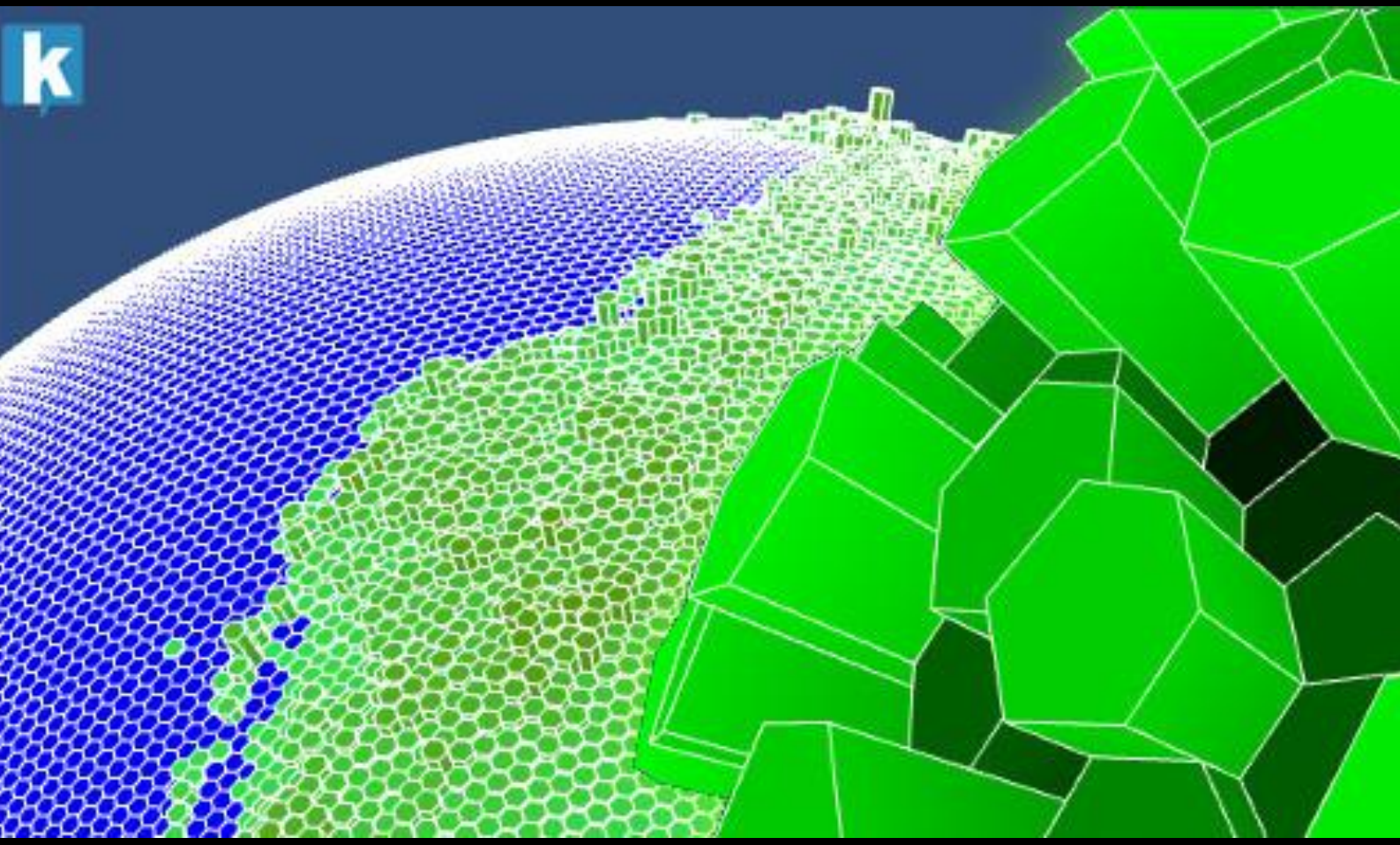
HEXASPHERE GRID SYSTEM



USER GUIDE

Index

Introduction

Thank you for purchasing!

