

# Table of Contents

[Intro](#)

[Articles](#)

[Intro](#)

[Tutorials](#)

[Getting Started](#)

[Big Tiles](#)

[3D Layouts](#)

[Path Constraints](#)

[Generating on Mesh Surfaces](#)

[Multiple Passes](#)

[Constraints](#)

[Count Constraint](#)

[Mirror Constraint](#)

[Path Constraint](#)

[Separation Constraint](#)

[Border Constraint](#)

[Extras](#)

[Animation](#)

[Infinite Generator](#)

[Bolt Support](#)

[Samples](#)

[Palettes](#)

[Cell Types](#)

[Overlapping Model](#)

[Controlling Output](#)

[Quality and Performance](#)

[Mouse and Keyboard Shortcuts](#)

[Using the API](#)

[Customization](#)

[Upgrading to Tessera Pro](#)

[Release Notes](#)

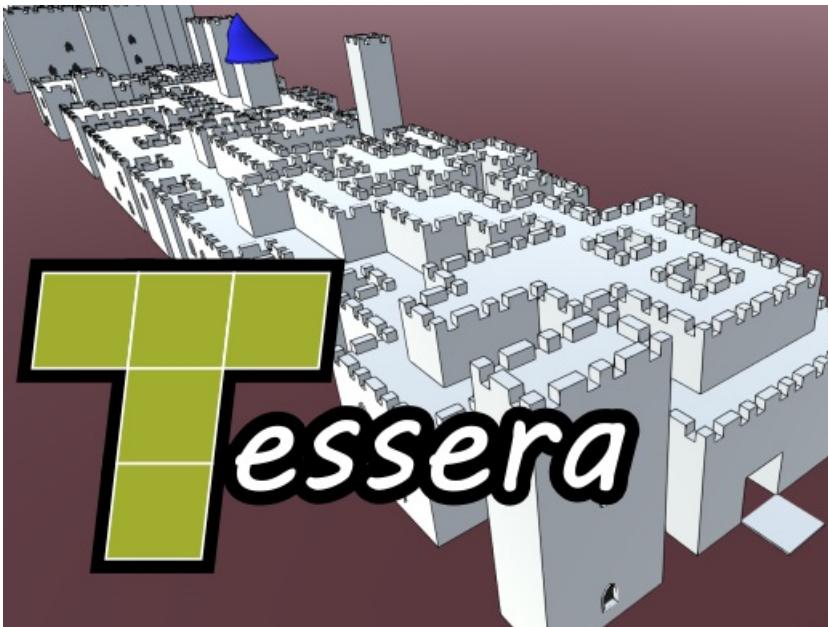
[Api Documentation](#)

[Tessera](#)

AnimatedGenerator  
AnimatedGenerator.AnimatedGeneratorState  
BiMap<U, V>  
BorderConstraint  
BorderItem  
CellFaceDir  
CellRotation  
ChunkCleanupType  
CountConstraint  
CubeCellType  
CubeFaceDir  
CubeFaceDirExtensions  
CubeRotation  
DefaultCellTypeImpl  
DefaultGridImpl  
EnumeratorWithResult<T>  
FaceDetails  
FailureMode  
GridSymmetry  
HexGeometryUtils  
HexPrismCellType  
HexPrismFaceDir  
HexPrismFaceDirExtensions  
HexRotation  
ICellType  
IEngineInterface  
IGrid  
InfiniteGenerator  
InstantiateOutput  
ITesseraInitialConstraint  
ITesseraTileOutput  
MeshData  
MeshDeformation  
MeshUtils  
MirrorConstraint  
MirrorConstraint.Axis  
ModelTile

ModelType  
OrientedFace  
PaletteEntry  
PathConstraint  
PinType  
PrefixLookup<T>  
QuadInterpolation  
RotationGroupType  
SeparationConstraint  
SquareCellType  
SquareFaceDir  
SquareFaceDirExtensions  
SquareRotation  
TesseraCompletion  
TesseraConstraint  
TesseraGenerateOptions  
TesseraGenerator  
TesseraHexTile  
TesseralInitialConstraint  
TesseralInitialConstraintBuilder  
TesseralInstantiateOutput  
TesseraMeshOutput  
TesseraMultipassGenerator  
TesseraMultipassPass  
TesseraMultipassPassType  
TesseraPalette  
TesseraPinConstraint  
TesseraPinned  
TesseraSquareTile  
TesseraTile  
TesseraTileBase  
TesseraTileInstance  
TesseraTilemapOutput  
TesseraTransformedTile  
TesseraTrianglePrismTile  
TesseraVolume  
TesseraVolumeFilter

[TesseraWfcAlgorithm](#)  
[TileEntry](#)  
[TileList](#)  
[TileMapping](#)  
[TriangleInterpolation](#)  
[TrianglePrismCellType](#)  
[TrianglePrismFaceDir](#)  
[TrianglePrismFaceDirExtensions](#)  
[TrianglePrismGeometryUtils](#)  
[TriangleRotation](#)  
[TRS](#)  
[UnityEngineInterface](#)  
[VolumeType](#)



**Tessera** is a Unity addon for procedurally generating 3d tile-based levels and builds. [Get it here.](#)

---

Tessera works by running the [Wave Function Collapse](#) algorithm. This algorithm is a powerful technique for generating maps from complicated tile sets with tight controls on behaviour.

For help, use [Discord](#) or post on the [Unity forums](#).

To get started with Tessera 5.2.0, see the [introduction](#).

# Introduction

**Tessera** is a Unity addon for procedurally generating 3d tile-based levels and builds.

Tessera works by running the [Wave Function Collapse](#) algorithm. This algorithm is a powerful technique for generating maps from complicated tile sets with tight controls on behaviour.

Tessera is available on the Unity asset store as two different packages. The [basic version](#) includes the fundamentals of generations and key features, while the [Pro version](#) contains more advanced features. These tutorials cover both, with pro-only features clearly marked.

Tessera has a lot of features. Try the following links for tutorials and an overview of what is available.

- [Getting Started Written Tutorial](#)
- [Getting Started Video Tutorial](#)
- [List of Samples](#)
- [Autogenerated API Reference](#)

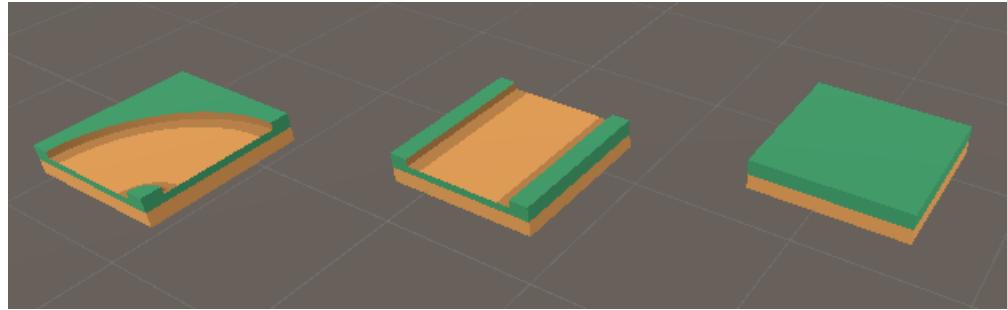
# Getting Started Tutorial

## Setup

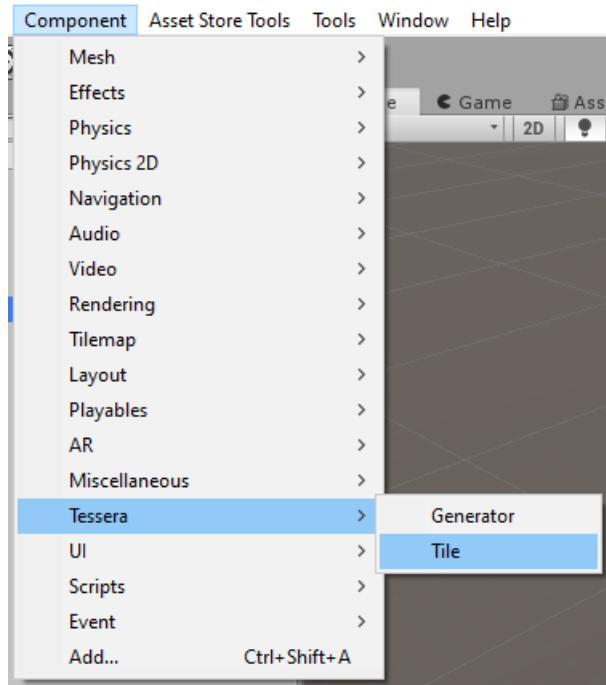
Start a new project. Then download [Tessera from the Unity Asset Store](#) and import it to your unity project. For this tutorial, we'll also use use [Kenney's Tower Defense Kit](#) so either download that or use the files included in `Samples/GrassPaths/Models`.

## Creating tiles

Lets create some tiles. From the tower defense assets, drag the prefabs for `tile`, `tile_cornerRound` and `tile_straight`. These tiles are a small selection of grass and path tiles.



Then, with those game objects selected, add the `TesseraTile` component from the menu.



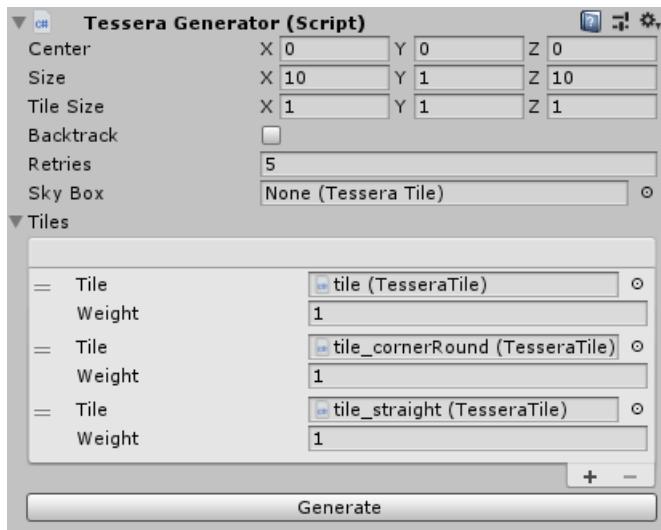
That's all for now, we'll configure the tiles later.

## Creating the generator

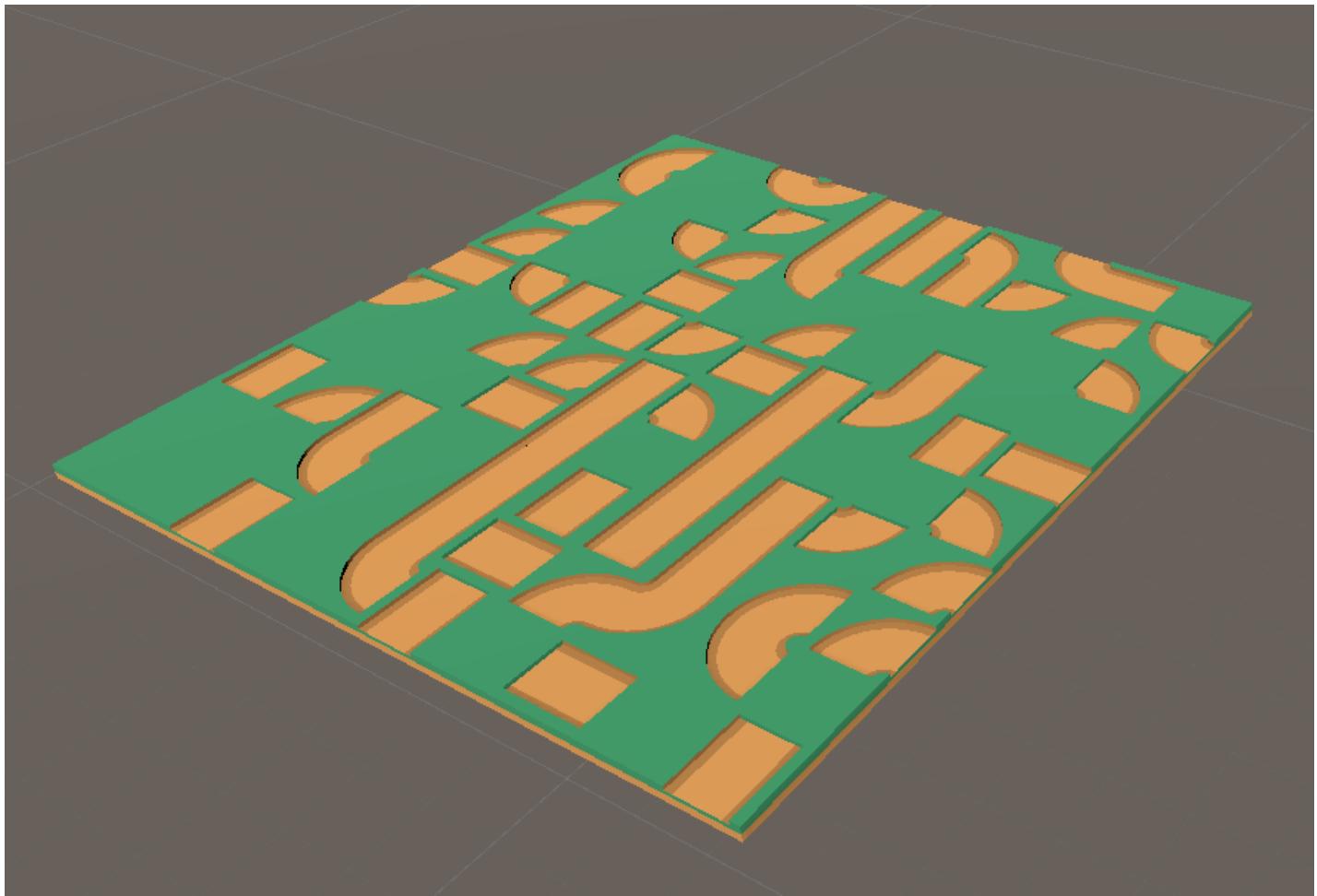
Next, create a new empty GameObject, and give it the `TesseraGenerator` component from the menu. Bring it up in the inspector. Add the tiles we created before to the list of tiles, either by dragging them from the hierarchy onto the Tiles section, or clicking the small plus button and selecting each tile.

Position the generator so that it does not overlap the tiles you created.

Afterwards, your configuration should look like this:



Now press the "Generate" button to create a new arrangement of those three tiles. You should get something looking like this:



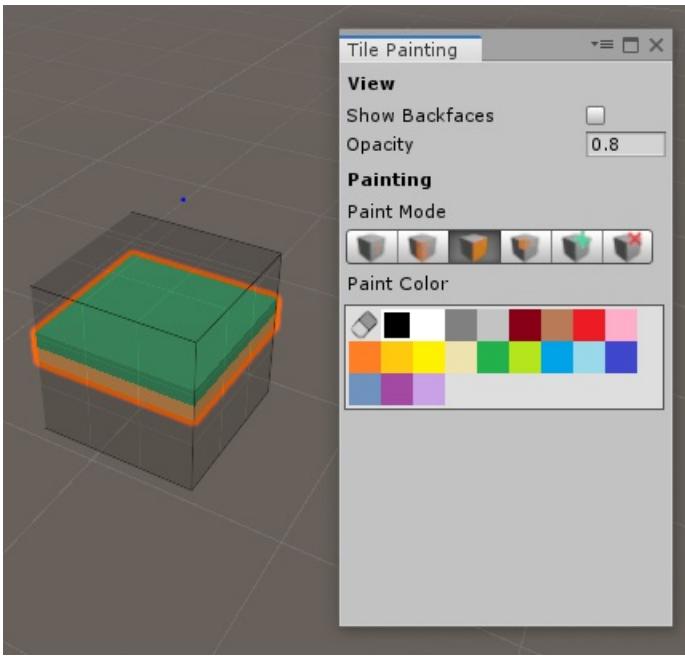
It's generated the tiles, but right now it doesn't know which tiles can be placed next to which other ones. So it has just placed them randomly. Hit undo to delete the created tiles.

To fix this, we need to paint the tiles.

## Tile painting

Select the first tile, called `tile`. In the inspector, click the "paint faces" button.

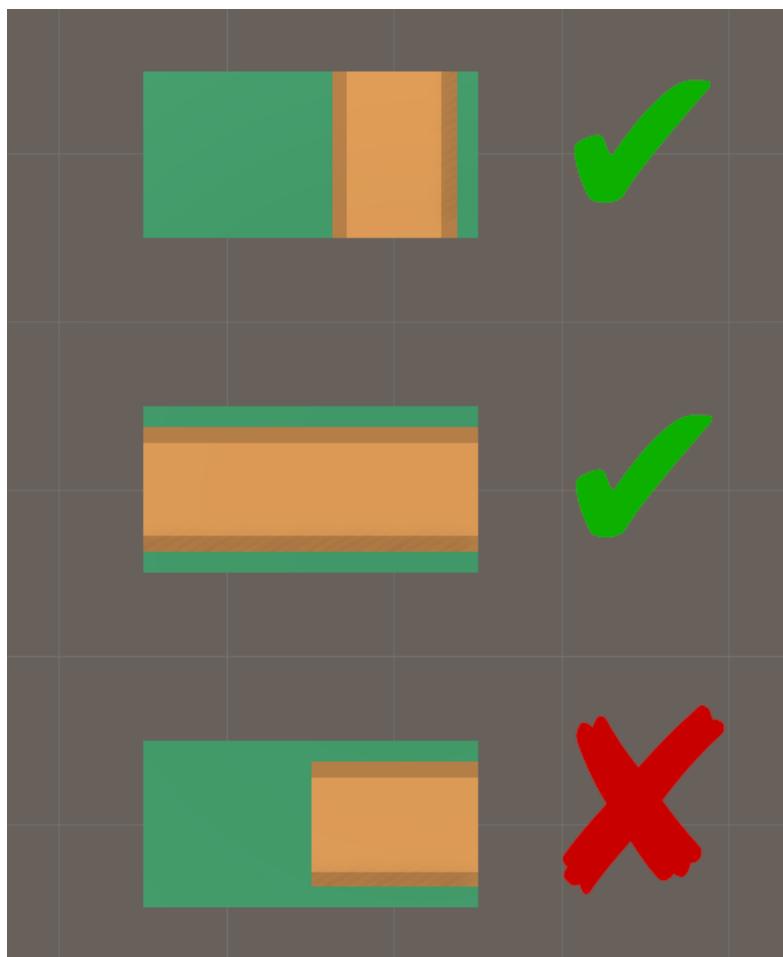
This should pop-up a "Tile Painting" window, and also show a semi-transparent cube around the tile.



You can use these tools to paint different colors onto the tile's cube. First, select a paint color from the palette, the click on the cube to apply that color. If you make a mistake, select the eraser from the palette and clear what has been painted.

**Tiles can only connect to each other if they have matching colors painted on their corresponding faces.** Specifically, each face is divided subdivided into 9 squares. A pair of adjacent tiles are compared by pairing up the squares on the opposing faces, and seeing if they match. Squares match if they are both the same color, or if either square is transparent, though this can be customized with a [Palette](#).

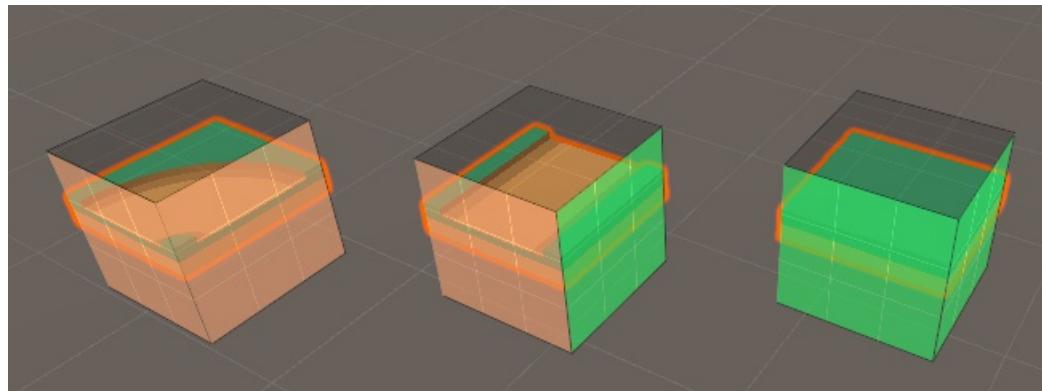
In this case, we want tiles to connect to each other if they are both grassy, or if they are both a path, but do not want paths to lead straight into grass.



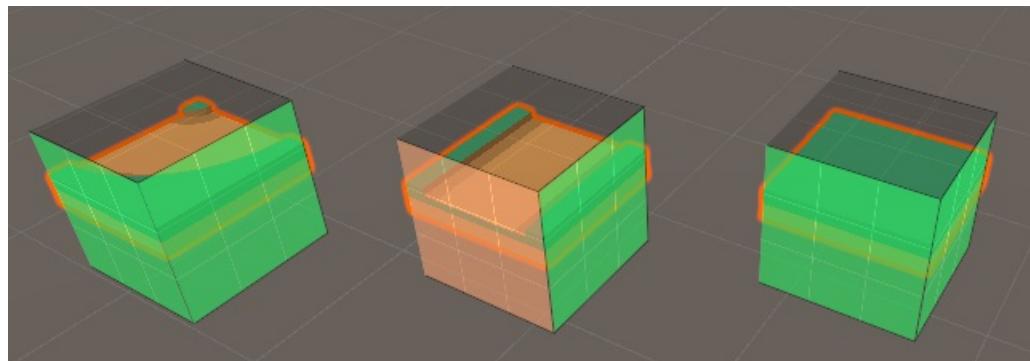
In order to achieve this, we will paint all the grassy faces of each tile green, and all the faces with paths brown. The top and bottom we will leave alone. That will connect grass to grass, paths to paths, and disallow grass connecting to paths.

Paint all 4 sides of the 3 tiles now. Afterwards, you should have three tiles that look like this.

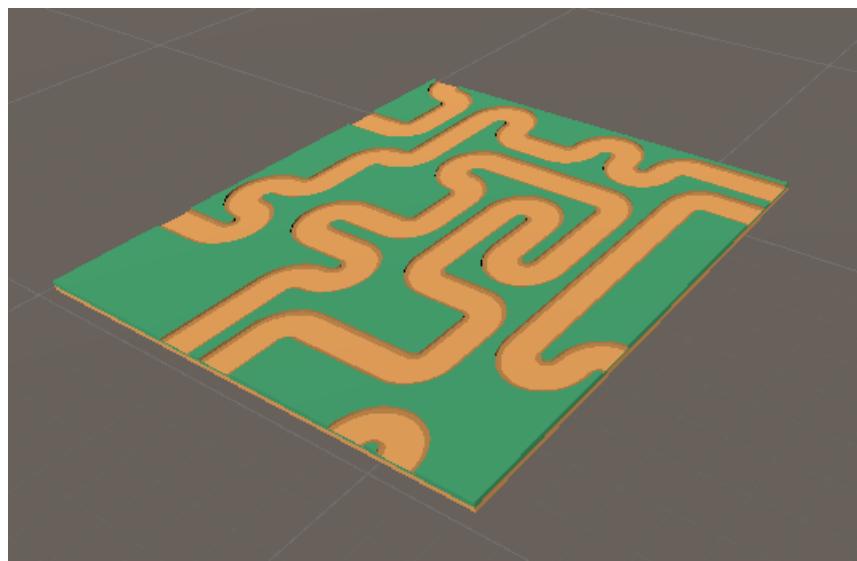
Front view:



Rear view:



Now we can go to the generator and press the "Generate" button again. Make sure you have deleted the tiles it created the first time around as it won't overwrite already generated tiles. If everything is set up correctly, it should look like this.



This concludes the tutorial. From here you can experiment with some of the settings in the inspector, try adding more tiles from the tower defense assets, or read the more advanced tutorials.

# Big Tiles Tutorial

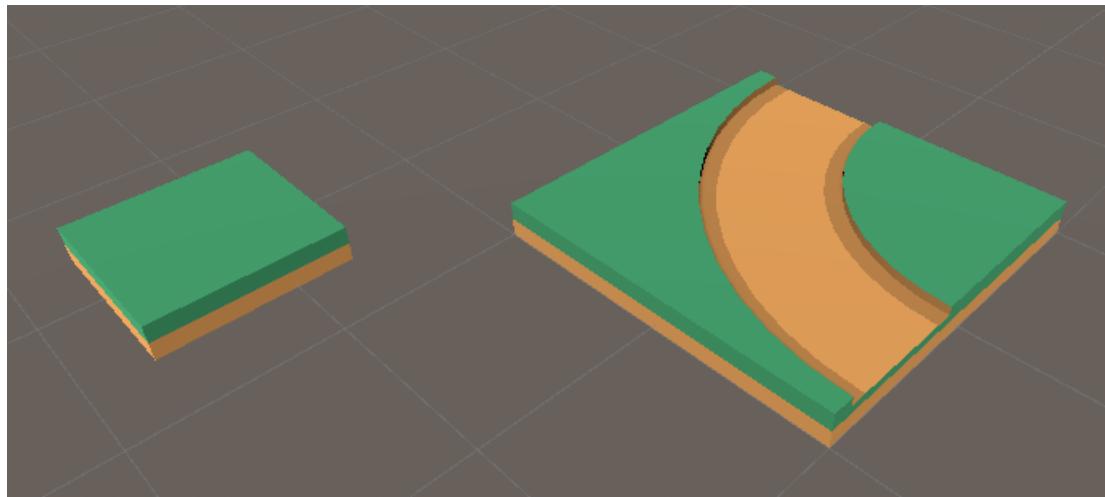
This tutorial continues from the [Getting Started](#) tutorial. It is recommended you complete that one before starting this.

So far we've looked at generating game objects that all have the same size. That is convenient, but the clear grid structure is not always desired. Here we look at one way of addressing that. If normal tiles only occupy one cell in the output, the big tiles can straddle several. That means you can design a larger set piece cohesively.

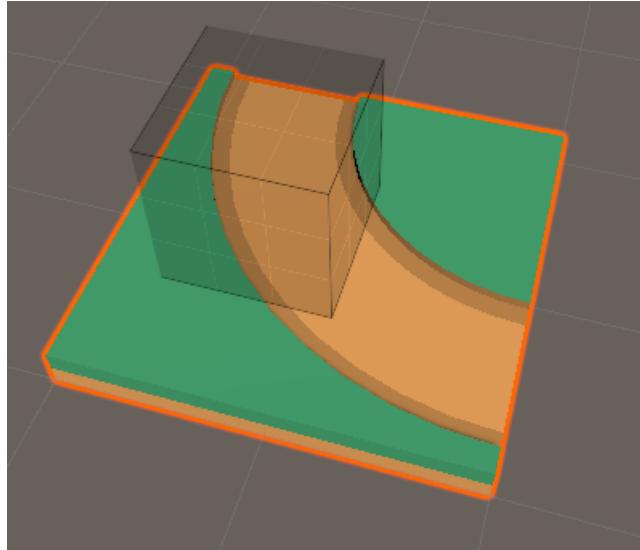
## NOTE

Big Tiles should only be used for tiles that are larger than your grid's tile size. If you are creating *every* tile using Big Tiles, you're better off just changing the grid.

Let's add a new tile from Kenney's Tower Defense Kit. This time, pick `tile_cornerLarge`. It is twice as big as a regular tile in each dimension.



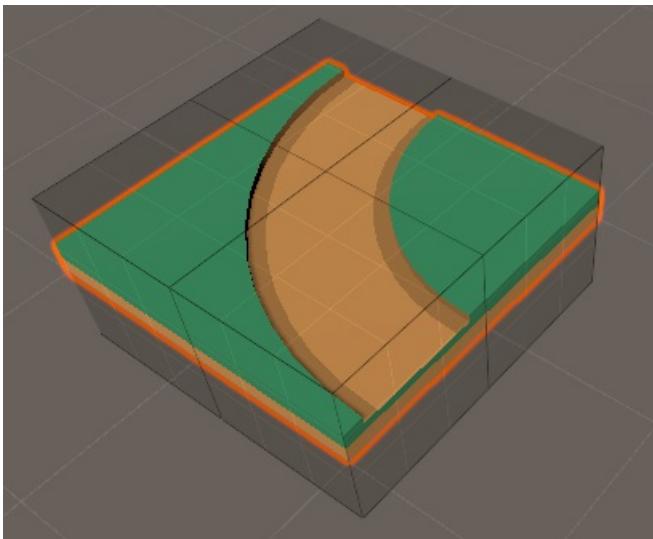
Lets set it up. As before, add the TesseraTile component. Then set the Center to (-0.5, 0, -0.5). This will place the paintable cube in one corner of the tile.



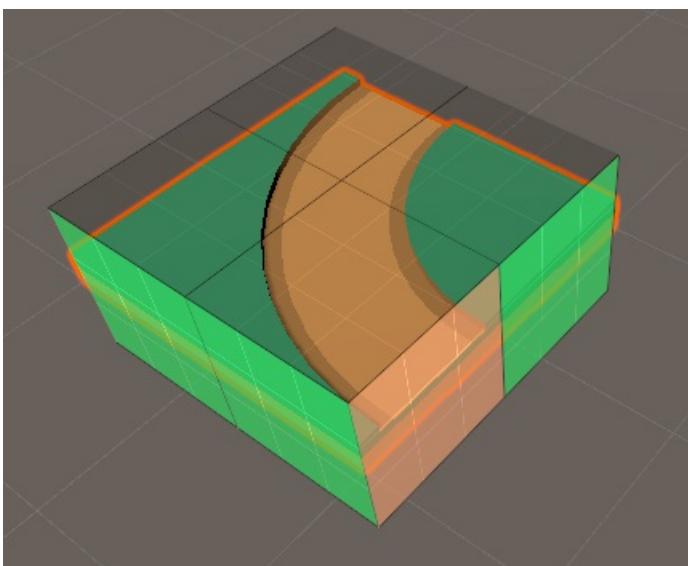
Then select the "Add Cube" tool from the paint menu. You can now click on faces of the paintable cube to make a tile with multiple cubes in it. If you make a mistake, you can use the "Remove Cube" tool to delete them.

*Note: Big Tiles should have the same Tile Size as other tiles. You must use the Add Cube tool to make big tiles*

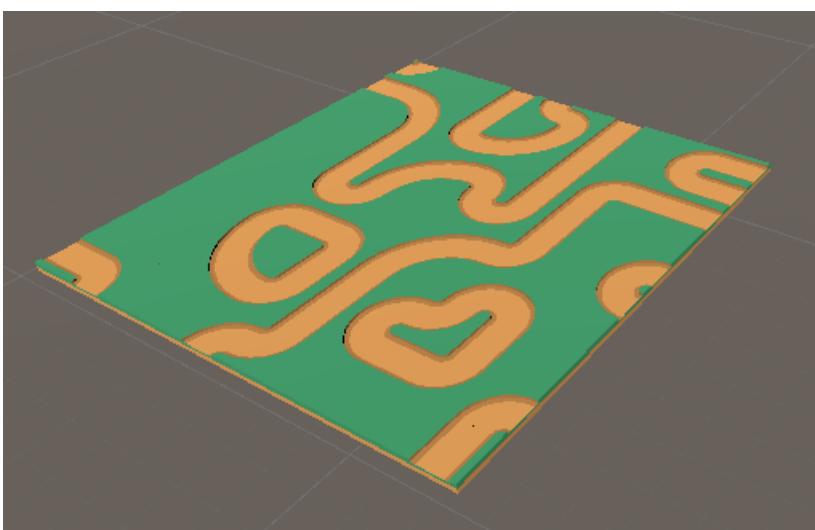
Add 3 extra cubes to the tile to cover it. This tile will now take up as much space as 4 regular tiles.



Paint the sides of the new tile green and brown, like the original tiles, to indicate how it connects.



Now we're ready to add this cube to the list of tiles in the generator, and hit Generate. The new tile will be seamlessly mixed with the smaller tiles.



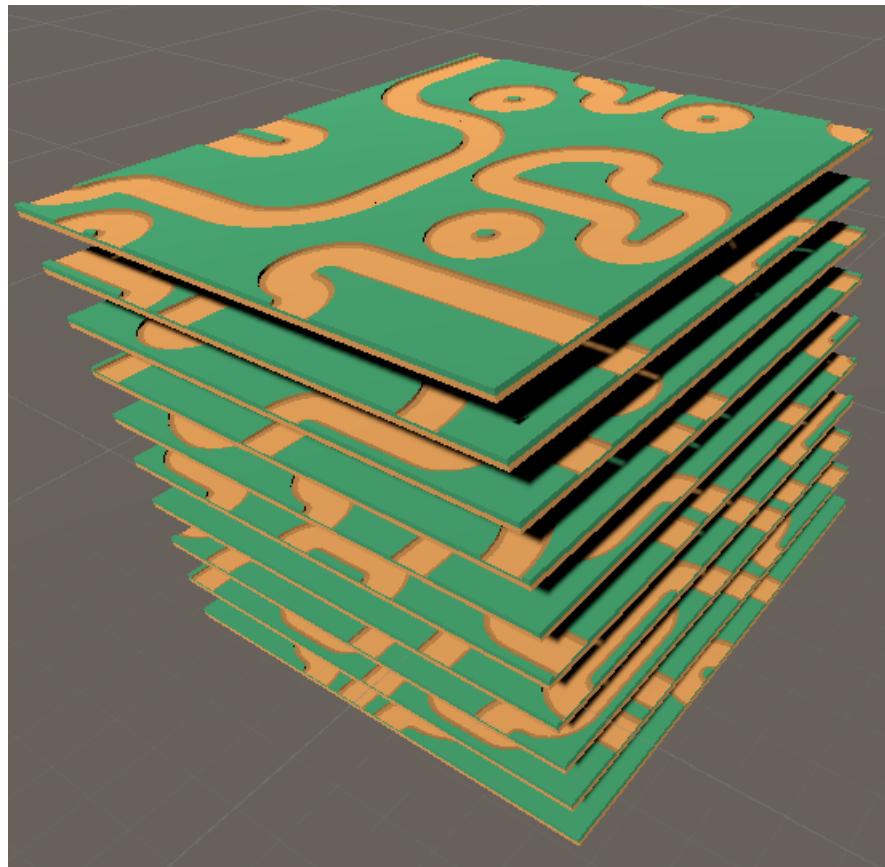
# Generating 3d layouts Tutorial

This tutorial continues from the [Big Tiles](#) tutorial. It is recommended you complete that one before starting this.

## Earth and air

So far we've only generated a single layer of tiles. But Tessera can work with 3d grids too. The principle is exactly the same - paint tiles to indicate how they connect and let Tessera do the rest.

Let's turn our previous example into a 3d example. First, go to the generator, and find the size setting and increase the Y size to 10. Now if we hit generate, we get something like the following:



There's two things to note:

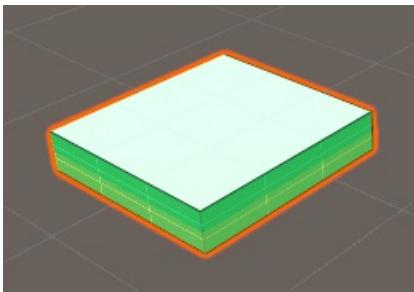
- Each layer is separated from the one above by 1 unit, even though our tiles aren't nearly that high.
- Just like in the first tutorial, Tessera doesn't know how things connect. We need to paint the tiles to indicate what can be put on top of what.

Let's fix those issues.

First, change the Tile Size property in the generator to (1, 0.2, 1). This is the size of the meshes we are using. Now do the same thing for the tiles. They should also have their center Y set to 0.1. All new tiles will want to share these same settings.

Second, let's paint tops and bottoms of the tiles. Paint the top faces of each tile as white, and the bottom ones as black. This will indicate above ground / below ground respectively.

*NB: You can use "Show Backfaces" to easily see the far sides of cubes so you don't need rotate all the time to paint everything.*

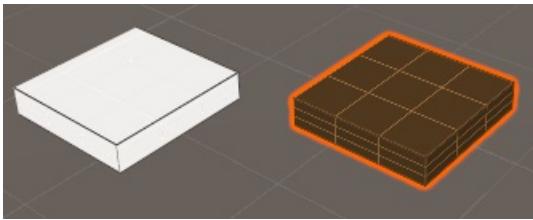


If we run the generation now, it will fail.

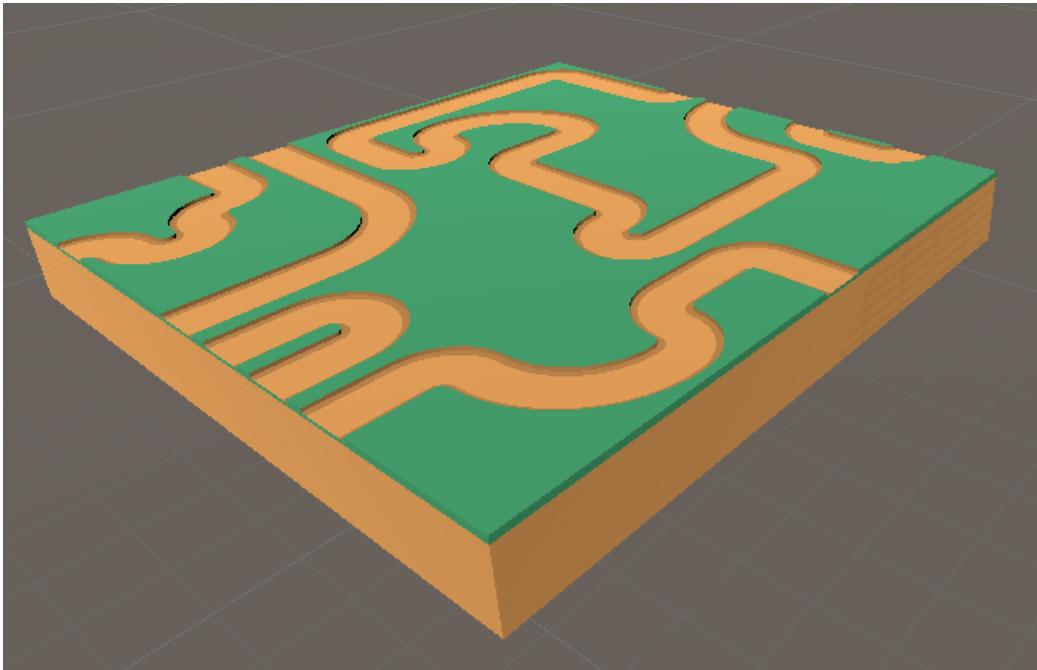
Failed to complete generation

This is because we've colored the tiles so that they no longer stack, but we haven't provided anything to stack above or below them. Tessera tries to fill the *entire* generator bounds with tiles, and fails if it cannot do so. So we need some more tiles.

Create an empty called `tile_air` and from the assets load `tile_dirtHigh`. Give both of these the TesseraTile component, add them to the Generator's tile list, and set their Tile Size and Center as with the other tiles. Now paint them. `tile_air` should have all 6 faces painted white, and `tile_dirtHigh` should have all 6 faces painted black.



Now generation should be working again, and you'll get a 3d, if flat, landscape.

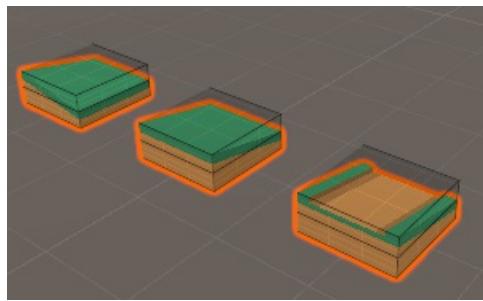


*NB: You may notice that instantiating this many tiles takes quite a bit of time. This can be undesirable for a game. If you select the air and dirt tile, and enable "Instantiate Children Only" then they will no longer be instantiated (as they have no children), speeding things up considerably.*

## Adding slopes

We need to add even more tiles before the surface can crinkle. Let's take 3 more from the tower defense assets: `tile_slope`, `tile_cornerOuter` and `tile_straightHill`. Again, TesseraTile component, Tile Size (1, 0.2, 1), Center (0, 0.1, 0).

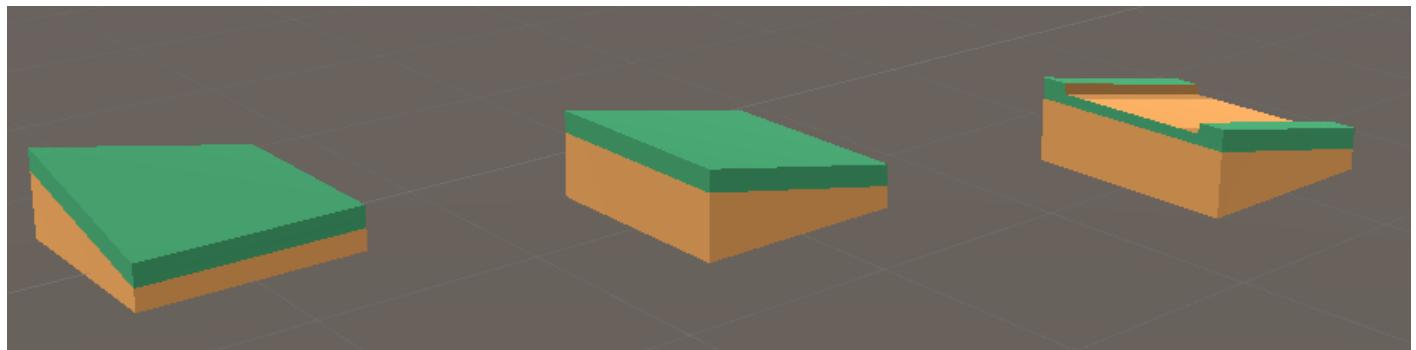
You'll notice that the tile meshes poke out the top of the paint cube. Use Add Cube to stack a second cube on top of the first so that the meshes are completely contained within. See the [big tiles tutorial](#) for details.



