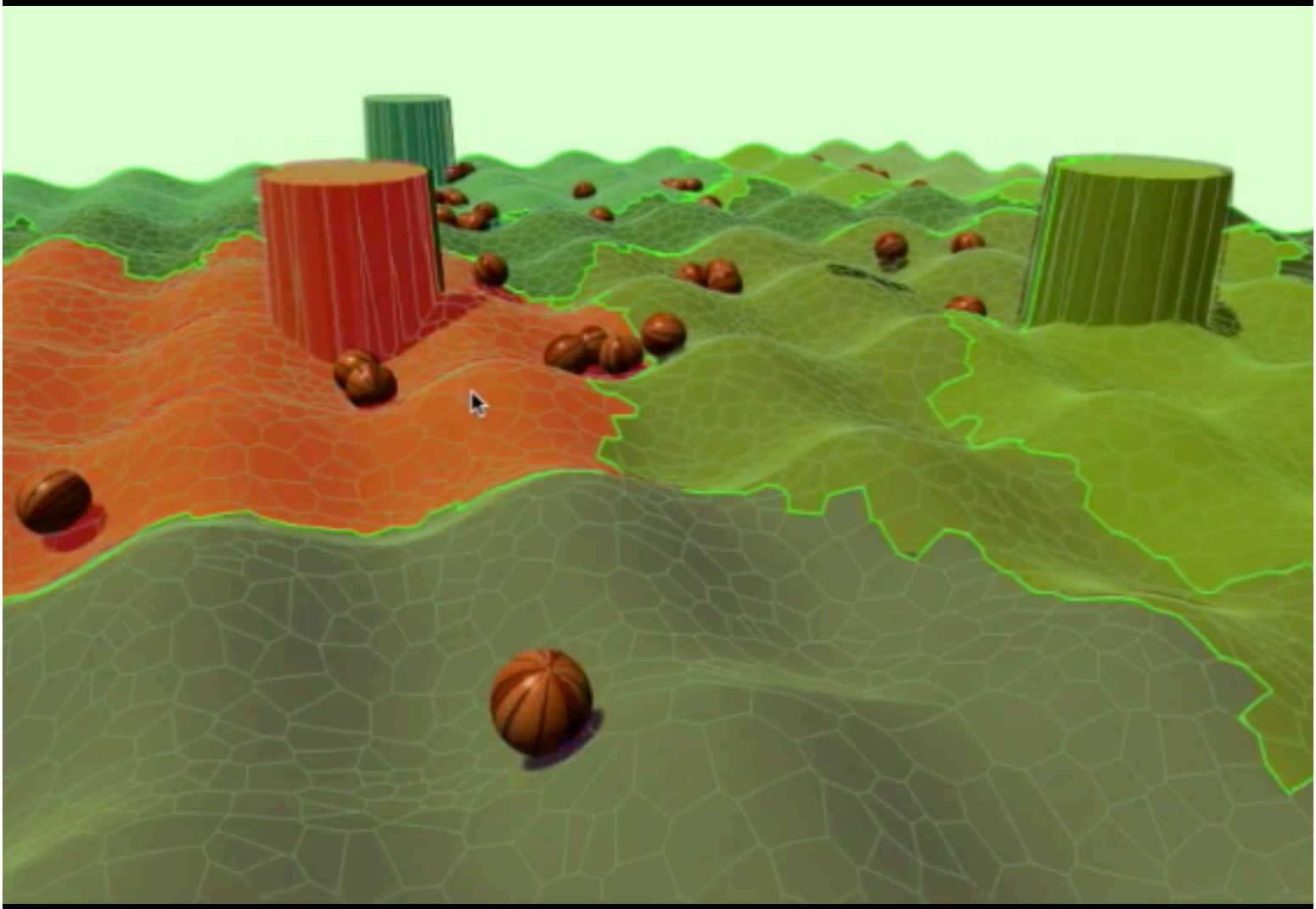
TERRAIN GRID SYSTEM



RELEASE NOTES

Index

Introduction

Thank you for purchasing!