Hexasphere Grid System is a commercial asset for Unity 2019.4 (or later) with the following features:

- Render and interact with hexaspheres
- High-performance, optimized with latest Unity rendering technology
- Wireframe, shaded or combination of styles
- Powerful selectable and highlighting system for tiles.
- Interactive spherical grid: drag to rotate, zoom, fly to desired tiles.
- Coloring and texturing support.
- Native A* Path Finding support.
- Extrusion with customizable color gradient
- Heightmap loading
- Extensive API (C#) for controlling the grid.

You can use this asset for:

- Organizing content or areas of interest over a spherical world
- Provide a generic selectable grid zone without using a terrain
- Allow the user to highlight and select a zone of the world
- Display with different colors regions of the map
- Visualize data around Earth, using extrusion to represent quantities

QuickStart and Demo Scene

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 Hexasphere 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#).

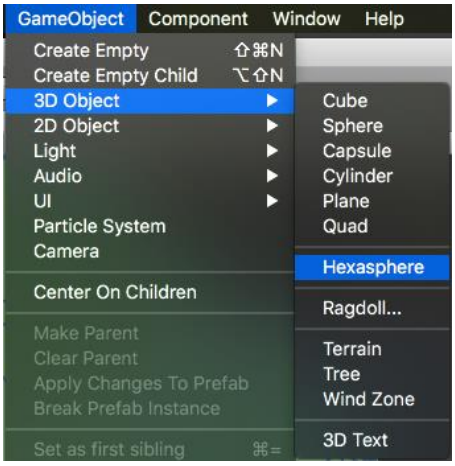
Support & Contact info

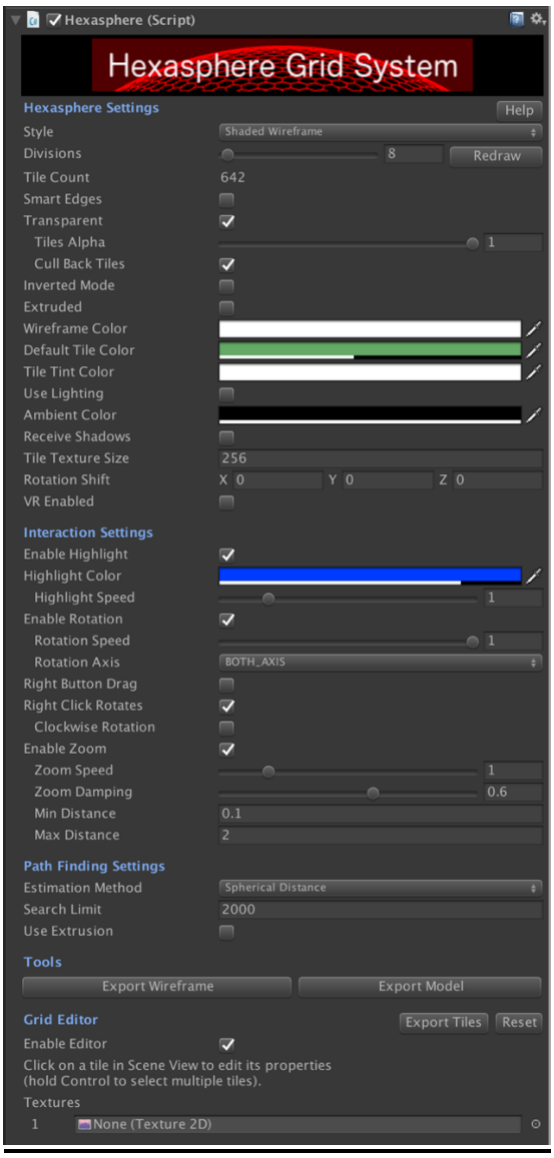
We hope you find the asset easy and fun to use. Feel free to contact us for any enquiry.
Visit our Support Forum for usage tips and access to the latest beta releases.