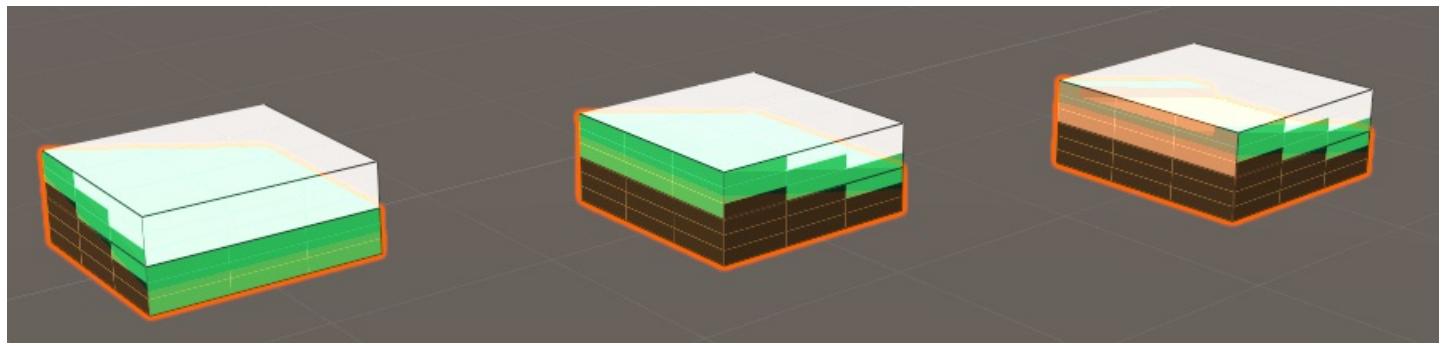
Now, we need to paint the cubes. We need to be careful when painting them. We want to ensure that all our existing tiles connect to these new tiles at the correct places. And we need to make sure that the sloped sides can connect only to each other.

Rather than give the sloped sides an extra color, we will paint them with a recognizable pattern. That will serve as a reminder. Sometimes we'll paint the pattern, and sometimes the reflection, indicating which direction the slope is running. Tessera will recognize this, and connect them appropriately. You can use the "Pencil" tool  to paint patterns.

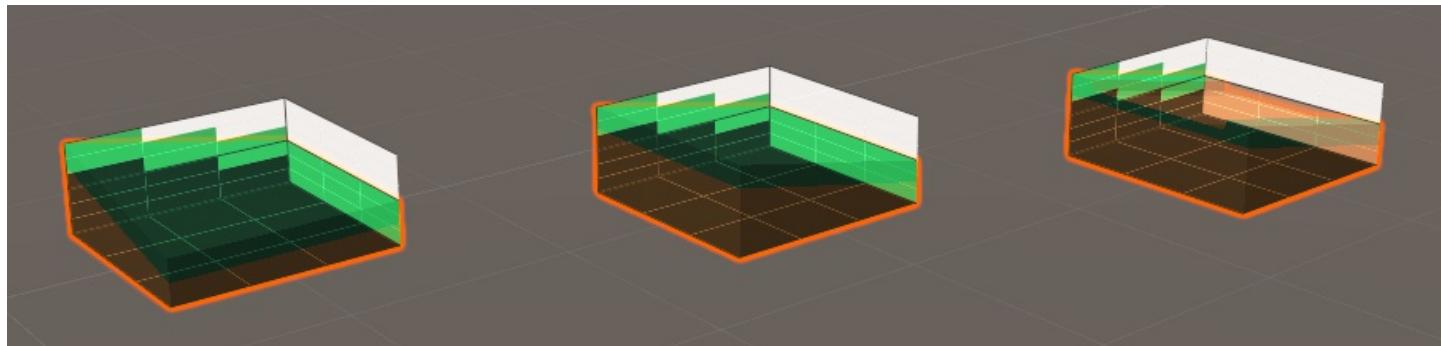
Paint the tiles. Before:



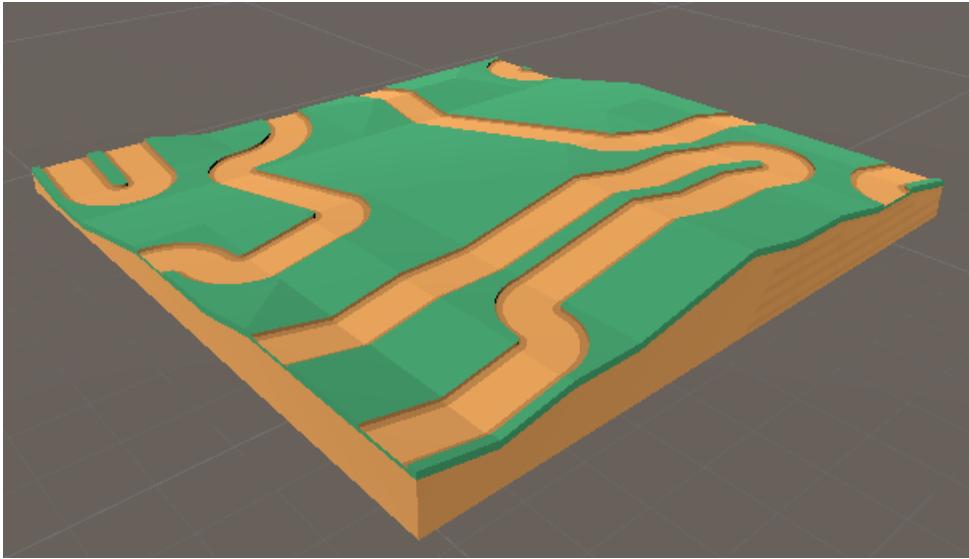
After painting:



After painting, showing backfaces:



Add the new tiles to the generator, and you should be rewarded with a undulating landscape:



## Adding a sky box

You may have noticed that sometimes the generator fills the entire volume with nothing but dirt, or nothing but air. There's nothing in what we've generated so far that prevents that. The generation algorithm will often surprise you out like that - anything that is a legal arrangement will occasionally be generated, and legal arrangements aren't always what you intended. One easy fix for this is adding a skybox to the generator. A skybox constrains what tiles can be placed on the boundary of the generated volume.

In this case, we want to force the top of the area to be air, and the bottom to be dirt, and we don't care about the sides. That will force there to be a surface somewhere between the top and bottom. Create an empty with the TesseraTile component, and paint the top face white and the bottom face black. Then assign it to the Skybox property of the generator. This will fix things as desired.

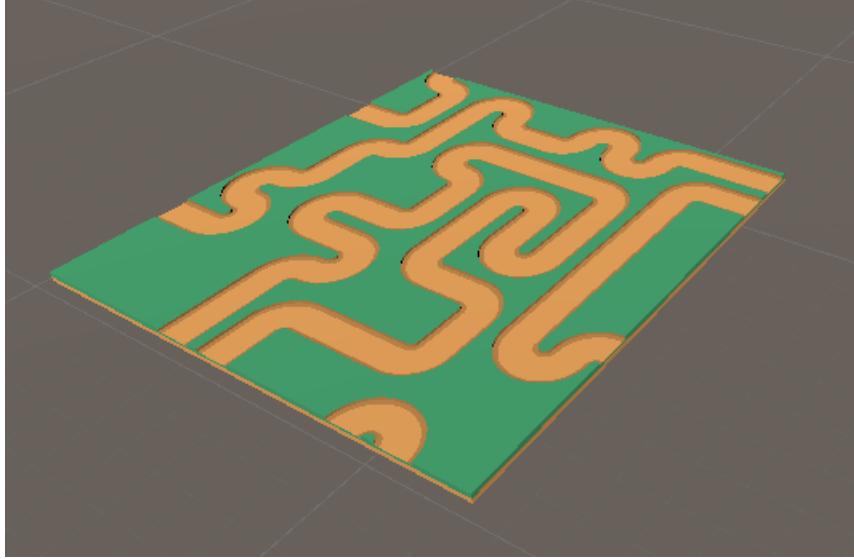
# Path Constraints Tutorial

This tutorial continues from the [Getting Started](#) tutorial. It is recommended you complete that one before starting this.

## NOTE

Path constraints are only available in Tessera Pro

So far we succeeded in generating a level composed of grass and path tiles:



However, for many purposes, a level like the above is not acceptable. If the player is only able to walk along the path, then it's not possible for the player to walk over all the path tiles of the map, there is simply no route between them.

So far, all our generation has been *local* - that is, we've controlled what tiles are placed next to each other, without any regard for the overall structure. Now we are going to use the [PathConstraint](#) to assert some *global* behaviour on the map. The path constraint can be used in various ways, this is only an introduction.

## Setup

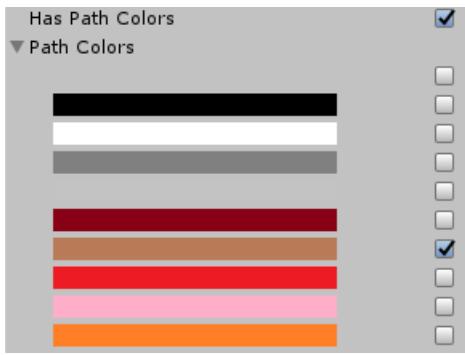
First select the generator object. Enable the `backtrack` option. Backtracking makes the generator try harder to find a viable solution. It's necessary when using the path constraint as the generator can otherwise get stuck trying to find a valid path.



Next, add a `Path Constraint` component to the generator. We need to configure the constraint, by telling it what we consider a path.

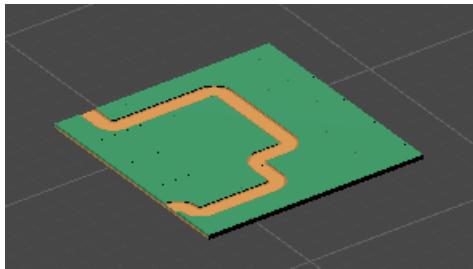
Recall that we colored the tile edge green if that edge was grassy, and brown if that edge had a path. We can give that information to the constraint.

Check the "Has Path Colors" checkbox, and then check the color corresponding to the path.



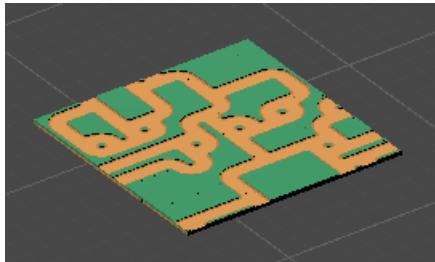
This tells the generator to search for all the sides of tiles that have that color in the center. If two such sides connect together, then it considers there is a path between those two tiles. The constraint then ensures that all path tiles connect to each other.

If we run the generator now, it'll only ever generate a single path:



## Getting fancier

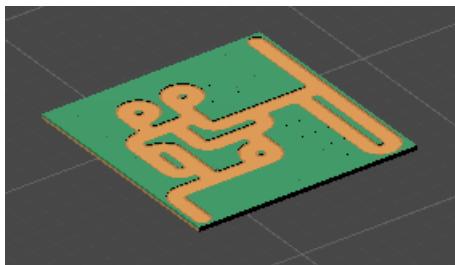
Let's add the `tile_split` tile to the generator, as you can get a lot more interesting paths once you allow junctions. The constraint will still ensure that it's always possible to walk across the map.

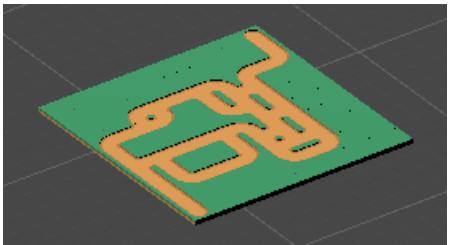
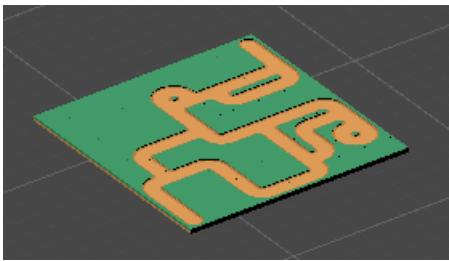


Add a `tile_endRound` to opposite corners of the generated area as a [pinned tile](#). Pinned tiles are ignored by constraints (unless using [PinType.Pin](#)), but the brown painting on these tiles forces adjacent tiles to be paths, and thus the path constraint must connect them together.

A few last tweaks: Set the skybox to the `tile` object to stop the path going off the edge of the map. And set the weight of the `tile` object to 10, increasing the ratio of grass tiles to path tiles.

Now we have a generator that makes self-contained full navigable maps. Here's a few results.





## Additional options

The Path constraint default to ensuring all tiles are connected, but has several other options too.

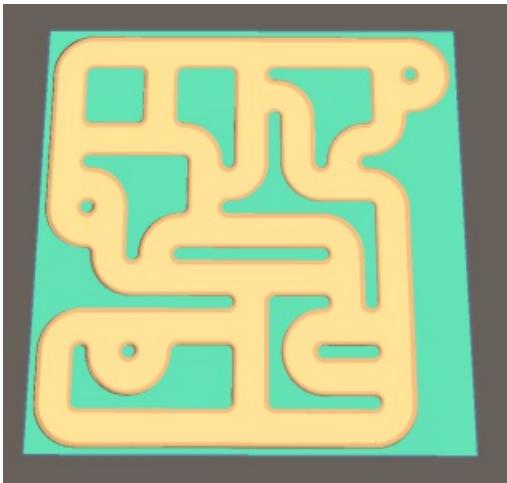
### Connected

When connected is true, then the constraint forces that all path tiles must have a contiguous path between them.

Connected is on by default.

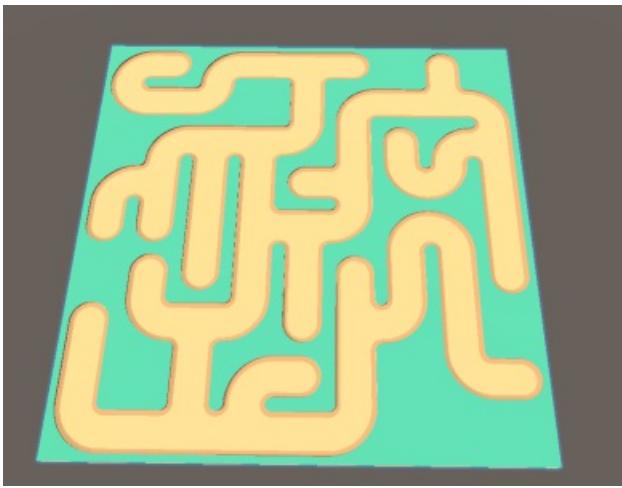
### Loops

Setting loop forces there to be at least two non-overlapping valid paths between any two connected path tiles.



### Acyclic

Setting acyclic to true bans all cycles, forcing a [tree](#) or [forest](#).



### Prioritise

Prioritise is an experimental features. When true, Tessera prefers picking cells near the path for generation over other tiles. This can improve search quality.

### Parity

Experimental setting. Enable this if your path tileset includes no forks or junctions, it can improve the search quality.

# Generating on Mesh Surfaces Tutorial

This tutorial continues from the [Getting Started](#) tutorial. It is recommended you complete that one before starting this.

## NOTE

Generating on Mesh Surfaces is only available in Tessera Pro

By default, Tessera generates tiles in a regular grid - every cell is the same and shape, and placed in the same relation to each other.

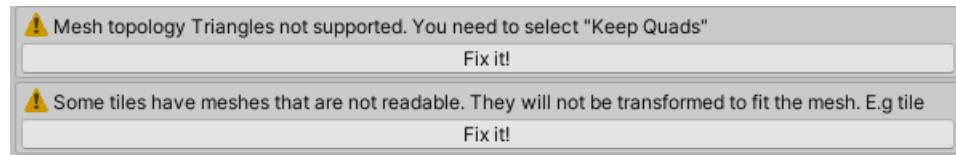
Surface mesh generation allows you to ignore this entirely. Instead, each tile will be placed in a cell corresponding to the face of a given mesh.

That means each cell will have a different shape, and the adjacency connections between cells are determined by the edges of the mesh. The local y-axis used when designing the tile will get rotated to match the normal of the face, and similarly the local x and z axes will point tangent to the face.

Let's make a micro-planet as an example.

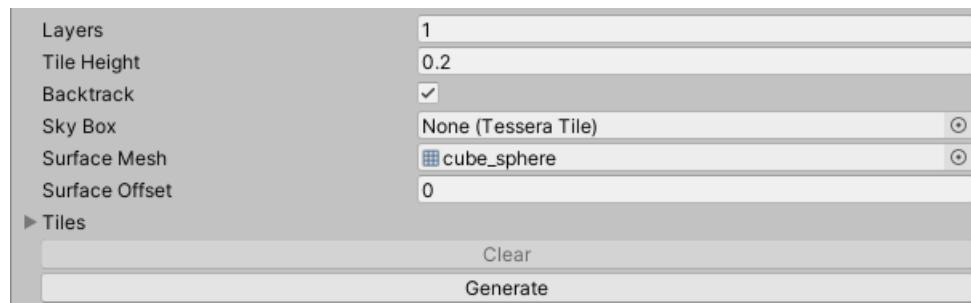
Start with the generator created in the previous tutorials and find the `Surface Mesh` property in the inspector. Click the  icon and select `cube_sphere`, a mesh that comes in the Samples folder of Tessera Pro. You can use any quad mesh (i.e. all the faces have exactly 4 edges).

Depending on which assets you use, you may see the following warnings:

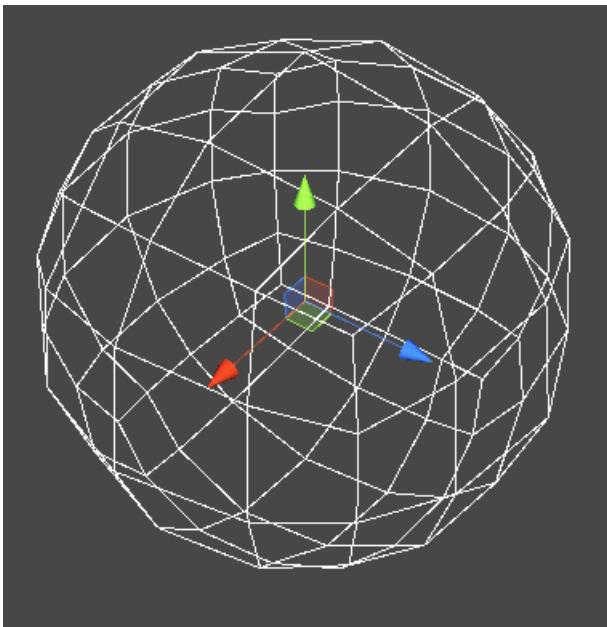


The surface mesh feature requires certain import settings to be configured. Press the fix it buttons will automatically change the assets as needed.

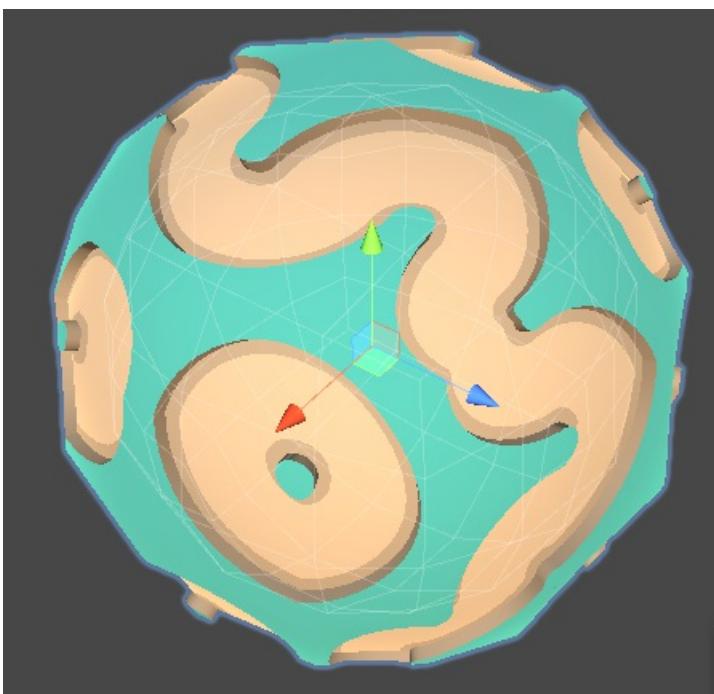
After setting a mesh, the inspector UI will have changed a little:



If you enable gizmos, you should be able to see a wire mesh.

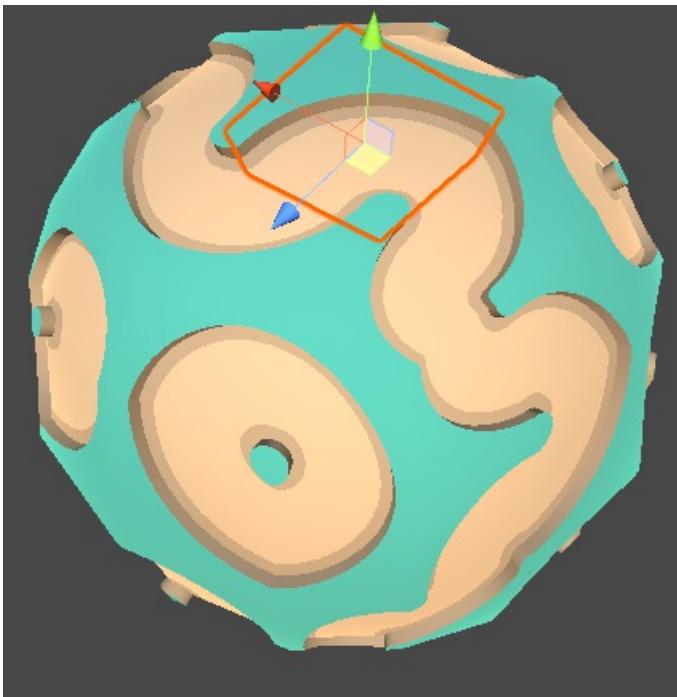


That's pretty much it. Hit generate, and see the results.



As you can see, one tile has been generated for each face of the mesh. Further, Tessera has transformed any meshes associated with the tile to make them fit the face. That gives us a totally seamless result with non square tiles, even though all the designed tiles were square.

Tessera will transform meshes found in MeshFilters, MeshColliders and BoxColliders. It'll also modify the position, rotation and scale, so the created GameObjects interact most compatibly with the other features of the game.



## Caveats

A few features do not work with meshes:

- You cannot use Tilemap Output as Unity Tilemaps expect a regular grid of tiles.
- The mirror constraint is not supported.

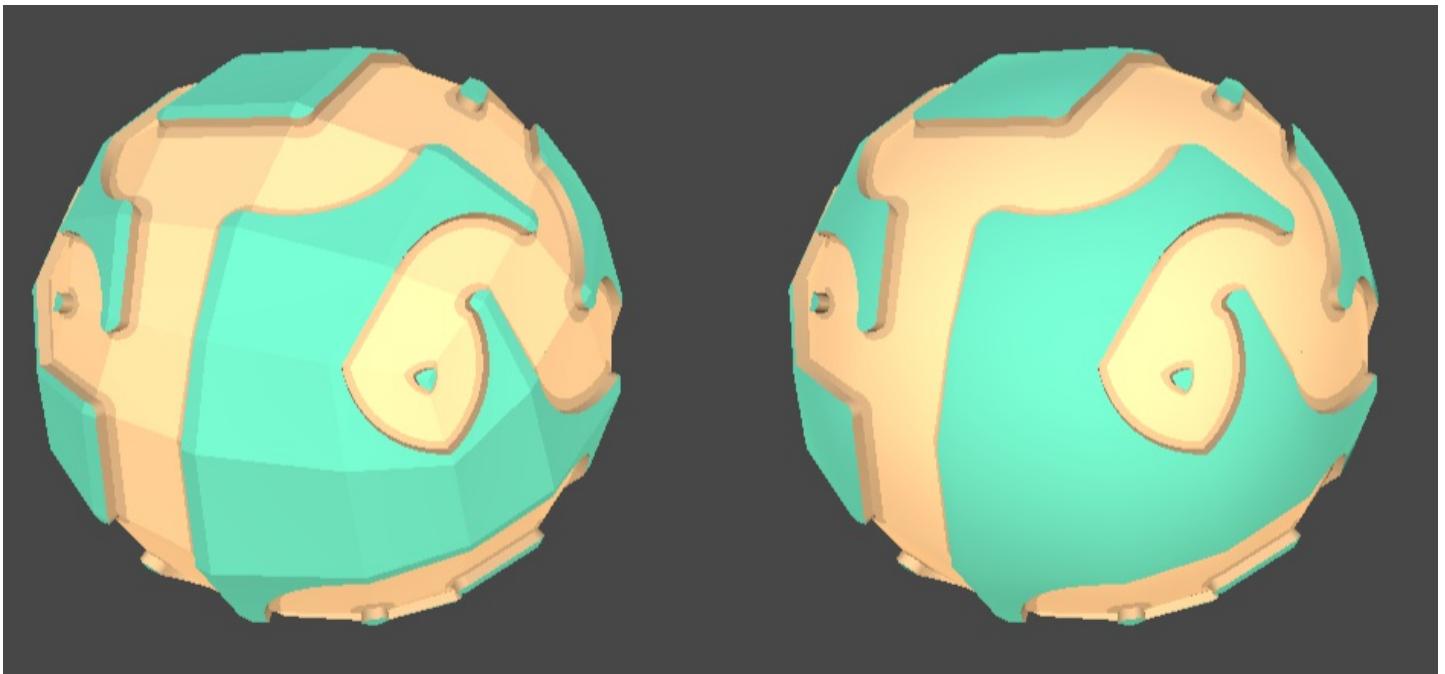
When pressing the Generate button in the Editor, it is not possible to undo the action as Unity can get extremely slow serializing all the transformed meshes.

The mesh distortion pushes around vertices, but never adds any. You may need to add extra vertices to your tile models so that adjacent tiles still line up after distortion.

You are recommended to use tiles with Reflectable and Rotatable enabled. It still works when they are turned off, but it can be hard to control the direction of each cell's local x-axis. It is determined by the order of vertices in the face of the mesh, and most editor programs do not let you easily manipulate this.

## Smooth normals

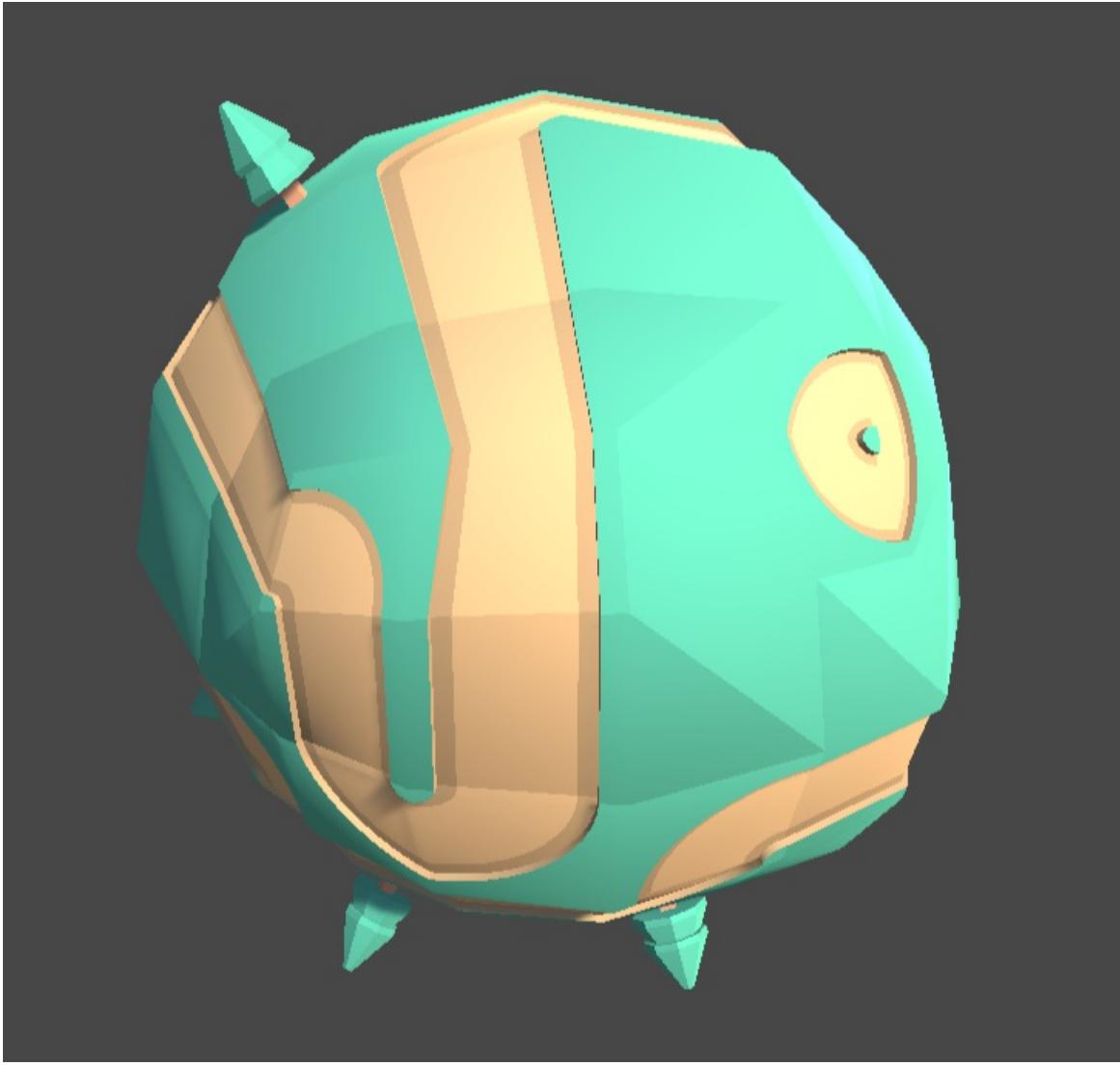
You can configure if normals should be smoothed between adjacent faces of the mesh. For smoothing to work, the surface mesh must have UVs configured.



## Multiple layers

Just as regular generators support a 3d layout of tiles, so does generating on a mesh surface. By setting the `Layers` property (or `size.y` in the API) to a value larger than one, you will stack multiple copies of the mesh surface above one another. Each layer is an expansion of the mesh along the vertex normals by the amount given in `Tile Height` (or `tileSize.y` in the API).

If you've set up your generator with the 3d tiles and skybox described in the [3d tutorial](#), then you can get results like the following on `cube_sphere`.



## Submeshes

If you are Mesh object you've selected for the surface has multiple submeshes, (i.e. multiple material slots), then you can filter which tiles are appropriate for which part of the submesh, similar to volumes, above.

The Generator inspector will automatically detect this case. Simply turn on "Filter By Submesh" and then select which tiles appear where.

Submesh 0 filter
checkboxes
inner_corner <input checked="" type="checkbox"/>
straight <input checked="" type="checkbox"/>
outer_corner <input checked="" type="checkbox"/>
innercorner <input type="checkbox"/>
wall <input type="checkbox"/>
roof <input type="checkbox"/>
outercorner <input type="checkbox"/>

Submesh 1 filter
checkboxes
inner_corner <input type="checkbox"/>
straight <input type="checkbox"/>
outer_corner <input type="checkbox"/>
innercorner <input checked="" type="checkbox"/>
wall <input checked="" type="checkbox"/>
roof <input checked="" type="checkbox"/>
outercorner <input checked="" type="checkbox"/>

Tessera Pro comes with a City example that demonstrates this.

## Triangle grids

The above tutorial demonstrated using cube tiles on a quad mesh. But if you use [triangle tiles](#) you can instead use triangle meshes.

# Multiple passes

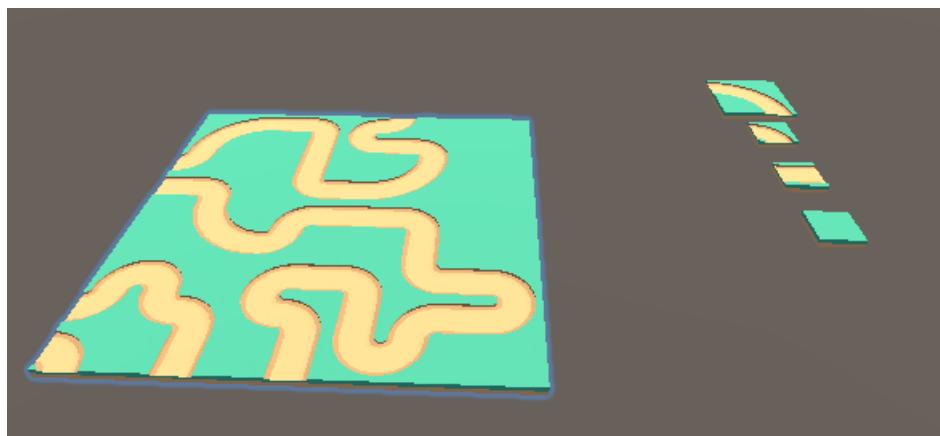
This tutorial continues from the [Getting Started](#) tutorial. It is recommended you complete that one before starting this.

So far we've looked at running the generator once, and filling in the entirety of the tiles at once. As an *advanced* technique, it's possible to run multiple linked generators in the same scene, each configured differently. Running separate passes like this can achieve a variety of effects, and usually runs faster and more reliably than a single generator.

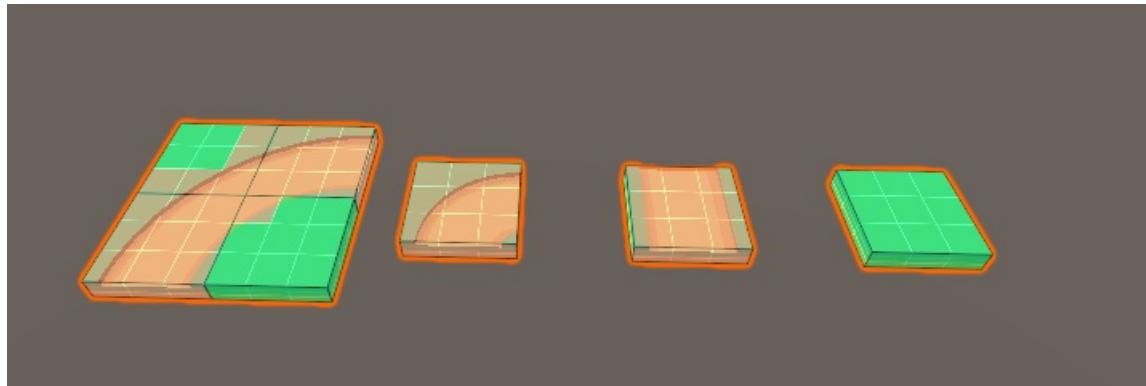
The key idea is that when you run a generator in Tessera, it creates additional game objects to fill the scene. Additionally, before the generator runs, **it scans the scene for any relevant game objects that should constrain the generation**. So if we run two generators sequentially, overlapping in space, then the output of the first will act as pins constraining the second.

This tutorial will show you how to generate ground tiles in one pass, then add decorations using a second. The decorations will be constrained to only attach to relevant tiles.

In the [previous tutorials](#), we made a generator that generates a series of paths through a grassy plane.



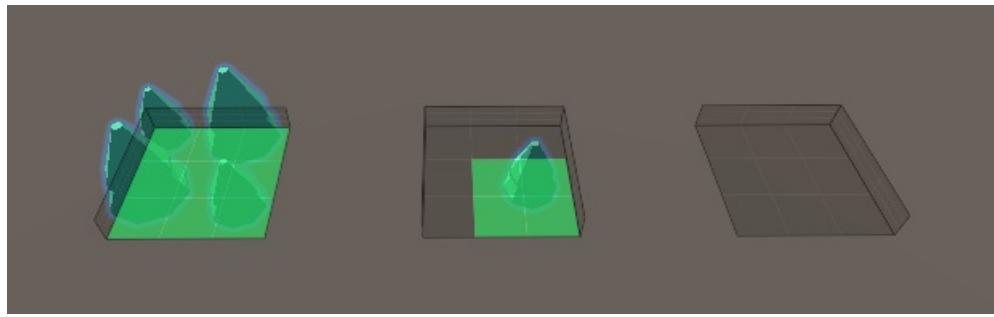
First, we'll add some extra paint on the top surface of the existing tiles to mark where is grassy, and where is path.



Now we need to make a new tileset for the decorations. With the assets supplied, make 3 tiles similar to the given ones.



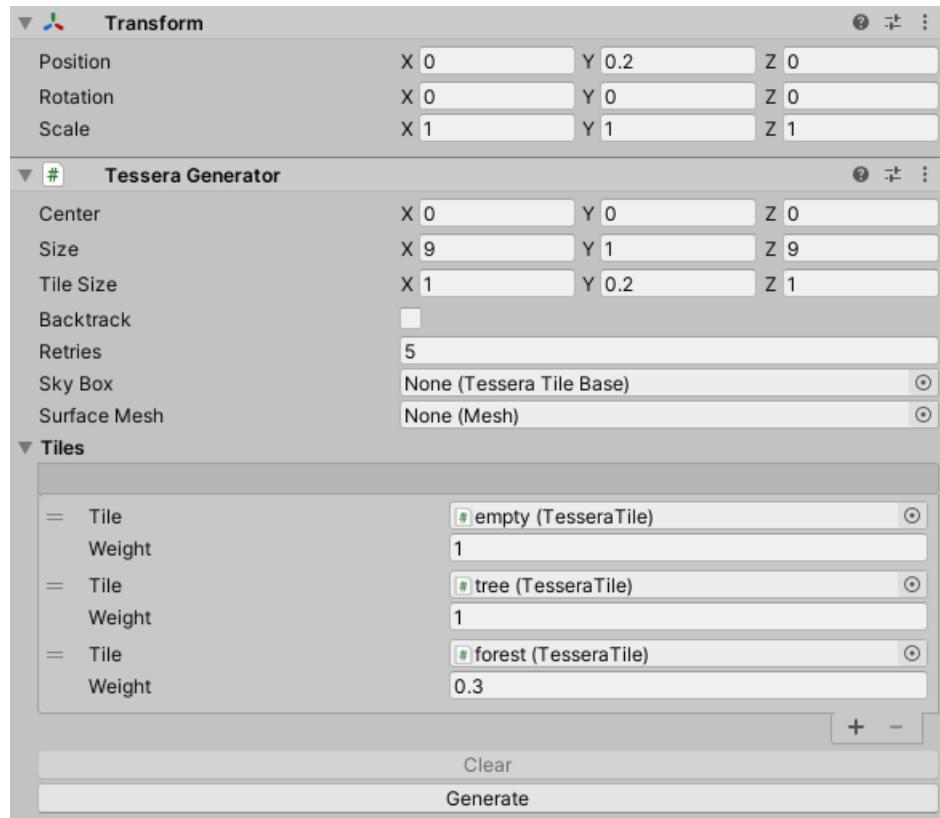
Then, using "Show Backfaces", paint the undersides of the tiles as indicated.



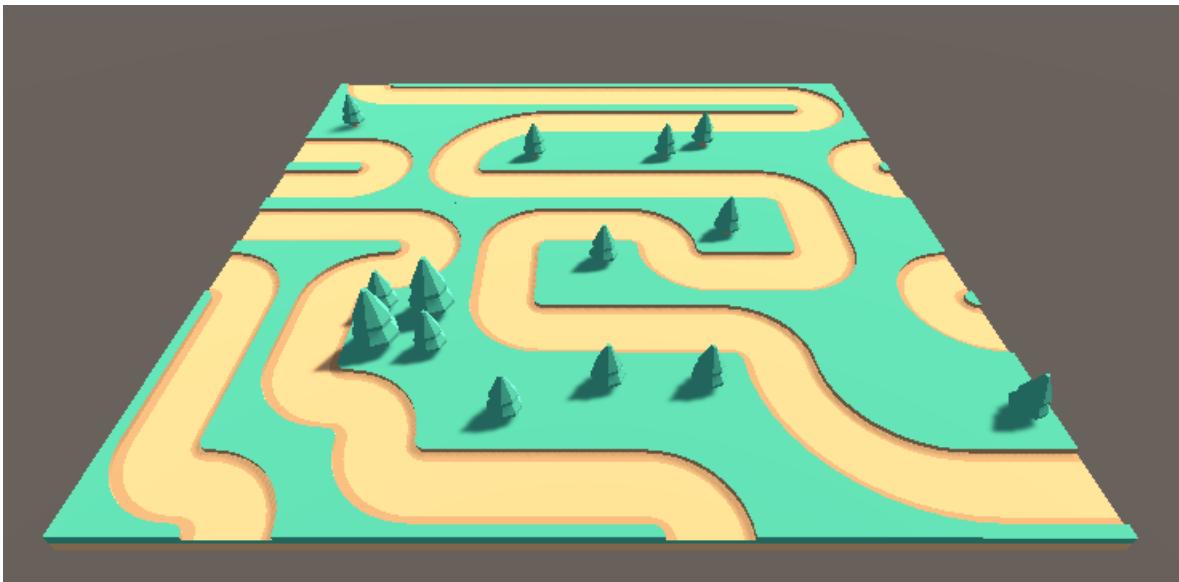
Most of the tile is left unpainted, as we don't care what it connects to. All we want to ensure is that the trees themselves are never placed directly above a path.

Now, create a second generator. This generator should be a new game object which is placed directly above the original one by 0.2 units.