Terrain Grid System is a commercial asset for Unity 5.1.1 (or later) that allows to:

- Add a configurable and flexible grid to Unity terrain
- Supports voronoi tessellation, boxed and hexagonal types.
- Two levels of regions: cells and territories.
- Powerful selectable and highlighting system for both cells and territories.
- Coloring and fade support.
- Different positioning options for best mesh adaptation to the terrain.
- Native A* Path Finding support.
- Extensive API (C#) for controlling the grid.
- Use textures to define visibility and cell ownership
- Can work alone or with Unity's Terrain object

You can use this asset for:

- Organizing content or areas of interest over a terrain
- Provide a generic selectable grid zone without using a terrain
- Allow the user to highlight and select a zone of the map
- Display with different colors regions of the map, either territories or cells
- Manage the grow of different regions, by combining cells into greater cells or other territories.

QuickStart and Demo Scenes

1. Import the asset into your project or create an empty project.
2. Go to Demo folders and run them to learn about common use cases.
3. Examine the code behind the script attached to the Demo game object.

The Demo scenes contain a TerrainGridSystem instance (the prefab) and a Demo gameobject which has a Demo script attached which you can browse to understand how to use some of the properties of the asset from code (C#).

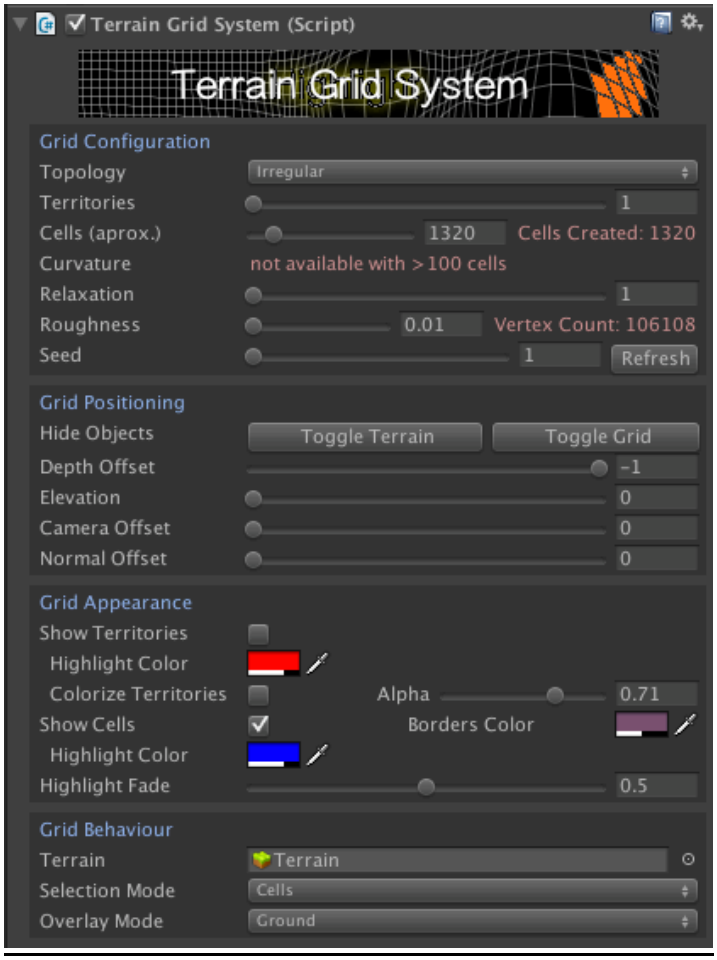
Support & Contact info

We hope you find the asset easy and fun to use. Feel free to contact us for any enquiry.
Visit our Support Forum for usage tips and access to the latest beta releases.

Kronnect
Email: contact@kronnect.me
Kronnect Support Forum: http://www.kronnect.me

How to use the asset in your project

Drag the prefab “TerrainGridSystem” from “Resources/Prefabs” folder to your scene.
Select the GameObject created to show custom properties and assign the terrain gameobject reference.



Custom Editor Properties

Grid Configuration: this section allows you to configure the grid irrespective of its appearance or usage. Basically these properties define the number of cells, territories and their shapes.

