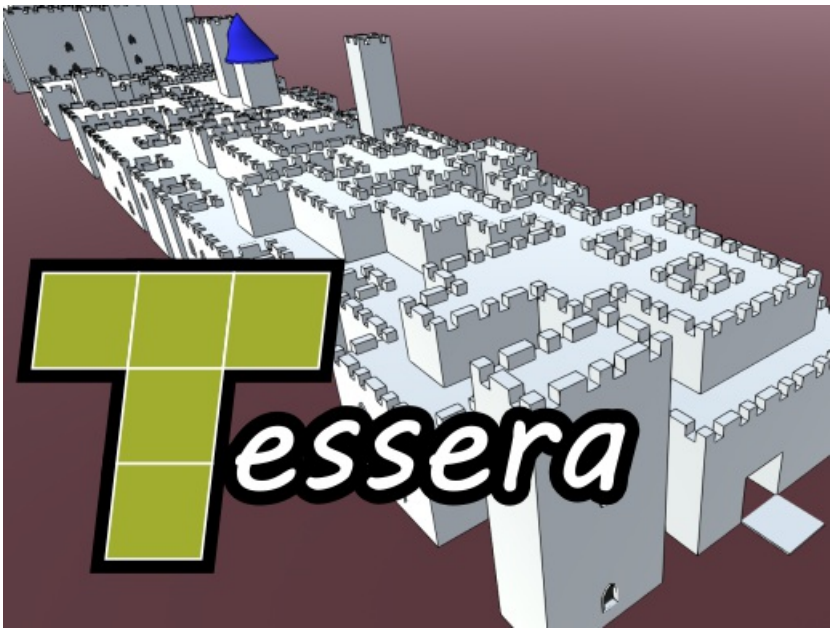
Table of Contents

TesseraTile

TesseraTileInstance

TesseraTilemapOutput

TileEntry



Tessera is a Unity addon for prodecurally generating 3d tile-based levels and builds. Get it here.

For help, contact boris@boristhebrave.com.

These docs contains tutorials and class based documentation for Tessera v2.2.1.

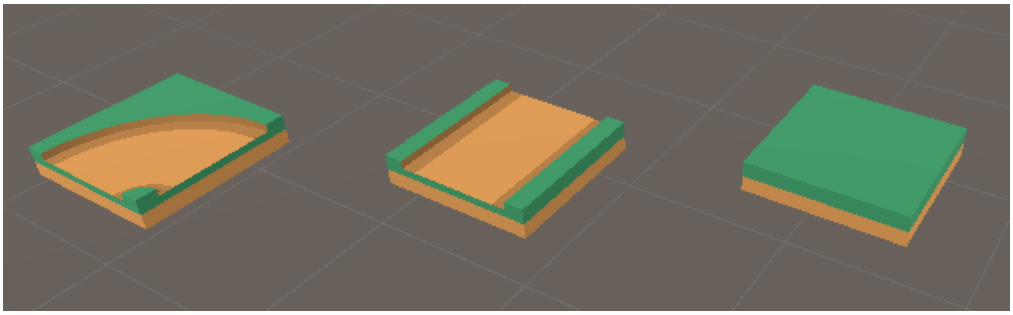
Getting Started Tutorial

Setup

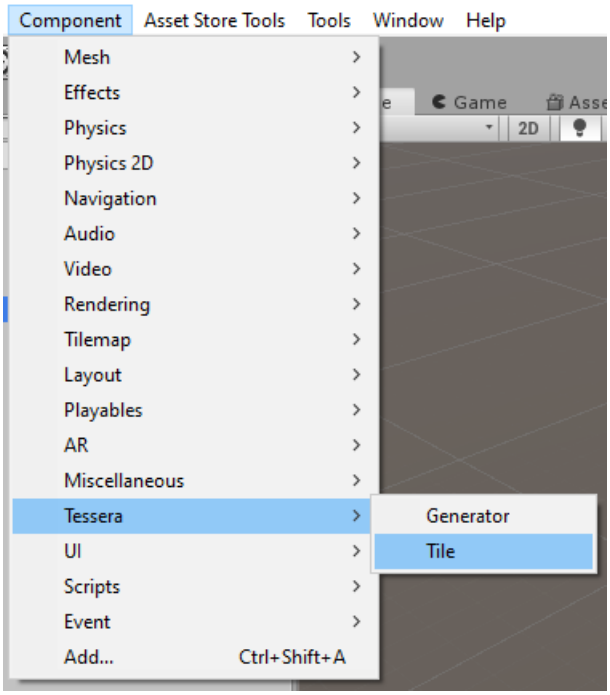
Start a new project. Then download Tessera from the Unity Asset Store and import it to your unity project. For this tutorial, we'll also use use Kenney's Tower Defense Kit so download that and add the `Models/FBX format` folder to your project assets.

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.



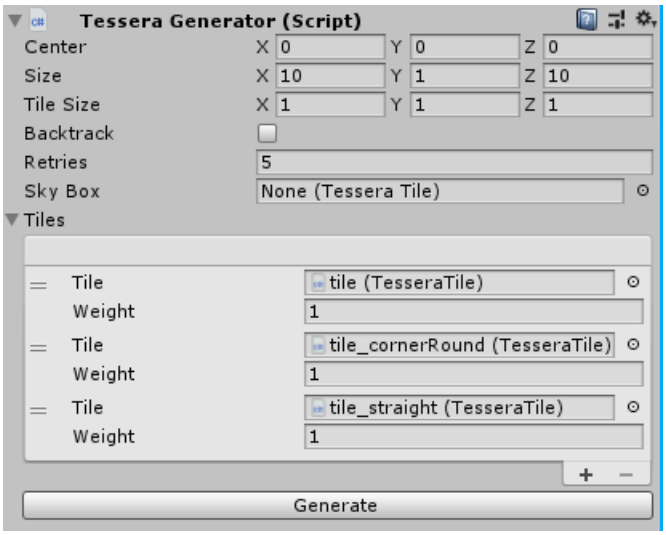
That's all for now, we'll configure the tiles later.