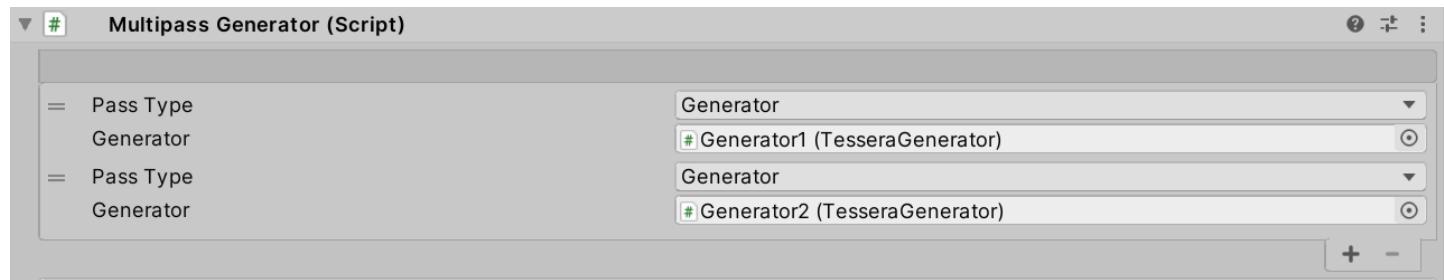
Set it up as follows.



Now run the two generators in order - starting with the path generator, then the tree generator. You should find that the trees are only generated where there is appropriate grass for them.



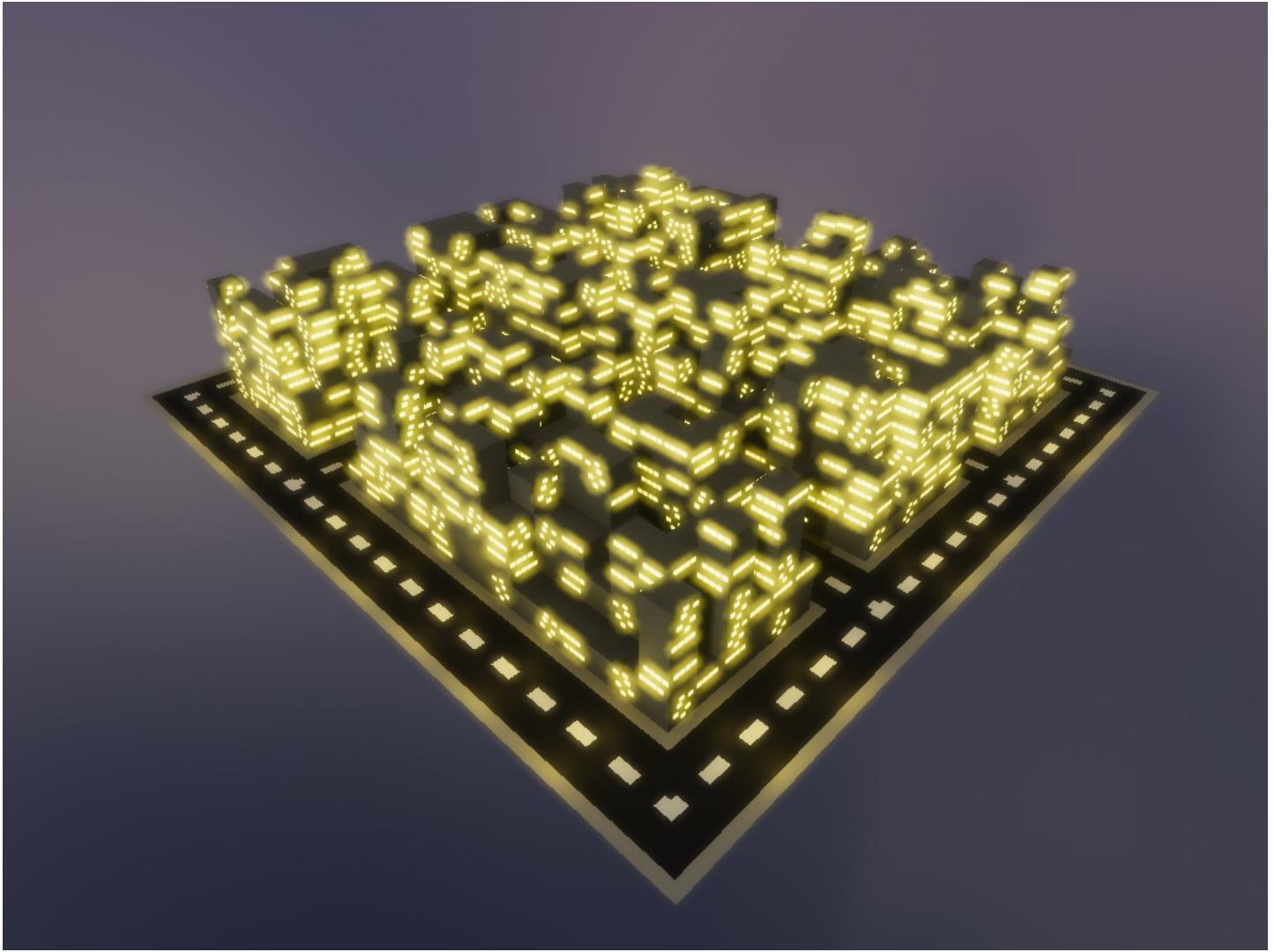
To make working with multiple passes easier, you can create a [Multipass Generator](#) that holds a reference to each of the generators you've just set up. This is a convenience for running both at once.



## Further customization

Multiple passes are even more powerful once you realize that *anything* created by the first pass will be used by the second pass. That means you are not limited to just adding extra tile paint as done above.

See the Skyscraper sample for an advanced usage.



In this sample, the road network is generated by one generator, then the skyscrapers are filled in by another. But the two generators use different grids: the buildings have a tilesize one third of the road tiles. To make this work, we instruct Tessera to replace the road tiles with big tiles that fit the smaller grid.

Tessera Instantiate Output

Parent: None (Transform)

Tile Mappings:

From	To
Straight_big (TesseraTile)	Straight_small
Threeway_big (TesseraTile)	Threeway_small
Corner_big (TesseraTile)	Corner_small

+ -

The sample also activates/deactivates [pins](#) so they only apply to particular generators.

# Constraints

The basic configuration of a generator involves setting up tiles, and painting those tiles to show how they can be placed next to each other.

This page documents further configuration you can do to tightly control the generation process.

There are many constraints available in Tessera:

- [Initial Constraints](#)
  - [Pins](#) fix a particular tile in place
  - [Volume constraints](#) filter a particular area to a subset of tiles
  - [Skyboxes](#) controls how tiles on the boundary of the generation work
  - [Submesh filters](#) applies per material settings when working with [surface meshes](#).
- [Generator Constraints](#)
  - [CountConstraint](#) - ensures the number of tiles in a given set is less than / more than a given number.
  - [MirrorConstraint](#) - ensures the output remains symmetric.
  - [PathConstraint](#) - detects contiguous paths between tiles, and ensures various properties about those paths, such as connectedness.
  - [SeparationConstraint](#) - ensures that the given tiles are spaced at least a certain distance apart

## Initial Constraints

Initial constraints configue the intial conditions of the generator. They generally set up by adding behaviours into the world.

- [Pins](#) fix a particular tile in place
- [Volume constraints](#) filter a particular area to a subset of tiles
- [Skyboxes](#) controls how tiles on the boundary of the generation work
- [Submesh filters](#) applies permaterial settings when working with [surface meshes](#).

### Pins

Sometimes you just want to place a tile at a particular location without Tessera choosing for you. For example, you might want to place a entrace manually at one side, or write custom code to draw path ways for you. Pinned tiles lets you do this, and then have Tessera generate the rest of the tiles for you.

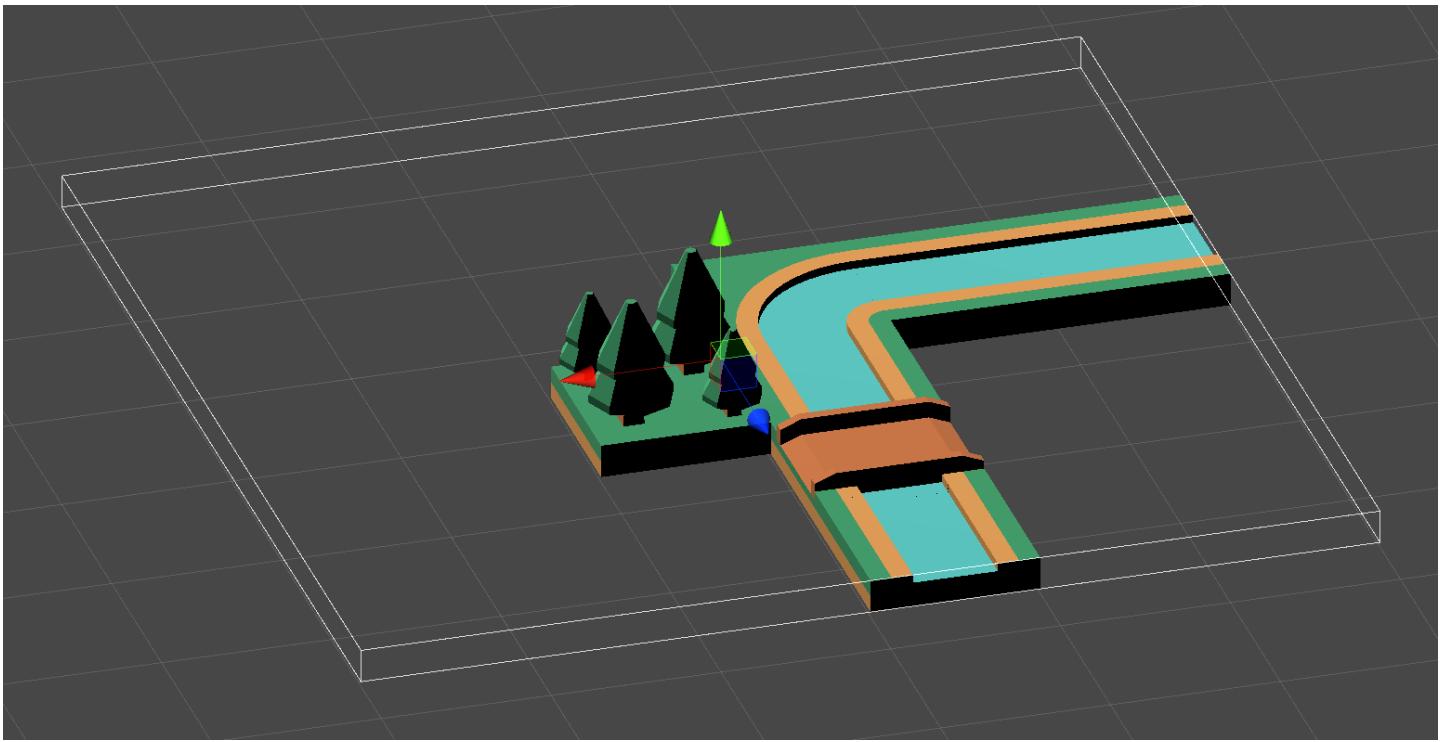
There's multiple ways to specify pins - all of them involve putting a game object inside or adjacent to the generator area, in the position of the cell you want to affect. They will be automatically detected and incorporated into the generation.

The most flexible way is to great a game object with the [TesseraPinned](#) component. You can then specify the [PinType](#) and the tile to pin as properties of that component.

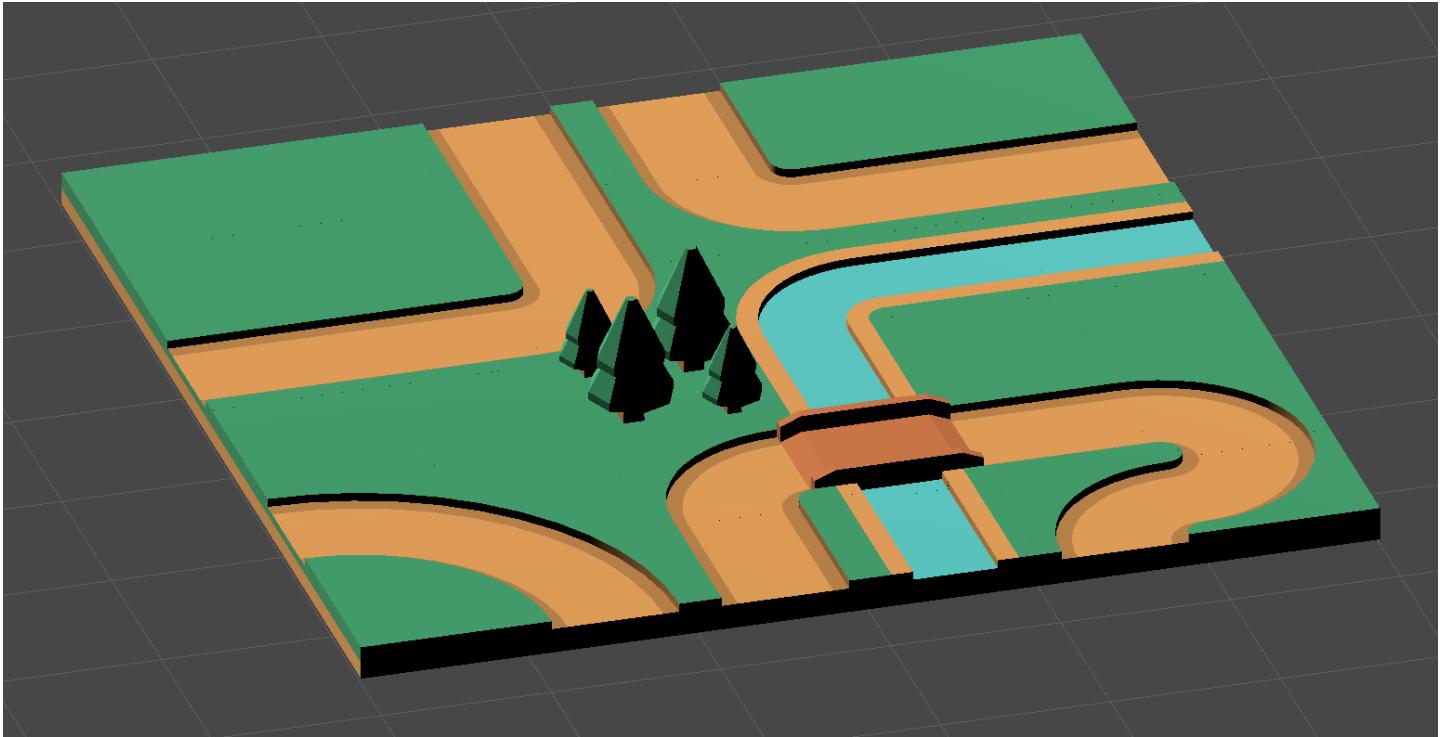
As a convenient short cut, you can also just place a Tessera tile directly. These will be interpreted the same as a TesseraPinned component that references that tile, and has pin type [FacesAndInterior](#).

You can also place an object with both a Tessera tile component and a TesseraPinned component. The TesseraPinned will be assumed to reference the tile component on the same game object.

Here is an example. A generator is set up and a few tiles manually placed.



When the generator is run, the new tiles join up with the placed tiles.



## Configuring pins

TesseraPinned has two key properties:

- **Tile** determines what tile to use for pinning.
- **PinType** controls how exactly the pinned tile effects generation.

These two values are inferred in some cases, as described above.

There are three types of pin:

### **PinType.Pin**

With this pin type, the generator is forced to generate the referenced tile at the location of the pin.

The referenced tile must already be in the generator's tile list (though it can have weight 0).

### **PinType.FacesOnly**

With this pin type, the faces of the referenced tile are used to constrain the cells adjacent to the location of the pin, and the generator is free to pick any tile for that cell.

### **PinType.FacesAndInterior**

With this pin type, the faces of the referenced tile are used to constrain the cells adjacent to the location of the pin. Also the cells covered by the pin are masked out so no tiles will be generated in that location. You may therefore need to manually place a tile or other game object to fill the gap. Unlike [PinType.Pin](#), the referenced tile *does not* need to be in the generator's tile list.

The different pin types can be used to generate a variety of effects. FacesOnly, for example, is good at restricting the generation along a given plane, without forcing any particular tile. FacesAndInterior allows you to put in custom tiles (often called "hero" pieces) that never otherwise appear in generation. And Pin is good for forcing a particular tile to be placed, while still allowing that cell to be part of generation. That can be important if you need the cell to interact with [generator constraints](#) or other Tessera features.

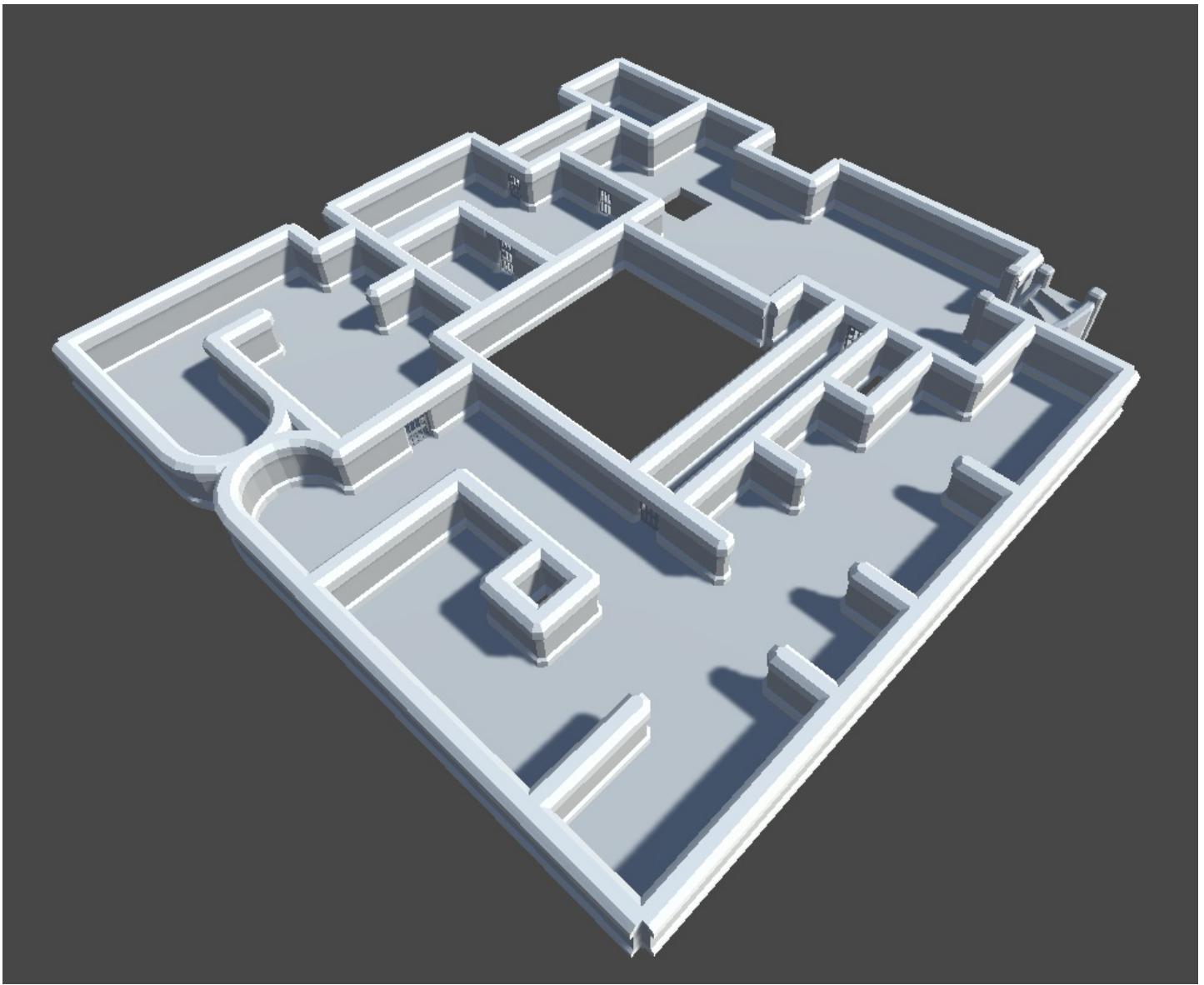
## Volumes

Add a GameObject with the [Tessera Volume](#) component to filter an area of cells to only use a subset of tiles.

To use it, add a game object with the volume component, and set the cells you want to filter to. You can optionally specify a generator for convenience, but it's not necessary.

Next add colliders to the game object to define the area selected. A cell is in the volume if its center is inside at least one collider.

Here's a dungeon generated with a box collider volume in the center, configured to filter to just an empty tile.



Unlike [pins](#), volumes have no way to configure the rotation and reflection of the generated tile.

Volumes can also be used to remove tiles from the generator entirely, by setting the volume type to [MaskOut](#). This behaves similarly to pinning tiles with [FacesAndInterior](#) - the missing cells interact differently with constraint.

## API

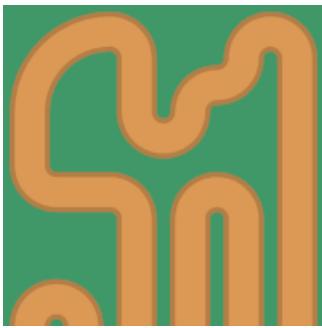
The default behaviour of generators is to search the scene for appropriate pins and volumes. This can be slow for large scenes, and it's not suitable for advanced customization. You can use the C# API to set pins and volumes directly.

First, disable the automatic detection by setting [searchInitialConstraints](#) to false. Then you can supply your own initial constraints by setting [initialConstraints](#). You need to call [GetInitialConstraintBuilder](#) to get a utility to convert various objects to initial constraints.

## Skybox

Setting the skybox property of the generator will automatically constrain all tiles on the boundary of the tile area. The skybox should be a TesseraTile component. Whatever is painted on the top of the skybox, will constrain the top of every tile on the topmost layer of the generator. Similarly for the other sides of the cube.

In this example, the skybox has been used to force the bottom edge to be all paths, and the other edges to have no paths.



## Notes

The tile used for a skybox should not be a big tile, this will cause weird behaviour.

If a pinned tile constraint and the skybox both apply at a particular location, the tile constraint takes precedence.

Note that a given skybox face is repeated as a constraint for *all* tiles which have a corresponding face on the boundary. This can sometimes cause counter-intuitive results. For example, if you have a 3d example like [in this tutorial](#), then you cannot have a skybox that is "grass" on the sides. Doing so allows surface tiles, but would stop air and solid tiles from touching the boundary, which will cause the boundary to fail.

The solution to this is to make a new [palette](#) color just for the skybox. You can then set that color connect with *multiple* other colors, so that it's possible to place all the tiles that can abut the boundary.

## Submesh filters

### NOTE

Submesh filters are only available in Tessera Pro

If you are [generating on a mesh surface](#), and that mesh has multiple submeshes, (i.e. multiple material slots), then you can filter which tiles are appropriate for which part of the submesh, similar to volumes, above.

The Generator inspector will automatically detect this case. Simply turn on "Filter By Submesh" and then select which tiles appear where.

The screenshot shows the Generator Inspector interface with two submesh filters expanded:

Submesh Filter	inner_corner	straight	outer_corner	innercorner	wall	roof	outercorner
Submesh 0 filter	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Submesh 1 filter	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

## Generator Constraints

### NOTE

Generator constraints are only available in Tessera Pro

These constraints control the *global* behaviour of generation. They are very powerful, but can use generation to fail more frequently.

To use them, find the game object that has the [generator](#), and add additional components to it from the [Component > Tessera](#) menu.

At present the constraints are:

- [CountConstraint](#) - ensures the number of tiles in a given set is less than / more than a given number.
- [MirrorConstraint](#) - ensures the output remains symmetric.
- [PathConstraint](#) - detects contiguous paths between tiles, and ensures various properties about those paths, such as connectedness.
- [SeparationConstraint](#) - ensures that the given tiles are spaced at least a certain distance apart

There is a tutorial on [how to use path constraints](#).

# Count Constraint

The [CountConstraint](#) is a [generator constraint](#) that counts the number of tiles in a given set that are generated and ensures that the total in the final output is above/below a given number.

## Eager

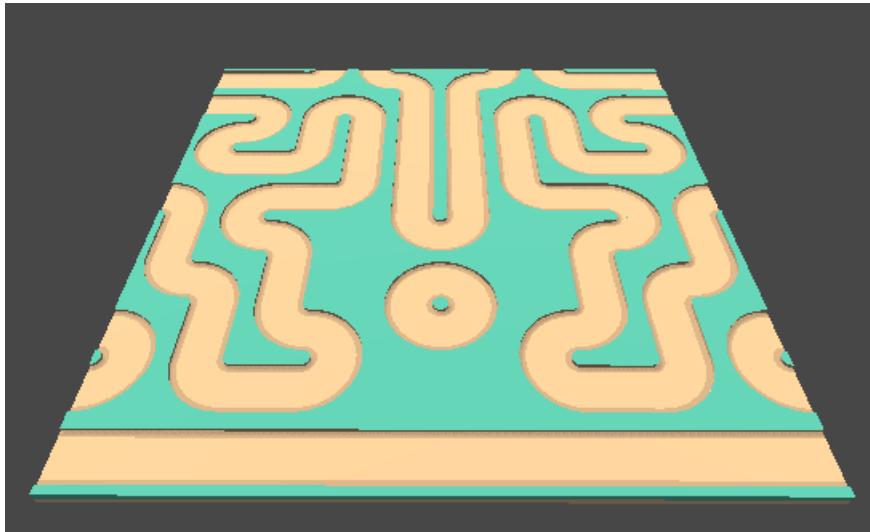
Normally, this constraint is run while generation occurs i.e. it'll count the tiles as generation occurs, and take action once the limit has been reached. This strategy generally works well, however in some cases it'll cause problems.

1. If a tile has a very high weight, but a low limit in the Count Constraint, then the limit will be reached early and the rest of the generation will have none of those tiles.
2. If the tile is very hard to place, and you've required at least a certain number of them, then the Count Constraint can wait too late before forcing placement, so the generation never succeeds.

These issues can be avoided by setting the constraint as "eager". When eager, it'll place the counted tiles first, before attempting to place any others, ensuring an even distribution, and priority positioning.

# Mirror Constraint

The [MirrorConstraint](#) is a [generator constraint](#) that ensures that the generated output is symmetric about a given axis.



You can select one axis, X, Y or Z. If you want symmetry about 2 axes, use two copies of this constraint.

The mirror constraint is unlikely to work very well unless you enable `reflectable` on your tiles.

If you have initial constraint tiles, then the empty cell that mirrors that is ignored by the mirror constraint.

## Odd vs evens

If the tile generator is an odd number of cells wide, then the center line reflects onto itself. All tiles placed on the center line must therefore be symmetric themselves. If the width is even, then there is less restriction as tiles on one side of the center line just have to connect to their own reflection.

## Specifying tile symmetry

By default, the constraint looks at the painted sides of each tile, and infers that the tile is symmetric in a given axis if the painted pattern is symmetric. It does not look at any other details of the tile. So, for example, if you had an asymmetric mesh, but with symmetric connectivity, you would need to override that.

To do so, enable "Override Symmetric Tiles" and check and uncheck which tiles should be considered symmetric. Marking a tile as symmetric means when it is placed in a cell, it'll also be placed in the mirrored cell, *with the same reflection and rotation*. Marking a tile as asymmetric means when it is placed in a cell, a mirrored copy of it will be placed in the mirrored cell. If the tile is marked as asymmetric and it does not have `reflectable` and `rotatable` set to enable that, then it cannot be placed at all.

Note that due to rotations, you may need to specify y and z symmetry even when reflecting on the x axis, and so on.

# Path Constraint

The [PathConstraint](#) is a [generator constraint](#) that forces there to be a connected path between all relevant generated tiles. See [the tutorial](#) for details on its usage.

# Separation Constraint

The [SeparationConstraint](#) is a [generator constraint](#) that forces a given set of tiles not to be placed too near each other. This can be useful to avoid things "clumping" too much, or as an alternative way of controlling the frequency of tiles, distinct from using a Count Constraint or setting the tile weight.

To use it, specify a tile or list of tiles to separate, and a distance. These tiles will never be placed within this many cells of each other. Distance is measured as individual steps from cell to cell ([Manhattan distance](#)), so on a square grid forms a diamond exclusion zone around each tile.

NB: Big tiles are not fully supported, and will always measure distances from one corner.

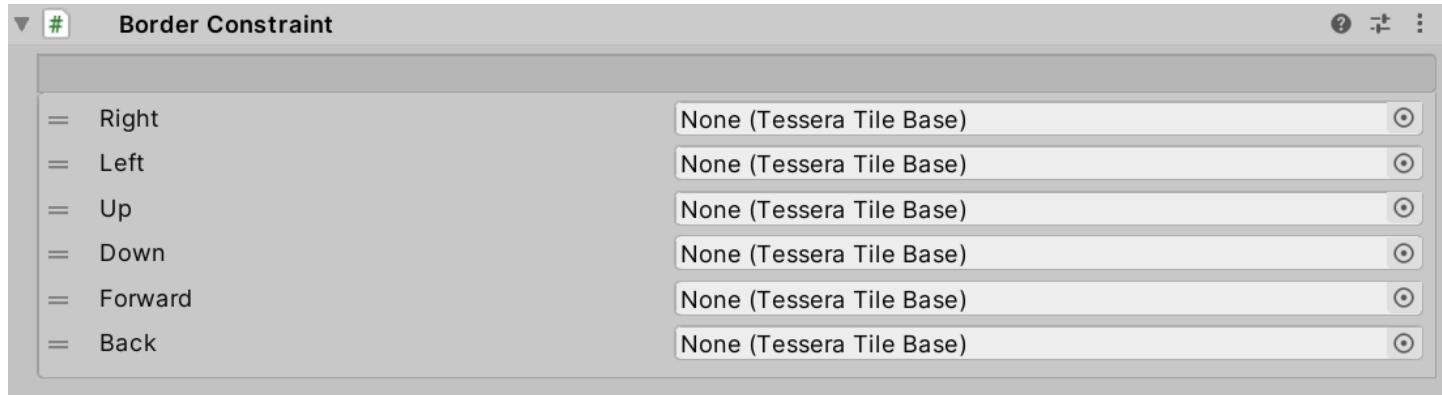
# Border Constraint

The border constraint restricts which tiles can appear on the outermost set of cells of the generator output.

It is similar to placing [volume initial constraints](#), except it automatically works out which cells to affect.

## Usage

Simply add the border constraint to the generator, then select which tiles to use for each border.



You can re-order the sides. Whichever side comes first takes priority for cells that are on multiple sides, i.e. corners.

## Comparison with Skybox

The border constraint has very similar functionality to [skyboxes](#). But they have different advantages.

	SKYBOX	BORDER
Constrains cells	by matching the paint they have along the border	by tile
Controls tile rotation	Yes	No
Works with overlapping models	only if you add paint	yes

# Animation

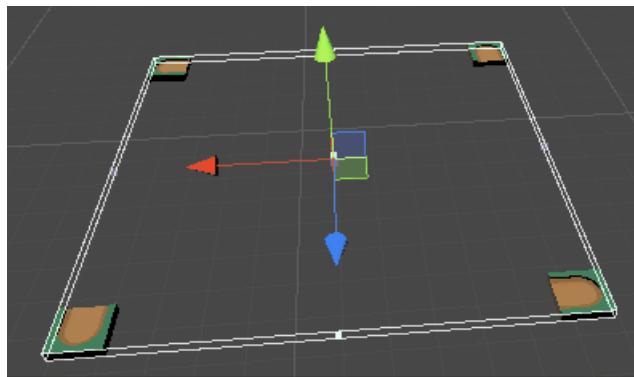
## NOTE

This feature is only available in Tessera Pro

This feature is mostly for fun!

If you add the [AnimatedGenerator](#) to a generator, you can hit Start to run the normal generation process tile-by-tile instead of all at once. It works in both the Unity Editor and in-game.

This animation is much slower than generating all the tiles at once, but it looks cool, and it can show you where the generator is having difficulty. This can be handy if the generation takes too long, or keeps failing, due to not having the right sort of tiles.



`Seconds Per Step` indicates how long to pause between each step and `Progress Per Step` indicates how many units of work to do in a step.

Each unit of work is one of the following:

- Add a tile, and work out all other tiles that are implied by it.
- Backtrack once, done when the current configuration is impossible (if [backtracking](#) is enabled).

`Uncertainty Tile` should be a game object to use to indicate that Tessera is still thinking about a particular tile. The size of the tile indicates how many possibilities still remain.

# Infinite Generator

## NOTE

This feature is only available in Tessera Pro

Tessera's generator only builds finite levels. [This utility](#) can be used to lazily generate infinite levels by repeatedly applying the generator.

It works by splitting up the space into a grid of chunks. Then it works out which chunks need to be generated, and fills each one with the output of the configured generator.

## Configuration

`generator` - sets the generator that is used to fill the chunks. It also determines the size of each chunk.

`parallelism` - the number of chunks that can be generated concurrently. Note that turning this up can cause slightly worse quality output.

`watchedColliders` - determines the volume in which chunks should be generated. You typically want to use a large trigger collider following the player or camera, to ensure that everything nearby is generated.

`infiniteX` - You can set each of X, Y and Z to repeat infinitely. If this is false, you can specify the min/max chunk to give a limit to the amount of chunks generated.

## Chunk Cleanup

By default, chunks are never removed. This can cause memory usage to grow unboundedly as your game runs. You can enable chunk cleanup to resolve this. The generator will periodically scan for chunks that haven't been near a watched collider recently, and clean them up according to [ChunkCleanupType](#):

- None - Chunks are never removed.
- Memoize - Remove the GameObjects, but keep tile data so they can be recreated exactly
- Full - Remove everything associated with the chunk.

Memoize is usually desirable if players can return to a chunk later, otherwise Full is more memory efficient.

In either case, you can set `chunkPersistTime` to control how long chunks stick around for when not needed, and `cleanupInterval` to control how frequently to scan for chunks.

## Caveats

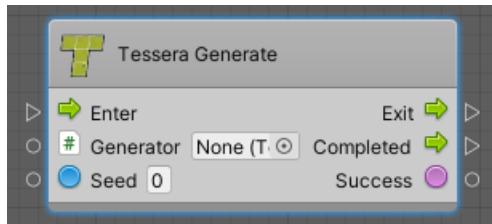
- Only works for square and cube grids
- Only works at runtime
- Generator constraints may not behave intuitively as they only operate on each chunk.
- Scene based constraints are ignored (as scanning for them quickly becomes extremely expensive as the level gets too large).
- Even with a fixed seed, this generation is non-deterministic, as it depends on the order of chunk evaluation.

# Bolt Support

Tessera includes some additional support for Bolt, Unity's visual scripting language.

To enable it, first install Bolt, select `Assets > Import Custom Package...` and select `TesseraBolt.unitypackage` from the Tessera folder. Finally, go to `Tools > Bolt > Build Unit Options` to load the package.

After doing this, a Unit called `Tessera Generate` should be in the fuzzy finder.



This unit will run a particular generator. It works with events run in coroutine mode.

You can also enable Bolt's autogenerated reflection by adding Tessera to the Bolt's assembly list, but this is not recommended as the full API is a bit complicated to use from a visual scripting language.

## Ports

### Generator

The `TesseraGenerator` to run. The default is Self.

### Seed

Fixes the random generation to particular values. The default is 0, which means a new seed each run

### Initial Constraints

This port is only visible if you enable `Set Initial Constraints` in the Unit Inspector. Otherwise, the usual generator behaviour of searching the scene for initial constraints occurs.

A `List<GameObject>` specifying `TesseraTile` and `TesseraVolume` to constraint the generation by. The default is the empty list.

### Exit

Runs immediately after Enter.

### Completed

Runs when the generation is actually completed.

### Success

Set after completion, to indicate if the generation was successful.

# Samples

This is a list of all samples that come with Tessera / Tessera Pro, and what they demonstrate.

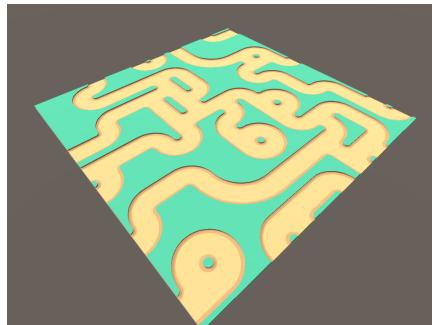
Please note that most samples do not use prefabs as they are more self contained that way. But you are encouraged to make your tiles prefabs in practice.

## GrassPaths

GrassPaths contains a collection of samples that all share a very basic tileset. They demonstrate many Tessera features in as simple a context as possible.

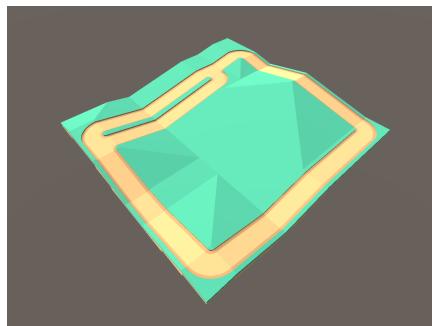
### GrassPaths

Demonstrates configuration of some simple tiles showing a brown path running through a green field of grass. Includes one [big tile](#) that occupies the space of 4 normal tiles. This sample serves the basis for the [getting started](#) and [big tile](#) tutorials.



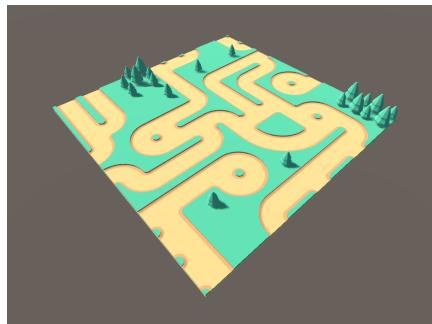
### GrassPaths3d

Adds sloped tiles to GrassPaths.scene, and extends the generator vertically into the 3rd dimension.



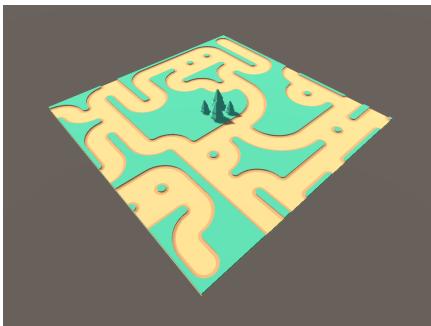
### GrassPathsMultipass

Adds a second generator to GrassPaths.scene, which builds on the first generator by adding vegetation in appropriate places. This is described more in [multipass](#).



### GrassPathsWithPins

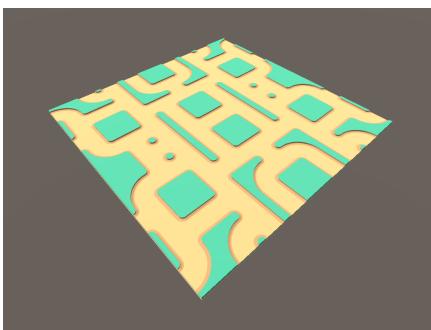
Includes some [pins](#) to GrassPaths.scene, showing how you can fix part of the generation manually.



## GrassPathsWithMirror

### Pro only

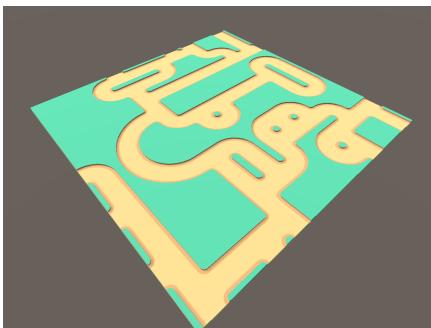
Adds a [mirror constraint](#) to GrassPaths.scene.



## GrassPathsWithPathConstraint

### Pro only

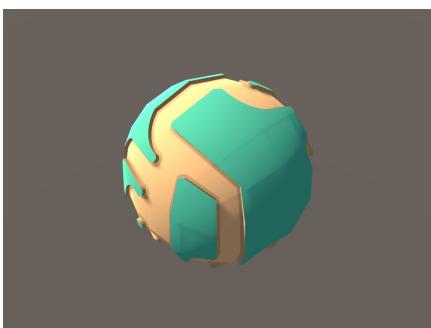
Adds a [path constraint](#) to GrassPaths.scene.



## GrassPlanet

### Pro only

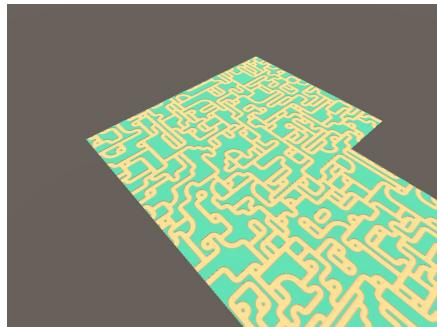
Configures GrassPaths.scene with a [mesh surface](#) to make a spherical planet.



## GrassPathsInfinite

### Pro only

Configures GrassPaths.scene with an [infinite generator](#) that lazy generates tiles near a collider.



## GrassPathsOverlapping

### Pro only

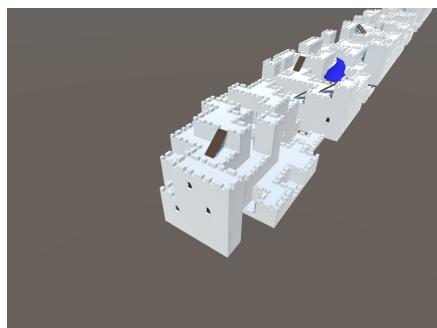
Uses the [overlapping](#) feature to generate an image from a given sample.

