw.Label("Scroll");
   using (Cell.RowPx(60))
    if (Draw.Button("Zero"))
     window.Scroll = window.ScrollZero;
   using (Cell.RowPx(60))
    if (Draw.Button("Center"))
     window.Scroll = window.ScrollCenter;
  }

  using (Cell.LineStd)
  {
   using (Cell.LineStd) Draw.ToggleLeft(ref lockScrollZoom, "Lock All Windows Scroll/Zoom");

   if (lockScrollZoom)
    {
     MatrixWindow[] windows = Resources.FindObjectsOfTypeAll<MatrixWindow>();
     for (int w=0; w<windows.Length; w++)
     {
      windows[w].Scroll = window.Scroll;
      windows[w].Zoom = window.Zoom;
     }
    }
  }
 }
}
```

```
public class PixelPlugin : IPlugin
{
 public bool Enabled { get; set; }
 public string Name => "Pixel";

 private float curValue;
 private Coord curCoord;
 private bool curDefined;
 private Coord oldCurCoord;

 public void Setup (Matrix matrix) { }

 public void DrawWindow (Matrix matrix, BaseMatrixWindow window)
 {
//   DragMarkers();
//   for (int r=0; r<markerRects.Length; r++)
//    Draw.Rect(FlipVertical(markerRects[r]), new Color(1,0,0,0.25f));

  if (Event.current.button == 2)
  {
   curCoord = Coord.Floor(UI.current.mousePos.x, -UI.current.mousePos.y);
   if (matrix!=null && matrix.rect.Contains(curCoord))
   {
    curValue = matrix[curCoord];
```

```
    curDefined = true;
   }
   else
    curDefined = false;

   if (curCoord != oldCurCoord)
   {
    UI.current.editorWindow.Repaint();
    oldCurCoord = curCoord;
   }
  }
}

public void DrawInspector (Matrix matrix, BaseMatrixWindow window)
{
 Cell.EmptyLinePx(4);
 using (Cell.LineStd) Draw.DualLabel("Current Coordinate", curCoord.x + ", "+curCoord.z);
 using (Cell.LineStd) Draw.DualLabel("Current Value", curDefined ? curValue.ToString("0.0000f") : "Unde
 using (Cell.LineStd) Draw.Label("Middle-click to refresh");
}

private void DragMarkers ()
{
 Vector2 cursorRect = new Vector2(UI.current.mousePos.x, -UI.current.mousePos.y);

 if (DragDrop.TryDrag(Cell.current, cursorRect))
```

```
   {
    Rect rect = new Rect(DragDrop.initialMousePos, DragDrop.totalDelta);
    Draw.Rect(FlipVertical(rect), new Color(1,0,0,0.5f));
   }
   DragDrop.TryStart(Cell.current, cursorRect, Cell.current.GetRect());
   if (DragDrop.TryRelease(Cell.current, cursorRect))
   {
    Rect rect = new Rect(DragDrop.initialMousePos, DragDrop.totalDelta);
//    ArrayTools.Add(ref markerRects, rect);
   }
  }

  private static Rect FlipVertical (Rect rect)
  {
   rect.y = -rect.y - rect.height;
   return rect;
  }
 }



 public class SlicePlugin : IPlugin
 {
  public bool Enabled { get; set; }
  public string Name => "Slice";

  private MatrixOps.Stripe slice;
```

```
private Rect sliceRect;

public void Setup (Matrix matrix) { }

public void DrawWindow (Matrix matrix, BaseMatrixWindow window)
{
 DragSlice(matrix);
 Draw.Rect(FlipVertical(sliceRect), new Color(0,1,0,0.3f));
}

public void DrawInspector (Matrix matrix, BaseMatrixWindow window)
{
 using (Cell.LinePx(256))
 {
  if (slice != null && slice.arr != null && slice.arr.Length != 0)
   Draw.Histogram(slice.arr, new Vector4(0,0,0,0.5f), new Vector4(1,1,1,0.3f));
  Draw.Grid(new Color(0,0,0,0.4f), cellsNumX:1, cellsNumY:8);
 }
}

private void DragSlice (Matrix matrix)
{
 Vector2 cursorRect = new Vector2(UI.current.mousePos.x, -UI.current.mousePos.y);

 if (DragDrop.TryDrag(Cell.current, cursorRect))
 {
```

```
 Rect rect = new Rect(DragDrop.initialMousePos, DragDrop.totalDelta);
 rect = To1PixelRect(rect);

 Draw.Rect(FlipVertical(rect), new Color(0,1,0,0.5f));
}
DragDrop.TryStart(Cell.current, cursorRect, Cell.current.GetRect());
if (DragDrop.TryRelease(Cell.current, cursorRect))
{
 Rect rect = new Rect(DragDrop.initialMousePos, DragDrop.totalDelta);
 rect = To1PixelRect(rect);

 int x = (int)rect.x;     // + matrix.rect.offset.x;
 int z = (int)(rect.y+1); //ceil?  //)matrix.rect.size.z - (int)rect.y + matrix.rect.offset.z - 1;

 if (rect.width!=0 && rect.height!=0)
 {
  if (rect.width > rect.height)
  {
   //line = new float[(int)rect.width];
   slice = new MatrixOps.Stripe((int)rect.width);
   MatrixOps.ReadLine(slice, matrix, x, z);
  }
  else
  {
   //line = new float[(int)rect.height];
   slice = new MatrixOps.Stripe((int)rect.height);
```

```
    MatrixOps.ReadRow(slice, matrix, x, z-slice.length);
   }
   //{
   // slice = new Matrix.Stripe( Mathf.Max((int)rect.width, (int)rect.height) );
   // matrix.ReadDiagonal(stripe, x, z);
   //}

   sliceRect = rect;

   //MatrixLineTesterWindow lineTesterWindow = (MatrixLineTesterWindow)GetWindow(typeof (MatrixLine
   //lineTesterWindow.src = slice;
  }
 }
}

private Rect To1PixelRect (Rect rect)
{
 if (rect.width > rect.height)
 {
  rect.y += rect.height/2;
  rect.height = 1;//*scrollZoom.zoom;
 }
 else
 {
  rect.x += rect.width/2;
  rect.width = 1;//*scrollZoom.zoom;
```

```
  }

  if (rect.width > 256) rect.width = 256;
  if (rect.height > 256) rect.height = 256;

  return rect;
 }

 private static Rect FlipVertical (Rect rect)
 {
  rect.y = -rect.y - rect.height;
  return rect;
 }
}


public class ProcessPlugin : IPlugin
{
 public bool Enabled { get; set; }
 public string Name => "Process";

 public void Setup (Matrix matrix) { }

 public void DrawWindow (Matrix matrix, BaseMatrixWindow window) { }

 public void DrawInspector (Matrix matrix, BaseMatrixWindow basewindow)
```

```
{
 if (!(basewindow is MatrixWindow window) || matrix==null) return;

 using (Cell.LineStd)
 {
  using (Cell.RowPx(60))
   if (Draw.Button("1 - m")) matrix.InvertOne();

  using (Cell.RowPx(60))
   if (Draw.Button("m + 0.5")) matrix.Add(0.5f);

  using (Cell.RowPx(60))
   if (Draw.Button("m * 2")) matrix.Multiply(2);
 }

 using (Cell.LineStd)
 {
  using (Cell.RowPx(60))
   if (Draw.Button("-m")) matrix.Invert();

  using (Cell.RowPx(60))
   if (Draw.Button("m - 0.5")) matrix.Add(-0.5f);

  using (Cell.RowPx(60))
   if (Draw.Button("m / 2")) matrix.Multiply(0.5f);
 }
```

```
  if (Cell.current.valChanged)
    window.SetMatrix(matrix, window.name);
 }
}


public class ImportPlugin : IPlugin
{
 public bool Enabled { get; set; }
 public string Name => "Import";

 public Texture2D importTexture;
 public TerrainData importTerrain;
 public MatrixAsset importAsset;

 public void Setup (Matrix matrix) { }

 public void DrawWindow (Matrix matrix, BaseMatrixWindow window) { }

 public void DrawInspector (Matrix matrix, BaseMatrixWindow basewindow)
 {
  if (!(basewindow is MatrixWindow window)) return;

  using (Cell.LineStd)
  {
```

```
 using (Cell.RowPx(50)) Draw.Label("Asset");
 using (Cell.Row) Draw.ObjectField(ref importAsset, allowSceneObject:true);
 using (Cell.RowPx(60))
 {
  //if (Draw.Button("Export")  &&  exportAsset!=null)
 }



 Cell.EmptyRowPx(10);
 using (Cell.RowPx(60))
 {
  //if (Draw.Button("Save...")) {}
 }
}

using (Cell.LineStd)
{
 using (Cell.RowPx(50)) Draw.Label("Texture");
 using (Cell.Row) Draw.ObjectField(ref importTexture, allowSceneObject:true);
 using (Cell.RowPx(60))
  if (Draw.Button("Import")  &&  importTexture!=null)
  {
   TextureToMatrix(importTexture, ref matrix);
   window.SetMatrix(matrix, importTexture.name);
  }
 Cell.EmptyRowPx(10);
```

```
 using (Cell.RowPx(60))
  if (Draw.Button("Load..."))
  {
   Texture2D texture = ScriptableAssetExtensions.LoadAsset<Texture2D>();
   if (texture != null)
   {
    TextureToMatrix(texture, ref matrix);
    window.SetMatrix(matrix, texture.name);
   }
  }
}

using (Cell.LineStd)
{
 using (Cell.RowPx(50)) Draw.Label("Terrain");
 using (Cell.Row) Draw.ObjectField(ref importTerrain, allowSceneObject:true);
 using (Cell.RowPx(60))
  if (Draw.Button("Import")  &&  importTexture!=null)
  {
   TerrainToMatrix(importTerrain, ref matrix);
   window.SetMatrix(matrix, importTerrain.name);
  }
 Cell.EmptyRowPx(10);
 using (Cell.RowPx(60))
  if (Draw.Button("Load..."))
  {
```

```
    TerrainData terrain = ScriptableAssetExtensions.LoadAsset<TerrainData>();
    if (terrain != null)
    {
     TerrainToMatrix(terrain, ref matrix);
     window.SetMatrix(matrix, terrain.name);
    }
   }
 }
}


public static void TextureToMatrix (Texture2D tex, ref Matrix matrix)
{
 if (matrix == null || tex.width != matrix.rect.size.x || tex.height != matrix.rect.size.z)
  matrix = new Matrix( new CoordRect(0,0,tex.width, tex.height) );
 matrix.ImportTexture(tex, channel:0);
}


public static void TerrainToMatrix (TerrainData terrainData, ref Matrix matrix)
{
 int res = terrainData.heightmapResolution;
 if (matrix == null || res != matrix.rect.size.x || res != matrix.rect.size.z)
  matrix = new Matrix( new CoordRect(0,0,res,res) );
 matrix.ImportHeights(terrainData.GetHeights(0,0, terrainData.heightmapResolution, terrainData.heightma
}
```

```
}



public class ExportPlugin : IPlugin
{
 public bool Enabled { get; set; }
 public string Name => "Export";
 public int margins = 0;

 public Texture2D exportTexture;
 public TerrainData exportTerrain;
 public MatrixAsset exportAsset;

 public void Setup (Matrix matrix) { }

 public void DrawWindow (Matrix matrix, BaseMatrixWindow window) { }

 public void DrawInspector (Matrix matrix, BaseMatrixWindow window)
 {
  if (matrix==null) return;

  using (Cell.LineStd) Draw.Field(ref margins, "Margins");

  using (Cell.LineStd)
  {
   using (Cell.RowPx(50)) Draw.Label("Asset");
```

```
 using (Cell.Row) Draw.ObjectField(ref exportAsset, allowSceneObject:true);
 using (Cell.RowPx(60))
  if (Draw.Button("Save..."))
  {
   MatrixAsset matrixAsset = new MatrixAsset();
   matrixAsset.matrix = matrix;
   matrixAsset.RefreshPreview();
   ScriptableAssetExtensions.SaveAsset(matrixAsset);
  }


 Cell.EmptyRowPx(10);
 using (Cell.RowPx(60))
 {
  //if (Draw.Button("Save...")) {}
 }
}

using (Cell.LineStd)
{
 using (Cell.RowPx(50)) Draw.Label("Texture");
 using (Cell.Row) Draw.ObjectField(ref exportTexture, allowSceneObject:true);
 using (Cell.RowPx(60))
  if (Draw.Button("Export")  &&  exportTexture!=null)
   MatrixToTexture(matrix, ref exportTexture);
```

```
  Cell.EmptyRowPx(10);
  using (Cell.RowPx(60))
   if (Draw.Button("Save..."))
   {
    Texture2D texture = null;
    MatrixToTexture(matrix, ref texture);
    ScriptableAssetExtensions.SaveTexture(texture);
   }

 }

 using (Cell.LineStd)
 {
  using (Cell.RowPx(50)) Draw.Label("Terrain");
  using (Cell.Row) Draw.ObjectField(ref exportTerrain, allowSceneObject:true);
  using (Cell.RowPx(60))
   if (Draw.Button("Export")  &&  exportTerrain!=null)
    MatrixToTerrain(matrix, exportTerrain);

  Cell.EmptyRowPx(10);
  using (Cell.RowPx(60))
  {
   //if (Draw.Button("Save...")) {}
  }
 }
}
```

```
 public void MatrixToTexture (Matrix matrix, ref Texture2D tex)
 {
  CoordRect rect = matrix.rect.Expanded(-margins);

  if (tex == null || tex.width != rect.size.x || tex.height != rect.size.z)
  {
   GameObject.DestroyImmediate(tex);
   tex = new Texture2D(rect.size.x, rect.size.z, textureFormat:TextureFormat.RGB24, mipChain:false);
   tex.filterMode = FilterMode.Point;
  }

  matrix.ExportTextureRaw(tex, new Coord(-margins, -margins));
 }


 public void MatrixToTerrain (Matrix matrix, TerrainData terrainData)
 {
  float[,] heights = new float[matrix.rect.size.x, matrix.rect.size.z];
  matrix.ExportHeights(heights);
  terrainData.SetHeights(0,0,heights);
 }
}
```

```
}
```

```
ï»¿// Operations with "maps"
// Note that functions are not inlined in editor, so keeping all method unfolded

using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;

namespace Den.Tools
{
 [Serializable, StructLayout (LayoutKind.Sequential)] //to pass to native
 public class FloatMatrix : Matrix2D<float>
 {
  public float GetInterpolatedValue (Vector2 pos) //for upscaling - gets value in-between two points
  {
   int x = Mathf.FloorToInt(pos.x); int z = Mathf.FloorToInt(pos.y);
   float xPercent = pos.x-x; float zPercent = pos.y-z;

   //if (!rect.CheckInRange(x+1,z+1)) return 0;

   float val1 = this[x,z];
   float val2 = this[x+1,z];
   float val3 = val1*(1-xPercent) + val2*xPercent;

   float val4 = this[x,z+1];
```

```
 float val5 = this[x+1,z+1];
 float val6 = val4*(1-xPercent) + val5*xPercent;

 return val3*(1-zPercent) + val6*zPercent;
}

public float GetAveragedValue (int x, int z, int steps) //for downscaling
{
 float sum = 0;
 int div = 0;
 for (int ix=0; ix<steps; ix++)
  for (int iz=0; iz<steps; iz++)
 {
  if (x+ix >= rect.offset.x+rect.size.x) continue;
  if (z+iz >= rect.offset.z+rect.size.z) continue;
  sum += this[x+ix, z+iz];
  div++;
 }
 return sum / div;
}

#region Overriding constructors and clone

 public FloatMatrix () { arr = new float[0]; rect = new CoordRect(0,0,0,0); count = 0; } //for serializer

 public FloatMatrix (int offsetX, int offsetZ, int sizeX, int sizeZ, float[] array=null)
```

```
{
 this.rect = new CoordRect(offsetX, offsetZ, sizeX, sizeZ);;
 count = rect.size.x*rect.size.z;
 if (array != null && array.Length<count) Debug.Log("Array length: " + array.Length + " is lower then matri
 if (array != null && array.Length>=count) this.arr = array;
 else this.arr = new float[count];
}

public FloatMatrix (CoordRect rect, float[] array=null)
{
 this.rect = rect;
 count = rect.size.x*rect.size.z;
 if (array != null && array.Length<count) Debug.Log("Array length: " + array.Length + " is lower then matri
 if (array != null && array.Length>=count) this.arr = array;
 else this.arr = new float[count];
}

public FloatMatrix (Coord offset, Coord size, float[] array=null)
{
 rect = new CoordRect(offset, size);
 count = rect.size.x*rect.size.z;
 if (array != null && array.Length<count) Debug.Log("Array length: " + array.Length + " is lower then matri
 if (array != null && array.Length>=count) this.arr = array;
 else this.arr = new float[count];
}
```

```
public FloatMatrix (FloatMatrix src)
{
 rect = src.rect;
 count = src.count;

 arr = new float[src.arr.Length];
 Array.Copy(src.arr, arr, arr.Length);
}

public override object Clone () //IClonable
{
 FloatMatrix result = new FloatMatrix(rect);

 //copy params
 result.rect = rect;
 result.count = count;

 //copy array
 if (result.arr.Length != arr.Length) result.arr = new float[arr.Length];
 Array.Copy(arr, result.arr, arr.Length);

 return result;
}

[Obsolete] public FloatMatrix Copy (FloatMatrix result=null)
```

```
{
 if (result==null) result = new FloatMatrix(rect);

 //copy params
 result.rect = rect;
 result.count = count;

 //copy array
 if (result.arr.Length != arr.Length) result.arr = new float[arr.Length];
 Array.Copy(arr, result.arr, arr.Length);

 return result;
}

public FloatMatrix CopyRegion (CoordRect region, FloatMatrix result=null) //SOON: optimize
{
 if (result==null) result = new FloatMatrix(region);

 //copy params
 result.rect = region;
 result.count = region.Count;

 //copy array
 Coord min = region.Min; Coord max = region.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
```

```
  {
   if (x<rect.offset.x || x>rect.offset.x+rect.size.x ||
    z<rect.offset.z || z>rect.offset.z+rect.size.z) continue;
   result[x,z] = this[x,z];
  }

 return result;
}

#endregion

#region Texture (obsolete)

 [Obsolete]
 public Texture2D ToTexture ()
 {
  Texture2D texture = new Texture2D(rect.size.x, rect.size.z);
  WriteIntersectingTexture(texture, rect.offset.x, rect.offset.z);
  return texture;
 }

 [Obsolete]
 public void WriteIntersectingTexture (Texture2D texture, int textureOffsetX, int textureOffsetZ, float range
 ///Filling texture using both matrix and texture offsets. Filling only intersecting rect/texture are, leaving oth
 {
  //TODO: use LoadRawTextureData
```

```
 CoordRect textureRect = new CoordRect(textureOffsetX, textureOffsetZ, texture.width, texture.height);
 CoordRect intersect = CoordRect.Intersected(rect, textureRect);

 Color[] colors = new Color[intersect.size.x*intersect.size.z];

 Coord min = intersect.Min; Coord max = intersect.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
 {
  float val = this[x,z]; //TODO: direct
  val -= rangeMin;
  val /= rangeMax-rangeMin;

  colors[(z-min.z)*(max.x-min.x) + (x-min.x)] = new Color(val, val, val); //TODO: r should not be == r and =
 }

 texture.SetPixels(
  intersect.offset.x-textureOffsetX,
  intersect.offset.z-textureOffsetZ,
  intersect.size.x,
  intersect.size.z,
  colors);
 texture.Apply();
}
```

```
[Obsolete]
public void WriteTextureInterpolated (Texture2D texture, CoordRect textureRect, CoordRect.TileMode wr
{
 float pixelSizeX = 1f * textureRect.size.x / texture.width;
 float pixelSizeZ = 1f * textureRect.size.z / texture.height;

 Rect pixelTextureRect = new Rect(0, 0, texture.width, texture.height);
 Rect pixelMatrixRect = new Rect(
  (textureRect.offset.x - rect.offset.x) / pixelSizeX,
  (textureRect.offset.z - rect.offset.z) / pixelSizeZ,
  rect.size.x/pixelSizeX,
  rect.size.z/pixelSizeZ);

 Rect pixelIntersection = CoordinatesExtensions.Intersect(pixelTextureRect, pixelMatrixRect);

 CoordRect intersect = new CoordRect(
  Mathf.CeilToInt(pixelIntersection.x),
  Mathf.CeilToInt(pixelIntersection.y),
  Mathf.FloorToInt(pixelIntersection.width),
  Mathf.FloorToInt(pixelIntersection.height) );

 Color[] colors = new Color[intersect.size.x*intersect.size.z];

 Coord min = intersect.Min; Coord max = intersect.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
```

```
 {
  float wx = x*pixelSizeX - textureRect.offset.x + rect.offset.x*2;
  float wz = z*pixelSizeZ - textureRect.offset.z + rect.offset.z*2;

  //float val = this[x,z]; //TODO: direct
  float val = GetInterpolated(wx, wz);
  val -= rangeMin;
  val /= rangeMax-rangeMin;

  //val = 1;

  colors[(z-min.z)*(max.x-min.x) + (x-min.x)] = new Color(val, val, val); //TODO: r should not be == r and =
 }

 texture.SetPixels(intersect.offset.x, intersect.offset.z, intersect.size.x, intersect.size.z, colors);
 texture.Apply();
}

[Obsolete]
public Texture2D SimpleToTexture (Texture2D texture=null, Color[] colors=null, float rangeMin=0, float ra
{
 if (texture == null) texture = new Texture2D(rect.size.x, rect.size.z);
 if (texture.width != rect.size.x || texture.height != rect.size.z) texture.Resize(rect.size.x, rect.size.z);
 if (colors == null || colors.Length != rect.size.x*rect.size.z) colors = new Color[rect.size.x*rect.size.z];

 for (int i=0; i<count; i++)
```

```
 {
  float val = arr[i];
  val -= rangeMin;
  val /= rangeMax-rangeMin;
  colors[i] = new Color(val, val, val);
 }

 texture.SetPixels(colors);
 texture.Apply();
 return texture;
}

[Obsolete]
public static FloatMatrix SimpleFromTexture (Texture2D texture)
{
 Color[] colors = texture.GetPixels();
 FloatMatrix matrix = new FloatMatrix(0,0, texture.width, texture.height);
 for (int i=0; i<colors.Length; i++)
  matrix.arr[i] = colors[i].r;
 return matrix;
}


#endregion
```

```
#region Resize (obsolete)

 [Obsolete]
 public FloatMatrix ResizeOld (CoordRect newRect, FloatMatrix result=null)
 {
  if (result==null) result = new FloatMatrix(newRect);
  else result.ChangeRect(newRect);

  Coord min = result.rect.Min; Coord max = result.rect.Max;
  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
  {
   float percentX = 1f*(x-result.rect.offset.x)/result.rect.size.x; float origX = percentX*this.rect.size.x + this.r
   float percentZ = 1f*(z-result.rect.offset.z)/result.rect.size.z; float origZ = percentZ*this.rect.size.z + this.re
   result[x,z] = this.GetInterpolated(origX, origZ);
  }

  return result;
 }


 public enum Interpolation { None, Linear, Bicubic } //TODO: Biquadratic

 [Obsolete]
 public float GetInterpolatedOld (float x, float z, CoordRect.TileMode wrap=CoordRect.TileMode.Clamp)
 {
```

```
//skipping value if it is out of bounds
if (wrap==CoordRect.TileMode.Clamp)
{
 if (x<rect.offset.x || x>=rect.offset.x+rect.size.x || z<rect.offset.z || z>=rect.offset.z+rect.size.z)
  return 0;
}

//neig coords
int px = (int)x; if (x<0) px--; //because (int)-2.5 gives -2, should be -3
int nx = px+1;

int pz = (int)z; if (z<0) pz--;
int nz = pz+1;

//local coordinates (without offset)
int lpx = px-rect.offset.x; int lnx = nx-rect.offset.x;
int lpz = pz-rect.offset.z; int lnz = nz-rect.offset.z;

//wrapping coordinates
if (wrap==CoordRect.TileMode.Clamp)
{
 if (lpx<0) lpx=0; if (lpx>=rect.size.x) lpx=rect.size.x-1;
 if (lnx<0) lnx=0; if (lnx>=rect.size.x) lnx=rect.size.x-1;
 if (lpz<0) lpz=0; if (lpz>=rect.size.z) lpz=rect.size.z-1;
 if (lnz<0) lnz=0; if (lnz>=rect.size.z) lnz=rect.size.z-1;
}
```

```
else if (wrap==CoordRect.TileMode.Tile)
{
 lpx = lpx % rect.size.x; if (lpx<0) lpx=rect.size.x+lpx;
 lpz = lpz % rect.size.z; if (lpz<0) lpz=rect.size.z+lpz;
 lnx = lnx % rect.size.x; if (lnx<0) lnx=rect.size.x+lnx;
 lnz = lnz % rect.size.z; if (lnz<0) lnz=rect.size.z+lnz;
}
else if (wrap==CoordRect.TileMode.PingPong)
{
 lpx = lpx % (rect.size.x*2); if (lpx<0) lpx=rect.size.x*2 + lpx; if (lpx>=rect.size.x) lpx = rect.size.x*2 - lpx -
 lpz = lpz % (rect.size.z*2); if (lpz<0) lpz=rect.size.z*2 + lpz; if (lpz>=rect.size.z) lpz = rect.size.z*2 - lpz -
 lnx = lnx % (rect.size.x*2); if (lnx<0) lnx=rect.size.x*2 + lnx; if (lnx>=rect.size.x) lnx = rect.size.x*2 - lnx -
 lnz = lnz % (rect.size.z*2); if (lnz<0) lnz=rect.size.z*2 + lnz; if (lnz>=rect.size.z) lnz = rect.size.z*2 - lnz -
}

//reading values
float val_pxpz = arr[lpz*rect.size.x + lpx];
float val_nxpz = arr[lpz*rect.size.x + lnx]; //array[pos_fxfz + 1]; //do not use fast calculations as they are n
float val_pxnz = arr[lnz*rect.size.x + lpx]; //array[pos_fxfz + rect.size.z];
float val_nxnz = arr[lnz*rect.size.x + lnx]; //array[pos_fxfz + rect.size.z + 1];

float percentX = x-px;
float percentZ = z-pz;

float val_fz = val_pxpz*(1-percentX) + val_nxpz*percentX;
float val_cz = val_pxnz*(1-percentX) + val_nxnz*percentX;
```

```
 float val = val_fz*(1-percentZ) + val_cz*percentZ;


 return val;
}



[Obsolete]
public float GetInterpolatedBicubicOld (float x, float z, CoordRect.TileMode wrap=CoordRect.TileMode.C
/// Sorry for the unfolded code, it's editor method that does not support inlining
{
 //skipping value if it is out of bounds
 //if (wrap==CoordRect.TileMode.Once)
 //{
 // if (x<rect.offset.x || x>=rect.offset.x+rect.size.x || z<rect.offset.z || z>=rect.offset.z+rect.size.z)
 //  return 0;
 //}

 //neig coords
 int px = (int)x; if (x<0) px--; //because (int)-2.5 gives -2, should be -3
 int nx = px+1;
 int ppx = px-1;
 int nnx = nx+1;

 int pz = (int)z; if (z<0) pz--;
 int nz = pz+1;
 int ppz = pz-1;
```

```
int nnz = nz+1;

float percentX = x-px;
float percentZ = z-pz;
float invPercentX = 1-percentX;
float invPercentZ = 1-percentZ;
float sqPercentX = 3*percentX*percentX - 2*percentX*percentX*percentX;
float sqPercentZ = 3*percentZ*percentZ - 2*percentZ*percentZ*percentZ;

//local coordinates (without offset)
px = px-rect.offset.x; nx = nx-rect.offset.x;
pz = pz-rect.offset.z; nz = nz-rect.offset.z;
ppx = ppx-rect.offset.x; nnx = nnx-rect.offset.x;
ppz = ppz-rect.offset.z; nnz = nnz-rect.offset.z;

//wrapping coordinates
if (wrap==CoordRect.TileMode.Clamp)
{
 if (px<0) px=0; if (px>=rect.size.x) px=rect.size.x-1;
 if (nx<0) nx=0; if (nx>=rect.size.x) nx=rect.size.x-1;
 if (pz<0) pz=0; if (pz>=rect.size.z) pz=rect.size.z-1;
 if (nz<0) nz=0; if (nz>=rect.size.z) nz=rect.size.z-1;

 if (ppx<0) ppx=0; if (ppx>=rect.size.x) ppx=rect.size.x-1;
 if (nnx<0) nnx=0; if (nnx>=rect.size.x) nnx=rect.size.x-1;
 if (ppz<0) ppz=0; if (ppz>=rect.size.z) ppz=rect.size.z-1;
```

```
 if (nnz<0) nnz=0; if (nnz>=rect.size.z) nnz=rect.size.z-1;
}



float p = arr[pz*rect.size.x + ppx];
float pp = arr[ppz*rect.size.x + ppx];
float n = arr[nz*rect.size.x + ppx];
float nn = arr[nnz*rect.size.x + ppx];
float plvz = (2*p - pp*percentZ + n*percentZ) * 0.5f;
float nlvz = (2*n - nn*invPercentZ + p*invPercentZ  )*0.5f;
float ppvz = plvz*(1-sqPercentZ) + nlvz*(sqPercentZ);

p = arr[pz*rect.size.x + px];
pp = arr[ppz*rect.size.x + px];
n = arr[nz*rect.size.x + px];
nn = arr[nnz*rect.size.x + px];
plvz = (2*p - pp*percentZ + n*percentZ) * 0.5f;
nlvz = (2*n - nn*invPercentZ + p*invPercentZ  )*0.5f;
float pvz = plvz*(1-sqPercentZ) + nlvz*(sqPercentZ);

p = arr[pz*rect.size.x + nx];
pp = arr[ppz*rect.size.x + nx];
n = arr[nz*rect.size.x + nx];
nn = arr[nnz*rect.size.x + nx];
plvz = (2*p - pp*percentZ + n*percentZ) * 0.5f;
nlvz = (2*n - nn*invPercentZ + p*invPercentZ  )*0.5f;
```

```
 float nvz = plvz*(1-sqPercentZ) + nlvz*(sqPercentZ);

 p = arr[pz*rect.size.x + nnx];
 pp = arr[ppz*rect.size.x + nnx];
 n = arr[nz*rect.size.x + nnx];
 nn = arr[nnz*rect.size.x + nnx];
 plvz = (2*p - pp*percentZ + n*percentZ) * 0.5f;
 nlvz = (2*n - nn*invPercentZ + p*invPercentZ  )*0.5f;
 float nnvz = plvz*(1-sqPercentZ) + nlvz*(sqPercentZ);

 plvz = (2*pvz - ppvz*percentX + nvz*percentX) * 0.5f;
 nlvz = (2*nvz - nnvz*invPercentX + pvz*invPercentX  )*0.5f;
 return plvz*(1-sqPercentX) + nlvz*(sqPercentX);
}

 //public float GetInterpolatedTiled (float x, float z)



#endregion

#region Blur (obsolete)

 [Obsolete]
 public void Spread (float strength=0.5f, int iterations=4, FloatMatrix copy=null)
 {
```

```
Coord min = rect.Min; Coord max = rect.Max;

for (int j=0; j<count; j++) arr[j] = Mathf.Clamp(arr[j],-1,1);

if (copy==null) copy = Copy(null);
else for (int j=0; j<count; j++) copy.arr[j] = arr[j];

for (int i=0; i<iterations; i++)
{
 float prev = 0;

 for (int x=min.x; x<max.x; x++)
 {
  prev = this[x,min.z]; SetPos(x,min.z); for (int z=min.z+1; z<max.z; z++) { prev = (prev+arr[pos])/2; arr[po
  prev = this[x,max.z-1]; SetPos(x,max.z-1); for (int z=max.z-2; z>=min.z; z--) { prev = (prev+arr[pos])/2; a
 }

 for (int z=min.z; z<max.z; z++)
 {
  prev = this[min.x,z]; SetPos(min.x,z); for (int x=min.x+1; x<max.x; x++) { prev = (prev+arr[pos])/2; arr[po
  prev = this[max.x-1,z]; SetPos(max.x-1,z); for (int x=max.x-2; x>=min.x; x--) { prev = (prev+arr[pos])/2; a
 }
}

for (int j=0; j<count; j++) arr[j] = copy.arr[j] + arr[j]*2*strength;
```

```
 float factor = Mathf.Sqrt(iterations);
 for (int j=0; j<count; j++) arr[j] /= factor;
}


[Obsolete]
public void Spread (System.Func<float,float,float> spreadFn=null, int iterations=4)
{
 Coord min = rect.Min; Coord max = rect.Max;

 for (int i=0; i<iterations; i++)
 {
  float prev = 0;

  for (int x=min.x; x<max.x; x++)
  {
   prev = this[x,min.z]; SetPos(x,min.z); for (int z=min.z+1; z<max.z; z++) { prev = spreadFn(prev,arr[pos]
   prev = this[x,max.z-1]; SetPos(x,max.z-1); for (int z=max.z-2; z>=min.z; z--) { prev = spreadFn(prev,arr
  }

  for (int z=min.z; z<max.z; z++)
  {
   prev = this[min.x,z]; SetPos(min.x,z); for (int x=min.x+1; x<max.x; x++) { prev = spreadFn(prev,arr[pos]
   prev = this[max.x-1,z]; SetPos(max.x-1,z); for (int x=max.x-2; x>=min.x; x--) { prev = spreadFn(prev,arr
  }
 }
```

```
}

[Obsolete]
public void SimpleBlur (int iterations, float strength)
{
 Coord min = rect.Min; Coord max = rect.Max;

 for (int iteration=0; iteration<iterations; iteration++)
 {
  for (int z=min.z; z<max.z; z++)
  {
   float prev = this[min.x,z];
   for (int x=min.x+1; x<max.x-1; x++)
   {
    int i = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
    float curr = arr[i];
    float next = arr[i+1];

    float val = (prev+next)/2*strength + curr*(1-strength);
    arr[i] = val;
    prev = val;
   }
  }

  for (int x=min.x; x<max.x; x++)
  {
```

```
   float prev = this[x,min.z];
   for (int z=min.z+1; z<max.z-1; z++)
   {
    int i = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
    float curr = arr[i];
    float next = arr[i+rect.size.x];

    float val = (prev+next)/2*strength + curr*(1-strength);
    arr[i] = val;
    prev = val;
   }
  }
 }
}

[Obsolete]
public void Blur (System.Func<float,float,float,float> blurFn=null, float intensity=0.666f, bool additive=fals
{
 if (reference==null) reference = this;
 Coord min = rect.Min; Coord max = rect.Max;

 if (horizontal)
 for (int z=min.z; z<max.z; z++)
 {
  int pos = (z-rect.offset.z)*rect.size.x + min.x - rect.offset.x;
```

```
 float prev = reference[min.x,z];
 float curr = prev;
 float next = prev;

 float blurred = 0;

 for (int x=min.x; x<max.x; x++)
 {
  prev = curr; //reference[x-1,z];
  curr = next; //reference[x,z];
  if (x<max.x-1) next = reference.arr[pos+1]; //reference[x+1,z];

  //blurring
  if (blurFn==null) blurred = (prev+next)/2f;
  else blurred = blurFn(prev, curr, next);
  blurred = curr*(1-intensity) + blurred*intensity;

  //filling
  if (additive) arr[pos] += blurred;
  else arr[pos] = blurred;

  pos++;
 }
}

if (vertical)
```

```
for (int x=min.x; x<max.x; x++)
{
 int pos = (min.z-rect.offset.z)*rect.size.x + x - rect.offset.x;

 float next = reference[x,min.z];
 float curr = next;
 float prev = next;

 float blurred = next;

 for (int z=min.z; z<max.z; z++)
 {
  prev = curr; //reference[x-1,z];
  curr = next; //reference[x,z];
  if (z<max.z-1) next = reference.arr[pos+rect.size.x]; //reference[x+1,z];

  //blurring
  if (blurFn==null) blurred = (prev+next)/2f;
  else blurred = blurFn(prev, curr, next);
  blurred = curr*(1-intensity) + blurred*intensity;

  //filling
  if (additive) arr[pos] += blurred;
  else if (takemax) { if (blurred > arr[pos]) arr[pos] = blurred; }
  else arr[pos] = blurred;
```

```
   pos+=rect.size.x;
  }
 }
}

[Obsolete]
public void LossBlur (int step=2, bool horizontal=true, bool vertical=true, FloatMatrix reference=null)
{
 if (reference==null) reference = this;
 Coord min = rect.Min; Coord max = rect.Max;
 int stepShift = step + step/2;

 if (horizontal)
 for (int z=min.z; z<max.z; z++)
 {
  SetPos(min.x, z);

  float sum = 0;
  int div = 0;

  float avg = this.arr[pos];
  float oldAvg = this.arr[pos];

  for (int x=min.x; x<max.x+stepShift; x++)
  {
   //gathering
```

```
  if (x < max.x) sum += reference.arr[pos];
  div ++;
  if (x%step == 0)
  {
   oldAvg=avg;
   if (x < max.x) avg=sum/div;
   sum=0; div=0;
  }

  //filling
  if (x-stepShift >= min.x)
  {
   float percent = 1f*(x%step)/step;
   if (percent<0) percent += 1; //for negative x
   this.arr[pos-stepShift] = avg*percent + oldAvg*(1-percent);
  }

  pos += 1;
 }
}

if (vertical)
for (int x=min.x; x<max.x; x++)
{
 SetPos(x, min.z);
```

```
float sum = 0;
int div = 0;

float avg = this.arr[pos];
float oldAvg = this.arr[pos];

for (int z=min.z; z<max.z+stepShift; z++)
{
 //gathering
 if (z < max.z) sum += reference.arr[pos];
 div ++;
 if (z%step == 0)
 {
  oldAvg=avg;
  if (z < max.z) avg=sum/div;
  sum=0; div=0;
 }