Topology: this parameters defines the grid overall shape. You can choose between irregular (which uses a highly optimized voronoi tesselation algorithm), boxed or hexagonal type.

Territories and Cells: each grid has a number of cells and OPTIONALLY a number of territories. A territories contains one or more cells, so you will always have more cells than territories (or same number).

Note that adding territories will make the system slower with a high number of cells. To prevent lag during runtime while creating the neccesary meshes you can pre-warm the geometry using the API. More on that later.

- **Curvature**: if your grid has 100 or less cells, you can apply an optional curvature. Note that using this parameter will x3 the number of segments of the grid.

- **Relaxation:** this option will make irregular shapes (Voronoi) more homogeneous. Basically it iterates over the voronoi calculation algorithm weighing and modulating the separation between cell centers. Each iteration will redo the voronoi calculation and even it’s been optimized that means that mesh generation times will go up with each relaxation level.

- **Roughness:** this option will determine the level of gripping of the grid to the terrain. The greater the value, more abrupt the grid will appear with respect to the terrain. You usually will want to leave this value to the minimum possible unless your terrain is quite flat (in that case, increasing roughness will reduce vertex count).

- **Seed:** this parameter will allow you to recreate the grid with same disposition of cells. This option basically affects the random functionality.

- **Max Slope:** specifies the maximum slope or inclination of the terrain that allows cells. A value of 1 means no slope is considered hence cells will be drawn regardless of this value. Reduce to prevent cells hang over steep terrain or walls.

- **Minimum Altitude:** specifies the minimum altitude to draw cells. Useful to hide cells under water for example.

- **Mask:** assign a texture in this slot to define cells visibility (alpha channel is used to determine visibility, where a=0 means that cell is invisible). An invisible cell won't be highlighted unless the "Highlight Invisible Cells" option is enabled. Invisible cells do not participate in pathfinding either.

- **Territories Texture:** assign a color texture to define territories shape or cell ownership. One territory for each different color will be created. Ensure you use solid colors (no smooth or gradients) in the texture. If you use Gimp, you can switch to Indexed Color and set a maximum number of colors, then switch back to RGBA and export the texture in PNG format. Optionally specify which color will be used as neutral, meaning those cell won't belong to any territory.

Grid Positioning

As the grid is a phisically generated mesh, it needs to cover the terrain subtly and effectively. This means that there should be an adequate offset between the terrain geometry and that of the grid. But at the same time that offset should be the minimum possible so the grid stays below any content put over the terrain.
To achieve the best possible effect, Terrain Grid System includes several parameters that allows you to control precisely how the grid is positioned with respect to the terrain mesh.

- **Depth Offset**: this parameter control how much offset is applied to the shaders used by Terrain Grid System. This value affects the Z-Buffer position of the pixels of both highlight surfaces (**Colored Depth Offset**) and grid mesh (**Mesh Depth Offset**).

- **Elevation** and **Elevation Base**: both values are summed to calculate the amount of displacement along the Y-axis. **Elevation Base** uses a numeric input entry format in the inspector instead of a slider to allow finer/greater height values.

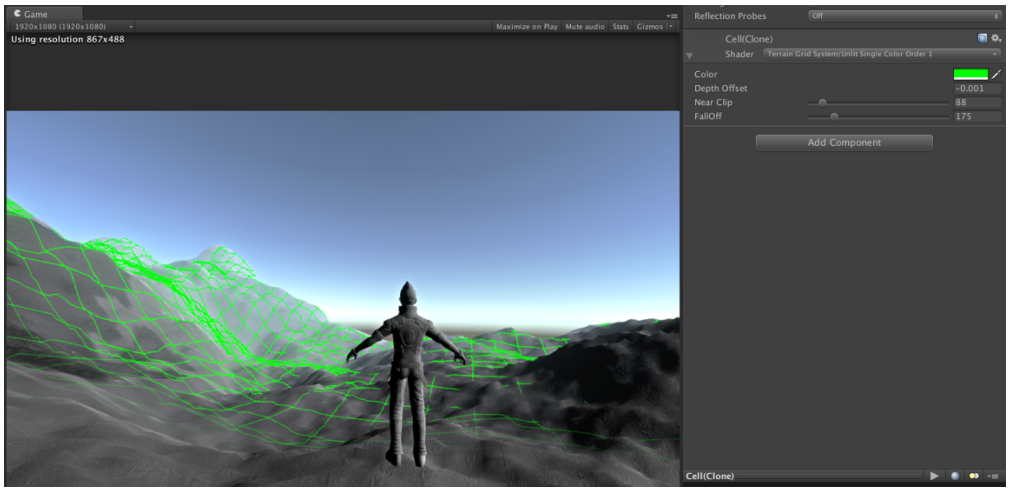
- **Camera Offset**: apply a displacemenet of the grid towards the camera position. This displacement is dynamic, meaning that it will move the grid automatically whenever the grid or the camera changes position.