Generated

Sample

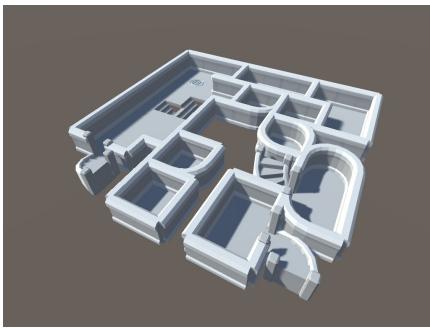
## Castle

A 3d generation showing multiple useful techniques. The tile paint in this sample is based off of Marching Cubes, a popular way of doing auto-tiling. This sample also comes with a small script showing how Tessera can be used to lazily construct infinite generators. (see also GrassPathsInfinite if you have Tessera Pro).



## Dungeon

A generator that creates Diablo-esque dungeons. Includes a [volume constraint](#) that forces empty space in the center of the generator.

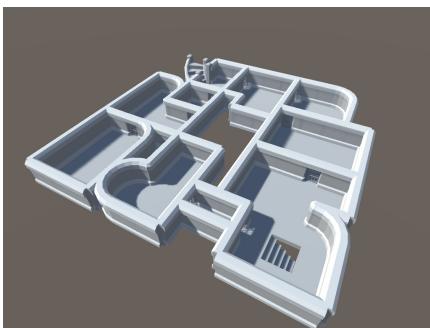


## DungeonWithGeneratorConstraints

### Pro only

Adds several constraints to Dungeon.scene to encourage the level to be more realistic and playable:

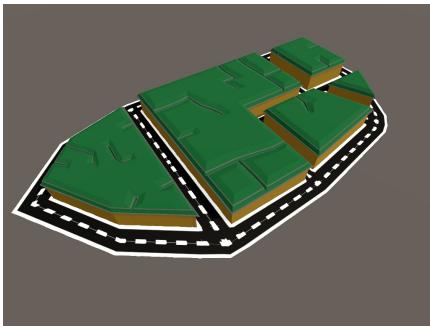
- The level must be fully navigable
- Exactly one staircase up and down.



## City

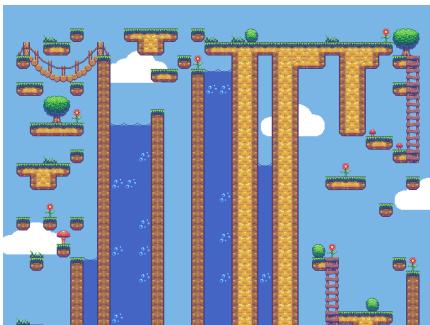
### Pro only

Uses a [mesh surface](#) to map out several city blocks, with different mesh materials representing roads and housing.



## Platformer

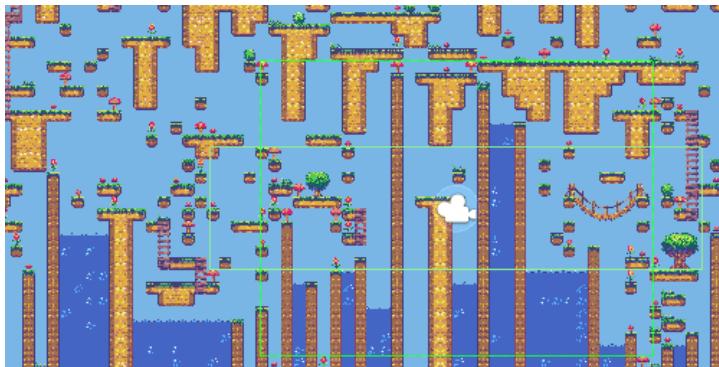
Demonstrates using [square tiles](#) instead of cubes. In Tessera Pro, this sample also demonstrates [writing to a Unity tilemap](#).



## PlatformerInfinite

### Pro only

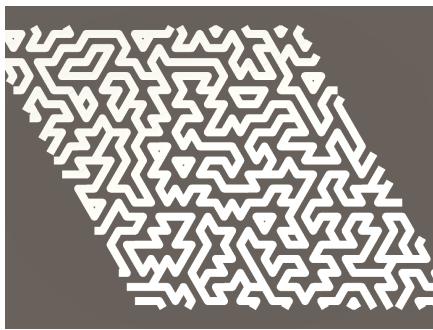
Uses the same tiles as Platformer, but with an [infinite](#) horizontal scroll.



## Hexes

### Pro only

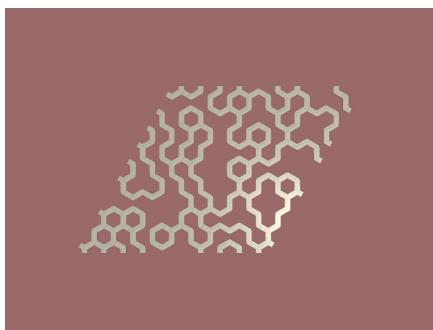
Simple set of tiles demonstrating using [Hex Prism tiles](#).



## Triangles

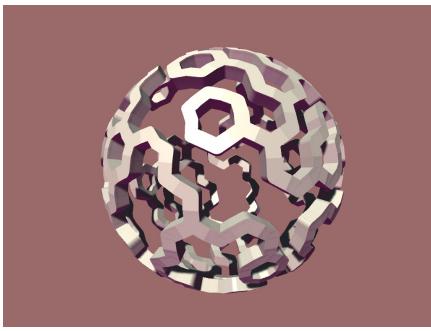
### Pro only

Simple set of tiles demonstrating using [Triangle Prism tiles](#).



### TriangleSphere

Configures Triangles.scene with a [mesh surface](#) to make a spherical example.

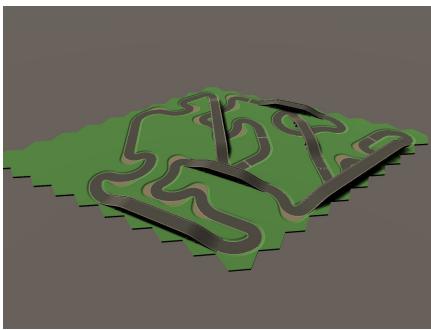


## Hex Raceway

**Pro only**

An example with [hex tiles](#) and a [path constraint](#). This is a 3d generator, with height 2, allowing bridges over the track.

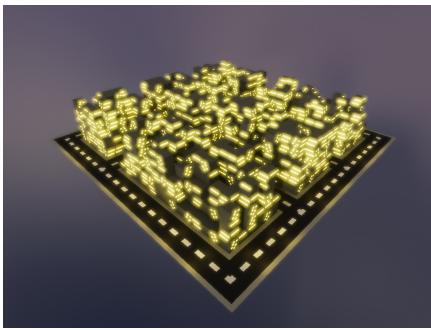
The path constraint has the "Parity" setting on which is specially designed for pathways with no forks, such as this circular track.



## Skyscrapers

**Pro only**

An advanced multipass example. The road network is generated in the first pass, and the skyscrapers generated as the second pass.

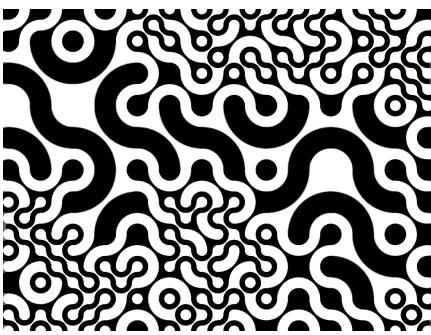


## Truchet

Implements [multi-scale truchet tiles](#), using two different techniques.

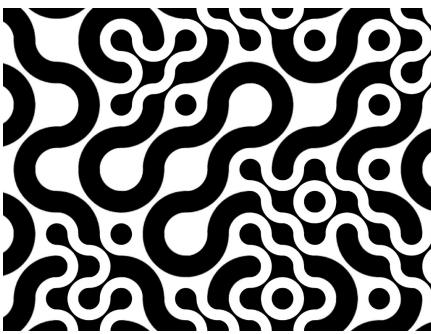
**Truchet Big**

Uses [Big Tiles](#) to support multiple scales of tile in one generator.



Truchet Multipass

Runs two different generators, one for each scale of tile.



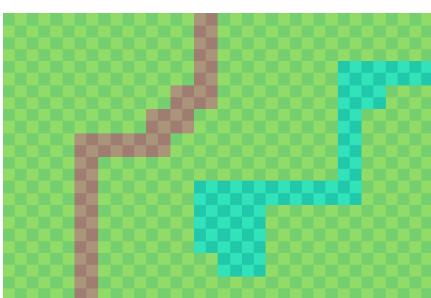
Plains

**Pro only**

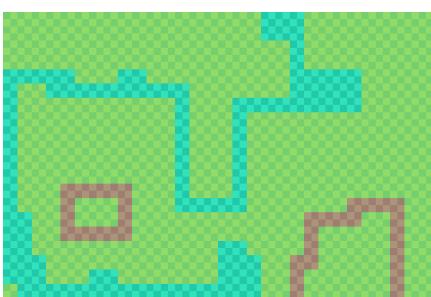
A complicated example using [overlapping model](#) with a Unity Tileset based sample, and multipass.

This shows how Tessera can be used for a simple sort of auto-tiling - the first pass picks the basic terrain type, then the second pass chooses a tile appropriate for the nearby terrain.

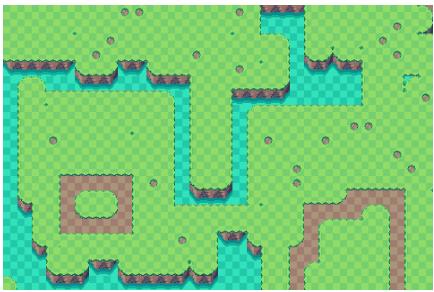
Sample



Pass1



Pass 2



# Palettes

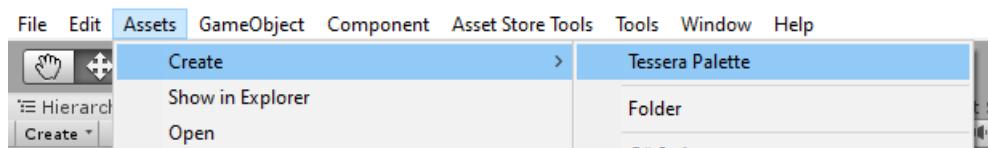
Palettes are an asset type for Tessera that lets you customize which colors you can paint onto tiles, and also controls how colors match.

By default, Tessera comes with a palette of 18 colors (plus transparent). Each color has a short name to help you identify it.

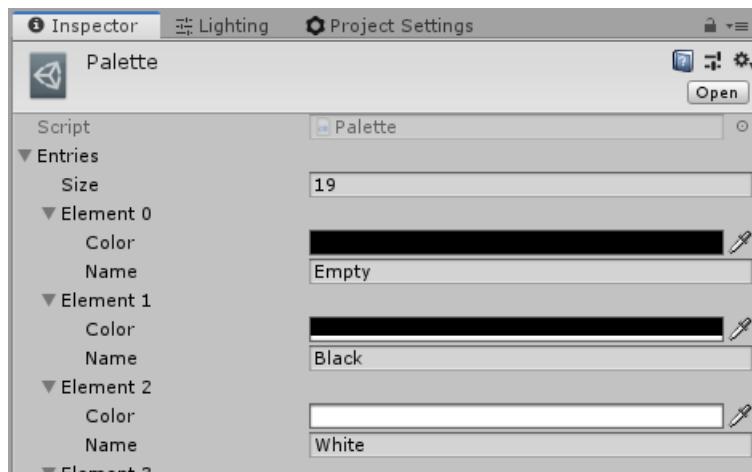
The default palette is configured so that two opposing squares "match" if the colors of those squares are the same, or one is transparent. Recall that two tiles can be placed next to each other if all 9 squares on the face of one tile match with the 9 squares of the opposing face.

## Creating a palette

To customize the palette, create a new Tessera Palette asset from the Assets menus.



This will create a new asset in your project. You can then select the asset and customize the colors in the inspector.



To use the palette, select your tiles and assign the new asset to the palette field.

## Customizing the matching rules

Near the bottom of the inspector, you can see an grid of checkboxes.

Matches															
Black	Black	White	Grey	Brown	Red	Pink	Orange	Yellow	Green	Cyan	Blue	Magenta	Purple	Grey	Black
Black	Black	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
White	Black	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Grey	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Brown	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Red	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Pink	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Orange	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Yellow	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Green	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cyan	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Blue	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Magenta	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Purple	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Grey	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Black	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Grey	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Brown	Brown	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Red	Red	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Pink	Pink	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Orange	Orange	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Yellow	Yellow	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Green	Green	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cyan	Cyan	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Blue	Blue	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Magenta	Magenta	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Purple	Purple	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

You can click any cell to toggle if that pair of colors matches.

For example, in the image above, black does not match with itself, but does match with white. That means if we had a tile colored black, and another tiled colored white, the generator cannot place two black tiles adjacent to each other, but can otherwise intermix black and white.

If you open the platformer example, you can see an example of this in practise. E.g. walls (black) cannot directly face another wall, but can face onto air (white) or water (blue). I made a second wall color (grey), that works similarly, but lacks the water matching. That allows us to control exactly where water can be placed.

# Cell Types

## NOTE

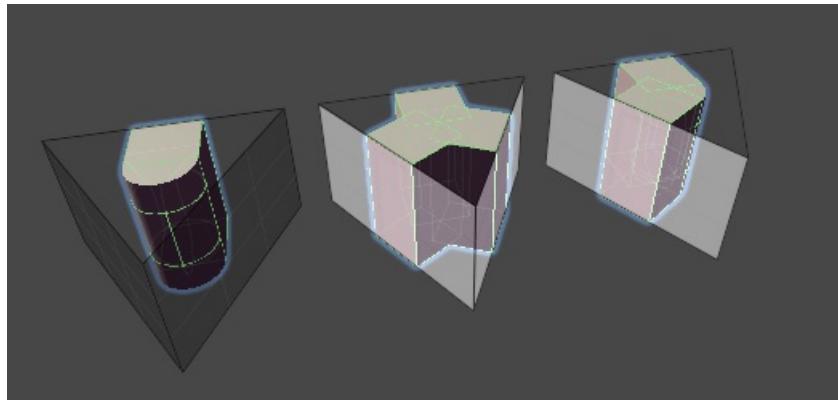
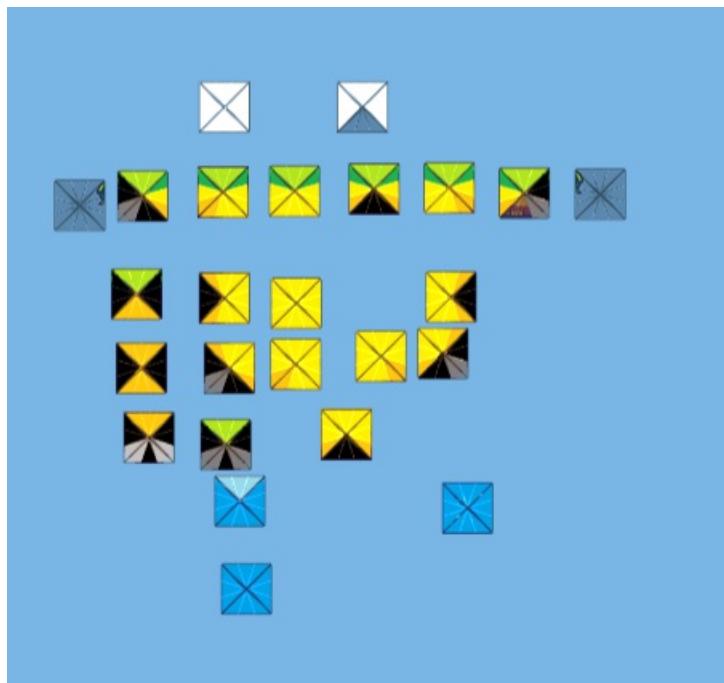
Cell types other than cube and square are only available in Tessera Pro

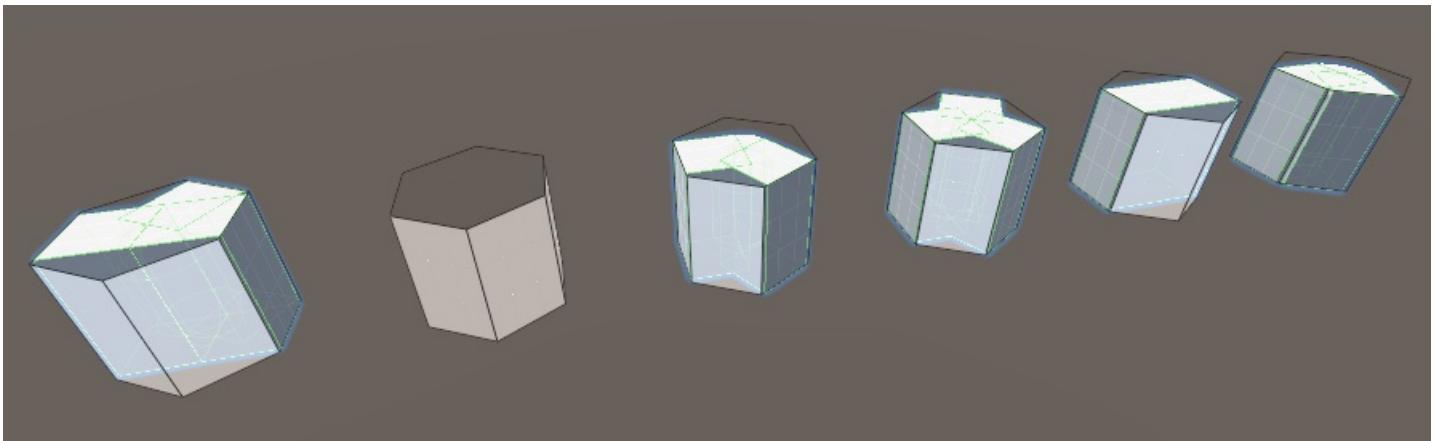
Tessera generally assumes you are working with cube grids, however square, hexagonal and triangular grids are also supported.

To use this, simply use [TesseraSquareTile](#), [TesseraHexTile](#) or [TesseraTrianglePrismTile](#) instead of [TesseraTile](#).

As usual, you can paint the sides of the tiles, and the generator will automatically rotate and place the tiles into cells connecting the sides.

The different tiles in paint mode:





The result of running the generator.



## Caveats

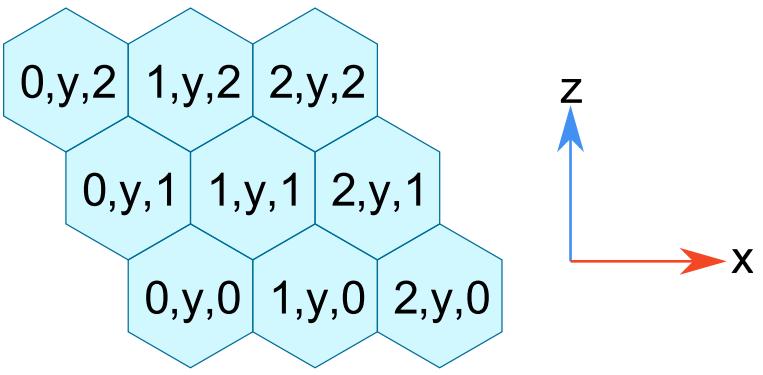
- Only cubes support 3 axes of rotations. Other cell types only rotate in a single plane.
- Triangles generally need rotations enabled, otherwise you must make separate "points up" and "points down" tile variants.
- Different cell types cannot be mixed in a given generator.

## Co-ordinates and measurements

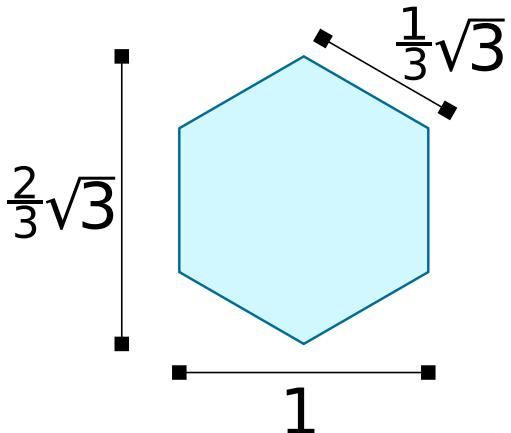
### Hexes

Hexes are laid out in the XZ plane. The Y-axis is used to stack hexes vertically. Hexes are pointy topped, not flat topped. If you need other axes or styles, simply rotate the generator appropriately.

Each hex is referred to using the following co-ordinate scheme:



For a hex of size one, you may find the following measurements helpful.

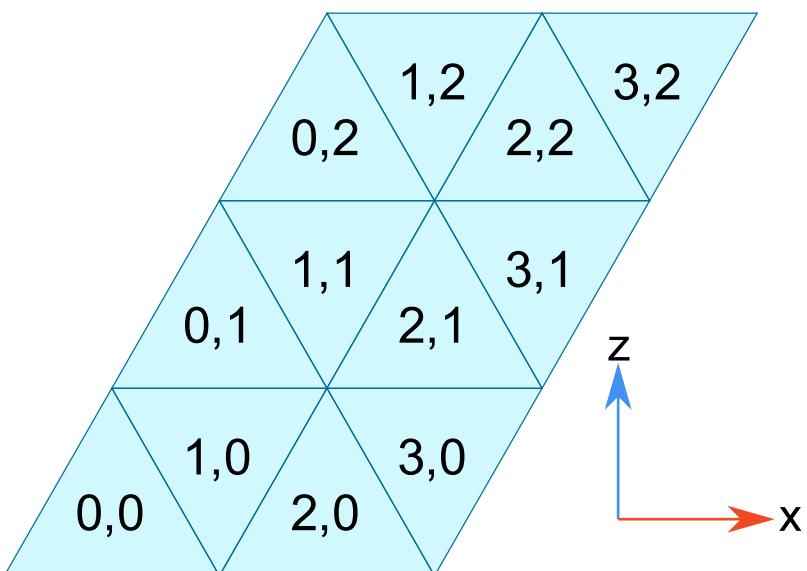


The distance between any two adjacent hex centers is 1.

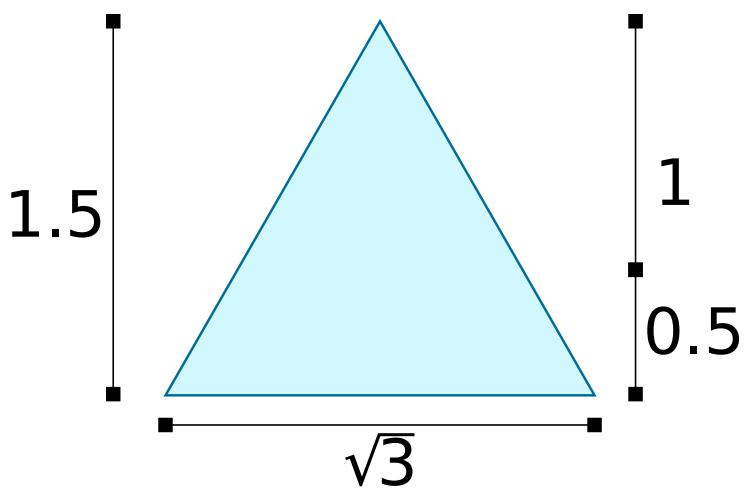
### Triangles

Triangles are laid out in the XZ plane. The Y-axis is used to stack triangles vertically. Triangles point up/down, not left right. If you need other axes or styles, simply rotate the generator appropriately.

Each triangle is referred to using the following co-ordinate scheme:



For a triangle of size one, you may find the following measurements helpful.



The distance between any two adjacent triangle centers is 1.

# Overlapping models

## NOTE

This feature is only available in Tessera Pro

Tessera's main generation algorithm is called WaveFunctionCollapse (WFC), which is responsible for choosing which tiles can be placed in the output.

WFC can be run in two different models: adjacent (aka "simple") and overlapping. By default, Tessera uses the adjacent model. This model forces that pairs of adjacent tiles must connect together, as specified by the tile paint.

There is **experimental** support for overlapping mode. In an overlapping model, you supply samples, which are pre-made levels that the model will learn from. The generator will then ensure that every rectangle in the output matches a rectangle found in one of the sample levels. This can copy the style of the sample level quite faithfully. More details on how it works can be found at <https://github.com/mxgmn/WaveFunctionCollapse> or <https://twitter.com/exppad/status/1267045331004469248>.

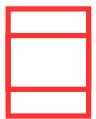
## Example

On the left, a small 6x7 map of tiles has been specified as the sample, and a larger map has been made from it. Every 2x2 rectangle in the output comes from somewhere in the input, with the red box showing one particular patch.

## Generated



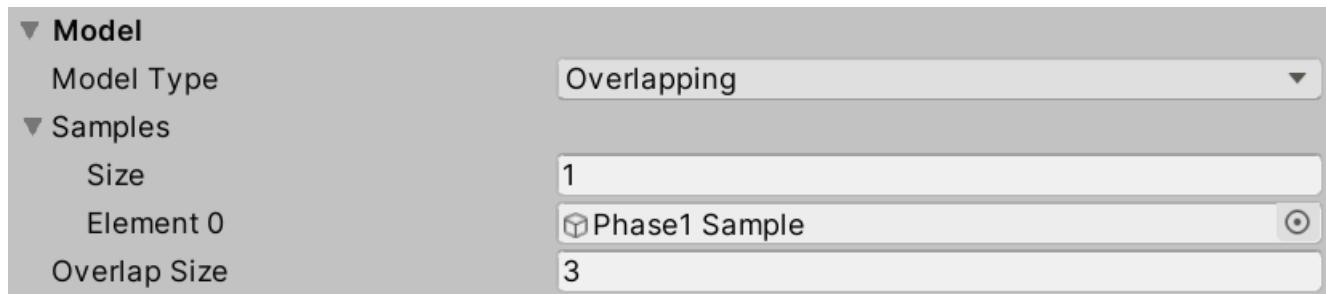
## Sample



## Usage

In the generator settings, set the **Model Type** to **Overlapping**.

You can then supply a list of samples, and the overlap size.



## Creating samples

To create a sample, create an empty game object and add it to the [samples](#) array of the generator.

Then add objects as children of the sample. The children objects should be arranged in a grid with spacing matching the tile size. The children don't need a TesseraTile component, they need to be "linked" to the original tiles instead. Tile are linked by either:

- The name of the child matching the name of a configured tile (suffixes are ignored)
- The child having a [TesseraPinned](#) component, with the tile property set to the tile in question.

Using grid alignment tools such as [ProBuilder](#) can help set up your sample.

For 2d grids, you can also use a Unity TileMap as the sample input. The tiles should have names that correspond with configured Tessera Tile objects. See the "Plains" sample for details.

### Rotation

If the tiles have rotation / reflection on, then reflected / rotated copies of the sample will also be used. It's highly recommended you enable the [symmetric](#) option if you have rotatable tiles, as overlapping model copes much less well than the adjacent model with redundant tile rotations. This feature requires tile paint.

## Caveats

- It is much slower than adjacent models. Using overlapSize greater than 3 is not recommended.
- Overlapping model is a strict constraint - every output rectangle must come from a sample. You will often need to make samples quite large and detailed to get sufficient variety for a good quality output.
- It supports cube and square grids only
- You must manually create sample levels.
- Overlapping interacts strangely with irregularly shaped outputs. You are recommended to avoid pins other than PinType.Pin, and to avoid the MaskOut feature of Tessera volumnes.
- Tile paint is not used for the model, but it will still be read by many other Tessera features such as rotational symmetry, pinned tiles, the path constraint and skyboxes. Thus you will likely need to paint the tiles in addition to making samples.

# Controlling Output

## NOTE

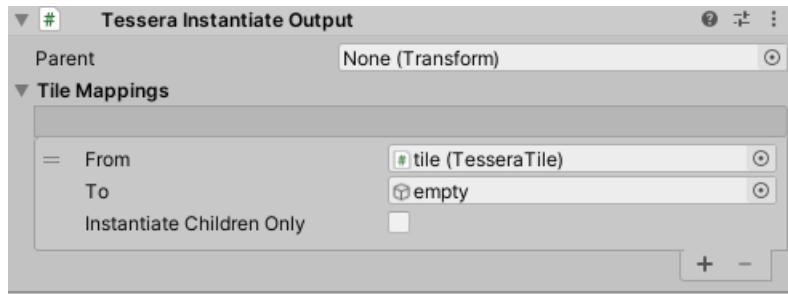
Output control is only available in Tessera Pro

By default, after doing generation, `TesseraGenerator` will instantiate copies of all the tiles as child objects. This is usually what you need, but it is possible to customize it further.

## Creating instances

By default Tessera will instantiate each tile as a child of the generator. If you want to customize this, you can use a `TesseraInstantiateOutput`.

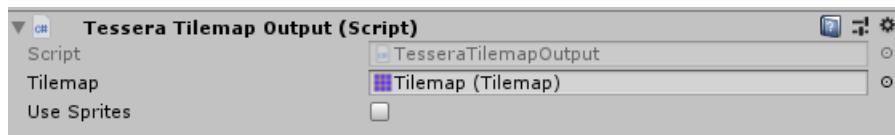
`TesseraInstantiateOutput` lets you customize where the instances are written to, and also which prefab should be instantiated for each tile.



## Writing to a Tilemap

Unity comes with a `Tilemap` component that lets you store sprites and components in a regular grid.

To enable this, select the game object with the `TesseraGenerator` component, and add the `TesseraTilemapOutput` component. Then set the `Tilemap` property to the tilemap component. Then instead of instantiating objects, it will find the appropriate for cell fo the tilemap, and fill that in instead.



## NOTE

You must ensure that the grid spacing of the Tilemap and of the generator are aligned.

Tessera comes with a sample called "Platformer" that demonstrates writing to Tilemaps.

If you check the `Use Sprites` property, then Tessera will attempt to detect game objects that contain a sprite, and write the sprite directly to the tilemap. This is considerably more efficient than inserting the entire game object into the tilemap, but you lose any other components.

## Writing to a Mesh

You can use `TesseraMeshOutput` to write directly to a mesh. Your tiles objects must have a `MeshFilter` and `MeshRenderer`, unless you check `Instantiate Children Only`, in which case this applies to the child objects of the tiles.

To use it, create an object with a MeshFilter and MeshRenderer. It doesn't need a mesh configured, but you should set the material(s) you want to use. Then add TesseraMeshOutput as a component to the generator, and set the target to the created object.

Tessera will automatically detect matching materials between the tiles and target object, and merge the meshes into submeshes to take advantage of it.

## Handling the output in code

When invoking the Generate method, you can set [onComplete](#) or [onComplete](#) to completely replace the the default behaviour with your own code. There's an example in the [API](#).

# Quality and Performance

Tessera uses [Wave Function Collapse](#) as its core algorithm, though with many custom enhancements. This is a constraint solving based approach to generation, and it is very powerful, but it comes with some limitations. The algorithm searches through the space of possible tile choices to find one that fits all the constraints and adjacency rules required. The fewer possible solutions there are, the longer it can take to find one. [Tile adjacency](#) is Turing complete, so there are inputs that will just take forever to generate.

This page gives some advice on how to get the most out of Tessera.

## Backtracking

Backtracking is a critical feature of most constraint solvers, though normally WFC doesn't feature it. With backtracking off, the generator will give up the moment it encounters any difficulty. With it on, it will instead undo the last tile generated, and try again with a different option.

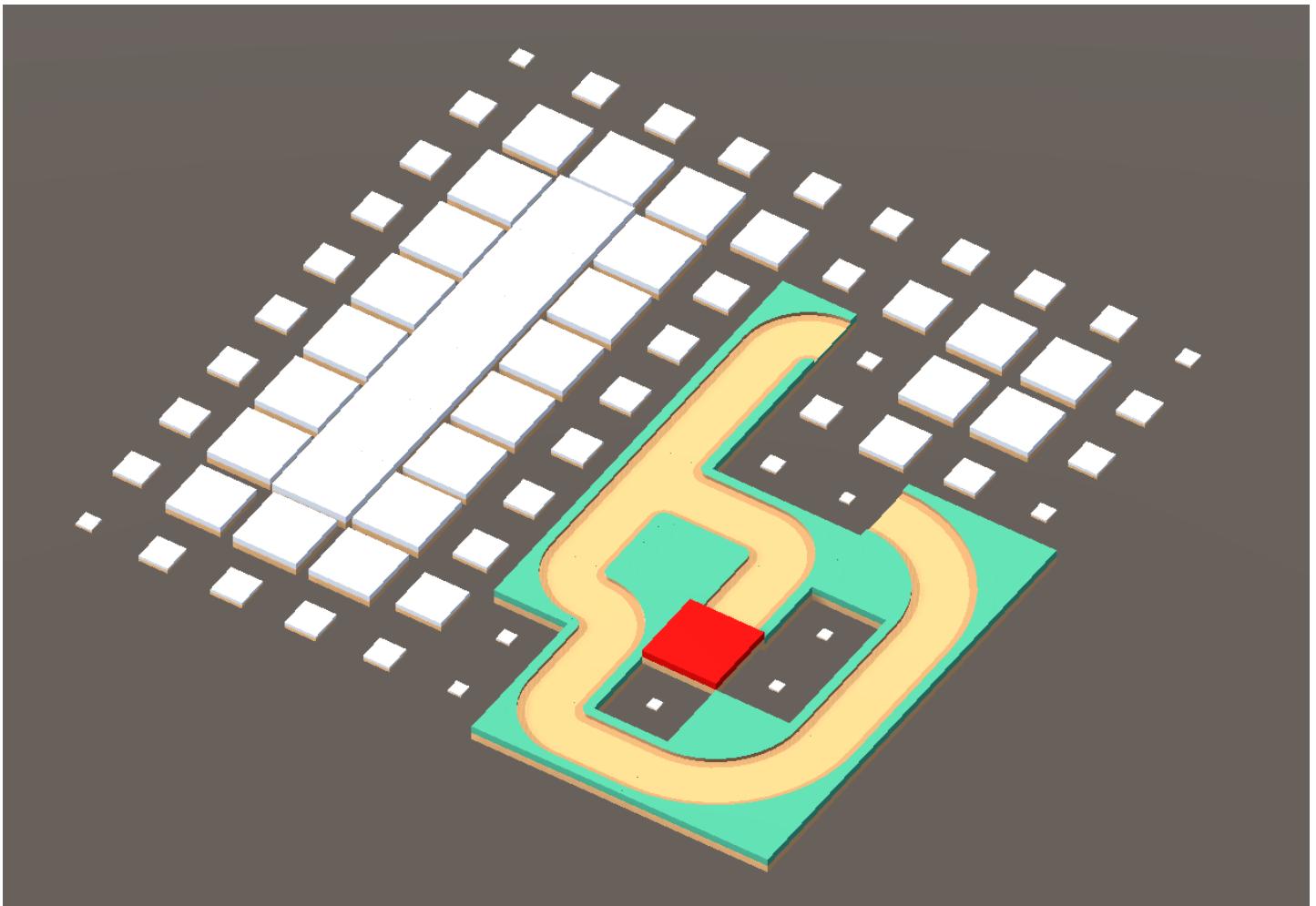
Enabling backtracking can therefore make the solver work much harder at finding a solution. The downside is that it can run *forever* looking for a solution, and can sometimes get stuck exploring possibilities that will never work.

To get the best of both worlds, you can introduce a [Step Limit](#). This will allow some backtracking to occur, but after a fixed amount of computation it will automatically retry with a fresh generation.

## Errors and contradictions

Not all setups for the generator have a solution at all. When the generator fails to find a solution, it'll log a warning detailing what has gone wrong.

To enable more diagnostics, set [Failure Mode](#) to Last, and add prefabs to Uncertainty Tile / Contradiction Tile. This will cause the generator to output a clearer picture of the problem.



The above picture shows a failed generation, with white tiles indicating what has yet to be generated, and the red tile indicating where the contradiction occurred. You will have to infer what the specific problem is. In the above case, the generator wasn't set up with a dead end tile, so there's no way for the path to complete.

Here are some other tips to help determine what is going wrong:

- You can use a fixed [seed](#) to make any issues reproducible
- You can select uncertainty tiles in the Inspector, they have some additional information attached.
- Try disabling the constraints (both initial constraints and generator constraints) until things start working.
- If you have no constraints, try removing tiles that are difficult to fit in, such as Big Tiles.
- Conversely, sometimes adding tiles can help. You can try adding a tile without any faces painted and with very low weight, which means it can be placed anywhere in the generator as a last resort.
- (Pro only) Use [AnimatedGenerator](#) to display the generation in realtime. This can be useful when backtracking is on, highlighting areas that cannot be successfully completed.

## Performance Tips

If you have very large levels, or a particularly hard to generate setup, then you may find Tessera's performance starts to become a problem. Here are some suggests to combat that.

### Disable "Record Undo"

This option is on by default, and affects TesseraGenerator when you run it by clicking the "Generate" button in the Inspector panel. Recording undo information is painfully slow in Unity, and will cause the Editor to freeze temporarily while creating all the generator tiles. This setting has no effect at runtime.

### Change generation algorithm.

Tessera's default [constraint resolution algorithm](#) is called "AC4", which resolves all constraints every step. You can instead try

"OneStep", which only resolves nearby constraints. It can be faster in some circumstances, but also has less lookahead so is more likely to fail or backtrack.

## Reduce the number of tiles

Doubling the number of tiles makes Tessera approximately 4x as slow, so it's worthwhile minimizing tiles. The easiest way of doing so is enabling the [symmetric](#) setting and avoiding similar/duplicate tiles.

## Path constraints

The [Path constraint](#) is very expensive compared with the rest of Tessera. Avoid if possible, for example, by pre-generating your path and using [pins](#) to make Tessera use it.

The path constraint does come with two experimental settings, [prioritize](#) and [parity](#) that improve search quality, so you may want to try enabling those.

# Mouse and Keyboard Shortcuts

All shortucts are used while in tile painting model.

KEY	DESCRIPTION
Click	Paint the currently selected paint color
Shift-Click	Erase the paint.
A	Toggle Show All Tiles
Shift-A	Toggle Hide All (sets opacity to zero)
X	Select the previously selected paint color

# Using the API

The other tutorials have shown you how to set up a tiles and a generator, without any coding. But now you need to hook up the generation to the rest of your level.

To do so, you need to call [Generate](#) or [StartGenerate](#). [Generate](#) will run synchronously, and return details about the generation. It's easier to use, but can cause noticeable stutter if you are doing a big generation. [StartGenerate](#) behaves exactly the same, but can be used from a Unity coroutine.

Because co-routines cannot return information, you can instead supply various callbacks using [TesseraGenerateOptions](#). Most commonly, you'll want to set [onCreate](#) to replace the behaviour for instantiating new tiles. The default behaviour instantiates them all at once, which can cause stutter.

```
using UnityEngine;
using Tessera;
using System.Collections;

public class MyBehaviour : MonoBehaviour
{
    private TesseraGenerator generator;

    void Start()
    {
        generator = GetComponent<TesseraGenerator>();
        StartCoroutine(MyCoro());
    }

    IEnumerator MyCoro()
    {
        var options = new TesseraGenerateOptions { onCreate = MyCreate };
        yield return generator.StartGenerate(options);
        // Any following code will be run after the generation
    }

    void MyCreate(TesseraTileInstance instance)
    {
        Debug.Log("Creating " + instance.Tile.gameObject.name);
        // Do the default behaviour
        TesseraGenerator.Instantiate(instance, generator.transform);
    }
}
```

Here's an example of overriding [onComplete](#), to so we can create the tiles one at a time rather than all at once:

```
using UnityEngine;
using Tessera;
using System.Collections;
using System.Collections.Generic;

public class MyBehaviour : MonoBehaviour
{
    private TesseraGenerator generator;

    void Start()
    {
        generator = GetComponent<TesseraGenerator>();
        StartCoroutine(MyCoro());
    }

    IEnumerator MyCoro()
    {
        IList<TesseraTileInstance> instances = null;
        var options = new TesseraGenerateOptions
        {
            onComplete = completion =>
            {
                if(completion.success)
                {
                    instances = completion.tileInstances;
                }
            }
        };
        yield return generator.StartGenerate(options);

        if(instances != null)
        {
            foreach(var instance in instances)
            {
                TesseraGenerator.Instantiate(instance, generator.transform);
                // Wait for next frame.
                yield return null;
            }
        }
    }
}
```

# Customization

If you have Tessera Pro, then you have the full source code of the project.

You can change this in any way you see fit, but here are some specific ideas you may find useful. Please note that we do not guarantee these will be stable with later versions of Tessera Pro.

## Customize the model.

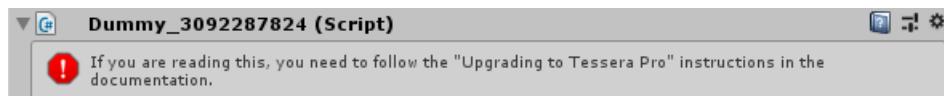
Tessera uses an open source library called DeBroglie for the actual generation process. DeBroglie is initialized in `TesseraGeneratorHelper.Setup`. It has many options that are not directly exposed by Tessera, which can be set by changing the code.

It is recommended you familiarize yourself with [DeBroglie's documentation](#) before trying this.

# Upgrading from Tessera to Tessera Pro

Unfortunately, it's not possible to preserve compatibility in Unity when replacing a .dll with a set of scripts.

If you previously developed a game with Tessera, and you have now downloaded Tessera Pro instead, you'll see the following error.



Don't panic. All your data is still there.

To fix it, please backup your project, then run `Tools > Tessera > Upgrade Scene to Tessera Pro`.

If you like, you can delete all the "dummy" files in the Tessera folder - these files only exist to simplify this transition.

## Advanced

If you don't like upgrading scene by scene, you can edit Unity's yaml directly. The following instructions replace the main script references.