Creating the generator

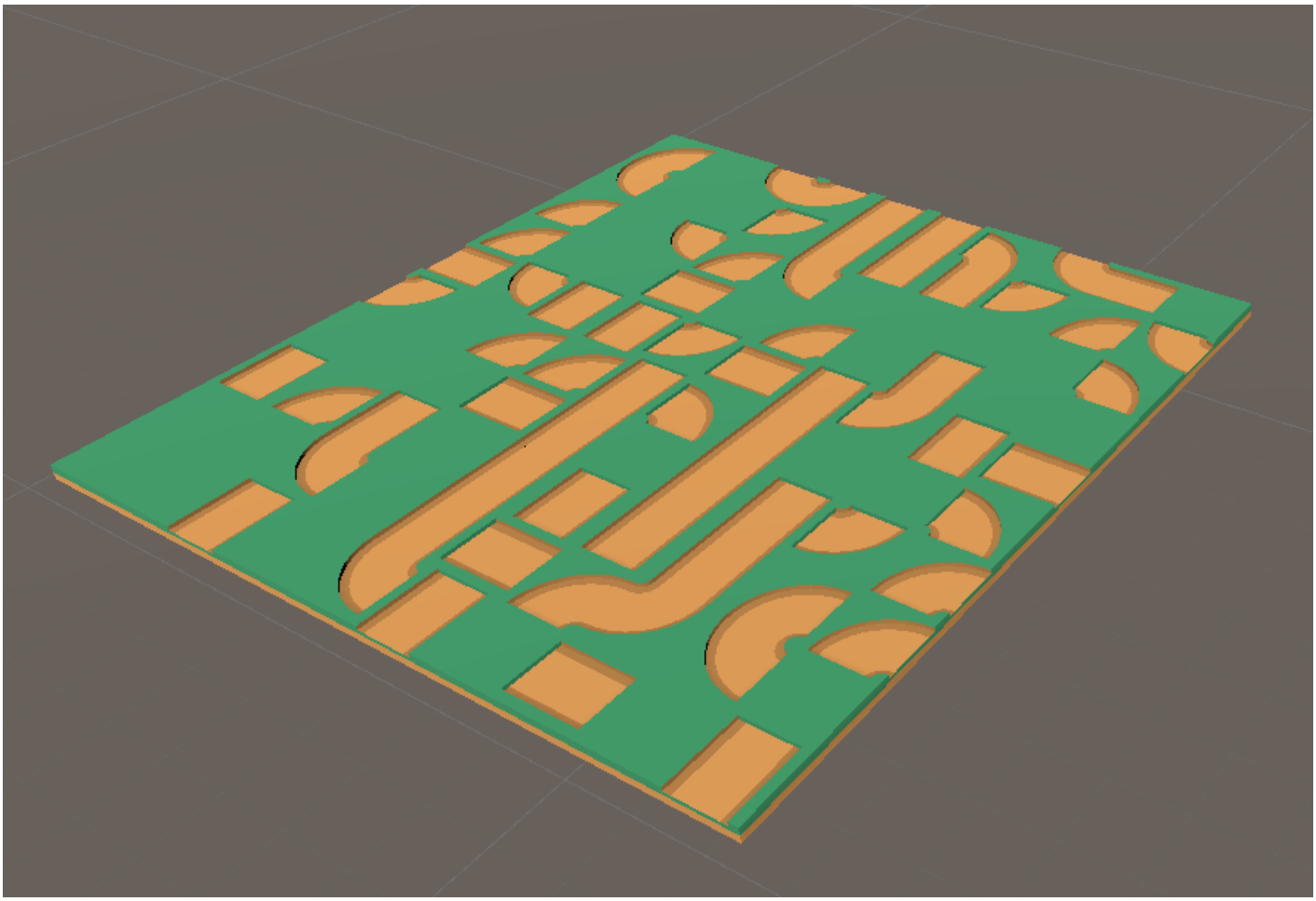
Next, create a new empty GameObject, and give it the `TesseraGenerator` component from the menu. Bring it up in the inspector. Add the tiles we created before to the list of tiles, either by dragging them from the hierarchy onto the Tiles section, or clicking the small plus button and selecting each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

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

Afterwards, your configuation 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:



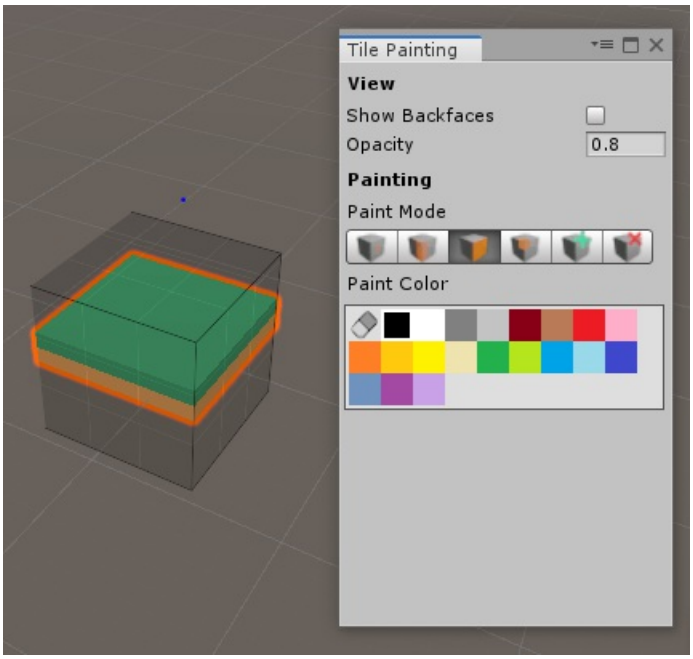
It's generated the tiles, but right now it doesn't know which tiles can be placed next to which other ones. So it has just placed them randomly. Hit undo to delete the created tiles.

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.  .

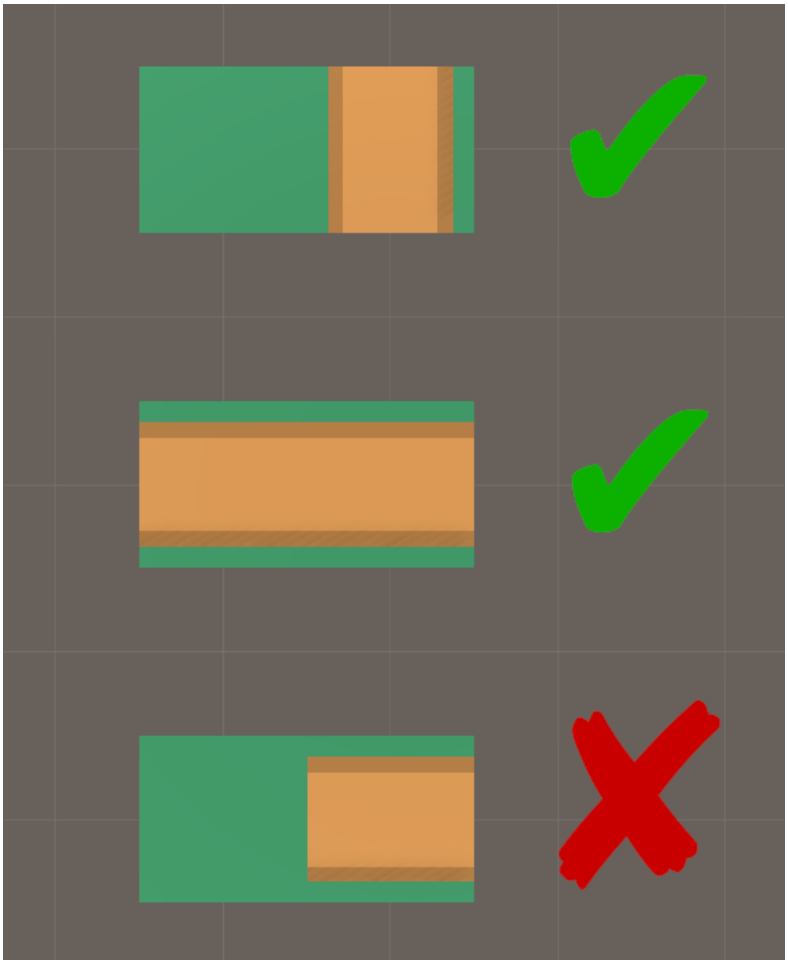
This should pop-up a "Tile Painting" window, and also show a semi-transparent cube around the tile.