Kronnect
Email: contact@kronnect.com
Kronnect Support Forum: https://kronnect.com/support

How to use the asset in your project

To add a hexasphere to your scene, select “Hexasphere” from the top menu GameObject -> 3D Object -> Hexasphere:





Custom Editor Properties

Hexasphere Settings: this section allows you to configure main grid appearance settings:

Style: wireframe, shaded or both. Note that when Extruded option is enabled, the wireframe styles use an optimized shader that culls hidden lines providing better performance.

Divisions: the hexasphere derives from a basic icosahedrons. This parameter specifies the number of iterations/divisions. The greater this number, more tiles will be created.

Smart Edges: only available when wireframe is visible. When enabled, edges on adjacent tiles with same materials won’t be rendered.

Inverted Mode: renders the hexasphere inwards, placing the camera at its center.

Extruded: enables “extrusion” of tiles which will activate custom geometry shaders and alternate interaction logic to present a 3d view of tiles. Each tile can have a different extrusion amount (form 0 to 1) which can be set using scripting (see scripting section).

- **Multiplier:** the height of a tile is determined by the extrusion amount x this global multiplier. You can quickly module the height of all tiles using this parameter.

- **Gradient Intensity**: when Extruded, the sides of the tiles will show a gradient color which intensity is controlled by this parameter.

- **Raycast 3D:** enable this option to improve tile highlighting or selection. When using Extrude option, tiles will exceed the sphere radius thus the default collider won't include them. Enabling this option will allow that and produce a more precise selection experience.

- **Wireframe Color:** the color of the lines when using the wireframe style.

- **Default Tile Color:** this is the default material color for each tile unless you change it using scripting.

- **Tile Tint Color:** optiona tintint color applied to all tiles regardless of their assigned colors. Applies to all tiles whereas default tile color only applies to tiles non-colored by user.

- **Use Lighting:** this property controls if the hexasphere cast shadows and also is influenced by the directional light in the scene.

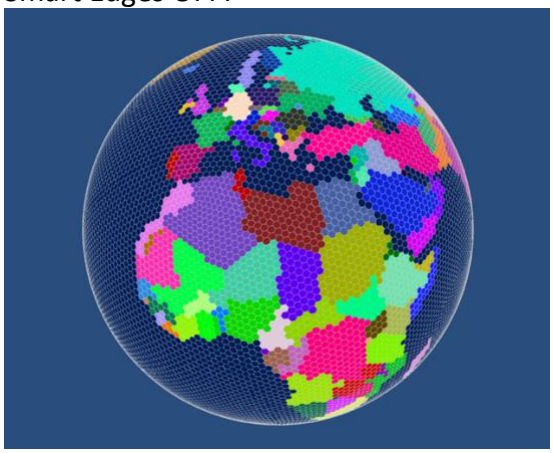
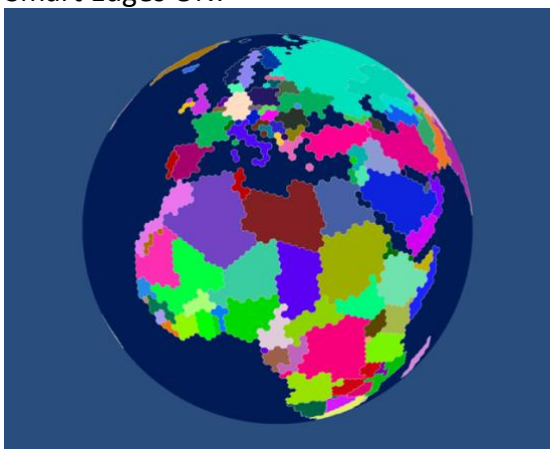
- **Tile Texture Size:** this is the size of the internal texture array used when assigning textures to individual tiles.

- **Tile Texture Stretch:** if true, the uv coordinates of the hexagon vertices will make the texture stretch to the enclosing rectangle. If false, the uv coordinates will match the position of the hexagon vertices as if it's enclosed in a circle. If you get distortion when applying textures to tiles, try disabling this option.