Find: `fileID: -65694588, guid: 5ec9deea42ffdf94eae3261973878f98`

Replace: `fileID: 11500000, guid: e3ad2bf01b7a6b7409eb683402aa8669`

Find: `fileID: 2003858105, guid: 5ec9deea42ffdf94eae3261973878f98`

Replace: `fileID: 11500000, guid: 8a3f7e4cbfb5a184b8e397a0175d7112`

Find: `fileID: -96226770, guid: 5ec9deea42ffdf94eae3261973878f98`

Replace: `fileID: 11500000, guid: 333e56fb2e5d1ff4bb53c10611586ded`

Find: `fileID: 1044799892, guid: 5ec9deea42ffdf94eae3261973878f98`

Replace: `fileID: 11500000, guid: d1efb6dc65363b7479c2f8be4b856e61`

Find: `fileID: 1700055444, guid: 5ec9deea42ffdf94eae3261973878f98`

Replace: `fileID: 11500000, guid: eae741d319c3bff43839bad0f95dceca`

Find: `fileID: -758439564, guid: 5ec9deea42ffdf94eae3261973878f98`

Replace: `fileID: 11500000, guid: b6f7252aa33bd554b9de1f6d885c2d7d`

Save your changes, then reload the scene in Unity. If done correctly, the scripts should now work.

## Downgrading

Downgrades can be done by following the Advanced instructions, reversing Find and Replace.

# Release notes

## 5.2.0

- Fix errors with nesting Foldout Headers in later Unity versions.
- Added a separate build for Unity 2021.2+. This fixes various failures and warnings Unity generates (non-Pro only)
- Add [Border Constraint](#) for setting the tiles on the border of the generation.

## 5.1.1

- Fix Tiles list not displaying in recent Unity versions
- Fix for AnimatedGenerator not backtracking properly
- Improved Inspector GUI for AnimatedGenerator
- Fix issue with Path constraint in DeBroglie

## 5.1.0

- Added [Overlapping model support](#) (Pro only)
- Added samples:
  - [GrassPathsOverlapping](#) (Pro only)
  - [Plains](#) (Pro only)
- Generator output now available in new [tileData](#) format.
- Generator seed can now be set in the UI.
- AnimatedGenerator now respects stepCount.
- Fix issues with square tiles/grids
- Fix issue with pins and big tiles
- Fix hex/triangle vertical adjacencies
- Performance improvement for long tile lists in the Inspector

## 5.0.0

- **Moved files, you'll need to delete the Tessera directory and reload from scratch**
- Added [symmetry](#) option improving performance in some cases.
- Added [multipass generator](#)
- Added [infinite generator](#) (Pro only)
- Added [instantiate output](#) (Pro only)
- Reworked MirrorConstraint somewhat (Pro only)
- Warn if palette is set inconsistently.
- Added [sample docs](#) describing what demos are available.
- Added more samples:
  - [GrassPathsInfinite](#)
  - [GrassPathsWithPathConstraint](#)
  - [PlatformerInfinite](#)
  - [Skyscrapers](#)
  - [Truchet](#)
- Added some keyboard shortcuts, Shift-Click, A, Shift+A and X.
- Fix pin constraints for square cells.

## 4.3.0

- Fixed warnings about obsolete APIs (Pro only)
- Fixes for hexagon and cube rotations, causing misaligned tiles.
- Fix Tessera Bolt integration.
- Fix MirrorConstraint editor exceptions (Pro only)

## 4.2.0

- Added [TesseraSquareTile](#) and updated the Platformer sample to use it
- Added [failureMode](#) and other options for diagnosing issues.
- Fixed using AnimatedGenerator and TileOutput at the same time. (Pro only)
- Fixed Regenerate function in play mode
- Fixed exceptions when using masking.

## 4.1.1

- Fix stack overflow error when using Path constraint on large maps

## 4.1.0

- Add multipass sample and [tutorial](#)
- Add HexRaceway sample (Pro only)
- Added many PathConstraint options (Pro only):
  - [loop](#)
  - [acyclic](#)
  - [parity](#)
- Added many extra generation options:
  - [stepLimit](#)
  - [algorithm](#)
  - [recordUndo](#)
- Added notes on [quality and performance](#)
- Fixed issue with normals when generating on mesh surfaces (Pro only)

## 4.0.0

- [Hexagonal and triangular tiles are now supported.](#) (Pro only)
- [Triangle mesh surfaces now supported](#) (instead of just quad meshes) (Pro only)
- AnimatedGenerator has some support for multithreading (Pro only)
- Major internal refactoring
- Some performance improvements
- Worker thread now registers with Unity's profiler
- Improvements to provided samples
- Better support for Unity 2020
- Now easier to access the completion object after calling StartGenerate.
- Fixes some issues with Big Tiles on mesh surfaces (Pro only)
- Fix surface meshes reflecting every tile by default (Pro only)
- Removed TesseraGenerator.initialConstraints (use TesseraGeneratorOptions.initialConstraints)
- Fixed using `instantiateChildrenOnly` with `TesseraMeshOutput`

## 3.4.1

- Instantiated tiles now have names that include their cell location.

- Fix exception in PathConstraint when Prioritize is on.
- Fix Volumes
- Fix rare issue with big tiles on mesh surfaces.
- Warn if meshes are not readable and using Mesh Output.

## 3.4.0

- Performance improvements with pins and volumes
- Volumes can now be set to [MaskOut](#)
- Refactored GetInitialConstraint methods and TesseraVolumeFilter (breaking)
- Fixed exception with out of bounds palette indices
- Fixed interaction between big tiles and path constraint (Pro only)

## 3.3.0

- Pinned Tile Constraints now work on Surface Meshes (Pro only).
- Refactored some code, changed API of ITesseraTileOutput (Pro only) (breaking)
- Added "Fix It" for inconsistent tile sizes.
- Added [TesseraPinned](#) for more initial constraint options.
- Added extra samples
- Added "[Upgrade to Tessera Pro](#)" utility (Pro only)

## 3.2.0

- Tile rotations no longer confined to XZ plane
- Added tooltips to inspector
- [Mirror constraint](#) now supports on all 3 axes (Pro only)
- [Add support for Bolt](#)

## 3.1.2

- Fixed Instantiate Only Children to work with mesh deformations
- Fixed "BoxColliders do not support negative scale or size" warning.
- Fixed generation on mesh surfaces with multiple layers
- Volumes work with generation on mesh surfaces
- Improved Smooth Normals setting

## 3.1.1

- Asset can now be loaded in Unity 2019.1 and 2019.2
- Fixed the missing script in the Castle sample

## 3.1.0

- Add [Tessera Volume](#)
- Dungeon example now demos Volumes
- WebGL builds now ignore the multithreaded option
- Can now choose whether [normals should be smooth](#) or not when generating on a surface (Pro only)
- Can now [filter tiles per-submesh](#) when generating on a surface (Pro only)
- Add [Prioritize](#) option to PathConstraint (Pro only)
- Add [Separation constraint](#) (Pro only)
- Added City example to Samples (Pro only)

- Fixed bug in AnimatedGenerator
- Fixed bug when normal smoothing

## 3.0.0

- Various api changes, larger internal refactorings (breaking)
- Support [generation on surface of a mesh](#) (Pro only)
- Paint tools no longer switches off when you click children of tiles
- Add [Clear](#) and Regenerate methods
- Improved validation messages
- Fixed exception for Mesh Output when materials are missing

## 2.3.0

- Tessera Palette now serializes correctly
- Fix some Inspector display glitches in Unity 2019.3
- Mesh output can now be animated (Pro only)

## 2.2.0

- [Generation can now be animated](#) (Pro only)
- [Tilemap output](#) (Pro only)
- [Mesh output](#) (Pro only)
- Added "Show all" view option when painting.

## 2.1.0

- Added a palette asset that lets you:
  - Customize the paint colors Tessera uses
  - Name the colors (shows in tooltips)
  - Control what colors match each other
- Added a new sample, Platformer
- Fixed a bug that prevented the use of big tiles as fixed tile constraints
- A warning is now emitted if inconsistent tileSizeS are used
- Multithreading can now be disabled, for platforms that don't support it.

## v2.0.0

- Some performance improvements
- Visible source code (Pro only)
- Added [CountConstraint](#) (Pro only)
- Added [MirrorConstraint](#) (Pro only)
- Added [PathConstraint](#) (Pro only)
- Seeds have changed (breaking)
- Removed [defaultParent](#) (breaking)
- Added a new sample, Dungeon.

## v1.1.1

- Fixed issue with reflected tiles using incorrect rotation for bottom face

## v1.1.0

- Fixed "BeginLayoutGroup must be called first" errors.
- Fixed issue with rotated initial tile constraints.
- Fixed a display glitch in orthographic views
- Added keyboard shortcuts:
  - Delete to remove tiles from the generators list.
  - Z to toggle backfaces.
- Added another scene to samples.
- Improved [documentation on constraints](#).
- Added [contradictionLocation](#)

## v1.0.1

- Removed a Debug.Log line
- [Random seed](#) can now be set. Default from Unity.Random.
- "Clear Children" button on Generator component.
- Fix spurious exceptions when calling Generate.

## v1.0.0

- Initial release

# Namespace Tessera

## Classes

### [AnimatedGenerator](#)

Attach this to a TesseraGenerator to run the generator stepwise over several updates, displaying the changes so far.

#### NOTE

This class is available only in Tessera Pro

### [BiMap<U, V>](#)

Represents a 1:1 mapping between two types

### [BorderConstraint](#)

Forces cells near the edge to be a particular tile. Compare with [skyBox](#).

#### NOTE

This class is available only in Tessera Pro

### [BorderItem](#)

### [CountConstraint](#)

Keeps track of the number of tiles in a given set, and ensure it is less than / more than a given number.

#### NOTE

This class is available only in Tessera Pro

### [CubeCellType](#)

### [CubeFaceDirExtensions](#)

### [DefaultCellTypeImpl](#)

### [DefaultGridImpl](#)

### [EnumeratorWithResult<T>](#)

An IEnumerator that also records a given result when it is finished. It is intended for use with Unity coroutines.

### [FaceDetails](#)

Records the painted colors for a single face of one cube in a [TesseraTile](#)

### [GridSymmetry](#)

### [HexGeometryUtils](#)

#### NOTE

This class is available only in Tessera Pro

### [HexPrismCellType](#)

### **i** NOTE

This class is available only in Tessera Pro

## [HexPrismFaceDirExtensions](#)

### **i** NOTE

This class is available only in Tessera Pro

## [InfiniteGenerator](#)

## [InstantiateOutput](#)

## [MeshData](#)

A replacement for UnityEngine.Mesh that stores all the data in memory, for fast access from C#.

### **i** NOTE

This class is available only in Tessera Pro

## [MeshDeformation](#)

Encapsulates an arbitrary deformation of mesh vertices

### **i** NOTE

This class is available only in Tessera Pro

## [MeshUtils](#)

Utility for working with meshes.

### **i** NOTE

This class is available only in Tessera Pro

## [MirrorConstraint](#)

Ensures that the generation is symmetric when x-axis mirrored. If there are any tile constraints, they will not be mirrored.

### **i** NOTE

This class is available only in Tessera Pro

## [PaletteEntry](#)

## [PathConstraint](#)

Forces a network of tiles to connect with each other, so there is always a complete path between them. Two tiles connect along the path if:

- Both tiles are in [pathTiles](#) (if [hasPathTiles](#) set); and
- The central color of the sides of the tiles leading to each other are in [pathColors](#) (if [pathColors](#) set)

### **NOTE**

This class is available only in Tessera Pro

## [PrefixLookup<T>](#)

Stores key-value pairs, with efficient searching for the longest key that is a prefix of a given string.

## [QuadInterpolation](#)

## [SeparationConstraint](#)

## [SquareCellType](#)

## [SquareFaceDirExtensions](#)

## [TesseraCompletion](#)

Returned by [TesseraGenerator](#) after generation finishes

## [TesseraConstraint](#)

Abstract class for all generator constraint components.

### **NOTE**

This class is available only in Tessera Pro

## [TesseraGenerateOptions](#)

Additional settings to customize the generation at runtime.

## [TesseraGenerator](#)

GameObjects with this behaviour contain utilities to generate tile based levels using Wave Function Collapse (WFC). Call [Generate\(TesseraGenerateOptions\)](#) or [StartGenerate\(TesseraGenerateOptions\)](#) to run. The generation takes the following steps:

- Inspect the tiles in [tiles](#) and work out how they rotate and connect to each other.
- Setup any initial constraints that fix parts of the generation ([initialConstraints](#)).
- Fix the boundary of the generation if [skyBox](#) is set.
- Generate a set of tile instances that fits the above tiles and constraints.
- Optionally [retries](#) or [backtrack](#).
- Instantiates the tile instances.

## [TesseraHexTile](#)

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

## [TesseraInitialConstraint](#)

Initial constraint objects fix parts of the generation process in places. Use the utility methods on [TesseraGenerator](#) to create these objects.

## [TesseraInitialConstraintBuilder](#)

## [TesseraInstantiateOutput](#)

Attach this to a [TesseraGenerator](#) to control how tiles are instantiated.

### **NOTE**

This class is available only in Tessera Pro

## TesseraMeshOutput

Attach this to a TesseraGenerator to output the tiles to a single mesh instead of instantiating them.

### **NOTE**

This class is available only in Tessera Pro

## TesseraMultipassGenerator

### TesseraMultipassPass

### TesseraPalette

### TesseraPinConstraint

### TesseraPinned

### TesseraSquareTile

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

### TesseraTile

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

### TesseraTileBase

### TesseraTileInstance

Represents a request to instantiate a TesseraTile, post generation.

### TesseraTilemapOutput

Attach this to a TesseraGenerator to output the tiles to a Unity Tilemap component instead of directly instantiating them.

### **NOTE**

This class is available only in Tessera Pro

## TesseraTransformedTile

Defines a Unity tile that has a specific transform applied to it. Used by [TesseraTilemapOutput](#)

### TesseraTrianglePrismTile

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

### TesseraVolume

### TesseraVolumeFilter

### TileEntry

Specifies a tile to be used by [TesseraGenerator](#)

### TileList

### TileMapping

### TriangleInterpolation

## [TrianglePrismCellType](#)

### **NOTE**

This class is available only in Tessera Pro

## [TrianglePrismFaceDirExtensions](#)

### **NOTE**

This class is available only in Tessera Pro

## [TrianglePrismGeometryUtils](#)

### **NOTE**

This class is available only in Tessera Pro

## [TRS](#)

Represents a position / rotation and scale. Much like a Transform, but without the association with a unity object.

## [UnityEngineInterface](#)

### Structs

#### [CubeRotation](#)

#### [HexRotation](#)

Represents rotations / reflections of a hexagon

### **NOTE**

This class is available only in Tessera Pro

## [ModelTile](#)

Actual tiles used internally. There's a many-to-one relationship between ModelTile and TesseraTile due to rotations and "big" tile support.

## [OrientedFace](#)

Records the painted colors and location of single face of one cube in a [TesseraTile](#)

## [SquareRotation](#)

Represents rotations / reflections of a square

## [TriangleRotation](#)

Represents rotations / reflections of a hexagon

### **NOTE**

This class is available only in Tessera Pro

## Interfaces

## [ICellType](#)

## [IEngineInterface](#)

### [IGrid](#)

Represents a arrangement of cells, including their adjacency and locations. Cells are uniquely identified by a Vector3Int. Tessera.IGrid is roughly equivalent to DeBroglie.Topo.ITopology.

### [ITesseraInitialConstraint](#)

### [ITesseraTileOutput](#)

### Enums

#### [AnimatedGenerator.AnimatedGeneratorState](#)

#### [CellFaceDir](#)

Represents a particular face of a generic cell. The enum is empty - to work with directions, you need to either:

- Use the methods on [ICellType](#).
- Cast to the enum specific to a given cell type, e.g. [CubeFaceDir](#).

#### [CellRotation](#)

#### [ChunkCleanupType](#)

#### [CubeFaceDir](#)

Enum of the 6 faces on a cube.

#### [FailureMode](#)

#### [HexPrismFaceDir](#)

Enum of the 8 faces on a hex prism.

#### **NOTE**

This class is available only in Tessera Pro

#### [MirrorConstraint.Axis](#)

#### [ModelType](#)

Different models Tessera supports. The model dictates how nearby tiles relate to each other.

#### [PinType](#)

#### [RotationGroupType](#)

#### [SquareFaceDir](#)

Enum of the 4 sides of a square.

#### [TesseraMultipassPassType](#)

#### [TesseraWfcAlgorithm](#)

#### [TrianglePrismFaceDir](#)

#### **NOTE**

This class is available only in Tessera Pro

#### [VolumeType](#)

# Class AnimatedGenerator

Attach this to a TesseraGenerator to run the generator stepwise over several updates, displaying the changes so far.

## **NOTE**

This class is available only in Tessera Pro

Inheritance

[Object](#)

[AnimatedGenerator](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class AnimatedGenerator : MonoBehaviour
```

Fields

**multithread**

If true, use threading to avoid stalling Unity. (ignored on WebGL builds)

Declaration

```
public bool multithread
```

Field Value

TYPE	DESCRIPTION
Boolean	

**progressPerStep**

Declaration

```
public float progressPerStep
```

Field Value

TYPE	DESCRIPTION
Single	

**scaleUncertaintyTile**

If true, the uncertainty tiles shrink as the solver gets more certain.

Declaration

```
public bool scaleUncertaintyTile
```

Field Value

TYPE	DESCRIPTION
Boolean	

## secondsPerStep

Declaration

```
public float secondsPerStep
```

Field Value

TYPE	DESCRIPTION
Single	

## uncertaintyTile

Game object to show in cells that have yet to be fully solved.

Declaration

```
public GameObject uncertaintyTile
```

Field Value

TYPE	DESCRIPTION
GameObject	

## Properties

### IsStarted

Declaration

```
public bool IsStarted { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

### State

Declaration

```
public AnimatedGenerator.AnimatedGeneratorState State { get; }
```

Property Value

TYPE	DESCRIPTION
AnimatedGenerator.AnimatedGeneratorState	

## Methods

### PauseGeneration()

Declaration

```
public void PauseGeneration()
```

### ResumeGeneration()

Declaration

```
public void ResumeGeneration()
```

## StartGeneration()

Declaration

```
public void StartGeneration()
```

## Step()

Declaration

```
public void Step()
```

## StopGeneration()

Declaration

```
public void StopGeneration()
```

# Enum AnimatedGenerator.AnimatedGeneratorState

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum AnimatedGeneratorState
```

Fields

NAME	DESCRIPTION
Initializing	
Paused	
Running	
Stopped	

# Class BiMap<U, V>

Represents a 1:1 mapping between two types

Inheritance

[Object](#)

[BiMap<U, V>](#)

Implements

[IEnumerable<\(T1, T2\)<U, V>>](#)

[IEnumerable](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class BiMap<U, V> : IEnumerable<(U, V)>, IEnumerable
```

Type Parameters

NAME	DESCRIPTION
U	
V	

Constructors

[BiMap\(IEnumerable<\(U, V\)>\)](#)

Declaration

```
public BiMap(IEnumerable<(U, V)> data)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">IEnumerable&lt;(T1, T2)&lt;U, V&gt;&gt;</a>	data	

Properties

Count

Declaration

```
public int Count { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">Int32</a>	

Item[U]

Declaration

```
public V this[U u] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
U	u	

Property Value

TYPE	DESCRIPTION
V	

## Item[V]

Declaration

```
public U this[V v] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
V	v	

Property Value

TYPE	DESCRIPTION
U	

## Methods

### GetEnumerator()

Declaration

```
public IEnumerator<(U, V)> GetEnumerator()
```

Returns

TYPE	DESCRIPTION
IEnumerator<(T1, T2)<U, V>>	

## Explicit Interface Implementations

### IEnumerable.GetEnumerator()

Declaration

```
IEnumerator IEnumerable.GetEnumerator()
```

Returns

TYPE	DESCRIPTION
IEnumerator	

## Implements

System.Collections.Generic.IEnumerable<T>

System.Collections.IEnumerable

# Class BorderConstraint

Forces cells near the edge to be a particular tile. Compare with [skyBox](#).

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

[TesseraConstraint](#)

[BorderConstraint](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class BorderConstraint : TesseraConstraint
```

Fields

[borders](#)

Declaration

```
public BorderItem[] borders
```

Field Value

TYPE	DESCRIPTION
<a href="#">BorderItem</a> []	

# Class BorderItem

Inheritance

[Object](#)

BorderItem

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class BorderItem
```

Fields

cellFaceDir

Declaration

```
public CellFaceDir cellFaceDir
```

Field Value

TYPE	DESCRIPTION
<a href="#">CellFaceDir</a>	

tile

Declaration

```
public TesseraTileBase tile
```

Field Value

TYPE	DESCRIPTION
<a href="#">TesseraTileBase</a>	

# Enum CellFaceDir

Represents a particular face of a generic cell. The enum is empty - to work with directions, you need to either:

- Use the methods on [ICellType](#).
- Cast to the enum specific to a given cell type, e.g. [CubeFaceDir](#).

Namespace: [Tessera](#)

Assembly: cs.temp.dll

Syntax

```
public enum CellFaceDir
```

# Enum CellRotation

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum CellRotation
```

# Enum ChunkCleanupType

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

## Syntax

```
public enum ChunkCleanupType
```

## Fields

NAME	DESCRIPTION
Full	Remove everything associated with the chunk.
Memoize	Remove the GameObjects, but keep tile data so they can be recreated exactly
None	Never cleanup chunks

# Class CountConstraint

Keeps track of the number of tiles in a given set, and ensure it is less than / more than a given number.

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

[TesseraConstraint](#)

[CountConstraint](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class CountConstraint : TesseraConstraint
```

Fields

[comparison](#)

How to compare the count of [tiles](#) to [count](#).

Declaration

```
public CountComparison comparison
```

Field Value

TYPE	DESCRIPTION
CountComparison	

[count](#)

The count to be compared against.

Declaration

```
public int count
```

Field Value

TYPE	DESCRIPTION
Int32	

[eager](#)

If set, this constraint will attempt to pick tiles as early as possible. This can give a better random distribution, but higher chance of contradictions.

Declaration

```
public bool eager
```

Field Value

TYPE	DESCRIPTION
Boolean	

## tiles

The set of tiles to count

Declaration

```
public List<TesseraTileBase> tiles
```

## Field Value

TYPE	DESCRIPTION
List<TesseraTileBase>	

# Class CubeCellType

Inheritance

[Object](#)

CubeCellType

Implements

[ICellType](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class CubeCellType : ICellType
```

Properties

Instance

Declaration

```
public static CubeCellType Instance { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">CubeCellType</a>	

Methods

[FindPath\(Vector3Int, Vector3Int\)](#)

Declaration

```
public IEnumerable<CellFaceDir> FindPath(Vector3Int startOffset, Vector3Int endOffset)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">Vector3Int</a>	startOffset	
<a href="#">Vector3Int</a>	endOffset	

Returns

TYPE	DESCRIPTION
<a href="#">IEnumerable&lt;CellFaceDir&gt;</a>	

[GetCellCenter\(Vector3Int, Vector3, Vector3\)](#)

Declaration

```
public Vector3 GetCellCenter(Vector3Int offset, Vector3 center, Vector3 tileSize)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
Vector3	center	
Vector3	tileSize	

Returns

TYPE	DESCRIPTION
Vector3	

### GetDisplayName(CellFaceDir)

Declaration

```
public string GetDisplayName(CellFaceDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
CellFaceDir	dir	

Returns

TYPE	DESCRIPTION
String	

### GetFaceDirPairs()

Declaration

```
public IEnumerable<(CellFaceDir, CellFaceDir> GetFaceDirPairs()
```

Returns

TYPE	DESCRIPTION
IEnumerable<(T1, T2)<CellFaceDir, CellFaceDir>>	

### GetFaceDirs()

Declaration

```
public IEnumerable<CellFaceDir> GetFaceDirs()
```

Returns

TYPE	DESCRIPTION
IEnumerable<CellFaceDir>	

### GetIdentity()

Declaration

```
public CellRotation GetIdentity()
```

Returns

TYPE	DESCRIPTION
CellRotation	

## GetMatrix(CellRotation)

Declaration

```
public Matrix4x4 GetMatrix(CellRotation cellRotation)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	cellRotation	

Returns

TYPE	DESCRIPTION
Matrix4x4	

## GetRotations(Boolean, Boolean, RotationGroupType)

Declaration

```
public IList<CellRotation> GetRotations(bool rotatable, bool reflectable, RotationGroupType rotationGroupType)
```

Parameters

TYPE	NAME	DESCRIPTION
Boolean	rotatable	
Boolean	reflectable	
RotationGroupType	rotationGroupType	

Returns

TYPE	DESCRIPTION
IList<CellRotation>	

## Invert(CellFaceDir)

Declaration

```
public CellFaceDir Invert(CellFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
CellFaceDir	faceDir	

Returns

TYPE	DESCRIPTION
CellFaceDir	

## Invert(CellRotation)

Declaration

```
public CellRotation Invert(CellRotation a)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	a	

Returns

TYPE	DESCRIPTION
CellRotation	

## Multiply(CellRotation, CellRotation)

Declaration

```
public CellRotation Multiply(CellRotation a, CellRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	a	
CellRotation	b	

Returns

TYPE	DESCRIPTION
CellRotation	

## Realign(ISet<Vector3Int>, CellRotation)

Declaration

```
public IDictionary<Vector3Int, Vector3Int> Realign(ISet<Vector3Int> shape, CellRotation rotation)
```

Parameters

TYPE	NAME	DESCRIPTION
ISet<Vector3Int>	shape	
CellRotation	rotation	

Returns

TYPE	DESCRIPTION
<a href="#">IDictionary&lt;Vector3Int, Vector3Int&gt;</a>	

## Rotate(CellFaceDir, CellRotation)

### Declaration

```
public CellFaceDir Rotate(CellFaceDir faceDir, CellRotation rotation)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">CellFaceDir</a>	faceDir	
<a href="#">CellRotation</a>	rotation	

### Returns

TYPE	DESCRIPTION
<a href="#">CellFaceDir</a>	

## RotateBy(CellFaceDir, FaceDetails, CellRotation)

### Declaration

```
public (CellFaceDir, FaceDetails) RotateBy(CellFaceDir faceDir, FaceDetails faceDetails, CellRotation rot)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">CellFaceDir</a>	faceDir	
<a href="#">FaceDetails</a>	faceDetails	
<a href="#">CellRotation</a>	rot	

### Returns

TYPE	DESCRIPTION
<a href="#">(T1, T2)&lt;CellFaceDir, FaceDetails&gt;</a>	

## TryMove(Vector3Int, CellFaceDir, out Vector3Int)

### Declaration

```
public bool TryMove(Vector3Int offset, CellFaceDir dir, out Vector3Int dest)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">Vector3Int</a>	offset	

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
<a href="#">CellFaceDir</a>	dir	
<a href="#">Vector3Int</a>	dest	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
<a href="#">Boolean</a>	

**TryMoveByOffset(Vector3Int, Vector3Int, Vector3Int, CellRotation, out Vector3Int)**

Declaration

```
public bool TryMoveByOffset(Vector3Int startCell, Vector3Int startOffset, Vector3Int destOffset, CellRotation rotation, out Vector3Int destCell)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
<a href="#">Vector3Int</a>	startCell	
<a href="#">Vector3Int</a>	startOffset	
<a href="#">Vector3Int</a>	destOffset	
<a href="#">CellRotation</a>	rotation	
<a href="#">Vector3Int</a>	destCell	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
<a href="#">Boolean</a>	

Implements

[ICellType](#)

# Enum CubeFaceDir

Enum of the 6 faces on a cube.

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum CubeFaceDir
```

Fields

NAME	DESCRIPTION
Back	
Down	
Forward	
Left	
Right	
Up	

# Class CubeFaceDirExtensions

Inheritance

[Object](#)

CubeFaceDirExtensions

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public static class CubeFaceDirExtensions
```

## Methods

[Forward\(CubeFaceDir\)](#)

Declaration

```
public static Vector3Int Forward(this CubeFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">CubeFaceDir</a>	faceDir	

Returns

TYPE	DESCRIPTION
<a href="#">Vector3Int</a>	The normal vector for a given face.

[Inverted\(CubeFaceDir\)](#)

Declaration

```
public static CubeFaceDir Inverted(this CubeFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">CubeFaceDir</a>	faceDir	

Returns

TYPE	DESCRIPTION
<a href="#">CubeFaceDir</a>	Returns the face dir with the opposite normal vector.

[Up\(CubeFaceDir\)](#)

Declaration

```
public static Vector3Int Up(this CubeFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeFaceDir	faceDir	

Returns

TYPE	DESCRIPTION
Vector3Int	Returns (0, 1, 0) vector for most faces, and returns (0, 0, 1) for the top/bottom faces.

# Struct CubeRotation

Inherited Members

[ValueType.ToString\(\)](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public struct CubeRotation
```

Properties

All

Declaration

```
public static readonly IEnumerable<CubeRotation> All { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<CubeRotation>	

Identity

Declaration

```
public static readonly CubeRotation Identity { get; }
```

Property Value

TYPE	DESCRIPTION
CubeRotation	

IsReflection

Declaration

```
public readonly bool IsReflection { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

ReflectX

Declaration

```
public static readonly CubeRotation ReflectX { get; }
```

Property Value

TYPE	DESCRIPTION
CubeRotation	

## ReflectY

### Declaration

```
public static readonly CubeRotation ReflectY { get; }
```

### Property Value

TYPE	DESCRIPTION
CubeRotation	

## ReflectZ

### Declaration

```
public static readonly CubeRotation ReflectZ { get; }
```

### Property Value

TYPE	DESCRIPTION
CubeRotation	

## RotateXY

### Declaration

```
public static readonly CubeRotation RotateXY { get; }
```

### Property Value

TYPE	DESCRIPTION
CubeRotation	

## RotateXZ

### Declaration

```
public static readonly CubeRotation RotateXZ { get; }
```

### Property Value

TYPE	DESCRIPTION
CubeRotation	

## RotateYZ

### Declaration

```
public static readonly CubeRotation RotateYZ { get; }
```

### Property Value

TYPE	DESCRIPTION
CubeRotation	

## Methods

## Equals(Object)

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[ValueType.Equals\(Object\)](#)

## GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

[ValueType.GetHashCode\(\)](#)

## Invert()

Declaration

```
public CubeRotation Invert()
```

Returns

TYPE	DESCRIPTION
CubeRotation	

## Operators

### Equality(CubeRotation, CubeRotation)

Declaration

```
public static bool operator ==(CubeRotation a, CubeRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	a	

TYPE	NAME	DESCRIPTION
CubeRotation	b	

Returns

TYPE	DESCRIPTION
Boolean	

## Implicit(CellRotation to CubeRotation)

Declaration

```
public static implicit operator CubeRotation(CellRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	r	

Returns

TYPE	DESCRIPTION
CubeRotation	

## Implicit(CubeRotation to CellRotation)

Declaration

```
public static implicit operator CellRotation(CubeRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	r	

Returns

TYPE	DESCRIPTION
CellRotation	

## Inequality(CubeRotation, CubeRotation)

Declaration

```
public static bool operator !=(CubeRotation a, CubeRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	a	
CubeRotation	b	

Returns

TYPE	DESCRIPTION
Boolean	

## Multiply(CubeRotation, BoundsInt)

Declaration

```
public static BoundsInt operator *(CubeRotation r, BoundsInt bounds)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	r	
BoundsInt	bounds	

Returns

TYPE	DESCRIPTION
BoundsInt	

## Multiply(CubeRotation, CubeFaceDir)

Declaration

```
public static CubeFaceDir operator *(CubeRotation r, CubeFaceDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	r	
CubeFaceDir	dir	

Returns

TYPE	DESCRIPTION
CubeFaceDir	

## Multiply(CubeRotation, CubeRotation)

Declaration

```
public static CubeRotation operator *(CubeRotation a, CubeRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	a	
CubeRotation	b	

Returns

TYPE	DESCRIPTION
CubeRotation	

## Multiply(CubeRotation, Vector3)

Declaration

```
public static Vector3 operator *(CubeRotation r, Vector3 v)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	r	
Vector3	v	

Returns

TYPE	DESCRIPTION
Vector3	

## Multiply(CubeRotation, Vector3Int)

Declaration

```
public static Vector3Int operator *(CubeRotation r, Vector3Int v)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	r	
Vector3Int	v	

Returns

TYPE	DESCRIPTION
Vector3Int	

# Class DefaultCellTypeImpl

## Inheritance

[Object](#)

DefaultCellTypeImpl

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

## Syntax

```
public static class DefaultCellTypeImpl
```

## Methods

[Realign\(ICellType, ISet<Vector3Int>, CellRotation\)](#)

### Declaration

```
public static IDictionary<Vector3Int, Vector3Int> Realign(ICellType cellType, ISet<Vector3Int> shape,  
CellRotation rotation)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">ICellType</a>	cellType	
<a href="#">ISet&lt;Vector3Int&gt;</a>	shape	
<a href="#">CellRotation</a>	rotation	

### Returns

TYPE	DESCRIPTION
<a href="#">IDictionary&lt;Vector3Int, Vector3Int&gt;</a>	

[TryMoveByOffset\(ICellType, Vector3Int, Vector3Int, Vector3Int, CellRotation, out Vector3Int\)](#)

### Declaration

```
public static bool TryMoveByOffset(ICellType cellType, Vector3Int startCell, Vector3Int startOffset,  
Vector3Int destOffset, CellRotation rotation, out Vector3Int destCell)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">ICellType</a>	cellType	
<a href="#">Vector3Int</a>	startCell	
<a href="#">Vector3Int</a>	startOffset	
<a href="#">Vector3Int</a>	destOffset	
<a href="#">CellRotation</a>	rotation	

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Vector3Int	destCell	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Boolean	

# Class DefaultGridImpl

Inheritance

[Object](#)

DefaultGridImpl

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public static class DefaultGridImpl
```

## Methods

[TryMoveByOffset\(IGrid, Vector3Int, Vector3Int, Vector3Int, CellRotation, out Vector3Int, out CellRotation\)](#)

Declaration

```
public static bool TryMoveByOffset(IGrid grid, Vector3Int startCell, Vector3Int startOffset, Vector3Int destOffset, CellRotation startRotation, out Vector3Int destCell, out CellRotation destRotation)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">IGrid</a>	grid	
<a href="#">Vector3Int</a>	startCell	
<a href="#">Vector3Int</a>	startOffset	
<a href="#">Vector3Int</a>	destOffset	
<a href="#">CellRotation</a>	startRotation	
<a href="#">Vector3Int</a>	destCell	
<a href="#">CellRotation</a>	destRotation	

Returns

TYPE	DESCRIPTION
<a href="#">Boolean</a>	

# Class EnumeratorWithResult<T>

An IEnumerator that also records a given result when it is finished. It is intended for use with Unity coroutines.

Inheritance

[Object](#)

[IEnumeratorWithResult<T>](#)

Implements

[IEnumerator](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class EnumeratorWithResult<T> : IEnumerator
```

Type Parameters

NAME	DESCRIPTION
T	

Constructors

[EnumeratorWithResult\(IEnumerator\)](#)

Declaration

```
public EnumeratorWithResult(IEnumerator e)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">IEnumerator</a>	e	

Properties

[Current](#)

Declaration

```
public object Current { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">Object</a>	

[Result](#)

The value returned by this enumerator. This will throw if you attempt to access it before fully iterating through the enumerator.

Declaration

```
public T Result { get; }
```

Property Value

TYPE	DESCRIPTION
T	

## Methods

### MoveNext()

Declaration

```
public bool MoveNext()
```

Returns

TYPE	DESCRIPTION
Boolean	

### Reset()

Declaration

```
public void Reset()
```

### TryGetResult(out T)

The value returned by this enumerator. This will return false if you attempt to access it before fully iterating through the enumerator.

Declaration

```
public bool TryGetResult(out T result)
```

Parameters

TYPE	NAME	DESCRIPTION
T	result	

Returns

TYPE	DESCRIPTION
Boolean	

## Implements

[System.Collections.IEnumerator](#)

# Class FaceDetails

Records the painted colors for a single face of one cube in a [TesseraTile](#)

Inheritance

[Object](#)

[FaceDetails](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]  
public class FaceDetails
```

Fields

**bottom**

Declaration

```
public int bottom
```

Field Value

TYPE	DESCRIPTION
Int32	

**bottomLeft**

Declaration

```
public int bottomLeft
```

Field Value

TYPE	DESCRIPTION
Int32	

**bottomRight**

Declaration

```
public int bottomRight
```

Field Value

TYPE	DESCRIPTION
Int32	

**center**

Declaration

```
public int center
```

Field Value

TYPE	DESCRIPTION
Int32	

## hexBottomRightAndRight

Declaration

```
public int hexBottomRightAndRight
```

Field Value

TYPE	DESCRIPTION
Int32	

## hexLeftAndBottomLeft

Declaration

```
public int hexLeftAndBottomLeft
```

Field Value

TYPE	DESCRIPTION
Int32	

## hexRightAndTopRight

Declaration

```
public int hexRightAndTopRight
```

Field Value

TYPE	DESCRIPTION
Int32	

## hexTopLeftAndLeft

Declaration

```
public int hexTopLeftAndLeft
```

Field Value

TYPE	DESCRIPTION
Int32	

## left

Declaration

```
public int left
```

Field Value

TYPE	DESCRIPTION
Int32	

## right

Declaration

```
public int right
```

Field Value

TYPE	DESCRIPTION
Int32	

## top

Declaration

```
public int top
```

Field Value

TYPE	DESCRIPTION
Int32	

## topLeft

Declaration

```
public int topLeft
```

Field Value

TYPE	DESCRIPTION
Int32	

## topRight

Declaration

```
public int topRight
```

Field Value

TYPE	DESCRIPTION
Int32	

## Properties

### hexBottomLeft

Declaration

```
public int hexBottomLeft { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

## hexBottomLeftAndBottomRight

Declaration

```
public int hexBottomLeftAndBottomRight { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

## hexBottomRight

Declaration

```
public int hexBottomRight { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

## hexCenter

Declaration

```
public int hexCenter { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

## hexLeft

Declaration

```
public int hexLeft { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

## hexRight

Declaration

```
public int hexRight { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

## hexTopLeft

Declaration

```
public int hexTopLeft { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

## hexTopRight

Declaration

```
public int hexTopRight { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

## hexTopRightAndTopLeft

Declaration

```
public int hexTopRightAndTopLeft { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

## Methods

### IsEquivalent(FaceDetails)

Checks if two FaceDetails have the same values. This is an exact match, with no reflection built in. See TesseraPalette.Match for a fuzzier match.

Declaration

```
public bool IsEquivalent(FaceDetails other)
```

Parameters

TYPE	NAME	DESCRIPTION
FaceDetails	other	

Returns

TYPE	DESCRIPTION
Boolean	

## ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

[Object.ToString\(\)](#)

# Enum FailureMode

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

## Syntax

```
public enum FailureMode
```

## Fields

NAME	DESCRIPTION
Cancel	If a failure occurs, don't output anything
Last	If a failure occurs, output the progress so far
LastGood	If a failure occurs, backtrack to the last safe point.
Minimal	Examines the progress so far for the minimal set of tiles that cause an issue

# Class GridSymmetry

Inheritance

[Object](#)

[GridSymmetry](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class GridSymmetry
```

Fields

**rotation**

Declaration

```
public CellRotation rotation
```

Field Value

TYPE	DESCRIPTION
<a href="#">CellRotation</a>	

**translation**

Declaration

```
public Vector3Int translation
```

Field Value

TYPE	DESCRIPTION
<a href="#">Vector3Int</a>	

# Class HexGeometryUtils

## NOTE

This class is available only in Tessera Pro

### Inheritance

#### Object

#### HexGeometryUtils

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

### Syntax

```
public static class HexGeometryUtils
```

### Methods

#### CubeRotate(HexRotation, Vector3Int)

##### Declaration

```
public static Vector3Int CubeRotate(HexRotation rotation, Vector3Int cc)
```

##### Parameters

TYPE	NAME	DESCRIPTION
HexRotation	rotation	
Vector3Int	cc	

##### Returns

TYPE	DESCRIPTION
Vector3Int	

#### FromCubeCords(Vector3Int, Int32)

##### Declaration

```
public static Vector3Int FromCubeCords(Vector3Int cc, int y = 0)
```

##### Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cc	
Int32	y	

##### Returns

TYPE	DESCRIPTION
Vector3Int	

#### FromSide(Int32)

## Declaration

```
public static HexPrismFaceDir FromSide(int side)
```

## Parameters

TYPE	NAME	DESCRIPTION
Int32	side	

## Returns

TYPE	DESCRIPTION
HexPrismFaceDir	

## GetCellCenter(Vector3Int, Vector3, Vector3)

### Declaration

```
public static Vector3 GetCellCenter(Vector3Int cell, Vector3 origin, Vector3 tileSize)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	
Vector3	origin	
Vector3	tileSize	

## Returns

TYPE	DESCRIPTION
Vector3	

## Rotate(HexRotation, Vector3Int)

### Declaration

```
public static Vector3Int Rotate(HexRotation rotation, Vector3Int cell)
```

## Parameters

TYPE	NAME	DESCRIPTION
HexRotation	rotation	
Vector3Int	cell	

## Returns

TYPE	DESCRIPTION
Vector3Int	

## ToCubeCoords(Vector3Int)

## Declaration

```
public static Vector3Int ToCubeCoords(Vector3Int cell)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	

## Returns

TYPE	DESCRIPTION
Vector3Int	

# Class HexPrismCellType

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

[HexPrismCellType](#)

Implements

[ICellType](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class HexPrismCellType : ICellType
```

Properties

Instance

Declaration

```
public static HexPrismCellType Instance { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">HexPrismCellType</a>	

Methods

[FindPath\(Vector3Int, Vector3Int\)](#)

Declaration

```
public IEnumerable<CellFaceDir> FindPath(Vector3Int startOffset, Vector3Int endOffset)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">Vector3Int</a>	startOffset	
<a href="#">Vector3Int</a>	endOffset	

Returns

TYPE	DESCRIPTION
<a href="#">IEnumerable&lt;CellFaceDir&gt;</a>	

[GetCellCenter\(Vector3Int, Vector3, Vector3\)](#)

Declaration

```
public Vector3 GetCellCenter(Vector3Int offset, Vector3 center, Vector3 tileSize)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
Vector3	center	
Vector3	tileSize	

## Returns

TYPE	DESCRIPTION
Vector3	

## GetDisplayName(CellFaceDir)

### Declaration

```
public string GetDisplayName(CellFaceDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
CellFaceDir	dir	

## Returns

TYPE	DESCRIPTION
String	

## GetFaceDirPairs()

### Declaration

```
public IEnumerable<(CellFaceDir, CellFaceDir)> GetFaceDirPairs()
```

## Returns

TYPE	DESCRIPTION
IEnumerable<(T1, T2)<CellFaceDir, CellFaceDir>>	

## GetFaceDirs()

### Declaration

```
public IEnumerable<CellFaceDir> GetFaceDirs()
```

## Returns

TYPE	DESCRIPTION
IEnumerable<CellFaceDir>	

## GetIdentity()

### Declaration

```
public CellRotation GetIdentity()
```

Returns

TYPE	DESCRIPTION
CellRotation	

## GetMatrix(CellRotation)

Declaration

```
public Matrix4x4 GetMatrix(CellRotation rotation)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	rotation	

Returns

TYPE	DESCRIPTION
Matrix4x4	

## GetRotations(Boolean, Boolean, RotationGroupType)

Declaration

```
public IList<CellRotation> GetRotations(bool rotatable, bool reflectable, RotationGroupType rotationGroupType)
```

Parameters

TYPE	NAME	DESCRIPTION
Boolean	rotatable	
Boolean	reflectable	
RotationGroupType	rotationGroupType	

Returns

TYPE	DESCRIPTION
IList<CellRotation>	

## Invert(CellFaceDir)

Declaration

```
public CellFaceDir Invert(CellFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
CellFaceDir	faceDir	

Returns

TYPE	DESCRIPTION
CellFaceDir	

## Invert(CellRotation)

Declaration

```
public CellRotation Invert(CellRotation a)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	a	

Returns

TYPE	DESCRIPTION
CellRotation	

## Multiply(CellRotation, CellRotation)

Declaration

```
public CellRotation Multiply(CellRotation a, CellRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	a	
CellRotation	b	

Returns

TYPE	DESCRIPTION
CellRotation	

## Realign(ISet<Vector3Int>, CellRotation)

Declaration

```
public IDictionary<Vector3Int, Vector3Int> Realign(ISet<Vector3Int> shape, CellRotation rotation)
```

Parameters

TYPE	NAME	DESCRIPTION
ISet<Vector3Int>	shape	
CellRotation	rotation	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
<a href="#">IDictionary</a> <Vector3Int, Vector3Int>	

## Rotate(CellFaceDir, CellRotation)

### Declaration

```
public CellFaceDir Rotate(CellFaceDir faceDir, CellRotation rotation)
```

### Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
<a href="#">CellFaceDir</a>	faceDir	
<a href="#">CellRotation</a>	rotation	

### Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
<a href="#">CellFaceDir</a>	

## RotateBy(CellFaceDir, FaceDetails, CellRotation)

### Declaration

```
public (CellFaceDir, FaceDetails) RotateBy(CellFaceDir faceDir, FaceDetails faceDetails, CellRotation rotation)
```

### Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
<a href="#">CellFaceDir</a>	faceDir	
<a href="#">FaceDetails</a>	faceDetails	
<a href="#">CellRotation</a>	rotation	

### Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
(T1, T2)<CellFaceDir, FaceDetails>	

## TryMove(Vector3Int, CellFaceDir, out Vector3Int)

### Declaration

```
public bool TryMove(Vector3Int offset, CellFaceDir dir, out Vector3Int dest)
```

### Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Vector3Int	offset	

TYPE	NAME	DESCRIPTION
<a href="#">CellFaceDir</a>	dir	
<a href="#">Vector3Int</a>	dest	

Returns

TYPE	DESCRIPTION
<a href="#">Boolean</a>	

**TryMoveByOffset(Vector3Int, Vector3Int, Vector3Int, CellRotation, out Vector3Int)**

Declaration

```
public bool TryMoveByOffset(Vector3Int startCell, Vector3Int startOffset, Vector3Int destOffset, CellRotation rotation, out Vector3Int destCell)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">Vector3Int</a>	startCell	
<a href="#">Vector3Int</a>	startOffset	
<a href="#">Vector3Int</a>	destOffset	
<a href="#">CellRotation</a>	rotation	
<a href="#">Vector3Int</a>	destCell	

Returns

TYPE	DESCRIPTION
<a href="#">Boolean</a>	

Implements

[ICellType](#)

# Enum HexPrismFaceDir

Enum of the 8 faces on a hex prism.