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.

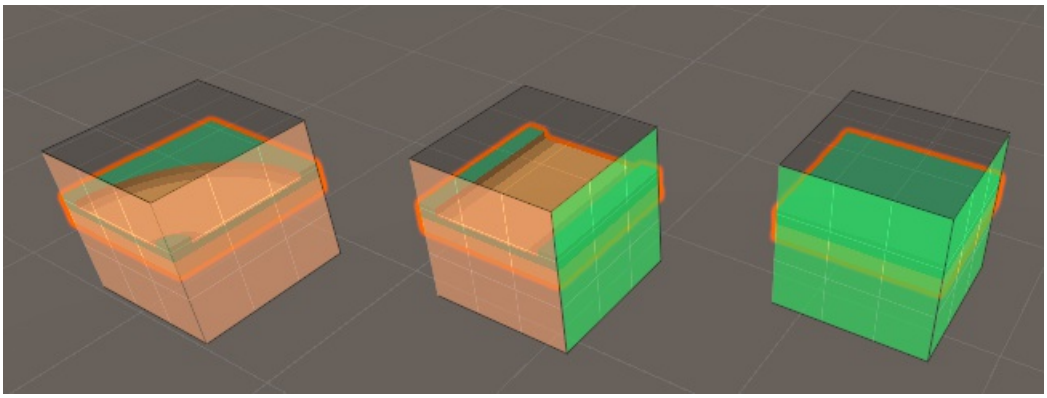
In this case, we want tiles to connect to each other if they are both grassy, or if they are both a path, but do not want paths to lead straight into grass.



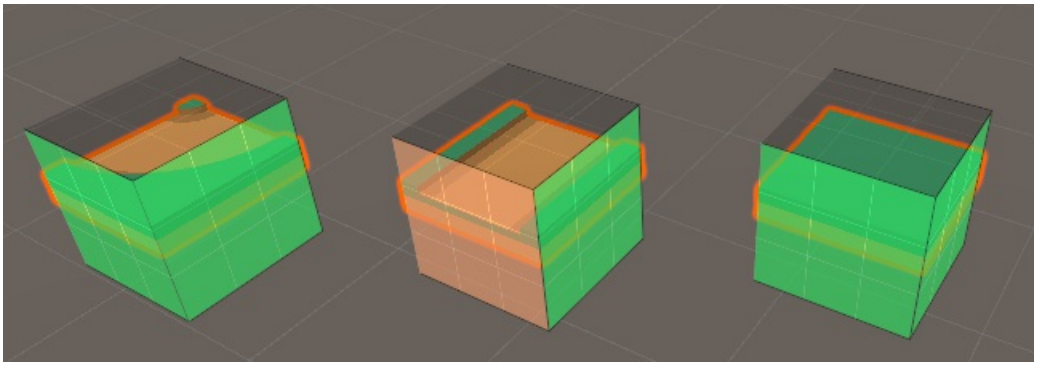
In order to acheive this, we will paint all the grassy faces of each tile green, and all the faces with paths brown. The top and bottom we will leave alone. That will connect grass to grass, paths to paths, and disallow grass connecting to paths.

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

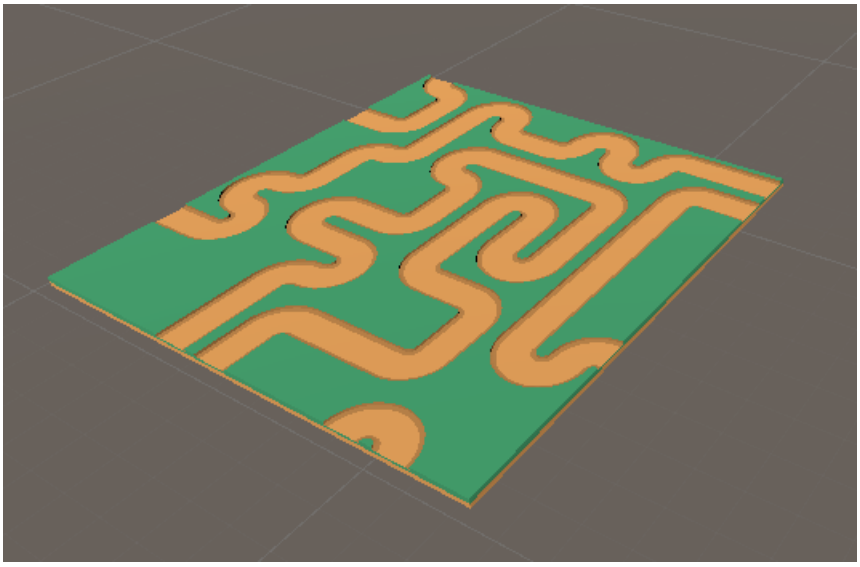
Front view:



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.



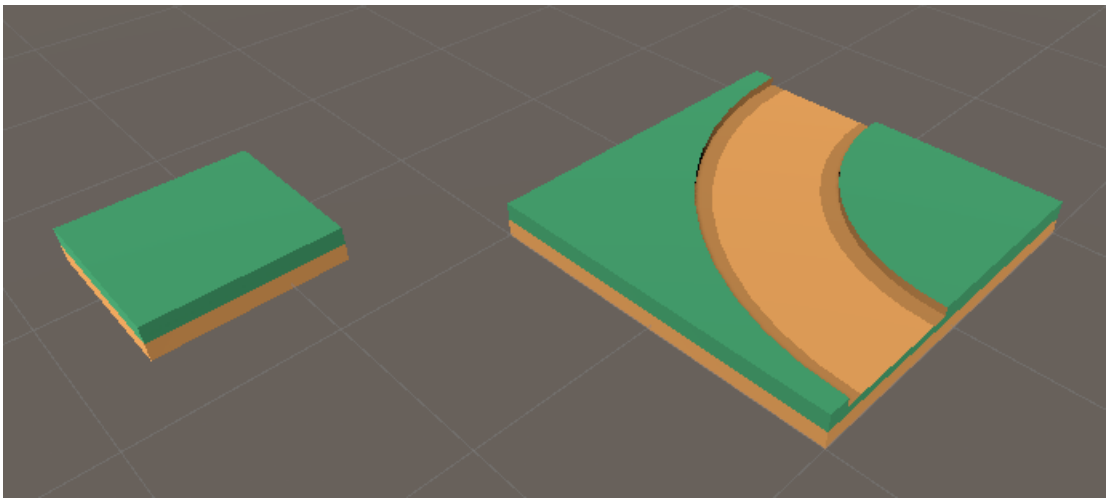
This concludes the tutorial. From here you can experiment with some of the settings in the inspector, try adding more tiles from the tower defense assets, or read the more advanced tutorials.