- **Rotation Shift:** allows you to apply an internal rotation to the generated vertices without changing the rotation of the transform. This option can be used to shift the pentagons on other positions.

- **VR Enabled:** enabled raycasting compatible with Virtual Reality (see VR section).

Smart Edges option effect:

Smart Edges OFF:	Smart Edges ON:
	

Interaction Settings

This section controls the behaviour of the hexasphere grid system.

- **Enable Highlight:** activates the tile highlighting when moving the cursor or pointer over the hexasphere grid.
- **Highlight Color & Speed:** color and speed for the highlight effect.
- **Enable Rotation:** allows the user to rotate the hexasphere using mouse drag.
- **Enable Zoom:** allows the user to zoom in/out the hexasphere using mouse wheel (or pinch in/out using touch controls). The min/max values determine the world space distance to the surface of the sphere. Zoom Damping defined the deceleration speed when mouse wheel is released (0 value will stop zoom immediately as the user released the mouse wheel, whereas a value near 1 will produce a deceleration effect until zoom stops).

PathFinding Settings

This section configures the basic options for the path-finding engine.

- **Estimation Method:** the formulae used to compute the estimated distance from a node to the target tile, used by the pathfinding algorithm: spherical distance (uses spherical angle difference between current node and destination tile), Euclidean (uses linear distance between node and destination) and “Euclidean Non SQR” (same than Euclidean but avoid the use of square root which results in a faster algorithm).
- **Search Limit:** the maximum number of tiles in the path.
- **Use Extrusion:** when enabled, the tile height (0..1) multiplied by Extrusion Weight factor will be added to the crossing cost of that tile.

Virtual Reality options

Hexasphere works in VR but in order to use the interaction options you need to enable the "**VR Enabled**" checkbox in the Hexasphere inspector.

Custom Gazes & Pointers (beta)

Google VR Pointer & Controller Touch

To enable tiles selection and interaction using controller touch edit HexaspherePriv.cs (located in Hexasphere/Scripts/Core) and uncomment the line that reads:

```
//#define VR_GOOGLE
```

Samsung Gear VR Pointer

To enable tiles selection and interaction using Samsung Gear VR pointer (laser) edit HexaspherePriv.cs (located in Hexasphere/Scripts/Core) and uncomment the line that reads:

```
//#define VR_SAMSUNG_GEAR_CONTROLLER
```

Programming Guide (API)

Once the Hexasphere object is in the scene, you can access the functionality from code through the static instance property:

```
Using HexasphereGrid;

Hexasphere hexa;

void Start () {
   hexa = Hexasphere.GetInstance ("Hexasphere");
   ...
}
```

Where “Hexasphere” is the name of the gameObject. Then you can use all the API functions and properties. Examples:

```
hexa.style = STYLE.Shaded;
hexa.numDivisions = 20;
hexa.OnTileClick += MyClickHandlerFunction;
```

To get the current highlighted tile:

```
int tileIndex = hexa.lastHighlightedIndex;
if (tileIndex>=0) {
    Debug.Log(“Tile highlighted = “ + tileIndex);
}
```

To customize a tile:

```
hexa.SetTileColor(tileIndex, Color.Red);
hexa.SetTileTexture(tileIndex, myTexture);
```

To make a tile block any path:

```
hexa.SetTileCanCross(tileIndex, false);
```

You can also use SetTileGroup to define different path-finding groups.
Look at the demo scene included in the asset for more code snippets!

Public API structure

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

- Hexasphere.cs: contains generic functions and properties, like grid divisions or extrusion.
- HexasphereTiles.cs: all tile-related functions, like setting tile color or texture.
- HexasphereInteraction.cs: all user interaction functions, like rotation/zoom.
- HexaspherePathFinding.cs: pathfinding APIs.
- HexasphereTools: assorted/misc functions, like heightmap loading.

Note that the above are just filenames, but the class is the same "Hexasphere". 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<Hexasphere>() or just Hexasphere.GetInstance 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 (general properties, interaction, pathfinding, …).