## NOTE

This class is available only in Tessera Pro

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum HexPrismFaceDir
```

Fields

NAME	DESCRIPTION
BackLeft	
BackRight	
Down	
ForwardLeft	
ForwardRight	
Left	
Right	
Up	

# Class HexPrismFaceDirExtensions

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

[HexPrismFaceDirExtensions](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public static class HexPrismFaceDirExtensions
```

Methods

[Forward\(HexPrismFaceDir\)](#)

Declaration

```
public static Vector3 Forward(this HexPrismFaceDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">HexPrismFaceDir</a>	dir	

Returns

TYPE	DESCRIPTION
<a href="#">Vector3</a>	

[ForwardInt\(HexPrismFaceDir\)](#)

Declaration

```
public static Vector3Int ForwardInt(this HexPrismFaceDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">HexPrismFaceDir</a>	dir	

Returns

TYPE	DESCRIPTION
<a href="#">Vector3Int</a>	

[GetSide\(HexPrismFaceDir\)](#)

Declaration

```
public static int GetSide(this HexPrismFaceDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
HexPrismFaceDir	dir	

## Returns

TYPE	DESCRIPTION
Int32	

## IsUpDown(HexPrismFaceDir)

### Declaration

```
public static bool IsUpDown(this HexPrismFaceDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
HexPrismFaceDir	dir	

## Returns

TYPE	DESCRIPTION
Boolean	

## Up(HexPrismFaceDir)

### Declaration

```
public static Vector3 Up(this HexPrismFaceDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
HexPrismFaceDir	dir	

## Returns

TYPE	DESCRIPTION
Vector3	

# Struct HexRotation

Represents rotations / reflections of a hexagon

## NOTE

This class is available only in Tessera Pro

Inherited Members

[ValueType.ToString\(\)](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public struct HexRotation
```

Properties

All

Declaration

```
public static readonly HexRotation[] All { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">HexRotation[]</a>	

Identity

Declaration

```
public static readonly HexRotation Identity { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">HexRotation</a>	

IsReflection

Declaration

```
public readonly bool IsReflection { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">Boolean</a>	

ReflectForwardLeft

Declaration

```
public static readonly HexRotation ReflectForwardLeft { get; }
```

Property Value

TYPE	DESCRIPTION
HexRotation	

## ReflectForwardRight

Declaration

```
public static readonly HexRotation ReflectForwardRight { get; }
```

Property Value

TYPE	DESCRIPTION
HexRotation	

## ReflectX

Declaration

```
public static readonly HexRotation ReflectX { get; }
```

Property Value

TYPE	DESCRIPTION
HexRotation	

## ReflectZ

Declaration

```
public static readonly HexRotation ReflectZ { get; }
```

Property Value

TYPE	DESCRIPTION
HexRotation	

## RotateCCW

Declaration

```
public static readonly HexRotation RotateCCW { get; }
```

Property Value

TYPE	DESCRIPTION
HexRotation	

## Rotation

Declaration

```
public readonly int Rotation { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

## Methods

### Equals(Object)

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

Returns

TYPE	DESCRIPTION
Boolean	

Overrides

[ValueType.Equals\(Object\)](#)

### GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

[ValueType.GetHashCode\(\)](#)

### Invert()

Declaration

```
public HexRotation Invert()
```

Returns

TYPE	DESCRIPTION
HexRotation	

### Rotate60(Int32)

Declaration

```
public static HexRotation Rotate60(int i)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	i	

Returns

TYPE	DESCRIPTION
HexRotation	

Operators

### Equality(HexRotation, HexRotation)

Declaration

```
public static bool operator ==(HexRotation a, HexRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	a	
HexRotation	b	

Returns

TYPE	DESCRIPTION
Boolean	

### Implicit(CellRotation to HexRotation)

Declaration

```
public static implicit operator HexRotation(CellRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	r	

Returns

TYPE	DESCRIPTION
HexRotation	

### Implicit(HexRotation to CellRotation)

Declaration

```
public static implicit operator CellRotation(HexRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	r	

Returns

TYPE	DESCRIPTION
CellRotation	

## Inequality(HexRotation, HexRotation)

Declaration

```
public static bool operator !=(HexRotation a, HexRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	a	
HexRotation	b	

Returns

TYPE	DESCRIPTION
Boolean	

## Multiply(HexRotation, Int32)

Declaration

```
public static int operator *(HexRotation a, int side)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	a	
Int32	side	

Returns

TYPE	DESCRIPTION
Int32	

## Multiply(HexRotation, HexPrismFaceDir)

Declaration

```
public static HexPrismFaceDir operator *(HexRotation rotation, HexPrismFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	rotation	
HexPrismFaceDir	faceDir	

Returns

TYPE	DESCRIPTION
HexPrismFaceDir	

## Multiply(HexRotation, HexRotation)

Declaration

```
public static HexRotation operator *(HexRotation a, HexRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	a	
HexRotation	b	

Returns

TYPE	DESCRIPTION
HexRotation	

# Interface ICellType

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public interface ICellType
```

## Methods

**FindPath(Vector3Int, Vector3Int)**

Declaration

```
IEnumerable<CellFaceDir> FindPath(Vector3Int startOffset, Vector3Int endOffset)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	startOffset	
Vector3Int	endOffset	

Returns

TYPE	DESCRIPTION
IEnumerable<CellFaceDir>	

**GetCellCenter(Vector3Int, Vector3, Vector3)**

Declaration

```
Vector3 GetCellCenter(Vector3Int offset, Vector3 center, Vector3 tileSize)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
Vector3	center	
Vector3	tileSize	

Returns

TYPE	DESCRIPTION
Vector3	

**GetDisplayName(CellFaceDir)**

Declaration

```
string GetDisplayName(CellFaceDir dir)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
CellFaceDir	dir	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
String	

## GetFaceDirPairs()

Declaration

```
IEnumerable<(CellFaceDir, CellFaceDir)> GetFaceDirPairs()
```

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
IEnumerable<(T1, T2)<CellFaceDir, CellFaceDir>>	

## GetFaceDirs()

Declaration

```
IEnumerable<CellFaceDir> GetFaceDirs()
```

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
IEnumerable<CellFaceDir>	

## GetIdentity()

Declaration

```
CellRotation GetIdentity()
```

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
CellRotation	

## GetMatrix(CellRotation)

Declaration

```
Matrix4x4 GetMatrix(CellRotation cellRotation)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
CellRotation	cellRotation	

Returns

TYPE	DESCRIPTION
Matrix4x4	

## GetRotations(Boolean, Boolean, RotationGroupType)

Declaration

```
IList<CellRotation> GetRotations(bool rotatable = true, bool reflectable = true, RotationGroupType rotationGroupType = RotationGroupType.All)
```

Parameters

TYPE	NAME	DESCRIPTION
Boolean	rotatable	
Boolean	reflectable	
RotationGroupType	rotationGroupType	

Returns

TYPE	DESCRIPTION
IList<CellRotation>	

## Invert(CellFaceDir)

Declaration

```
CellFaceDir Invert(CellFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
CellFaceDir	faceDir	

Returns

TYPE	DESCRIPTION
CellFaceDir	

## Invert(CellRotation)

Declaration

```
CellRotation Invert(CellRotation a)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	a	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
<a href="#">CellRotation</a>	

## Multiply(CellRotation, CellRotation)

Declaration

```
CellRotation Multiply(CellRotation a, CellRotation b)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
<a href="#">CellRotation</a>	a	
<a href="#">CellRotation</a>	b	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
<a href="#">CellRotation</a>	

## Realign(ISet<Vector3Int>, CellRotation)

Given a shape, and a rotation, finds the translation that puts the rotated shape back on itself, and applies that translation to each of the offsets in the shape. Returns null if no such mapping is possible.

Declaration

```
IDictionary<Vector3Int, Vector3Int> Realign(ISet<Vector3Int> shape, CellRotation rotation)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
<a href="#">ISet&lt;Vector3Int&gt;</a>	shape	
<a href="#">CellRotation</a>	rotation	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
<a href="#">IDictionary&lt;Vector3Int, Vector3Int&gt;</a>	

## Rotate(CellFaceDir, CellRotation)

Declaration

```
CellFaceDir Rotate(CellFaceDir faceDir, CellRotation rotation)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
<a href="#">CellFaceDir</a>	faceDir	

TYPE	NAME	DESCRIPTION
CellRotation	rotation	

Returns

TYPE	DESCRIPTION
CellFaceDir	

### RotateBy(CellFaceDir, FaceDetails, CellRotation)

Declaration

```
(CellFaceDir, FaceDetails) RotateBy(CellFaceDir faceDir, FaceDetails faceDetails, CellRotation rot)
```

Parameters

TYPE	NAME	DESCRIPTION
CellFaceDir	faceDir	
FaceDetails	faceDetails	
CellRotation	rot	

Returns

TYPE	DESCRIPTION
(T1, T2)<CellFaceDir, FaceDetails>	

### TryMove(Vector3Int, CellFaceDir, out Vector3Int)

Declaration

```
bool TryMove(Vector3Int offset, CellFaceDir dir, out Vector3Int dest)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellFaceDir	dir	
Vector3Int	dest	

Returns

TYPE	DESCRIPTION
Boolean	

### TryMoveByOffset(Vector3Int, Vector3Int, Vector3Int, CellRotation, out Vector3Int)

Note startCell/destCell are actually offsets, but naming is hard...

Declaration

```
bool TryMoveByOffset(Vector3Int startCell, Vector3Int startOffset, Vector3Int destOffset, CellRotation rotation, out Vector3Int destCell)
```

#### Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	startCell	
Vector3Int	startOffset	
Vector3Int	destOffset	
CellRotation	rotation	
Vector3Int	destCell	

#### Returns

TYPE	DESCRIPTION
Boolean	

# Interface IEngineInterface

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public interface IEngineInterface
```

## Methods

Destroy(UnityEngine.Object)

Declaration

```
void Destroy(UnityEngine.Object o)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Object	o	

RegisterCompleteObjectUndo(UnityEngine.Object)

Declaration

```
void RegisterCompleteObjectUndo(UnityEngine.Object objectToUndo)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Object	objectToUndo	

RegisterCreatedObjectUndo(UnityEngine.Object)

Declaration

```
void RegisterCreatedObjectUndo(UnityEngine.Object objectToUndo)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Object	objectToUndo	

# Interface IGrid

Represents a arrangement of cells, including their adjacency and locations. Cells are uniquely identified by a Vector3Int. Tessera.IGrid is roughly equivalent to DeBroglie.Topo.ITopology.

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public interface IGrid
```

## Properties

### CellType

Describes what sort of cell can be found in this grid

Declaration

```
ICellType CellType { get; }
```

## Property Value

TYPE	DESCRIPTION
ICellType	

### IndexCount

Finds a number one larger than the maximum index for an in bounds cell.

Declaration

```
int IndexCount { get; }
```

## Property Value

TYPE	DESCRIPTION
Int32	

## Methods

### FindCell(Vector3, Matrix4x4, out Vector3Int, out CellRotation)

Returns the cell and rotation for a tile placed in the grid. NB: If you pass TesseraTileBase.center as the center, the cell returned corresponds to offset (0,0,0). The tile may not actually occupy that offset.

Declaration

```
bool FindCell(Vector3 tileCenter, Matrix4x4 tileLocalToGridMatrix, out Vector3Int cell, out CellRotation rotation)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector3	tileCenter	
Matrix4x4	tileLocalToGridMatrix	

TYPE	NAME	DESCRIPTION
Vector3Int	cell	
CellRotation	rotation	

Returns

TYPE	DESCRIPTION
Boolean	

### FindCell(Vector3, out Vector3Int)

Finds the cell containg the give position

Declaration

```
bool FindCell(Vector3 position, out Vector3Int cell)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	position	
Vector3Int	cell	

Returns

TYPE	DESCRIPTION
Boolean	

### GetBoundsSymmetries(Boolean, Boolean, RotationGroupType)

Declaration

```
IEnumerable<(GridSymmetry, Vector3Int)> GetBoundsSymmetries(bool rotatable = true, bool reflectable = true, RotationGroupType rotationGroupType = RotationGroupType.All)
```

Parameters

TYPE	NAME	DESCRIPTION
Boolean	rotatable	
Boolean	reflectable	
RotationGroupType	rotationGroupType	

Returns

TYPE	DESCRIPTION
IEnumerable<(T1, T2)<GridSymmetry, Vector3Int>>	

### GetCell(Int32)

Finds the cell associated with a given index.

Declaration

```
Vector3Int GetCell(int index)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	index	

Returns

TYPE	DESCRIPTION
Vector3Int	

**GetCellCenter(Vector3Int)**

Returns the center of the cell in local space

Declaration

```
Vector3 GetCellCenter(Vector3Int cell)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	

Returns

TYPE	DESCRIPTION
Vector3	

**GetCells()**

Gets a full list of cells in bounds.

Declaration

```
IEnumerable<Vector3Int> GetCells()
```

Returns

TYPE	DESCRIPTION
IEnumerable<Vector3Int>	

**GetCellsIntersectsApprox(Bounds, Boolean)**

Gets the set of cells that potentially overlap bounds.

Declaration

```
IEnumerable<Vector3Int> GetCellsIntersectsApprox(Bounds bounds, bool useBounds)
```

Parameters

TYPE	NAME	DESCRIPTION
Bounds	bounds	
Boolean	useBounds	

Returns

TYPE	DESCRIPTION
IEnumerable<Vector3Int>	

## GetIndex(Vector3Int)

Finds the index associated with a given cell. Cell must be in bounds.

Declaration

```
int GetIndex(Vector3Int cell)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	

Returns

TYPE	DESCRIPTION
Int32	

## GetMoveRotations()

Returns the full set of rotations that TryMove can output. Can just default to CellType.GetRotations();

Declaration

```
IEnumerable<CellRotation> GetMoveRotations()
```

Returns

TYPE	DESCRIPTION
IEnumerable<CellRotation>	

## GetTRS(Vector3Int)

Returns the appropriate transform for the cell. The translation will always be to GetCellCenter. Not inclusive of cell rotation, that should be applied first.

Declaration

```
TRS GetTRS(Vector3Int cell)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	

Returns

TYPE	DESCRIPTION
TRS	

### GetValidFaceDirs(Vector3Int)

Returns directions we might expect TryMove to work for (though it is not guaranteed). This is mostly useful for heterogenous grids where not every cell is identical.

Declaration

```
IEnumerable<CellFaceDir> GetValidFaceDirs(Vector3Int cell)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	

Returns

TYPE	DESCRIPTION
IEnumerable<CellFaceDir>	

### InBounds(Vector3Int)

Returns true if the cell is actually in the size specified. Some grids support out of bounds cells, in which case operations like TryMove may need to be filtered.

Declaration

```
bool InBounds(Vector3Int cell)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	

Returns

TYPE	DESCRIPTION
Boolean	

### TryApplySymmetry(GridSymmetry, Vector3Int, out Vector3Int, out CellRotation)

Declaration

```
bool TryApplySymmetry(GridSymmetry s, Vector3Int cell, out Vector3Int dest, out CellRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
GridSymmetry	s	
Vector3Int	cell	
Vector3Int	dest	
CellRotation	r	

Returns

TYPE	DESCRIPTION
Boolean	

### TryMove(Vector3Int, CellFaceDir, out Vector3Int, out CellFaceDir, out CellRotation)

Attempts to move from a face in a given direction, and returns information about the move if successful. Note that for some grids, this can succeed, and return a cell outside of bounds.

Declaration

```
bool TryMove(Vector3Int cell, CellFaceDir faceDir, out Vector3Int dest, out CellFaceDir inverseFaceDir, out CellRotation rotation)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	
CellFaceDir	faceDir	
Vector3Int	dest	
CellFaceDir	inverseFaceDir	
CellRotation	rotation	

Returns

TYPE	DESCRIPTION
Boolean	

### TryMoveByOffset(Vector3Int, Vector3Int, Vector3Int, CellRotation, out Vector3Int, out CellRotation)

Maps between cell offsets and cells in the grid. This could be done with [FindPath\(Vector3Int, Vector3Int\)](#), but this can be more efficient.

Declaration

```
bool TryMoveByOffset(Vector3Int startCell, Vector3Int startOffset, Vector3Int destOffset, CellRotation startRotation, out Vector3Int destCell, out CellRotation destRotation)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Vector3Int	startCell	
Vector3Int	startOffset	
Vector3Int	destOffset	
CellRotation	startRotation	
Vector3Int	destCell	
CellRotation	destRotation	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Boolean	

# Class InfiniteGenerator

Inheritance

[Object](#)

[InfiniteGenerator](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class InfiniteGenerator : MonoBehaviour
```

Fields

**chunkCleanupType**

Determines what to do with unused chunks

Declaration

```
public ChunkCleanupType chunkCleanupType
```

Field Value

TYPE	DESCRIPTION
<a href="#">ChunkCleanupType</a>	

**chunkPersistTime**

Time a chunk is kept around for even if not near a watched collider.

Declaration

```
public float chunkPersistTime
```

Field Value

TYPE	DESCRIPTION
<a href="#">Single</a>	

**cleanupInterval**

Time between cleanups that remove chunks that are no longer needed.

Declaration

```
public float cleanupInterval
```

Field Value

TYPE	DESCRIPTION
<a href="#">Single</a>	

**generator**

The generator that is used to fill the chunks. It also determines the size of each chunk.

Declaration

```
public TesseraGenerator generator
```

#### Field Value

TYPE	DESCRIPTION
TesseraGenerator	

#### infiniteX

If true, chunks repeat infinitely on this axis. If false, you can specify the [minXChunk/maxXChunk](#) to give a limit to the amount of chunks generated.

#### Declaration

```
public bool infiniteX
```

#### Field Value

TYPE	DESCRIPTION
Boolean	

#### infiniteY

If true, chunks repeat infinitely on this axis. If false, you can specify the [minYChunk/maxYChunk](#) to give a limit to the amount of chunks generated.

#### Declaration

```
public bool infiniteY
```

#### Field Value

TYPE	DESCRIPTION
Boolean	

#### infiniteZ

If true, chunks repeat infinitely on this axis. If false, you can specify the [minZChunk/maxZChunk](#) to give a limit to the amount of chunks generated.

#### Declaration

```
public bool infiniteZ
```

#### Field Value

TYPE	DESCRIPTION
Boolean	

#### maxCleanupPerUpdate

Maximum number of chunks to remove per update. Zero means unbounded.

#### Declaration

```
public int maxCleanupPerUpdate
```

Field Value

TYPE	DESCRIPTION
Int32	

maxInstantiatePerUpdate

Maximum number of tiles to create per-update. Negative means unbounded.

Declaration

```
public float maxInstantiatePerUpdate
```

Field Value

TYPE	DESCRIPTION
Single	

maxXChunk

Declaration

```
public int maxXChunk
```

Field Value

TYPE	DESCRIPTION
Int32	

maxYChunk

Declaration

```
public int maxYChunk
```

Field Value

TYPE	DESCRIPTION
Int32	

maxZChunk

Declaration

```
public int maxZChunk
```

Field Value

TYPE	DESCRIPTION
Int32	

minXChunk

Declaration

```
public int minXChunk
```

## Field Value

TYPE	DESCRIPTION
Int32	

## minYChunk

### Declaration

```
public int minYChunk
```

## Field Value

TYPE	DESCRIPTION
Int32	

## minZChunk

### Declaration

```
public int minZChunk
```

## Field Value

TYPE	DESCRIPTION
Int32	

## parallelism

The number of chunks that can be generated concurrently. Note that turning this up can cause slightly worse quality output.

### Declaration

```
public int parallelism
```

## Field Value

TYPE	DESCRIPTION
Int32	

## scanInterval

Time between scans that detect if new chunks need creating

### Declaration

```
public float scanInterval
```

## Field Value

TYPE	DESCRIPTION
Single	

## seed

Fixes the seed for random number generator. If the value is zero, the seed is taken from Unity.Random. Note that generation is

still non-deterministic even with a fixed seed.

Declaration

```
public int seed
```

Field Value

TYPE	DESCRIPTION
Int32	

### watchedColliders

Determines the volume in which chunks should be generated. You typically want to use a large trigger collider following the player or camera, to ensure that everything nearby is generated.

Declaration

```
public List<Collider> watchedColliders
```

Field Value

TYPE	DESCRIPTION
List<Collider>	

# Class InstantiateOutput

Inheritance

[Object](#)

[InstantiateOutput](#)

Implements

[ITesseraTileOutput](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class InstantiateOutput : ITesseraTileOutput
```

Constructors

[InstantiateOutput\(Transform\)](#)

Declaration

```
public InstantiateOutput(Transform transform)
```

Parameters

TYPE	NAME	DESCRIPTION
Transform	transform	

Properties

[IsEmpty](#)

Declaration

```
public bool IsEmpty { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

[SupportsIncremental](#)

Declaration

```
public bool SupportsIncremental { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Methods

[ClearTiles\(IEngineInterface\)](#)

Declaration

```
public void ClearTiles(IEngineInterface engine)
```

## Parameters

TYPE	NAME	DESCRIPTION
IEngineInterface	engine	

UpdateTiles(TesseraCompletion, IEngineInterface)

## Declaration

```
public void UpdateTiles(TesseraCompletion completion, IEngineInterface engine)
```

## Parameters

TYPE	NAME	DESCRIPTION
TesseraCompletion	completion	
IEngineInterface	engine	

## Implements

ITesseraTileOutput

# Interface ITesseraInitialConstraint

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public interface ITesseraInitialConstraint
```

Properties

Name

Declaration

```
string Name { get; }
```

Property Value

TYPE	DESCRIPTION
String	

# Interface ITesseraTileOutput

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public interface ITesseraTileOutput
```

## Properties

### IsEmpty

Is the output currently empty.

Declaration

```
bool IsEmpty { get; }
```

## Property Value

TYPE	DESCRIPTION
Boolean	

### SupportsIncremental

Is this output safe to use with AnimatedGenerator

Declaration

```
bool SupportsIncremental { get; }
```

## Property Value

TYPE	DESCRIPTION
Boolean	

## Methods

### ClearTiles(IEngineInterface)

Clear the output

Declaration

```
void ClearTiles(IEngineInterface engine)
```

## Parameters

TYPE	NAME	DESCRIPTION
IEngineInterface	engine	

### UpdateTiles(TesseraCompletion, IEngineInterface)

Update a chunk of tiles. If incremental updates are supported, then:

- Tiles can replace other tiles, as indicated by the [Cells](#) field.
- A tile of null indicates that the tile should be erased

## Declaration

```
void UpdateTiles(TesseraCompletion completion, IEngineInterface engine)
```

## Parameters

TYPE	NAME	DESCRIPTION
TesseraCompletion	completion	
IEngineInterface	engine	

# Class MeshData

A replacement for UnityEngine.Mesh that stores all the data in memory, for fast access from C#.

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

[MeshData](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class MeshData
```

Constructors

[MeshData\(\)](#)

Declaration

```
public MeshData()
```

[MeshData\(Mesh\)](#)

Declaration

```
public MeshData(Mesh mesh)
```

Parameters

TYPE	NAME	DESCRIPTION
Mesh	mesh	

Fields

[indices](#)

Declaration

```
public int[][] indices
```

Field Value

TYPE	DESCRIPTION
Int32[]	

[normals](#)

Declaration

```
public Vector3[] normals
```

Field Value

TYPE	DESCRIPTION
Vector3[]	

## subMeshCount

Declaration

```
public int subMeshCount
```

Field Value

TYPE	DESCRIPTION
Int32	

## tangents

Declaration

```
public Vector4[] tangents
```

Field Value

TYPE	DESCRIPTION
Vector4[]	

## topologies

Declaration

```
public MeshTopology[] topologies
```

Field Value

TYPE	DESCRIPTION
MeshTopology[]	

## uv

Declaration

```
public Vector2[] uv
```

Field Value

TYPE	DESCRIPTION
Vector2[]	

## vertices

Declaration

```
public Vector3[] vertices
```

Field Value

TYPE	DESCRIPTION
Vector3[]	

## Methods

### GetIndices(Int32)

Declaration

```
public int[] GetIndices(int submesh)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	submesh	

Returns

TYPE	DESCRIPTION
Int32[]	

### GetTopology(Int32)

Declaration

```
public MeshTopology GetTopology(int submesh)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	submesh	

Returns

TYPE	DESCRIPTION
MeshTopology	

# Class MeshDeformation

Encapsulates an arbitrary deformation of mesh vertices

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

[MeshDeformation](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class MeshDeformation
```

Constructors

`MeshDeformation(Func<Vector3, Vector3>, Func<Vector3, Vector3, Vector3>, Func<Vector3, Vector4, Vector4>, Boolean)`

Declaration

```
public MeshDeformation(Func<Vector3, Vector3> deformPoint, Func<Vector3, Vector3, Vector3> deformNormal,
Func<Vector3, Vector4, Vector4> deformTangent, bool invertWinding)
```

Parameters

TYPE	NAME	DESCRIPTION
<code>Func&lt;Vector3, Vector3&gt;</code>	deformPoint	
<code>Func&lt;Vector3, Vector3, Vector3&gt;</code>	deformNormal	
<code>Func&lt;Vector3, Vector4, Vector4&gt;</code>	deformTangent	
<code>Boolean</code>	invertWinding	

Fields

[PostDeform](#)

Declaration

```
public Matrix4x4 PostDeform
```

Field Value

TYPE	DESCRIPTION
<code>Matrix4x4</code>	

[PostDeformIT](#)

Declaration

```
public Matrix4x4 PostDeformIT
```

## Field Value

TYPE	DESCRIPTION
Matrix4x4	

## PreDeform

### Declaration

```
public Matrix4x4 PreDeform
```

## Field Value

TYPE	DESCRIPTION
Matrix4x4	

## PreDeformIT

### Declaration

```
public Matrix4x4 PreDeformIT
```

## Field Value

TYPE	DESCRIPTION
Matrix4x4	

## Properties

### InnerDeformNormal

#### Declaration

```
public Func<Vector3, Vector3, Vector3> InnerDeformNormal { get; set; }
```

#### Property Value

TYPE	DESCRIPTION
Func<Vector3, Vector3, Vector3>	

### InnerDeformPoint

#### Declaration

```
public Func<Vector3, Vector3> InnerDeformPoint { get; set; }
```

#### Property Value

TYPE	DESCRIPTION
Func<Vector3, Vector3>	

### InnerDeformTangent

#### Declaration

```
public Func<Vector3, Vector4, Vector4> InnerDeformTangent { get; set; }
```

## Property Value

TYPE	DESCRIPTION
Func<Vector3, Vector4, Vector4>	

## InnerInvertWinding

### Declaration

```
public bool InnerInvertWinding { get; set; }
```

## Property Value

TYPE	DESCRIPTION
Boolean	

## InvertWinding

### Declaration

```
public bool InvertWinding { get; }
```

## Property Value

TYPE	DESCRIPTION
Boolean	

## Methods

### Clone()

#### Declaration

```
public MeshDeformation Clone()
```

#### Returns

TYPE	DESCRIPTION
MeshDeformation	

### Deform(Mesh)

Deforms the vertices and normals of a mesh as specified.

#### Declaration

```
public Mesh Deform(Mesh mesh)
```

#### Parameters

TYPE	NAME	DESCRIPTION
Mesh	mesh	

#### Returns

TYPE	DESCRIPTION
Mesh	

## DeformNormal(Vector3, Vector3)

### Declaration

```
public Vector3 DeformNormal(Vector3 p, Vector3 v)
```

### Parameters

TYPE	NAME	DESCRIPTION
Vector3	p	
Vector3	v	

### Returns

TYPE	DESCRIPTION
Vector3	

## DeformPoint(Vector3)

### Declaration

```
public Vector3 DeformPoint(Vector3 p)
```

### Parameters

TYPE	NAME	DESCRIPTION
Vector3	p	

### Returns

TYPE	DESCRIPTION
Vector3	

## Transform(Mesh, Int32)

Transforms the vertices and normals of a submesh mesh as specified.

### Declaration

```
public Mesh Transform(Mesh mesh, int submesh)
```

### Parameters

TYPE	NAME	DESCRIPTION
Mesh	mesh	
Int32	submesh	

### Returns

TYPE	DESCRIPTION
Mesh	

## Operators

### Multiply(Matrix4x4, MeshDeformation)

Declaration

```
public static MeshDeformation operator *(Matrix4x4 m, MeshDeformation meshDeformation)
```

Parameters

TYPE	NAME	DESCRIPTION
Matrix4x4	m	
MeshDeformation	meshDeformation	

Returns

TYPE	DESCRIPTION
MeshDeformation	

### Multiply(MeshDeformation, Matrix4x4)

Declaration

```
public static MeshDeformation operator *(MeshDeformation meshDeformation, Matrix4x4 m)
```

Parameters

TYPE	NAME	DESCRIPTION
MeshDeformation	meshDeformation	
Matrix4x4	m	

Returns

TYPE	DESCRIPTION
MeshDeformation	

# Class MeshUtils

Utility for working with meshes.

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

[MeshUtils](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public static class MeshUtils
```

## Methods

[GetDeformation\(MeshData, Single, Single, Boolean, Int32, Int32, Int32\)](#)

Transforms from a unit cube centered on the origin to the surface of the mesh

Declaration

```
public static MeshDeformation GetDeformation(MeshData surfaceMesh, float tileHeight, float surfaceOffset, bool smoothNormals, int face, int layer, int subMesh)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">MeshData</a>	surfaceMesh	
<a href="#">Single</a>	tileHeight	
<a href="#">Single</a>	surfaceOffset	
<a href="#">Boolean</a>	smoothNormals	
<a href="#">Int32</a>	face	
<a href="#">Int32</a>	layer	
<a href="#">Int32</a>	subMesh	

Returns

TYPE	DESCRIPTION
<a href="#">MeshDeformation</a>	

[GetDeformation\(MeshData, Single, Single, Boolean, TesseraTileInstance\)](#)

Deforms from tile local space to the surface of the mesh

Declaration

```
public static MeshDeformation GetDeformation(MeshData surfaceMesh, float tileHeight, float surfaceOffset, bool smoothNormals, TesseraTileInstance i)
```

#### Parameters

TYPE	NAME	DESCRIPTION
MeshData	surfaceMesh	
Single	tileHeight	
Single	surfaceOffset	
Boolean	smoothNormals	
TesseraTileInstance	i	

#### Returns

TYPE	DESCRIPTION
MeshDeformation	

### TileToCube(TesseraTile, Vector3Int)

Matrix that transforms from tile local co-ordinates to a unit centered cube, mapping the cube at the given offset to the unit cube.

#### Declaration

```
public static Matrix4x4 TileToCube(TesseraTile tile, Vector3Int offset)
```

#### Parameters

TYPE	NAME	DESCRIPTION
TesseraTile	tile	
Vector3Int	offset	

#### Returns

TYPE	DESCRIPTION
Matrix4x4	

### TileToTri(TesseraTrianglePrismTile, Vector3Int)

#### Declaration

```
public static Matrix4x4 TileToTri(TesseraTrianglePrismTile tile, Vector3Int offset)
```

#### Parameters

TYPE	NAME	DESCRIPTION
TesseraTrianglePrismTile	tile	
Vector3Int	offset	

## Returns

Type	Description
Matrix4x4	

## TransformRecursively(GameObject, MeshDeformation)

Applies Transform gameObject and its children. Components affected:

- MeshFilter
- MeshCollider
- BoxCollider

## Declaration

```
public static void TransformRecursively(GameObject gameObject, MeshDeformation meshDeformation)
```

## Parameters

Type	Name	Description
GameObject	gameObject	
MeshDeformation	meshDeformation	

# Class MirrorConstraint

Ensures that the generation is symmetric when x-axis mirrored. If there are any tile constraints, they will not be mirrored.

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

[TesseraConstraint](#)

[MirrorConstraint](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class MirrorConstraint : TesseraConstraint
```

Fields

`axis`

Declaration

```
public MirrorConstraint.Axis axis
```

Field Value

TYPE	DESCRIPTION
<a href="#">MirrorConstraint.Axis</a>	

# Enum MirrorConstraint.Axis

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

## Syntax

```
public enum Axis
```

## Fields

NAME	DESCRIPTION
W	
X	
Y	
Z	

# Struct ModelTile

Actual tiles used internally. There's a many-to-one relationship between ModelTile and TesseraTile due to rotations and "big" tile support.

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public struct ModelTile
```

Constructors

`ModelTile(TesseraTileBase, CellRotation, Vector3Int)`

Declaration

```
public ModelTile(TesseraTileBase tile, CellRotation rotation, Vector3Int offset)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileBase	tile	
CellRotation	rotation	
Vector3Int	offset	

Properties

Offset

Declaration

```
public Vector3Int Offset { readonly get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector3Int	

Rotation

Declaration

```
public CellRotation Rotation { readonly get; set; }
```

Property Value

TYPE	DESCRIPTION
CellRotation	

Tile

Declaration

```
public TesseraTileBase Tile { readonly get; set; }
```

Property Value

Type	Description
TesseraTileBase	

## Methods

### Equals(Object)

Declaration

```
public override bool Equals(object obj)
```

Parameters

Type	Name	Description
Object	obj	

Returns

Type	Description
Boolean	

Overrides

[ValueType.Equals\(Object\)](#)

### Equals(ModelTile)

Declaration

```
public bool Equals(ModelTile other)
```

Parameters

Type	Name	Description
ModelTile	other	

Returns

Type	Description
Boolean	

### GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

Type	Description
Int32	

Overrides

[ValueType.GetHashCode\(\)](#)

## ToString()

### Declaration

```
public override string ToString()
```

### Returns

TYPE	DESCRIPTION
String	

### Overrides

[ValueType.ToString\(\)](#)

# Enum ModelType

Different models Tessera supports. The model dictates how nearby tiles relate to each other.

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum ModelType
```

Fields

NAME	DESCRIPTION
Adjacent	See <a href="#">overlapping</a> .
AdjacentPaint	The default model using the painting system to determine tile adjacencies.
Overlapping	See <a href="#">overlapping</a> .

# Struct OrientedFace

Records the painted colors and location of single face of one cube in a [TesseraTile](#)

Inherited Members

[ValueType.Equals\(Object\)](#)

[ValueType.GetHashCode\(\)](#)

[ValueType.ToString\(\)](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public struct OrientedFace
```

Constructors

[OrientedFace\(Vector3Int, CellFaceDir, FaceDetails\)](#)

Declaration

```
public OrientedFace(Vector3Int offset, CellFaceDir faceDir, FaceDetails faceDetails)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
<a href="#">CellFaceDir</a>	faceDir	
<a href="#">FaceDetails</a>	faceDetails	

Fields

[faceDetails](#)

Declaration

```
public FaceDetails faceDetails
```

Field Value

TYPE	DESCRIPTION
<a href="#">FaceDetails</a>	

[faceDir](#)

Declaration

```
public CellFaceDir faceDir
```

Field Value

TYPE	DESCRIPTION
<a href="#">CellFaceDir</a>	

## offset

Declaration

```
public Vector3Int offset
```

Field Value

TYPE	DESCRIPTION
Vector3Int	

## Methods

Deconstruct(out Vector3Int, out CellFaceDir, out FaceDetails)

Declaration

```
public void Deconstruct(out Vector3Int offset, out CellFaceDir faceDir, out FaceDetails faceDetails)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellFaceDir	faceDir	
FaceDetails	faceDetails	

# Class PaletteEntry

Inheritance

[Object](#)

[PaletteEntry](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class PaletteEntry
```

Fields

**color**

Declaration

```
public Color color
```

Field Value

TYPE	DESCRIPTION
Color	

**name**

Declaration

```
public string name
```

Field Value

TYPE	DESCRIPTION
String	

# Class PathConstraint

Forces a network of tiles to connect with each other, so there is always a complete path between them. Two tiles connect along the path if:

- Both tiles are in `pathTiles` (if `hasPathTiles` set); and
- The central color of the sides of the tiles leading to each other are in `pathColors` (if `pathColors` set)

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

[TesseraConstraint](#)

[PathConstraint](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class PathConstraint : TesseraConstraint
```

Fields

`acyclic`

If set, bans all cycles, forcing a tree or forest.

Declaration

```
public bool acyclic
```

Field Value

TYPE	DESCRIPTION
<a href="#">Boolean</a>	

`connected`

If Set, then the constraint forces that all path tiles must have a contiguous path between them.

Declaration

```
public bool connected
```

Field Value

TYPE	DESCRIPTION
<a href="#">Boolean</a>	

`hasPathColors`

If set, `pathColors` is used to determine path tiles and sides.

Declaration

```
public bool hasPathColors
```

#### Field Value

TYPE	DESCRIPTION
Boolean	

#### hasPathTiles

If set, [pathTiles](#) is used to determine path tiles.

#### Declaration

```
public bool hasPathTiles
```

#### Field Value

TYPE	DESCRIPTION
Boolean	

#### loops

If set, forces there to be at least two non-overlapping valid paths between any two connected path tiles.

#### Declaration

```
public bool loops
```

#### Field Value

TYPE	DESCRIPTION
Boolean	

#### parity

Enable this if your path tileset includes no forks or junctions, it can improve the search quality.

#### Declaration

```
public bool parity
```

#### Field Value

TYPE	DESCRIPTION
Boolean	

#### pathColors

If [hasPathColors](#), this set filters tiles that the path can connect through. Only the central square on each face is inspected.

#### Declaration

```
public List<int> pathColors
```

#### Field Value

TYPE	DESCRIPTION
List<Int32>	

## pathTiles

If [hasPathTiles](#), this set filters tiles that the path can connect through.

Declaration

```
public List<TesseraTileBase> pathTiles
```

## Field Value

TYPE	DESCRIPTION
List<TesseraTileBase>	

## prioritize

If set, the the generator will prefer generating tiles near the path.

Declaration

```
public bool prioritize
```

## Field Value

TYPE	DESCRIPTION
Boolean	

# Enum PinType

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum PinType
```

Fields

NAME	DESCRIPTION
FacesAndInterior	The faces of the pinned tile are used to constrain the cells adjacent to the location of the pinned tile and the cells covered by the pin tile are masked out so no tiles will be generated in that location.
FacesOnly	The faces of the pinned tile are used to constrain the cells adjacent to the location of the pinned tile.
Pin	Forces generation the pinned tile at the location of the pin.