 //filling
 if (z-stepShift >= min.z)
 {
  float percent = 1f*(z%step)/step;
  if (percent<0) percent += 1;
  this.arr[pos-stepShift*rect.size.x] = avg*percent + oldAvg*(1-percent);
 }
```

```
    pos += rect.size.x;
   }
  }
 }

 #region Outdated
/*public void OverBlur (int iterations=20)
{
 Matrix blurred = this.Clone(null);

 for (int i=1; i<=iterations; i++)
 {
  if (i==1 || i==2) blurred.Blur(step:1);
  else if (i==3) { blurred.Blur(step:1); blurred.Blur(step:1); }
  else blurred.Blur(step:i-2); //i:4, step:2

  for (int p=0; p<count; p++)
  {
   float b = blurred.array[p] * i;
   float a = array[p];

   array[p] = a + b + a*b;
  }
 }
}*/
```

```
/*public void LossBlur (System.Func<float,float,float,float> blurFn=null, //prev, curr, next = output
 float intensity=0.666f, int step=1, Matrix reference=null, bool horizontal=true, bool vertical=true)
{
 Coord min = rect.Min; Coord max = rect.Max;

 if (reference==null) reference = this;
 int lastX = max.x-1;
 int lastZ = max.z-1;

 if (horizontal)
 for (int z=min.z; z<=lastZ; z++)
 {
  float next = reference[min.x,z];
  float curr = next;
  float prev = next;

  float blurred = next;
  float lastBlurred = next;

  for (int x=min.x+step; x<=lastX; x+=step)
  {
   //blurring
   if (blurFn==null) blurred = (prev+next)/2f;
   else blurred = blurFn(prev, curr, next);
   blurred = curr*(1-intensity) + blurred*intensity;
```

```
  //shifting values
  prev = curr; //this[x,z];
  curr = next; //this[x+step,z];
  try { next = reference[x+step*2,z]; } //this[x+step*2,z];
  catch { next = reference[lastX,z]; }

  //filling between-steps distance
  if (step==1) this[x,z] = blurred;
  else for (int i=0; i<step; i++)
  {
   float percent = 1f * i / step;
   this[x-step+i,z] = blurred*percent + lastBlurred*(1-percent);
  }
  lastBlurred = blurred;
 }
}

if (vertical)
for (int x=min.x; x<=lastX; x++)
{
 float next = reference[x,min.z];
 float curr = next;
 float prev = next;

 float blurred = next;
 float lastBlurred = next;
```

```
  for (int z=min.z+step; z<=lastZ; z+=step)
  {
   //blurring
   if (blurFn==null) blurred = (prev+next)/2f;
   else blurred = blurFn(prev, curr, next);
   blurred = curr*(1-intensity) + blurred*intensity;

   //shifting values
   prev = curr;
   curr = next;
   try { next = reference[x,z+step*2]; }
   catch { next = reference[x,lastZ]; }

   //filling between-steps distance
   if (step==1) this[x,z] = blurred;
   else for (int i=0; i<step; i++)
   {
    float percent = 1f * i / step;
    this[x,z-step+i] = blurred*percent + lastBlurred*(1-percent);
   }
   lastBlurred = blurred;
  }
 }
}*/
#endregion
```

```
#endregion



#region Other

 static public void BlendLayers (FloatMatrix[] matrices, float[] opacity=null)
 /// Changes splatmaps in photoshop layered style so their summary value does not exceed 1
 {
  //finding any existing matrix
  int anyMatrixNum = -1;
  for (int i=0; i<matrices.Length; i++)
   if (matrices[i]!=null) { anyMatrixNum = i; break; }
  if (anyMatrixNum == -1) { Debug.LogError("No matrices were found to blend " + matrices.Length); return


  //finding rect
  CoordRect rect = matrices[anyMatrixNum].rect;


  //checking rect size
  #if WDEBUG
  for (int i=0; i<matrices.Length; i++)
   if (matrices[i]!=null && matrices[i].rect!=rect) { Debug.LogError("Matrix rect mismatch " + rect + " " + mat
  #endif


  int rectCount = rect.Count;
  for (int pos=0; pos<rectCount; pos++)
```

```
{
 float left = 1;
 for (int i=matrices.Length-1; i>=0; i--) //layer 0 is background, layer Length-1 is the top one
 {
  if (matrices[i] == null) continue;

  float val = matrices[i].arr[pos];

  if (opacity != null) val *= opacity[i];

  val = val * left;
  matrices[i].arr[pos] = val;
  left -= val;

  if (left < 0) break;


  /*float overly = sum + val - 1;
  if (overly < 0) overly = 0; //faster then calling Math.Clamp
  if (overly > 1) overly = 1;

  matrices[i].array[pos] = val - overly;
  sum += val - overly;*/
 }
}
}
```

```
static public void NormalizeLayers (FloatMatrix[] matrices, float[] opacity)
/// Changes splatmaps so their summary value does not exceed 1
{
 //finding any existing matrix
 int anyMatrixNum = -1;
 for (int i=0; i<matrices.Length; i++)
  if (matrices[i]!=null) { anyMatrixNum = i; break; }
 if (anyMatrixNum == -1) { Debug.LogError("No matrices were found to blend " + matrices.Length); return

 //finding rect
 CoordRect rect = matrices[anyMatrixNum].rect;

 //checking rect size
 #if WDEBUG
 for (int i=0; i<matrices.Length; i++)
  if (matrices[i]!=null && matrices[i].rect!=rect) { Debug.LogError("Matrix rect mismatch " + rect + " " + mat
 #endif


 int rectCount = rect.Count;
 for (int pos=0; pos<rectCount; pos++)
 {
  for (int i=0; i<matrices.Length; i++) matrices[i].arr[pos] *= opacity[i];

  float sum = 0;
```

```
  for (int i=0; i<matrices.Length; i++) sum += matrices[i].arr[pos];
  if (sum > 1f) for (int i=0; i<matrices.Length; i++) matrices[i].arr[pos] /= sum;
 }
}



static public void Blend (FloatMatrix src, FloatMatrix dst, float factor)
{
 if (dst.rect != src.rect) Debug.LogError("Matrix Blend: maps have different sizes");

 for (int i=0; i<dst.count; i++)
 {
  dst.arr[i] = dst.arr[i]*factor + src.arr[i]*(1-factor);
 }
}

static public void Mask (FloatMatrix src, FloatMatrix dst, FloatMatrix mask) //changes dst, not src
{
 if (src != null &&
  (dst.rect != src.rect || dst.rect != mask.rect)) Debug.LogError("Matrix Mask: maps have different sizes");

 for (int i=0; i<dst.count; i++)
 {
  float percent = mask.arr[i];
  if (percent > 1 || percent < 0) continue;
```

```
   dst.arr[i] = dst.arr[i]*percent + (src==null? 0:src.arr[i]*(1-percent));
  }
 }

 static public void SafeBorders (FloatMatrix src, FloatMatrix dst, int safeBorders) //changes dst, not src
 {
  Coord min = dst.rect.Min; Coord max = dst.rect.Max;
  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
  {
   int distFromBorder = Mathf.Min( Mathf.Min(x-min.x,max.x-x), Mathf.Min(z-min.z,max.z-z) );
   float percent = 1f*distFromBorder / safeBorders;
   if (percent > 1) continue;

   dst[x,z] = dst[x,z]*percent + (src==null? 0:src[x,z]*(1-percent));
  }
 }

#endregion


#region Histogram

 public float[] Histogram (int resolution, float max=1, bool normalize=true)
 /// Evaluates all pixels in matrix and returns an array of pixels count by their value.
 {
```

```
 float[] quants = new float[resolution];


 for (int i=0; i<count; i++)
 {
  float val = arr[i];
  float percent = val / max;
  int num = (int)(percent*resolution);
  if (num==resolution) num--; //this could happen when val==max

  quants[num]++;
 }

 if (normalize)
 {
  float maxQuant = 0;

  for (int i=0; i<resolution; i++)
   if (quants[i] > maxQuant) maxQuant = quants[i];

  for (int i=0; i<resolution; i++)
   quants[i] /= maxQuant;
 }

 return quants;
}
```

```
public float[] Slice (Coord coord, int length, bool vertical)
/// A single line (or row) of a matrix to show a vertical slice
{
 if (vertical) return SliceVertical(coord, length);
 else return SliceHorizontal(coord, length);
}


public float[] SliceHorizontal (Coord coord, int length)
{
 if (coord.z < rect.offset.z || coord.z >= rect.offset.z+rect.size.z) return new float[0];

 if (coord.x < rect.offset.x) { length -= rect.offset.x-coord.x; coord.x = rect.offset.x; }
 if (length <= 0) return new float[0];
 int max = coord.x + length;
 if (max >= rect.offset.x+rect.size.x) length = rect.offset.x+rect.size.x - coord.x;

 MatrixLine line = new MatrixLine(coord.x-rect.offset.x, length);
 line.ReadLine(this, coord.z);
 return line.arr;
}


public float[] SliceVertical (Coord coord, int length)
{
```

```
 if (coord.x < rect.offset.x || coord.x >= rect.offset.x+rect.size.x) return new float[0];

 if (coord.z < rect.offset.z) { length -= rect.offset.z-coord.z; coord.x = rect.offset.z; }
 if (length <= 0) return new float[0];
 int max = coord.z + length;
 if (max >= rect.offset.z+rect.size.z) length = rect.offset.z+rect.size.z - coord.z;

 MatrixLine row = new MatrixLine(coord.z-rect.offset.z, length);
 row.ReadLine(this, coord.x);
 return row.arr;
}


public static byte[] HistogramToTextureBytes (float[] quants, int height, byte empty=0, byte top=255, byte
/// Converts an array from Histogram to texture bytes
{
 int width = quants.Length;
 byte[] bytes = new byte[width * height];

 for (int x=0; x<width; x++)
 {
  int max = (int)(quants[x] * height);
  if (max==height) max--;

  for (int z=0; z<height; z++)
  {
```

```
   byte val = empty;
   if (z==max) val=top;
   else if (z<max) val=filled;

   bytes[z*width + x] = val;
  }
 }

 return bytes;
}

public static Texture2D HistogramToTextureR8 (float[] quants, int height, byte empty=0, byte top=255, by
/// Converts an array from Histogram to texture (format R8)
{
 byte[] bytes = HistogramToTextureBytes(quants, height, empty, top, filled);

 Texture2D tex = new Texture2D(quants.Length, height, TextureFormat.R8, false, linear:true);
 tex.LoadRawTextureData(bytes);
 tex.Apply(updateMipmaps:false);

 return tex;
}

[Obsolete("Use static method instead")] public byte[] HistogramTexture (int width, int height, byte empty=
{
 float[] quants = Histogram(width);
```

```
 byte[] bytes = HistogramToTextureBytes(quants, height, empty, top, filled);
 return bytes;
}

#endregion


#region Import

 public void ImportTexture (Texture2D tex, int channel, bool useRaw=true) { ImportTexture(tex, rect.offset
 public void ImportTexture (Texture2D tex, Coord texOffset, int channel, bool useRaw=true)
 {
  Coord texSize = new Coord(tex.width, tex.height);

  //raw bytes
  TextureFormat format = tex.format;
  if (useRaw && (format==TextureFormat.RGBA32 || format==TextureFormat.ARGB32 || format==Texture
  {
   byte[] bytes = tex.GetRawTextureData();
   switch(format)
   {
    case TextureFormat.RGBA32: ImportRaw(bytes, texOffset, texSize, channel, 4); break;
    case TextureFormat.ARGB32: channel++; if (channel == 5) channel = 0; ImportRaw(bytes, texOffset, te
    case TextureFormat.RGB24: ImportRaw(bytes, texOffset, texSize, channel, 3); break;
    case TextureFormat.R8: ImportRaw(bytes, texOffset, texSize, 0, 1); break;
    case TextureFormat.R16: ImportRaw16(bytes, texOffset, texSize); break;
```

```
 }
}

//colors
else
{
 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(texOffset,texSize)); //to get array

 Color[] colors = tex.GetPixels(intersection.offset.x-texOffset.x, intersection.offset.z-texOffset.z, intersect
 ImportColors(colors, intersection.offset, intersection.size, channel);
}

tex.Apply();
}


public void ImportColors (Color[] colors, int width, int height, int channel) { ImportColors(colors, rect.offset
public void ImportColors (Color[] colors, Coord colorsSize, int channel) { ImportColors(colors, rect.offset,
public void ImportColors (Color[] colors, Coord colorsOffset, Coord colorsSize,  int channel)
{
 if (colors.Length != colorsSize.x*colorsSize.z)
  throw new Exception("Array count does not match texture dimensions");

 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(colorsOffset, colorsSize));
 Coord min = intersection.Min; Coord max = intersection.Max;
```

```
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int colorsPos = (z-colorsOffset.z)*colorsSize.x + x - colorsOffset.x;

   float val;
   switch (channel)
   {
    case 0: val = colors[colorsPos].r; break;
    case 1: val = colors[colorsPos].g; break;
    case 2: val = colors[colorsPos].b; break;
    case 3: val = colors[colorsPos].a; break;
    default: val = colors[colorsPos].r; break; //(colors[colorsPos].r + colors[colorsPos].g + colors[colorsPos]
   }

   arr[matrixPos] = val;
  }
}


public void ImportRaw (byte[] bytes, int width, int height, int start, int step) { ImportRaw(bytes, new Coord
public void ImportRaw (byte[] bytes, Coord bytesSize, int start, int step) { ImportRaw(bytes, bytesSize, re
public void ImportRaw (byte[] bytes, Coord bytesOffset, Coord bytesSize, int start, int step)
{
 if (bytes.Length != bytesSize.x*bytesSize.z*step &&
```

```
  (bytes.Length < bytesSize.x*bytesSize.z*step*1.3f || bytes.Length > bytesSize.x*bytesSize.z*step*1.366
   throw new Exception("Array count does not match texture dimensions");

 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(bytesOffset, bytesSize));
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int bytesPos = (z-bytesOffset.z)*bytesSize.x + x - bytesOffset.x;
   bytesPos = bytesPos * step + start;

   float val = bytes[bytesPos] / 255f;  //matrix has the range 0-1 _inclusive_, it could be 1, so using 255
   arr[matrixPos] = val;
  }
}



public void ImportRaw16 (byte[] bytes, int width, int height) { ImportRaw16(bytes, new Coord(width,heigh
public void ImportRaw16 (byte[] bytes, Coord texSize) { ImportRaw16(bytes, texSize, rect.offset); }
public void ImportRaw16 (byte[] bytes, Coord texOffset, Coord texSize)
{
 if (texSize.x*texSize.z*2 != bytes.Length)
  throw new Exception("Array count does not match texture dimensions");
```

```
 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(texOffset, texSize));
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int bytesPos = (z-texOffset.z)*texSize.x + x - texOffset.x;
   bytesPos *= 2;

   float val = (bytes[bytesPos+1]*255f + bytes[bytesPos]) / 65025f;
   arr[matrixPos] = val;
  }
}


public void ImportHeights (float[,] heights) { ImportHeights(heights, rect.offset); }
public void ImportHeights (float[,] heights, Coord heightsOffset)
{
 Coord heightsSize = new Coord(heights.GetLength(1), heights.GetLength(0));  //x and z swapped
 CoordRect heightsRect = new CoordRect(heightsOffset, heightsSize);

 CoordRect intersection = CoordRect.Intersected(rect, heightsRect);
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
```

```
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int heightsPosZ = x - heightsRect.offset.x;
   int heightsPosX = z - heightsRect.offset.z;

   arr[matrixPos] = heights[heightsPosX, heightsPosZ];
  }
 }



public void ImportSplats (float[,,] splats, int channel) { ImportSplats(splats, rect.offset, channel); }
public void ImportSplats (float[,,] splats, Coord splatsOffset, int channel)
{
 Coord splatsSize = new Coord(splats.GetLength(1), splats.GetLength(0));  //x and z swapped
 CoordRect splatsRect = new CoordRect(splatsOffset, splatsSize);

 CoordRect intersection = CoordRect.Intersected(rect, splatsRect);
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int heightsPosZ = x - splatsRect.offset.x;
   int heightsPosX = z - splatsRect.offset.z;
```

```
    arr[matrixPos] = splats[heightsPosX, heightsPosZ, channel];
   }
}
```

```
public void ImportData (TerrainData data, int channel=-1) { ImportData (data, rect.offset, channel=-1); }
public void ImportData (TerrainData data, Coord dataOffset, int channel=-1)
/// Partial terrain data (loading only the part intersecting with matrix). Do not work in thread!
/// If channel is -1 getting height
{
 int resolution = channel==-1 ?  data.heightmapResolution : data.alphamapResolution;

 Coord dataSize = new Coord(resolution, resolution);
 CoordRect dataIntersection = CoordRect.Intersected(rect, new CoordRect(dataOffset, dataSize));
 if (dataIntersection.size.x==0 || dataIntersection.size.z==0) return;

 if (channel == -1)
 {
  float[,] heights = data.GetHeights(dataIntersection.offset.x-dataOffset.x, dataIntersection.offset.z-dataOf
  ImportHeights(heights, dataIntersection.offset);
 }

 else
 {
  float[,,] splats = data.GetAlphamaps(dataIntersection.offset.x-dataOffset.x, dataIntersection.offset.z-data
```

```
   ImportSplats(splats, dataIntersection.offset, channel);
  }
 }

#endregion



#region Export

 public void ExportTexture (Texture2D tex, int channel, bool useRaw=true) { ExportTexture(tex, rect.offset
 public void ExportTexture (Texture2D tex, Coord texOffset, int channel, bool useRaw=true)
 {
  Coord texSize = new Coord(tex.width, tex.height);

  //raw bytes
  TextureFormat format = tex.format;
  if (useRaw && (format==TextureFormat.RGBA32 || format==TextureFormat.ARGB32 || format==Texture
  {
   byte[] bytes = tex.GetRawTextureData();
   switch(format)
   {
    case TextureFormat.RGBA32:
     //bytes = new byte[texSize.x*texSize.z*4];
     ExportRaw(bytes, texOffset, texSize, channel, 4); break;

    case TextureFormat.ARGB32:
```

```
   //bytes = new byte[texSize.x*texSize.z*4];
   channel++; if (channel == 5) channel = 0;
   ExportRaw(bytes, texOffset, texSize, channel, 4); break;

  case TextureFormat.RGB24:
   //bytes = new byte[texSize.x*texSize.z*3];
   ExportRaw(bytes, texOffset, texSize, channel, 3); break;

  case TextureFormat.R8:
   //bytes = new byte[texSize.x*texSize.z];
   ExportRaw(bytes, texOffset, texSize, 0, 1); break;

  case TextureFormat.R16:
   //bytes = new byte[texSize.x*texSize.z*2];
   ExportRaw16(bytes, texOffset, texSize); break;
 }

 tex.LoadRawTextureData(bytes);
}

//colors
else
{
 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(texOffset,texSize)); //to get array

 //Color[] colors = tex.GetPixels(intersection.offset.x-texOffset.x, intersection.offset.z-texOffset.z, intersec
```

```
  Color[] colors = new Color[intersection.size.x * intersection.size.z];
  ExportColors(colors, intersection.offset, intersection.size, channel);
  tex.SetPixels(intersection.offset.x-texOffset.x, intersection.offset.z-texOffset.z, intersection.size.x, inters
 }

 tex.Apply();
}



public void ExportColors (Color[] colors, int width, int height, int channel) { ExportColors(colors, rect.offse
public void ExportColors (Color[] colors, Coord colorsSize, int channel) { ExportColors(colors, rect.offset,
public void ExportColors (Color[] colors, Coord colorsOffset, Coord colorsSize,  int channel)
{
 if (colors.Length != colorsSize.x*colorsSize.z)
  throw new Exception("Array count does not match texture dimensions");

 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(colorsOffset, colorsSize));
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int colorsPos = (z-colorsOffset.z)*colorsSize.x + x - colorsOffset.x;

   float val = arr[matrixPos];
```

```
   switch (channel)
   {
    case 0: colors[colorsPos].r = val; break;
    case 1: colors[colorsPos].g = val; break;
    case 2: colors[colorsPos].b = val; break;
    case 3: colors[colorsPos].a = val; break;
    default: colors[colorsPos].r = val; break; //(colors[colorsPos].r + colors[colorsPos].g + colors[colorsPos]
   }
  }
}



public void ExportRaw (byte[] bytes, int width, int height, int start, int step) { ExportRaw(bytes, new Coord
public void ExportRaw (byte[] bytes, Coord bytesSize, int start, int step) { ExportRaw(bytes, bytesSize, re
public void ExportRaw (byte[] bytes, Coord bytesOffset, Coord bytesSize, int start, int step)
{
 if (bytes.Length != bytesSize.x*bytesSize.z*step &&
  (bytes.Length < bytesSize.x*bytesSize.z*step*1.3f || bytes.Length > bytesSize.x*bytesSize.z*step*1.366
   throw new Exception("Array count does not match texture dimensions");

 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(bytesOffset, bytesSize));
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
```

```
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int bytesPos = (z-bytesOffset.z)*bytesSize.x + x - bytesOffset.x;
   bytesPos = bytesPos * step + start;

   float val = arr[matrixPos];
   bytes[bytesPos] = (byte)(val * 255f); //matrix has the range 0-1 _inclusive_, it could be 1
  }
}



public void ExportRaw16 (byte[] bytes, int width, int height) { ExportRaw16(bytes, new Coord(width,heigh
public void ExportRaw16 (byte[] bytes, Coord texSize) { ExportRaw16(bytes, texSize, rect.offset); }
public void ExportRaw16 (byte[] bytes, Coord texOffset, Coord texSize)
{
 if (texSize.x*texSize.z*2 != bytes.Length)
  throw new Exception("Array count does not match texture dimensions");

 CoordRect intersection = CoordRect.Intersected(rect, new CoordRect(texOffset, texSize));
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int bytesPos = (z-texOffset.z)*texSize.x + x - texOffset.x;
   bytesPos *= 2;
```

```
    float val = arr[matrixPos]; //this[x+regionRect.offset.x, z+regionRect.offset.z];

    int intVal = (int)(val*65025);
    byte hb = (byte)(intVal/255f);
    bytes[bytesPos+1] = hb;  //TODO: test if the same will work on macs with non-inverted byte order
    bytes[bytesPos] = (byte)(intVal-hb*255);
   }
  }

public void ExportHeights (float[,] heights) { ExportHeights(heights, rect.offset); }
public void ExportHeights (float[,] heights, Coord heightsOffset)
{
 Coord heightsSize = new Coord(heights.GetLength(1), heights.GetLength(0));  //x and z swapped
 CoordRect heightsRect = new CoordRect(heightsOffset, heightsSize);

 CoordRect intersection = CoordRect.Intersected(rect, heightsRect);
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int heightsPosZ = x - heightsRect.offset.x;
   int heightsPosX = z - heightsRect.offset.z;
```

```
   float val = arr[matrixPos];
   heights[heightsPosX, heightsPosZ] = val;
  }
 }

public void ExportSplats (float[,,] splats, int channel) { ExportSplats(splats, rect.offset, channel); }
public void ExportSplats (float[,,] splats, Coord splatsOffset, int channel)
{
 Coord splatsSize = new Coord(splats.GetLength(1), splats.GetLength(0));  //x and z swapped
 CoordRect splatsRect = new CoordRect(splatsOffset, splatsSize);

 CoordRect intersection = CoordRect.Intersected(rect, splatsRect);
 Coord min = intersection.Min; Coord max = intersection.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int matrixPos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
   int heightsPosZ = x - splatsRect.offset.x;
   int heightsPosX = z - splatsRect.offset.z;

   float val = arr[matrixPos];
   splats[heightsPosX, heightsPosZ, channel] = val;
  }
```

```
}


//partial terrain data (loading only the part intersecting with matrix). Do not work in thread!
public void ExportData (TerrainData data) { ExportData (data, rect.offset, -1); }
public void ExportData (TerrainData data, int channel) { ExportData (data, rect.offset, channel); }
public void ExportData (TerrainData data, Coord dataOffset, int channel) //if channel is -1 getting height
{
 int resolution = channel==-1 ?  data.heightmapResolution : data.alphamapResolution;

 Coord dataSize = new Coord(resolution, resolution);
 CoordRect dataIntersection = CoordRect.Intersected(rect, new CoordRect(dataOffset, dataSize));
 if (dataIntersection.size.x==0 || dataIntersection.size.z==0) return;

 if (channel == -1)
 {
  float[,] heights = new float[dataIntersection.size.z, dataIntersection.size.x];  //x and z swapped
  ExportHeights(heights, dataIntersection.offset);
  data.SetHeights(dataIntersection.offset.x-dataOffset.x, dataIntersection.offset.z-dataOffset.z, heights);
 }

 else
 {
  float[,,] splats = data.GetAlphamaps(dataIntersection.offset.x-dataOffset.x, dataIntersection.offset.z-data
  ExportSplats(splats, dataIntersection.offset, channel);
  data.SetAlphamaps(dataIntersection.offset.x-dataOffset.x, dataIntersection.offset.z-dataOffset.z, splats)
```

```
 }
}

#endregion



#region Stamp

 public void Stamp (float centerX, float centerZ, float radius, float hardness, FloatMatrix stamp)
 /// Applies stamp to this matrix, blending it in a smooth circular way using radius and hardness
 /// All values are in pixels and using matrix offset
 /// Center does not need to be the real center, it's just used to calculate fallof
 /// Hardness is the percent (0-1) of the stamp that has 100% fallof
 /// Invert fills all matrix except the center area
 /// Tested
 {
  CoordRect intersection = CoordRect.Intersected(rect, stamp.rect);
  Coord min = intersection.Min; Coord max = intersection.Max;

  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    float dist = Mathf.Sqrt((x-centerX)*(x-centerX) + (z-centerZ)*(z-centerZ));
    if (dist > radius) continue;

    int pos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;
```

```
   int stampPos = (z-stamp.rect.offset.z)*stamp.rect.size.x + x - stamp.rect.offset.x;

   if (dist < radius*hardness) { arr[pos] = stamp.arr[stampPos]; continue; }

   float fallof = (radius - dist) / (radius - radius*hardness); //linear yet
   if (fallof>1) fallof = 1; if (fallof<0) fallof = 0;
   fallof = 3*fallof*fallof - 2*fallof*fallof*fallof;

   arr[pos] = arr[pos]*(1-fallof) + stamp.arr[stampPos]*fallof;
  }
}

public void Stamp (float centerX, float centerZ, float radius, float hardness, float value)
/// Applies value stamp to this matrix. Same as above, but using uniform value
{
 Coord min = rect.Min; Coord max = rect.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   float dist = Mathf.Sqrt((x-centerX)*(x-centerX) + (z-centerZ)*(z-centerZ));
   if (dist > radius) continue;

   int pos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;

   if (dist < radius*hardness) { arr[pos] = value; continue; }
```

```
   float fallof = (radius - dist) / (radius - radius*hardness); //linear
   if (fallof>1) fallof = 1; if (fallof<0) fallof = 0;
   fallof = 3*fallof*fallof - 2*fallof*fallof*fallof;

   arr[pos] = arr[pos]*(1-fallof) + value*fallof;
  }
}

public void StampInverted (float centerX, float centerZ, float radius, float hardness, float value)
/// Applies value to all of the matrix except the area in the center. Changes from the original version by it's
{
 Coord min = rect.Min; Coord max = rect.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   int pos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;

   float dist = Mathf.Sqrt((x-centerX)*(x-centerX) + (z-centerZ)*(z-centerZ));
   if (dist > radius) { arr[pos] = value; continue; }

   if (dist < radius*hardness) continue; //differs from the original one

   float fallof = (radius - dist) / (radius - radius*hardness); //linear
   if (fallof>1) fallof = 1; if (fallof<0) fallof = 0;
```

```
    fallof = 3*fallof*fallof - 2*fallof*fallof*fallof;

    arr[pos] = arr[pos]*fallof + value*(1-fallof); //differs here too
   }
 }

#endregion


#region Arithmetic (per-pixel operation that does not involve neighbor pixels)

 public void Mix (FloatMatrix m, float opacity=1)
 {
  float invOpacity = 1-opacity;
  for (int i=0; i<count; i++)
   arr[i] = m.arr[i]*opacity + arr[i]*invOpacity;
 }

 public void Add (FloatMatrix add, float opacity=1)
 {
  for (int i=0; i<count; i++)
   arr[i] += add.arr[i] * opacity;
 }

 public void Add (FloatMatrix add, FloatMatrix mask, float opcaity=1)
 {
```

```
 for (int i=0; i<count; i++)
  arr[i] += add.arr[i] * mask.arr[i] * opcaity;
}

public void Add (float add)
{
 for (int i=0; i<count; i++)
  arr[i] += add;
}

public void Subtract (FloatMatrix m, float opacity=1)
{
 for (int i=0; i<count; i++)
  arr[i] -= m.arr[i] * opacity;
}

public void InvSubtract (FloatMatrix m, float opacity=1)
/// subtracting this matrix from m
{
 for (int i=0; i<count; i++)
  arr[i] = m.arr[i]*opacity - arr[i];
}

public void Multiply (FloatMatrix m, float opacity=1)
{
 float invOpacity = 1-opacity;
```

```
 for (int i=0; i<count; i++)
  arr[i] *= m.arr[i]*opacity + invOpacity;
}

public void Multiply (float m)
{
 for (int i=0; i<count; i++)
  arr[i] *= m;
}

public void Sqrt ()
{
 for (int i=0; i<count; i++)
  arr[i] = Mathf.Sqrt(arr[i]);
}

public void Fallof ()
{
 for (int i=0; i<count; i++)
 {
  float val = arr[i];
  arr[i] = 3*val*val - 2*val*val*val;
 }
}

public void Contrast (float m)
```

```
/// Mid-matrix (value 0.5) multiply
{
 for (int i=0; i<count; i++)
 {
  float val = arr[i]*2 -1;
  val *= m;
  arr[i] = (val+1) / 2;
 }
}

public void Divide (FloatMatrix m, float opacity=1)
{
 float invOpacity = 1-opacity;
 for (int i=0; i<count; i++)
  arr[i] *= opacity/m.arr[i] + invOpacity;
}

public void Difference (FloatMatrix m, float opacity=1)
{
 for (int i=0; i<count; i++)
 {
  float val = arr[i] - m.arr[i]*opacity;
  if (val < 0) val = -val;
  arr[i] = val;
 }
}
```

```
public void Overlay (FloatMatrix m, float opacity=1)
{
 for (int i=0; i<count; i++)
 {
  float a = arr[i];
  float b = m.arr[i];

  b = b*opacity + (0.5f - opacity/2); //enhancing contrast via levels

  if (a > 0.5f) b = 1 - 2*(1-a)*(1-b);
  else b = 2*a*b;

  arr[i] = b;// b*opacity + a*(1-opacity); //the same
 }
}

public void HardLight (FloatMatrix m, float opacity=1)
/// Same as overlay but estimating b>0.5
{
 for (int i=0; i<count; i++)
 {
  float a = arr[i];
  float b = m.arr[i];

  if (b > 0.5f) b = 1 - 2*(1-a)*(1-b);
```

```
  else b = 2*a*b;


  arr[i] = b*opacity + a*(1-opacity);
 }
}

public void SoftLight (FloatMatrix m, float opacity=1)
{
 for (int i=0; i<count; i++)
 {
  float a = arr[i];
  float b = m.arr[i];
  b = (1-2*b)*a*a + 2*b*a;
  arr[i] = b*opacity + a*(1-opacity);
 }
}

public void Max (FloatMatrix m, float opacity=1)
{
 for (int i=0; i<count; i++)
 {
  float val = m.arr[i]>arr[i] ? m.arr[i] : arr[i];
  arr[i] = val*opacity + arr[i]*(1-opacity);
 }
}
```

```
public void Min (FloatMatrix m, float opacity=1)
{
 for (int i=0; i<count; i++)
 {
  float val = m.arr[i]<arr[i] ? m.arr[i] : arr[i];
  arr[i] = val*opacity + arr[i]*(1-opacity);
 }
}

public new void Fill(float val)
{
 for (int i=0; i<count; i++)
  arr[i] = val;
}

public void Fill (FloatMatrix m)
{
 m.arr.CopyTo(arr,0);
}

public void Invert()
{
 for (int i=0; i<count; i++)
  arr[i] = -arr[i];
}
```

```
public void InvertOne()
{
 for (int i=0; i<count; i++)
  arr[i] = 1-arr[i];
}

public void SelectRange (float min0, float min1, float max0, float max1)
/// Fill all values within min1-max0 with 1, while min0-1 and max0-1 are filled with blended
{
 for (int i=0; i<arr.Length; i++)
 {
  float delta = arr[i];

  //if (steepness.x<0.0001f) array[i] = 1-(delta-max0)/(max1-max0);
  //else
  {
   float minVal = (delta-min0)/(min1-min0);
   float maxVal = 1-(delta-max0)/(max1-max0);
   float val = minVal>maxVal? maxVal : minVal;
   if (val<0) val=0; if (val>1) val=1;

   arr[i] = val;
  }
 }
}
```

```
public void Clamp01 ()
{
 for (int i=0; i<count; i++)
 {
  float val = arr[i];
  if (val > 1) arr[i] = 1;
  else if (val < 0) arr[i] = 0;
 }
}

public void ToRange01 ()
{
 for (int i=0; i<count; i++)
  arr[i] = (arr[i]+1) / 2;
}

public float MaxValue ()
{
 float max=-20000000;
 for (int i=0; i<count; i++)
 {
  float val = arr[i];
  if (val > max) max = val;
 }
 return max;
}
```

```
public float MinValue ()
{
 float min=20000000;
 for (int i=0; i<count; i++)
 {
  float val = arr[i];
  if (val < min) min = val;
 }
 return min;
}

public virtual bool IsEmpty ()
/// Better than MinValue since it can quit if matrix is not empty
{
 for (int i=0; i<count; i++)
  if (arr[i] > 0.0001f) return false;
 return true;
}

public virtual bool IsEmpty (float delta)
{
 for (int i=0; i<count; i++)
  if (arr[i] > delta) return false;
 return true;
}
```

```
public void BlackWhite (float mid)
/// Sets all values bigger than mid to white (1), and those lower to black (0)
{
 for (int i=0; i<count; i++)
 {
  float val = arr[i];
  if (val > mid) arr[i] = 1;
  else arr[i] = 0;
 }
}

public void Terrace (float[] terraces, float steepness, float intensity)
{
 for (int i=0; i<count; i++)
 {
  float val = arr[i];
  if (val > 0.999f) continue; //do nothing with values that are out of range

  int terrNum = 0;
  for (int t=0; t<terraces.Length-1; t++)
  {
   if (terraces[terrNum+1] > val || terrNum+1 == terraces.Length) break;
   terrNum++;
  }
```

```
  //kinda curve evaluation
  float delta = terraces[terrNum+1] - terraces[terrNum];
  float relativePos = (val - terraces[terrNum]) / delta;

  float percent = 3*relativePos*relativePos - 2*relativePos*relativePos*relativePos;

  percent = (percent-0.5f)*2;
  bool minus = percent<0; percent = Mathf.Abs(percent);

  percent = Mathf.Pow(percent,1f-steepness);

  if (minus) percent = -percent;
  percent = percent/2 + 0.5f;

  arr[i] = (terraces[terrNum]*(1-percent) + terraces[terrNum+1]*percent)*intensity + arr[i]*(1-intensity);
 }
}

public void Levels (Vector2 min, Vector2 max, float gamma)
{
 for (int i=0; i<count; i++)
 {
  float val = arr[i];

  if (val < min.x) { arr[i] = 0; continue; }
  if (val > max.x) { arr[i] = 1; continue; }
```

```
  val = 1 * ( ( val - min.x ) / ( max.x - min.x ) );

  if (gamma != 1) // (gamma>1.00001f || gamma<0.9999f)
  {
   if (gamma<1) val = Mathf.Pow(val, gamma);
   else val = Mathf.Pow(val, 1/(2-gamma));
  }

  arr[i] = val;
 }
}

public void ReadMatrix (FloatMatrix src, CoordRect.TileMode tileMode = CoordRect.TileMode.Clamp)
{
 Coord min = rect.Min; Coord max = rect.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
   Coord tiledCoord = src.rect.Tile(new Coord(x,z), tileMode);
   this[x,z] = src[tiledCoord];
  }
}
```

```
 /*public Matrix FastDownscaled (int ratio)
 {
  CoordRect downRect = new CoordRect( rect.offset, new Coord(rect.size.x/ratio, rect.size.z/ratio) );
  Matrix downMatrix = new Matrix(downRect);

  Coord min = downRect.Min; Coord max = downRect.Max;
  for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
   {
    float avgVal = 0;
    for (int sx=0; sx<ratio; sx++)
     for (int sz=0; sz<ratio; sz++)
      avgVal += arr[(z*ratio + sz) * rect.size.x  +  (x*ratio + sx)]; //this[x*ratio + sx, z*ratio + sz];
    avgVal /= ratio*ratio;

    downMatrix[x,z] = avgVal;
   }

  return downMatrix;
 }*/

#endregion



#region Simple Conversions
```

```
/// No offset is used in all the Simple Conversions
// TODO: use arrRect

public void ReadArray (float[,] arr2D)
{
 int maxX = rect.size.x; if (arr2D.GetLength(1) < maxX) maxX = arr2D.GetLength(1);
 int maxZ = rect.size.z; if (arr2D.GetLength(0) < maxZ) maxZ = arr2D.GetLength(0);

 for (int x=0; x<maxX; x++)
  for (int z=0; z<maxZ; z++)
   arr[z*rect.size.x + x] = arr2D[z,x];
}

public void ApplyArray (float[,] arr2D)
{
 int maxX = rect.size.x; if (arr2D.GetLength(1) < maxX) maxX = arr2D.GetLength(1);
 int maxZ = rect.size.z; if (arr2D.GetLength(0) < maxZ) maxZ = arr2D.GetLength(0);

 for (int x=0; x<maxX; x++)
  for (int z=0; z<maxZ; z++)
   arr2D[z,x] = arr[z*rect.size.x + x];
}