Complete list of public properties, methods and events

General Grid API

GetInstance(name): gets the reference to the hexasphere component attached to a gameobject with name "name".

numDivisions: hexasphere derives from a basic icosahedron and numDivisions specifies the number of iterations (subdivisions). You can set any number although a very high value (>200) will take more processing time and memory.

style: how to render the hexasphere: wireframe only, shaded only or both.

extruded: enables tiles to render with a custom height above the sphere surface. This will also enable advances custom shaders to optimize the rendering.

raycast3D: enables a more precise tile raycasting function. Useful when extruded option is enabled.

tileTextureSize: size of the internal texture array used to render textured tiles (if textures are assigned to tiles). Defaults to 256.

extrudeMultiplier: multiplier or "roof" for the extruded tiles. An extruded tile will have a relative height of 0..1. This value is multiplied by "extrudeMultiplier" and the result is relative to the scale of the sphere.

wireframeColor: the color for the lines of the wireframe.

tiles: an array with all generated tiles.

Individual Tile API

GetTileIndex(tile): given a Tile object, this function returns its index in the tiles array.

SetTileMaterial (tileIndex, material, temporary): assigns a material to a given tile. The temporary parameter specifies if the material is not permanent (useful for quick effects, like highlighting tiles or visual effects – defaults to false). If temporary is set to true, a overlay hexagon will be added to the geometry with the given material to prevent updating the whole grid.

SetTileColor(tileIndex, color, temporary): like SetTileMaterial but it only assigns a color. An internal cache of materials is used to optimize draw count.

GetTileColor(tileIndex, ignoreTemporary): returns current tile color. If a temporary color has been assigned, this function will return that color unless ignoreTemporary argument is set to true. Otherwise the assigned color or default color will be returned.

SetTileTexture(tileIndex, texture, tint, temporary): like SetTileMaterial but assigns a texture and optional tint color. An internal cache of materials is used to optimize draw count.

SetTileTextureRotation(tileIndex, float radians): sets the tile texture rotation in radians.

SetTileTextureRotationNorth(tileIndex): aligns tile texture rotation to North.

GetTileTextureRotation(tileIndex): returns the tile texture rotation in radians.

ClearTile(tileIndex, clearTemporaryColor, clearAllColors, clearObstacles): resets one or more attributes of a tile: clearTemporaryColor just removes a previous temporary assigned color; clearAllColors removes any assigned color and returns the tile color to the default value; clearObstacles resets tile canCross flag to true.

ClearTiles(clearTemporaryColors, clearAllColors, clearObstacles): same than before but affect all tiles.

SetTileCanCross(tileIndex, canCross): sets the "canCross" Boolean flag of a given tile used in pathfinding functions.

GetTileCanCross(tileIndex): returns tile canCross boolean flag value.

SetTileGroup(tileIndex, newGroup): assigns a new group to a given tile. Groups are useful to control which tiles can be crossed with FindPath. A group is defined by an integer value that act like a bitwise layer mask. For instance, you can assign a tile the group value 5 (bits 1 and 4). Later you can use FindPath and pass a bitwise mask which is compared to the group value to consider those tiles or not (see FindPath method).

GetTileGroup(tileIndex, newGroup): gets the current group of a given tile (default = 1).

Set/GetTileCrossCost(tileIndex, newCost): sets or gets the individual pathfinding cost for any given tile.

SetTileExtrudeAmount(tileIndex, extrudeAmount): sets the extrude amount or height for a tile (0..1). Note that "Extrude" property of hexasphere must be true before assigning heights to tiles.

GetTileExtrudeAmount(tileIndex): returns the tile's height or extrude amount.

SetTileVertexElevation (tileIndex, vertexIndex, elevation): sets the elevation for an individual vertex. Note that setting different elevations for vertices will distort the hexagon. Usually, SetTileExtrudeAmount is preferred since it affects the entire hexagon. However SetTileVertexElevation can produce soft slopes between adjacent tiles which cannot be produced by SetTileExtrudeAmount.