- **Normal Offset**: this option will apply a displacement per vertex along the normal of the terrain below its position. It will make the grid "grow" around the terrain in all directions. Note that this is an expensive operation. You usually don't need to use this parameter.

Grid Appearance

This section controls the look and feel of the grid system and it should be self-explanatory. However it's important to know that activating either "Show Territories" or "Colorize Territories" (or set the highlight mode to Territory as well) will enable the territory mesh generation. So, if you don't use the territory functionality, make sure all options regarding territories are unchecked to save time and memory.

Grid lines are drawn using custom shaders that allow you to specify a near clip and falloff. These properties should be useful for those situations where the camera is really near the grid (like in FPS):



To change the default clipping and falloff values, use the Near Clip Fade and Near Clip Fade FallOff sliders.

Grid Behaviour

This section controls what the grid can do.

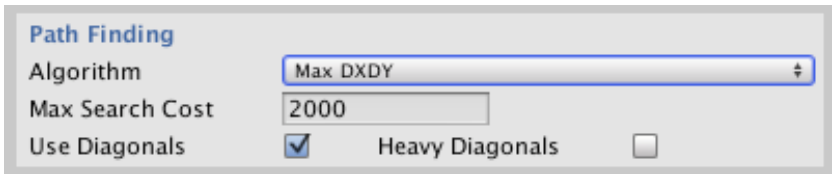
The first and most important field is the Terrain reference, which you should set to make the grid system start working with your terrain.

In addition to that, you may choose:

- **Selection Mode**: choose between None, Cells or Territories.
- **Overlay Mode**: choose between Ground or Overlay modes. The overlay mode will make the highlight surfaces to be displayed on top of everything. Not very fancy but it's the fastest and prevents any visual artifact with complex terrains. The default option, Ground, will make the highlight surfaces to appear below any content put over the terrain, effectively seeming that it's being painted over the terrain.
- **Respect Other UI**: when enabled, grid interaction is ignored while pointer is over a UI element (only uGUI elements are supported, those under a Canvas root element, not immediate GUI or elements created from OnGUI() functions).

Path Finding

This section controls the path finding behaviour.



- **Algorithm**: choose among available algorithms:

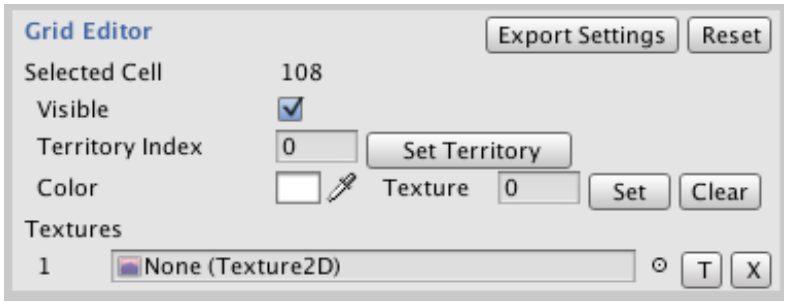
- **MaxDXDY**: use estimations based on horizontal/vertical distance to target. Produces more natural paths althouth not optimal.
- **Manhattan**: produces jaggy paths.
- **Diagonal Shortcut**: favours diagonals in path.
- **Euclidean**: estimation based on the straight distance to target. Produces the optimal path.
- **Euclidean Non SQR**: similar but don't uses sqrt which makes it faster.
- **Custom1**: experimental method.