#endregion
```

```
#region Blend Algorithms

 public enum BlendAlgorithm {
  mix=0,
  add=1,
  subtract=2,
  multiply=3,
  divide=4,
  difference=5,
  min=6,
  max=7,
  overlay=8,
  hardLight=9,
  softLight=10}

 //not using const arrays or dictionaries for native compatibility

 public void Blend (FloatMatrix m, BlendAlgorithm algorithm, float opacity=1)
 {
  switch (algorithm)
  {
   case BlendAlgorithm.mix: default: Mix(m, opacity); break;
   case BlendAlgorithm.add: Add(m, opacity); break;
   case BlendAlgorithm.subtract: Subtract(m, opacity); break;
   case BlendAlgorithm.multiply: Multiply(m, opacity); break;
```

```
  case BlendAlgorithm.divide: Divide(m, opacity); break;
  case BlendAlgorithm.difference: Difference(m, opacity); break;
  case BlendAlgorithm.min: Min(m, opacity); break;
  case BlendAlgorithm.max: Max(m, opacity); break;
  case BlendAlgorithm.overlay: Overlay(m, opacity); break;
  case BlendAlgorithm.hardLight: HardLight(m, opacity); break;
  case BlendAlgorithm.softLight: SoftLight(m, opacity); break;
 }
}

public static System.Func<float,float,float> GetBlendAlgorithm (BlendAlgorithm algorithm)
{
 switch (algorithm)
 {
  case BlendAlgorithm.mix: return delegate (float a, float b) { return b; };
  case BlendAlgorithm.add: return delegate (float a, float b) { return a+b; };
  case BlendAlgorithm.subtract: return delegate (float a, float b) { return a-b; };
  case BlendAlgorithm.multiply: return delegate (float a, float b) { return a*b; };
  case BlendAlgorithm.divide: return delegate (float a, float b) { return a/b; };
  case BlendAlgorithm.difference: return delegate (float a, float b) { return Mathf.Abs(a-b); };
  case BlendAlgorithm.min: return delegate (float a, float b) { return Mathf.Min(a,b); };
  case BlendAlgorithm.max: return delegate (float a, float b) { return Mathf.Max(a,b); };
  case BlendAlgorithm.overlay: return delegate (float a, float b)
  {
   if (a > 0.5f) return 1 - 2*(1-a)*(1-b);
   else return 2*a*b;
```

```
  };
  case BlendAlgorithm.hardLight: return delegate (float a, float b)
  {
    if (b > 0.5f) return 1 - 2*(1-a)*(1-b);
    else return 2*a*b;
  };
  case BlendAlgorithm.softLight: return delegate (float a, float b) { return (1-2*b)*a*a + 2*b*a; };
  default: return delegate (float a, float b) { return b; };
 }
}

#endregion



#region Convert

 public void WriteTexture (Texture2D texture, CoordRect regionRect = new CoordRect())
 /// Converts matrix to texture (with rescale if needed). Will crop matrix to regionRect (if specified) before c
 {
  //rescaling if needed
  FloatMatrix src = this;
  if (texture.width != regionRect.size.x || texture.height != regionRect.size.z)
   src = Resized(new Coord(texture.width, texture.height), regionRect);

  Color[] colors = new Color[texture.width * texture.height];
  for (int i=0; i<colors.Length; i++)
```

```
 {
  float val = src.arr[i];
  colors[i] = new Color(val, val, val, 1);
 }

 texture.SetPixels(colors);
 texture.Apply();
}



public byte[] ToRawBytes (CoordRect regionRect = new CoordRect())
{
 if (regionRect.size.x==0 && regionRect.size.z==0)
  regionRect = rect;
 Coord min = regionRect.Min; Coord max = regionRect.Max;

 byte[] bytes = new byte[regionRect.size.x * regionRect.size.z * 2];

 int bytePos = 0;
 for (int z=0; z<regionRect.size.z; z++)
 {
  int matrixRowStart = (z+regionRect.offset.z-rect.offset.z)*rect.size.x + regionRect.offset.x-rect.offset.x;
  for (int x=0; x<regionRect.size.x; x++)
  {
   int matrixPos = matrixRowStart + x;
   float val = arr[matrixPos]; //this[x+regionRect.offset.x, z+regionRect.offset.z];
```

```
   if (val > 1) val = 1;


   int intVal = (int)(val*65025);
   byte hb = (byte)(intVal/255f);
   bytes[bytePos+1] = hb;  //TODO: test if the same will work on macs with non-inverted byte order
   bytes[bytePos] = (byte)(intVal-hb*255);
   bytePos+=2;
  }
 }

 return bytes;
}



public void ReadRawBytes (byte[] bytes, CoordRect regionRect = new CoordRect())
{
 if (regionRect.size.x==0 && regionRect.size.z==0)
  regionRect = rect;
 Coord min = regionRect.Min; Coord max = regionRect.Max;

 int bytePos = 0;
 for (int z=0; z<regionRect.size.z; z++)
 {
  int matrixRowStart = (z+regionRect.offset.z-rect.offset.z)*rect.size.x + regionRect.offset.x-rect.offset.x;
  for (int x=0; x<regionRect.size.x; x++)
  {
```

```
    float val = (bytes[bytePos+1]*256f + bytes[bytePos]) / 65025f;
    bytePos+=2;

    arr[matrixRowStart + x] = val;
   }
  }
 }

#endregion


#region Line Operations (operate with neighbor pixel per-line/row)

 public FloatMatrix Resized (CoordRect newRect, CoordRect regionRect = new CoordRect())
 {
  FloatMatrix newMatrix = Resized(newRect.size, regionRect);
  newMatrix.rect.offset = newRect.offset;
  return newMatrix;
 }

 public FloatMatrix Resized (Coord newSize, CoordRect regionRect = new CoordRect())
 /// Returns the new matrix of given dstRect size. Will read only srcRect if specified. Downscaling linear, u
 {
  FloatMatrix dst;
```

```
 if (regionRect.size.x == newSize.x  &&  regionRect.size.z == newSize.z)  // returning clone if both sizes r
  dst = CopyRegion(regionRect);

 else if (rect.size.z == newSize.z) //if vertical size match
  dst = ResizedHorizontally(newSize.x, regionRect);

 else if (rect.size.x == newSize.x) //if horizontal size match
  dst = ResizedVertically(newSize.z, regionRect);

 else
 {
  FloatMatrix intermediate = ResizedHorizontally(newSize.x, regionRect);
  dst = intermediate.ResizedVertically(newSize.z);
 }

 return dst;
}

public FloatMatrix ResizedHorizontally (int newWidth, CoordRect regionRect = new CoordRect())
/// Will resize matrix only horizontally. Offset will not change.
{
 if (regionRect.size.x==0 && regionRect.size.z==0)
  regionRect = rect;
 Coord min = regionRect.Min; Coord max = regionRect.Max;

 FloatMatrix result = new FloatMatrix( new CoordRect(regionRect.offset.x, regionRect.offset.z, newWidth
```

```
 MatrixLine src = new MatrixLine(regionRect.offset.x, regionRect.size.x);
 MatrixLine dst = new MatrixLine(regionRect.offset.x, newWidth);

 for (int z=min.z; z<max.z; z++)
 {
  src.ReadLine(this, z); //srcStart.x);

  if (dst.length > src.length) MatrixLine.ResampleCubic(src,dst);
  else MatrixLine.ResampleLinear(src,dst);

  dst.WriteLine(result, z);
 }

 return result;
}


public FloatMatrix ResizedVertically (int newHeight, CoordRect regionRect = new CoordRect())
/// Will resize matrix only horizontally
{
 if (regionRect.size.x==0 && regionRect.size.z==0)
  regionRect = rect;
 Coord min = regionRect.Min; Coord max = regionRect.Max;

 FloatMatrix result = new FloatMatrix( new CoordRect(regionRect.offset.x, regionRect.offset.z, regionRec
```

```
 MatrixLine src = new MatrixLine(regionRect.offset.z, rect.size.z);
 MatrixLine dst = new MatrixLine(regionRect.offset.z, newHeight);

 for (int x=min.x; x<max.x; x++)
 {
  src.ReadRow(this, x);

  if (dst.length > src.length) MatrixLine.ResampleCubic(src,dst);
  else MatrixLine.ResampleLinear(src,dst);

  dst.WriteRow(result, x);
 }

 return result;
}


public FloatMatrix FastDownscaled (int ratio, CoordRect regionRect = new CoordRect())
{
 if (regionRect.size.x==0 && regionRect.size.z==0)
  regionRect = rect;
 Coord min = regionRect.Min; Coord max = regionRect.Max;

 CoordRect intermediateRect = new CoordRect( regionRect.offset, new Coord(regionRect.size.x/ratio, re
 FloatMatrix intermediate = new FloatMatrix(intermediateRect);
```

```
 MatrixLine srcH = new MatrixLine(regionRect.offset.x, regionRect.size.x);
 MatrixLine dstH = new MatrixLine(regionRect.offset.x, intermediateRect.size.x);
 for (int z=min.z; z<max.z; z++)
 {
  srcH.ReadLine(this, z);
  MatrixLine.DownsampleFast(srcH,dstH,ratio);
  dstH.WriteLine(intermediate, z);
 }


 CoordRect downRect = new CoordRect( regionRect.offset, new Coord(regionRect.size.x/ratio, regionRe
 FloatMatrix downMatrix = new FloatMatrix(downRect);

 MatrixLine srcV = new MatrixLine(regionRect.offset.z, intermediateRect.size.z);
 MatrixLine dstV = new MatrixLine(regionRect.offset.z, downRect.size.z);
 for (int x=min.x; x<min.x+intermediateRect.size.x; x++)
 {
  srcV.ReadRow(intermediate, x);
  MatrixLine.DownsampleFast(srcV,dstV,ratio);
  dstV.WriteRow(downMatrix, x);
 }

 return downMatrix;
}
```

```
public FloatMatrix Cavity (float intensity)
{
 FloatMatrix dstMatrix = new FloatMatrix(rect);
 Coord min = rect.Min; Coord max = rect.Max;

 MatrixLine line = new MatrixLine(rect.offset.x, rect.size.x);
 for (int z=min.z; z<max.z; z++)
 {
  line.ReadLine(this,z);
  line.Cavity(intensity);
  line.WriteLine(dstMatrix, z);
 }

 line = new MatrixLine(rect.offset.z, rect.size.z);
 for (int x=min.x; x<max.x; x++)
 {
  line.ReadRow(this, x);
  line.Cavity(intensity);

  //apply row additively (with mid-point 0.5)
  line.Add(-0.5f);
  line.AppendRow(dstMatrix, x);
 }

 return dstMatrix;
}
```

```
public FloatMatrix NormalRelief (float horizontalHeight=1, float verticalHeight=1)
{
 FloatMatrix dstMatrix = new FloatMatrix(rect);
 Coord min = rect.Min; Coord max = rect.Max;

 MatrixLine line = new MatrixLine(rect.offset.x, rect.size.x);
 for (int z=min.z; z<max.z; z++)
 {
  line.ReadLine(this,z);
  line.Normal(horizontalHeight);
  line.WriteLine(dstMatrix, z);
 }

 line = new MatrixLine(rect.offset.z, rect.size.z);
 for (int x=min.x; x<max.x; x++)
 {
  line.ReadRow(this, x);
  line.Normal(verticalHeight);
  line.AppendRow(dstMatrix, x);
 }

 dstMatrix.Multiply(0.5f); //each line has range 0-1, and they are combined additively

 return dstMatrix;
}
```

```
public FloatMatrix Delta ()
/// For each pixel it evaluates 4 neighbors and sets maximum value delta
{
 FloatMatrix dstMatrix = new FloatMatrix(rect);
 Coord min = rect.Min; Coord max = rect.Max;

 MatrixLine line = new MatrixLine(rect.offset.x, rect.size.x);
 for (int z=min.z; z<max.z; z++)
 {
  line.ReadLine(this,z);
  line.Delta();
  line.WriteLine(dstMatrix, z);
 }

 MatrixLine src = new MatrixLine(rect.offset.z, rect.size.z);
 MatrixLine dst = new MatrixLine(rect.offset.z, rect.size.z);
 for (int x=min.x; x<max.x; x++)
 {
  src.ReadRow(this, x);
  src.Delta();

  dst.ReadRow(dstMatrix, x); //reading dst row to compare with horizontal delta
  src.Max(dst);
```

```
  src.WriteRow(dstMatrix, x);
 }

 return dstMatrix;
}

public void Spread (int iterations, float subtract=0.01f, float multiply=1f)
{
 FloatMatrix dstMatrix = new FloatMatrix(this);
 Coord min = rect.Min; Coord max = rect.Max;

 for (int i=0; i<iterations; i++)
 {
  dstMatrix.Mix(this, 0.5f);

  MatrixLine line = new MatrixLine(rect.offset.x, rect.size.x);
  for (int z=min.z; z<max.z; z++)
  {
   line.ReadLine(dstMatrix, z);
   line.Spread(subtract, multiply);
   line.WriteLine(dstMatrix, z);
  }

  //dstMatrix.GaussianBlur(1);

  line = new MatrixLine(rect.offset.z, rect.size.z);
```

```
 for (int x=min.x; x<max.x; x++)
 {
  line.ReadRow(dstMatrix, x);
  line.Spread(subtract, multiply);
  line.WriteRow(dstMatrix, x);
 }

 //dstMatrix.GaussianBlur(1);
}

 arr = dstMatrix.arr;
}



public void GaussianBlur (float blur)
{
 //float[] arr = new float[rect.size.x];
 //float[] tmp = new float[rect.size.x];
 Coord min = rect.Min; Coord max = rect.Max;

 MatrixLine line = new MatrixLine(rect.offset.x, rect.size.x);
 float[] temp = new float[rect.size.x];

 for (int z=min.z; z<max.z; z++)
 {
  line.ReadLine(this, z);
```

```
  line.GaussianBlur(temp,blur);
  line.WriteLine(this, z);
 }

 line = new MatrixLine(rect.offset.z, rect.size.z);
 temp = new float[rect.size.z];

 for (int x=min.x; x<max.x; x++)
 {
  line.ReadRow(this, x);
  line.GaussianBlur(temp,blur);
  line.WriteRow(this, x);
 }
}


public void DownsampleBlur (int downsample, float blur)
{
 int downsamplePot = (int)Mathf.Pow(2,downsample-1);
 Coord min = rect.Min; Coord max = rect.Max;

 MatrixLine hiLine = new MatrixLine(rect.offset.x, rect.size.x);
 MatrixLine loLine = new MatrixLine(rect.offset.x, rect.size.x / downsample);
 float[] tmp = new float[rect.size.x / downsample];

 for (int z=min.z; z<max.z; z++)
```

```
 {
  hiLine.ReadLine(this, z);

  MatrixLine.ResampleLinear(hiLine, loLine);
  loLine.GaussianBlur(tmp, blur);
  MatrixLine.ResampleCubic(loLine, hiLine);

  hiLine.WriteLine(this, z);
 }

 hiLine = new MatrixLine(rect.offset.z, rect.size.z);
 loLine = new MatrixLine(rect.offset.z, rect.size.z / downsample);
 tmp = new float[rect.size.z / downsample];

 for (int x=min.x; x<max.x; x++)
 {
  hiLine.ReadRow(this, x);

  MatrixLine.ResampleLinear(hiLine, loLine);
  loLine.GaussianBlur(tmp, blur);
  MatrixLine.ResampleCubic(loLine, hiLine);

  hiLine.WriteRow(this, x);
 }
}
```

```
public void DownsampleOverblur (int downsample, float blur=1, float fallof=2)
{
 FloatMatrix mip = this;
 Clamp01();

 for (int i=0; i<downsample; i++)
 {
  if (mip.rect.size.x/2 == 0) break; //already at the lowest level

  mip = mip.Resized( new Coord( mip.rect.size.x/2, mip.rect.size.z/2) ); //downscaling to mip matrix
  mip.GaussianBlur(blur);
  mip.Contrast(fallof);

  FloatMatrix blurred = mip.Resized( this.rect.size );

  //float contrast = i*fallof + sharpness;

  for (int a=0; a<arr.Length; a++)
  {
   float valA = arr[a];
   float valB = blurred.arr[a];

   //apply overlay
   if (valA<0.5f) valA = 2*valA*valB;
   else valA = 1 - 2*(1-valA)*(1-valB);
```

```
    //clamp 0-1 preventing over-value in next iterations
    if (valA > 1) valA = 1;
    if (valB < 0) valB = 0;

    arr[a] = valA;
   }
  }
 }

public void ExtendCircular (Coord center, int radius, int extendRange, int predictEvaluateRange)
{
 //resetting area out of radius
 StampInverted(center.x, center.z, radius, 1, -Mathf.Epsilon);

 //creating radial lines
 int numLines = Mathf.CeilToInt( Mathf.PI * radius ); //using only the half of the needed lines
 float angleStep = Mathf.PI * 2 / numLines; //in radians

 MatrixLine line = new MatrixLine(0, Mathf.CeilToInt(extendRange + predictEvaluateRange + 1));

 for (int i=0; i<numLines; i++)
 {
  float angle = i*angleStep;
  Vector2 direction = new Vector2( Mathf.Sin(angle), Mathf.Cos(angle) );
  Vector2 start = center.vector2 + direction*(radius-predictEvaluateRange);
```

```
 //making any of the step components equal to 1
 Vector2 posDir = new Vector2 (
  (direction.x>0 ? direction.x : -direction.x),
  (direction.y>0 ? direction.y : -direction.y) );
 float max = posDir.x>posDir.y ? posDir.x : posDir.y;
 Vector2 step = direction / max;
 int predictStart = (int)(predictEvaluateRange * max);

 line.ReadInclined(this, start, step);
 line.PredictExtend(predictStart);
 line.WriteInclined(this, start, step);
}

//filling gaps between lines
line = new MatrixLine(0, (int)(radius+extendRange+1)*2*4);

for (int i=(int)(radius*0.7f); i<radius+extendRange; i++)
{
 line.ReadSquare(this, center, i);
 line.FillGaps(0, i*2*4);
 line.WriteSquare(this, center, i);
}

//blurring circular
/*float[] tmp = new float[line.length];
```

```
 Line loLine = new Line(0, line.length);


 for (int i=(int)radius; i<radius+extendRange; i++)
 {
  line.ReadSquare(this, cCenter, i);


  loLine.length = line.length / 16;


  //Line.DownsampleFast(line, loLine, 16);
  for (int j=0; j<16; j++)
   line.GaussianBlur(tmp, 1f);
  //Line.ResampleCubic(loLine, line);


  line.WriteSquare(this, cCenter, i);
 }*/
}



public void SpreadBlurCircular (Coord center, float radius, int blur=2)
/// Spread-blurs the circular stamp stroke at the center position (does not need to at matrix center) after th
{
 SpreadBlurCircularCorner(center, radius, blur:blur);
 SpreadBlurCircularCorner(center, radius, blur:blur, bottom:true);
 SpreadBlurCircularCorner(center, radius, blur:blur, left:true);
 SpreadBlurCircularCorner(center, radius, blur:blur, left:true, bottom:true);
}
```

```
private void SpreadBlurCircularCorner (Coord center, float radius, int blur=2, bool left=false, bool bottom=
/// Circular spread blur iteration, it takes one corner relatively to center only
{
 FloatMatrix src = (FloatMatrix)Clone();

 //the maximum possible rect (includes center)
 CoordRect maxRect = rect;
 maxRect.Encapsulate(center);

 //upper left corner
 CoordRect cornerRect = new CoordRect (center.x, center.z, maxRect.size.x, maxRect.size.z); //creating
 if (left) cornerRect.offset.x -= maxRect.size.x;
 if (bottom) cornerRect.offset.z -= maxRect.size.z;
 cornerRect = CoordRect.Intersected(rect, cornerRect);

 if (cornerRect.size.x ==0 || cornerRect.size.z == 0) return;

 Coord min = cornerRect.Min; Coord max = cornerRect.Max;

 Vector2 gradientStart = (center.vector2 - min.vector2) / radius; //the value at which the rect's gradient sta
 Vector2 gradientEnd = (center.vector2 - max.vector2) / radius;
 if (!left) { gradientStart.x = - gradientStart.x;  gradientEnd.x = - gradientEnd.x; }
 if (!bottom) { gradientStart.y = - gradientStart.y;  gradientEnd.y = - gradientEnd.y; }
```

```
MatrixLine currLine = new MatrixLine(cornerRect.offset.x, cornerRect.size.x);
MatrixLine prevLine = new MatrixLine(cornerRect.offset.x, cornerRect.size.x);
MatrixLine mask = new MatrixLine(cornerRect.offset.x, cornerRect.size.x);



if (bottom) prevLine.ReadLine(src, min.z);
else  prevLine.ReadLine(src, max.z-1);

for (int iz=0; iz<cornerRect.size.z; iz++)
{
 int z = bottom ? max.z-iz-1 : min.z+iz;

 currLine.ReadLine(src, z);
 MatrixLine.SpreadBlur(ref currLine, ref prevLine, blur);

 float gradientPercentZ = 1f * (z-min.z) / (max.z - min.z);
 float gradientZ = gradientStart.y*(1-gradientPercentZ) + gradientEnd.y*gradientPercentZ;
 mask.Gradient((gradientStart.x + (1-gradientZ))*0.5f, (gradientEnd.x + (1-gradientZ))*0.5f);
 //mask.WriteLine(this, z); // to test

 prevLine.WriteLine(this, z, mask);  //writing prev line since lines are swapped now
}

currLine = new MatrixLine(cornerRect.offset.z, cornerRect.size.z);
prevLine = new MatrixLine(cornerRect.offset.z, cornerRect.size.z);
mask = new MatrixLine(cornerRect.offset.z, cornerRect.size.z);
```

```
  if (left) prevLine.ReadRow(src, min.x);
  else   prevLine.ReadRow(src, max.x-1);

  //for (int x=max.x-1; x>=min.x; x--)
  for (int ix=0; ix<cornerRect.size.x; ix++)
  {
   int x = left ? max.x-ix-1 : min.x+ix;

   currLine.ReadRow(src, x);
   MatrixLine.SpreadBlur(ref currLine, ref prevLine, blur);

   float gradientPercentX = 1f * (x-min.x) / (max.x - min.x);
   float gradientX = gradientStart.x*(1-gradientPercentX) + gradientEnd.x*gradientPercentX;
   mask.Gradient((gradientStart.y + (1-gradientX))*0.5f, (gradientEnd.y + (1-gradientX))*0.5f);
   //mask.WriteRow(this, x); // to test

   prevLine.AppendRow(this, x, mask);  //writing prev line since lines are swapped now
  }
 }

#endregion


#region Native (under construction)
```

```
[DllImport ("NativePlugins", EntryPoint = "MatrixResize")]
public static extern void Resize (FloatMatrix src, FloatMatrix dst);

[DllImport ("NativePlugins", EntryPoint = "MatrixFastDownscale")]
public static extern void FastDownscale (FloatMatrix src, FloatMatrix dst, int ratio);

/*public void ReadMatrix (Matrix srcMatrix,
 CoordRect srcRect = new CoordRect(),
 CoordRect.TileMode tileMode = CoordRect.TileMode.Clamp,
 Interpolation upscaleInterpolation = Interpolation.Bicubic,
 Interpolation downscaleInterpolation = Interpolation.Linear)
/// Copies matrix with resize. If rect is defined then copies only the given rect.
{
 if (srcRect.size.x == 0 && srcRect.size.z == 0 && srcRect.offset.x == 0 && srcRect.offset.z == 0)
  srcRect = srcMatrix.rect;

 Interpolation interpolation = Interpolation.None;
 if (rect.size == srcRect.size) interpolation = Interpolation.None;
 else if (rect.size.x > srcRect.size.x || rect.size.z > srcRect.size.z) interpolation = upscaleInterpolation;
 else interpolation = downscaleInterpolation;

 Coord min = rect.Min; Coord max = rect.Max;
 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
 {
  double percentX = 1.0 * (x-min.x) / rect.size.x;
```

```
   double percentZ = 1.0 * (z-min.z) / rect.size.z;

   float srcX = (float)(percentX*srcRect.size.x) + srcRect.offset.x;
   float srcZ = (float)(percentZ*srcRect.size.z) + srcRect.offset.z;

   switch (interpolation)
   {
    case Interpolation.Linear: this[x,z] = srcMatrix.GetInterpolated(srcX, srcZ, tileMode:tileMode); break;
    case Interpolation.Bicubic: this[x,z] = srcMatrix.GetInterpolatedBicubic(srcX, srcZ, tileMode:tileMode); b
    default: this[x,z] = srcMatrix.GetTiled((int)srcX, (int)srcZ, tileMode); break;
   }
  }
 }*/

#endregion



#region Native Per-pixel (outdated?)

 public float GetInterpolated (float x, float z, CoordRect.TileMode tileMode = CoordRect.TileMode.Clamp)
 {
  //neig coords
  int px = (int)x; if (x<0) px--; //because (int)-2.5 gives -2, should be -3
  int nx = px+1;

  int pz = (int)z; if (z<0) pz--;
```

```
 int nz = pz+1;

 //blending percent between pixels
 float percentX = x-px;
 float percentZ = z-pz;

 //reading values
 float val_pxpz = GetTiled(px,pz, tileMode);
 float val_nxpz = GetTiled(nx,pz, tileMode);
 float val_fz = val_pxpz*(1-percentX) + val_nxpz*percentX;

 float val_pxnz = GetTiled(px,nz, tileMode);
 float val_nxnz = GetTiled(nx,nz, tileMode);
 float val_cz = val_pxnz*(1-percentX) + val_nxnz*percentX;

 float val = val_fz*(1-percentZ) + val_cz*percentZ;

 return val;
}


public void AddInterpolated (float x, float z, float val)
{
 //neig coords
 int px = (int)x; if (x<0) px--; //because (int)-2.5 gives -2, should be -3
 int nx = px+1;
```

```
 int pz = (int)z; if (z<0) pz--;
 int nz = pz+1;

 //blending percent between pixels
 float percentX = x-px;
 float percentZ = z-pz;

 //writing values
 int pos = (pz-rect.offset.z)*rect.size.x + px - rect.offset.x;
 arr[pos]    += val * (1-percentX) *  (1-percentZ);
 arr[pos+1]    += val * percentX *  (1-percentZ);
 arr[pos+rect.size.x] += val * (1-percentX) *  percentZ;
 arr[pos+rect.size.x+1] += val * percentX *  percentZ;
}


public float GetInterpolatedBicubic (float x, float z, CoordRect.TileMode tileMode = CoordRect.TileMode.
{
 //neig coords - z axis
 int p = (int)z; if (z<0) z--; //because (int)-2.5 gives -2, should be -3
 int n = p+1;
 int pp = p-1;
 int nn = n+1;

 //blending percent
```

```
 float percent = z-p;

 //reading values
 float vp = GetInterpolateCubic (x, p, tileMode);
 float vpp = GetInterpolateCubic (x, pp, tileMode);
 float vn = GetInterpolateCubic (x, n, tileMode);
 float vnn = GetInterpolateCubic (x, nn, tileMode);

 return vp + 0.5f * percent * (vn - vpp + percent*(2.0f*vpp - 5.0f*vp + 4.0f*vn - vnn + percent*(3.0f*(vp - vn
}

public float GetInterpolateCubic (float x, int z, CoordRect.TileMode tileMode = CoordRect.TileMode.Clam
/// Gets interpolated result using a horizontal level only
{
 //neig coords - x axis
 int p = (int)x; if (x<0) p--; //because (int)-2.5 gives -2, should be -3
 int n = p+1;
 int pp = p-1;
 int nn = n+1;

 //blending percent
 float percent = x-p;

 //reading values
 float vp = GetTiled(p,z,tileMode);
 float vpp = GetTiled(pp,z,tileMode);
```

```
 float vn = GetTiled(n,z,tileMode);
 float vnn = GetTiled(nn,z,tileMode);

 return vp + 0.5f * percent * (vn - vpp + percent*(2.0f*vpp - 5.0f*vp + 4.0f*vn - vnn + percent*(3.0f*(vp - vn
}

public float GetTiled (Coord coord, CoordRect.TileMode tileMode)
/// Returns the usual value if coord is in rect, handles tiling if it is not
{
 if (rect.Contains(coord))
  return this[coord];

 Coord tiledCoord = rect.Tile(coord, tileMode);
 return this[tiledCoord];
}

public float GetTiled (int x, int z, CoordRect.TileMode tileMode) { return GetTiled(new Coord(x,z), tileMod

public void AddLine (Vector3 start, Vector3 end, float valStart, float valEnd, bool antialised=false)
{
 float rectSizeX = Mathf.Abs(end.x-start.x);
 float rectSizeZ = Mathf.Abs(end.z-start.z);

 int length = (int)(rectSizeX>rectSizeZ ? rectSizeX : rectSizeZ) + 1;
```

```
float stepX = (end.x-start.x) / length;
float stepZ = (end.z-start.z) / length;

if (rectSizeX > rectSizeZ)
{
 stepZ /= Mathf.Abs(stepX);
 stepX = stepX>0 ? 1 : -1;
}
else
{
 stepX /= Mathf.Abs(stepZ);
 stepZ = stepZ>0 ? 1 : -1;
}

for (int i=0; i<length; i++)
{
 float x = start.x + stepX*i;
 float z = start.z + stepZ*i;

 int ix = (int)(float)x;  if (x<0) ix--;
 int iz = (int)(float)z;  if (z<0) iz--;

 if (ix<rect.offset.x || ix>=rect.offset.x+rect.size.x ||
  iz<rect.offset.z || iz>=rect.offset.z+rect.size.z )
   continue;
```

```
    int pos = (iz-rect.offset.z)*rect.size.x + ix - rect.offset.x;
    float percent = 1f*i/length;
    float val = valStart*(1-percent) + valEnd+percent;

    arr[pos] = val;

    if (antialised)
    {
     if (rectSizeX > rectSizeZ)
     {
      arr[pos-rect.size.x] = val*(1-(z-iz));
      arr[pos+rect.size.x] = val*(z-iz);
     }
     else
     {
      arr[pos-1] = val*(1-(x-ix));
      arr[pos+1] = val*(x-ix);
     }
    }
   }
  }

 #endregion
}
```

```
}//namespace
```

```
ï»¿// Operations with "maps"
// Note that functions are not inlined in editor, so keeping all method unfolded

using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;

namespace Den.Tools
{
 [Serializable, StructLayout (LayoutKind.Sequential)] //to pass to native
 public class MatrixLine
 /// A single matrix line (or row) to be used in fast native matrix operations
 {
  public int offset; //IDEA: use two coordinates in read/write line. IDEA: don't keep offset parameter. Line op
  public int length; //for c++ compatibility and for re-use cases (when array is longer than length)
  public float[] arr;

  public MatrixLine (int offset, int length)
  {
   arr = new float[length];
   this.offset = offset;
   this.length = length;
  }
```

```
/*public Line (int length)
{
 arr = new float[length];
 this.length = length;
}*/

public MatrixLine (float[] arr, int offset, int length)
{
 this.arr = arr;
 this.offset = offset;
 this.length = length;
}

#region Sampling

 public static void ResampleCubic (MatrixLine src, MatrixLine dst)
 /// Scales the line filling dst with interpolated values. Cubic for upscale
 {
  for (int x=0; x<dst.length; x++)
  {
   float percent = 1.0f * x / dst.length;
   float sx = percent * src.length;
   dst.arr[x] = src.CubicSample(sx);
  }
 }
```

```
public static void ResampleLinear (MatrixLine src, MatrixLine dst)
/// Scales the line filling dst with interpolated values. Linear for downscale
{
 float radius = 1.0f * src.length / dst.length;

 for (int x=0; x<dst.length; x++)
 {
  float percent = 1.0f * x / dst.length;
  float sx = percent * src.length;
  dst.arr[x] = src.LinearSample(sx, radius);
 }
}

public static void DownsampleFast (MatrixLine src, MatrixLine dst, int ratio)
/// Scales the line filling dst with interpolated values. Linear for downscale
{
 for (int x=0; x<dst.length; x++)
 {
  float sumVal = 0;
  for (int ix=0; ix<ratio; ix++)
   sumVal += src.arr[x*ratio + ix];
  dst.arr[x] = sumVal / ratio;
 }
}

public float CubicSample (float x)
```

```
/// Interpolated sampling for upscaling
{
 int p = (int)x;  if (p<0) p=0;
 int n = p+1;  if (n>length-1) n = length-1;
 int pp = p-1;  if (pp<0) pp = 0;
 int nn = n+1;  if (nn>length-1) nn = length-1;

 float percent = x-p;

 float vp = arr[p]; float vpp = arr[pp];
 float vn = arr[n]; float vnn = arr[nn];

 return vp + 0.5f * percent * (vn - vpp + percent*(2.0f*vpp - 5.0f*vp + 4.0f*vn - vnn + percent*(3.0f*(vp - vn
}

public float LinearSample (float x, float radius)
/// Weighted radius sampling for downscaling
{
 int ix = (int)x;
 int iRadius = (int)(radius+0.5f);

 float factorSum = 0;
 float valueSum = 0;

 for (int rx = ix-iRadius+1; rx < ix+iRadius+1; rx++)
 //skipping edge vertices since their factor is 0
```

```
 //evaluating +1 point if x is between ix and ix+1
 {
  float dist = x - rx;
  if (dist < 0) dist = -dist;

  float factor = 1 - dist/radius;
  if (factor<0) factor = 0;

  int crx = rx;
  if (crx<0) crx = 0;
  if (crx>length-1) crx = length-1;

  factorSum += factor;
  valueSum += arr[crx] * factor;
 }

 return valueSum / factorSum;
}

public float AverageSample (int x)
{
 int p = x-1;  if (p<0) p=0;
 int n = x+1;  if (n>length-1) n = length-1;

 float vp = arr[p];
 float vx = arr[x];
```

```
  float vn = arr[n];


  return vp*0.25f + vx*0.5f + vn*0.25f;
 }

#endregion



#region Operations


 public void Spread (float subtract=0.01f, float multiply=1f)
 /// Spreads higher values (whites) over the lower (darker) ones
 // Warning: spreading with multiply is WRONG!
 // It will not steer the corner
 {
  //to right
  float prevVal = arr[0];
  for (int x=1; x<length-1; x++)
  {
   float val = arr[x];
   if (prevVal > val)
   {
    val = prevVal*multiply - subtract*(1-val/prevVal);
    if (val<0) val =0;


    arr[x] = val;
```

```
  }
  prevVal = val;
 }

 //to left
 prevVal = arr[length-1];
 for (int x=length-2; x>=0; x--)
 {
  float val = arr[x];
  if (prevVal > val)
  {
   val = prevVal*multiply - subtract*(1-val/prevVal);
   if (val<0) val =0;

   arr[x] = val;
  }
  prevVal = val;
 }
}

public void SpreadMultiply (float multiply=1f)
{
 //to right
 float prevVal = arr[0];
 for (int x=1; x<length-1; x++)
```

```
{
 float currVal = arr[x];

 if (prevVal > currVal)
 {
  currVal = currVal*(1-multiply) + prevVal*multiply;
  arr[x] = currVal;
 }

 prevVal = currVal;
}

//to left
prevVal = arr[length-1];
for (int x=length-2; x>=0; x--)
{
 float currVal = arr[x];

 if (prevVal > currVal)
 {
  currVal = currVal*(1-multiply) + prevVal*multiply;
  arr[x] = currVal;
 }

 prevVal = currVal;
}
```

```
}

public void Cavity (float intensity=1)
{
 float prev = arr[0];
 float curr = arr[1];

 for (int x=1; x<length-1; x++)
 {
  //float prev = src.arr[x-1];
  //float curr = src.arr[x];
  float next = arr[x+1];

  float val = curr - (next + prev)/2;
  float sign = val>0 ? 1 : -1;
  val = (val*val*sign)*intensity*1000;
  val = (val+1) / 2;
  arr[x] = val;

  prev = curr;
  curr = next;
 }
 arr[0] = arr[1];
 arr[length-1] = arr[length-2];
}
```

```
public void Delta ()
{
 float prev = arr[0];
 float curr = arr[1];

 for (int x=1; x<length-1; x++)
 {
  //float prev = arr[x-1];
  //float curr = arr[x];
  float next = arr[x+1];

  float prevDelta = prev-curr; if (prevDelta < 0) prevDelta = -prevDelta;
  float nextDelta = next-curr; if (nextDelta < 0) nextDelta = -nextDelta;
  float delta = prevDelta>nextDelta? prevDelta : nextDelta;

  if (delta > arr[x]) arr[x] = delta;

  prev = curr;
  curr = next;
 }
 arr[0] = arr[1];
 arr[length-1] = arr[length-2];
}
```

```
public void Normal (float height)
/// Will return normal direction in range -1, 1
{
 float prev = arr[0];
 float curr = arr[1];

 for (int x=1; x<length-1; x++)
 {
  //float prev = arr[x-1];
  //float curr = arr[x];
  float next = arr[x+1];

  //Vector3(prev-next, height, 0)).normalized;
  float delta = prev-next;
  float magnitude = Mathf.Sqrt(delta*delta + height*height + delta*delta);
  float normal = delta*magnitude;
  normal = (normal+1) / 2;

  arr[x] = normal;

  prev = curr;
  curr = next;
 }
 arr[0] = arr[1];
 arr[length-1] = arr[length-2];
}
```

```
public void GaussianBlur (float[] tmp, float blur)
{
 int iterations = (int)blur;

 MatrixLine src = new MatrixLine(arr, 0, length); //for switching arrays between iterations
 MatrixLine dst = new MatrixLine(tmp, 0, length);

