DIGGABLE TERRAINS 2D

version 2.1.0

Scripting API

Introduction

You are reading the Scripting API, which describes the main classes and functions.

Topics

Getting Started

Namespace

All classes are inside the "ScriptBoy.DiggableTerrains2D" namespace.

Classes

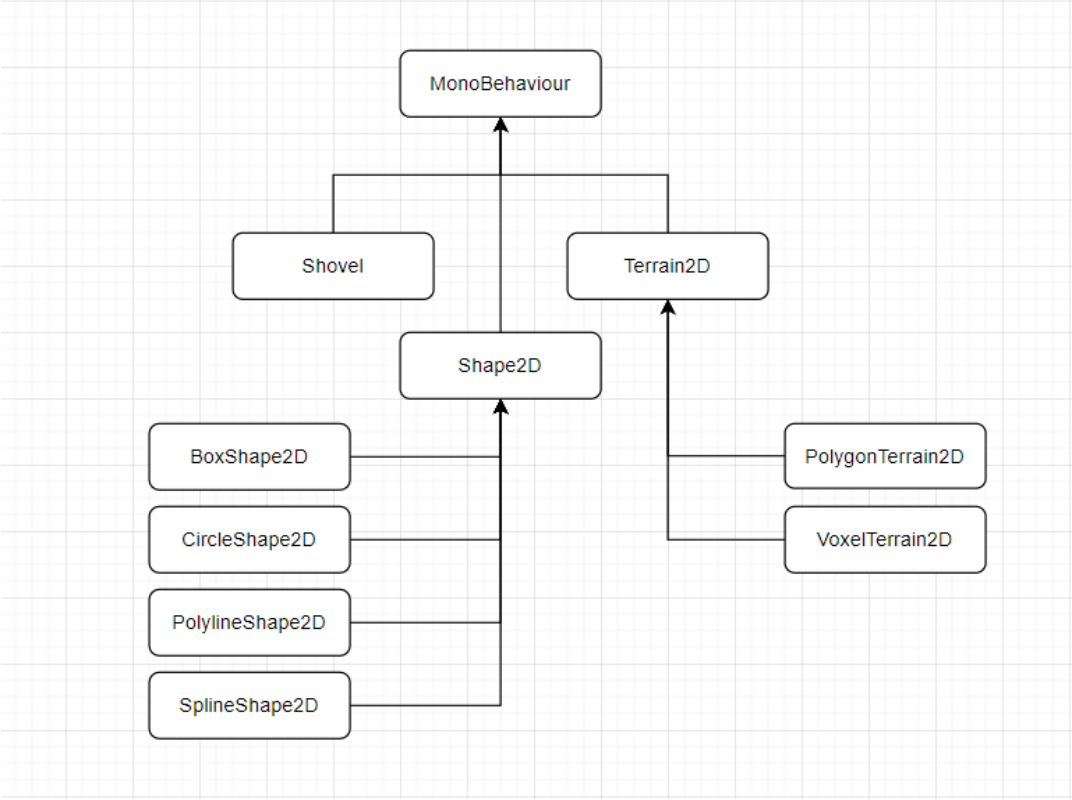
Shovel

Terrain2D
PolygonTerrain2D
VoxelTerrain2D

Shape2D
BoxShape2D
CircleShape2D
PolylineShape2D
SplineShape2D

TerrainParticleUtility

Hierarchy



How to dig terrains?

There are two ways to dig terrains.

A. You can create a shovel component and then call the **Dig** function.

```
using UnityEngine;
using ScriptBoy.DiggableTerrains2D;

public class Player : MonoBehaviour
{
    [SerializeField] Shovel m_Shovel;

    void Update()
    {
        if (Input.GetMouseButton(0)) m_Shovel.Dig();

        transform.position = (Vector2)Camera.main.ScreenToWorldPoint(Input.mousePosition);
    }
}
```

Please go to the Shovel class for more information.

B. You can dig terrains directly. However, using a shovel component is better.

```
using UnityEngine;
using ScriptBoy.DiggableTerrains2D;

public class Player : MonoBehaviour
{
    [SerializeField] Terrain2D m_Terrain;

    void Update()
    {
        if (Input.GetMouseButton(0)) m_Terrain.DigCircle(transform.position, 1);

        transform.position = (Vector2)Camera.main.ScreenToWorldPoint(Input.mousePosition);
    }
}
```

Please go to the Terrain2D class for more information.