- **Max Search Cost**: the maximum path length.
- **Use Diagonals**: if diagonals between cells can be used.
- **Heavy DIagonals**: add an extra cost for diagonals, making them less frequent.

The Path Finding algorithm is very fast but TGS also includes an optimized variant of the algorithm for grids that have column count power of 2 (4, 8, 16, 32, 64, 256, …).
Check demo scene 12 for example on using Path Finding with Terrain Grid System. Also read the available APIs for using path finding feature below.

Grid Editor

This section allows you to customize the cells in Editor time.



With the Scene View selected (not Game View), you can use the mouse to select any cell and change their properties.

Export Settings will create add a "TGS Config" script which serves as a container for all the settings and will load them when activated.

Reset will clear all cell settings and reverts them to default values.

Textures array is a convenient way to load several textures and use them quickly to paint several cells. Just click the "T" small buton to enable the "paint mode". Once enabled, you can click several cells to assign the chosen texture.

You can have several "TGS Config" scripts added. To load a specific configuration just enable the desired TGS Config component or call "LoadConfiguration()" method of that script.

Positioning the Grid on your terrain

Placing the grid at the center of the terrain and making it fit the entire terrain may be not a good idea if your terrain is very big. In this case and depending on your requirements, it's possible that you will need to draw a lot of cells. Although this can be done just entering the proper number of rows and columns in the inspector, it's not performance wise.

A solution for big terrains is just draw a portion of the grid around the character. For that, you need to set **gridCenter** and **gridScale** properties.

- GridCenter values range from -0.5 (left most side of the terrain) to 0.5 (right most side of the terrain). By default gridCenter is set to 0,0, which matches the center of the terrain. Setting it to -0.5, 0.5 for example will center the grid on the top/left corner of the terrain.
- GridScale values range from 0 to 1 and are relative to the terrain size. So a scale of 0.1,0.1 will make the grid a 10% of the size of the terrain.

Another useful property is gridCenterWorldSpace which lets you assign a world space position directly and the grid will be displayed around that point. For example you can assign the character transform.position to this property.

Check out demo scenes 10 and 10b for useful examples.

Programming Guide (API)

You can instantiate the prefab “TerrainGridSystem“ or add it to the scene from the Editor. Once the prefab is in the scene, you can access the functionality from code through the static instance property:

```
TerrainGridSystem tgs;

void Start () {
   tgs = TerrainGridSystem.instance;
         ...
}
```

To get the cell beneath a gameobject in your scene use CellGetAtPosition and pass the world space coordinate:

```
Vector3 characterPosition = transform.position;
Cell cell = tgs.CellGetAtPosition(characterPosition, true);
int row = cell.row;
int column = cell.column;
```

To fade surrounding cells:

```
List<Cell>neighbours = tgs.CellGetNeighbours(sphereCell);
foreach(Cell cell in neighbours) {
    tgs.CellFadeOut(cell, Color.red, 2.0f);
}
```

To fade cells by row and column:

```
const int size = 3;
for (int j=sphereCell.row - size; j<=sphereCell.row + size;j++) {
       for (int k=sphereCell.column - size; k<=sphereCell.column + size; k++) {
            int cellIndex = tgs.CellGetIndex(j,k, true);
            tgs.CellFadeOut(cellIndex, Color.blue, 1.0f);
       }
}
```

Take a look at the demo scenes included in the asset for more code snippets!

Public API structure

All TerrainGridSystem API is exposed through TerrainGridSystem class. This is a single component that has been split for convenience in several files located in TerrainGridSystem/Scripts folder:

- TerrainGridSystem: contains generic functions and properties.
- TGSCells: all cell-related.
- TGSTerritories: all territory-related.
- TGSPathfinding: pathfinding stuff.
- TGSConfig: this is the component that stores the cell configuration when you use and export the cell setup using the Grid Editor.