 //iteration blur
 for (int i=0; i<iterations; i++)
 {
  for (int x=0; x<length; x++)
   dst.arr[x] = src.AverageSample(x);

  float[] t = src.arr;
  src.arr = dst.arr;
  dst.arr = t;
 }

 //last iteration - percentage
 float percent = blur - iterations;
 if (percent > 0.0001f)
 {
  for (int x=0; x<length; x++)
   dst.arr[x] = src.AverageSample(x)*percent + src.arr[x]*(1-percent);
```

```
  float[] t = src.arr;
  src.arr = dst.arr;
  dst.arr = t;
 }

 //copy values to arr for non-even iteration count
 for (int x=0; x<length; x++)
  dst.arr[x] = src.arr[x];
}


public void DownsampleBlur (int downsample, float blur, MatrixLine tmpDownsized, MatrixLine tmpBlur)
/// both temp lines length is length/downsample
{
 ResampleLinear(this, tmpDownsized);
 tmpDownsized.GaussianBlur(tmpBlur.arr, blur);
 ResampleCubic(tmpDownsized, this);
}


public static void SpreadBlur (ref MatrixLine curr, ref MatrixLine prev, int blur)
{
 for (int x=0; x<curr.arr.Length; x++)
 {
  if (curr.arr[x] > 0.001f) continue;
```

```
  float val = 0;
  float sum = 0;

  if (prev.arr[x] > 0.001f)  { val += prev.arr[x]; sum++; }

  for (int i=1; i<=blur; i++)
  {
   if (x-i >= 0  &&  prev.arr[x-i] > 0.001f)     { val += prev.arr[x-i]; sum++; }
   if (x+i < prev.arr.Length-1  &&  prev.arr[x+i] > 0.001f) { val += prev.arr[x+i]; sum++; }
  }

  if (sum != 0)
   curr.arr[x] = val / sum;
 }

 //swapping lines
 MatrixLine tmp = prev;
 prev = curr;
 curr = tmp;
}


public void PredictExtend (int start)
/// Averages the line behavior before start and continues it after start
{
```

```
//int max = start < length-start ? start : length-start;  //iterating both in two ways from start, whatever is les

float negPrev = arr[0];
float posPrev = arr[0];

//finding the average pivot and vector
float prevVals = 0; //the values starting from start and going deeper into array. And smoother
float prevVector = 0;
float prevSum = 0;
for (int x=0; x<start; x++)
{
 float negCurr = arr[x];

 float weight = 1f; // 1f / (x+1);
 //weight = 1 - (1-weight)*(1-weight);

 prevVals += negCurr * weight;
 prevVector += (negCurr-negPrev) * weight;  //IDEA: if weight=1 avgVector is just a difference between fi
 prevSum += weight;

 negPrev = negCurr;
}

float avgPivot = prevVals / prevSum;
float avgVector = prevVector / prevSum;
```

```
 //shifting pivot to make it start according to avgVector
 avgPivot += avgVector * (start/2f);

 avgPivot = arr[start-1];

 //applying
 negPrev = arr[start];
 posPrev = arr[start];
 float pivotPrev = avgPivot;

 for (int x=start; x<arr.Length; x++)
 {
  float pivotCurr = pivotPrev + avgVector;
  arr[x] = pivotCurr; //posCurr*(1-pivotBlend) + pivotCurr*pivotBlend;
  pivotPrev = pivotCurr;
 }
}


public void FillGaps (int start=0, int end=0)
{
 if (start==0 && end==0) end = length;

 int gapSize = 0;
 float gapStartVal = 0;
 float gapEndVal = 0;
```

```
//finding first start value
int firstDefined = start;
while (arr[firstDefined] < 0  &&  firstDefined < end)
 firstDefined++;
gapStartVal = arr[firstDefined];

//filling gaps
for (int x=start; x<end; x++)
{
 float val = arr[x];

 //if val is gap
 if (val < 0) gapSize ++;

 //if gap just ended
 else if (gapSize != 0)
 {
  gapEndVal = val;
  for (int ix=0; ix<gapSize; ix++)
  {
   float percent = 1f * (ix+1) / (gapSize+1);
   arr[x-gapSize+ix] = gapStartVal*(1-percent) + gapEndVal*percent;
  }
  gapSize = 0;
  gapStartVal = val;
```

```
  }

  //if no gap
  else
   gapStartVal = val;
 }

 //filling the remaining dist if ent is gap
 if (gapSize != 0)
  for (int ix=0; ix<gapSize; ix++)
   arr[end-gapSize+ix] = gapStartVal;
}

public void FillGapsOld (int start=0, int end=0, MatrixLine tmpFront=null, MatrixLine tmpBack=null, Matrix
{
 if (start==0 && end==0) end = length;

 if (tmpFront==null) tmpFront = new MatrixLine(offset, length);
 if (tmpBack==null) tmpBack = new MatrixLine(offset, length);
 if (tmpFrontDist==null) tmpFrontDist = new MatrixLine(offset, length);
 if (tmpBackDist==null) tmpBackDist = new MatrixLine(offset, length);

 //filling front line
 float prevVal = -1;
 float prevDist = 0;
 for (int x=start; x<end; x++)
```

```
{
 float val = arr[x];
 if (val >= 0)
 {
  prevVal = val;
  prevDist = 0;
 }

 tmpFront.arr[x] = prevVal;
 tmpFrontDist.arr[x] = prevDist;
 prevDist++;
}

//extended front line with delta vector - works fine but the result is so-so
/*float prevVal = -1;
float prevDist = 0;
float prevDelta = 0;
bool prevEnabled = false; //to avoid calculating delta on single pixels
for (int x=0; x<length; x++)
{
 float val = arr[x];
 if (val >= 0)
 {
  prevDelta = val-prevVal;
  prevVal = val;
  prevDist = 0;
```

```
 if (!prevEnabled)
 prevDelta = 0; //linear delta if prev pixel is a gap
 prevEnabled = true;
}
else
{
 prevDist++;
 prevVal += prevDelta * (1f/prevDist);
}

tmpFront.arr[x] = prevVal;
tmpFrontDist.arr[x] = prevDist;
}*/

//back line
prevVal = -1;
prevDist = 0;
for (int x=end-1; x>=start; x--)
{
 float val = arr[x];
 if (val >= 0)
 {
  prevVal = val;
  prevDist = 0;
 }
```

```
   tmpBack.arr[x] = prevVal;
   tmpBackDist.arr[x] = prevDist;
   prevDist++;
  }

  //blending lines
  for (int x=start; x<end; x++)
  {
   //float val = arr[x];
   //if (val >= 0) continue;

   //float frontVal = tmpFront.arr[x];
   //float backVal = tmpBack.arr[x];

   float distSum = tmpFrontDist.arr[x] + tmpBackDist.arr[x];
   if (distSum == 0) continue;
   arr[x] = tmpFront.arr[x]*(tmpBackDist.arr[x]/distSum) + tmpBack.arr[x]*(tmpFrontDist.arr[x]/distSum); //n
  }
 }

#endregion


#region Arithmetic
```

```
public void Add (float f) { for (int i = 0; i<length; i++) arr[i] += f; }

public void Fill (float f) { for (int i = 0; i<length; i++) arr[i] = f; }

public void Max (MatrixLine l)

{

 for (int i = 0; i<length; i++)

 {

  float v1 = arr[i];

  float v2 = l.arr[i];

  arr[i] = v1>v2 ? v1 : v2;

 }

}


public void Gradient (float startVal, float endVal)

{

 for (int i = 0; i<length; i++)

 {

  float percent = 1f * i / length;

  arr[i] = startVal*(1-percent) + endVal*percent;

 }

}


public void Invert () { for (int i = 0; i<length; i++) arr[i] = 1-arr[i]; }


public float Average (bool skipNegative=false)

{

 float sum = 0;
```

```
 int count = 0;
 for (int i = 0; i<length; i++)
 {
  float val = arr[i];
  if (!skipNegative  ||  val>0) { sum += val; count++; }
 }
 if (count == 0) return -Mathf.Epsilon;
 else return sum / count;
}

#endregion



#region Reading/Writing Matrix

 //All line readings are zero-based (no offset)

 public void ReadLine (FloatMatrix matrix, int z)
 {
  int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  offset;
  for (int x=0; x<length; x++)
   arr[x] = matrix.arr[start+x];  //matrix[x+offset, z];
 }

 public void ReadRow (FloatMatrix matrix, int x)
 {
```

```
 int start = (offset-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;
 for (int z=0; z<length; z++)
  arr[z] = matrix.arr[start + z*matrix.rect.size.x];  //matrix[x, z+offset];
}

public void WriteLine (FloatMatrix matrix, int z)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x   +  offset;
 for (int x=0; x<length; x++)
  matrix.arr[start+x] = arr[x];  //matrix[x+offset, z];
}

public void WriteLine (FloatMatrix matrix, int z, MatrixLine mask)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x   +  offset;
 for (int x=0; x<length; x++)
  matrix.arr[start+x] = arr[x] * mask.arr[x];  //matrix[x+offset, z];

}

public void AppendLine (FloatMatrix matrix, int z)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x   +  offset;
 for (int x=0; x<length; x++)
  matrix.arr[start+x] += arr[x];  //matrix[x+offset, z];
}
```

```
public void AppendLine (FloatMatrix matrix, int z, MatrixLine mask)
{
 int start = (z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x   +  offset;
 for (int x=0; x<length; x++)
  matrix.arr[start+x] += arr[x] * mask.arr[x];  //matrix[x+offset, z];
}

public void WriteRow (FloatMatrix matrix, int x)
{
 int start = (offset-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;
 for (int z=0; z<length; z++)
  matrix.arr[start + z*matrix.rect.size.x] = arr[z];
}

public void WriteRow (FloatMatrix matrix, int x, MatrixLine mask)
{
 int start = (offset-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;
 for (int z=0; z<length; z++)
  matrix.arr[start + z*matrix.rect.size.x] = arr[z] * mask.arr[z];
}

public void AppendRow (FloatMatrix matrix, int x)
{
 int start = (offset-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;
 for (int z=0; z<length; z++)
```

```
  matrix.arr[start + z*matrix.rect.size.x] += arr[z];
}

public void AppendRow (FloatMatrix matrix, int x, MatrixLine mask)
{
 int start = (offset-matrix.rect.offset.z)*matrix.rect.size.x + x - matrix.rect.offset.x;
 for (int z=0; z<length; z++)
  matrix.arr[start + z*matrix.rect.size.x] += arr[z] * mask.arr[z];
}



//inclined lines
public void ReadInclined (FloatMatrix matrix, Vector2 start, Vector2 step)
{
 for (int i=0; i<length; i++)
 {
  float x = start.x + step.x*i;
  float z = start.y + step.y*i;

  if (x<0) x--;  int ix = (int)(float)(x + 0.5f);
  if (z<0) z--;  int iz = (int)(float)(z + 0.5f);

  if (ix<matrix.rect.offset.x || ix>=matrix.rect.offset.x+matrix.rect.size.x ||
   iz<matrix.rect.offset.z || iz>=matrix.rect.offset.z+matrix.rect.size.z )
    arr[i] = -Mathf.Epsilon;
  else
```

```
  {
   int pos = (iz-matrix.rect.offset.z)*matrix.rect.size.x + ix - matrix.rect.offset.x;
   arr[i] = matrix.arr[pos];
  }
 }
}

public void WriteInclined (FloatMatrix matrix, Vector2 start, Vector2 step)
{
 for (int i=0; i<length; i++)
 {
  float x = start.x + step.x*i;
  float z = start.y + step.y*i;

  if (x<0) x--;  int ix = (int)(float)(x + 0.5f);
  if (z<0) z--;  int iz = (int)(float)(z + 0.5f);

  if (ix<matrix.rect.offset.x || ix>=matrix.rect.offset.x+matrix.rect.size.x ||
   iz<matrix.rect.offset.z || iz>=matrix.rect.offset.z+matrix.rect.size.z )
    continue;
  else
  {
   int pos = (iz-matrix.rect.offset.z)*matrix.rect.size.x + ix - matrix.rect.offset.x;
   matrix.arr[pos] = arr[i];
  }
 }
```

```
}


public void ReadCircular (FloatMatrix matrix, Coord center, float radius)
{
 int counter = 0;

 ReadArcX(matrix, center, radius, ref counter);
 ReadArcZ(matrix, center, radius, ref counter, reverse:true);

 ReadArcZ(matrix, center, radius, ref counter, flipZ:-1);
 ReadArcX(matrix, center, radius, ref counter, flipZ:-1, reverse:true);

 ReadArcX(matrix, center, radius, ref counter, flipX:-1, flipZ:-1);
 WriteArcZ(matrix, center, radius, ref counter, flipX:-1, flipZ:-1,  reverse:true);

 ReadArcZ(matrix, center, radius, ref counter, flipX:-1);
 ReadArcX(matrix, center, radius, ref counter, flipX:-1, reverse:true);
}


public void WriteCircular (FloatMatrix matrix, Coord center, float radius)
{
 int counter = 0;

 WriteArcX(matrix, center, radius, ref counter);
```

```
 WriteArcZ(matrix, center, radius, ref counter, reverse:true);

 WriteArcZ(matrix, center, radius, ref counter, flipZ:-1);
 WriteArcX(matrix, center, radius, ref counter, flipZ:-1, reverse:true);

 WriteArcX(matrix, center, radius, ref counter, flipX:-1, flipZ:-1);
 WriteArcZ(matrix, center, radius, ref counter, flipX:-1, flipZ:-1,  reverse:true);

 WriteArcZ(matrix, center, radius, ref counter, flipX:-1);
 WriteArcX(matrix, center, radius, ref counter, flipX:-1, reverse:true);
}

private void ReadArcX (FloatMatrix matrix, Coord center, float radius, ref int counter, int flipX=1, int flipZ=
{
 int radius45 = (int)(radius*0.7071068f + 1);

 for (int ix=0; ix<radius45; ix++)
 {
  int x = reverse ? radius45-ix : ix;

  float fz = Mathf.Sqrt(radius*radius - x*x);
  if (fz<0) fz--;  int z = (int)(float)(fz + 0.5f); //presuming here center is 0,0

  int cx = center.x + x*flipX;
  int cz = center.z + z*flipZ;
```

```
  int pos = (cz-matrix.rect.offset.z)*matrix.rect.size.x + cx - matrix.rect.offset.x;
  arr[counter] = matrix.arr[pos];
  counter++;
 }
}

private void ReadArcZ (FloatMatrix matrix, Coord center, float radius, ref int counter, int flipX=1, int flipZ=
{
 int radius45 = (int)(radius*0.7071068f + 1);

 for (int iz=0; iz<radius45; iz++)
 {
  int z = reverse ? radius45-iz : iz;

  float fx = Mathf.Sqrt(radius*radius - z*z);
  if (fx<0) fx--;  int x = (int)(float)(fx + 0.5f);

  int cx = center.x + x*flipX;
  int cz = center.z + z*flipZ;

  int pos = (cz-matrix.rect.offset.z)*matrix.rect.size.x + cx - matrix.rect.offset.x;
  arr[counter] = matrix.arr[pos];
  counter++;
 }
}
```

```
private void WriteArcX (FloatMatrix matrix, Coord center, float radius, ref int counter, int flipX=1, int flipZ=
{
 int radius45 = (int)(radius*0.7071068f + 1);

 for (int ix=0; ix<radius45; ix++)
 {
  int x = reverse ? radius45-ix : ix;

  float fz = Mathf.Sqrt(radius*radius - x*x);
  if (fz<0) fz--;  int z = (int)(float)(fz + 0.5f); //presuming here center is 0,0

  int cx = center.x + x*flipX;
  int cz = center.z + z*flipZ;

  int pos = (cz-matrix.rect.offset.z)*matrix.rect.size.x + cx - matrix.rect.offset.x;
  matrix.arr[pos] = arr[counter];
  counter++;
 }
}

private void WriteArcZ (FloatMatrix matrix, Coord center, float radius, ref int counter, int flipX=1, int flipZ=
{
 int radius45 = (int)(radius*0.7071068f + 1);

 for (int iz=0; iz<radius45; iz++)
```

```
 {
  int z = reverse ? radius45-iz : iz;

  float fx = Mathf.Sqrt(radius*radius - z*z);
  if (fx<0) fx--;  int x = (int)(float)(fx + 0.5f);

  int cx = center.x + x*flipX;
  int cz = center.z + z*flipZ;

  int pos = (cz-matrix.rect.offset.z)*matrix.rect.size.x + cx - matrix.rect.offset.x;
  matrix.arr[pos] = arr[counter];
  counter++;
 }
}


public void ReadSquare (FloatMatrix matrix, Coord center, int radius)
/// Same as circular, but in form of square. 4 lines one-by-one. Useful for blurs and spreads
{
 int side = radius*2 + 1;

 //resetting line
 for (int i=0; i<side*4; i++)
  arr[i] = - Mathf.Epsilon;

 Coord min = center-radius;
```

```
Coord max = center+radius;


Coord rectMin = matrix.rect.offset;
Coord rectMax = matrix.rect.offset + matrix.rect.size;


int start = (min.z-matrix.rect.offset.z-1)*matrix.rect.size.x - matrix.rect.offset.x  +  min.x; //matrix[min.x, mi
if (min.z-1 >= rectMin.z  &&  min.z-1 < rectMax.z)
 for (int x=0; x<side; x++)
  if (x+min.x >= rectMin.x  &&  x+min.x < rectMax.x)
   arr[x] = matrix.arr[start+x];


start = (min.z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  max.x; //matrix[max.x, min.z]
if (max.x >= rectMin.x  &&  max.x < rectMax.x)
 for (int z=0; z<side; z++)
  if (z+min.z >= rectMin.z  &&  z+min.z < rectMax.z)
   arr[z+side] = matrix.arr[start + z*matrix.rect.size.x];


start = (max.z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  max.x - 1; //matrix[max.x-1,
if (max.z >= rectMin.z  &&  max.z < rectMax.z)
for (int x=0; x<side; x++)
 if (max.x-1-x >= rectMin.x  &&  max.x-1-x < rectMax.x)
  arr[x+side*2] = matrix.arr[start -x];


start = (max.z-matrix.rect.offset.z-1)*matrix.rect.size.x - matrix.rect.offset.x  +  min.x - 1; //matrix[min.x-1,
if (min.x-1 >= rectMin.x  &&  min.x-1 < rectMax.x)
for (int z=0; z<side; z++)
```

```
  if (max.z-1-z >= rectMin.z  &&  max.z-1-z < rectMax.z)
   arr[z+side*3] = matrix.arr[start - z*matrix.rect.size.x]; //matrix[min.x, max.z-1-z]
}



public void WriteSquare (FloatMatrix matrix, Coord center, int radius)
/// Same as circular, but in form of square. 4 lines one-by-one. Useful for blurs and spreads
/// Line length should be radius*8 + 4 corners
{
 int side = radius*2 + 1;

 Coord min = center-radius;
 Coord max = center+radius;

 Coord rectMin = matrix.rect.offset;
 Coord rectMax = matrix.rect.offset + matrix.rect.size;

 int start = (min.z-matrix.rect.offset.z-1)*matrix.rect.size.x - matrix.rect.offset.x  +  min.x; //matrix[min.x, mi
 if (min.z-1 >= rectMin.z  &&  min.z-1 < rectMax.z)
  for (int x=0; x<side; x++)
   if (x+min.x >= rectMin.x  &&  x+min.x < rectMax.x)
    matrix.arr[start+x] = arr[x];

 start = (min.z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  max.x; //matrix[max.x, min.z]
 if (max.x >= rectMin.x  &&  max.x < rectMax.x)
  for (int z=0; z<side; z++)
```

```
      if (z+min.z >= rectMin.z  &&  z+min.z < rectMax.z)
       matrix.arr[start + z*matrix.rect.size.x] = arr[z+side];

    start = (max.z-matrix.rect.offset.z)*matrix.rect.size.x - matrix.rect.offset.x  +  max.x - 1; //matrix[max.x-1,
    if (max.z >= rectMin.z  &&  max.z < rectMax.z)
    for (int x=0; x<side; x++)
     if (max.x-1-x >= rectMin.x  &&  max.x-1-x < rectMax.x)
      matrix.arr[start -x] = arr[x+side*2];

    start = (max.z-matrix.rect.offset.z-1)*matrix.rect.size.x - matrix.rect.offset.x  +  min.x - 1; //matrix[min.x-1,
    if (min.x-1 >= rectMin.x  &&  min.x-1 < rectMax.x)
    for (int z=0; z<side; z++)
     if (max.z-1-z >= rectMin.z  &&  max.z-1-z < rectMax.z)
      matrix.arr[start - z*matrix.rect.size.x] = arr[z+side*3]; //matrix[min.x, max.z-1-z]
   }

  #endregion

 } //class

}//namespace
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using UnityEngine;

namespace Den.Tools
{
 public class MatrixNative
 {
  [DllImport ("NativePlugins", EntryPoint = "?Test@Matrix@@QEAAHXZ")]
  public static extern int TestCall ();
 }

}
```

```
ï»¿using System;
using UnityEngine;
using UnityEditor;

namespace Den.Tools.SceneEdit
{
 public static class MoveRotateScale
 {
  public static bool isDragging;
  public static Vector3 origPivot;
  public static Vector3[] origPositions;


  public static bool Update (Vector3[] poses, Vector3 pivot)
  /// Moves, rotates or scales the given positions. Returns true if any change was made.
  /// Should be fired once per frame
  {
   if (UnityEditor.Tools.current == Tool.Move)
    return Move(poses, pivot);

   if (UnityEditor.Tools.current == Tool.Scale)
    return Scale(poses, pivot);

   if (UnityEditor.Tools.current == Tool.Rotate)
    return Rotate(poses, pivot);
```

```
 return false;
}



public static bool Scale (Vector3[] poses, Vector3 pivot)
{
 //don't ask
 bool mouseReleased = Event.current.rawType == EventType.MouseUp;

 if (isDragging) pivot = origPivot;

 float gizmoSize = HandleUtility.GetHandleSize(pivot);
 Vector3 newScale = Handles.ScaleHandle(new Vector3(1,1,1), pivot, Quaternion.identity, gizmoSize);

 //resetting gizmo on mouse up (after drawing gizmo)
 if (mouseReleased)
 {
  origPositions = null;
  isDragging = false;
  return false;
 }

 //on dragging
 if (newScale != new Vector3(1,1,1))
 {
  //if just started - saving original positions
```

```
 if (!isDragging)
 {
  origPositions = new Vector3[poses.Length];
  Array.Copy(poses, origPositions, poses.Length);

  isDragging = true;

  origPivot = pivot;
 }

 //moving positions
 for (int p=0; p<poses.Length; p++)
 {
  Vector3 relPos = origPositions[p] - pivot;
  relPos = new Vector3(relPos.x*newScale.x, relPos.y*newScale.y, relPos.z*newScale.z); // relPos *= sca
  poses[p] = relPos + pivot;

 }

 return true;
}

return false;
}
```

```
public static bool Rotate (Vector3[] poses, Vector3 pivot)
{
 //don't ask
 bool mouseReleased = Event.current.rawType == EventType.MouseUp;

 if (isDragging) pivot = origPivot;

 float gizmoSize = HandleUtility.GetHandleSize(pivot);
 Quaternion rotation = Handles.RotationHandle(Quaternion.identity, pivot);

 //resetting gizmo on mouse up (after drawing gizmo)
 if (mouseReleased)
 {
  origPositions = null;
  isDragging = false;
  return false;
 }

 //on dragging
 if (rotation != Quaternion.identity)
 {
  //if just started - saving original positions
  if (!isDragging)
  {
   origPositions = new Vector3[poses.Length];
   Array.Copy(poses, origPositions, poses.Length);
```

```
   isDragging = true;


   origPivot = pivot;
  }


  //moving positions
  for (int p=0; p<poses.Length; p++)
  {
   Vector3 relPos = origPositions[p] - pivot;
   relPos = rotation * relPos;
   poses[p] = relPos + pivot;


  }


   return true;
  }


 return false;
}



public static bool Move (Vector3[] poses, Vector3 pivot)
{
 Vector3 newPivot = Handles.PositionHandle(pivot, Quaternion.identity);
 Vector3 delta = pivot - newPivot;
```

```
if (delta.sqrMagnitude != 0)
{
 for (int p=0; p<poses.Length; p++)
  poses[p] -= delta;

 return true;
}
return false;

/*//don't ask
bool mouseReleased = Event.current.rawType == EventType.MouseUp;

if (isDragging) pivot = origPivot;

float gizmoSize = HandleUtility.GetHandleSize(pivot);
Vector3 position = Handles.PositionHandle(pivot, Quaternion.identity);

//resetting gizmo on mouse up (after drawing gizmo)
if (mouseReleased)
{
 origPositions = null;
 isDragging = false;
 return false;
}

//on dragging
```

```
   if (position != origPivot)
   {
    //if just started - saving original positions
    if (!isDragging)
    {
     origPositions = new Vector3[poses.Length];
     Array.Copy(poses, origPositions, poses.Length);

     isDragging = true;

     origPivot = pivot;
    }

    //moving positions
    for (int p=0; p<poses.Length; p++)
     poses[p] = origPositions[p] + position-pivot;

    return true;
   }

   return false;*/
  }
 }
}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//using UnityEngine.Profiling;

using Den.Tools.GUI;

namespace Den.Tools.GUI
{
 public class PolyLine
 {
  public Mesh mesh;
  private Vector3[] vertices;
  private Vector3[] neigCoords;
  private Vector2[] uv;
  private Vector2[] uv2; //point num and length
  private int[] tris;
  public int pointsCount; //number of visible points, assigned on SetPoints

  private static Material lineMat;
  private static Texture2D lineTex;

  public enum ZMode { Occluded, Overlay, Both };
```

```
public int MaxPoints { get{ return vertices.Length/4-1; }}


public PolyLine (int maxPoints)
{
 //SetMesh(maxPoints);
}



public void SetPoints (Vector3[] points, int numPoints=-1)
/// Shapes polyline before drawing it
{
 SetPointsThread(points, numPoints);
 SetPointsApply();
}



public void SetPointsThread (Vector3[] points, int numPoints=-1)
 { SetPointsThread( new Vector3[][] { points }, numPoints); }


public void SetPointsThread (Vector3[][] points, int numPoints=-1)
/// Prepares points in thread. SetPointsApply should be called after
{
 if (numPoints < 0)
 {
  numPoints = 0;
  for (int s=0; s<points.Length; s++)
```

```
  numPoints += points[s].Length;
}

if (numPoints<=1)
{
 //throw new System.Exception("Line points number is <= 1");
 //I see no reason why empty spline should be an error

 vertices = new Vector3[0];
}

pointsCount = numPoints;
//if (mesh == null) SetMesh(numPoints);
//if (numPoints >= mesh.vertexCount)
// throw new System.Exception("Line points number is more than maximum allowed");

//preparing arrays
if (vertices == null  ||  vertices.Length != numPoints*4) vertices = new Vector3[numPoints*4];
if (neigCoords == null  ||  neigCoords.Length != numPoints*4) neigCoords = new Vector3[numPoints*4];
if (uv == null  ||  uv.Length != numPoints*4) uv = new Vector2[numPoints*4];  //uvs are always the same,
if (uv2 == null  ||  uv2.Length != numPoints*4) uv2 = new Vector2[numPoints*4];

//filling arrays
int v = 0;
for (int s=0; s<points.Length; s++)
{
```

```
Vector3[] line = points[s];
float totalLength = 0;

for (int p=0; p<points[s].Length; p++)
{
 Vector3 vert = line[p];
 vertices[v] = vert;
 vertices[v+1] = vert;// + new Vector3(0,0,1f);
 vertices[v+2] = vert;// + new Vector3(1f,0,1f);
 vertices[v+3] = vert;// + new Vector3(1f,0,0);

 Vector3 prev = p!=0 ? line[p-1] : line[0]-line[1];
 neigCoords[v] = prev;
 neigCoords[v+1] = prev;

 Vector3 next = p!=line.Length-1 ? line[p+1] : line[line.Length-1];
 neigCoords[v+2] = next;
 neigCoords[v+3] = next;

 uv[v] = new Vector2(-1,-1);
 uv[v+1] = new Vector2(-1,1);
 uv[v+2] = new Vector2(1,1);
 uv[v+3] = new Vector2(1,-1);

 //float segLength = (points[i-1] - points[i]).magnitude;  //using X and Z axis only
 float segLength = p!=0 ?
```

```
   Mathf.Sqrt( (line[p-1].x - line[p].x)*(line[p-1].x - line[p].x) + (line[p-1].z - line[p].z)*(line[p-1].z - line[p].z) )
   0;
  totalLength += segLength;

  uv2[v].x = p; uv2[v].y = totalLength;
  uv2[v+1].x = p; uv2[v+1].y = totalLength;
  uv2[v+2].x = p; uv2[v+2].y = totalLength;
  uv2[v+3].x = p; uv2[v+3].y = totalLength;

  v += 4;
 }
}

//tris
int numTris = 0;
for (int s=0; s<points.Length; s++)
 numTris += ( (points[s].Length-1)*4 + 2 )*3;

if (tris == null  ||  tris.Length != numTris)
 tris = new int[numTris];

v = 0;
int t = 0;
for (int s=0; s<points.Length; s++)
{
 Vector3[] line = points[s];
```

```
 for (int p=0; p<points[s].Length; p++)
 {
  tris[t] = v;
  tris[t+1] = v+2;
  tris[t+2] = v+1;

  tris[t+3] = v+1;
  tris[t+4] = v+2;
  tris[t+5] = v+3;

  if (p != points[s].Length-1)
  {
   tris[t+6] = v+2;
   tris[t+7] = v+4;
   tris[t+8] = v+3;

   tris[t+9] = v+3;
   tris[t+10] = v+4;
   tris[t+11] = v+5;
  }

  v += 4;
  t += p!=points[s].Length-1 ? 12 : 6;
 }
}
```

```
}



public void SetPointsApply ()
/// Applies SetPointsThread
{
 if (mesh == null) { mesh = new Mesh(); mesh.MarkDynamic(); }
 if (vertices.Length < mesh.vertices.Length) mesh.triangles = new int[0]; //otherwise "The supplied vertex a

 mesh.vertices = vertices;
 mesh.normals = neigCoords;
 mesh.uv = uv;
 mesh.uv2 = uv2;
 mesh.triangles = tris;
}



public void DrawLine (Color color, float width, float dottedSpace=0, float offset=0.01f, ZMode zMode=ZMo
/// Draws a line with the points previously set
{
 if (mesh == null) //in some cases mesh turns null (when switching MicroSplat output to Both/Splats for so
  SetPointsApply();

 int numPoints = pointsCount;
 if (numPoints < 0) numPoints = vertices.Length/4;
```

```
 if (lineMat == null) lineMat = new Material( Shader.Find("Hidden/DPLayout/PolyLine") );
 if (lineTex == null) lineTex = Resources.Load("DPUI/PolyLineTex") as Texture2D;

 lineMat.SetTexture("_MainTex", lineTex);
 lineMat.SetColor("_Color", color);
 lineMat.SetFloat("_Width", width);
 lineMat.SetFloat("_Offset", offset);
 lineMat.SetFloat("_NumPoints", numPoints-1);
 lineMat.SetFloat("_Dotted", dottedSpace);
 lineMat.SetInt("_ZTest", zMode==ZMode.Occluded ? 2 : 0); //2 for LEqual

 lineMat.SetPass(0);
 Graphics.DrawMeshNow(mesh, parent==null ? Matrix4x4.identity : parent.localToWorldMatrix);
}


public void DrawLine (Vector3[] points, Color color, float width, float dottedSpace=0, float offset=0.01f, int
/// Sets points and draws a line
{
 if (numPoints < 0) numPoints = points.Length;

 SetPoints(points, numPoints);
 DrawLine(color, width, dottedSpace:dottedSpace, offset:offset, zMode:zMode, parent:parent);
}
```

```
public void DrawLine (Vector3[][] points, Color color, float width, float dottedSpace=0, float offset=0.01f, in
/// Draws multiple lines
{
 SetPointsThread(points, numPoints);
 SetPointsApply();
 DrawLine(color, width, dottedSpace:dottedSpace, offset:offset, zMode:zMode, parent:parent);
}


public static void InstantLine (Vector3[] points, Color color, float width, float dottedSpace=0, float offset=0.
/// Creates a new polyline, sets points and draws it at once
{
 if (numPoints < 0) numPoints = points.Length;

 PolyLine line = new PolyLine(numPoints);
 line.SetPoints(points, numPoints:numPoints);
 line.DrawLine(color, width, dottedSpace:dottedSpace,  offset:offset, zMode:zMode);
}


private static Mesh TestMesh ()
{
 Mesh mesh = new Mesh();
 mesh.vertices = new Vector3[] { new Vector3(0,0,0), new Vector3(0,0,0.01f), new Vector3(10,0,0.01f), ne
 mesh.normals = new Vector3[] { new Vector3(-10,0,0), new Vector3(-10,0,0), new Vector3(20,0,0), new V
 mesh.uv = new Vector2[] { new Vector2(-1,-1), new Vector2(-1,1), new Vector2(1,1), new Vector2(1,-1) };
```

```
   mesh.uv2 = new Vector2[] { new Vector2(0,0), new Vector2(0,0), new Vector2(1,10), new Vector2(1,10) }
   mesh.triangles = new int[] { 0,2,1,1,2,3 };

   return mesh;
  }

  public static void DrawTest ()
  {
   Mesh mesh = TestMesh();

   if (lineMat==null) lineMat = new Material( Shader.Find("Hidden/DPLayout/PolyLine") );
   if (lineTex==null) lineTex = Resources.Load("DPUI/PolyLineTex") as Texture2D;

   lineMat.SetTexture("_MainTex", lineTex);
   lineMat.SetFloat("_Width", 10);
   lineMat.SetInt("_ZTest", 0);
   lineMat.SetColor("_Color", Color.red);

   lineMat.SetPass(0);
   Graphics.DrawMeshNow(mesh, Matrix4x4.identity);
  }
 }
}
```

```
ï»¿using System;
using UnityEngine;
using UnityEditor;

namespace Den.Tools.SceneEdit
{
 public static class Select
 {
  public static bool isDragging; //mouse pressed and not yet released
  public static bool isFrame;  //true if dragging 5 pixels from the original
  public static bool justStarted;  //drag was started. isDragging is true
  public static bool justReleased;  //frame was released. Not fired if frame was not started. Both isDragging

  public static Rect frameRect;
  public static Vector2 origFramePos; //can't use screenRect position (since mouse could be moved up left)

  public static void UpdateFrame ()
  {
   Event eventCurrent = Event.current;
   bool isAlt = eventCurrent.alt;

   //just pressed
   if (eventCurrent.type==EventType.MouseDown  &&  eventCurrent.button==0  &&  !isAlt)
   {
    justStarted = true;
    isDragging = true;
```

```
 origFramePos = eventCurrent.mousePosition;
 frameRect = new Rect(origFramePos, new Vector2(0,0));

 //eventCurrent.Use();
 if (UnityEditor.EditorWindow.focusedWindow != null) UnityEditor.EditorWindow.focusedWindow.Repaint(

 return;
}

//just released
if (eventCurrent.rawType == EventType.MouseUp) //any button, any drag state
{
 if (isFrame)
 {
  justReleased = true;

  //if (Event.current.isMouse) eventCurrent.Use();
  if (UnityEditor.EditorWindow.focusedWindow != null) UnityEditor.EditorWindow.focusedWindow.Repaint

  //isDragging = false;  //disabling next frame
  //isFrame = false;
 }
 else
 {
  isDragging = false;
  isFrame = false;
```

```
 }



 return;
}

//frame after released - disabling both released and dragged
if (justReleased)
{
 isFrame = false;
 isDragging = false;
 justReleased = false;
}

//creating frame if dragged 5 pixels from origin
if (isDragging  &&  !isFrame  &&  !isAlt)
{
 if ((eventCurrent.mousePosition - origFramePos).sqrMagnitude > 250)
  isFrame = true;
}



if (isFrame  &&  !eventCurrent.alt)
{
 Vector2 max = Vector2.Max(eventCurrent.mousePosition, origFramePos);
 Vector2 min = Vector2.Min(eventCurrent.mousePosition, origFramePos);
```

```
  frameRect = new Rect(min, max-min);

  DrawSelectionFrame (frameRect);

  //eventCurrent.Use();
  if (UnityEditor.EditorWindow.focusedWindow != null) UnityEditor.EditorWindow.focusedWindow.Repaint(

  return;
 }

 //making frame equal to mouse cursor if not dragging
 if (!isDragging)
  frameRect = new Rect(eventCurrent.mousePosition, Vector2.zero);
}

public static void CancelFrame ()
{
 isFrame = false;
 isDragging = false;
 justStarted = false;
 justReleased = false;

 frameRect = new Rect(0,0,0,0);
 origFramePos = new Vector2(0,0);
}
```

```
static readonly Color offsetSizeRectColor = new Color(0.7f,0.8f, 0.9f, 1);
static readonly Color offsetSizeRectColorTransparent = new Color(0.3f,0.5f, 0.9f,0.125f);

public static void DrawSelectionFrame (Rect rect)
{
 Vector3[] steps = new Vector3[] {
  new Vector3 (rect.position.x, rect.position.y, 0),
  new Vector3 (rect.position.x+rect.width, rect.position.y, 0),
  new Vector3 (rect.position.x+rect.width, rect.position.y+rect.height, 0),
  new Vector3 (rect.position.x, rect.position.y+rect.height, 0),
  new Vector3 (rect.position.x, rect.position.y, 0) };

 for (int i=0; i<steps.Length; i++)
 {
  Ray worldRay = HandleUtility.GUIPointToWorldRay(steps[i]);
  Vector3 worldPos = worldRay.origin + worldRay.direction*100;
  steps[i] = worldPos;
 }