GetTileVertexElevation(tileIndex, vertexIndex): returns the tile vertex elevation previously set by SetTileVertexElevation (defaults to 0).

GetTileNeighbours(tileIndex): gets all neighbours indices of a given tile.

GetTileNeighboursTiles(tileIndex): gets all neighbours (tile objects, not indices) of a given tile.

GetTileAtPolarOpposite(tileIndex): gets the tile index at the exact opposite pole position.

GetTileAtUV(tileIndex): gets the tile index that would map against a uv coordinate.

GetTilesWithinTwoTiles(Vector2 uv): returns the tile index corresponding to a given texture position wrapped around the hexasphere.

GetTilesWithinDistance(tileIndex, distance, worldSpace): returns the tile indices of tiles near a given tile within a distance in local space (0..1 range) or in world space units.

GetTilesWithinDistance (tileIndex, maxSteps, worldSpace, criteria): same than **GetTilesWithinDistance** but will call a custom criteria function for each potential tile. The criteria function will receive the tile index and must return a Boolean specifying if the tile is valid or not.

GetTilesWithinSteps(tileIndex, maxSteps): returns the tile indices of tiles near a given tile within a number of steps. This function uses internally the path finding functions to ensure the tiles are reachable.

GetTilesWithinSteps(tileIndex, maxSteps, criteria): same than GetTilesWithinSteps but will call a custom criteria function for each potential tile. The criteria function will receive the tile index and must return a Boolean specifying if the tile is valid or not.

GetTilesWithinSteps(tileIndex, minSteps, maxSteps): returns the tile indices of tiles near a given tile within a number of steps. This function uses internally the path finding functions to ensure the tiles are reachable.

GetTilesWithinSteps(tileIndex, minSteps, maxSteps, criteria): returns the tile indices of tiles near a given tile within a number of steps. This function uses internally the path finding functions to ensure the tiles are reachable.

DestroyTile(tileIndex): clears all tile colors and also destroys any cached geometry created when assigning a temporary color to the tile.

ToggleTile(tileIndex, visible): toggles on/off the temporary color.

HideTile(tileIndex): hides the temporary color assigned to the tile.

ShowTile(tileIndex): shows the temporary color assigned to the tile.

GetTileCenter(tileIndex, …): returns the tile's center in local or world space coordinates with extrusion option.

GetTileVertexPosition(tileIndex, vertexIndex): returns the tile's vertex in local or world space coordinates with extrusion option.

GetTileAtPosition(position, worldSpace?): returns the tile index found under the given position, in world space or local space (default to world space).

GetTileAtLocalPosition(localPosition): returns the tile index found under the given local space position.

GetTileLatLon(tileIndex): returns the latitude and longitude of the tile center in a Vector2 field.

GetTileUV(tileIndex): returns the texture coordinates of the tile center mapped to a rectangular texture in a Vector2 field.

GetTileVertexLatLon(tileIndex, vertexIndex): returns the latitude and longitude of a tile vertex in a Vector2 field.

IsTileVisibleFromCamera(tileIndex, camera): returns true if a given tile is visible from a camera.

Interaction API

rotationEnabled: enables/disables user-drag/rotation on the sphere grid.

rotationSpeed: custom rotation speed. Defaults to 1.

rightClickRotates: enables/disables rotation of sphere by pressing right mouse button.

rightClickRotatesClockwise: direction of the rotation.

zoomEnabled: enables/disables user zoom using the mouse wheel or pinch in/out.

zoomSpeed: custom zoom speed. Defaults to 1.

zoomMinDistance / zoomMaxDistance: min/max distance to the sphere surface.

FlyTo(tileIndex, duration): makes the grid rotate until designated tile is centered on the screen. The rotation lasts for "duration" seconds.

FlyTo(position, duration): makes the grid rotate until target. The rotation lasts for "duration" seconds. Position is given in local spherical coordinates (eg. the tile center).

PathFinding API

pathFindingHeuristicFormula: the formulae used to estimate distance from current node to destination tile.

pathFindingSearchLimit: maximum number of tiles in the path.