How to play sounds?

The **Dig** function returns true if it modifies any terrains.

If the digging is not continuous, you can easily play a sound.

```
if (Input.GetMouseButtonDown(0))
{
    if (m_Shovel.Dig())
    {
        m_AudioSource.PlayOneShot(m_DigSound);
    }
}
```

If the digging is continuous, it's better to use a float variable to play sounds with a delay.

```
if (Input.GetMouseButton(0))
{
    if (m_Shovel.Dig())
    {
        if (m_DigSoundDelay < 0)
        {
            m_DigSoundDelay = 0.15f;
            m_AudioSource.PlayOneShot(m_DigSound);
        }
    }
}

m_DigSoundDelay -= Time.deltaTime;
```

How to emit particles?

The easiest way could be to call the **Play** and **Stop** functions of the **ParticleSystem**. However, there are also some better ways. We can use the **Emit** function of the **ParticleSystem** to manually create particles.

The **Dig** function provides the amount of modified area. It can be used to calculate the number of particles to emit.

```
if (m_Shovel.Dig(out float area))
{
    int particleCount = area * 100;
    m_ParticleSystem.Emit(particleCount);
}
```

If you want a more accurate result, you can use the **TerrainParticleUtility** class.

```
List<Vector2> particles = new List<Vector2>();
TerrainParticleUtility.GetParticles(m_Shovel, particles, m_ParticleCount);

if (m_Shovel.Dig())
{
    for (int i = 0; i < particles.Count; i++)
    {
        ParticleSystem.EmitParams emit = new ParticleSystem.EmitParams();
        emit.position = particles[i];
        m_ParticleSystem.Emit(emit, 1);
    }
}
```

How to save and load terrains? (added in v2.1.0)

You can save and load terrain data to a file using the **Terrain2D.Save(string path)** and **Terrain2D.Load(string path)** methods.

```
using UnityEngine;
using ScriptBoy.DiggableTerrains2D;

public class SaveAndLoadTerrain : MonoBehaviour
{
    [SerializeField] Terrain2D m_Terrain;
    [SerializeField] string m_FileName;

    string filePath => Application.persistentDataPath + "/" + m_SaveName;

    void Save()
    {
        m_Terrain.Save(filePath);
    }

    void Load()
    {
        m_Terrain.Load(filePath);
    }

    void OnGUI()
    {
        using (new GUILayout.AreaScope(new Rect(5, 5, 100, 100)))
        {
            if (GUILayout.Button("Save")) Save();
            if (GUILayout.Button("Load")) Load();
        }
    }
}
```

If your target platform does not support directly saving to a file, you can use **byte[] Terrain2D.GetData()** and **Terrain2D.LoadData(byte[] data)** methods instead.

```
void Save()
{
   byte[] data = m_Terrain.GetData();
   YourTargetPlatfomSaveMethod("Terrain", data)
}

void Load()
{
    byte[] data = YourTargetPlatfomLoadMethod("Terrain")
    m_Terrain.LoadData(data);
}
```

The save options should be set in the Inspector window; however, you can also change them from scripts if needed.

```
m_Terrain.compressionMethod = BinaryCompressionMethod.NoCompression;
byte[] data = m_Terrain.GetData();
data = YourCustomCompressionMethod(data);
YourCustomSaveMethod(filePath, data);
```

Please go to the Terrain2D class for more information about save options.

When a scene loads, the **Terrain2D.Build()** method will be called automatically.
If you are going to use **Terrain2D.Load()** when a scene loads, you can disable **"Build On Awake"** from the Inspector window and only call **Terrain2D.Build()** if there is no save file to load.

```
void Start()
{
    if (System.IO.File.Exists(filePath))
    {
        m_Terrain.Load(filePath);
    }
    else
    {
        m_Terrain.Build();
    }
}
```

Shovel

Public Functions

bool Dig()
bool Dig(int layerMask)
bool Dig(out float dugArea)
bool Dig(out float dugArea, int layerMask)
The Dig function removes the area within the shove polygon and returns false if it fails.
dugArea: The amount of dug area.
layerMask: A layer mask that is used to selectively ignore some terrains.

bool Fill()
bool Fill(int layerMask)
bool Fill(out float filledArea)
bool Fill(out float filledArea, int layerMask)
The Fill function fills the area within the shove polygon and returns false if it fails.
filledArea: The amount of filled area.
layerMask: A layer mask that is used to selectively ignore some terrains.