 //EditorGUI.DrawRect(rect, new Color(1,0,0,1));
 //UnityEngine.GUI.Button(new Rect(10,10,100,100), "Test");

 Color hColor = Handles.color;
 Handles.color = new Color(1, 1, 1, 1);
```

```
 UnityEditor.Handles.DrawSolidRectangleWithOutline(steps, offsetSizeRectColorTransparent, offsetSizeR

 Handles.color = hColor;
}



public static bool objSelected;

public static void CheckSelected (ref bool origSelected, ref bool dispSelected, Predicate<Rect> isInFrame
/// Selects an object with by his screenRect area. Both by click and by frame
/// Frame should be Updated beforehand
/// origSelected is the "real" selection, dispSelected is for displaying as selected
{
 dispSelected = origSelected;

 Event eventCurrent = Event.current;
 EventType eventType = eventCurrent.type;
 Vector2 mousePos = eventCurrent.mousePosition;

 if (eventCurrent.button != 0  ||  eventCurrent.alt) return;

 bool add = eventCurrent.shift;
 bool remove = eventCurrent.control;

 //direct click
 if (eventType == EventType.MouseDown)
```

```
{
 bool clicked = isInFrame(new Rect(mousePos,Vector2.zero)); //screenRect.Contains(mousePos);

 if (!add && !remove)
  origSelected = clicked;

 if (add)
  origSelected = origSelected || clicked;

 if (remove)
  if (clicked) origSelected = false;

 dispSelected = origSelected;

 if (clicked)
 // eventCurrent.Use(); //to avoid clicking next node underneath
  objSelected = true; //using this instead of Use, otherwise further objects will not be deselected
}
else
 objSelected = false;

//selection frame
if (isFrame)
{
 bool inFrame = isInFrame(Select.frameRect); //Select.frameRect.Contains(screenRect.center);
```

```
  if (!add && !remove)
  {
   if (inFrame) dispSelected = true;
   else dispSelected = false;
  }

  if (add  &&  inFrame)
   dispSelected = true;

  if (remove  &&  inFrame)
   dispSelected = false;

  //setting real selection on release
  if (justReleased)
   origSelected = dispSelected;

 }
}


public static void CheckSelected (Rect screenRect, ref bool origSelected, ref bool dispSelected)
/// Simplified CheckSelected if screen rect is already known
{
 CheckSelected(ref origSelected, ref dispSelected, fr => IsInFrame(screenRect, fr));
}
```

```
 public static void CheckSelected (Vector3 worldPos, float screenExt, ref bool origSelected, ref bool dispS
 /// Same CheckSelected, but using world position instead of screen rect
 {
  CheckSelected(ref origSelected, ref dispSelected, fr => {
   Vector2 screenPos = HandleUtility.WorldToGUIPoint(worldPos);
   Rect screenRect = new Rect(screenPos.x-screenExt, screenPos.y-screenExt, screenExt*2, screenExt*2
   return IsInFrame(screenRect, fr);
  });
 }


 private static bool IsInFrame (Rect screenRect, Rect frameRect)
 {
  //if just click with no frame
  if (frameRect.width<0.1f && frameRect.height<0.1f)
   return screenRect.Contains(frameRect.position); //if click is within screen rect

  //if dragging fame
  else
   return frameRect.Contains(screenRect.center); //if screen rect center is within frame
 }

}
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using System.Text;
using UnityEngine;

namespace Den.Tools.Serialization
{
 public partial class Serializer
 {
  public static object Deserialize (string str, IList<UnityEngine.Object> unityObjs=null)
  {
   if (str == "null")
    return null;

   object[] splitString = (object[])Tools.SplitBlock(str, ',', '<', '>');
   splitString = (object[])splitString[0]; //initially will get object[1]
   return DeserializeRecursive(splitString, new Dictionary<int,object>(), unityObjs).val;
  }


  private static (string name, object val) DeserializeRecursive (object[] splitString, Dictionary<int,object> des
  /// Returns field name with a variable that should be assigned here
  {
   //check adding to deserializedIdsObjs if id!=0 before each return!
```

```
    string name = ((string)splitString[0]).Trim("");



    //finding type
    Type type = null;
    string typeLine = (string)splitString[1];
    //typeLine = typeLine.Replace(';', ',');
    typeLine = typeLine.Trim("");
    type = Type.GetType(typeLine);

    if (type == null) //not exist anymore
//    throw new Exception("Could not find type for: " + typeLine);
{ Debug.LogError("Could not find type for: " + typeLine); return (name,null); }
//TODO: should be exception

    //id
    int id = 0; //0 is no id
    if (!type.IsPrimitive)
     id = int.Parse( ((string)splitString[2]).TrimStart(new []{'i','d'}) );

    if (splitString.Length<=3  &&  deserializedIdsObjs.TryGetValue(id, out object serVal)) //already serialized
     return (name, serVal);
     //return only if serializedand no data since we can append already loaded objects with other fields (split c
```

```
//value: most common primitives
if (type == typeof(float))
 return (name, float.Parse( (string)splitString[3], System.Globalization.CultureInfo.InvariantCulture.Numbe
else if (type == typeof(int))
 return (name, int.Parse( (string)splitString[3] ));



//guid
else if (type == typeof(Guid))
{
 Guid guid = Guid.Parse((string)splitString[3]);
 return (name, guid);
}



//other primitives
else if (type.IsPrimitive)
{
 object val = Convert.ChangeType((string)splitString[3], Type.GetTypeCode(type));
 return (name, val);
}



//null
else if (splitString.Length == 4  &&  splitString[3] is string strFrstVal  &&  strFrstVal == "null")
 return (name, null);
```

```
//string
else if (type == typeof(string))
{
 if (deserializedIdsObjs.TryGetValue(id, out object val))
  return (name, val);

 else
 {
  string sval = ((string)splitString[3]).Trim("");
  deserializedIdsObjs.Add(id, sval);
  return (name, sval);
 }
}


//unity object
else if (type.IsSubclassOf(typeof(UnityEngine.Object)))
{
 //are serialized as value, with no reference
 //they are wrote to a special array instead. No way to deserialize their GUID - editor only

 if (deserializedIdsObjs.TryGetValue(id, out object val))
  return (name, val);
```

```
 string strVal = (string)splitString[3];
 if (strVal == "null")
  return (name, null);
 else
 {
  int index = int.Parse(strVal);
  return (name,unityObjs[index]);
 }
}



//reflections
else if (type.IsSubclassOf(typeof(Type))) //could be MonoType
{
 Type tVal = Type.GetType( ((string)splitString[3]).Trim("") );
 if (id!=0) deserializedIdsObjs.Add(id, tVal);
 return (name,tVal);
}


else if (type.IsSubclassOf(typeof(MemberInfo)))
{
 Type mt = Type.GetType(((string)splitString[4]).Trim(""));
 MemberInfo mi = mt.GetField(((string)splitString[3]).Trim(""), BindingFlags.Public | BindingFlags.NonPub
 if (id!=0) deserializedIdsObjs.Add(id, mi);
 return (name, mi);
}
```

```
//array
else if (type.IsArray)
{
 if (deserializedIdsObjs.TryGetValue(id, out object val))
  return (name, val);

 Array array = (Array)Activator.CreateInstance(type, splitString.Length-3);
 deserializedIdsObjs.Add(id, array); //assigning object now in case we will have reference to it (array of its

 Type elementType = type.GetElementType();
 TypeCode elementTypeCode =  Type.GetTypeCode(elementType);

 for (int i=0; i<array.Length; i++)
 {
  object serElement = splitString[i+3];

  if (serElement is string stringElement)
  {
   object elVal = Convert.ChangeType(stringElement, elementTypeCode);
   array.SetValue(elVal, i);
  }

  else if (serElement is object[] arrElement)
  {
```

```
   (string elName, object elVal) = DeserializeRecursive(arrElement, deserializedIdsObjs, unityObjs);
   array.SetValue(elVal, i);
  }
 }

 //adding to deserializedIdsObjs in the beginning
 return (name, array);
}



//classes/structs
else
{
 //getting value - do not return since we can add some fields
 if (!deserializedIdsObjs.TryGetValue(id, out object val))
 {
  //if (val is ICustomSerialization customObj)
  // customObj.PreprocessBeforeDeserialize(serObj, serialized, deserialized);

  val = Activator.CreateInstance(type);
  if (id != 0) deserializedIdsObjs.Add(id, val);  //assigning object now in case we will have reference to it
 }

 //for (int f=splitString.Length-1; f>=3; f--) //deserializing from end to start, to pass NewOverridedTests
 for (int f=3; f<splitString.Length; f++)  //deserializing from the beginning to deserialize ordered dicts (amd
 {
```

```
(string fieldName, object fieldVal) = DeserializeRecursive( (object[])splitString[f], deserializedIdsObjs, ur

MemberInfo member = type.GetField(fieldName, BindingFlags.Public | BindingFlags.NonPublic | Bindin

if (member==null) //trying to load from baser type (for classes derived from list and dict)
 member = type.BaseType.GetField(fieldName, BindingFlags.Public | BindingFlags.NonPublic | Binding

if (member==null) //serialized property for animation curve or other special types
 member = type.GetProperty(fieldName, BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.In

if (member==null) //Unity 2021.2 beta dictionary field names have _ prefix, previous ones - don't
{
 if (fieldName == "buckets" || fieldName == "entries" || fieldName == "count" || fieldName == "comparer"
  fieldName = "_" + fieldName;

 member = type.GetField(fieldName, BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Insta
 if (member==null) member = type.BaseType.GetField(fieldName, BindingFlags.Public | BindingFlags.N
}

if (member==null)  //if new variable was set in code at this position
#if !MM_DEBUG
 continue;
#else
 throw new Exception ("Deserialize: Field " + fieldName + " not found");
#endif
```

```
      if (member is FieldInfo field)
      {
       if (field.IsNotSerialized) continue; //although it was serialized when data was saved, it could be not seria
       field.SetValue(val, fieldVal);
      }

      else if (member is PropertyInfo prop)
       prop.SetValue(val, fieldVal);
     }

     //adding to deserializedIdsObjs in the beginning
     return (name, val);
    }


    throw new Exception("Could not deserialize:\n" + splitString);
   }
  }


}
```

```
ï»¿using System;
using System.Reflection;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;

namespace Den.Tools.Serialization
{
 public static class Tools
 {
  public static string ReadBlock (string str, int start, char blockOpen, char blockClose, bool skipStrings=true
  /// Reads some block (area marked with open and close symbols) block with internal blocks: (A, B(c,d,e),
  /// Block start and end symbols (like '(' ) are included in return string. It starts with '(' and ends with ')'
  {
   int blockOpenings = 0;
   bool blockStarted = false; //have we ever reached block start (return on blockOpenings==0 if so)
   bool stringOpened = false; //don't process if we've met "

   List<char> blockChars = new List<char>();

   for (int i=start; i<str.Length; i++)
   {
    char ch = str[i];

    if (skipStrings && ch=='"')
```

```
{
 if (stringOpened  &&  i!=0  &&  str[i-1]=='\\') //it's " inside the string
  { blockChars.Add(""); continue; }

 stringOpened = !stringOpened;
}

if (!stringOpened)
{
 if (ch==blockOpen)
 {
  blockOpenings ++;
  blockStarted = true;
 }

 if (blockStarted)
 {
  blockChars.Add(ch);

  if (ch==blockClose)
   blockOpenings --;

  if (blockOpenings == 0)
   break;
 }
}
```

```
  else
    blockChars.Add(ch);
 }

 return new string(blockChars.ToArray());
}


public static object SplitBlock (string str, char splitSymbol, char blockOpen, char blockClose, bool skipStri
/// Splits one string in block arrays: [string, string, [string,string,string], [string,string], string]
/// Returns either one string, or object[] if block has internal ones
/// Open/close and split symbols are not included
/// Note that open and close symbols kinda treated like split symbols too
/// skipStrings will take internal strings into account and will not use open/close/split symbol from them
/// trim will remove spaces for each member
{
 bool stringOpened = false; //don't process if we've met ''

 List<object> currMember = new List<object>();
 List<char> currString = new List<char>();

 Stack< List<object> > upperMembers = new Stack< List<object> >();
 Stack< List<char> > upperStrings = new Stack< List<char> >();

 for (int i=0; i<str.Length; i++)
```

```
{
 char ch = str[i];

 //checking if internal string is opening
 if (skipStrings && ch=="")
 {
  if (stringOpened  &&  i!=0  &&  str[i-1]=='\\') //it's " inside the string
   { currString.Add(""); continue; }

  stringOpened = !stringOpened;
 }

 //just reading internal string if it's opened
 if (stringOpened)
  currString.Add(ch);

 //common case (non-string)
 else
 {
  if (ch==splitSymbol)
  {
   AddString(currMember, currString, trim);
   currString.Clear();
  }

  else if (ch==blockOpen)
```

```
  {
   upperMembers.Push(currMember);
   currMember = new List<object>();

   upperStrings.Push(currString);
   currString = new List<char>();
  }

  else if (ch==blockClose)
  {
   AddString(currMember, currString, trim);
   object[] intMember = currMember.ToArray();

   currMember = upperMembers.Pop();
   currString = upperStrings.Pop();

   currMember.Add(intMember);
  }

  else
   currString.Add(ch);
 }

}

//finalizing current member
```

```
 AddString(currMember, currString, trim);
 return currMember.ToArray();
}



private static void AddString (List<object> strList, List<char> charArr, bool trim=true)
/// Adds char array to string list (as new string) if it has anything
/// Called on all charArr closings (split symbol, close symbol, end)
{
 string aplString = new string(charArr.ToArray());
 if (trim) aplString = aplString.Trim();
 if (aplString.Length!=0)
  strList.Add(aplString);
}



public static int CheckEquality (object srcObj, object dstObj, HashSet<object> used=null)
/// Testing if the class was copied right
{
 if (used == null) used = new HashSet<object>();
 if (used.Contains(srcObj)) return 0;
 used.Add(srcObj);
 int numChecked = 1;

 if (srcObj==null && dstObj==null) return 1;
 if (srcObj==null && dstObj!=null) throw new Exception("CheckEquality: Src is null while dst is " + dstObj);
```

```
if (srcObj!=null && dstObj==null) throw new Exception("CheckEquality: Dst is null while src is " + srcObj);

Type type = srcObj.GetType();

if (type.IsSubclassOf(typeof(UnityEngine.Object))) return 0;
if (type.IsSubclassOf(typeof(MemberInfo))) return 0; //do not check reflections - they are same
if (typeof(IDictionary).IsAssignableFrom(type)) return 0;

if (type != dstObj.GetType())
 throw new Exception("CheckEquality: Types differ " + type + " and " + dstObj.GetType());

if (!type.IsValueType  &&  type!=typeof(string)  &&  srcObj==dstObj)
  throw new Exception("CheckEquality: Are same " + srcObj);

//array
if (type.IsArray)
{
 Array srcArray = (Array)srcObj;
 Array dstArray = (Array)dstObj;

 if (type.GetElementType().IsValueType)
 {
  for (int i=0; i<srcArray.Length; i++)
   if (!srcArray.GetValue(i).Equals(dstArray.GetValue(i)))
    throw new Exception("CheckEquality: arrays not equal "  + srcObj + " and " + dstArray);
 }
```

```
 else
  for (int i=0; i<srcArray.Length; i++)
   numChecked += CheckEquality(srcArray.GetValue(i), dstArray.GetValue(i), used);
}

//common case
else
{
  //fields
  FieldInfo[] fields = type.GetFields(BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instance
  for (int i=0; i<fields.Length; i++)
  {
   FieldInfo field = fields[i];

   if (field.IsLiteral) continue; //leaving constant fields blank
   if (field.FieldType.IsPointer) continue; //skipping pointers (they make unity crash. Maybe require unsafe
   //if (field.IsNotSerialized) continue;

   if (field.FieldType.IsValueType)
   {
    if (!field.GetValue(srcObj).Equals(field.GetValue(dstObj)))
     throw new Exception("CheckEquality: not equal " + field.Name + " " + field.GetValue(srcObj) + " and "
   }
   else
   {
```

```
        numChecked += CheckEquality(field.GetValue(srcObj), field.GetValue(dstObj), used);
      }
     }
    }


    return numChecked;
   }
  }
 }
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using System.Reflection;
using System.Text;
using UnityEngine;

[assembly:System.Runtime.CompilerServices.InternalsVisibleTo("Tests.Editor")]

namespace Den.Tools.Serialization
{
 public partial class Serializer
 {
  // Serailization format:
  // <FieldName, <TypeFullName, AssemblyName>, Id, SubField1, SubField2, ..., SubFieldX>
  // id is the number of serialized/deserialized class in serializedObjsIds dict

  // Serialized foo object string
  //   < class0, <Den.Tools.CoordRect, Tools>, 0,
  //  < offset, <Den.Tools.Coord, Tools>, 0,
  //    <x, System.Single, 0, 3 >,
  //    <z, System.Single, 0, 42 > >,
  //  < size, <Den.Tools.CoordWithRef, Tools>, 0,
  //    <x, System.Single, 3 >,
  //    <z, System.Single, 42 >,
  //    <reference, <SomeType, Tools>, 1 > >,
```

```
//  < selfReference, ref, 0 >,
//  < vector, <UnityEngine.Vector2, UnityEngine.CoreModule>, 3.1415, 42.2 >,
//  < floatVal, System.Single, 3.1415 >,
//  < floatArr, System.Single[], <1,2,3,4,5> >,
//  < refArr, ref[], <1,1,1,1,1> >,
//  < vectorArr,
//    <UnityEngine.Vector2[], UnityEngine.CoreModule>,
//    < 1, <UnityEngine.Vector2, UnityEngine.CoreModule>, 3.1415, 42.2 >,
//    < 2, <UnityEngine.Vector2, UnityEngine.CoreModule>, 3.1415, 42.2 > > >,
//   < class1,
//  <SomeType, Tools>,
//  <x, System.Single, 3> >



// TODO:
// Find out what is the best way to serialize referenced objects:
// Ideas: serialize all as it is (with inlined class), assign class ids, and then just link to inlined classes (how

public static string Serialize (object val)
/// Serializes all to string without Unity assets
{
 if (val==null)
  return "null";

 return SerializeRecursive("root", val, new Dictionary<object,int>(), null);
}
```

```
public static string Serialize (object val, out UnityEngine.Object[] unityObjs)
/// This call will serialize all with assets. UnityObjs array should be saved as well.
{
 if (val==null)
  { unityObjs = new UnityEngine.Object[0]; return "null"; }

 Dictionary<object,int> serializedObjsIds = new Dictionary<object,int>();
 List<UnityEngine.Object> unityObjsList = new List<UnityEngine.Object>();

 string str = SerializeRecursive("root", val, serializedObjsIds, unityObjsList);

 unityObjs = unityObjsList.ToArray();
 return str;
}


private static string SerializeRecursive (string name, object val,
 Dictionary<object,int> serializedObjsIds, //all ids are 1-based. 0 means "don't look up - it's value or null"
 List<UnityEngine.Object> unityObjs,
 string offset="")
// If serializedVals defined will assign index to reference values. Otherwise will set them to null.
// Will expand serialized object with new reference objects
// BTW fastest way to create string is "x"+"y"+"z". Interpolated string $"{x}{y}{z}" just a bit slower, but more
{
```

```
//using ifs instead of switch since we need to check them all in this order, and some are same (like IList a
//name should be in " since some names (automatic) have < and > in them

//null
if (val == null)
{
 string typeName = "System.Object"; //nullType!=null ? TypeName(nullType) : "System.Object"; //all nulls
 return $"{offset}< \"{name}\", {typeName}, id0, null >";
}

//most common types
switch (val)
{
 case float fVal: return $"{offset}< \"{name}\", System.Single, id0, {fVal.ToString("R")} >";  //The round-trip
 case double dVal: return $"{offset}< \"{name}\", System.Double, id0, {dVal.ToString("R")} >"; //same for d
 case int iVal: return $"{offset}< \"{name}\", System.Int32, id0, {iVal.ToString()} >";
}

//loading type only now (to speed up all above)
Type type = val.GetType(); //at this point val should not be null
//bool isClass = type.IsClass; //IsClass is _extremely_ long operation (10 times longer than IsPrimitive, 10

//already serialized (no mater of it's array, string, other class)
if (serializedObjsIds!=null  &&  serializedObjsIds.TryGetValue(val, out int existingId))
 return $"{offset}< \"{name}\", \"{TypeName(type)}\", id{existingId} >";
```

```
//string
if (val is string)
{
 int id = serializedObjsIds.Count + 1;
 serializedObjsIds.Add(val, id); //checking if it was serialized in the beginning

 return $"{offset}< \"{name}\", System.String, id{id}, \"{val}\" >";
}

//guid
if (val is Guid gVal)
 return $"{offset}< \"{name}\", System.Guid, id0, {gVal.ToString()} >";

//other primitives (IsPrimitive is rather slow)
if (type.IsPrimitive)
 return $"{offset}< \"{name}\", {type.FullName}, id0, {val} >";


//unity object
if (val is UnityEngine.Object)
{
 //are serialized as value, with no reference
 //they are wrote to a special array instead. No way to serialize their GUID - editor only

 UnityEngine.Object uObj = (UnityEngine.Object)val;
```

```
 if (unityObjs==null)
  return $"{offset}<  \"{name}\", \"{TypeName(type)}\", id0, null >";

 if (uObj == (UnityEngine.Object)null)
  return $"{offset}<  \"{name}\", \"{TypeName(type)}\", id0, null >";
 //doesnt work in deserialize thread, so here is workaround (but seems to be working in serialize):
 //if (type == typeof(UnityEngine.Object))
 //yet don't know a case when pure Object could be assigned, so considering it as null

 if (unityObjs.Contains(uObj, out int index))
  return $"{offset}<  \"{name}\", \"{TypeName(type)}\", id0, {index} >";

 else
 {
  unityObjs.Add(uObj);
  return $"{offset}<  \"{name}\", \"{TypeName(type)}\", id0, {unityObjs.Count-1} >"; //unityObjs.Count-1+1 a
 }
}


//reflections
if (val is MemberInfo mi)
{
 int id = serializedObjsIds.Count + 1;
 serializedObjsIds.Add(val, id); //checking if it was serialized in the beginning
```

```
 if (val is Type typeVal)
  return $"{offset}< \"{name}\", \"{TypeName(type)}\", id{id}, \"{TypeName(typeVal)}\" >";

 else
  return $"{offset}< \"{name}\", \"{TypeName(type)}\", id{id}, \"{mi.Name}\", \"{TypeName(mi.DeclaringType
}

//array
if (val is Array array)
{
 Builder builder = new Builder(name, val, serializedObjsIds, offset);
 using (builder)
 {
  Type elementType = type.GetElementType();

  if (elementType.IsPrimitive)
  {
   builder.AddSpace(); //space after id
   if (elementType == typeof(float)) //R case for float
    for (int i=0; i<array.Length; i++)
     builder.Add(((float)array.GetValue(i)).ToString("R"));

   else //other primitive types
    for (int i=0; i<array.Length; i++)
     builder.Add(array.GetValue(i).ToString());
  }
```

```
  else
   for (int i=0; i<array.Length; i++)
   {
    object subVal = array.GetValue(i);
    Type subType = subVal!=null ? subVal.GetType() : elementType; //need to serialize real type, field typ

    string subSer = SerializeRecursive(i.ToString(), subVal, serializedObjsIds, unityObjs, offset+"  ");

    builder.AddLine(subSer);
   }
 }
 return builder.ToString();
}

//class/struct
//else //if not returned on anything else
{
 Builder builder = new Builder(name, val, serializedObjsIds, offset:offset, addId:!type.IsValueType);
 using (builder)
 {
  if (val is ISerializationCallbackReceiver serReciever)
   serReciever.OnBeforeSerialize();

  //FieldInfo[] fields = type.GetFields(BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instanc
```

```
   foreach (MemberInfo member in Fields(type,val))
   {
    object subVal = null; Type subNullType = null;

    if (member is FieldInfo field)
     { subVal = field.GetValue(val); subNullType = field.FieldType; } //Emit only available in editor
    else if (member is PropertyInfo prop)
     { subVal = prop.GetValue(val); subNullType = prop.PropertyType; }

    string subSer = SerializeRecursive(member.Name, subVal, serializedObjsIds, unityObjs, offset+"  ");//,

    builder.AddLine(subSer);
   }
  }
  return builder.ToString();
 }
}



private static IEnumerable<MemberInfo> Fields (Type type, object val)
/// Iterates proper fields for an object depending on it's type
/// For most cases it will be type.GetFields (BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.I
/// TODO: cache fields by type
{
 Type baseType = type.BaseType;
 if (baseType != typeof(object)  &&  baseType != typeof(ValueType))
```

```
 foreach (FieldInfo field3 in Fields(baseType, val))
  yield return field3;

BindingFlags bind =
 BindingFlags.DeclaredOnly |  //only top-level fields, other ones were listed before
 BindingFlags.Public |
 BindingFlags.NonPublic |
 BindingFlags.Instance;

if (val is IList  &&  baseType == typeof(object)) //list itself, not it's inherited type
{
 yield return type.GetField("_items", bind);
 yield return type.GetField("_size", bind);
 yield return type.GetField("_version", bind); //the number of change made with list
}

else if (val is IDictionary  &&  baseType == typeof(object))
{
 if (type.GetField("_buckets", bind) != null)
 {
  yield return type.GetField("_buckets", bind);
  yield return type.GetField("_entries", bind);
  yield return type.GetField("_count", bind);
  yield return type.GetField("_comparer", bind);
  yield return type.GetField("_version", bind);
 }
```

```
 else
 {
  yield return type.GetField("buckets", bind);
  yield return type.GetField("entries", bind);
  yield return type.GetField("count", bind);
  yield return type.GetField("comparer", bind);
  yield return type.GetField("version", bind);
 }
}

else if (val is AnimationCurve  &&  baseType == typeof(object))
{
 yield return type.GetProperty("keys", bind);
 yield return type.GetProperty("preWrapMode", bind);
 yield return type.GetProperty("postWrapMode", bind);
}

else
{
 FieldInfo[] fields = type.GetFields(bind);

 foreach (FieldInfo field in fields)
 {
  if (field.IsNotSerialized) continue;
  if (field.IsLiteral) continue; //leaving constant fields blank
  if (field.FieldType.IsPointer) continue; //skipping pointers - obviously they make Unity crash (at first glan
```

```
   yield return field;
  }
 }
}


internal class Builder : IDisposable
/// Disposable StringBuilder
{
 StringBuilder builder;

 public Builder (string name, object val, Dictionary<object,int> serializedObjsIds, string offset="", bool addI
 {
  int id = 0;
  if (addId)
  {
   id = serializedObjsIds.Count + 1;
   serializedObjsIds.Add(val, id); //checking if it was serialized in the beginning of SerializeREcursive
  }

  if (val is ISerializationCallbackReceiver serReciever)
   serReciever.OnBeforeSerialize();

  builder = new StringBuilder();
  builder.Append($"{offset}< \"{name}\", \"{TypeName(val.GetType())}\", id{id}");
```

```
 }

 public void Dispose ()
 {
  builder.Append(" >");
 }

 public void AddLine (string s)
 // Not confuse with StringBuilder.AppendLine! Here line symbol goes first, and adds comma as well
 {
  builder.AppendLine(","); //, and \n goes before line. This way it makes , after id and does not make , at th
  builder.Append(s);
 }

 public void Add (string s) { builder.Append(","); builder.Append(s); }

 public void AddSpace () { builder.Append(" "); }

 public override string ToString () => builder.ToString();
}


private static string TypeName (Type type, char div=',', bool withAssembly=true)
/// Returns properly formated type
/// Generics non-asm qualified aw well - without culture, version, etc
/// Using custom char div instead of comma
```

```
/// TODO: cache type names
{
 /*string declaringTypeName = "";
 Type declaringType = type.DeclaringType;
 while (declaringType != null)
 {
  declaringTypeName = $"{declaringType.Name}+{declaringTypeName}"; //+ after declaring type
  declaringType = declaringType.DeclaringType;
 }

 string str = $"{type.Namespace}.{declaringTypeName}{type.Name}{div} {type.Assembly.GetName().Nam
 //not reliable

 string typeName = type.ToString();

 if (type.IsArray)
 {
  typeName = $"{TypeName(type.GetElementType(), div, withAssembly:false)}[]";

  if (withAssembly)
   typeName = $"{typeName}{div} {type.Assembly.GetName().Name}";

  return typeName;
 }

 if (type.IsGenericType)
```

```
  {
   /*typeName = typeName.Substring(0, typeName.IndexOf('`', 0)); //note it's not the standard apostrophe

   Type[] subTypes = type.GenericTypeArguments;
   typeName += $"`{subTypes.Length}[";*/

   typeName = typeName.Substring(0, typeName.IndexOf('[', 0)+1);

   Type[] subTypes = type.GenericTypeArguments;
   for (int s=0; s<subTypes.Length; s++)
   {
    typeName += $"[{TypeName(subTypes[s])}]";
    if (s!=subTypes.Length-1)
     typeName += div;
   }

   typeName += ']';
  }

  if (withAssembly)
   typeName = $"{typeName}{div} {type.Assembly.GetName().Name}";

  return typeName;
 }
}
}
```

```
ï»¿using UnityEngine;


using System;
using System.Collections;
using System.Collections.Generic;




namespace Den.Tools.Splines
{
 [System.Serializable]
 public class Line
 {
  public Segment[] segments = new Segment[0];

  public bool looped;
  public float length;

  public Vector3 startPos
  {
   get => segments[0].start.pos;
   set => segments[0].start.pos = value;
  }

  public Vector3 endPos
  {
```

```
 get => segments[segments.Length-1].end.pos;
 set => segments[segments.Length-1].end.pos = value;
}

public Line () {}
public Line (Line src)
{
 segments = new Segment[src.segments.Length];
 Array.Copy(src.segments, segments, segments.Length);

 looped = src.looped;
 length = src.length;
}

public Line (Vector3 start, Vector3 end)
 { segments = new Segment[] { new Segment(start,end) }; }


public Vector3 GetPoint (int segNum, float segPercent)
{
 return segments[segNum].GetPoint(segPercent);
}

public void SetSegmentsCount (int count)
 { Array.Resize(ref segments, count); }
```

```
public void InsertSegment (int index, Segment segment)
 { ArrayTools.Insert(ref segments, index, segment); }


public void AddSegment (Segment segment)
 { ArrayTools.Add(ref segments, segment); }




public void SetAllTangentTypes (Node.TangentType type)
{
 for (int s=0; s<segments.Length; s++)
 {
  segments[s].start.type = type;
  segments[s].end.type = type;
 }
}


#region Nodes Operations

 // segments (3):     0  1  2
 // nodes (4):  0 ----- 1 ----- 2 ----- 3
```

```
public int NodesCount
{get{
 if (segments.Length == 0) return 0;
 if (looped) return segments.Length;
 else return segments.Length+1;
}}

public void InsertNode (int n, Node node)
{
 InsertSegment(n-1, new Segment(segments[n-1].start, node));
 segments[n].start = node;
 segments[n].start.dir = -segments[n].start.dir;
}

public void RemoveNode (int n)
{
 if (n == segments.Length) ArrayTools.RemoveAt(ref segments, segments.Length-1);
 else if (n == 0) ArrayTools.RemoveAt(ref segments, 0);
 else
 {
  Segment joined = Segment.Join(segments[n-1], segments[n]);
  segments[n-1] = joined;
  ArrayTools.RemoveAt(ref segments, n);
 }
}
```

```
public void AddNode (Node node)
{
 AddSegment(new Segment(segments[segments.Length-1].end, node));
}

public void SetNodes (Vector3[] poses)
{
 segments = new Segment[poses.Length-1];
 for (int s=0; s<segments.Length; s++)
  segments[s] = new Segment(poses[s], poses[s+1]);
}

public Vector3 GetNodePos (int n)
{
 if (looped) n = n % segments.Length;
 if (n < segments.Length) return segments[n].start.pos;
 else return segments[n-1].end.pos;
}

public void SetNodePos (int n, Vector3 pos)
{
 if (n != segments.Length)
  segments[n].start.pos = pos;
 if (n != 0)
  segments[n-1].end.pos = pos;
}
```

```
 public Vector3? GetNodeInTangent (int n)
 {
  if (n == 0) return null;
  return segments[n-1].end.dir;
 }

 public Vector3? GetNodeOutTangent (int n)
 {
  if (n == segments.Length) return null;
  else return segments[n].start.dir;
 }

#endregion


#region Update

 public void Update ()
 /// Called by GUI on every line change and by system on global change
 {
  UpdatePositions();
  UpdateTangents();
  UpdateLength();
 }
```

```
public void UpdatePositions ()
{
 for (int s=0; s<segments.Length-1; s++)
  segments[s].end.pos = segments[s+1].start.pos;
}


public void UpdateTangents ()
{
 for (int n=0; n<segments.Length+1; n++)
  UpdateTangent(n);
}


public void UpdateTangent (int n)
{
 //first
 if (n == 0)
 {
  Node.TangentType type = segments[n].start.type;
  if (type == Node.TangentType.auto || type == Node.TangentType.linear)
   segments[n].start.dir = Node.LinearTangent(segments[n].start.pos, segments[n].end.pos);
  //do nothing for correlated and broken
 }
```

```
//common case
else if (n != segments.Length)
{
 Node.TangentType type = segments[n].start.type;
 segments[n-1].end.type = type;
 if (type == Node.TangentType.broken) return; //do nothing for broken

 Vector3 inDir = segments[n-1].end.dir;
 Vector3 outDir = segments[n].start.dir;

 if (type == Node.TangentType.auto)
  (inDir, outDir) = Node.AutoTangents(segments[n-1].start.pos, segments[n-1].end.pos, segments[n].end

 else if (type == Node.TangentType.linear)
  (inDir, outDir) = Node.LinearTangents(segments[n-1].start.pos, segments[n-1].end.pos, segments[n].en

 else if (type == Node.TangentType.correlated)
  inDir = Node.CorrelatedInTangent(inDir, outDir);

 segments[n-1].end.dir = inDir;
 segments[n].start.dir = outDir;
}

//last
else
{
```

```
  Node.TangentType type = segments[n-1].end.type;
  if (type == Node.TangentType.auto || type == Node.TangentType.linear)
   segments[n-1].end.dir = Node.LinearTangent(segments[n-1].end.pos, segments[n-1].start.pos);
 }
}


public void UpdateLength (int iterations=32, float[] tmp=null)
//much slower than UpdateTangents and not always needed
{
 if (tmp == null) tmp = new float[iterations];

 length = 0;
 for (int s=0; s<segments.Length; s++)
 {
  segments[s].UpdateLength(iterations, tmp);
  length += segments[s].length;
 }
}


public void UpdateLength (int num, int iterations=32)
{
 segments[num].UpdateLength(iterations);
 length = 0;
 for (int s=0; s<segments.Length-1; s++)
```

```
    length += segments[s].length;
  }

#endregion



#region Ops

  public Vector3[] GetAllPoints (float resPerUnit=0.1f, int minRes=3, int maxRes=20)
  /// Converts line into array of points to draw polyline
  /// resPerUnit defines how many points will be created for 1 meter of spline. Then min/maxRes clamps th
  /// Requires length updated
  {
   //calculating number of points
   int numPoints = 0;
   for (int s=0; s<segments.Length; s++)
   {
    int modRes = (int)( segments[s].length * resPerUnit );
    if (modRes < minRes) modRes = minRes;
    if (modRes > maxRes) modRes = maxRes;

    numPoints += modRes;
   }

   Vector3[] points = new Vector3[numPoints + 1];
```

```
 int i=0;
 for (int s=0; s<segments.Length; s++)
 {
  int modRes = (int)( segments[s].length * resPerUnit );
  if (modRes < minRes) modRes = minRes;
  if (modRes > maxRes) modRes = maxRes;

  for (int p=0; p<modRes; p++)
  {
   float percent = 1f*p / modRes;
   points[i] = segments[s].GetPoint(percent);
   i++;
  }
 }

 //the last one
 points[points.Length-1] = segments[segments.Length-1].end.pos;

 return points;
}

public (Vector3[], Vector3[]) GetAllPointsDerivatives (float resPerUnit=0.1f, int minRes=3, int maxRes=20
/// Copy of GetAllPoints, but returns derivatives as well
/// TODO: replace with CalcPointsNum and FillPoints
{
```

```
//calculating number of points
int numPoints = 0;
for (int s=0; s<segments.Length; s++)
{
 int modRes = (int)( segments[s].length * resPerUnit );
 if (modRes < minRes) modRes = minRes;
 if (modRes > maxRes) modRes = maxRes;

 numPoints += modRes;
}

Vector3[] points = new Vector3[numPoints + 1];
Vector3[] derivatives = new Vector3[numPoints + 1];

int i=0;
for (int s=0; s<segments.Length; s++)
{
 int modRes = (int)( segments[s].length * resPerUnit );
 if (modRes < minRes) modRes = minRes;
 if (modRes > maxRes) modRes = maxRes;

 for (int p=0; p<modRes; p++)
 {
  float percent = 1f*p / modRes;
  points[i] = segments[s].GetPoint(percent);
  derivatives[i] = segments[s].GetDerivative(percent);
```

```
    i++;
   }
  }

  //the last one
  points[points.Length-1] = segments[segments.Length-1].end.pos;
  derivatives[points.Length-1] = -segments[segments.Length-1].end.dir;

  return (points, derivatives);
}


public void Split (int s, float p)
{
  (Segment prev, Segment next) = segments[s].GetSplitted(p);
  segments[s] = prev;
  ArrayTools.Insert(ref segments, s+1, next);
}


public void Subdivide (int num)
/// Splits each segment in several shorter segments
{
  Segment[] newSegments = new Segment[segments.Length*num];
  for (int s=0; s<segments.Length; s++)
  {
```