Big Tiles Tutorial

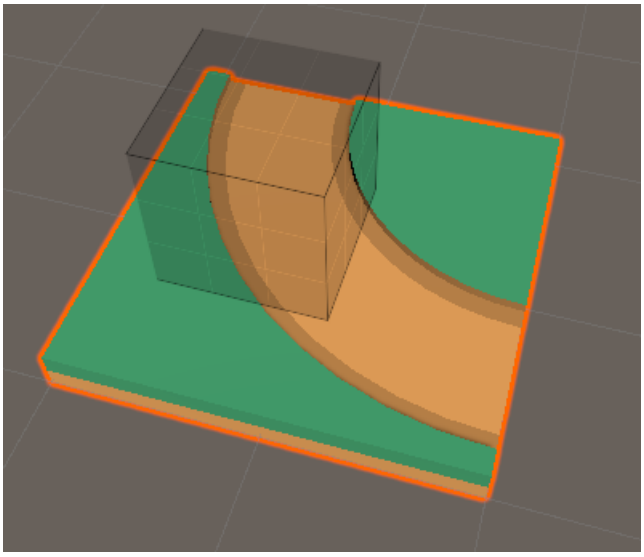
This tutorial continues from the Getting Started tutorial. It is recommended you complete that one before starting this.

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.

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



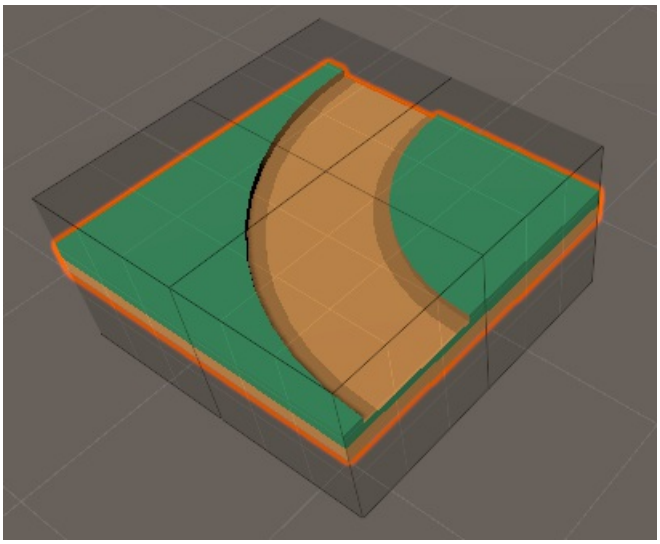
Lets set it up. As before, add the TesseraTile component. Then set the Center to (-0.5, 0, -0.5). This will place the paintable cube in one corner of the tile.



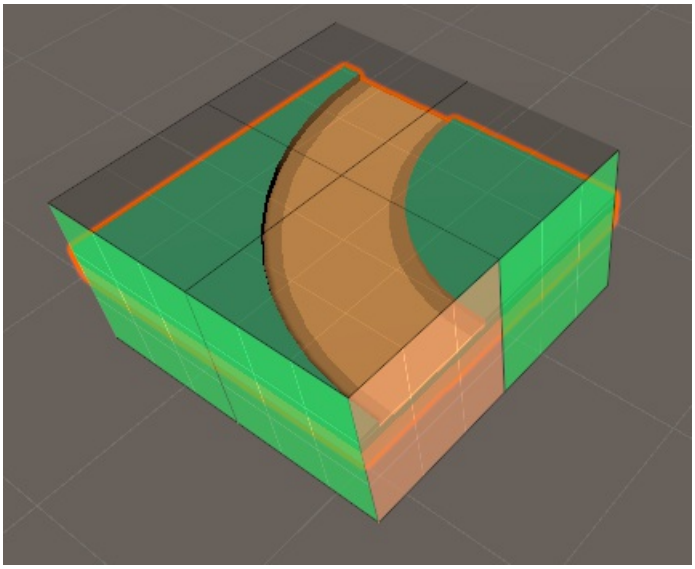
Then select the "Add Cube" tool from the paint menu  . You can now click on faces of the paintable cube to make a tile with multiple cubes in it. If you make a mistake, you can use the "Remove Cube" tool to delete them.

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

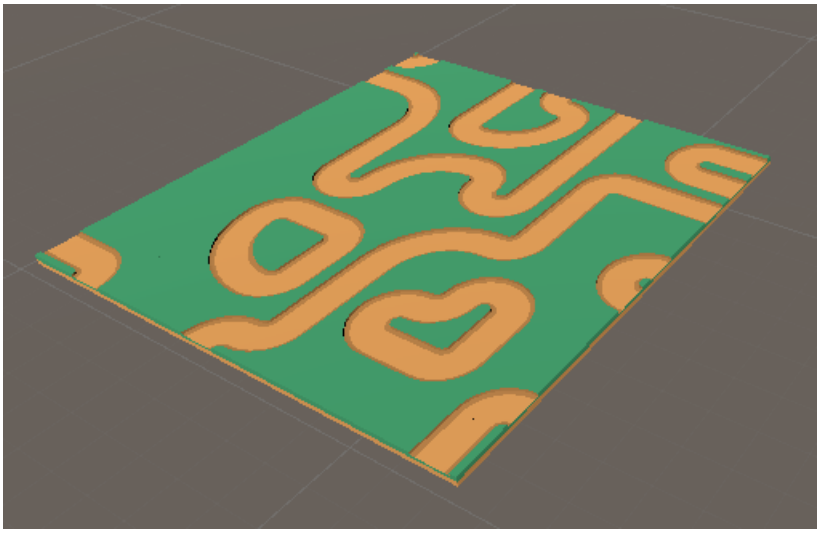
Add 3 extra cubes to the tile to cover it. This tile will now take up as much space as 4 regular tiles.



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.