Note: The 'Fill' function is not supported when there are multiple terrain components in the scene.

Vector2[] GetPolygon()
This function returns the shovel polygon in the world space.

Public Properties

Shape2D shape {get; set;}
The shape that is used to create the shove polygon.

float simplification {get; set;}
The simplification threshold.

bool enableDemo {get; set;}
Enables a quick demo for testing the dig function.

bool enableWave {get; set;}
Enables an effect that randomizes the shovel polygon.

bool waveLength {get; set;}
The length of the wave.

bool waveAmplitude {get; set;}
The amplitude of the wave.

Terrain2D

Public Functions

float DigCircle(Vector2 position, float radius)
This function removes the area within a circle and returns the amount of modified area.
position: The position of the circle in world space.
radius: The radius of the circle in world space.

float FillCircle(Vector2 position, float radius)
This function fills the area within a circle and returns the amount of modified area.
position: The position of the circle in world space.
radius: The radius of the circle in world space.

float EditByCircle(Vector2 position, float radius, bool fill)
This function edits the area within a circle and returns the amount of modified area.
position: The position of the circle in world space.
radius: The radius of the circle in world space.
fill: Set 'true' to fill the area, set 'false' to remove the area.

float DigPolygon(Vector2[] polygon)
This function removes the area within a polygon and returns the amount of modified area.
polygon: An array of points in world space that form a polygon.

float FillPolygon(Vector2[] polygon)
This function fills the area within a polygon and returns the amount of modified area.
polygon: An array of points in world space that form a polygon.

float EditByPolygon(Vector2[] polygon, bool fill)
This function edits the area within a polygon and returns the amount of modified area.
polygon: An array of points in world space that form a polygon.
fill: Set 'true' to fill the area, set 'false' to remove the area.

Note: The polygon should not have self-intersection.

void PaintSplatMap(Vector2 point, float radius, float softness, float opacity, int layer)
void PaintSplatMap(Vector2 point, float radius, float softness, float opacity, float layer)
void PaintSplatMap(Vector2 point, float radius, float softness, float opacity, Color color)
These functions paint the splat map with a circle brush.
point: The position of the brush in world space.
radius: The radius of the circle in world space.
softness: The softness of the brush, ranging from 0 to 1.
opacity: The opacity of the brush, ranging from 0 to 1.
layer(int): Select a layer by index.
layer(float): Select 2 layers by transition. (e.g. The value 2.3f means that the weight of Layer 2 is 0.7f and the weight of Layer 3 is 0.3f.)
color: Select layers by color channels. (R = Layer 0, G = Layer 1, B = Layer 2, A = Layer 4, R + G + B + A = 1)

void PaintSplatMap(Vector2 point, Texture texture, float size, float opacity, int layer)
void PaintSplatMap(Vector2 point, Texture texture, float size, float opacity, float layer)
void PaintSplatMap(Vector2 point, Texture texture, float size, float opacity, Color color)
These functions paint the splat map with a texture brush.
point: The position of the brush in world space.
texture: The texture of the brush.
size: The size of the brush in world space.
opacity: The opacity of the brush, ranging from 0 to 1.
layer(int): Select a layer by index.
layer(float): Select 2 layers by transition. (e.g. The value 2.3f means that the weight of Layer 2 is 0.7f and the weight of Layer 3 is 0.3f.)
color: Select layers by color channels. (R = Layer 0, G = Layer 1, B = Layer 2, A = Layer 4, R + G + B + A = 1)

void GetParticles(List<Vector2> points, List<Vector2> particles)
void GetParticles(List<Vector2> points, List<TerrainParticle> particles)
These functions collect particles at the given points and add them to the particles list.
If a point is inside the terrain, it will be removed from the points list and the corresponding particle will be added to the particles list.
points: A list of points in world space.
particles: A list to add the collected particles to.

Note: You can use the TerrainParticleUtility class instead.

void Build()
This function builds the terrain. If the terrain is already built, it will delete everything and build it again. However, it is not recommended to use this function as a reset/rebuild function due to its heavy resource usage.

void Save(string path) (added in v2.1.0)
This function saves the terrain data to a file at the specified path.

void Load(string path) (added in v2.1.0)
This function loads terrain data from a file at the specified path.

byte[] GetData() (added in v2.1.0)
This function returns a byte array containing the serialized terrain data.

void LoadData(byte[] data) (added in v2.1.0)
This function loads terrain data from the provided byte array.