```
  Segment currSegment = segments[s];

  for (int i=0; i<num; i++)
  {
   float percent = 1f / (num-i);
   (Segment,Segment) split = currSegment.GetSplitted(percent);

   newSegments[s*num+i] = split.Item1;
   currSegment = split.Item2;
  }
 }

 segments = newSegments;
}

public (int l, int s, float p, float dist) GetClosest (Vector3 point, int initialApprox=10, int recursiveApprox=1
 GetClosest(null, point, initialApprox, recursiveApprox);

public (int l, int s, float p, float dist) GetClosest (Func<Vector3,Vector3,float> distanceFn, int initialApprox
 GetClosest(distanceFn, initialApprox, recursiveApprox);

public (int l, int s, float p, float dist) GetClosest (Func<Vector3,Vector3,float> distanceFn, Vector3 point, i
{
 float minDist = float.MaxValue;
 int ml=0; int ms=0; float mp=0;
```

```
 for (int s=0; s<segments.Length; s++)
 {
  if (distanceFn==null  &&  !segments[s].IsWithinRange(point, Mathf.Sqrt(minDist)))
   continue;

  (float curPercent, float curDist) = segments[s].GetClosest(distanceFn, point, initialApprox, recursiveApp
  if (curDist < minDist)
  {
   minDist = curDist;
   ms = s;
   mp = curPercent;
  }
 }

 return (ml, ms, mp, minDist);
}

public void Optimize (float deviation)
/// Removes those nodes that should not change the shape a lot
/// Works with auto-tangents only
{
 int iterations = segments.Length-1; //in worst case should remove all but start/end
 if (iterations <= 0) return;
 for (int i=0; i<iterations; i++) //using recorded itterations since nodes count will change
 //for (int i=0; i<deviation; i++)
```

```
{
 float minDeviation = float.MaxValue;
 int minN = -1;

 for (int s=1; s<segments.Length; s++)
 {
  //checking how far point placed from tangent-tangent line
  float currDistToLine = DistanceToLine(
   segments[s-1].start.pos + segments[s-1].start.dir,
   segments[s].end.pos + segments[s].end.dir,
   segments[s].start.pos);
  //float currLine = ((nodes[n-1].pos+nodes[n-1].outDir) - (nodes[n+1].pos+nodes[n+1].inDir)).magnitude;
  //float currDeviation = (currDistToLine*currDistToLine) / currLine;
  float currDeviation = currDistToLine;

  if (currDeviation < minDeviation)
  {
   minN = s;
   minDeviation = currDeviation;
  }
 }

 if (minDeviation > deviation) break;

 segments[minN-1].end = segments[minN].end;
 ArrayTools.RemoveAt(ref segments, minN);
```

```
  UpdateTangents(); //(minN-1);
 }
}

private static float DistanceToLine (Vector3 lineStart, Vector3 lineEnd, Vector3 point)
/// Just helper fn for optimize, isn't related with beizer
{
 Vector3 lineDir = lineStart-lineEnd;
 float lineLengthSq = lineDir.x*lineDir.x + lineDir.y*lineDir.y + lineDir.z*lineDir.z;
 float lineLength = Mathf.Sqrt(lineLengthSq);
 float startDistance = (lineStart-point).magnitude;
 float endDistance = (lineEnd-point).magnitude;

 //finding height of triangle
 float halfPerimeter = (startDistance + endDistance + lineLength) / 2;
 float square = Mathf.Sqrt( halfPerimeter*(halfPerimeter-endDistance)*(halfPerimeter-startDistance)*(half
 float height = 2/lineLength * square;

 //dealing with out of line cases
 float distFromStartSq = startDistance*startDistance - height*height;
 float distFromEndSq = endDistance*endDistance - height*height;

 if (distFromStartSq > lineLengthSq && distFromStartSq > distFromEndSq) return endDistance;
 else if (distFromEndSq > lineLengthSq) return startDistance;
 else return height;
```

```
}



public void Relax (float blur, int iterations)
/// Moves nodes to make the spline smooth
/// Works with auto-tangents only
{
 for (int i=0; i<iterations; i++)
  Relax(blur);
}

public void Relax (float blur)
/// Moves nodes to make the spline smooth
/// Works with auto-tangents only
{
 for (int n=1; n<segments.Length; n++)
 {
  Vector3 midPos = (segments[n-1].start.pos + segments[n].end.pos)/2;
  segments[n].start.pos = midPos*blur/2f + segments[n].start.pos*(1-blur/2f);
  segments[n-1].end.pos = segments[n].start.pos;
 }
}



public void CutByRect (Vector3 pos, Vector3 size)
/// Splits all segments so that each intersection with AABB rect has a node
```

```
{
 for (int i=0; i<13; i++) //spline could be divided in 12 parts maximum
 {
  List<Segment> newSegments = new List<Segment>();

  for (int s=0; s<segments.Length; s++)
  {
   //early check - if inside/outside rect
   Vector3 min = segments[s].ApproxMin();
   Vector3 max = segments[s].ApproxMax();

   if (max.x < pos.x  ||  min.x > pos.x+size.x ||
    max.z < pos.z  ||  min.z > pos.z+size.z)
     { newSegments.Add(segments[s]); continue; } //fully outside
   if (min.x > pos.x  &&  max.x < pos.x+size.x &&
    min.z > pos.z  &&  max.z < pos.z+size.z)
     { newSegments.Add(segments[s]); continue; } //fully inside

   //splitting
   float sp = segments[s].IntersectRect(pos, size);
   if (sp < 0.0001f  ||  sp > 0.999f)
    { newSegments.Add(segments[s]); continue; }  //no intersection

   (Segment s1, Segment s2) = segments[s].GetSplitted(sp);
   newSegments.Add(s1);
   newSegments.Add(s2);
```

```
  }

  bool segemntsCountChanged = segments.Length != newSegments.Count;
  segments = newSegments.ToArray();
  if (!segemntsCountChanged) break; //if no nodes added - exiting 12 iterations
 }
}


public List<Line> OuterSegmentsRemoved (Vector3 pos, Vector3 size)
/// Removes segments with center placed out of pos-size AABB rect
/// Since some of the segments removed this should create several new lines
{
 List<Line> newLines = new List<Line>();
 List<Segment> newSegments = new List<Segment>();

 for (int s=0; s<segments.Length; s++)
 {
  Vector3 center = segments[s].GetPoint(0.5f);

  bool inRect = center.x > pos.x  &&  center.x < pos.x+size.x  &&
       center.z > pos.z  &&  center.z < pos.z+size.z;

  //if in rect - going on picking nodes
  if (inRect) newSegments.Add(segments[s]);
```

```
  //if first one not in rect - closing line
  else if (newSegments.Count != 0)
  {
   Line line = new Line {segments = newSegments.ToArray()};
   newLines.Add(line);
   newSegments.Clear();
  }
 }

 //closing line when it's ended within rect
 if (newSegments.Count != 0)
 {
  Line line = new Line {segments = newSegments.ToArray()};
  newLines.Add(line);
 }

 return newLines;
}



public List<Line> Clamped (Vector3 pos, Vector3 size)
/// Splits all segments so they are within AABB
/// This will create new lines
{
 Line cpy = new Line(this);
 cpy.CutByRect(pos, size);
```

```
 return cpy.OuterSegmentsRemoved(pos, size);
}



public void PushPoints (Vector3[] points, float[] ranges, bool horizontalOnly=true, float distFactor=1)
{
 for (int s=0; s<segments.Length; s++)
  segments[s].PushPoints(points, ranges, horizontalOnly, distFactor);
}



public void PushStartEnd (Vector3[] points, float[] ranges, bool horizontalOnly=true, float distFactor=1)
{
 for (int s=0; s<segments.Length; s++)
  segments[s].PushStartEnd(points, ranges, horizontalOnly, distFactor);
}



public void SplitNearPoints (Vector3[] points, float[] ranges, bool horizontalOnly=true, float startEndProxi
/// Splits segments within the range from each point
/// If skipCloseStartEnd enabled disregards points that are withinrange from start or end
{
 if (maxIterations == 0) return;

 List<Segment> newSegments = new List<Segment>();
 for (int s=0; s<segments.Length; s++)
```

```
 {
  bool isNear = segments[s].IsNearPoints(points, ranges, out float nearPercent, out int nearPointNum, ho
  if (!isNear) { newSegments.Add(segments[s]); continue; }

  (Segment seg1, Segment seg2) = segments[s].GetSplitted(nearPercent);
  newSegments.Add(seg1);
  newSegments.Add(seg2);
 }

 bool countChanged = segments.Length != newSegments.Count;
 segments = newSegments.ToArray();

 if (!countChanged  ||  maxIterations<=1) return;
 else SplitNearPoints(points, ranges, horizontalOnly, startEndProximityFactor, maxIterations-1);
}

#endregion

#region Weld

 public static bool AreCloseToWeld (Line line1, Line line2, float threshold)
 {
  Vector3[] line2Mins = new Vector3[line2.segments.Length];
  Vector3[] line2Maxs = new Vector3[line2.segments.Length];

  for (int s2=0; s2<line2.segments.Length; s2++)
```

```
 {
  line2Mins[s2] = line2.segments[s2].ApproxMin();
  line2Maxs[s2] = line2.segments[s2].ApproxMax();
 }

 for (int s1=0; s1<line1.segments.Length; s1++)
 {
  Vector3 min1 = line1.segments[s1].ApproxMin(); min1.x-=threshold; min1.y-=threshold; min1.z-=thresho
  Vector3 max1 = line1.segments[s1].ApproxMax(); min1.x+=threshold; min1.y+=threshold; min1.z+=thre

  for (int s2=0; s2<line2.segments.Length; s2++)
  {
   if (line2Mins[s2].x > max1.x || line2Maxs[s2].x < min1.x ||
    line2Mins[s2].y > max1.y || line2Maxs[s2].y < min1.y ||
    line2Mins[s2].z > max1.z || line2Maxs[s2].z < min1.z)
     continue;

   return true;
  }
 }

 return false;
}

public static Line[] WeldClose (Line line1, Line line2, float threshold)
```

```
{
 line1.CutSegmentsForWeld(line2, threshold);
 line2.CutSegmentsForWeld(line1, threshold);

 OverlapLines(line1, line2, threshold);
 //return new Line[] { line1, line2 };

 Segment[] allSegs = new Segment[line1.segments.Length+line2.segments.Length];
 Array.Copy(line1.segments, allSegs, line1.segments.Length);
 Array.Copy(line2.segments, 0, allSegs, line1.segments.Length, line2.segments.Length);
 return NeatWeldSegments(allSegs);
}


public void CutSegmentsForWeld (Line line2, float threshold)
/// Cuts line1 segment if line2 has a point within threshold
{
 for (int i=0; i<=line2.segments.Length+2; i++)
 //while true
 {
  List<Segment> newSegs = new List<Segment>();

  for (int s1=0; s1<segments.Length; s1++)
  {
   Vector3 min = segments[s1].ApproxMin() - new Vector3(threshold,threshold,threshold);
   Vector3 max = segments[s1].ApproxMax() + new Vector3(threshold,threshold,threshold);
```

```
float splitPercent = -1;

for (int p2=0; p2<line2.segments.Length+1; p2++)
{
 Vector3 point2 = p2!=line2.segments.Length ? line2.segments[p2].start.pos : line2.segments[p2-1].enc

 if (point2.x < min.x || point2.x > max.x ||
  point2.y < min.y || point2.y > max.y ||
  point2.z < min.z || point2.z > max.z)
   continue;

 if ((point2-segments[s1].start.pos).sqrMagnitude < threshold*threshold ||
  (point2-segments[s1].end.pos).sqrMagnitude < threshold*threshold )
   continue;
   //if close to segment start or end

 (float percent, float distSq) = segments[s1].GetClosest(point2, 5,5);

 if (distSq > threshold*threshold)
  continue;
 else
  splitPercent = percent;
}

if (splitPercent < 0) //no split for this segment
```

```
    newSegs.Add(segments[s1]);


   else //splitting
   {
    (Segment prevSeg, Segment nextSeg) = segments[s1].GetSplitted(splitPercent);
    newSegs.Add(prevSeg);
    newSegs.Add(nextSeg);
   }
  }

  if (newSegs.Count == segments.Length)
   return;
  segments = newSegs.ToArray();


  if (i==line2.segments.Length+2)
   throw new Exception("CutSegmentsForWeld reached maximum number of iterations");
 }
}

public static Line[] NeatWeldSegments (Segment[] segments)
/// Creates a spline system from splitted segments heap
{
 bool[] usedSegments = new bool[segments.Length];

 //removing degradated segments
 for (int s=0; s<segments.Length; s++)
```

```
 if ((segments[s].start.pos-segments[s].end.pos).sqrMagnitude < 0.0001f)
  usedSegments[s] = true;

//removing duplicating segments
for (int s1=0; s1<segments.Length; s1++)
{
 if (usedSegments[s1]) continue;

 for (int s2=0; s2<segments.Length; s2++)
 {
  if (usedSegments[s2]) continue;
  if (s2 == s1) continue;

  if ((segments[s1].start.pos-segments[s2].start.pos).sqrMagnitude < 0.0001f  &&
   (segments[s1].end.pos-segments[s2].end.pos).sqrMagnitude < 0.0001f)
    usedSegments[s2] = true;

  if ((segments[s1].start.pos-segments[s2].end.pos).sqrMagnitude < 0.0001f  &&
   (segments[s1].end.pos-segments[s2].start.pos).sqrMagnitude < 0.0001f)
    usedSegments[s2] = true;
 }
}

//finding weld indexes
Dictionary<int,int> weldLut = CompileWeldIndexLut(segments, usedSegments);
```

```
//compiling line segments
List< List<int> > allLinesSegs = new List<List<int>>(); //Each sub-list is a line, in that order. Storing segm
for (int s=0; s<segments.Length; s++)
{
 if (usedSegments[s]) continue;

 if (!weldLut.ContainsKey(s*2) || !weldLut.ContainsKey(s*2+1)) //finding any line start
  allLinesSegs.Add( CompileLineSegments(weldLut, usedSegments, s) );
}

//creating lines
Line[] lines = new Line[allLinesSegs.Count];
for (int l=0; l<lines.Length; l++)
{
 Segment[] lineSegs = new Segment[allLinesSegs[l].Count];
 for (int s=0; s<lineSegs.Length; s++)
  lineSegs[s] = segments[allLinesSegs[l][s]];

 lines[l] = new Line() { segments=lineSegs };
}

//inversing line segments if needed
foreach (Line line in lines)
 line.AutoInverseSegments();

return lines;
```

```
}

private static Dictionary<int,int> CompileWeldIndexLut (Segment[] segments, bool[] usedSegments)
/// Creates welding lut - for each of the segments start (index s*2) and end (s*2+1) finds other segments s
/// Will ignore segment tip if it has no connections OR has 2 or more connections
/// (to get segment num from index just use t/2)
{
 Dictionary<int,int> weldLut = new Dictionary<int,int>();

 for (int t1=0; t1<segments.Length * 2; t1++)
 {
  int s1 = t1/2;
  if (usedSegments[s1]) continue;

  Vector3 point1 = t1%2 == 0 ?
   segments[s1].start.pos :
   segments[s1].end.pos;

  int numSegmentsConnected = 0;
  int lastConnectedIndex = -1;

  for (int s2=0; s2<segments.Length; s2++)
  {
   if (usedSegments[s2]) continue;
   if (s2==s1) continue;
```

```
   if ((point1-segments[s2].start.pos).sqrMagnitude < 0.0001f)
    { lastConnectedIndex=s2*2; numSegmentsConnected++; }

   if ((point1-segments[s2].end.pos).sqrMagnitude < 0.0001f)
    { lastConnectedIndex=s2*2+1; numSegmentsConnected++; }
  }

  if (numSegmentsConnected == 1)
   weldLut.Add(t1, lastConnectedIndex);
 }

 return weldLut;
}

private static List<int> CompileLineSegments (Dictionary<int,int> weldLut, bool[] usedSegments, int start
{
 List<int> lineSegs = new List<int>();
 lineSegs.Add(startSegNum);
 usedSegments[startSegNum] = true;

 int startIndex = startSegNum*2;
 int endIndex = startSegNum*2+1;

 int nextIndex = -1;
 if (weldLut.ContainsKey(startIndex)) nextIndex = startIndex;
 if (weldLut.ContainsKey(endIndex)) nextIndex = endIndex;
```

```
if (nextIndex == -1) return lineSegs;

for (int i=0; i<=usedSegments.Length; i++) //while true
{
 int prevIndex = weldLut[nextIndex];

 int segNum = prevIndex/2;

 startIndex = segNum*2;
 endIndex = segNum*2+1;

 nextIndex = prevIndex==startIndex ? endIndex : startIndex;

 if (usedSegments[segNum])
  throw new Exception("CompileLineSegments trying to add used segment to line");

 lineSegs.Add(segNum);
 usedSegments[segNum] = true;

 if (!weldLut.ContainsKey(nextIndex)) break;

 if (i==usedSegments.Length)
  throw new Exception("CompileLineSegments reached maximum number of iterations");
}

return lineSegs;
```

```
}



public static void OverlapLines (Line line1, Line line2, float threshold)
/// Welds line points if they are within threshold distance
/// Doesn't combine points, just places them in a same position
{
 line1.OverlapTo(line2, threshold, moveFactor:0.5f);
 line2.OverlapTo(line1, threshold);
 line1.OverlapTo(line2, threshold);
}



public void OverlapTo (Line refLine, float threshold, float moveFactor=1)
/// Makes this line points take form of refLine points if they are within threshold
{
 for (int n=0; n<segments.Length+1; n++)
 {
  Vector3 point = n!=segments.Length ? segments[n].start.pos : segments[n-1].end.pos;

  int n2 = refLine.GetClosestNodeIndex(point, maxThreshold:threshold);
  if (n2 < 0) //no points within threshold
   continue;

  Vector3 refPoint = refLine.GetNodePos(n2);
  SetNodePos(n, refPoint*moveFactor + point*(1-moveFactor));
```

```
 }
}



public static bool AreCodirected (Line line1, Line line2)
/// Checks if two lines have the same orientation (should not be inverted to merge) by comparing the clos
/// More reliable for merge operations than comparing lines starts/ends
{
 (int n1, int n2) = GetClosestNodeIndexes(line1, line2);

 Vector3 prev1 = line1.GetNodePos(n1>0 ? n1-1 : n1);
 Vector3 next1 = line1.GetNodePos(n1<line1.NodesCount-1 ? n1+1 : line1.NodesCount-1);

 Vector3 prev2 = line2.GetNodePos(n2>0 ? n2-1 : n2);
 Vector3 next2 = line2.GetNodePos(n2<line2.NodesCount-1 ? n2+1 : line2.NodesCount-1);

 float prevDist = (prev1-prev2).sqrMagnitude;
 float nextDist = (next1-next2).sqrMagnitude;
 float aCrossDist = (next1-prev2).sqrMagnitude;
 float bCrossDist = (prev1-next2).sqrMagnitude;

 if (prevDist<aCrossDist  &&  prevDist<bCrossDist  &&  prevDist<nextDist) return true;
 if (nextDist<aCrossDist  &&  nextDist<bCrossDist  &&  nextDist<prevDist) return true;
 if (aCrossDist<prevDist  &&  aCrossDist<nextDist  &&  aCrossDist<bCrossDist) return false;
 if (bCrossDist<prevDist  &&  bCrossDist<nextDist  &&  bCrossDist<aCrossDist) return false;
```

```
 throw new Exception("Failed to determine codirection");
}



private static (int,int) GetClosestNodeIndexes (Line line1, Line line2,
 float maxThreshold=float.MaxValue, float minThreshold=-1,
 bool[] ignore1=null, bool[] ignore2=null)
/// Finds two closest nodes of two splines
/// Will ignore points closer than minThreshold and further than maxThreshold
/// Or points that used
{
 int closestIndex1 = -1;
 int closestIndex2 = -1;
 float minDist = float.MaxValue;

 for (int n1=0; n1<line1.segments.Length+1; n1++)
 {
  if (ignore1!=null && ignore1[n1]) continue;

  Vector3 point1 = n1!=line1.segments.Length ? line1.segments[n1].start.pos : line1.segments[n1-1].end.

  int n2 = line2.GetClosestNodeIndex(point1, minThreshold, maxThreshold, ignore2);
  if (n2 < 0) continue; //no points within threshold
  Vector3 point2 = line2.GetNodePos(n2);

  float dist = (point1-point2).sqrMagnitude;
```

```
  if (dist < minDist)
  {
   minDist = dist;
   closestIndex1 = n1;
   closestIndex2 = n2;
  }
 }

 return (closestIndex1, closestIndex2);
}



public int GetClosestNodeIndex (Vector3 point, float maxThreshold=float.MaxValue, float minThreshold=
/// Finds the closest segment start (or end for the last one) to the given point
/// Will ignore points closer than minThreshold and further than maxThreshold
{
 float minDistSq = float.MaxValue;
 int minIndex = -1;

 for (int s=0; s<segments.Length+1; s++)
 {
  if (ignore!=null && ignore[s]) continue;

  Vector3 npoint = s!=segments.Length ? segments[s].start.pos : segments[s-1].end.pos;

  float distSq =  (point.x-npoint.x)*(point.x-npoint.x) +
```

```
      (point.y-npoint.y)*(point.y-npoint.y) +
      (point.z-npoint.z)*(point.z-npoint.z);

  if (distSq < minDistSq  &&  distSq >= minThreshold*minThreshold  &&  distSq < maxThreshold*maxThr
   { minDistSq=distSq; minIndex=s; }
 }

 return minIndex;
}


public IEnumerable<int> NodeIndexesWithinRange (Vector3 point, float threshold)
{
 for (int s=0; s<segments.Length+1; s++)
 {
  Vector3 npoint = s!=segments.Length ? segments[s].start.pos : segments[s-1].end.pos;

  float distSq =  (point.x-npoint.x)*(point.x-npoint.x) +
      (point.y-npoint.y)*(point.y-npoint.y) +
      (point.z-npoint.z)*(point.z-npoint.z);

  if (distSq < threshold*threshold)
   yield return s;
 }
}
```

```
  private void AutoInverseSegments ()
  /// Automatically changes segment directions based on start/end positions
  {
   if (segments.Length == 1) return;

   if ((segments[0].start.pos - segments[1].start.pos).sqrMagnitude < 0.0001f  ||
    (segments[0].start.pos - segments[1].end.pos).sqrMagnitude < 0.0001f )
     segments[0] = segments[0].Inverted;

   for (int s=1; s<segments.Length; s++)
   {
    if ((segments[s].end.pos - segments[s-1].end.pos).sqrMagnitude < 0.0001f)
     segments[s] = segments[s].Inverted;
   }
  }

 #endregion


}
}
```

```
ï»¿using System;
using UnityEngine;

namespace Den.Tools.Splines
{



 [System.Serializable]
 public struct Node
 {
  public Vector3 pos;
  public Vector3 dir;

  public enum TangentType { auto, linear, correlated, broken }
  public TangentType type;



  #if UNITY_EDITOR
  public bool selected;
  public bool dispSelected; //to display node as selected when selecting by frame
  public bool freezed; //node could not be selected, moved and displayed with SceneGUI. Used to hide junc
  #endif



  public static (Vector3,Vector3) AutoTangents (Vector3 prevPos, Vector3 thisPos, Vector3 nextPos)
```

```
{
 Vector3 outDir = nextPos - thisPos;
 float outDirLength = outDir.magnitude;
 if (outDirLength > 0.00001f) //prevPos match with pos, usually on first segment
  outDir /= outDirLength;
 else
  outDir = new Vector3();

 Vector3 inDir = prevPos - thisPos;
 float inDirLength = inDir.magnitude;
 if (inDirLength > 0.00001f)
  inDir /= inDirLength;
 else
  inDir = new Vector3();

 Vector3 newInDir = (inDir - outDir).normalized;
 Vector3 newOutDir = -newInDir; //(outDir - inDir).normalized;

 inDir = newInDir.normalized * inDirLength * 0.35f;
 outDir = newOutDir.normalized * outDirLength * 0.35f;

 return (inDir,outDir);
}

public static (Vector3,Vector3) LinearTangents (Vector3 prevPos, Vector3 thisPos, Vector3 nextPos)
{
```

```
 return (
  (prevPos - thisPos) * 0.333f,
  (nextPos - thisPos) * 0.333f  );
}

public static Vector3 LinearTangent (Vector3 thisPos, Vector3 nextPos) //for first and last segments
 { return (nextPos - thisPos) * 0.333f; }



public static Vector3 CorrelatedOutTangent (Vector3 inDir, Vector3 outDir)
 { return -inDir.normalized * outDir.magnitude; }

public static Vector3 CorrelatedInTangent (Vector3 inDir, Vector3 outDir)
 { return -outDir.normalized * inDir.magnitude; }

/*public static (Vector3,Vector3) AlignTangents (Vector3 prevPos, Vector3 thisPos, Vector3 nextPos)
{
 switch (type)
 {
   case TangentType.auto:
   {
    outDir = nextPos - pos;
    float outDirLength = outDir.magnitude;
    if (outDirLength > 0.00001f) //prevPos match with pos, usually on first segment
     outDir /= outDirLength;
    else
```

```
  outDir = new Vector3();

 inDir = prevPos - pos;
 float inDirLength = inDir.magnitude;
 if (inDirLength > 0.00001f)
  inDir /= inDirLength;
 else
  inDir = new Vector3();

 Vector3 newInDir = inDir - outDir;
 Vector3 newOutDir = outDir - inDir;

 inDir = newInDir.normalized * inDirLength * 0.35f;
 outDir = newOutDir.normalized * outDirLength * 0.35f;

 break;
}

case TangentType.linear:
{
 outDir = (nextPos - pos) * 0.333f;
 inDir = (prevPos - pos) * 0.333f;
 break;
}

case TangentType.correlated:
```

```
    {
     float inDirLength = inDir.magnitude;
     float outDirLength = outDir.magnitude;

     if (inDirLength > 0) inDir /= inDirLength;
     if (outDirLength > 0) outDir /= outDirLength;

     outDir = outDir-inDir / 2;
     inDir = -outDir;

     outDir *= outDirLength; inDir *= inDirLength;

     break;
    }

    //do nothing for broken
   }
  }
  */
 }


}
```

```
ï»¿using System;
using System.Collections.Generic;
using UnityEngine;

namespace Den.Tools.Splines
{

 [System.Serializable]
 public struct Segment
 {
  public Node start;
  public Node end;

  public StructArray lengthToPercentLut; // lengthToPercentLut[normalizedLength] = percent
  public float length;

  public Segment (Node start, Node end)
   { this.start = start; this.end = end; lengthToPercentLut=new StructArray(); length=0; }

  public Segment (Vector3 start, Vector3 end)
   { this.start = new Node() {pos=start}; this.end = new Node() {pos=end}; lengthToPercentLut=new StructA

  public override string ToString() => $"Segment ({start.pos.x},{start.pos.y},{start.pos.z})-({end.pos.x},{end.p

  public float StartEndLength => (start.pos-end.pos).magnitude;
```

```
#region Segment Operations


 public Vector3 GetPoint (float p)

 /// Returns non-equalized beizer curve point

 {

  float ip = 1f-p;

  return

   ip*ip*ip*start.pos +

   3*p*ip*ip*(start.pos+start.dir) +

   3*p*p*ip*(end.pos+end.dir) +

   p*p*p*end.pos;

 }



 public Vector3 GetDerivative (float p)

 /// Finds a derivative at p, useful for quick tangent calculations

 {

  float it = 1-p;

  return

   it*it  * 3*(start.pos+start.dir - start.pos) +

   2 * it * p *3*(end.pos+end.dir - (start.pos+start.dir)) +

   p*p  *  3*(end.pos - (end.pos+end.dir));

 }



 public (Vector3 inDir, Vector3 outDir) GetSplitTangents (float p, Vector3 point)
```

```
/// Returns tangents as if segment was split at this percent
/// Differs from GetDerivative in tangent length and 2x slower performance. Use for split purpose only
/// For other tangent calculations use GetDerivative
{
 float ip = 1-p;
 Vector3 a = start.pos + start.dir*p;
 Vector3 b = (start.pos+start.dir)*ip + (end.pos+end.dir)*p;
 Vector3 c = end.pos + end.dir*ip;

 return (
  a*ip + b*p - point,
  c*p + b*ip - point );
}

public (Segment,Segment) GetSplitted (float p)
/// Same as Median, but adjusts start and end tangents too to preserve the spline look
{
 float ip = 1-p;
 Vector3 a = start.pos + start.dir*p;
 Vector3 b = (start.pos+start.dir)*ip + (end.pos+end.dir)*p;
 Vector3 c = end.pos + end.dir*ip;

 Vector3 point = GetPoint(p);

 Segment prev = new Segment() {
```

```
  start = new Node() { pos = start.pos, dir = start.dir*p,   type = start.type },
  end = new Node() {  pos = point,  dir = a*(1-p) + b*p-point, type = start.type } };
 Segment next = new Segment() {
  start = new Node() { pos = point,  dir = c*p + b*ip - point, type = start.type },
  end = new Node() {  pos = end.pos,  dir = end.dir*ip,   type = end.type } };

 return (prev,next);
}



public static Segment Join (Segment prev, Segment next)
/// Adjusts neighbor tangents to minimize the remove effect
{
 float beforeDist = (prev.start.pos - prev.end.pos).magnitude;
 float afterDist = (next.start.pos - next.end.pos).magnitude;
 float p = beforeDist / (beforeDist+afterDist);

 return new Segment {
  start = new Node() { pos = prev.start.pos, dir = prev.start.dir / p },
  end = new Node() {pos = next.end.pos, dir = prev.end.dir / (1-p) } };
}



public Vector3 GetPerpendicular (float p, Vector3 cross)
/// Finds a normal using reference vector
{
```

```
 Vector3 der = GetDerivative(p);
 return Vector3.Cross(der, cross);
}



public Vector2D GetPerpendicular2D (float p)
/// Finds a normal at XZ plane by taking derivative and swapping coordinates
/// Equivalent to GetPerpendicular(cross:Vector3.up)
{
 Vector3 der = GetDerivative(p);
 return new Vector2D(-der.z, der.x);
}



public Vector3 ApproxMin ()
/// Uses the node and tangent positions to evaluate AABB min
{
 Vector3 min = new Vector3(float.MaxValue, float.MaxValue, float.MaxValue);

 if (start.pos.x < min.x) min.x = start.pos.x;  if (start.pos.y < min.y) min.y = start.pos.y;  if (start.pos.z < min
 if (end.pos.x < min.x) min.x = end.pos.x;  if (end.pos.y < min.y) min.y = end.pos.y;  if (end.pos.z < min.z)
 if (start.pos.x+start.dir.x < min.x) min.x = start.pos.x+start.dir.x;  if (start.pos.y+start.dir.y < min.y) min.y =
 if (end.pos.x+end.dir.x < min.x) min.x = end.pos.x+end.dir.x;  if (end.pos.y+end.dir.y < min.y) min.y = en

 return min;
}
```

```
public Vector3 ApproxMax ()
/// Uses the node and tangent positions to evaluate AABB max
{
 Vector3 max = new Vector3(float.MinValue, float.MinValue, float.MinValue);

 if (start.pos.x > max.x) max.x = start.pos.x;  if (start.pos.y > max.y) max.y = start.pos.y;  if (start.pos.z > m
 if (end.pos.x > max.x) max.x = end.pos.x;  if (end.pos.y > max.y) max.y = end.pos.y;  if (end.pos.z > max
 if (start.pos.x+start.dir.x > max.x) max.x = start.pos.x+start.dir.x;  if (start.pos.y+start.dir.y > max.y) max.y
 if (end.pos.x+end.dir.x > max.x) max.x = end.pos.x+end.dir.x;  if (end.pos.y+end.dir.y > max.y) max.y = e

 return max;
}

public Vector3 Min ()
{
 throw new System.NotImplementedException();
}

public Vector3 Max ()
{
 throw new System.NotImplementedException();
}
```

```
public Segment Inverted => new Segment(end, start);



public float IntersectRect (Vector3 pos, Vector3 size)
/// Returns closest to start percent intersecting rect
{
 Vector3 min = ApproxMin();
 Vector3 max = ApproxMax();

 float minP = float.MaxValue;

 //left
 if (min.x < pos.x  &&  max.x > pos.x)
 {
  bool IntersectFn (Vector3 point) { return point.x > pos.x; }
  float p = GetIntersectPercent(IntersectFn);
  if (p<minP) minP = p;
 }

 //right
 if (min.x < pos.x+size.x  &&  max.x > pos.x+size.x)
 {
  bool IntersectFn (Vector3 point) { return point.x < pos.x+size.x; }
  float p = GetIntersectPercent(IntersectFn);
  if (p<minP) minP = p;
```

```
 }

 //top
 if (min.z < pos.z  &&  max.z > pos.z)
 {
  bool IntersectFn (Vector3 point) { return point.z > pos.z; }
  float p = GetIntersectPercent(IntersectFn);
  if (p<minP) minP = p;
 }

 //bottom
 if (min.z < pos.z+size.z  &&  max.z > pos.z+size.z)
 {
  bool IntersectFn (Vector3 point) { return point.z < pos.z+size.z; }
  float p = GetIntersectPercent(IntersectFn);
  if (p<minP) minP = p;
 }

 return minP;
}



public void ClampByRect (Vector3 pos, Vector3 size)
{

}
```

```
public void PushPoints (Vector3[] points, float[] ranges, bool horizontalOnly=true, float distFactor=1)
/// Moves points away from the segment so that they lay no closer than range
/// Changes the points array
/// DistFactor multiplies the range to move point (for iteration push it should be less than 1)
{
 Vector3 min = ApproxMin();
 Vector3 max = ApproxMax();

 Func<Vector3,Vector3,float> distFn;
 if (horizontalOnly) distFn = (p1,p2) => (p1.x-p2.x)*(p1.x-p2.x) + (p1.z-p2.z)*(p1.z-p2.z);
 else distFn = (p1,p2) => (p1.x-p2.x)*(p1.x-p2.x) + (p1.z-p2.z)*(p1.z-p2.z) + (p1.y-p2.y)*(p1.y-p2.y);

 for (int p=0; p<points.Length; p++)
 {
  Vector3 point = points[p];
  float range = ranges[p];

  if (min.x > point.x+range || max.x < point.x-range ||
```

```
   min.z > point.z+range || max.z < point.z-range ||
   !horizontalOnly && (min.y > point.y+range || max.y < point.y-range))
    continue;

  (float percent, float distSq) = GetClosest(distFn, point, 5, 5);

  if (distSq > range*range)
   continue;

  Vector3 linePoint = GetPoint(percent);
  points[p] = PushPoint(point, linePoint, range, horizontalOnly, distFactor);
 }
}


public bool PushStartEnd (Vector3[] points, float[] ranges, bool horizontalOnly=true, float distFactor=1)
/// Moves segment's start and end point positions away from points
/// Returns true if segments has been moved
{
 bool change = false;

 for (int p=0; p<points.Length; p++)
 {
  Vector3 point = points[p];
  float range = ranges[p];
```

```
 if (start.pos.x < point.x+range && start.pos.x > point.x-range &&
  start.pos.z < point.z+range && start.pos.z > point.z-range &&
  (horizontalOnly || (start.pos.y < point.y+range && start.pos.y > point.y-range)))
  {
   float dist = (point-start.pos).sqrMagnitude;
   if (dist<range*range)
   {
    start.pos = PushPoint(start.pos, point, range, horizontalOnly, distFactor);
    change = true;
   }
  }

 if (end.pos.x < end.pos.x+range && end.pos.x > point.x-range &&
  end.pos.z < end.pos.z+range && end.pos.z > point.z-range &&
  (horizontalOnly || (end.pos.y < point.y+range && end.pos.y > point.y-range)))
  {
   float dist = (point-end.pos).sqrMagnitude;
   if (dist<range*range)
   {
    end.pos = PushPoint(end.pos, point, range, horizontalOnly, distFactor);
    change = true;
   }
  }
}

return change;
```

```
}


 private Vector3 PushPoint (Vector3 point, Vector3 otherPoint, float otherRange, bool horizontalOnly=true
 {
  Vector3 awayDelta = otherPoint-point;
  if (horizontalOnly) awayDelta.y = 0;

  Vector3 awayDir = awayDelta.normalized; //awayDelta/lineDist
  float dist = awayDelta.magnitude;
  float distLeft = otherRange-dist;

  return point - awayDir*distLeft*distFactor;
 }



#endregion


#region Distance Operations

 public (float percent, float distSq) ApproximateClosest  (Vector3 point, int iterations) => ApproximateClos
 private (float percent, float distSq) ApproximateClosest (Func<Vector3,Vector3,float> distanceFn, Vector
 /// Finds approximate closest position by evaluating beizer points
 {
```

```
float minDist = float.MaxValue;
float minPercent = 0;
Vector3 minPoint = start.pos;

float step = 1f / (iterations-1);

for (int p=0; p<iterations; p++)
{
 float curPercent = p*step;