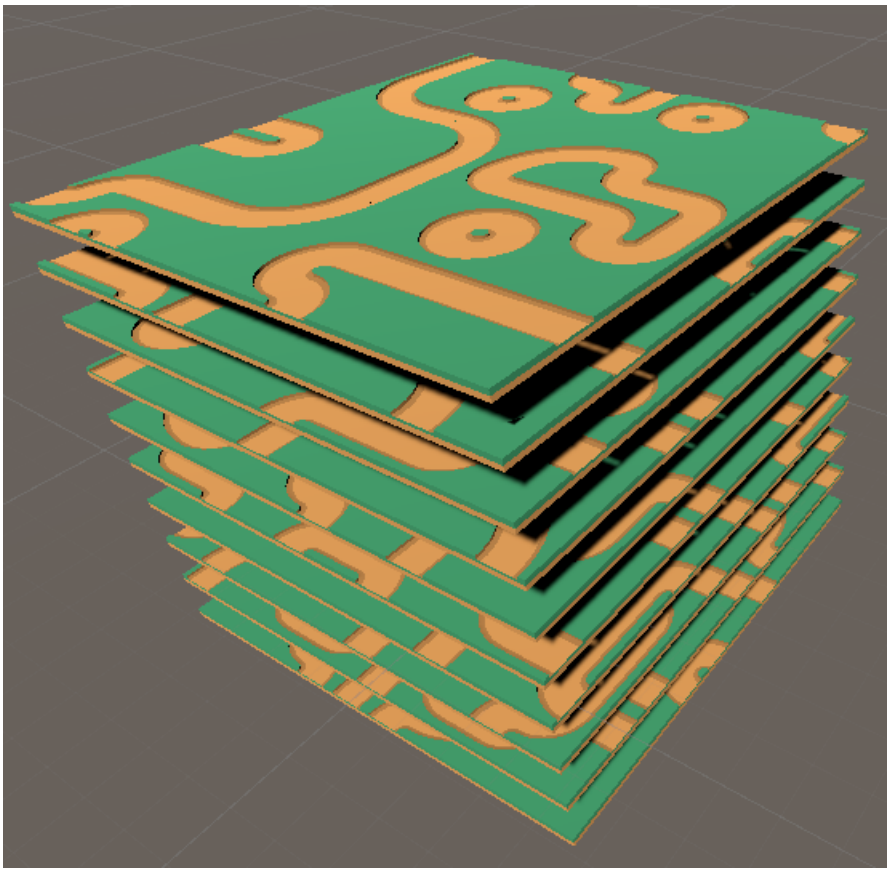
Generating 3d layouts Tutorial

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:

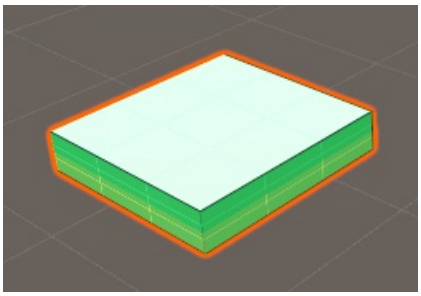
- Each layer is separated from the one above by 1 unit, even though our tiles aren't nearly that high.
- Just like in the first tutorial, Tessera doesn't know how things connect. We need to paint the tiles to indicate what can be put on top of what.

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.

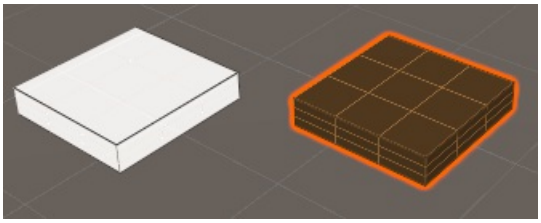
NB: You can use "Show Backfaces" to easily see the far sides of cubes so you don't need rotate all the time to paint everything.



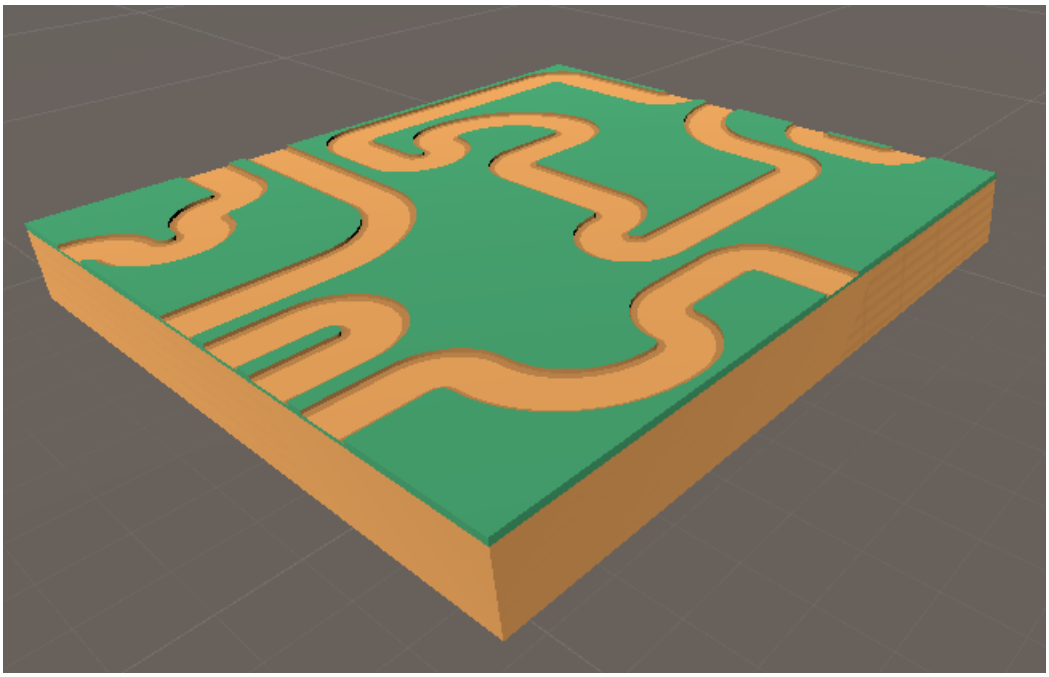
If we run the generation now, it will fail.

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.

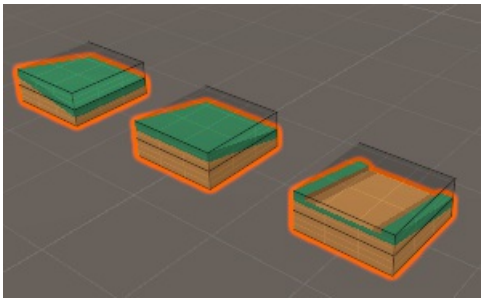


NB: You may notice that instantiating this many tiles takes quite a bit of time. This can be undesirable for a game. If you select the air and dirt tile, and enable "Instantiate Children Only" then they will no longer be instantiated (as they have no children), speeding things up considerably.