Public Properties

bool isBuilt {get;} (added in v2.1.0)
Is the terrain built?

bool isDiggable {get;}
Is the terrain diggable?

bool isFillable {get;}
Is the terrain fillable?

Runtime Save Options (added in v2.1.0)

BinaryCompressionMethod compressionMethod {get; set;}
Compression method used when saving data.

BinaryCompressionLevel compressionLevel {get; set;}
Compression level used when saving data.

bool includeSplatMapInSave {get; set;}
Whether to include the splat map in the save data.

bool compressSplatMapInSave {get; set;}
Whether to compress the splat map in the save data.

Note: There are no APIs to modify main terrain properties at runtime, so you need set up everything in editor.

Public Static Functions

Terrain2D[] FindByMask(int layerMask, bool includeInactive)
This function finds and returns an array of Terrain2D objects based on a layer mask.

Terrain2D[] FindByTag(int layerMask, bool includeInactive)
This function finds and returns an array of Terrain2D objects based on a tag.

Public Static Properties

Terrain2D[] instances {get;}
This property returns an array of Terrain2D objects.

Terrain2D[] activeInstances {get;}
This property returns an array of active Terrain2D objects.

PolygonTerrain2D

Notes

> The 'Fill' function is not fully supported when the 'Physics' feature is enabled.

> The 'PaintSplatMap' function is not fully supported when the 'Physics' feature is enabled.

Public Static Functions

PolygonTerrain2D[] FindByMask(int layerMask, bool includeInactive)
This function finds and returns an array of PolygonTerrain2D objects based on a layer mask.

PolygonTerrain2D[] FindByTag(int layerMask, bool includeInactive)
This function finds and returns an array of PolygonTerrain2D objects based on a tag.

Public Static Properties

PolygonTerrain2D[] instances {get;}
This property returns an array of PolygonTerrain2D objects.

PolygonTerrain2D[] activeInstances {get;}
This property returns an array of active PolygonTerrain2D objects.

VoxelTerrain2D

Notes

> Editing voxel terrains using circles is much faster than using polygons. A shovel with a circle shape uses a circle polygon for voxel terrains. However, if you set the number of points to 20 or higher, it will use a real circle instead.

Public Static Functions

VoxelTerrain2D[] FindByMask(int layerMask, bool includeInactive)
This function finds and returns an array of VoxelTerrain2D objects based on a layer mask.

VoxelTerrain2D[] FindByTag(int layerMask, bool includeInactive)
This function finds and returns an array of VoxelTerrain2D objects based on a tag.

Public Static Properties

VoxelTerrain2D[] instances {get;}
This property returns an array of VoxelTerrain2D objects.

VoxelTerrain2D[] activeInstances {get;}
This property returns an array of active VoxelTerrain2D objects.

Shape2D

Public Properties

bool fill {get; set;}
If the shape is used for terrain, Set 'true' to fill areas, and set 'false' to remove areas.
If the shape is used for shovel, the value of this property does not matter.

Public Functions

Vector2[] GetLocalPoints()
This function returns the shape points in local space.

Vector2[] GetWorldPoints()
This function returns the shape points in world space.

Vector2[] GetRelativePoints(Transform transform)
Vector2[] GetRelativePoints(Matrix4x4 worldToLocalMatrix)
These functions return the shape points in local space relative to the given transform or matrix.

BoxShape2D

Public Properties

float width {get; set;}
The width of the box.

float height {get; set;}
The height of the box.

float cornerRadius {get; set;}
The radius that is used to round the corners.

int cornerPointCount {get; set;}
The number of additional points that are added to round the corners.

CircleShape2D

Public Properties

float radius {get; set;}
The radius of the circle.

int pointCount {get; set;}
The number of points that form the cicrle.

PolylineShape2D

Public Properties

bool loop {get; set;}
Connects the first and last control points to create a closed shape.

float thickness {get; set;}
The thickness of the polyline (when the loop is false).

int capPointCount {get; set;}
The number of points that are added to round the start and end of the polyline (when the loop is false).

int cornerPointCount {get; set;}
The number of additional points that are added to round the corners.

float cornerRadius {get; set;}
The radius used to round the corners.

int controlPointCount {get; set;}
The number of the control points.