 Vector3 curPoint = GetPoint(curPercent);
 float curDist = distanceFn!=null  ?
  distanceFn(curPoint,point)  :
  (curPoint.x-point.x)*(curPoint.x-point.x) + (curPoint.z-point.z)*(curPoint.z-point.z) + (curPoint.y-point.y)*

 if (curDist < minDist)
 {
  minDist = curDist;
  minPercent = curPercent;
  minPoint.x = curPoint.x; minPoint.y = curPoint.y; minPoint.z = curPoint.z;
 }
}

if (minPercent > 1) minPercent = 1;
if (minPercent < 0) minPercent = 0;
```

```
 //return new ClosestData() { dist=minDist, point=minPoint, location=new Location(minPercent) };
 return (minPercent, minDist);
}



public (float percent, float distSq) RefineClosest (Vector3 point, float percent, float range, int iterations) =>
private (float percent, float distSq) RefineClosest (Func<Vector3,Vector3,float> distanceFn, Vector3 point
/// Adjusts closest position (percent), making it more exact. Returns new closer percent
/// Searches within full range (not half) to left and full range right
/// If distanceFn is not defined using point V3, disregarding point otherwise
{
 float minDist = float.MaxValue;

 for (int i=0; i<iterations; i++)
 {
  range /= 2;

  float biggerPercent = percent + range;  if (biggerPercent > 1) biggerPercent = 1;
  Vector3 biggerPoint = GetPoint(biggerPercent);
  float biggerDist = distanceFn!=null  ?
   distanceFn(biggerPoint,point)  :
   (biggerPoint.x-point.x)*(biggerPoint.x-point.x) + (biggerPoint.z-point.z)*(biggerPoint.z-point.z) + (bigger

  float lowerPercent = percent - range; if (lowerPercent < 0) lowerPercent = 0;
  Vector3 lowerPoint = GetPoint(lowerPercent);
  float lowerDist = distanceFn!=null  ?
```

```
   distanceFn(lowerPoint,point)  :
   (lowerPoint.x-point.x)*(lowerPoint.x-point.x) + (lowerPoint.z-point.z)*(lowerPoint.z-point.z) + (lowerPoin


  if (biggerDist < lowerDist) { percent = biggerPercent; minDist = biggerDist; }
  else { percent = lowerPercent; minDist = lowerDist; }
 }

 return (percent, minDist);
}

public (float percent, float distSq) GetClosest  (Vector3 point, int initialApprox=10, int recursiveApprox=10
{
 float percent = ApproximateClosest(point, iterations:initialApprox).percent;
 return RefineClosest(point, percent, range:1f/(initialApprox-1), iterations:recursiveApprox);
}

public (float percent, float distSq) GetClosest  (Func<Vector3,Vector3,float> distanceFn, Vector3 point, in
{
 float percent = ApproximateClosest(distanceFn, point, iterations:initialApprox).percent;
 return RefineClosest(distanceFn, point, percent, range:1f/(initialApprox-1), iterations:recursiveApprox);
}

public bool IsWithinRange (Vector3 point, float range)
/// Used to optimize line/sys distance evaluation by disregarding obviously far segments
/// Note that range is not squared
{
```

```
 Vector3 min = ApproxMin();
 if (point.x+range < min.x || point.y+range < min.y || point.z+range < min.z)
  return false;

 Vector3 max = ApproxMax();
 if (point.x-range > max.x || point.y-range > max.y || point.z-range > max.z)
  return false;

 return true;
}

public static (float,float) ClosestDistance (Segment s1, Segment s2, int initialApprox=6, int recursiveAppr
/// Returns points (percent) in segments that are closest to each other
{
 float initialStep = 1f / (initialApprox-1);

 float minDist = float.MaxValue;
 float minPercent1 = 0; float minPercent2 = 0;

 for (int p1=0; p1<initialApprox; p1++)
 {
  float percent1 = p1*initialStep;
  Vector3 point1 = s1.GetPoint(percent1);

  (float percent2, float dist) = s2.ApproximateClosest(point1, initialApprox);
```

```
 if (dist < minDist)
 {
  minDist = dist;
  minPercent1 = percent1;  minPercent2 = percent2;
 }
}

//adjusting
float recursiveStep = initialStep;
for (int i1=0; i1<recursiveApprox; i1++)
{
 recursiveStep /= 2;

 float biggerPercent1 = minPercent1 + recursiveStep; if (biggerPercent1 > 1) biggerPercent1 = 1;
 Vector3 biggerPoint1 = s1.GetPoint(biggerPercent1);
 (float biggerPercent2, float biggerDist) = s2.RefineClosest(biggerPoint1, minPercent2, initialStep, recurs

 float lowerPercent1 = minPercent1 - recursiveStep; if (lowerPercent1 < 0) lowerPercent1 = 0;
 Vector3 lowerPoint1 = s1.GetPoint(lowerPercent1);
 (float lowerPercent2, float lowerDist) = s2.RefineClosest(lowerPoint1, minPercent2, initialStep, recursive

 if (biggerDist < lowerDist) { minPercent1 = biggerPercent1; minPercent2 = biggerPercent2; }
 else { minPercent1 = lowerPercent1; minPercent2 = lowerPercent2; }
}

return (minPercent1, minPercent2);
```

```
}



private float GetIntersectPercent (Predicate<Vector3> intersectFn, int initialApprox=10, int recursiveAppr
/// Like GetClosestPoint, but evaluating if line intersects (bool) something (plane, axis, sphere, etc). A bit
/// IDEA: use Predicate<Vector3 oldPos, Vector3 newPos> to find intersection with planes
/// Works from start to end (returns intersection closest to start), but determines intersection both ways
{
 bool initIntersect = intersectFn(start.pos);
 float beforeIntersectPercent = 0;

 float step = 1f / (initialApprox-1);

 for (int p=0; p<initialApprox; p++)
 {
  float percent = p*step;

  Vector3 point = GetPoint(percent);
  bool newIntersect = intersectFn(point);

  if (newIntersect != initIntersect)
   break;

  beforeIntersectPercent = percent;
 }
```

```
 if (beforeIntersectPercent > 1) beforeIntersectPercent = 1;
 if (beforeIntersectPercent < 0) beforeIntersectPercent = 0;

 //adjusting initial position
 for (int i=0; i<recursiveApprox; i++)
 {
  step /= 2;

  float midPercent = beforeIntersectPercent + step; if (midPercent > 1) midPercent = 1;
  Vector3 midPoint = GetPoint(midPercent);
  bool midIntersect = intersectFn(midPoint);

  if (initIntersect == midIntersect) //no intersection until midPercent
   beforeIntersectPercent = midPercent;
 }

 return beforeIntersectPercent + step; //returning percent AFTER intersection in case we will intersect thr
}

public bool IsNearPoints (Vector3[] points, float[] ranges, bool horizontalOnly=true, float startEndProximit
 IsNearPoints(points, ranges, out float nearPercent, out int nearPointNum, horizontalOnly, startEndProxi

public bool IsNearPoints (Vector3[] points, float[] ranges, out float nearPercent, out int nearPointNum, bo
/// Checks if segment lays within range from any of the points, and returns percent and index if it does
/// If point is closer to start or end than range*startEndProximityFactor it will be disregarded
{
```

```
Vector3 min = ApproxMin();
Vector3 max = ApproxMax();

Func<Vector3,Vector3,float> distFn;
if (horizontalOnly) distFn = (p1,p2) => (p1.x-p2.x)*(p1.x-p2.x) + (p1.z-p2.z)*(p1.z-p2.z);
else distFn = (p1,p2) => (p1.x-p2.x)*(p1.x-p2.x) + (p1.z-p2.z)*(p1.z-p2.z) + (p1.y-p2.y)*(p1.y-p2.y);

for (int p=0; p<points.Length; p++)
{
 Vector3 point = points[p];
 float range = ranges[p];

 if (min.x > point.x+range || max.x < point.x-range ||
  min.z > point.z+range || max.z < point.z-range ||
  !horizontalOnly && (min.y > point.y+range || max.y < point.y-range))
   continue;

 if (startEndProximityFactor>0.00001f)
 {
  float maxDist = range*startEndProximityFactor * range*startEndProximityFactor;
  if ((start.pos-point).sqrMagnitude < maxDist ||
   (end.pos-point).sqrMagnitude < maxDist)
    continue;
 }

 (float percent, float distSq) = GetClosest(distFn, point, 5, 5);
```

```
  if (percent > 0.999f || percent < 0.001f)
   continue;

  if (distSq < range*range)
  {
   nearPercent = percent;
   nearPointNum = p;
   return true;
  }
 }

 nearPercent = -1;
 nearPointNum = -1;
 return false;
}

#endregion

#region Length Operations

 public float WorldLengthToPercent (float worldLength) { return NormLengthToPercent(worldLength/lengt

 public float NormLengthToPercent (float normLength)
 /// Converts relative length (range 0-1) to segemnt percent
```

```
{
 int prevMark = (int)(normLength*9);  //8 points, do not include 0 and 1 => 9 intervals
 float markPercent = (normLength - prevMark/9f)*9f;

 float prevPercent;
 float nextPercent;
 switch (prevMark)
 {
  case 0: prevPercent=0; nextPercent = lengthToPercentLut.b0/255f; break;
  case 1: prevPercent=lengthToPercentLut.b0/255f; nextPercent = lengthToPercentLut.b1/255f; break;
  case 2: prevPercent=lengthToPercentLut.b1/255f; nextPercent = lengthToPercentLut.b2/255f; break;
  case 3: prevPercent=lengthToPercentLut.b2/255f; nextPercent = lengthToPercentLut.b3/255f; break;
  case 4: prevPercent=lengthToPercentLut.b3/255f; nextPercent = lengthToPercentLut.b4/255f; break;
  case 5: prevPercent=lengthToPercentLut.b4/255f; nextPercent = lengthToPercentLut.b5/255f; break;
  case 6: prevPercent=lengthToPercentLut.b5/255f; nextPercent = lengthToPercentLut.b6/255f; break;
  case 7: prevPercent=lengthToPercentLut.b6/255f; nextPercent = lengthToPercentLut.b7/255f; break;
  case 8: prevPercent=lengthToPercentLut.b7/255f; nextPercent = 1; break;
  default: prevPercent=0; nextPercent=0; break;
 }

 return (prevPercent*(1-markPercent) + nextPercent*markPercent);
}


public float GetLinearDistance (float a, float b)
/// Calculates the line distance between two points
```

```
{
 float ia = 1f-a;
 float ib = 1f-b;

 Vector3 delta =
  (3*a*ia*ia - 3*b*ib*ib)*(start.dir) +
  (3*a*a*ia - 3*b*b*ib)*(end.pos+end.dir-start.pos) +
  (a*a*a - b*b*b)*(end.pos-start.pos);

 return delta.magnitude;
}



public float ApproxLength (float pStart=0, float pEnd=1, int iterations=32)
/// Splits the curve, calcs distance between points, measures the approximate length of the segment part
{
 float length = 0;
 float pDelta = pEnd - pStart;
 float pStep = pDelta / (iterations-1);

 float prevPercent = 0;
 for (int i=1; i<iterations; i++)
 {
  float percent = pStart + pStep*i;
  length += GetLinearDistance(prevPercent,percent); //(point - prevPoint).magnitude;
  prevPercent = percent;
```

```
 }

 return length;
}

public void UpdateLength (int iterations=32, float[] tmp=null)
{
 float linearLength = (start.pos-end.pos).magnitude;
 float tangentLength = start.dir.magnitude + ((start.dir+start.pos) - (end.dir+end.pos)).magnitude + end.dir

 if (linearLength > tangentLength-tangentLength/100 && linearLength < tangentLength+tangentLength/10
  //tangents lay approximately on the line
  FillLinearLengthLut();
 else
  FillApproxLengthLut(iterations, tmp);

 if (length < linearLength) //hack: FillLinearLengthLut returns 0 sometimes. Anyways beizer segment can'
  length = linearLength;
}

public void FillApproxLengthLut (int iterations=32, float[] tmp=null)
/// Calculates length and updates length lut
/// Fills the distance for each length (i.e. if lengthPercents is 4 then finds dist for 0.25,0.5,075,1)
{
```

```
float totalLength = 0;

//calculating lengths
float[] lengths = tmp==null ? new float[iterations] : tmp;
for (int p=0; p<lengths.Length; p++)
{
 float startPercent = 1f * p / iterations;
 float endPercent = 1f * (p+1) / iterations;

 float length = GetLinearDistance(startPercent, endPercent);
 lengths[p] = totalLength + length;
 totalLength += length;
}
length = totalLength;

//transforming all lengths to normalized
if (totalLength >= 0)
 for (int p=0; p<lengths.Length; p++)
  lengths[p] /= totalLength;

//filling length-to-percent lut
for (int i=0; i<8; i++)
{
 float normLength = (i+1) / 9f; //8 points, do not include 0 and 1, so 9 intevals

 int startNum = 0;
```

```
  for (int p=0; p<lengths.Length; p++)
   if (lengths[p]>normLength) { startNum = p-1; break; }
  int endNum = startNum+1;

  float stEndPercent = (normLength-lengths[startNum]) / (lengths[endNum]-lengths[startNum]);
  float segPercent = (startNum*(1-stEndPercent) + endNum*stEndPercent) / lengths.Length;
  lengthToPercentLut.SetFloat(i, segPercent);
 }
}

public void FillLinearLengthLut ()
{
 for (int i=0; i<8; i++)
 {
  float percent = (i+1) / 9f;
  lengthToPercentLut.SetFloat(i, percent);
 }
}

public float IntegralLength (float t=1, int iterations=8)
/// A variation of Legendre-Gauss solution from Primer (https://pomax.github.io/bezierinfo/#arclength)
{
 float z = t / 2;
 float sum = 0;
```

```
 double[] abscissaeLutArr = abscissaeLut[iterations];
 double[] weightsLutArr = weightsLut[iterations];

 float correctedT;
 for (int i=0; i<iterations; i++)
 {
  correctedT = z * (float)abscissaeLutArr[i] + z;

  Vector3 derivative = GetDerivative(correctedT);
  float b = Mathf.Sqrt(derivative.x*derivative.x + derivative.z*derivative.z);
  sum += (float)weightsLutArr[i] * b;
 }

 return z * sum;
}


public void FillIntegralLengthLut (float fullLength=-1, int iterations=32)
/// Fills the distance for each length (i.e. if lengthPercents is 4 then finds dist for 0.25,0.5,075,1)
{
 if (fullLength < 0) fullLength = IntegralLength();

 float pHalf = LengthToPercent(fullLength*0.5f, iterations:iterations);
 float p025 = LengthToPercent(fullLength*0.25f, pFrom:0, pTo:pHalf, iterations:iterations/2);
 float p075 = LengthToPercent(fullLength*0.75f, pFrom:pHalf, pTo:1, iterations:iterations/2);
```

```
 float p0125 = LengthToPercent(fullLength*0.125f, pFrom:0, pTo:p025, iterations:iterations/4);
 float p0375 = LengthToPercent(fullLength*0.375f, pFrom:p025, pTo:pHalf, iterations:iterations/4);
 float p0625 = LengthToPercent(fullLength*0.625f, pFrom:pHalf, pTo:p075, iterations:iterations/4);
 float p0875 = LengthToPercent(fullLength*0.875f, pFrom:p075, pTo:1, iterations:iterations/4);

 lengthToPercentLut.b0 = (byte)(p0125*255 + 0.5f);
 lengthToPercentLut.b1 = (byte)(p025*255 + 0.5f);
 lengthToPercentLut.b2 = (byte)(p0375*255 + 0.5f);
 lengthToPercentLut.b3 = (byte)(pHalf*255 + 0.5f);
 lengthToPercentLut.b4 = (byte)(p0625*255 + 0.5f);
 lengthToPercentLut.b5 = (byte)(p075*255 + 0.5f);
 lengthToPercentLut.b6 = (byte)(p0875*255 + 0.5f);
}


public float LengthToPercent (float length, float pFrom=0, float pTo=1, int iterations=8)
/// Converts world length to segments percent without using segment length LUT. Used to fill that LUT.
{
 float pMid = (pFrom+pTo)/2;
 float lengthMid = IntegralLength(pMid);

 if (length < lengthMid)
 {
  if (iterations <= 1) return (pFrom+pMid) /2;
  else return LengthToPercent(length, pFrom:pFrom, pTo:pMid, iterations:iterations-1);
 }
```

```
 else
 {
  if (iterations <= 1) return (pMid+pTo) /2;
  else return LengthToPercent(length, pFrom:pMid, pTo:pTo, iterations:iterations-1);
 }
}


// https://pomax.github.io/bezierinfo/legendre-gauss.html
// using doubles in case higher precision will be needed

static double[][] abscissaeLut = new double[][] {
 new double[] {},
 new double[] {},
 new double[] {-0.57735026918962576450914878050195745564760,0.57735026918962576450914878050
 new double[] {0,-0.77459666924148337703585307995647992216650,0.77459666924148337703585307
 new double[] {-0.33998104358485626480266575910324468720050,0.33998104358485626480266575
 new double[] {0,-0.53846931010568309103631442070020880496720,0.53846931010568309103631442
 new double[] {0.66120938646626451366139959501990534700640,-0.66120938646626451366139959
 new double[] {0, 0.40584515137739716690660641207696146334730,-0.40584515137739716690660641
 new double[] {-0.18343464249564980493947614236018398066670,0.18343464249564980493947614
 new double[] {0,-0.83603110732663579429942978806973487654410,0.83603110732663579429942978
 new double[] {-0.14887433898163121088482600112971998461750,0.14887433898163121088482600
 new double[] {0,-0.26954315595234497233153198540086152467960,0.26954315595234497233153198
 new double[] {-0.12523340851146891547244136946385312998330,0.12523340851146891547244136946
```

```
 new double[] {0,-0.23045831595513479406552812109798883521 15,0.230458315955134794065528121
 new double[] {-0.10805494870734366206624465021983474761 19,0.108054948707343662066244650
 new double[] {0,-0.20119409399743452230062830339459620781 28,0.201194093997434522300628303
 new double[] {-0.09501250983763744018531933542495806313 03,0.095012509837637440185319335
 new double[] {0,-0.17848418149584785585067749365406557547 54,0.178484181495847855850677493

static readonly double[][] weightsLut = new double[][] {
 new double[] {},
 new double[] {},
 new double[] {1.0,1.0},
 new double[] {0.88888888888888888888888888888888888888 88,0.555555555555555555555555555
 new double[] {0.65214515486254614262693605077800059276 46,0.652145154862546142626936050
 new double[] {0.56888888888888888888888888888888888888 88,0.478628670499366468041291514
 new double[] {0.36076157304813860756983351383771611166 15,0.360761573048138607569833513
 new double[] {0.41795918367346938775510204081632653061 22,0.381830050505118944950369775
 new double[] {0.36268378337836198296515044927719561219 41,0.362683783378361982965150449
 new double[] {0.33023935500125976316452506928697404887 88,0.180648160694857404058472031
 new double[] {0.29552422471475287017389299465133832942 10,0.295524224714752870173892994
 new double[] {0.27292508677790063071448352833634218915 60,0.262804544510246662180688869
 new double[] {0.24914704581340278500056243604295121083 04,0.249147045813402785000562436
 new double[] {0.23255155323087391019458951526883594815 66,0.226283180262897238412090186
 new double[] {0.21526385346315779019587644331626003527 49,0.215263853463157790195876443
 new double[] {0.20257824192556127288062019996751931483 86,0.198431485327111576456118326
 new double[] {0.18945061045506849628539672320828310514 69,0.189450610455068496285396723
 new double[] {0.17944647035620652545826564426188562144 87,0.176562705366992646325270990
```

```
    #endregion


    }


}
```

```
ï»¿using System;
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;

using Den.Tools;
using Den.Tools.Splines;
using Den.Tools.Matrices;
using Den.Tools.GUI;

namespace Den.Tools.Splines
{
public static class SplineMatrixOps
{
 public static void Stroke (SplineSys spline, MatrixWorld matrix,
  bool white=false, float intensity=1,
  bool antialiased=false, bool padOnePixel=false)
 /// Draws a line on matrix
 /// White will fill the line with 1, when disabled it will use spline height
 /// PaddedOnePixel works similarly to AA, but fills border pixels with full value (to create main tex for the m
 {
  foreach (Line line in spline.lines)
   for (int s=0; s<line.segments.Length; s++)
  {
   int numSteps = (int)(line.segments[s].length / matrix.PixelSize.x * 0.1f + 1);
```

```
  Vector3 startPos = line.segments[s].start.pos;
  Vector3 prevCoord = matrix.WorldToPixelInterpolated(startPos.x, startPos.z);
  float prevHeight = white ? intensity : (startPos.y / matrix.worldSize.y);

  for (int i=0; i<numSteps; i++)
  {
   float percent = numSteps!=1 ? 1f*i/(numSteps-1) : 1;
   Vector3 pos = line.segments[s].GetPoint(percent);
   float posHeight = white ? intensity : (pos.y / matrix.worldSize.y);
   pos = matrix.WorldToPixelInterpolated(pos.x, pos.z);

   matrix.Line(
    (Vector2D)prevCoord,
    (Vector2D)pos,
    prevHeight,
    posHeight,
    antialised:antialiased,
    paddedOnePixel:padOnePixel,
    endInclusive:i==numSteps-1);

   prevCoord = pos;
   prevHeight = posHeight;
  }
 }
}
```

```
public static void Silhouette (SplineSys spline, MatrixWorld matrix)
/// Fills all pixels within closed spline with 1, and all outer pixels with 0
/// Pixels directly in the spline are filled with 0.5
/// Internally strokes matrix first
{
 Stroke (spline, matrix, white:true, intensity:0.5f, antialiased:false, padOnePixel:false);
 Silhouette(spline, matrix, matrix);
}

public static void Silhouette (SplineSys spline, MatrixWorld strokeMatrix, MatrixWorld dstMatrix)
/// Fills all pixels within closed spline with 1, and all outer pixels with 0
/// Pixels directly in the spline are filled with 0.5
/// Requires the matrix with line stroked. StrokeMatrix and DstMatrix could be the same
{
 if (strokeMatrix != dstMatrix)
  dstMatrix.Fill(strokeMatrix);
  //and then using dst matrix only

 CoordRect rect = dstMatrix.rect;
 Coord min = rect.Min; Coord max = rect.Max;

 for (int x=min.x; x<max.x; x++)
  for (int z=min.z; z<max.z; z++)
  {
```

```
   int pos = (z-rect.offset.z)*rect.size.x + x - rect.offset.x;

   if (dstMatrix.arr[pos] < 0.01f) //free from stroke and fill
   {
    Vector3 pixelPos = dstMatrix.PixelToWorld(x, z);
    bool handness = spline.Handness(pixelPos) >= 0;

    dstMatrix.PaintBucket(new Coord(x,z), handness ? 0.75f : 0.25f);
   }
  }
}

public static void PaintBucket (this MatrixWorld matrix, Coord coord, float val, float threshold=0.0001f, int r
/// Like a paintBucket tool in photoshop
/// Fills all zero (lower than threshold) values with val, until meets borders
/// Doesnt guarantee filling (areas after the corner could be missed)
/// Use threshold to change between mask -1 or 0
{
 CoordRect rect = matrix.rect;
 Coord min = rect.Min; Coord max = rect.Max;

 MatrixOps.Stripe stripe = new MatrixOps.Stripe( Mathf.Max(rect.size.x, rect.size.z) );

 stripe.length = rect.size.x;

 matrix[coord] = -256; //starting mask
```

```
//first vertical spread is one row-only
MatrixOps.ReadRow(stripe, matrix, coord.x, matrix.rect.offset.z);
PaintBucketMaskStripe(stripe, threshold);
MatrixOps.WriteRow(stripe, matrix, coord.x, matrix.rect.offset.z);

for (int i=0; i<maxIterations; i++) //ten tries, but hope it will end before that
{
 bool change = false;

 //horizontally
 for (int z=min.z; z<max.z; z++)
 {
  MatrixOps.ReadLine(stripe, matrix, rect.offset.x, z);
  change = PaintBucketMaskStripe(stripe, threshold)  ||  change;
  MatrixOps.WriteLine(stripe, matrix, rect.offset.x, z);
 }

 //vertically
 for (int x=min.x; x<max.x; x++)
 {
  MatrixOps.ReadRow(stripe, matrix, x, matrix.rect.offset.z);
  change = PaintBucketMaskStripe(stripe, threshold)  ||  change;
  MatrixOps.WriteRow(stripe, matrix, x, matrix.rect.offset.z);
 }
```

```
  if (!change)
   break;

  //if (i==maxIterations-1 && !change)
  // Debug.Log("Reached max iterations");
 }

 //filling masked values with val
 for (int i=0; i<matrix.arr.Length; i++)
  if (matrix.arr[i] < -255) matrix.arr[i] = val;
}

private static bool PaintBucketMaskStripe (MatrixOps.Stripe stripe, float threshold=0.0001f)
/// Fills stripe until first unmasked value with -256
/// Returns true if anything masked
{
 bool changed = false;

 //to right
 bool masking = false;
 for (int i=0; i<stripe.length; i++)
 {
  if (stripe.arr[i] < -255)
  {
   masking = true;
   continue;
```

```
 }

 if (stripe.arr[i] < threshold)
 {
  if (masking)
  {
   stripe.arr[i] = -256;
   changed = true;
  }
 }
 else
  masking = false;
}

//to left
masking = false;
for (int i=stripe.length-1; i>=0; i--)
{
 if (stripe.arr[i] < -255)
 {
  masking = true;
  continue;
 }

 if (stripe.arr[i] < threshold)
 {
```

```
   if (masking)
   {
    stripe.arr[i] = -256;
    changed = true;
   }
  }
  else
   masking = false;
 }

 return changed;
}



public static void CombineSilhouetteSpread (Matrix silhouetteMatrix, Matrix spreadMatrix, Matrix dstMatrix
/// Blends silhouette and spread the way it's done in Silhouette node, so that silhouette have the spreaded
/// All matrices could be the same
{
 for (int i=0; i<dstMatrix.count; i++)
 {
  float spread = spreadMatrix.arr[i];
  float silhouette = silhouetteMatrix.arr[i];
  dstMatrix.arr[i] = silhouette > 0.5f ? spread : 1-spread;
 }
}
```

```
#region Isoline

 [StructLayout(LayoutKind.Sequential)]
 public struct MetaLine
 {
  public Coord c0;
  public Coord c1;
  public Coord c2;
  public Coord c3;
  public byte count;
  public bool closed;

  public const int length = 256;

  public Coord this[int n]
  {
   get
   {
    switch (n)
    {
     case 0: return c0;
     case 1: return c1;
     case 2: return c2;
     case 3: return c3;
     default: throw new Exception($"PixelLine array index ({n}) out of range");
    }
```

```
 }
 set
 {
  switch (n)
  {
   case 0: c0 = value; break;
   case 1: c1 = value; break;
   case 2: c2 = value; break;
   case 3: c3 = value; break;
   default: throw new Exception($"PixelLine array index ({n}) out of range");
  }
 }
}

public static void AddToList (List<Coord> list, MetaLine line)
{
 if (line.count==0) return;
 if (line.count>=1) list.Add(line.c0);
 if (line.count>=2) list.Add(line.c1);
 if (line.count>=3) list.Add(line.c2);
 if (line.count==4) list.Add(line.c3);
 if (line.closed) list.Add(line.c0);
}

public Coord Last
{get{
```

```
  switch (count)
  {
   case 1: return c0;
   case 2: return c1;
   case 3: return c2;
   case 4: return c3;
   default: throw new Exception($"PixelLine array index ({count-1}) out of range");
  }
 }}

 public MetaLine (Coord c0, Coord c1, Coord c2, Coord c3) { this.c0=c0; this.c1=c1; this.c2=c2; this.c3=c3
 public MetaLine (int c0x, int c0z, int c1x, int c1z, int c2x, int c2z, int c3x, int c3z) { c0.x=c0x; c0.z=c0z; c1.
 public MetaLine (int c0x, int c0z, int c1x, int c1z, int c2x, int c2z) { c0.x=c0x; c0.z=c0z; c1.x=c1x; c1.z=c1z
 public MetaLine (int c0x, int c0z, int c1x, int c1z) { c0.x=c0x; c0.z=c0z; c1.x=c1x; c1.z=c1z; c2.x=0; c2.z=
}


public static List<MetaLine> MatrixToMetaLines (Matrix matrix, float threshold)
/// Create isolines in form of short unwelded meta-lines around each pixel
{
 Coord size = matrix.rect.size;
 List<MetaLine> lines = new List<MetaLine>();

 for (int x=0; x<size.x; x++)
  for (int z=0; z<size.z; z++)
  {
```

```
int pos = z*size.x + x;
if (matrix.arr[pos] < threshold) continue;

int c = 0;
if (z==size.z-1  ||  matrix.arr[pos+size.z]>=threshold)  c = c | 0b_0000_1000;  //z==size.z-1: out of borde
if (z==0  ||  matrix.arr[pos-size.z]>=threshold)  c = c | 0b_0000_0100;
if (x==0  ||  matrix.arr[pos-1]>=threshold)  c = c | 0b_0000_0010;
if (x==size.x-1  ||  matrix.arr[pos+1]>=threshold)  c = c | 0b_0000_0001;

switch (c)
{
 case 0b_0000_0000:  //if (!top && !bottom && !left && !right)
  lines.Add( new MetaLine(x,z,  x,z+1,  x+1,z+1,  x+1,z) { closed=true } );
  break;

 case 0b_0000_0001:  //else if (!top && !bottom && !left &&  right)
  lines.Add( new MetaLine(x+1,z,  x,z,  x,z+1,  x+1,z+1) );
  break;

 case 0b_0000_0100:  //else if (!top &&  bottom && !left && !right)
  lines.Add( new MetaLine(x,z,  x,z+1,  x+1,z+1,  x+1,z) );
  break;

 case  0b_0000_0010:  //else if (!top && !bottom &&  left && !right)
  lines.Add( new MetaLine(x,z+1,  x+1,z+1,  x+1,z,  x,z) );
  break;
```

```
case  0b_0000_1000:  //else if ( top && !bottom && !left && !right)
 lines.Add( new MetaLine(x+1,z+1,  x+1,z,  x,z,  x,z+1) );
 break;

case  0b_0000_0011:  //else if (!top && !bottom &&  left &&  right)
 lines.Add( new MetaLine(x,z+1,  x+1,z+1) );
 lines.Add( new MetaLine(x+1,z,  x,z) );
 break;

case 0b_0000_1100:  //else if ( top &&  bottom && !left && !right)
 lines.Add( new MetaLine(x,z,  x,z+1) );
 lines.Add( new MetaLine(x+1,z+1,  x+1,z) );
 break;

case 0b_0000_0101:  //else if (!top &&  bottom && !left &&  right)
 lines.Add( new MetaLine(x,z,  x,z+1,  x+1,z+1) );
 break;

case 0b_0000_0110:  //else if (!top &&  bottom &&  left && !right)
 lines.Add( new MetaLine(x,z+1,  x+1,z+1,  x+1,z) );
 break;

case 0b_0000_1010:  //else if ( top && !bottom &&  left && !right)
 lines.Add( new MetaLine(x+1,z+1,  x+1,z,  x,z) );
 break;
```

```
case 0b_0000_1001:  //else if ( top && !bottom && !left &&  right)
 lines.Add( new MetaLine(x+1,z,  x,z,  x,z+1) );
 break;

case 0b_0000_0111:  //else if (!top &&  bottom &&  left &&  right)
 lines.Add( new MetaLine(x,z+1,  x+1,z+1) );
 break;

case 0b_0000_1110:  //else if ( top &&  bottom &&  left && !right)
 lines.Add( new MetaLine(x+1,z+1,  x+1,z) );
 break;

case 0b_0000_1011:  //else if ( top && !bottom &&  left &&  right)
 lines.Add( new MetaLine(x+1,z,  x,z) );
 break;

case 0b_0000_1101:  //else if ( top &&  bottom && !left &&  right)
 lines.Add( new MetaLine(x,z,  x,z+1) );
 break;

case 0b_0000_1111:  //else if (top && bottom && left && right)
 break;

default:
 throw new Exception("Impossible pixels combination found");
```

```
  }
 }

 return lines;
}

public static List< List<Coord> > WeldMetaLines (List<MetaLine> metaLines)
{
 //metaLines start coord lut
 Dictionary<Coord,int> startLut = new Dictionary<Coord,int>(capacity:metaLines.Count);
 int metaLinesCount = metaLines.Count;
 for (int i=0; i<metaLinesCount; i++)
  startLut.Add(metaLines[i].c0, i);

 //meta lines with the start coordinate that is not covered with other metaline end coordinate
 //they will begin a new line
 Stack<Coord> initialCoords = new Stack<Coord>();

 HashSet<Coord> endCoords = new HashSet<Coord>();
 for (int i=0; i<metaLinesCount; i++)
  endCoords.Add(metaLines[i].Last);

 for (int i=0; i<metaLinesCount; i++)
  if (!endCoords.Contains(metaLines[i].c0)) initialCoords.Push(metaLines[i].c0);

 //welded lines
```

```
List< List<Coord> > weldedLines = new List< List<Coord> >();
List<Coord> currentLine = new List<Coord>();

//welding
while (startLut.Count != 0)
{
 //finding first point to weld
 Coord start;
 if (initialCoords.Count != 0)
  start = initialCoords.Pop();
 else
  start = startLut.AnyKey(); //if no initial coords left - then only looped lines remain - and then starting anyw

 for (int i=0; i<metaLinesCount+2; i++) //should not reach maximum
 {
  if (startLut.TryGetValue(start, out int num)) //if has continuation
  {
   MetaLine.AddToList(currentLine, metaLines[num]);
   startLut.Remove(start);
   start = metaLines[num].Last;
   continue;
  }

  else //finishing line
  {
   weldedLines.Add(currentLine);
```

```
      currentLine = new List<Coord>();
      break;
     }

    throw new Exception("Welding reached maximum");
   }
  }

  return weldedLines;
 }

 /*public static List<Vector2D> RelaxLine (List<Vector2D> line, float relax)
 {
  int lineCount = line.Count;

  Vector2D next =

  for (int n=0; n<lineCount; n++)
  {

  }
 }*/

 #endregion
}
```

```
}//namespace
```

```
ï»¿using UnityEngine;


namespace Den.Tools.Splines
{
 [System.Serializable]
 public class SplineObject : MonoBehaviour
 {
  public SplineSys splineSys = new SplineSys();
 }
}
```

```
ï»¿using UnityEngine;

using System;
using System.Collections.Generic;

namespace Den.Tools.Splines
{

 [System.Serializable]
 public class SplineSys : ICloneable
 {
  public Line[] lines = new Line[0];
  //public Junction[] junctions;

  public bool guiDrawNodes = true;
  public bool guiDrawSegments = true;
  public bool guiDrawDots;
  public int guiDotsCount = 10;
  public bool guiDotsEquidist;


  public SplineSys () { }
  public SplineSys (SplineSys src)
  {
   CopyLinesFrom(src.lines);
```

```
 guiDrawNodes = src.guiDrawNodes;
 guiDrawSegments = src.guiDrawSegments;
 guiDrawDots = src.guiDrawDots;
 guiDotsCount = src.guiDotsCount;
 guiDotsEquidist = src.guiDotsEquidist;
}



public object Clone () { return new SplineSys(this); }

public Vector3 GetPoint ( (int l, int s, float p) loc ) => lines[loc.l].segments[loc.s].GetPoint(loc.p);
public Vector3 GetPoint (int l, int s, float p) => lines[l].segments[s].GetPoint(p);

public Vector3 GetDerivative (int l, int s, float p) => lines[l].segments[s].GetDerivative(p);
public Vector3 GetDerivative ( (int l, int s, float p) loc ) => lines[loc.l].segments[loc.s].GetDerivative(loc.p);

public void AddLine (Line line)
 { ArrayTools.Add(ref lines, line); }

public Line AddLine (Vector3 start, Vector3 end)
{
 Line line = new Line(start, end);
 ArrayTools.Add(ref lines, line);
 return line;
}
```

```
public void AddLines (Line[] otherLines)
 { ArrayTools.AddRange(ref lines, otherLines); }

public void CopyLinesFrom (Line[] otherLines)
{
 lines = new Line[otherLines.Length];
 for (int l=0; l<lines.Length; l++)
  lines[l] = new Line(otherLines[l]);
}

public void RemoveLine (int l)
 { ArrayTools.RemoveAt(ref lines, l); }

public static SplineSys CreateDefault ()
{
 SplineSys spline = new SplineSys();
 spline.lines = new Line[] { new Line() };
 spline.lines[0].segments = new Segment[] { new Segment(new Vector3(0,0,0), new Vector3(1,0,0)) };
 return spline;
}


public void Update ()
/// Called by GUI on every line change and by system on global change
{ for (int l=0; l<lines.Length; l++) lines[l].Update(); }
```

```
public void UpdateTangents ()
 { for (int l=0; l<lines.Length; l++) lines[l].UpdateTangents(); }

public void UpdateLength ()
 { for (int l=0; l<lines.Length; l++) lines[l].UpdateLength(); }


public Vector3[][] GetAllPoints (float resPerUnit=0.1f, int minRes=3, int maxRes=20)
/// evaluates each of the splines, returning the positions
/// if lengthFactor==1 resolution is multiplied with segment length
{
 Vector3[][] points = new Vector3[lines.Length][];

 for (int l=0; l<lines.Length; l++)
  points[l] = lines[l].GetAllPoints(resPerUnit, minRes, maxRes);

 return points;
}


public int NodesCount
{get{
 int nodesCount = 0;
 for (int l=0; l<lines.Length; l++)
  nodesCount += lines[l].NodesCount;
```

```
 return nodesCount;
}}


public (int l, int s, float p, float dist) GetClosest (Vector3 point, int initialApprox=10, int recursiveApprox=10
 GetClosest(null, point, initialApprox, recursiveApprox);

public (int l, int s, float p, float dist) GetClosest (Func<Vector3,Vector3,float> distanceFn, int initialApprox=
 GetClosest(distanceFn, new Vector3(), initialApprox, recursiveApprox);

public (int l, int s, float p, float dist) GetClosest (Func<Vector3,Vector3,float> distanceFn, Vector3 point, in
/// Returns the closest location and distance from point to spline sys
/// If distFn defined using it instead of point
{
 float minDist = float.MaxValue;
 int ml=0;  int ms=0;  float mp=0;

 for (int l=0; l<lines.Length; l++)
 {
  (int cl, int cs, float cp, float curDist) = lines[l].GetClosest(distanceFn, point, initialApprox, recursiveApprox
  if (curDist < minDist)
  {
   minDist = curDist;
   ms=cs; mp=cp; ml=cl;
  }
 }
```