# Class PrefixLookup<T>

Stores key-value pairs, with efficient searching for the longest key that is a prefix of a given string.

Inheritance

[Object](#)

[PrefixLookup<T>](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class PrefixLookup<T>
```

Type Parameters

NAME	DESCRIPTION
T	

Constructors

[PrefixLookup\(\)](#)

Declaration

```
public PrefixLookup()
```

Methods

[Add\(String, Object\)](#)

Declaration

```
public void Add(string name, object value)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">String</a>	name	
<a href="#">Object</a>	value	

[TryFindLongestPrefix\(String, out T\)](#)

Declaration

```
public bool TryFindLongestPrefix(string name, out T value)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">String</a>	name	
T	value	

Returns

TYPE	DESCRIPTION
Boolean	

# Class QuadInterpolation

Inheritance

[Object](#)

[QuadInterpolation](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public static class QuadInterpolation
```

## Methods

[Interpolate\(Vector2, Vector2, Vector2, Vector2\)](#)

Declaration

```
public static Func<Vector3, Vector2> Interpolate(Vector2 v1, Vector2 v2, Vector2 v3, Vector2 v4)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2	v1	
Vector2	v2	
Vector2	v3	
Vector2	v4	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector2>	

[Interpolate\(Vector2, Vector2, Vector2, Vector2, Vector2, Vector2, Vector2, Vector2\)](#)

Declaration

```
public static Func<Vector3, Vector2> Interpolate(Vector2 v1, Vector2 v2, Vector2 v3, Vector2 v4, Vector2 v5,
Vector2 v6, Vector2 v7, Vector2 v8)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2	v1	
Vector2	v2	
Vector2	v3	
Vector2	v4	
Vector2	v5	

TYPE	NAME	DESCRIPTION
Vector2	v6	
Vector2	v7	
Vector2	v8	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector2>	

### Interpolate(Vector3, Vector3, Vector3, Vector3)

Declaration

```
public static Func<Vector3, Vector3> Interpolate(Vector3 v1, Vector3 v2, Vector3 v3, Vector3 v4)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	v1	
Vector3	v2	
Vector3	v3	
Vector3	v4	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector3>	

### Interpolate(Vector3, Vector3, Vector3, Vector3, Vector3, Vector3, Vector3, Vector3)

Declaration

```
public static Func<Vector3, Vector3> Interpolate(Vector3 v1, Vector3 v2, Vector3 v3, Vector3 v4, Vector3 v5,
Vector3 v6, Vector3 v7, Vector3 v8)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	v1	
Vector3	v2	
Vector3	v3	
Vector3	v4	

TYPE	NAME	DESCRIPTION
Vector3	v5	
Vector3	v6	
Vector3	v7	
Vector3	v8	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector3>	

## Interpolate(Vector4, Vector4, Vector4, Vector4)

Declaration

```
public static Func<Vector3, Vector4> Interpolate(Vector4 v1, Vector4 v2, Vector4 v3, Vector4 v4)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector4	v1	
Vector4	v2	
Vector4	v3	
Vector4	v4	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector4>	

## Interpolate(Vector4, Vector4, Vector4, Vector4, Vector4, Vector4, Vector4, Vector4)

Declaration

```
public static Func<Vector3, Vector4> Interpolate(Vector4 v1, Vector4 v2, Vector4 v3, Vector4 v4, Vector4 v5,
Vector4 v6, Vector4 v7, Vector4 v8)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector4	v1	
Vector4	v2	
Vector4	v3	

TYPE	NAME	DESCRIPTION
Vector4	v4	
Vector4	v5	
Vector4	v6	
Vector4	v7	
Vector4	v8	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector4>	

## InterpolateNormal(MeshData, Int32, Int32)

Declaration

```
public static Func<Vector3, Vector3> InterpolateNormal(MeshData mesh, int submesh, int face)
```

Parameters

TYPE	NAME	DESCRIPTION
MeshData	mesh	
Int32	submesh	
Int32	face	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector3>	

## InterpolatePosition(MeshData, Int32, Int32, Single, Single)

Sets up a function that does trilinear interpolation from a unit cube centered on the origin to a cube made by extruding a given face of the mesh by meshOffset1 (for y=-0.5) and meshOffset2 (for y=0.5)

Declaration

```
public static Func<Vector3, Vector3> InterpolatePosition(MeshData mesh, int submesh, int face, float meshOffset1, float meshOffset2)
```

Parameters

TYPE	NAME	DESCRIPTION
MeshData	mesh	
Int32	submesh	

TYPE	NAME	DESCRIPTION
Int32	face	
Single	meshOffset1	
Single	meshOffset2	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector3>	

### InterpolateTangent(MeshData, Int32, Int32)

Declaration

```
public static Func<Vector3, Vector4> InterpolateTangent(MeshData mesh, int submesh, int face)
```

Parameters

TYPE	NAME	DESCRIPTION
MeshData	mesh	
Int32	submesh	
Int32	face	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector4>	

### InterpolateUv(MeshData, Int32, Int32)

Declaration

```
public static Func<Vector3, Vector2> InterpolateUv(MeshData mesh, int submesh, int face)
```

Parameters

TYPE	NAME	DESCRIPTION
MeshData	mesh	
Int32	submesh	
Int32	face	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector2>	

# Enum RotationGroupType

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

## Syntax

```
public enum RotationGroupType
```

## Fields

NAME	DESCRIPTION
All	
None	
XY	
XZ	
YZ	

# Class SeparationConstraint

Inheritance

[Object](#)

[TesseraConstraint](#)

SeparationConstraint

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class SeparationConstraint : TesseraConstraint
```

Fields

**minDistance**

The count to be compared against.

Declaration

```
public int minDistance
```

Field Value

TYPE	DESCRIPTION
<a href="#">Int32</a>	

**tiles**

The set of tiles to count

Declaration

```
public List<TesseraTileBase> tiles
```

Field Value

TYPE	DESCRIPTION
<a href="#">List&lt;TesseraTileBase&gt;</a>	

# Class SquareCellType

Inheritance

[Object](#)

SquareCellType

Implements

[ICellType](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class SquareCellType : ICellType
```

Properties

Instance

Declaration

```
public static SquareCellType Instance { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">SquareCellType</a>	

Methods

[FindPath\(Vector3Int, Vector3Int\)](#)

Declaration

```
public IEnumerable<CellFaceDir> FindPath(Vector3Int startOffset, Vector3Int endOffset)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">Vector3Int</a>	startOffset	
<a href="#">Vector3Int</a>	endOffset	

Returns

TYPE	DESCRIPTION
<a href="#">IEnumerable&lt;CellFaceDir&gt;</a>	

[GetCellCenter\(Vector3Int, Vector3, Vector3\)](#)

Declaration

```
public Vector3 GetCellCenter(Vector3Int offset, Vector3 center, Vector3 tileSize)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
Vector3	center	
Vector3	tileSize	

Returns

TYPE	DESCRIPTION
Vector3	

### GetDisplayName(CellFaceDir)

Declaration

```
public string GetDisplayName(CellFaceDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
CellFaceDir	dir	

Returns

TYPE	DESCRIPTION
String	

### GetFaceDirPairs()

Declaration

```
public IEnumerable<(CellFaceDir, CellFaceDir> GetFaceDirPairs()
```

Returns

TYPE	DESCRIPTION
IEnumerable<(T1, T2)<CellFaceDir, CellFaceDir>>	

### GetFaceDirs()

Declaration

```
public IEnumerable<CellFaceDir> GetFaceDirs()
```

Returns

TYPE	DESCRIPTION
IEnumerable<CellFaceDir>	

### GetIdentity()

Declaration

```
public CellRotation GetIdentity()
```

Returns

TYPE	DESCRIPTION
CellRotation	

## GetMatrix(CellRotation)

Declaration

```
public Matrix4x4 GetMatrix(CellRotation cellRotation)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	cellRotation	

Returns

TYPE	DESCRIPTION
Matrix4x4	

## GetRotations(Boolean, Boolean, RotationGroupType)

Declaration

```
public IList<CellRotation> GetRotations(bool rotatable, bool reflectable, RotationGroupType rotationGroupType)
```

Parameters

TYPE	NAME	DESCRIPTION
Boolean	rotatable	
Boolean	reflectable	
RotationGroupType	rotationGroupType	

Returns

TYPE	DESCRIPTION
IList<CellRotation>	

## Invert(CellFaceDir)

Declaration

```
public CellFaceDir Invert(CellFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
CellFaceDir	faceDir	

Returns

TYPE	DESCRIPTION
CellFaceDir	

## Invert(CellRotation)

Declaration

```
public CellRotation Invert(CellRotation a)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	a	

Returns

TYPE	DESCRIPTION
CellRotation	

## Multiply(CellRotation, CellRotation)

Declaration

```
public CellRotation Multiply(CellRotation a, CellRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	a	
CellRotation	b	

Returns

TYPE	DESCRIPTION
CellRotation	

## Realign(ISet<Vector3Int>, CellRotation)

Declaration

```
public IDictionary<Vector3Int, Vector3Int> Realign(ISet<Vector3Int> shape, CellRotation rotation)
```

Parameters

TYPE	NAME	DESCRIPTION
ISet<Vector3Int>	shape	
CellRotation	rotation	

Returns

TYPE	DESCRIPTION
<a href="#">IDictionary&lt;Vector3Int, Vector3Int&gt;</a>	

## Rotate(CellFaceDir, CellRotation)

### Declaration

```
public CellFaceDir Rotate(CellFaceDir faceDir, CellRotation rotation)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">CellFaceDir</a>	faceDir	
<a href="#">CellRotation</a>	rotation	

### Returns

TYPE	DESCRIPTION
<a href="#">CellFaceDir</a>	

## RotateBy(CellFaceDir, FaceDetails, CellRotation)

### Declaration

```
public (CellFaceDir, FaceDetails) RotateBy(CellFaceDir faceDir, FaceDetails faceDetails, CellRotation rot)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">CellFaceDir</a>	faceDir	
<a href="#">FaceDetails</a>	faceDetails	
<a href="#">CellRotation</a>	rot	

### Returns

TYPE	DESCRIPTION
<a href="#">(T1, T2)&lt;CellFaceDir, FaceDetails&gt;</a>	

## TryMove(Vector3Int, CellFaceDir, out Vector3Int)

### Declaration

```
public bool TryMove(Vector3Int offset, CellFaceDir dir, out Vector3Int dest)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">Vector3Int</a>	offset	

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
<a href="#">CellFaceDir</a>	dir	
<a href="#">Vector3Int</a>	dest	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
<a href="#">Boolean</a>	

**TryMoveByOffset(Vector3Int, Vector3Int, Vector3Int, CellRotation, out Vector3Int)**

Declaration

```
public bool TryMoveByOffset(Vector3Int startCell, Vector3Int startOffset, Vector3Int destOffset, CellRotation rotation, out Vector3Int destCell)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
<a href="#">Vector3Int</a>	startCell	
<a href="#">Vector3Int</a>	startOffset	
<a href="#">Vector3Int</a>	destOffset	
<a href="#">CellRotation</a>	rotation	
<a href="#">Vector3Int</a>	destCell	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
<a href="#">Boolean</a>	

Implements

[ICellType](#)

# Enum SquareFaceDir

Enum of the 4 sides of a square.

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum SquareFaceDir
```

Fields

NAME	DESCRIPTION
Down	
Left	
Right	
Up	

# Class SquareFaceDirExtensions

Inheritance

[Object](#)

SquareFaceDirExtensions

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public static class SquareFaceDirExtensions
```

## Methods

[Forward\(SquareFaceDir\)](#)

Declaration

```
public static Vector3Int Forward(this SquareFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">SquareFaceDir</a>	faceDir	

Returns

TYPE	DESCRIPTION
<a href="#">Vector3Int</a>	The normal vector for a given face.

[GetSide\(SquareFaceDir\)](#)

Declaration

```
public static int GetSide(this SquareFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">SquareFaceDir</a>	faceDir	

Returns

TYPE	DESCRIPTION
<a href="#">Int32</a>	

[Inverted\(SquareFaceDir\)](#)

Declaration

```
public static SquareFaceDir Inverted(this SquareFaceDir faceDir)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
SquareFaceDir	faceDir	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
SquareFaceDir	Returns the face dir with the opposite normal vector.

# Struct SquareRotation

Represents rotations / reflections of a square

Inherited Members

[ValueType.ToString\(\)](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public struct SquareRotation
```

## Properties

All

Declaration

```
public static readonly SquareRotation[] All { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">SquareRotation[]</a>	

## Identity

Declaration

```
public static readonly SquareRotation Identity { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">SquareRotation</a>	

## IsReflection

Declaration

```
public readonly bool IsReflection { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">Boolean</a>	

## ReflectX

Declaration

```
public static readonly SquareRotation ReflectX { get; }
```

Property Value

TYPE	DESCRIPTION
SquareRotation	

## ReflectY

### Declaration

```
public static readonly SquareRotation ReflectY { get; }
```

### Property Value

TYPE	DESCRIPTION
SquareRotation	

## RotateCCW

### Declaration

```
public static readonly SquareRotation RotateCCW { get; }
```

### Property Value

TYPE	DESCRIPTION
SquareRotation	

## Rotation

### Declaration

```
public readonly int Rotation { get; }
```

### Property Value

TYPE	DESCRIPTION
Int32	

## Methods

### Equals(Object)

#### Declaration

```
public override bool Equals(object obj)
```

#### Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

#### Returns

TYPE	DESCRIPTION
Boolean	

#### Overrides

[ValueType.Equals\(Object\)](#)

[GetHashCode\(\)](#)

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
Int32	

Overrides

[ValueType.GetHashCode\(\)](#)

[Invert\(\)](#)

Declaration

```
public SquareRotation Invert()
```

Returns

TYPE	DESCRIPTION
SquareRotation	

[Rotate90\(Int32\)](#)

Declaration

```
public static SquareRotation Rotate90(int i)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	i	

Returns

TYPE	DESCRIPTION
SquareRotation	

Operators

[Equality\(SquareRotation, SquareRotation\)](#)

Declaration

```
public static bool operator ==(SquareRotation a, SquareRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	a	

TYPE	NAME	DESCRIPTION
SquareRotation	b	

Returns

TYPE	DESCRIPTION
Boolean	

## Implicit(CellRotation to SquareRotation)

Declaration

```
public static implicit operator SquareRotation(CellRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	r	

Returns

TYPE	DESCRIPTION
SquareRotation	

## Implicit(SquareRotation to CellRotation)

Declaration

```
public static implicit operator CellRotation(SquareRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	r	

Returns

TYPE	DESCRIPTION
CellRotation	

## Inequality(SquareRotation, SquareRotation)

Declaration

```
public static bool operator !=(SquareRotation a, SquareRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	a	
SquareRotation	b	

Returns

TYPE	DESCRIPTION
Boolean	

## Multiply(SquareRotation, BoundsInt)

Declaration

```
public static BoundsInt operator *(SquareRotation r, BoundsInt bounds)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	r	
BoundsInt	bounds	

Returns

TYPE	DESCRIPTION
BoundsInt	

## Multiply(SquareRotation, Int32)

Declaration

```
public static int operator *(SquareRotation a, int side)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	a	
Int32	side	

Returns

TYPE	DESCRIPTION
Int32	

## Multiply(SquareRotation, SquareFaceDir)

Declaration

```
public static SquareFaceDir operator *(SquareRotation rotation, SquareFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	rotation	
SquareFaceDir	faceDir	

Returns

TYPE	DESCRIPTION
SquareFaceDir	

## Multiply(SquareRotation, SquareRotation)

Declaration

```
public static SquareRotation operator *(SquareRotation a, SquareRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	a	
SquareRotation	b	

Returns

TYPE	DESCRIPTION
SquareRotation	

## Multiply(SquareRotation, Vector3Int)

Declaration

```
public static Vector3Int operator *(SquareRotation r, Vector3Int v)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	r	
Vector3Int	v	

Returns

TYPE	DESCRIPTION
Vector3Int	

# Class TesseraCompletion

Returned by TesseraGenerator after generation finishes

Inheritance

## Object

### TesseraCompletion

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraCompletion
```

## Properties

### backtrackCount

The number of times the generation process backtracked.

Declaration

```
public int backtrackCount { get; set; }
```

## Property Value

TYPE	DESCRIPTION
Int32	

### contradictionLocation

If success is false, indicates where the generation failed.

Declaration

```
public Vector3Int? contradictionLocation { get; set; }
```

## Property Value

TYPE	DESCRIPTION
Nullable<Vector3Int>	

### grid

Gives details about the cells.

Declaration

```
public IGrid grid { get; set; }
```

## Property Value

TYPE	DESCRIPTION
IGrid	

### gridTransform

The relationship of the grid

Declaration

```
public TRS gridTransform { get; set; }
```

Property Value

TYPE	DESCRIPTION
TRS	

isIncremental

Indicates these instances should be added to the previous set of instances.

Declaration

```
public bool isIncremental { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

retries

The number of times the generation process was restarted.

Declaration

```
public int retries { get; set; }
```

Property Value

TYPE	DESCRIPTION
Int32	

success

True if all tiles were successfully found.

Declaration

```
public bool success { get; set; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

tileData

The raw tile data

Declaration

```
public IDictionary<Vector3Int, ModelTile> tileData { get; set; }
```

Property Value

TYPE	DESCRIPTION
<code>IDictionary&lt;Vector3Int, ModelTile&gt;</code>	

## tileInstances

The list of tiles to create.

Declaration

```
public IList<TesseraTileInstance> tileInstances { get; }
```

## Property Value

TYPE	DESCRIPTION
<code>IList&lt;TesseraTileInstance&gt;</code>	

## Methods

### LogError()

Writes error information to Unity's log.

Declaration

```
public void LogError()
```

# Class TesseraConstraint

Abstract class for all generator constraint components.

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

[TesseraConstraint](#)

[BorderConstraint](#)

[CountConstraint](#)

[MirrorConstraint](#)

[PathConstraint](#)

[SeparationConstraint](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class TesseraConstraint : MonoBehaviour
```

# Class TesseraGenerateOptions

Additional settings to customize the generation at runtime.

Inheritance

## Object

### TesseraGenerateOptions

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraGenerateOptions
```

Fields

#### cancellationToken

Allows interuption of the calculations

Declaration

```
public CancellationToken cancellationToken
```

Field Value

TYPE	DESCRIPTION
<a href="#">CancellationToken</a>	

#### initialConstraints

If sets, overrides TesseraGenerator.initialConstraints and TesseraGenerator.searchInitialConstraints.

Declaration

```
public List<ITesseraInitialConstraint> initialConstraints
```

Field Value

TYPE	DESCRIPTION
<a href="#">List&lt;ITesseraInitialConstraint&gt;</a>	

#### multithreaded

If set, then generation is offloaded to another thread stopping Unity from freezing. Requires you to use StartGenerate in a coroutine. Multithreaded is ignored in the WebGL player, as it doesn't support threads.

Declaration

```
public bool multithreaded
```

Field Value

TYPE	DESCRIPTION
<a href="#">Boolean</a>	

#### onComplete

Called when the generation is complete. By default, checks for success then invokes [onCreate](#) on each instance.

Declaration

```
public Action<TesseraCompletion> onComplete
```

Field Value

TYPE	DESCRIPTION
Action<TesseraCompletion>	

**onCreate**

Called for each newly generated tile. By default, [Instantiate\(TesseraTileInstance, Transform\)](#) is used.

Declaration

```
public Action<TesseraTileInstance> onCreate
```

Field Value

TYPE	DESCRIPTION
Action<TesseraTileInstance>	

**progress**

Called with a string describing the current phase of the calculations, and the progress from 0 to 1. Progress can move backwards for retries or backtracing. Note progress can be called from threads other than the main thread.

Declaration

```
public Action<string, float> progress
```

Field Value

TYPE	DESCRIPTION
Action<String, Single>	

**seed**

Fixes the seed for random number generator. By defult, random numbers from from Unity.Random.

Declaration

```
public int? seed
```

Field Value

TYPE	DESCRIPTION
Nullable<Int32>	

# Class TesseraGenerator

GameObjects with this behaviour contain utilities to generate tile based levels using Wave Function Collapse (WFC). Call [Generate\(TesseraGenerateOptions\)](#) or [StartGenerate\(TesseraGenerateOptions\)](#) to run. The generation takes the following steps:

- Inspect the tiles in [tiles](#) and work out how they rotate and connect to each other.
- Setup any initial constraints that fix parts of the generation ([initialConstraints](#)).
- Fix the boundary of the generation if [skyBox](#) is set.
- Generate a set of tile instances that fits the above tiles and constraints.
- Optionally [retries](#) or [backtrack](#).
- Instantiates the tile instances.

Inheritance

[Object](#)

TesseraGenerator

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraGenerator : MonoBehaviour
```

Fields

[algorithm](#)

Controls the algorithm used internally for Wave Function Collapse.

Declaration

```
public TesseraWfcAlgorithm algorithm
```

Field Value

TYPE	DESCRIPTION
<a href="#">TesseraWfcAlgorithm</a>	

[backtrack](#)

If set, backtracking will be used during generation. Backtracking can find solutions that would otherwise be failures, but can take a long time.

Declaration

```
public bool backtrack
```

Field Value

TYPE	DESCRIPTION
<a href="#">Boolean</a>	

[contradictionTile](#)

Game object to show in cells that cannot be solved.

Declaration

```
public GameObject contradictionTile
```

## Field Value

TYPE	DESCRIPTION
GameObject	

## failureMode

Controls what is output when the generation fails.

### Declaration

```
public FailureMode failureMode
```

## Field Value

TYPE	DESCRIPTION
FailureMode	

## filterSurfaceSubmeshTiles

If true, and a [surfaceMesh](#) is set with multiple submeshes (materials), then use [surfaceSubmeshTiles](#).

### Declaration

```
public bool filterSurfaceSubmeshTiles
```

## Field Value

TYPE	DESCRIPTION
Boolean	

## modelType

Sets which sort of model the generator uses. The model dictates how nearby tiles relate to each other.

### Declaration

```
public ModelType modelType
```

## Field Value

TYPE	DESCRIPTION
ModelType	

## overlapSize

The size of the overlap parameter for the overlapping model. [Overlapping](#)

### Declaration

```
public Vector3Int overlapSize
```

## Field Value

TYPE	DESCRIPTION
Vector3Int	

## recordUndo

Records undo/redo when run by pressing the Generate button in the Inspector.

Declaration

```
public bool recordUndo
```

Field Value

TYPE	DESCRIPTION
Boolean	

## retries

If backtracking is off, how many times to retry generation if a solution cannot be found.

Declaration

```
public int retries
```

Field Value

TYPE	DESCRIPTION
Int32	

## samples

For overlapping models, a list of objects to use as input samples. Each one will have its children inspected and read out.

### Overlapping

Declaration

```
public List<GameObject> samples
```

Field Value

TYPE	DESCRIPTION
List<GameObject>	

## scaleUncertaintyTile

If true, the uncertainty tiles shrink as the solver gets more certain.

Declaration

```
public bool scaleUncertaintyTile
```

Field Value

TYPE	DESCRIPTION
Boolean	

## searchInitialConstraints

If true, then active tiles in the scene will be taken as initial constraints. If false, then no initial constraints are used. Using [initialConstraints](#) overrides either outcome.

Declaration

```
public bool searchInitialConstraints
```

Field Value

TYPE	DESCRIPTION
Boolean	

### seed

Fixes the seed for random number generator. If the value is zero, the seed is taken from Unity.Random

Declaration

```
public int seed
```

Field Value

TYPE	DESCRIPTION
Int32	

### skyBox

If set, this tile is used to define extra initial constraints for the boundary.

Declaration

```
public TesseraTileBase skyBox
```

Field Value

TYPE	DESCRIPTION
TesseraTileBase	

### stepLimit

How many steps to take before retrying from the start.

Declaration

```
public int stepLimit
```

Field Value

TYPE	DESCRIPTION
Int32	

### surfaceMesh

If set, then tiles are generated on the surface of this mesh instead of a regular grid.

Declaration

```
public Mesh surfaceMesh
```

Field Value

TYPE	DESCRIPTION
Mesh	

### surfaceOffset

Height above the surface mesh that the bottom layer of tiles is generated at.

Declaration

```
public float surfaceOffset
```

Field Value

TYPE	DESCRIPTION
Single	

### surfaceSmoothNormals

Controls how normals are treated for meshes deformed to fit the surfaceMesh.

Declaration

```
public bool surfaceSmoothNormals
```

Field Value

TYPE	DESCRIPTION
Boolean	

### surfaceSubmeshTiles

A list of tiles to filter each submesh of [surfaceMesh](#) to. Ignored unless [filterSurfaceSubmeshTiles](#) is true.

Declaration

```
public List<TileList> surfaceSubmeshTiles
```

Field Value

TYPE	DESCRIPTION
List<TileList>	

### tiles

The list of tiles eligible for generation.

Declaration

```
public List<TileEntry> tiles
```

Field Value

TYPE	DESCRIPTION
List<TileEntry>	

## tileSize

The stride between each cell in the generation. "big" tiles may occupy a multiple of this tile size.

Declaration

```
public Vector3 tileSize
```

Field Value

TYPE	DESCRIPTION
Vector3	

## uncertaintyTile

Game object to show in cells that have yet to be fully solved.

Declaration

```
public GameObject uncertaintyTile
```

Field Value

TYPE	DESCRIPTION
GameObject	

## Properties

### bounds

The area of generation. Setting this will cause the size to be rounded to a multiple of `tileSize`

Declaration

```
public Bounds bounds { get; set; }
```

Property Value

TYPE	DESCRIPTION
Bounds	

### CellType

Indicates the cell type of the tiles set up.

Declaration

```
public ICellType CellType { get; }
```

Property Value

TYPE	DESCRIPTION
ICellType	

### center

The local position of the center of the area to generate.

Declaration

```
public Vector3 center { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector3	

## origin

Declaration

```
public Vector3 origin { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector3	

## palette

Inherited from the first tile in [tiles](#).

Declaration

```
public TesseraPalette palette { get; }
```

Property Value

TYPE	DESCRIPTION
TesseraPalette	

## size

The size of the generator area, counting in cells each of size [tileSize](#).

Declaration

```
public Vector3Int size { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector3Int	

## Methods

### Clear()

Clears previously generated content.

Declaration

```
public void Clear()
```

### Generate(TesseraGenerateOptions)

Synchronously runs the generation process described in the class docs.

Declaration

```
public TesseraCompletion Generate(TesseraGenerateOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraGenerateOptions	options	

Returns

TYPE	DESCRIPTION
TesseraCompletion	

## GetCellTypes()

For validation purposes

Declaration

```
public IList<ICellType> GetCellTypes()
```

Returns

TYPE	DESCRIPTION
IList<ICellType>	

## GetInitialConstraintBuilder()

Declaration

```
public TesseraInitialConstraintBuilder GetInitialConstraintBuilder()
```

Returns

TYPE	DESCRIPTION
TesseraInitialConstraintBuilder	

## GetMissizedTiles()

For validation purposes

Declaration

```
public IEnumerable<TesseraTileBase> GetMissizedTiles()
```

Returns

TYPE	DESCRIPTION
IEnumerable<TesseraTileBase>	

## GetTileOutput(Boolean)

Declaration

```
public ITesseraTileOutput GetTileOutput(bool forceIncremental = false)
```

#### Parameters

TYPE	NAME	DESCRIPTION
Boolean	forceIncremental	

#### Returns

TYPE	DESCRIPTION
ITesseraTileOutput	

### Instantiate(TesseraTileInstance, Transform)

Utility function that instantiates a tile instance in the scene. This is the default function used when you do not pass `onCreate` to the Generate method. It is essentially the same as Unity's normal Instantiate method with extra features:

- respects `instantiateChildrenOnly`
- applies mesh transformations (Pro only)

#### Declaration

```
public static GameObject[] Instantiate(TesseraTileInstance instance, Transform parent)
```

#### Parameters

TYPE	NAME	DESCRIPTION
TesseraTileInstance	instance	The instance being created.
Transform	parent	The game object to parent the new game object to. This does not affect the world position of the instance

#### Returns

TYPE	DESCRIPTION
GameObject[]	The game objects created.

### Instantiate(TesseraTileInstance, Transform, GameObject, Boolean)

Utility function that instantiates a tile instance in the scene. This is the default function used when you do not pass `onCreate` to the Generate method. It is essentially the same as Unity's normal Instantiate method with extra features:

- respects `instantiateChildrenOnly`
- applies mesh transformations (Pro only)

#### Declaration

```
public static GameObject[] Instantiate(TesseraTileInstance instance, Transform parent, GameObject gameObject,  
bool instantiateChildrenOnly)
```

#### Parameters

TYPE	NAME	DESCRIPTION
TesseraTileInstance	instance	The instance being created.
Transform	parent	The game object to parent the new game object to. This does not affect the world position of the instance
GameObject	gameObject	The game object to actually instantiate
Boolean	instantiateChildrenOnly	Should gameObject be created, or just its children.

Returns

TYPE	DESCRIPTION
GameObject[]	The game objects created.

## Regenerate(TesseraGenerateOptions)

Runs Clear, then Generate

Declaration

```
public TesseraCompletion Regenerate(TesseraGenerateOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraGenerateOptions	options	

Returns

TYPE	DESCRIPTION
TesseraCompletion	

## StartGenerate(TesseraGenerateOptions)

Asynchronously runs the generation process described in the class docs, for use with StartCoroutine.

Declaration

```
public EnumeratorWithResult<TesseraCompletion> StartGenerate(TesseraGenerateOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraGenerateOptions	options	

Returns

TYPE	DESCRIPTION
EnumeratorWithResult<TesseraCompletion>	

#### Remarks

The default instantiation is still synchronous, so this can still cause frame glitches unless you override onCreate.

# Class TesseraHexTile

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

Inheritance

[Object](#)

[TesseraTileBase](#)

TesseraHexTile

Inherited Members

[TesseraTileBase.palette](#)

[TesseraTileBase.faceDetails](#)

[TesseraTileBase.offsets](#)

[TesseraTileBase.center](#)

[TesseraTileBase.tileSize](#)

[TesseraTileBase.rotatable](#)

[TesseraTileBase.reflectable](#)

[TesseraTileBase.rotationGroupType](#)

[TesseraTileBase.symmetric](#)

[TesseraTileBase.instantiateChildrenOnly](#)

[TesseraTileBase.Get\(Vector3Int, CellFaceDir\)](#)

[TesseraTileBase.TryGet\(Vector3Int, CellFaceDir, FaceDetails\)](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraHexTile : TesseraTileBase
```

Constructors

[TesseraHexTile\(\)](#)

Declaration

```
public TesseraHexTile()
```

Properties

[CellType](#)

Declaration

```
public override ICellType CellType { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ICellType</a>	

Overrides

[TesseraTileBase.CellType](#)

Methods

[AddOffset\(Vector3Int\)](#)

Configures the tile as a "big" tile that occupies several cells. Keeps [offsets](#) and [faceDetails](#) in sync.

Declaration

```
public override void AddOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Overrides

[TesseraTileBase.AddOffset\(Vector3Int\)](#)

**GetBounds()**

Declaration

```
public BoundsInt GetBounds()
```

Returns

TYPE	DESCRIPTION
BoundsInt	

**RemoveOffset(Vector3Int)**

Configures the tile as a "big" tile that occupies several cells. Keeps [offsets](#) and [faceDetails](#) in sync.

Declaration

```
public override void RemoveOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Overrides

[TesseraTileBase.RemoveOffset\(Vector3Int\)](#)

# Class TesseraInitialConstraint

Initial constraint objects fix parts of the generation process in places. Use the utility methods on [TesseraGenerator](#) to create these objects.

Inheritance

[Object](#)

TesseraInitialConstraint

Implements

[ITesseraInitialConstraint](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class TesseraInitialConstraint : ITesseraInitialConstraint
```

## Properties

Name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">String</a>	

Implements

[ITesseraInitialConstraint](#)

# Class TesseraInitialConstraintBuilder

Inheritance

[Object](#)

TesseraInitialConstraintBuilder

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraInitialConstraintBuilder
```

Properties

Grid

Declaration

```
public IGrid Grid { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">IGrid</a>	

Methods

[GetInitialConstraint\(GameObject\)](#)

Gets the initial constraint for a given game object. It checks for a TesseraPinned, TesseraTile or TesseraVolume component.

Declaration

```
public ITesseraInitialConstraint GetInitialConstraint(GameObject gameObject)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">GameObject</a>	gameObject	

Returns

TYPE	DESCRIPTION
<a href="#">ITesseraInitialConstraint</a>	

[GetInitialConstraint\(TesseraPinned\)](#)

Gets the initial constraint from a given pin at a given position. It should be aligned with the grid defined by this generator.

Declaration

```
public ITesseraInitialConstraint GetInitialConstraint(TesseraPinned pin)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
TesseraPinned	pin	The pin to inspect

Returns

TYPE	DESCRIPTION
ITesseraInitialConstraint	

### GetInitialConstraint(TesseraPinned, Matrix4x4)

Gets the initial constraint from a given pin at a given position. It should be aligned with the grid defined by this generator.

Declaration

```
public ITesseraInitialConstraint GetInitialConstraint(TesseraPinned pin, Matrix4x4 localToWorldMatrix)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraPinned	pin	The pin to inspect
Matrix4x4	localToWorldMatrix	The matrix indicating the position and rotation of the tile

Returns

TYPE	DESCRIPTION
ITesseraInitialConstraint	

### GetInitialConstraint(TesseraTileBase)

Gets the initial constraint from a given tile. The tile should be aligned with the grid defined by this generator.

Declaration

```
public TesseraInitialConstraint GetInitialConstraint(TesseraTileBase tile)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileBase	tile	The tile to inspect

Returns

TYPE	DESCRIPTION
TesseraInitialConstraint	

### GetInitialConstraint(TesseraTileBase, Matrix4x4)

Gets the initial constraint from a given tile at a given position. The tile should be aligned with the grid defined by this generator.

Declaration

```
public TesseraInitialConstraint GetInitialConstraint(TesseraTileBase tile, Matrix4x4 localToWorldMatrix)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileBase	tile	The tile to inspect
Matrix4x4	localToWorldMatrix	The matrix indicating the position and rotation of the tile

Returns

TYPE	DESCRIPTION
TesseraInitialConstraint	

### GetInitialConstraint(TesseraTileInstance, PinType)

Converts a TesseraTileInstance to a ITesseraInitialConstraint. This allows you to easily use the output of one generation for later generations

Declaration

```
public ITesseraInitialConstraint GetInitialConstraint(TesseraTileInstance tileInstance, PinType pinType = PinType.Pin)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileInstance	tileInstance	
PinType	pinType	

Returns

TYPE	DESCRIPTION
ITesseraInitialConstraint	

### GetInitialConstraint(TesseraVolume)

Gets the initial constraint from a given tile. The tile should be aligned with the grid defined by this generator.

Declaration

```
public TesseraVolumeFilter GetInitialConstraint(TesseraVolume volume)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
TesseraVolume	volume	

Returns

TYPE	DESCRIPTION
TesseraVolumeFilter	

### SearchInitialConstraints()

Searches the scene for all applicable game objects and converts them to ITesseralInitialConstraint

Declaration

```
public List<ITesseraInitialConstraint> SearchInitialConstraints()
```

Returns

TYPE	DESCRIPTION
List<ITesseralInitialConstraint>	

# Class TesseraInstantiateOutput

Attach this to a TesseraGenerator to control how tiles are instantiated.

## **NOTE**

This class is available only in Tessera Pro

Inheritance

[Object](#)

[TesseraInstantiateOutput](#)

Implements

[ITesseraTileOutput](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraInstantiateOutput : MonoBehaviour, ITesseraTileOutput
```

Constructors

[TesseraInstantiateOutput\(\)](#)

Declaration

```
public TesseraInstantiateOutput()
```

Fields

**parent**

Declaration

```
public Transform parent
```

Field Value

TYPE	DESCRIPTION
Transform	

**tileMappings**

Declaration

```
public TileMapping[] tileMappings
```

Field Value

TYPE	DESCRIPTION
TileMapping[]	

Properties

**IsEmpty**

Declaration

```
public bool IsEmpty { get; }
```

#### Property Value

TYPE	DESCRIPTION
Boolean	

### SupportsIncremental

#### Declaration

```
public bool SupportsIncremental { get; }
```

#### Property Value

TYPE	DESCRIPTION
Boolean	

### Methods

#### ClearTiles(IEngineInterface)

#### Declaration

```
public void ClearTiles(IEngineInterface engine)
```

#### Parameters

TYPE	NAME	DESCRIPTION
IEngineInterface	engine	

#### UpdateTiles(TesseraCompletion, IEngineInterface)

#### Declaration

```
public void UpdateTiles(TesseraCompletion completion, IEngineInterface engine)
```

#### Parameters

TYPE	NAME	DESCRIPTION
TesseraCompletion	completion	
IEngineInterface	engine	

### Implements

#### ITesseraTileOutput

# Class TesseraMeshOutput

Attach this to a TesseraGenerator to output the tiles to a single mesh instead of instantiating them.

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

TesseraMeshOutput

Implements

[ITesseraTileOutput](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraMeshOutput : MonoBehaviour, ITesseraTileOutput
```

Fields

**targetMeshFilter**

Declaration

```
public MeshFilter targetMeshFilter
```

Field Value

TYPE	DESCRIPTION
MeshFilter	

Properties

**IsEmpty**

Declaration

```
public bool IsEmpty { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

**SupportsIncremental**

Declaration

```
public bool SupportsIncremental { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

## Methods

### ClearTiles(IEngineInterface)

Declaration

```
public void ClearTiles(IEngineInterface engine)
```

Parameters

TYPE	NAME	DESCRIPTION
IEngineInterface	engine	

### UpdateTiles(TesseraCompletion, IEngineInterface)

Declaration

```
public void UpdateTiles(TesseraCompletion completion, IEngineInterface engine)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraCompletion	completion	
IEngineInterface	engine	

Implements

[ITesseraTileOutput](#)

# Class TesseraMultipassGenerator

Inheritance

[Object](#)

TesseraMultipassGenerator

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraMultipassGenerator : MonoBehaviour
```

Fields

**passes**

Declaration

```
public TesseraMultipassPass[] passes
```

Field Value

TYPE	DESCRIPTION
<a href="#">TesseraMultipassPass[]</a>	

Methods

**Clear()**

Declaration

```
public void Clear()
```

**Generate()**

Declaration

```
public void Generate()
```

# Class TesseraMultipassPass

Inheritance

[Object](#)

TesseraMultipassPass

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class TesseraMultipassPass
```

Fields

**clearEvent**

Declaration

```
public UnityEvent clearEvent
```

Field Value

TYPE	DESCRIPTION
UnityEvent	

**generateEvent**

Declaration

```
public UnityEvent generateEvent
```

Field Value

TYPE	DESCRIPTION
UnityEvent	

**generator**

Declaration

```
public TesseraGenerator generator
```

Field Value

TYPE	DESCRIPTION
TesseraGenerator	

**passType**

Declaration

```
public TesseraMultipassPassType passType
```

Field Value

TYPE	DESCRIPTION
TesseraMultipassPassType	

# Enum TesseraMultipassPassType

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum TesseraMultipassPassType
```

Fields

NAME	DESCRIPTION
Event	
Generator	

# Class TesseraPalette

Inheritance

[Object](#)

TesseraPalette

Implements

[ISerializationCallbackReceiver](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraPalette : ScriptableObject
```

Constructors

[TesseraPalette\(\)](#)

Declaration

```
public TesseraPalette()
```

Fields

[entries](#)

Declaration

```
public List<PaletteEntry> entries
```

Field Value

TYPE	DESCRIPTION
List<PaletteEntry>	

[matchOverrides](#)

Declaration

```
public Dictionary<(int, int), bool> matchOverrides
```

Field Value

TYPE	DESCRIPTION
Dictionary<(T1, T2)<Int32, Int32>, Boolean>	

Properties

[defaultPalette](#)

Declaration

```
public static TesseraPalette defaultPalette { get; }
```

Property Value

TYPE	DESCRIPTION
TesseraPalette	

## entryCount

### Declaration

```
public int entryCount { get; }
```

### Property Value

TYPE	DESCRIPTION
Int32	

### Methods

#### GetColor(Int32)

##### Declaration

```
public Color GetColor(int i)
```

##### Parameters

TYPE	NAME	DESCRIPTION
Int32	i	

##### Returns

TYPE	DESCRIPTION
Color	

#### GetEntry(Int32)

##### Declaration

```
public PaletteEntry GetEntry(int i)
```

##### Parameters

TYPE	NAME	DESCRIPTION
Int32	i	

##### Returns

TYPE	DESCRIPTION
PaletteEntry	

#### Match(Int32, Int32)

##### Declaration

```
public bool Match(int a, int b)
```

##### Parameters

TYPE	NAME	DESCRIPTION
Int32	a	

TYPE	NAME	DESCRIPTION
Int32	b	

Returns

TYPE	DESCRIPTION
Boolean	

## Match(FaceDetails, FaceDetails)

Declaration

```
public bool Match(FaceDetails a, FaceDetails b)
```

Parameters

TYPE	NAME	DESCRIPTION
FaceDetails	a	
FaceDetails	b	

Returns

TYPE	DESCRIPTION
Boolean	

## OnAfterDeserialize()

Declaration

```
public void OnAfterDeserialize()
```

## OnBeforeSerialize()

Declaration

```
public void OnBeforeSerialize()
```

## Implements

ISerializationCallbackReceiver

# Class TesseraPinConstraint

Inheritance

[Object](#)

TesseraPinConstraint

Implements

[ITesseraiInitialConstraint](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraPinConstraint : ITesseraiInitialConstraint
```

Properties

Name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">String</a>	

Implements

[ITesseraiInitialConstraint](#)

# Class TesseraPinned

Inheritance

[Object](#)

TesseraPinned

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraPinned : MonoBehaviour
```

Fields

**pinType**

Sets the type of pin to apply.