Adding slopes

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

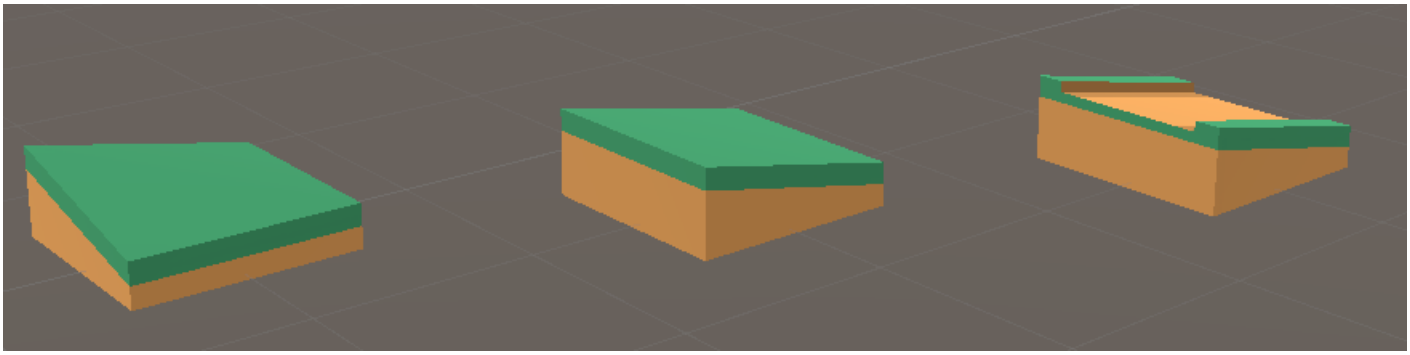
You'll notice that the tile meshes poke out the top of the paint cube. Use Add Cube to stack a second cube on top of the first so that the meshes are completely contained within. See the big tiles tutorial for details.



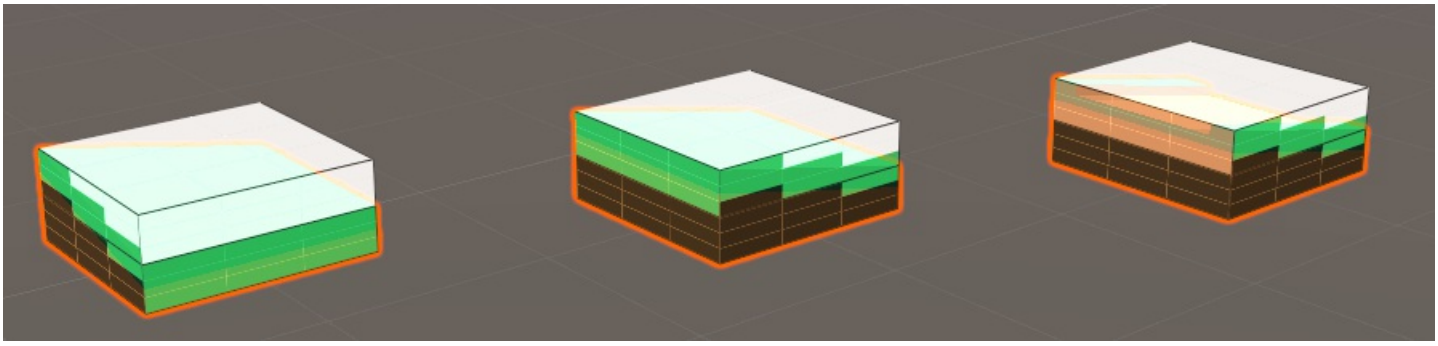
Now, we need to paint the cubes. We need to careful when painting them. We want to ensure that all our existing tiles connect to these new tiles at the correct places. And we need to make sure that the sloped sides can connect only to each other.

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.

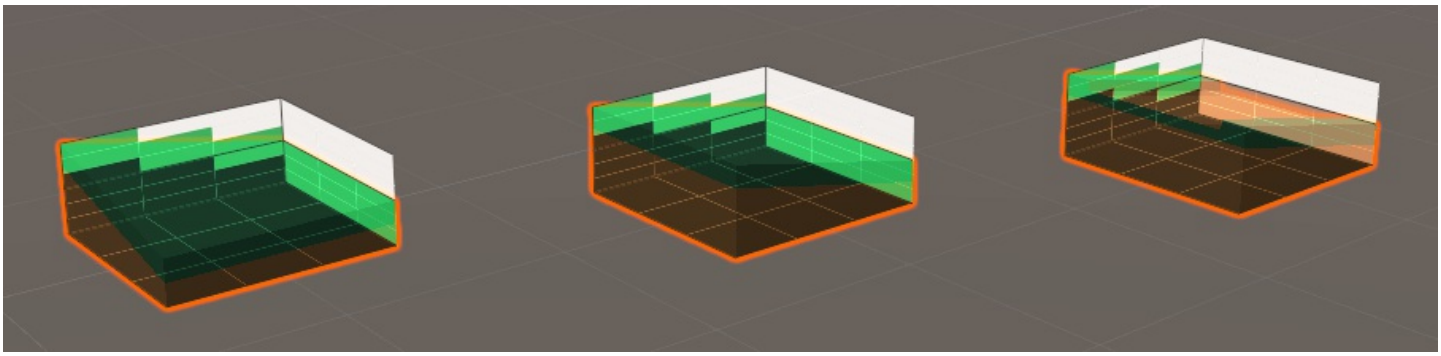
Paint the tiles. Before:



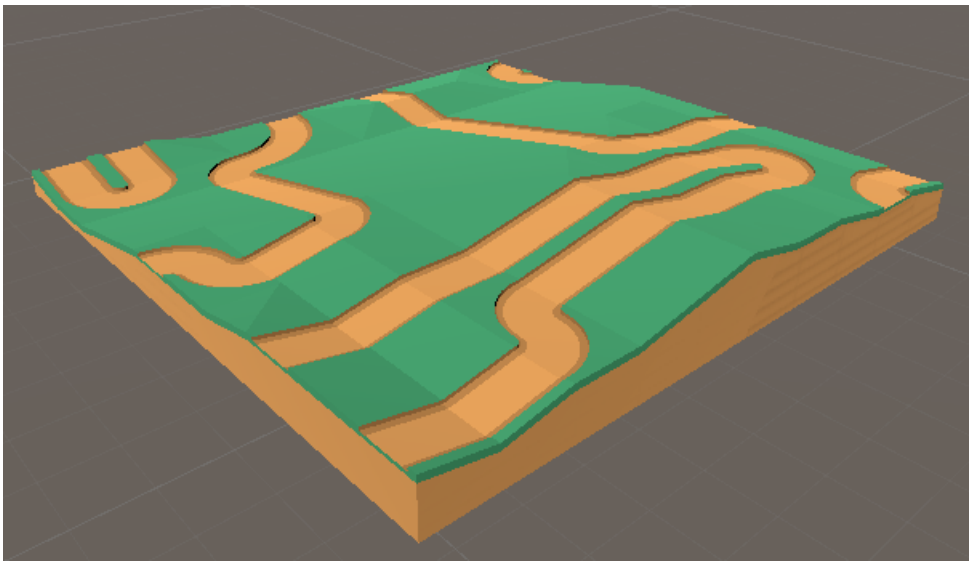
After painting:



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 suprise 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.