Note that the above are just filenames, but the class is the same "TerrainGridSystem". This approach follows the partial class standard in C# to make easy the reorganization of "God" (or big) utility classes. This means that you will be just using the traditional gameObject.GetComponent<TerrainGridSystem>() or just TerrainGridSystem.instance to get a reference to the API, irrespective of the number of classes files.

Thanks to this file organization you can easily check the different public properties and functions of the API by category (cell-related, territory-related, …)

Complete list of public properties, methods and events

Basic grid functions

tgs.numCells: desired number of cells. The actual generated cells will vary depending on the topology.

tgs.ColumnCount: number of columns for boxed and hexagonal types.

tgs.RowCount: number of rows for boxed and hexagonal types.

tgs.seed: seed constant for the random generator.

tgs.gridCenter: the center of the grid with respect to its parent (0,0 = center).

tgs.gridCenterWorldPosition: sets or retrieves the center of the grid in world space coordinates. You can also call **tgs.SetGridCenterWorldPosition** method instead.

tgs.gridScale: the scale of the grid with respect to its parent (1,1 = full terrain/grid size)

tgs.gridMask: optional texture used to defined cells visibility. The texture is matched against the whole grid parent surface. An alpha value of 0 means cell will be invisible.

tgs.gridTopology: type of cells (Irregular, Box, Hexagonal, …)

tgs.gridRelaxation: relaxation factor for the irregular topology (defaults 1). It creates more homogeneous irregular grids but has a performance hit during the creation of the grid (not afterwards).

tgs.gridCurvature: will bend cell edges for all topologies. It will add more vertex count to the mesh so it's only available on grids with less than 100 cells.

tgs.terrain: sets or returns the reference to the terrain (if used).

tgs.terrainCenter: returns the world space position of the center of the terrain.

tgs.gridElevation: vertical offset of the grid with respect to the terrain for fine-tuning z-fighting issues.

tgs.gridElevationBase: vertical position for the grid, used to elevate grid over terrain. Both gridElevation and gridElevationBase are added when computing the position of the grid. Use gridElevationBase if you want the grid to float above the terrain and gridElevation for small height adjustments.

tgs.gridCameraOffset: dynamic camera-oriented displacement. This will move the mesh towards the camera.

tgs.gridNormalOffset: general offset of the mesh from the terrain. This will make the grid "grow" in all directions. This will impact performance during the mesh creation.

tgs.gridDepthOffset: Z-Buffer offset for the grid system shaders. Very cheap method to prevent z-fighting which can be combined with above methods. You should use this property first to adjust your grid.

tgs.gridRoughness: gripping factor for the mesh. The lower the value the better fit of the mesh to the terrain.

tgs.nearClipFade: distance from the camera for the fade in effect of the grid. Useful in first-person-view games.

tgs.nearClipFadeFallOff: falloff or gradient amount for the fade in effect.

Cell related

tgs.cells: return a List<Cell> of all cells/states records.

tgs.showCells: whether to show or not the cells borders.

tgs.cellHighlighted: returns the cell/state object in the cells list under the mouse cursor (or null if none).

tgs.cellHighlightedIndex: returns the cell/state index in the cells list under the mouse cursor (or -1 if none).

tgs.cellHighlightNonVisible: sets if invisible cells should be highlighted when pointer is over them (default is true).

tgs.cellLastClickedIndex: returns the index of last cell clicked.

tgs.cellHighlightColor: color for the highlight of cells.

tgs.cellHighlightNonVisible: set to true if you want to be able to highlight invisible cells.

tgs.cellBorderColor color for all cells borders.

tgs.cellBorderAlpha: helper method to change only the alpha of cell borders.

tgs.CellSetTag: assigns a cell a user-defined integer tag. Cell can be later retrieved very quickly with CellGetWithTag.

tgs.CellGetWithTag: retrieves a cell previously tagged with an integer using the method CellSetTag.

tgs.CellGetIndex(Cell): returns the index of the provided Cell object in the Cells collection.

tgs.CellGetIndex(row, column, clampToBorders): returns the index of the provided Cell by row and column. Use clampToBorders = true to limit row/column to edges of the grid or to allow wrap around edges otherwise.