Declaration

```
public PinType pinType
```

Field Value

TYPE	DESCRIPTION
<a href="#">PinType</a>	

**tile**

The tile to pin. Defaults to a tile component found on the same GameObject

Declaration

```
public TesseraTile tile
```

Field Value

TYPE	DESCRIPTION
<a href="#">TesseraTile</a>	

# Class TesseraSquareTile

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

Inheritance

[Object](#)

[TesseraTileBase](#)

**TesseraSquareTile**

Inherited Members

[TesseraTileBase.palette](#)

[TesseraTileBase.faceDetails](#)

[TesseraTileBase.offsets](#)

[TesseraTileBase.center](#)

[TesseraTileBase.tileSize](#)

[TesseraTileBase.rotatable](#)

[TesseraTileBase.reflectable](#)

[TesseraTileBase.rotationGroupType](#)

[TesseraTileBase.symmetric](#)

[TesseraTileBase.instantiateChildrenOnly](#)

[TesseraTileBase.Get\(Vector3Int, CellFaceDir\)](#)

[TesseraTileBase.TryGet\(Vector3Int, CellFaceDir, FaceDetails\)](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraSquareTile : TesseraTileBase
```

Constructors

[TesseraSquareTile\(\)](#)

Declaration

```
public TesseraSquareTile()
```

Properties

[CellType](#)

Declaration

```
public override ICellType CellType { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ICellType</a>	

Overrides

[TesseraTileBase.CellType](#)

Methods

[AddOffset\(Vector3Int\)](#)

Configures the tile as a "big" tile that occupies several cells. Keeps [offsets](#) and [faceDetails](#) in sync.

Declaration

```
public override void AddOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Overrides

[TesseraTileBase.AddOffset\(Vector3Int\)](#)

**GetBounds()**

Declaration

```
public BoundsInt GetBounds()
```

Returns

TYPE	DESCRIPTION
BoundsInt	

**RemoveOffset(Vector3Int)**

Configures the tile as a "big" tile that occupies several cells. Keeps [offsets](#) and [faceDetails](#) in sync.

Declaration

```
public override void RemoveOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Overrides

[TesseraTileBase.RemoveOffset\(Vector3Int\)](#)

# Class TesseraTile

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

Inheritance

[Object](#)

[TesseraTileBase](#)

[TesseraTile](#)

Inherited Members

[TesseraTileBase.palette](#)

[TesseraTileBase.faceDetails](#)

[TesseraTileBase.offsets](#)

[TesseraTileBase.center](#)

[TesseraTileBase.tileSize](#)

[TesseraTileBase.rotatable](#)

[TesseraTileBase.reflectable](#)

[TesseraTileBase.rotationGroupType](#)

[TesseraTileBase.symmetric](#)

[TesseraTileBase.instantiateChildrenOnly](#)

[TesseraTileBase.Get\(Vector3Int, CellFaceDir\)](#)

[TesseraTileBase.TryGet\(Vector3Int, CellFaceDir, FaceDetails\)](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraTile : TesseraTileBase
```

Constructors

[TesseraTile\(\)](#)

Declaration

```
public TesseraTile()
```

Properties

[CellType](#)

Declaration

```
public override ICellType CellType { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ICellType</a>	

Overrides

[TesseraTileBase.CellType](#)

Methods

[AddOffset\(Vector3Int\)](#)

Configures the tile as a "big" tile that occupies several cells. Keeps [offsets](#) and [faceDetails](#) in sync.

Declaration

```
public override void AddOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Overrides

[TesseraTileBase.AddOffset\(Vector3Int\)](#)

**GetBounds()**

Declaration

```
public BoundsInt GetBounds()
```

Returns

TYPE	DESCRIPTION
BoundsInt	

**RemoveOffset(Vector3Int)**

Configures the tile as a "big" tile that occupies several cells. Keeps [offsets](#) and [faceDetails](#) in sync.

Declaration

```
public override void RemoveOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Overrides

[TesseraTileBase.RemoveOffset\(Vector3Int\)](#)

# Class TesseraTileBase

Inheritance

[Object](#)

[TesseraTileBase](#)

[TesseraHexTile](#)

[TesseraSquareTile](#)

[TesseraTile](#)

[TesseraTrianglePrismTile](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class TesseraTileBase : MonoBehaviour
```

Fields

center

Where the center of tile is. For big tiles that occupy more than one cell, it's the center of the cell with offset (0, 0, 0). The tile may not actually occupy that cell!

Declaration

```
public Vector3 center
```

Field Value

TYPE	DESCRIPTION
Vector3	

faceDetails

A list of outward facing faces. For a normal cube tile, there are 6 faces. Each face contains adjacency information that indicates what other tiles can connect to it. It is recommended you only edit this via the Unity Editor, or [Get\(Vector3Int, CellFaceDir\)](#) and [AddOffset\(Vector3Int\)](#)

Declaration

```
public List<OrientedFace> faceDetails
```

Field Value

TYPE	DESCRIPTION
List<OrientedFace>	

instantiateChildrenOnly

If set, when being instantiated by a Generator, only children will get constructed. If there are no children, then this effectively disables the tile from instantiation.

Declaration

```
public bool instantiateChildrenOnly
```

Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
Boolean	

## offsets

A list of cells that this tile occupies. For a normal cube tile, this just contains Vector3Int.zero, but it will be more for "big" tiles. It is recommended you only edit this via the Unity Editor, or [AddOffset\(Vector3Int\)](#) and [RemoveOffset\(Vector3Int\)](#)

### Declaration

```
public List<Vector3Int> offsets
```

### Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
List<Vector3Int>	

## palette

Set this to control the colors and names used for painting on the tile. Defaults to [defaultPalette](#).

### Declaration

```
public TesseraPalette palette
```

### Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
TesseraPalette	

## reflectable

If true, when generating, reflections in the x-axis will be used.

### Declaration

```
public bool reflectable
```

### Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
Boolean	

## rotatable

If true, when generating, rotations of the tile will be used.

### Declaration

```
public bool rotatable
```

### Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
Boolean	

TYPE	DESCRIPTION

## rotationGroupType

If rotatable is on, specifies what sorts of rotations are used.

Declaration

```
public RotationGroupType rotationGroupType
```

Field Value

TYPE	DESCRIPTION
RotationGroupType	

## symmetric

If true, Tessera assumes that the tile paint matches the the symmetry of the tile. Disable this if there are important details of your tile that the paint doesn't show. Turning symmetric on can have some performance benefits, and affects the behaviour of the mirror constraint.

Declaration

```
public bool symmetric
```

Field Value

TYPE	DESCRIPTION
Boolean	

## tileSize

The size of one cell in the tile. NB: This field is only used in the Editor - you must set `tileSize` to match.

Declaration

```
public Vector3 tileSize
```

Field Value

TYPE	DESCRIPTION
Vector3	

## Properties

### CellType

Declaration

```
public abstract ICellType CellType { get; }
```

Property Value

TYPE	DESCRIPTION
ICellType	

## Methods

### AddOffset(Vector3Int)

Configures the tile as a "big" tile that occupies several cells. Keeps [offsets](#) and [faceDetails](#) in sync.

#### Declaration

```
public abstract void AddOffset(Vector3Int o)
```

#### Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

### Get(Vector3Int, CellFaceDir)

Finds the face details for a cell with a given offset.

#### Declaration

```
public FaceDetails Get(Vector3Int offset, CellFaceDir faceDir)
```

#### Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellFaceDir	faceDir	

#### Returns

TYPE	DESCRIPTION
FaceDetails	

### RemoveOffset(Vector3Int)

Configures the tile as a "big" tile that occupies several cells. Keeps [offsets](#) and [faceDetails](#) in sync.

#### Declaration

```
public abstract void RemoveOffset(Vector3Int o)
```

#### Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

### TryGet(Vector3Int, CellFaceDir, out FaceDetails)

Finds the face details for a cell with a given offset.

## Declaration

```
public bool TryGet(Vector3Int offset, CellFaceDir faceDir, out FaceDetails details)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellFaceDir	faceDir	
FaceDetails	details	

## Returns

TYPE	DESCRIPTION
Boolean	

# Class TesseraTileInstance

Represents a request to instantiate a TesseraTile, post generation.

Inheritance

## Object

### TesseraTileInstance

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraTileInstance
```

## Properties

### Cell

The grid cell this instance fills. (for big tiles, this is the cell of the first offset)

Declaration

```
public Vector3Int Cell { get; }
```

Property Value

TYPE	DESCRIPTION
Vector3Int	

### CellRotation

The rotation this instance is placed at (for big tiles, this is the cell of the first offset)

Declaration

```
public CellRotation CellRotation { get; }
```

Property Value

TYPE	DESCRIPTION
CellRotation	

### CellRotations

The rotations this instance instance, in the same order as the tile offsets. Most grids will have same rotation for all offsets.

Declaration

```
public CellRotation[] CellRotations { get; }
```

Property Value

TYPE	DESCRIPTION
CellRotation[]	

### Cells

The cells this instance fills, in the same order as the tile offsets.

## Declaration

```
public Vector3Int[] Cells { get; }
```

## Property Value

TYPE	DESCRIPTION
Vector3Int[]	

## LocalPosition

### Declaration

```
public Vector3 LocalPosition { get; }
```

## Property Value

TYPE	DESCRIPTION
Vector3	

## LocalRotation

### Declaration

```
public Quaternion LocalRotation { get; }
```

## Property Value

TYPE	DESCRIPTION
Quaternion	

## LocalScale

### Declaration

```
public Vector3 LocalScale { get; }
```

## Property Value

TYPE	DESCRIPTION
Vector3	

## LossyScale

### Declaration

```
public Vector3 LossyScale { get; }
```

## Property Value

TYPE	DESCRIPTION
Vector3	

## MeshDeformation

Gives a mesh deformation from tile space to generator space. Null for grids that do not have deformed tiles.

## Declaration

```
public MeshDeformation MeshDeformation { get; }
```

## Property Value

TYPE	DESCRIPTION
MeshDeformation	

## Position

### Declaration

```
public Vector3 Position { get; }
```

## Property Value

TYPE	DESCRIPTION
Vector3	

## Rotation

### Declaration

```
public Quaternion Rotation { get; }
```

## Property Value

TYPE	DESCRIPTION
Quaternion	

## Tile

### Declaration

```
public TesseraTileBase Tile { get; }
```

## Property Value

TYPE	DESCRIPTION
TesseraTileBase	

## Methods

### Align(TRS)

Sets Position/Rotation/Scale from the local versions and a given transform

### Declaration

```
public void Align(TRS transform)
```

## Parameters

TYPE	NAME	DESCRIPTION
TRS	transform	

## Clone()

### Declaration

```
public TesseraTileInstance Clone()
```

### Returns

TYPE	DESCRIPTION
TesseraTileInstance	

# Class TesseraTilemapOutput

Attach this to a TesseraGenerator to output the tiles to a Unity Tilemap component instead of directly instantiating them.

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

[TesseraTilemapOutput](#)

Implements

[ITesseraTileOutput](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraTilemapOutput : MonoBehaviour, ITesseraTileOutput
```

Fields

**tilemap**

The tilemap to write results to.

Declaration

```
public Tilemap tilemap
```

Field Value

TYPE	DESCRIPTION
Tilemap	

**useSprites**

If true, TesseraTiles that have a SpriteRenderer will be recorded to the Tilemap as that sprite. This is more efficient, but you will lose any other components on the object.

Declaration

```
public bool useSprites
```

Field Value

TYPE	DESCRIPTION
Boolean	

**useWorld**

If true, tiles will be transformed to align with the world space position of the generator.

Declaration

```
public bool useWorld
```

## Field Value

TYPE	DESCRIPTION
Boolean	

## Properties

### IsEmpty

Declaration

```
public bool IsEmpty { get; }
```

### Property Value

TYPE	DESCRIPTION
Boolean	

### SupportsIncremental

Declaration

```
public bool SupportsIncremental { get; }
```

### Property Value

TYPE	DESCRIPTION
Boolean	

## Methods

### ClearTiles(IEngineInterface)

Declaration

```
public void ClearTiles(IEngineInterface engine)
```

### Parameters

TYPE	NAME	DESCRIPTION
IEngineInterface	engine	

### UpdateTiles(TesseraCompletion, IEngineInterface)

Declaration

```
public void UpdateTiles(TesseraCompletion completion, IEngineInterface engine)
```

### Parameters

TYPE	NAME	DESCRIPTION
TesseraCompletion	completion	
IEngineInterface	engine	

## Implements

ITesseraTileOutput

# Class TesseraTransformedTile

Defines a Unity tile that has a specific transform applied to it. Used by [TesseraTilemapOutput](#)

Inheritance

[Object](#)

TesseraTransformedTile

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraTransformedTile : Tile
```

Fields

localScale

Declaration

```
public Vector3 localScale
```

Field Value

TYPE	DESCRIPTION
Vector3	

position

Declaration

```
public Vector3 position
```

Field Value

TYPE	DESCRIPTION
Vector3	

rotation

Declaration

```
public Quaternion rotation
```

Field Value

TYPE	DESCRIPTION
Quaternion	

useWorld

Declaration

```
public bool useWorld
```

Field Value

TYPE	DESCRIPTION
Boolean	

## Methods

### StartUp(Vector3Int, ITilemap, GameObject)

Declaration

```
public override bool StartUp(Vector3Int position, ITilemap tilemap, GameObject go)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	position	
ITilemap	tilemap	
GameObject	go	

Returns

TYPE	DESCRIPTION
Boolean	

# Class TesseraTrianglePrismTile

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

Inheritance

[Object](#)

[TesseraTileBase](#)

**TesseraTrianglePrismTile**

Inherited Members

[TesseraTileBase.palette](#)

[TesseraTileBase.faceDetails](#)

[TesseraTileBase.offsets](#)

[TesseraTileBase.center](#)

[TesseraTileBase.tileSize](#)

[TesseraTileBase.rotatable](#)

[TesseraTileBase.reflectable](#)

[TesseraTileBase.rotationGroupType](#)

[TesseraTileBase.symmetric](#)

[TesseraTileBase.instantiateChildrenOnly](#)

[TesseraTileBase.Get\(Vector3Int, CellFaceDir\)](#)

[TesseraTileBase.TryGet\(Vector3Int, CellFaceDir, FaceDetails\)](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraTrianglePrismTile : TesseraTileBase
```

Constructors

[TesseraTrianglePrismTile\(\)](#)

Declaration

```
public TesseraTrianglePrismTile()
```

Properties

[CellType](#)

Declaration

```
public override ICellType CellType { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ICellType</a>	

Overrides

[TesseraTileBase.CellType](#)

Methods

[AddOffset\(Vector3Int\)](#)

Configures the tile as a "big" tile that occupies several cells. Keeps [offsets](#) and [faceDetails](#) in sync.

Declaration

```
public override void AddOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Overrides

[TesseraTileBase.AddOffset\(Vector3Int\)](#)

**GetBounds()**

Declaration

```
public BoundsInt GetBounds()
```

Returns

TYPE	DESCRIPTION
BoundsInt	

**RemoveOffset(Vector3Int)**

Configures the tile as a "big" tile that occupies several cells. Keeps [offsets](#) and [faceDetails](#) in sync.

Declaration

```
public override void RemoveOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Overrides

[TesseraTileBase.RemoveOffset\(Vector3Int\)](#)

# Class TesseraVolume

Inheritance

Object

TesseraVolume

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraVolume : MonoBehaviour
```

Fields

generator

No effect on behaviour, setting this improves the UI in the Unity inspector.

Declaration

```
public TesseraGenerator generator
```

Field Value

TYPE	DESCRIPTION
<a href="#">TesseraGenerator</a>	

invertArea

If false, affect all cells inside the volume's colliders. If true, affect all cells outside.

Declaration

```
public bool invertArea
```

Field Value

TYPE	DESCRIPTION
<a href="#">Boolean</a>	

tiles

The list of tiles to filter on.

Declaration

```
public List<TesseraTileBase> tiles
```

Field Value

TYPE	DESCRIPTION
<a href="#">List&lt;TesseraTileBase&gt;</a>	

volumeType

Controls the behaviour of this volume

Declaration

```
public VolumeType volumeType
```

Field Value

TYPE	DESCRIPTION
VolumeType	

# Class TesseraVolumeFilter

Inheritance

[Object](#)

TesseraVolumeFilter

Implements

[ITesseraInitialConstraint](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraVolumeFilter : ITesseraInitialConstraint
```

Fields

volumeType

Declaration

```
public VolumeType volumeType
```

Field Value

TYPE	DESCRIPTION
<a href="#">VolumeType</a>	

Properties

Name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">String</a>	

Implements

[ITesseraInitialConstraint](#)

# Enum TesseraWfcAlgorithm

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum TesseraWfcAlgorithm
```

Fields

NAME	DESCRIPTION
Ac3	
Ac4	
Default	
OneStep	

# Class TileEntry

Specifies a tile to be used by [TesseraGenerator](#)

Inheritance

[Object](#)

[TileEntry](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]  
public class TileEntry
```

Fields

**tile**

The tile to use

Declaration

```
public TesseraTileBase tile
```

Field Value

TYPE	DESCRIPTION
<a href="#">TesseraTileBase</a>	

**weight**

The weight controls the relative probability of this tile being selected. I.e. tile with weight of 2.0 is twice common in the generation than a tile with weight 1.0.

Declaration

```
public float weight
```

Field Value

TYPE	DESCRIPTION
<a href="#">Single</a>	

# Class TileList

Inheritance

[Object](#)

[TileList](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]  
public class TileList
```

Fields

[tiles](#)

Declaration

```
public List<TesseraTileBase> tiles
```

Field Value

TYPE	DESCRIPTION
<a href="#">List&lt;TesseraTileBase&gt;</a>	

# Class TileMapping

Inheritance

[Object](#)

[TileMapping](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class TileMapping
```

Fields

[from](#)

Declaration

```
public TesseraTileBase from
```

Field Value

TYPE	DESCRIPTION
TesseraTileBase	

[instantiateChildrenOnly](#)

Declaration

```
public bool instantiateChildrenOnly
```

Field Value

TYPE	DESCRIPTION
Boolean	

[to](#)

Declaration

```
public GameObject to
```

Field Value

TYPE	DESCRIPTION
GameObject	

# Class TriangleInterpolation

Inheritance

[Object](#)

[TriangleInterpolation](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public static class TriangleInterpolation
```

## Methods

[Interpolate\(Vector2, Vector2, Vector2\)](#)

Declaration

```
public static Func<Vector3, Vector2> Interpolate(Vector2 v1, Vector2 v2, Vector2 v3)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2	v1	
Vector2	v2	
Vector2	v3	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector2>	

[Interpolate\(Vector2, Vector2, Vector2, Vector2, Vector2, Vector2\)](#)

Declaration

```
public static Func<Vector3, Vector2> Interpolate(Vector2 v1, Vector2 v2, Vector2 v3, Vector2 v4, Vector2 v5, Vector2 v6)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2	v1	
Vector2	v2	
Vector2	v3	
Vector2	v4	
Vector2	v5	
Vector2	v6	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector2>	

## Interpolate(Vector3, Vector3, Vector3)

Declaration

```
public static Func<Vector3, Vector3> Interpolate(Vector3 v1, Vector3 v2, Vector3 v3)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	v1	
Vector3	v2	
Vector3	v3	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector3>	

## Interpolate(Vector3, Vector3, Vector3, Vector3, Vector3, Vector3)

Declaration

```
public static Func<Vector3, Vector3> Interpolate(Vector3 v1, Vector3 v2, Vector3 v3, Vector3 v4, Vector3 v5, Vector3 v6)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	v1	
Vector3	v2	
Vector3	v3	
Vector3	v4	
Vector3	v5	
Vector3	v6	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector3>	

## Interpolate(Vector4, Vector4, Vector4)

## Declaration

```
public static Func<Vector3, Vector4> Interpolate(Vector4 v1, Vector4 v2, Vector4 v3)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector4	v1	
Vector4	v2	
Vector4	v3	

## Returns

TYPE	DESCRIPTION
Func<Vector3, Vector4>	

## Interpolate(Vector4, Vector4, Vector4, Vector4, Vector4, Vector4)

## Declaration

```
public static Func<Vector3, Vector4> Interpolate(Vector4 v1, Vector4 v2, Vector4 v3, Vector4 v4, Vector4 v5, Vector4 v6)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector4	v1	
Vector4	v2	
Vector4	v3	
Vector4	v4	
Vector4	v5	
Vector4	v6	

## Returns

TYPE	DESCRIPTION
Func<Vector3, Vector4>	

## InterpolateNormal(MeshData, Int32, Int32)

## Declaration

```
public static Func<Vector3, Vector3> InterpolateNormal(MeshData mesh, int submesh, int face)
```

## Parameters

TYPE	NAME	DESCRIPTION
MeshData	mesh	
Int32	submesh	
Int32	face	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector3>	

### InterpolatePosition(MeshData, Int32, Int32, Single, Single)

Declaration

```
public static Func<Vector3, Vector3> InterpolatePosition(MeshData mesh, int submesh, int face, float
meshOffset1, float meshOffset2)
```

Parameters

TYPE	NAME	DESCRIPTION
MeshData	mesh	
Int32	submesh	
Int32	face	
Single	meshOffset1	
Single	meshOffset2	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector3>	

### InterpolateTangent(MeshData, Int32, Int32)

Declaration

```
public static Func<Vector3, Vector4> InterpolateTangent(MeshData mesh, int submesh, int face)
```

Parameters

TYPE	NAME	DESCRIPTION
MeshData	mesh	
Int32	submesh	
Int32	face	

Returns

TYPE	DESCRIPTION
Func<Vector3, Vector4>	

## InterpolateUv(MeshData, Int32, Int32)

### Declaration

```
public static Func<Vector3, Vector2> InterpolateUv(MeshData mesh, int submesh, int face)
```

### Parameters

TYPE	NAME	DESCRIPTION
MeshData	mesh	
Int32	submesh	
Int32	face	

### Returns

TYPE	DESCRIPTION
Func<Vector3, Vector2>	

# Class TrianglePrismCellType

## NOTE

This class is available only in Tessera Pro

Inheritance

[Object](#)

[TrianglePrismCellType](#)

Implements

[ICellType](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TrianglePrismCellType : ICellType
```

Properties

Instance

Declaration

```
public static TrianglePrismCellType Instance { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">TrianglePrismCellType</a>	

Methods

[FindPath\(Vector3Int, Vector3Int\)](#)

Declaration

```
public IEnumerable<CellFaceDir> FindPath(Vector3Int startOffset, Vector3Int endOffset)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">Vector3Int</a>	startOffset	
<a href="#">Vector3Int</a>	endOffset	

Returns

TYPE	DESCRIPTION
<a href="#">IEnumerable&lt;CellFaceDir&gt;</a>	

[GetCellCenter\(Vector3Int, Vector3, Vector3\)](#)

Declaration

```
public Vector3 GetCellCenter(Vector3Int offset, Vector3 center, Vector3 tileSize)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
Vector3	center	
Vector3	tileSize	

## Returns

TYPE	DESCRIPTION
Vector3	

## GetDisplayName(CellFaceDir)

### Declaration

```
public string GetDisplayName(CellFaceDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
CellFaceDir	dir	

## Returns

TYPE	DESCRIPTION
String	

## GetFaceDirPairs()

### Declaration

```
public IEnumerable<(CellFaceDir, CellFaceDir)> GetFaceDirPairs()
```

## Returns

TYPE	DESCRIPTION
IEnumerable<(T1, T2)<CellFaceDir, CellFaceDir>>	

## GetFaceDirs()

### Declaration

```
public IEnumerable<CellFaceDir> GetFaceDirs()
```

## Returns

TYPE	DESCRIPTION
IEnumerable<CellFaceDir>	

## GetIdentity()

### Declaration

```
public CellRotation GetIdentity()
```

Returns

TYPE	DESCRIPTION
CellRotation	

## GetMatrix(CellRotation)

Declaration

```
public Matrix4x4 GetMatrix(CellRotation rotation)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	rotation	

Returns

TYPE	DESCRIPTION
Matrix4x4	

## GetRotations(Boolean, Boolean, RotationGroupType)

Declaration

```
public IList<CellRotation> GetRotations(bool rotatable = true, bool reflectable = true, RotationGroupType rotationGroupType = RotationGroupType.All)
```

Parameters

TYPE	NAME	DESCRIPTION
Boolean	rotatable	
Boolean	reflectable	
RotationGroupType	rotationGroupType	

Returns

TYPE	DESCRIPTION
IList<CellRotation>	

## Invert(CellFaceDir)

Declaration

```
public CellFaceDir Invert(CellFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
CellFaceDir	faceDir	

Returns

TYPE	DESCRIPTION
CellFaceDir	

## Invert(CellRotation)

Declaration

```
public CellRotation Invert(CellRotation a)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	a	

Returns

TYPE	DESCRIPTION
CellRotation	

## Multiply(CellRotation, CellRotation)

Declaration

```
public CellRotation Multiply(CellRotation a, CellRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	a	
CellRotation	b	

Returns

TYPE	DESCRIPTION
CellRotation	

## Realign(ISet<Vector3Int>, CellRotation)

Declaration

```
public IDictionary<Vector3Int, Vector3Int> Realign(ISet<Vector3Int> shape, CellRotation rotation)
```

Parameters

TYPE	NAME	DESCRIPTION
ISet<Vector3Int>	shape	

TYPE	NAME	DESCRIPTION
CellRotation	rotation	

Returns

TYPE	DESCRIPTION
IDictionary<Vector3Int, Vector3Int>	

## Rotate(CellFaceDir, CellRotation)

Declaration

```
public CellFaceDir Rotate(CellFaceDir faceDir, CellRotation rotation)
```

Parameters

TYPE	NAME	DESCRIPTION
CellFaceDir	faceDir	
CellRotation	rotation	

Returns

TYPE	DESCRIPTION
CellFaceDir	

## RotateBy(CellFaceDir, FaceDetails, CellRotation)

Declaration

```
public (CellFaceDir, FaceDetails) RotateBy(CellFaceDir faceDir, FaceDetails faceDetails, CellRotation rotation)
```

Parameters

TYPE	NAME	DESCRIPTION
CellFaceDir	faceDir	
FaceDetails	faceDetails	
CellRotation	rotation	

Returns

TYPE	DESCRIPTION
(T1, T2)<CellFaceDir, FaceDetails>	

## TryMove(Vector3Int, CellFaceDir, out Vector3Int)

Declaration

```
public bool TryMove(Vector3Int offset, CellFaceDir dir, out Vector3Int dest)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellFaceDir	dir	
Vector3Int	dest	

## Returns

TYPE	DESCRIPTION
Boolean	

**TryMoveByOffset(Vector3Int, Vector3Int, Vector3Int, CellRotation, out Vector3Int)**

## Declaration

```
public bool TryMoveByOffset(Vector3Int startCell, Vector3Int startOffset, Vector3Int destOffset, CellRotation rotation, out Vector3Int destCell)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	startCell	
Vector3Int	startOffset	
Vector3Int	destOffset	
CellRotation	rotation	
Vector3Int	destCell	

## Returns

TYPE	DESCRIPTION
Boolean	

## Implements

[ICellType](#)

# Enum TrianglePrismFaceDir

## NOTE

This class is available only in Tessera Pro

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

## Syntax

```
public enum TrianglePrismFaceDir
```

## Fields

NAME	DESCRIPTION
Back	
BackLeft	
BackRight	
Down	
Forward	
ForwardLeft	
ForwardRight	
Up	

# Class TrianglePrismFaceDirExtensions

## NOTE

This class is available only in Tessera Pro

## Inheritance

### Object

### TrianglePrismFaceDirExtensions

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

## Syntax

```
public static class TrianglePrismFaceDirExtensions
```

## Methods

### Forward(TrianglePrismFaceDir)

#### Declaration

```
public static Vector3 Forward(this TrianglePrismFaceDir dir)
```

#### Parameters

TYPE	NAME	DESCRIPTION
TrianglePrismFaceDir	dir	

#### Returns

TYPE	DESCRIPTION
Vector3	

### GetSide(TrianglePrismFaceDir)

#### Declaration

```
public static int GetSide(this TrianglePrismFaceDir dir)
```

#### Parameters

TYPE	NAME	DESCRIPTION
TrianglePrismFaceDir	dir	

#### Returns

TYPE	DESCRIPTION
Int32	

### IsUpDown(TrianglePrismFaceDir)

#### Declaration

```
public static bool IsUpDown(this TrianglePrismFaceDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
TrianglePrismFaceDir	dir	

## Returns

TYPE	DESCRIPTION
Boolean	

IsValid(TrianglePrismFaceDir, Boolean)

## Declaration

```
public static bool IsValid(this TrianglePrismFaceDir dir, bool pointsUp)
```

## Parameters

TYPE	NAME	DESCRIPTION
TrianglePrismFaceDir	dir	
Boolean	pointsUp	

## Returns

TYPE	DESCRIPTION
Boolean	

IsValid(TrianglePrismFaceDir, Vector3Int)

## Declaration

```
public static bool IsValid(this TrianglePrismFaceDir dir, Vector3Int offset)
```

## Parameters

TYPE	NAME	DESCRIPTION
TrianglePrismFaceDir	dir	
Vector3Int	offset	

## Returns

TYPE	DESCRIPTION
Boolean	

OffsetDelta(TrianglePrismFaceDir)

## Declaration

```
public static Vector3Int OffsetDelta(this TrianglePrismFaceDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
TrianglePrismFaceDir	dir	

Returns

TYPE	DESCRIPTION
Vector3Int	

## Up(TrianglePrismFaceDir)

Declaration

```
public static Vector3 Up(this TrianglePrismFaceDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
TrianglePrismFaceDir	dir	

Returns

TYPE	DESCRIPTION
Vector3	

# Class TrianglePrismGeometryUtils

## NOTE

This class is available only in Tessera Pro

### Inheritance

### Object

### TrianglePrismGeometryUtils

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

### Syntax

```
public static class TrianglePrismGeometryUtils
```

### Methods

#### CoordRotate(TriangleRotation, Vector3Int)

##### Declaration

```
public static Vector3Int CoordRotate(TriangleRotation rotation, Vector3Int coords)
```

##### Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	rotation	
Vector3Int	coords	

##### Returns

TYPE	DESCRIPTION
Vector3Int	

#### FindCell(Vector3, Vector3, Vector3, out Vector3Int)

##### Declaration

```
public static bool FindCell(Vector3 origin, Vector3 tileSize, Vector3 position, out Vector3Int cell)
```

##### Parameters

TYPE	NAME	DESCRIPTION
Vector3	origin	
Vector3	tileSize	
Vector3	position	
Vector3Int	cell	

##### Returns

TYPE	DESCRIPTION
Boolean	

## FromSide(Int32)

Declaration

```
public static TrianglePrismFaceDir FromSide(int side)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	side	

Returns

TYPE	DESCRIPTION
TrianglePrismFaceDir	

## FromTriCoords(Vector3Int, Int32)

Declaration

```
public static Vector3Int? FromTriCoords(Vector3Int coords, int y = 0)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	coords	
Int32	y	

Returns

TYPE	DESCRIPTION
Nullable<Vector3Int>	

## GetCellCenter(Vector3Int, Vector3, Vector3)

Declaration

```
public static Vector3 GetCellCenter(Vector3Int cell, Vector3 origin, Vector3 tileSize)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	
Vector3	origin	
Vector3	tileSize	

Returns

TYPE	DESCRIPTION
Vector3	

## Pack(Vector2Int, Boolean, Int32)

Declaration

```
public static Vector3Int Pack(Vector2Int tri, bool pointsUp, int y)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2Int	tri	
Boolean	pointsUp	
Int32	y	

Returns

TYPE	DESCRIPTION
Vector3Int	

## PointsUp(Vector3Int)

Declaration

```
public static bool PointsUp(Vector3Int cell)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	

Returns

TYPE	DESCRIPTION
Boolean	

## Standardize(Vector2)

Declaration

```
public static Vector2 Standardize(Vector2 p)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2	p	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector2	

### ToTriCoords(Vector3Int)

Declaration

```
public static Vector3Int ToTriCoords(Vector3Int cell)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Vector3Int	cell	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector3Int	

### Unpack(Vector3Int)

Declaration

```
public static (Vector2Int, bool, int) Unpack(Vector3Int cell)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Vector3Int	cell	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
(T1, T2, T3)<Vector2Int, Boolean, Int32>	

### Unstandardize(Vector2)

Declaration

```
public static Vector2 Unstandardize(Vector2 p)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Vector2	p	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector2	

# Struct TriangleRotation

Represents rotations / reflections of a hexagon

## NOTE

This class is available only in Tessera Pro

Inherited Members

[ValueType.ToString\(\)](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public struct TriangleRotation
```

Properties

All

Declaration

```
public static readonly TriangleRotation[] All { get; }
```

Property Value

TYPE	DESCRIPTION
TriangleRotation[]	

Identity

Declaration

```
public static readonly TriangleRotation Identity { get; }
```

Property Value

TYPE	DESCRIPTION
TriangleRotation	

IsReflection

Declaration

```
public readonly bool IsReflection { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

ReflectX

Declaration

```
public static readonly TriangleRotation ReflectX { get; }
```

## Property Value

TYPE	DESCRIPTION
TriangleRotation	

## ReflectY

### Declaration

```
public static readonly TriangleRotation ReflectY { get; }
```

## Property Value

TYPE	DESCRIPTION
TriangleRotation	

## RotateCCW

### Declaration

```
public static readonly TriangleRotation RotateCCW { get; }
```

## Property Value

TYPE	DESCRIPTION
TriangleRotation	

## RotateCW

### Declaration

```
public static readonly TriangleRotation RotateCW { get; }
```

## Property Value

TYPE	DESCRIPTION
TriangleRotation	

## Rotation

### Declaration

```
public readonly int Rotation { get; }
```

## Property Value

TYPE	DESCRIPTION
Int32	

## Methods

### Equals(Object)

#### Declaration

```
public override bool Equals(object obj)
```

## Parameters

TYPE	NAME	DESCRIPTION
Object	obj	

## Returns

TYPE	DESCRIPTION
Boolean	

## Overrides

[ValueType.Equals\(Object\)](#)

## GetHashCode()

### Declaration

```
public override int GetHashCode()
```

## Returns

TYPE	DESCRIPTION
Int32	

## Overrides

[ValueType.GetHashCode\(\)](#)

## Invert()

### Declaration

```
public TriangleRotation Invert()
```

## Returns

TYPE	DESCRIPTION
TriangleRotation	

## RotateCCW60(Int32)

### Declaration

```
public static TriangleRotation RotateCCW60(int i)
```

## Parameters

TYPE	NAME	DESCRIPTION
Int32	i	

## Returns

TYPE	DESCRIPTION
TriangleRotation	

## Operators

## Equality(TriangleRotation, TriangleRotation)

Declaration

```
public static bool operator ==(TriangleRotation a, TriangleRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	a	
TriangleRotation	b	

Returns

TYPE	DESCRIPTION
Boolean	

## Implicit(CellRotation to TriangleRotation)

Declaration

```
public static implicit operator TriangleRotation(CellRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	r	

Returns

TYPE	DESCRIPTION
TriangleRotation	

## Implicit(TriangleRotation to CellRotation)

Declaration

```
public static implicit operator CellRotation(TriangleRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	r	

Returns

TYPE	DESCRIPTION
CellRotation	

## Inequality(TriangleRotation, TriangleRotation)

Declaration

```
public static bool operator !=(TriangleRotation a, TriangleRotation b)
```

## Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	a	
TriangleRotation	b	

## Returns

TYPE	DESCRIPTION
Boolean	

## Multiply(TriangleRotation, Int32)

### Declaration

```
public static int operator *(TriangleRotation a, int side)
```

## Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	a	
Int32	side	

## Returns

TYPE	DESCRIPTION
Int32	

## Multiply(TriangleRotation, TrianglePrismFaceDir)

### Declaration

```
public static TrianglePrismFaceDir operator *(TriangleRotation rotation, TrianglePrismFaceDir faceDir)
```

## Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	rotation	
TrianglePrismFaceDir	faceDir	

## Returns

TYPE	DESCRIPTION
TrianglePrismFaceDir	

## Multiply(TriangleRotation, TriangleRotation)

### Declaration

```
public static TriangleRotation operator *(TriangleRotation a, TriangleRotation b)
```

## Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	a	
TriangleRotation	b	

## Returns

TYPE	DESCRIPTION
TriangleRotation	

# Class TRS

Represents a position / rotation and scale. Much like a Transform, but without the association with a unity object.

Inheritance

[Object](#)

[TRS](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class TRS
```

Constructors

[TRS\(Matrix4x4\)](#)

Declaration

```
public TRS(Matrix4x4 m)
```

Parameters

TYPE	NAME	DESCRIPTION
Matrix4x4	m	

[TRS\(Vector3\)](#)

Declaration

```
public TRS(Vector3 position)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	position	

[TRS\(Vector3, Quaternion, Vector3\)](#)

Declaration

```
public TRS(Vector3 position, Quaternion rotation, Vector3 scale)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	position	
Quaternion	rotation	
Vector3	scale	

Properties

[Position](#)

Declaration

```
public Vector3 Position { get; }
```

#### Property Value

TYPE	DESCRIPTION
Vector3	

#### Rotation

##### Declaration

```
public Quaternion Rotation { get; }
```

#### Property Value

TYPE	DESCRIPTION
Quaternion	

#### Scale

##### Declaration

```
public Vector3 Scale { get; }
```

#### Property Value

TYPE	DESCRIPTION
Vector3	

#### Methods

##### Local(Transform)

##### Declaration

```
public static TRS Local(Transform t)
```

##### Parameters

TYPE	NAME	DESCRIPTION
Transform	t	

##### Returns

TYPE	DESCRIPTION
TRS	

##### ToMatrix()

##### Declaration

```
public Matrix4x4 ToMatrix()
```

##### Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Matrix4x4	

## World(Transform)

### Declaration

```
public static TRS World(Transform t)
```

### Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Transform	t	

### Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
TRS	

## Operators

### Multiply(TRS, TRS)

### Declaration

```
public static TRS operator *(TRS a, TRS b)
```

### Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
TRS	a	
TRS	b	

### Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
TRS	

# Class UnityEngineInterface

Inheritance

[Object](#)

UnityEngineInterface

Implements

[IEngineInterface](#)

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public class UnityEngineInterface : IEngineInterface
```

Properties

Instance

Declaration

```
public static UnityEngineInterface Instance { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">UnityEngineInterface</a>	

Methods

[Destroy\(UnityEngine.Object\)](#)

Declaration

```
public void Destroy(UnityEngine.Object o)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">UnityEngine.Object</a>	<a href="#">o</a>	

[RegisterCompleteObjectUndo\(UnityEngine.Object\)](#)

Declaration

```
public void RegisterCompleteObjectUndo(UnityEngine.Object objectToUndo)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">UnityEngine.Object</a>	<a href="#">objectToUndo</a>	

[RegisterCreatedObjectUndo\(UnityEngine.Object\)](#)

Declaration

```
public void RegisterCreatedObjectUndo(UnityEngine.Object objectToUndo)
```

Parameters

TYPE	NAME	DESCRIPTION
UnityEngine.Object	objectToUndo	

Implements

[IEngineInterface](#)

# Enum VolumeType

Namespace: [Tessera](#)

Assembly: cs.temp.dll.dll

Syntax

```
public enum VolumeType
```

Fields

NAME	DESCRIPTION
MaskOut	Removes the cells inside the volume from generation
TilesetFilter	Restricts the set of tiles inside the volume