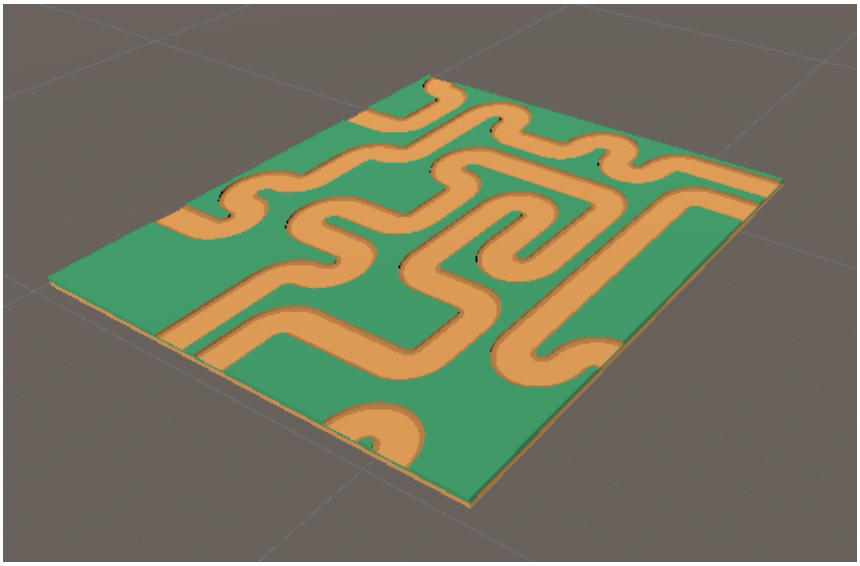
Path Constraints Tutorial

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.

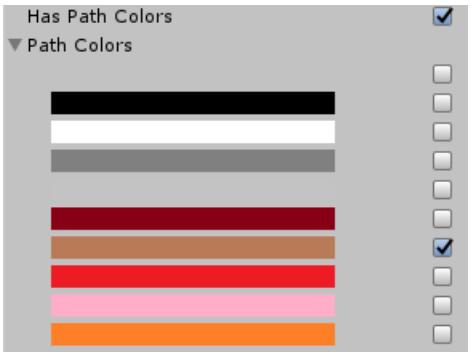
Setup

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.

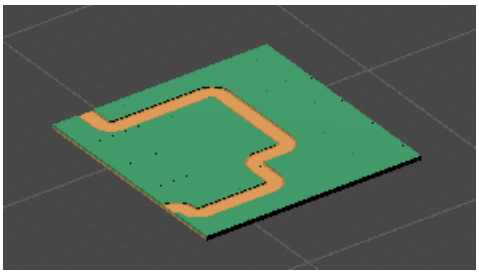
Recall that we colored the tile edge green if that edge was grassy, and brown if that edge had a path. We can give that information to the constraint.

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



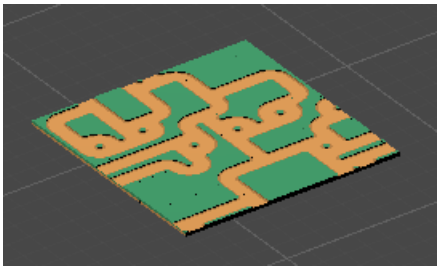
This tells the generator to search for all the sides of tiles that have that color in the center. If two such sides connect together, then it considers there is a path between those two tiles. The constaint then ensures that all path tiles connect to each other.

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



Getting fancier

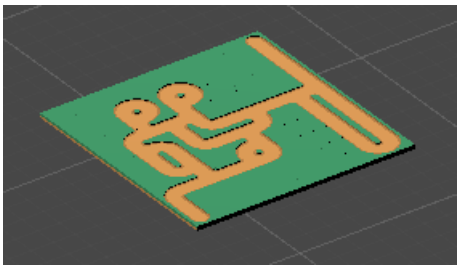
Let's add the `tile_split` tile to the generator, as you can get a lot more interesting paths once you allow junctions. The constraint will still ensure that it's always possible to walk accross the map.

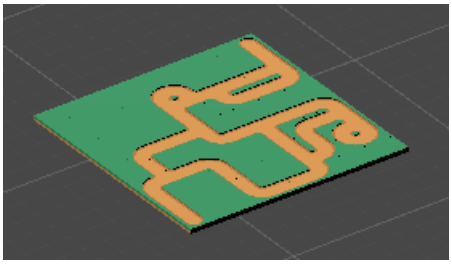


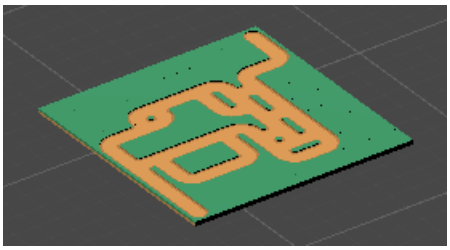
Add a `tile_endRound` to opposite corners of the generated area as a tile constraint. Because these tiles are also painted with the brown path color, they are considered part of the path. And because they are pre-placed, it forces the constraint to make sure both are connected.

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.







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.