tgs.CellGetPosition(cellIndex): returns the center of a cell in the world space.

tgs.CellGetRect(cellIndex): returns the 2D rectangle enclosing the cell in local space (-0.5..0.5 range).

tgs.CellGetRectWorldSpace(cellIndex): returns the 3D bounds of the cell in world space.

tgs.CellGetVertexCount(cellIndex): returns the number of vértices of any cell.

tgs.CellGetVertexPosition(cellIndex, vertexIndex): returns the world space position of any vertex of a cell.

tgs.CellGetAtPosition(position, wordSpace): returns the cell that contains position (in local space coordinates which ranges from -0.5 to 0.5 or from a point in world space).

tgs.CellGetAtPosition(column, row): returns the cell located at given row and col (only boxed and hexagonal topologies).

tgs.CellGetNeighbours(cellIndex): returns a list of neighbour cells to a specific cell.

tgs.CellMerge(Cell 1, Cell 2): merges two cells into the first one.

tgs.CellSetTerritory(cellIndex, territoryIndex): changes the territory of a cell and updates boundaries. Call Redraw() to update the grid afterwards.

tgs.CellToggleRegionSurface(cellIndex, visible, color, refreshGeometry): colorize one cell's surface with specified color. Use refreshGeometry to ignore cached surfaces (usually you won't do this).

tgs.CellToggleRegionSurface(cellIndex, visible, color, refreshGeometry, texture): color and texture one cell's surface with specified color. Use refreshGeometry to ignore cached surfaces (usually you won't do this). Texture can be scaled, panned and/or rotated adding optional parameters to this function call.

tgs.CellToggleRegionSurface(cellIndex, visible, color, refreshGeometry, texture, textureScale, textureRotation, overlay): same but allows to specify a texture scale and rotation and optionally if the colored surface will be shown on top of objects.

tgs.CellHideRegionSurface(cellIndex): uncolorize/clears one cell's surface.

tgs.CellHideRegionSurfaces(): uncolorize/clears all cells.

tgs.CellFadeOut(cellIndex, color, duration): colorizes a cell and fades it out for duration in seconds.

tgs.CellFlash(cellIndex, color, duration): flashes a cell from given color to default cell color for duration in seconds.

tgs.CellBlink(cellIndex, color, duration): blinks a cell from current color to a given color and back to original cell color for duration in seconds.

tgs.CellGetColor(cellIndex): returns current cell's fill color.

tgs.CellGetTexture(cellIndex): returns current cell's fill texture.

tgs.CellGetTexture(cellIndex): returns current cell's fill texture.

tgs.CellIsBorder: returns true if the cell sits at the edges of the grid.

tgs.CellIsVisible: returns true if the cell is visible.

tgs.CellISetVisible: makes the cell visible or invisible. Call Redraw() to update the grid afterwards.

tgs.CellRemove: completely removes a cell from the grid. Call Redraw() to update the grid afterwards.

tgs.CellSetCanCross: specifies if a cell can be part fo a route returned by Pathfinding methods.

tgs.CellGetConfigurationData: retrieves a string-packed description of the cells properties. Use CellSetConfigurationData to load the configuration.

tgs.CellSetConfigurationData: loads a configuration of cells.

Territory related

tgs.numTerritoriess: desired number of territories (1-number of cells).

tgs.territories: return a List<Territory> of all territories.

tgs.showTerritories: whether to show or not the territories borders.

tgs.showTerritoriesOuterBorder: whether to show or not the territories perimeter around the grid.

tgs.territoryHighlighted: returns the Territory object for the territory under the mouse cursor (or null).

tgs.territoryHighlightedIndex: returns the index of territory under the mouse cursor (or -1 if none).

tgs.territoryLastClickedIndex: returns the index of last territory clicked.

tgs.territoryHighlightColor: color for the highlight of territories.

tgs.territoryFrontiersColor: color for all territory frontiers.

tgs.territoryDisputedFrontiersColor: color for shared frontiers between two territories.

tgs.territoryFrontiersAlpha: helper method to change only the alpha of territory frontiers.

tgs.colorizeTerritories: whether to fill or not the territories.