Vector2[] localControlPoints {get; set;}
Gets or sets an array of the control points in local space.

Vector2[] worldControlPoints {get; set;}
Gets or sets an array of the control points in world space.

Public Functions

Vector2 GetLocalControlPoint(int index)
This function gets the control point in local space at the given index.

Vector2 GetWorldControlPoint(int index)
This function gets the control point in world space at the given index

void SetLocalControlPoint(Vector2 point, int index)
This function sets the control point in local space at the given index.

void SetWorldControlPoint(Vector2 point, int index)
This function sets the control point in world space at the given index.

void InsertLocalControlPoint(Vector2 point, int index)
This function inserts a control point in local space at the given index.

void InsertWorldControlPoint(Vector2 point, int index)
This function inserts a control point in world space at the given index.

void RemoveControlPoint(int index)
This function removes a control point at the given index.

SplineShape2D

Public Properties

bool loop {get; set;}
Connects the first and last control points to create a closed shape.

float thickness {get; set;}
The thickness of the spline (when the loop is false).

int capPointCount {get; set;}
The number of points that are added to round the start and end of the spline (when the loop is false).

int midPointCount {get; set;}
The number of points between two control points.

int controlPointCount {get; set;}
The number of the control points.

SplineControlPoint[] localControlPoints {get; set;}
Gets or sets an array of the control points in local space.

SplineControlPoint[] worldControlPoints {get; set;}
Gets or sets an array of the control points in world space.

Public Functions

SplineControlPoint GetLocalControlPoint(int index)
This function gets the control point in local space at the given index.

SplineControlPoint GetWorldControlPoint(int index)
This function gets the control point in world space at the given index

void SetLocalControlPoint(SplineControlPoint point, int index)
This function sets the control point in local space at the given index.

void SetWorldControlPoint(SplineControlPoint point, int index)
This function sets the control point in world space at the given index.

void InsertLocalControlPoint(SplineControlPoint point, int index)
This function inserts a control point in local space at the given index.

void InsertWorldControlPoint(SplineControlPoint point, int index)
This function inserts a control point in world space at the given index.

void RemoveControlPoint(int index)
This function removes a control point at the given index.

TerrainParticleUtility

Public Static Functions

void GetParticles(List<Vector2> points, List<TerrainParticle> particles, int layerMask = -1)
void GetParticles(List<Vector2> points, List<Vector2> particles, int layerMask = -1)
These functions collect particles at the given points and add them to the particles list.
If a point is inside a terrain, it will be removed from the points list and the corresponding particle will be added to the particles list.
points: A list of points in world space.
particles: A list to add the collected particles to.
layerMask: A layer mask that is used to selectively ignore some terrains.

void GetParticles(Shovel shovel, List<TerrainParticle> particles, int count, int seed = 0, int layerMask = -1)
void GetParticles(Shovel shovel, List<Vector2> particles, int count, int seed = 0, int layerMask = -1)
void GetParticles(Vector2[] shovelPolygon ,List<TerrainParticle> particles, int count, int seed = 0, int layerMask = -1)
void GetParticles(Vector2[] shovelPolygon, List<Vector2> particles, int count, int seed = 0, int layerMask = -1)
These functions generate random points inside the shovel polygon, then use them to collect the corresponding particles on terrains and add them to the particles list.
shovel/shovelPolygon: The shovel polygon is used to generate random points.
particles: A list to add the collected particles to.
count: The number of random points.
seed: The random state.
layerMask: A layer mask that is used to selectively ignore some terrains.

void GetParticles(Vector2 center, float radius, List<Vector2> particles, int count, int seed = 0, int layerMask = -1)
void GetParticles(Vector2 center, float radius, List<TerrainParticle> particles, int count, int seed = 0, int layerMask = -1)
These functions generate random points inside a circle, then use them to collect the corresponding particles on terrains and add them to the particles list.
center: The center position of the circle.
radius: The radius of the circle.
particles: A list to add the collected particles to.
count: The number of random points.
seed: The random state.
layerMask: A layer mask that is used to selectively ignore some terrains.

Links & Contact Info

https://youtube.com/DiggableTerrains2D

ScriptBoyTools@outlook.com

ScriptBoyUnity@gmail.com

Have Fun!
Script Boy
:)