FindPath(tileIndexStart, tileIndexEnd, searchLimit, groupMask, ignoreTileCanCross): returns the list of tile indices from starting tile to destination tile. Optionally pass a maximum number of tiles (defaults to pathFindingSearchLimit setting) and a groupMask which will be compared to the group field of each tile to determine if that tile is suitable for this path.
The groupmask is a bitwise mask. For instance, if you have two groups of tiles, one group is assigned the value 1 and another is assigned the value 2, then you can call FindPath with a groupMask value of 3 (bits 1 + 2) and it will consider both groups. But if you pass a groupMask value of 1 or 2 then it will only consider one of the two groups (group 1 or group 2).
Use ignoreTileCanCross to override the canCross field of each tile.

To set custom tile weighs and per-tile costs use: **SetTileGroup**, **SetTileCanCross** and **SetTileCrossCost** methods as well as **OnPathFindingCrossTile** event which you may use to provide on-the-fly custom costs as path is being calculated.

Highlighting API

highlightEnabled: enables tile highlight/selection by user using the mouse.

highlightColor: color for the highlighting effect.

highlightSpeed: speed for the flashing loop used in the highlighting effect.

lastHighlightedTileIndex: the index of the currently highlighted tile in the “tiles” array.

lastHighlightedTile: the Tile object currently being highlighted.

isMouseOver: returns true if mouse is over the spherical grid.

Tools/Misc API

ApplyHeightMap(texture2D, seaLevel, rampColors): maps the texture to the sphere using a spherical projection and assigns a relative height/extrude amount based on the red channel of the pixel color (0..1). the rampColors parameter is a texture with gradient colors used to distribute “height colors” among the values, where 0 is the leftmost pixel and 1 is the rightmost pixel. The seaLevel parameter is used to clamp low values so everything under this value is treated as zero (0).

ApplyHeightMap(texture2D, waterMask, rampColors): maps the texture to the sphere using a spherical projection and assigns a relative height/extrude amount based on the red channel of the pixel color (0..1). the rampColors parameter is a texture with gradient colors used to distribute “height colors” among the values, where 0 is the leftmost pixel and 1 is the rightmost pixel. The waterMask texture is used to define water areas (alpha value is used to determine water positions and it’s must be below 16 in a 255 scale).

ApplyColors(texture2D): maps the texture to the sphere using a spherical projection.

ParentAndAlignToTile(GameObject, tileIndex, altitude, snapRotationToVertex0, adjustScale, scaleRatio): changes gameobject transform so:

- Parent it to the hexasphere and positions the gameobject onto specific tile (tileIndex).
- Optionally offsets the sprite from the hexasphere by a given altitude.
- Optionally makes the sprite rotation face to vertex 0 (otherwise it faces to North).
- Optionally adjusts scale of the sprite to fit within tile boundaries.

Raycast(ray, out hitPoint): returns true (and the hitPoint) if a given ray intersects with the hexasphere (compatible with extrusion).

GetExtrudedPosition(position, worldSpace): adjusts the position to the surface of the hexasphere, including the extrusion applied at that position. The worldSpace parameter specifies if the position is given in world or local space (and it’s returned accordingly).

Events

OnPathFindingCrossTile(tileIndex): if used, this event will be triggered each time a tile is computed. A custom cost should be returned assuming a default cost of 1 for any tile. You can use this event to increase the cost of crossing a number of tiles returning a higher value.

OnTileClick(tileIndex): if used, this event will be triggered each time a tile is clicked.

OnTileMouseOver(tileIndex): if used, this event will be triggered each time the mouse hovers a new tile.

OnFlyStart(): triggered when a FlyTo() operation starts.

OnFlyEnd(): triggered when a FlyTo() operation ends.

OnDragStart(): triggered when user starts a drag gesture.

OnDragEnd(): triggered when a drag gesture ends.

OnZoom(): triggered when user zooms in/out.

Please, refer to demo script included in demo scene for code example of API usage.