tgs.colorizedTerritoriesAlpha: transparency level for colorized territories.

tgs.TerritoryToggleRegionSurface(territoryIndex, visible, color, refreshGeometry): colorize one territory's surface with specified color. Use refreshGeometry to ignore cached surfaces (usually you won't do this).

tgs.TerritoryToggleRegionSurface(territoryIndex, visible, color, refreshGeometry, texture): color and texture one territory's surface with specified color. Use refreshGeometry to ignore cached surfaces (usually you won't do this). Texture can be scaled, panned and/or rotated adding optional parameters to this function call.

tgs.TerritoryToggleRegionSurface(cellIndex, visible, color, refreshGeometry, texture, textureScale, textureRotation, overlay): same but allows to specify a texture scale and rotation and optionally if the colored surface will be shown on top of objects.

tgs.TerritoryHideRegionSurface(territoryIndex): uncolorize/clears one territory's surface.

tgs.TerritoryGetNeighbours(territoryIndex): returns a list of neighbour territories to a specific territory.

tgs.TerritoryGetAtPosition(position): returns the territory that contains position (in local space coordinates).

tgs.TerritoryFadeOut(territoryIndex, color, duration): colorizes a territory and fades it out for duration in seconds.

tgs.TerritoryFlash(territoryIndex, color, duration): flashes a territory from given color to default territory color for duration in seconds.

tgs.TerritoryBlink(territoryIndex, color, duration): blinks a territory from current color to a given color and back to original territory color for duration in seconds.

tgs.TerritoryIsVisible(territoryIndex): returns true if territory is visible.

tgs.TerritorySetVisible(territoryIndex): makes a territory visible or invisible.

tgs.TerritorySetFrontierColor(territoryIndex, color): assigns a new color for the frontiers of a given territory.

tgs.CreateTerritories(texture, neutral): generate territories based on distinct colors contained in the provided texture ignoring neutral color.

User interaction

tgs.highlightMode: choose between None, Territories or Cells. Defaults to Cells.

tgs.highlightFadeAmount: alpha range for the fading effect of highlighted cells/territories (0-1).

tgs.highlightMinimumTerrainDistance: minimum distance to the terrain to allow highlighting of cells/territories.

tgs.overlayMode: decide whether the highlight is shown always on top or takes into account z-buffer which will make it more fancy and part of the scene.

Path Finding related

tgs.CellSetCanCross: specifies if this cell blocks any path.

tgs.FindPath(cellStartIndex, cellEndIndex): returns a list of cell indices that form the path.

tgs.FindPath(cellStartIndex, cellEndIndex, out totalCost, maxCost, maxSteps): returns a list of cell indices that form the path and the totalCost for the path. Optionally pass a maxCost and maxSteps (use OnCellCross event to return custom costs for any cell).

Other useful methods

tgs.instance: reference to the actual Terrain Grid System component on the scene.

tgs.Redraw: forces a redraw of the grid mesh. Optionally set the parameter reuseTerrainData to true to speed up computation if terrain has not changed.

Events

tgs.OnCellEnter: triggered when pointer enters a cell.
tgs.OnCellExit: triggered when pointer exits a cell.
tgs.OnCellClick: triggered when user clicks or taps a cell (visible or not).
tgs.OnCellHighlight: triggered when a cell is goint to be highlighted (usually when pointer is over it). A ref parameter allows you to cancel highlight and implement your own coloring system.

tgs.OnTerritoryEnter: triggered when pointer enters a territory.
tgs.OnTerritoryExit: triggered when pointer exits a territory.
tgs.OnTerritoryClick: triggered when user clicks or taps a territory.
tgs.OnTerritoryHighlight: triggered when a territory is goint to be highlighted (usually when pointer is over it). A ref parameter allows you to cancel highlight and implement your own coloring system.

tgs.OnPathFindingCrossCell: triggered when path finding algorithm estimates cost to this cell. You can use this event to return extra cost and customize path results.

Handling cells and territories

Please, refer to demo scripts included in demo scenes for example of API usage, like selecting, merging and toggling cells visibility, …