```
 return (ml, ms, mp, minDist);
}

public float Handness (Vector3 point)
/// Determines wheter the point is on the left or on the right of a spline (from top view)
/// returns either positiove (left) or negative (right) value;
{
 (int l, int s, float p, float dist) = GetClosest(point);
 Vector3 closest = GetPoint(l,s,p);
 Vector3 tangent = GetDerivative(l,s,p);

 Vector2 lineStart = new Vector2(closest.x, closest.z);
 Vector2 lineEnd = new Vector2(closest.x+tangent.x, closest.z+tangent.z);
 return (new Vector2(point.x,point.z)).Handness(lineStart, lineEnd);
}


public void Optimize (float deviation)
/// Removes those nodes that should not change the shape a lot
/// Works with auto-tangents only
{ for (int l=0; l<lines.Length; l++) lines[l].Optimize(deviation); }


public void Subdivide (int num, bool equiDistance=false)
/// Splits each segment in several shorter segments
```

```
{ for (int l=0; l<lines.Length; l++) lines[l].Subdivide(num); }

public void Relax (float blur, int iterations)
/// Moves nodes to make the spline smooth
/// Works with auto-tangents only
{ for (int l=0; l<lines.Length; l++) lines[l].Relax(blur, iterations); }

public void CutByRect (Vector3 pos, Vector3 size)
/// Splits all segments so that each intersection with AABB rect has a node
{ for (int l=0; l<lines.Length; l++) lines[l].CutByRect(pos, size); }

public void RemoveOuterSegments (Vector3 pos, Vector3 size)
/// Removes segments with center placed out of pos-size AABB rect
{
 List<Line> newLines = new List<Line>();
 for (int l=0; l<lines.Length; l++)
  newLines.AddRange (lines[l].OuterSegmentsRemoved(pos, size));
 lines = newLines.ToArray();
}

public void Clamp (Vector3 pos, Vector3 size)
/// Splits all segments so they are within AABB
{
 CutByRect(pos, size);
 RemoveOuterSegments(pos, size);
}
```

```
public void PushPoints (Vector3[] points, float[] ranges, bool horizontalOnly=true, float distFactor=1)
{
 for (int l=0; l<lines.Length; l++)
  lines[l].PushPoints(points, ranges, horizontalOnly, distFactor);
}


public void PushStartEnd (Vector3[] points, float[] ranges, bool horizontalOnly=true, float distFactor=1)
{
 for (int l=0; l<lines.Length; l++)
  lines[l].PushStartEnd(points, ranges, horizontalOnly, distFactor);
}

public void SplitNearPoints (Vector3[] points, float[] ranges, bool horizontalOnly=true, float startEndProxim
{
 for (int l=0; l<lines.Length; l++)
  lines[l].SplitNearPoints(points, ranges, horizontalOnly, startEndProximityFactor, maxIterations);
}

public void WeldCloseLines (float threshold)
{
 List<Line> linesList = new List<Line>();
 linesList.AddRange(lines);
```

```
int i = 0;
Repeat:
i++;
if (i == lines.Length+1)
 throw new Exception("WeldCloseLines reached maximum iterations");

for (int l1=0; l1<linesList.Count; l1++)
 for (int l2=0; l2<l1; l2++)
 {
  Line line1 = linesList[l1];
  Line line2 = linesList[l2];

  if (!Line.AreCloseToWeld(line1,line2,threshold))
   continue;

  Line[] weldedLines = Line.WeldClose(new Line(line1), new Line(line2), threshold);

  if (weldedLines.Length==1) //ignoring welding two lines in one
   continue;

  if (weldedLines.Length==2  &&  weldedLines[0].segments.Length + weldedLines[1].segments.Length ==
  //no change (two lines and summary nodes count has not been changed, although they might be swapp
  //TODO: compare point by point?
  {
   linesList[l1] = weldedLines[0];
   linesList[l2] = weldedLines[1]; //assigning lines just in case they have minor change
```

```
        continue;
      }

      linesList.RemoveAt(l1); //l2 is always less l1,  l1 is always bigger l2
      linesList.RemoveAt(l2);
      linesList.AddRange(weldedLines);

      goto Repeat; //exiting nested loop and restarting
     }

    lines = linesList.ToArray();
   }
  }
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;

using Den.Tools;
using Den.Tools.GUI;

namespace Den.Tools.AutoSplines
{
 //[CustomEditor(typeof(SplineObject))]
/* public static class SplineInspector// : Editor
 {
  public static bool guiLines = true;
  public static bool guiKnobs = true;
  public static bool guiDisplay = true;


  public static void DrawSpline (SplineSys sys)
  {
   using (Cell.LineStd)
    using (new Draw.FoldoutGroup(ref guiDisplay, "Display", isLeft:true))
     if (guiDisplay)
     {
      using (Cell.LineStd) Draw.ToggleLeft(ref sys.guiDrawNodes, "Draw Nodes");
```

```
   using (Cell.LineStd) Draw.ToggleLeft(ref sys.guiDrawSegments, "Draw Segments");
   using (Cell.LineStd) Draw.ToggleLeft(ref sys.guiDrawDots, "Draw Dots");
   using (Cell.LineStd) Draw.Field(ref sys.guiDotsCount, "Dots Count");
   using (Cell.LineStd) Draw.Toggle(ref sys.guiDotsEquidist, "Dots Equidist");

   if (Cell.current.valChanged)
    SceneView.lastActiveSceneView?.Repaint();
  }

Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref guiLines, "Lines", isLeft:true))
  if (guiLines)
  {
   using (Cell.LineStd) Draw.DualLabel("Lines Count", sys.lines.Length.ToString());

   if (SplineEditor.selectedKnobs.Count == 1)
   {
    (int l, int n) = SplineEditor.selectedKnobs.Any();
    Line line = sys.lines[l];

    using (Cell.LineStd) Draw.DualLabel("Length", line.length.ToString());
    using (Cell.LineStd) Draw.DualLabel("Segments", line.segments.Length.ToString());
    using (Cell.LineStd) Draw.Toggle(ref line.looped, "Looped");

    using (Cell.LineStd)
```

```
    if (Draw.Button("Remove"))
    {
     sys.RemoveLine(l);
     SplineEditor.selectedKnobs.Clear();
     SplineEditor.dispSelectedKnobs.Clear();
    }
  }

  using (Cell.LineStd)
   if (Draw.Button("Add"))
    sys.AddLine(
     SceneView.lastActiveSceneView.pivot - new Vector3(SceneView.lastActiveSceneView.cameraDistan
     SceneView.lastActiveSceneView.pivot + new Vector3(SceneView.lastActiveSceneView.cameraDista
 }

Cell.EmptyLinePx(4);
using (Cell.LineStd)
 using (new Draw.FoldoutGroup(ref guiKnobs, "Knobs", isLeft:true))
  if (guiKnobs)
  {
   using (Cell.LineStd) Draw.DualLabel("Selected", SplineEditor.selectedKnobs.Count.ToString());

   if (SplineEditor.selectedKnobs.Count == 1)
   {
    Cell.EmptyLinePx(4);
    (int l, int n) = SplineEditor.selectedKnobs.Any();
```

```
    using (Cell.LinePx(0)) DrawKnob(sys, l, n);
   }
  }

 Cell.EmptyLinePx(4);
 using (Cell.LineStd)
  if (Draw.Button("Update"))
  {
   sys.Update();
   SceneView.lastActiveSceneView?.Repaint();
  }
}

public static void DrawKnob (SplineSys sys, int l, int n)
{
 using (Cell.LineStd) Draw.DualLabel("Number", "Line:" + l.ToString() + ", Node:" + n.ToString());
 using (Cell.LineStd) Draw.Toggle(n==0, "Is First");
 using (Cell.LineStd) Draw.Toggle(n==sys.lines[l].segments.Length, "Is Last");

 SplineEditor.Knob knob = sys.lines[l].GetKnob(n);

 using (Cell.LinePx(0))
 {
  using (Cell.LineStd) Draw.Field(ref knob.pos, "Position");
  //using (Cell.LineStd) Draw.DualLabel("In Dir", knob.inDir.ToString());
  //using (Cell.LineStd) Draw.DualLabel("Out Dir", knob.outDir.ToString());
```

```
    using (Cell.LineStd) Draw.Field(ref knob.inDir, "In Dir");
    using (Cell.LineStd) Draw.Field(ref knob.outDir, "Out Dir");
    using (Cell.LineStd) Draw.Field(ref knob.type, "Type");

    if (Cell.current.valChanged)
    {
     sys.lines[l].SetKnob(knob, n);
     sys.lines[l].Update();
     SceneView.lastActiveSceneView?.Repaint();
    }
   }
  }
 }*/
}
```

```
ï»¿using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;

using Den.Tools.GUI;

namespace Den.Tools.Splines
{
 [CustomEditor(typeof(SplineObject))]
 public partial class SplineObjectInspector : Editor
 {
  [DrawGizmo(GizmoType.NonSelected | GizmoType.Selected)]
  static void DrawInactiveGizmo (SplineObject obj, GizmoType gizmoType)
  {
   //if (gizmoType.HasFlag(GizmoType.Selected)) Handles.color = new Color(1, 0.5f, 0, 1);
   //else if (gizmoType.HasFlag(GizmoType.NonSelected)) Handles.color = new Color(0.75f, 0.25f, 0, 1);
   //SplineEditor.DrawSplineSys(obj.splineSys, obj.transform.localToWorldMatrix);
   //drawing selected in OnSceneGUI - otherwise will be erased by matrix gizmo drawn in tester's OnScene

   if (gizmoType.HasFlag(GizmoType.NonSelected))
   {
    Handles.color = new Color(0.75f, 0.25f, 0, 1);
    SplineEditor.DrawSplineSys(obj.splineSys, obj.transform.localToWorldMatrix);
   }
```

```
  }

  private void OnSceneGUI ()
  {
   SplineObject splineObj = (SplineObject)target;

   Handles.color = new Color(1, 0.5f, 0, 1);
   SplineEditor.DrawSplineSys(splineObj.splineSys, splineObj.transform.localToWorldMatrix);

   SplineEditor.EditSplineSys(splineObj.splineSys, splineObj.transform.localToWorldMatrix, splineObj);
  }


  UI ui = new UI();
  public override void OnInspectorGUI ()
   { ui.Draw(DrawGUI, inInspector:true); }

  private void DrawGUI ()
  {
   SplineObject splineObj = (SplineObject)target;
   //SplineInspector.DrawSpline(splineObj.splineSys);
  }
 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
using Den.Tools.SceneEdit;

namespace Den.Tools.Splines
{
 public static class SplineEditor
 {
  const float knobSize = 15f;
  const float junctionSize = 5f;

  public static HashSet<(int,int)> selectedKnobs = new HashSet<(int,int)>();
  public static HashSet<(int,int)> dispSelectedKnobs = new HashSet<(int,int)>(); //knobs that are virtually se
  public static int[] selectionNumsBounds = new int[0]; //to clear selection if line number or num nodes in ea

  /*public struct Nullable<T>
  {
   public T obj;
   public bool defined;

   public Nullable(T obj) { this.obj=obj; defined=true; }

   public static explicit operator T(Nullable<T> n) => n.obj;
   public static explicit operator Nullable<T>(T o) => new Nullable<T>(o);
```

```
}*/

#region Knob

 // segments (3):     0  1  2
 // nodes (4):  0 ----- 1 ----- 2 ----- 3


 public struct KnobLocation
 {
  public int l; //line number
  public int n; //node number
  public KnobLocation (int l, int n) { this.l=l; this.n=n; }
 }

 public struct Knob
 /// Combined data from two adjacent nodes
 {
  public Vector3 pos;
  public Vector3 inDir;
  public Vector3 outDir;

  public bool inDefined;
  public bool outDefined;

  public Node.TangentType type;
```

```
}

public static Knob GetKnob (this Line line, int n)
{
 Knob knob = new Knob();

 //common case
 if (n > 0  &&  n < line.segments.Length)
 {
  knob.pos = line.segments[n].start.pos;
  knob.type = line.segments[n].start.type;
  knob.outDir = line.segments[n].start.dir;
  knob.inDir = line.segments[n-1].end.dir;
  knob.outDefined = knob.inDefined = true;
 }

 //first
 else if (n == 0  &&  n < line.segments.Length)
 {
  knob.pos = line.segments[n].start.pos;
  knob.type = line.segments[n].start.type;
  knob.outDir = line.segments[n].start.dir;
  knob.outDefined = true;
 }

 //last
```

```
 else if (n > 0  &&  n == line.segments.Length)
 {
  knob.pos = line.segments[n-1].end.pos;
  knob.type = line.segments[n-1].end.type;
  knob.inDir = line.segments[n-1].end.dir;
  knob.inDefined = true;
 }

 //only one
 else
 {
  knob.pos = line.segments[n].start.pos;
  knob.type = line.segments[n].start.type;
 }

 return knob;
}

public static void SetKnob (this Line line, Knob knob, int n)
{
 if (n > 0)
 {
  line.segments[n-1].end.pos = knob.pos;
  if (knob.inDefined) line.segments[n-1].end.dir = knob.inDir;
  line.segments[n-1].end.type = knob.type;
 }
```

```
 if (n < line.segments.Length)
 {
  line.segments[n].start.pos = knob.pos;
  if (knob.outDefined) line.segments[n].start.dir = knob.outDir;
  line.segments[n].start.type = knob.type;
 }
}


#endregion

#region Draw

 public static void DrawSplineSys (SplineSys sys, Matrix4x4 trs)
 /// Drawing inactive (non-editable) system
 {
  for (int l=0; l<sys.lines.Length; l++)
  {
   Line line = sys.lines[l];

   //segments
   for (int s=0; s<line.segments.Length; s++)
   {
    if (sys.guiDrawSegments) DrawSegment(line.segments[s], trs);
    if (sys.guiDrawDots) DrawDots(line.segments[s], trs, sys.guiDotsCount, sys.guiDotsEquidist);
```

```
  }

  //nodes
  if (sys.guiDrawNodes)
   for (int n=0; n<line.NodesCount; n++)
    DrawNode(line.GetKnob(n), trs);
 }
}

private static void DrawNode (Knob knob, Matrix4x4 trs)
{
 Vector3 pos = knob.pos + new Vector3(trs.m03, trs.m13, trs.m23);

 Handles.DotHandleCap(0, pos, Quaternion.identity, HandleUtility.GetHandleSize(pos)/knobSize, EventT

 if (knob.type == Node.TangentType.correlated  ||  knob.type  == Node.TangentType.broken)
 {
  if (knob.inDefined) Handles.DrawLine(pos, pos+knob.inDir);
  if (knob.outDefined) Handles.DrawLine(pos, pos+knob.outDir);
 }
}

private static void DrawSegment (Segment segment, Matrix4x4 trs, bool dotted=false, bool equidistMarke
{
 Vector3 pos = new Vector3(trs.m03, trs.m13, trs.m23);
 Handles.DrawBezier(
```

```
  segment.start.pos + pos,
  segment.end.pos + pos,
  segment.start.pos + segment.start.dir + pos,
  segment.end.pos + segment.end.dir + pos,
   Handles.color, null, 2);
}


 private static void DrawDots (Segment segment, Matrix4x4 trs, int num, bool equidist=false)
 {
  for (int i=1; i<num+1; i++)
  {
   float percent = 1f*i/(num+1);

   if (equidist)
    percent = segment.NormLengthToPercent(percent);

   Vector3 point = segment.GetPoint(percent);
   Handles.DotHandleCap(0, point, Quaternion.identity, HandleUtility.GetHandleSize(point)/knobSize/3, Ev
  }
 }


#endregion
```

```
#region Edit


 public static bool EditSplineSys (SplineSys sys, Matrix4x4 trs, Object undoObject=null)
 /// Select, move, rotate, scale nodes (and draw selection/transform gizmos)
 /// Returns true if there was a change
 {
  bool added = false;
  bool removed = false;
  bool moved = false;


  //clearing selection if it's bounds changed
  if (!IsSplineNodeNumbersMatch(sys))
  {
   selectedKnobs.Clear();
   dispSelectedKnobs.Clear();
  }



  //adding/removing  points (before selection to remove selected)
  Event eventCurrent = Event.current;
  if (eventCurrent.type == EventType.MouseUp  &&  eventCurrent.button == 0  &&  !eventCurrent.alt  &&
  {
   //adding
   if (eventCurrent.shift)
    added = AddNode(sys, eventCurrent.mousePosition, undoObject);
```

```
 //removing
 else if (eventCurrent.control)
  removed = RemoveNode(sys, eventCurrent.mousePosition, undoObject);
}



//if selected single - drawing full ui with tangents
if (selectedKnobs.Count == 1)
{
 (int l, int n) = selectedKnobs.Any();
 Knob knob = sys.lines[l].GetKnob(n);

 bool dirChange, posChange = false;
 posChange = MoveNode(ref knob);
 dirChange = EditTangents(ref knob);

 moved = posChange || dirChange;

 if (moved)
 {
  if (undoObject!=null)
   Undo.RecordObject(undoObject, "Spline Node Move");

  sys.lines[l].SetKnob(knob, n);
  sys.lines[l].Update();
```

```
 }
}



//if selected many - transforming them
if (selectedKnobs.Count > 1)
{
 Knob[] knobs = new Knob[selectedKnobs.Count];

 int i=0;
 foreach ((int l, int n) in selectedKnobs)
 {
  knobs[i] = sys.lines[l].GetKnob(n);
  i++;
 }

 moved = MoveSelectedNodes(knobs);

 if (moved) //updating all selected nodes
 {
  if (undoObject!=null)
   Undo.RecordObject(undoObject, "Spline Node Move");

  HashSet<Line> linesToUpdate = new HashSet<Line>(); //don't update lines per-node since several nod

  i=0;
```

```
  foreach ((int l, int n) in selectedKnobs)
  {
   sys.lines[l].SetKnob(knobs[i], n);
   if (!linesToUpdate.Contains(sys.lines[l])) linesToUpdate.Add(sys.lines[l]);
   i++;
  }

  foreach (Line line in linesToUpdate)
   line.Update();
 }
}


//selecting nodes (after move, to avoid drawing frame instead of moving)
SelectNodes(sys, trs);

//re-drawing selected nodes
if (selectedKnobs.Count != 0)
{
 Color hColor = Handles.color;
 Handles.color = new Color(1,1,0,1);

 foreach ((int l, int n) in selectedKnobs)
 {
  Knob knob = sys.lines[l].GetKnob(n);
  DrawNode(knob, trs);
```

```
  }


  Handles.color = hColor;
 }


 return added || removed || moved;
}



private static bool IsSplineNodeNumbersMatch (SplineSys sys)
/// Compares sys lines count (and number of nodes in each) with stored array
/// BTW Will modify the array if not match
{
 bool match = true;


 if (selectionNumsBounds.Length != sys.lines.Length)
  { match = false; selectionNumsBounds = new int[sys.lines.Length]; }


 if (match)
  for (int l=0; l<sys.lines.Length; l++)
  {
   int nodesCount = sys.NodesCount;
   if (selectionNumsBounds[l] != nodesCount)
    match = false;
   selectionNumsBounds[l] = nodesCount;
  }
```

```
 return match;
}


private static void SelectNodes (SplineSys sys, Matrix4x4 trs)
/// Draws selection and selects new nodes (segment starts and junctions). Returns the list of currently sel
{
 //disabling selection
 HandleUtility.AddDefaultControl(GUIUtility.GetControlID(FocusType.Passive));

 SceneEdit.Select.UpdateFrame();

 bool selectionChanged = false;

 for (int l=0; l<sys.lines.Length; l++)
 {
  Line line = sys.lines[l];
  for (int n=0; n<line.NodesCount; n++)
  {
   Knob knob = line.GetKnob(n);

   bool isSelected = selectedKnobs.Contains((l,n));
   bool isDispSelected = dispSelectedKnobs.Contains((l,n));

   bool newSelected = isSelected;
```

```
  bool newDispSelected = isDispSelected;

  SceneEdit.Select.CheckSelected(knob.pos, knobSize/2, ref newSelected, ref newDispSelected);

  if (newSelected && !isSelected)
  {
   if (!SceneEdit.Select.isFrame)  selectedKnobs.Clear(); //selecting only one (but de-selecting all)
   selectedKnobs.Add((l,n)); selectionChanged=true;
  }

  if (!newSelected && isSelected)
   { selectedKnobs.Remove((l,n)); selectionChanged=true; }

  if (newDispSelected && !isDispSelected)
   dispSelectedKnobs.Add((l,n));

  if (!newDispSelected && isDispSelected)
   dispSelectedKnobs.Remove((l,n));
 }
}

if (selectionChanged)
{
 Editor[] allEditors = Resources.FindObjectsOfTypeAll<Editor>();
 for (int e=0; e<allEditors.Length; e++)
  allEditors[e].Repaint();
```

```
 }
}



private static bool MoveNode (ref Knob knob)
/// Draws transform gizmo for the node, and transform it with it's tangents (i.e. it could be rotated and scal
{
 //hiding the defaul tool
 UnityEditor.Tools.hidden = true;

 //gathering all positions
 Vector3[] allPositions = new Vector3[] {
  knob.pos,
  knob.pos + knob.outDir,
  knob.pos + knob.inDir };

 //transforming
 bool changed = MoveRotateScale.Update(allPositions, knob.pos);

 //setting back positions
 if (changed)
 {
  knob.pos = allPositions[0];
  knob.outDir = allPositions[1] - knob.pos;
  knob.inDir = allPositions[2] - knob.pos;
 }
```

```
 return changed;
}


private static bool EditTangents (ref Knob knob)
{
 bool change = false;

 //moving tans
 if (knob.type == Node.TangentType.broken || knob.type == Node.TangentType.correlated)
 {
  if (knob.outDefined)
  {
   Vector3 outTanPos = knob.pos+knob.outDir;
   Vector3 newOutTanPos = Handles.PositionHandle(outTanPos, Quaternion.identity);
   if ((outTanPos - newOutTanPos).sqrMagnitude != 0)
   {
    knob.outDir = newOutTanPos - knob.pos;
    if (knob.type == Node.TangentType.correlated) knob.inDir = Node.CorrelatedInTangent(knob.inDir, kn
    change = true;
   }
  }

  if (knob.inDefined)
  {
```

```
   Vector3 inTanPos = knob.pos+knob.inDir;
   Vector3 newInTanPos = Handles.PositionHandle(inTanPos, Quaternion.identity);
   if ((inTanPos - newInTanPos).sqrMagnitude != 0)
   {
    knob.inDir = newInTanPos - knob.pos;
    if (knob.type == Node.TangentType.correlated) knob.outDir = Node.CorrelatedOutTangent(knob.inDir,
    change = true;
   }
  }
 }

 return change;
}



private static bool MoveSelectedNodes (Knob[] knobs)
/// Draws currently selected tool gizmo and translates the selected nodes
/// Not only moves, but rotates and scales
{
 //hiding the defaul tool
 UnityEditor.Tools.hidden = true;

 //gathering all positions
 Vector3[] allPositions = new Vector3[knobs.Length*3]; //for each node 3 positions: 1 pos and 2 tangents
 Vector3 pivot = new Vector3(0,0,0);
 for (int k=0; k<knobs.Length; k++)
```

```
{
 allPositions[k*3] = allPositions[k*3+1] = allPositions[k*3+2] = knobs[k].pos;
 if (knobs[k].inDefined) allPositions[k*3+1] += knobs[k].inDir;
 if (knobs[k].outDefined) allPositions[k*3+2] += knobs[k].outDir;

 pivot += knobs[k].pos;
}
pivot /= knobs.Length;

//transforming
bool changed = MoveRotateScale.Update(allPositions, pivot);

//setting back positions
if (changed)
{
 for (int k=0; k<knobs.Length; k++)
 {
  knobs[k].pos = allPositions[k*3];
  knobs[k].inDir = allPositions[k*3+1] - knobs[k].pos;
  knobs[k].outDir = allPositions[k*3+2] - knobs[k].pos;
 }
}

 return changed;
}
```

```
private static bool AddNode (SplineSys sys, Vector2 mousePos, Object undoObject=null)
{
 bool change = false;

 float DistFn (Vector3 pointOnLine, Vector3 tempPoint)
 {
  Vector2 screenPoint = HandleUtility.WorldToGUIPoint(pointOnLine);
  return (screenPoint-mousePos).magnitude;
 }

 (int l, int s, float p, float dist) = sys.GetClosest(DistFn);

 if (dist < 10) //10 pixels from line on screen
 {
  //avoiding adding a node instead selection
  float minNodeDist =  GetNodeDistance(sys, mousePos);
  if (minNodeDist < knobSize/2)
   return false;

  if (undoObject!=null)
   Undo.RecordObject(undoObject, "Spline Node Add");

  Line line = sys.lines[l];
```

```
  line.Split(s, p);

  change = true;
  line.Update();
  EditorWindow.focusedWindow?.Repaint();
 }

 return change;
}

private static bool RemoveNode (SplineSys sys, Vector2 mousePos, Object undoObject)
{
 bool change = false;

 int lineNum = 0;
 int nodeNum = 0;
 float minDist = GetNodeDistance(sys, mousePos, out lineNum, out nodeNum);

 if (minDist < knobSize/2) //10 pixels from line on screen
 {
  if (undoObject!=null)
   Undo.RecordObject(undoObject, "Spline Node Remove");

  Line line = sys.lines[lineNum];
  line.RemoveNode(nodeNum);
```

```
  change = true;
  line.Update();
  if (UnityEditor.EditorWindow.focusedWindow != null) UnityEditor.EditorWindow.focusedWindow.Repaint
 }

 return change;
}

private static float GetNodeDistance (SplineSys sys, Vector2 mousePos, out int lineNum, out int nodeNu
/// Gets the closest node in screen-space, returns the distance to this node
{
 float minDist = Mathf.Infinity;
 lineNum = 0;
 nodeNum = 0;

 for (int l=0; l<sys.lines.Length; l++)
 {
  Line line = sys.lines[l];
  for (int n=0; n<line.NodesCount; n++)
  {
   Vector2 nodeScreenPos = HandleUtility.WorldToGUIPoint( line.GetNodePos(n) );
   float curDist = (nodeScreenPos-mousePos).magnitude;
   if (curDist < minDist)
   {
    lineNum = l;
    nodeNum = n;
```

```
        minDist = curDist;
      }
     }
    }

    return minDist;
   }

   private static float GetNodeDistance (SplineSys sys, Vector2 mousePos)
   {
    int lineNum = 0;
    int nodeNum = 0;
    return GetNodeDistance(sys, mousePos, out lineNum, out nodeNum);
   }

  #endregion
 }
}
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using UnityEngine.Profiling;

namespace Den.Tools.Tasks
{

 public static class CoroutineManager
 {
  public class Task
  {
   public string name = null;  //not used, for debug purpose
   public int priority;

   public List<Action> actions;
   public int actionNum = 0;

   public IEnumerator routine; //enumerator currently running

   public void Add (Action action)
   {
    if (actions == null)
     actions = new List<Action>();
```

```
  actions.Add(action);
 }

 public void Start () { CoroutineManager.Enqueue(this); }
 public void Stop () { CoroutineManager.Stop(this); }

 public bool Enqueued {get{ return queue.Contains(this); }}
 public bool Active {get{ return active==this; }}
}


private static List<Task> queue = new List<Task>();
private static Task active = null;

public static float timePerFrame = 3;
private static System.Diagnostics.Stopwatch timer = new System.Diagnostics.Stopwatch();

public static long updateNum = 0;


static CoroutineManager ()
{
 #if UNITY_EDITOR
 UnityEditor.EditorApplication.playModeStateChanged -= AbortOnPlaymodeChange;
 Application.wantsToQuit -= AbortOnExit;
 //UpdateCaller.Update -= UpdateMain;
```

```
 UnityEditor.EditorApplication.playModeStateChanged += AbortOnPlaymodeChange;
 Application.wantsToQuit += AbortOnExit;
 //UpdateCaller.Update += UpdateMain; //updating main actions/routines every frame
 #endif
}



public static Task Enqueue (Action action, int priority=0, string name=null)
{
 Task task = new Task() { actions = new List<Action>() {action}, priority = priority, name = name };
 CoroutineManager.Enqueue(task);
 return task;
}

public static Task Enqueue (IEnumerator routine, int priority=0, string name=null)
{
 Task task = new Task() { routine = routine, priority = priority, name = name };
 CoroutineManager.Enqueue(task);
 return task;
}

public static void Enqueue (Task task)
{
 lock (queue)
 {
```

```
  if (queue.Contains(task)) return;  //already enqueued

  if (active == task)  //already running - restarting
  {
   active.routine = null;
   active = null;
  }

  queue.Add(task);
 }
}

public static void Dequeue (Task task)
{
 lock (queue)
  if (queue.Contains(task))
   queue.Remove(task);
}

public static void Stop (Task task)
{
 lock (queue)
 {
  if (queue.Contains(task))
   queue.Remove(task);
```

```
 }

 if (active == task)
  active = null;
}



public static void Update ()
{
 updateNum++;

 timer.Reset();
 while (timer.ElapsedMilliseconds < timePerFrame  ||  !timer.IsRunning) //!IsRunning to iterate only once
 {
  if (!timer.IsRunning) timer.Start();

  //taking new task
  if (active == null)
   lock (queue) //new tasks could be added at the end of threads
  {
   if (queue.Count == 0) break;

   int taskNum = GetMaxPriorityNum(queue);
   active = queue[taskNum];
   queue.RemoveAt(taskNum);
  }
```

```
  //moving actions
  bool move = false;
  if (active.actions != null  &&  active.actions.Count != 0  &&  active.actionNum < active.actions.Count)
  {
   try  { active.actions[active.actionNum](); }
   catch(Exception e) { throw new Exception("Routine error: " + e); }
   finally
   {
    active.actionNum ++;
    move = true;
   }
  }

  //moving active routine
  if (!move  &&  active.routine != null)
   move = active.routine.MoveNext();

  if (!move)
  {
   if (active!=null) active.routine = null;
   active = null;
  }
 }
 timer.Stop();
}
```

```
public static int GetMaxPriorityNum (List<Task> list)
{
 int maxPriority = int.MinValue;
 int maxPriorityNum = -1;

 int listCount = list.Count;
 for (int i=list.Count-1; i>=0; i--) //for FIFO
 {
  int priority = list[i].priority;
  if (priority > maxPriority)
  {
   maxPriority = priority;
   maxPriorityNum = i;
  }
 }

 return maxPriorityNum;
}


public static void Abort ()
{
 //clearing queue (in that order so no job will pass from queue to active)
 lock (queue)
```

```
  queue.Clear();


 //clearing active
 if (active != null)
 {
  try
  {
   if (active.routine!=null)
    active.routine.Reset();
  }
  catch (Exception) {} //error on project start/stop
  active.routine = null;
 }
}

#if UNITY_EDITOR
static void AbortOnPlaymodeChange (UnityEditor.PlayModeStateChange state)
{
 if (state==UnityEditor.PlayModeStateChange.ExitingEditMode || state==UnityEditor.PlayModeStateChan
  Abort();
}
#endif



static bool AbortOnExit ()
{
```

```
 Abort(); return true;
}

public static bool IsWorking {get{ return queue.Count!=0 || active!=null; }}

public static bool IsQueueEmpty {get{ return queue.Count==0; }} //useful for checking if there any tasks le

public static bool IsNameEnqueued (string name)
{
 lock (queue)
  if (queue.FindIndex(c=>c.name==name) >= 0)
   return true;
 return false;
}

public static bool IsNameActive (string name)
{
 if (active != null && active.name == name) return true;
 return false;
}

public static string DebugState ()
/// returns active thread names and queue names to debug
{
```

```
    //debug usually happens on border state, so locking
    string queueNames = "";
    lock (queue)
     queueNames = queue.ToStringMemberwise<Task>(t => t.name);

    return "Coroutine active: " + (active==null ? "null" : active.name) + "\n" + "Coroutine queue: " + queueNam
   }
  }
 }
```

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[ExecuteInEditMode] //to call onEnable, then it will subscribe to editor update
public class CoroutineManagerObject : MonoBehaviour
/// More of a snippet on how to update coroutines
{
 public int timePerFrame = 30;

 public void OnEnable ()
 {
  #if UNITY_EDITOR
  UnityEditor.EditorApplication.update += Update;
  #endif
 }

 public void Update ()
 {
  Den.Tools.Tasks.CoroutineManager.timePerFrame = timePerFrame;
  Den.Tools.Tasks.CoroutineManager.Update();
 }



}
```

```
ï»¿using UnityEngine;
using UnityEditor;
using System;
using System.Threading;
using System.Collections.Generic;
//using UnityEngine.Profiling;
using Unity.Jobs;

namespace Den.Tools.Tasks
{
 public static class ThreadManager
 {
  public class Task
  {
   public string name; //for debug purpose, never used
   public int priority;
   public Action action;
   public Thread thread;

   public bool Enqueued {get{ return queue.Contains(this); }}
   public bool Active {get{ return active.Contains(this); }}
   public bool IsAlive {get{ return queue.Contains(this) || active.Contains(this); }}
  }

  private static List<Task> queue = new List<Task>();
  private static List<Task> active = new List<Task>();
```

```
public static int maxThreads = 3;
public static int processorThreads = -1;
public static bool autoMaxThreads = true;
public static bool useMultithreading = true;

static ThreadManager ()
{
 #if UNITY_EDITOR
 EditorApplication.playModeStateChanged -= AbortOnPlaymodeChange;
 Application.wantsToQuit -= AbortOnExit;
 //UpdateCaller.Update -= UpdateMain;

 EditorApplication.playModeStateChanged += AbortOnPlaymodeChange;
 Application.wantsToQuit += AbortOnExit;
 //UpdateCaller.Update += UpdateMain; //updating main actions/routines every frame
 #endif
}


public static Task Enqueue (Action action, int priority=0, string name=null)
{
 Task task = new Task() { action=action, priority=priority, name=name };
 ThreadManager.Enqueue(task);
 return task;
}
```

```
public static void Enqueue (ref Task task, Action action, int priority=0, string name=null)
{
 task = new Task() { action=action, priority=priority, name=name };
 ThreadManager.Enqueue(task);
}


public static void Enqueue (Task task)
{
 //if already executing - do nothing
 lock (active)
  if (active.Contains(task))
   return;

 lock (queue)
 {
  //if not in queue - enqueueing
  if (!queue.Contains(task))
   queue.Add(task);

  //if in queue  - do nothing
  else
   return;
 }
```

```
 LaunchThreads();
}


public static void Dequeue (Task task)
{
 //if job is in queue and has not been started - just remove from queue
 lock (queue)
  if (queue.Contains(task))
  {
   queue.Remove(task);
   return;
  }
}


public static void LaunchThreads ()
{
 lock (active)
  while (true) //active.Count < maxThreads && queue.Count != 0)
 {
  int curMaxThreads = maxThreads;
  if (autoMaxThreads)
  {
   if (processorThreads < 0) processorThreads = SystemInfo.processorCount; //the first time LaunchThrea
```

```
 curMaxThreads = processorThreads-1;
}

if (active.Count >= curMaxThreads) break;

Task task;
lock (queue)
{
 if (queue.Count == 0) break;

 int jobNum = GetMaxPriorityNum(queue);
 task = queue[jobNum];
 queue.RemoveAt(jobNum);
}

active.Add(task);

if (useMultithreading)
{
 Thread thread = new Thread(task.TaskThreadAction);
 lock (task)
  task.thread = thread;
 thread.Start();
}
else
 task.TaskThreadAction();
```

```
 }
}



public static void TaskThreadAction (this Task task)
{
 try
  { task.action(); }

 catch (ThreadAbortException) { }

 #if !MM_DEBUG
 catch (Exception e)
  { Debug.LogError("Thread failed: " + e); }  //throw exception ignores VS
  //{ throw new Exception("Thread failed: " + e); }
 #endif

 finally
 {
  lock (active)
   active.Remove(task); //it SHOULD be in active

  //task.ended = true;
  LaunchThreads(); //updating threads once one done
 }
}
```

```
  public static int GetMaxPriorityNum (List<Task> list)
  {
   int maxPriority = int.MinValue;
   int maxPriorityNum = -1;

   for (int i=list.Count-1; i>=0; i--) //for FIFO
   {
    int priority = list[i].priority;
    if (priority > maxPriority)
    {
     maxPriority = priority;
     maxPriorityNum = i;
    }
   }

//   Debug.Log("Selected " + maxPriority + " from list of " + list.Count + "\n" + list.ToStringMemberwise<Tas

   return maxPriorityNum;
  }


  public static void Abort ()
  {
   //clearing queue (in that order so no job will pass from queue to active)
```

```
 lock (queue)
  queue.Clear();

 //clearing active
 List<Task> activeCopy;
 lock (active)
 {
  activeCopy = new List<Task>(active);
  active.Clear();
 }

 for (int i=0; i<activeCopy.Count; i++)
 {
  Task task = activeCopy[i];

  lock (task)
   if (task.thread != null)
    task.thread.Abort();
 }
}

#if UNITY_EDITOR
static void AbortOnPlaymodeChange (PlayModeStateChange state)
{
 if (state==PlayModeStateChange.ExitingEditMode || state==PlayModeStateChange.ExitingPlayMode)
  Abort();
```

```
}
#endif



static bool AbortOnExit ()
{
 Abort(); return true;
}



public static int Count
{get{
 return queue.Count + active.Count;
}}

public static bool IsWorking {get{ return queue.Count!=0 || active.Count!=0; }}



public static string DebugState ()
/// returns active thread names and queue names to debug
{
 string activeNames;
 string queueNames;

 //debug usually happens on border state, so locking
 lock (active)
```

```
    activeNames = active.ToStringMemberwise<Task>(t => t.name);
   lock (queue)
    queueNames = queue.ToStringMemberwise<Task>(t => t.name);

   return "Thread active: " + activeNames + "\n" + "Thread queue: " + queueNames;
  }
 }
}
```