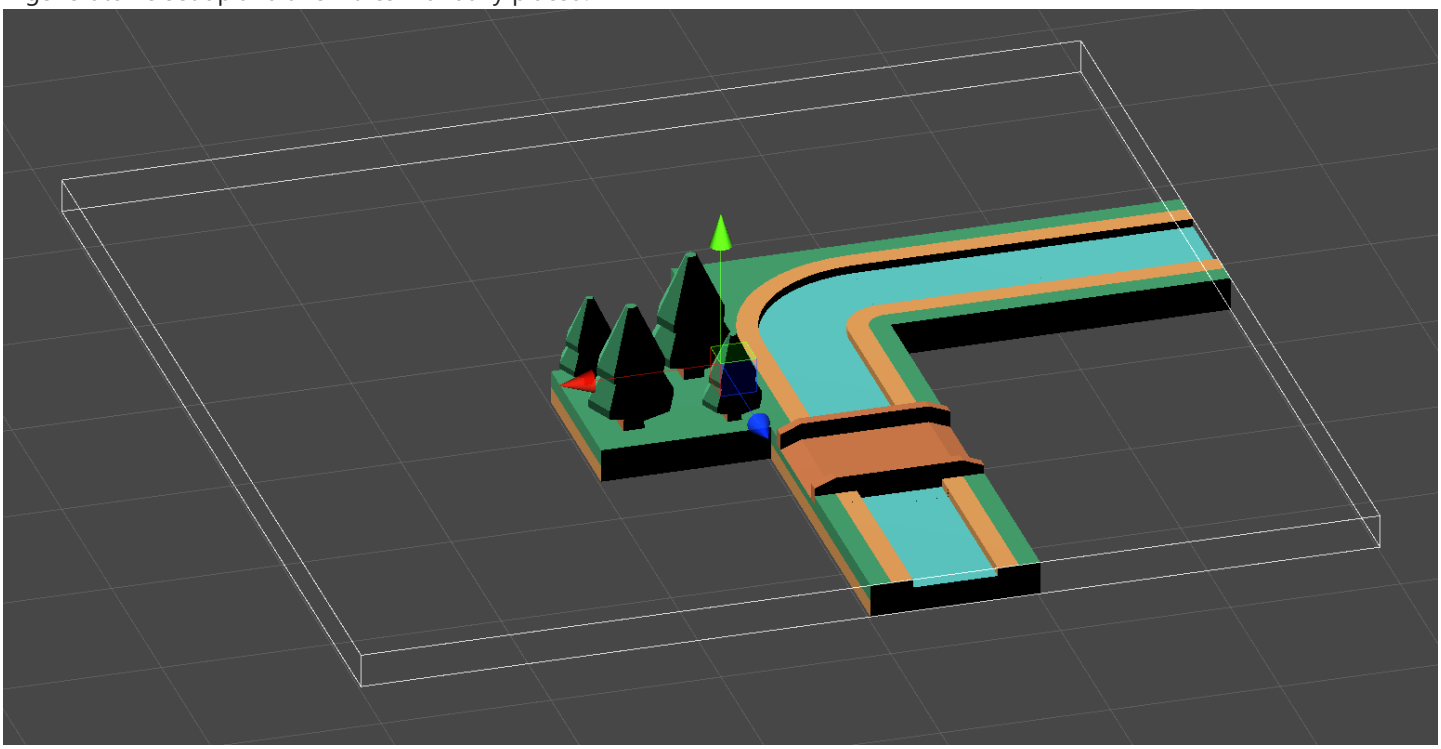
Tile

The generator will automatically recognize any GameObjects in the scene with the TesseraTile component which are inside or adjacent to the generator area. These are **tile constraints**.

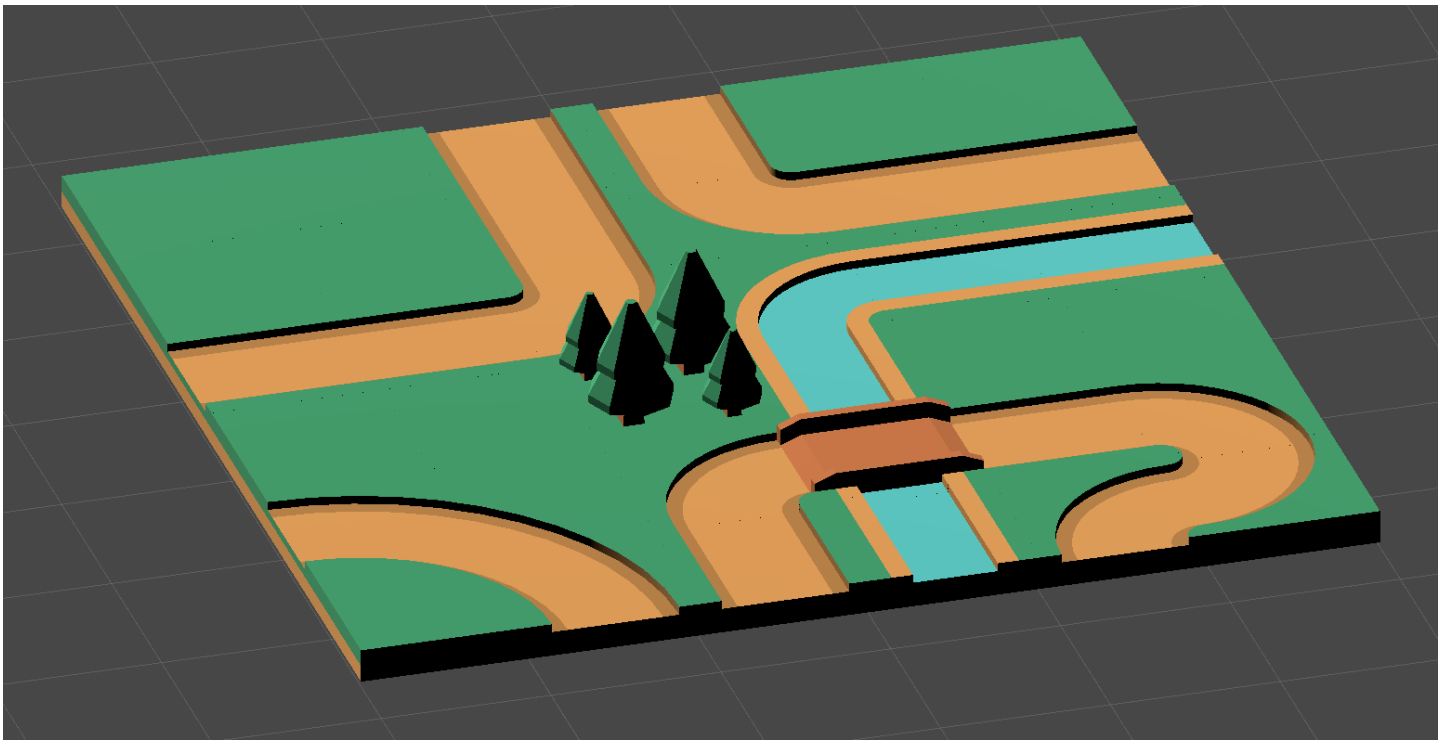
The generator will automatically ensure that any tiles it creates connect to the existing ones correctly, and won't generate tiles that overlap with it.

The tile used for the tile constraint *does not* need to be in the generator's tile list. You can use this for example, to place an elaborate set-piece manually, and then use Tessera to fill in all the nearby tiles.

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.

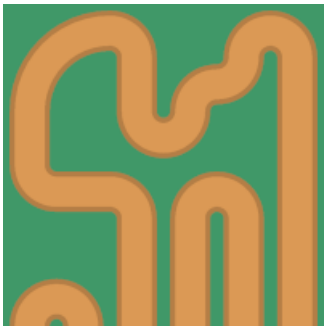


Using the API, you can control this behaviour even closer. First, disable the automatic detection of tiles with by setting searchInitialConstraints to false. Then you can supply your own initial constraints by setting initialConstraints. You need to call GetInitialConstraint to convert from TesseraTile components to 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.



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

Generator Components

Note

Generator constraints are only available in Tessera Pro

There are some components that can be added to the generator to control the global behaviour of generation. These constraints are very powerful, but can use generation to fail more frequently.

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 - ensures that there is a connected path between tiles, where you define which tiles connect to each other.

There is a tutorial on how to use path constraints.

Palettes

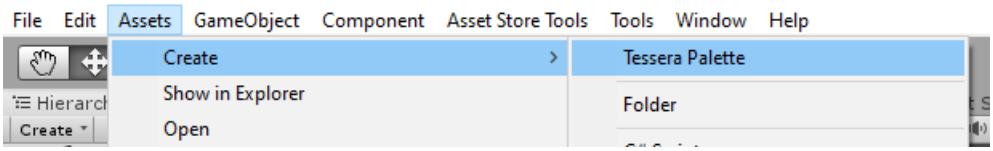
Palettes are an asset type for Tessera that lets you customize which colors you can paint onto tiles, and also controls how colors match.

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

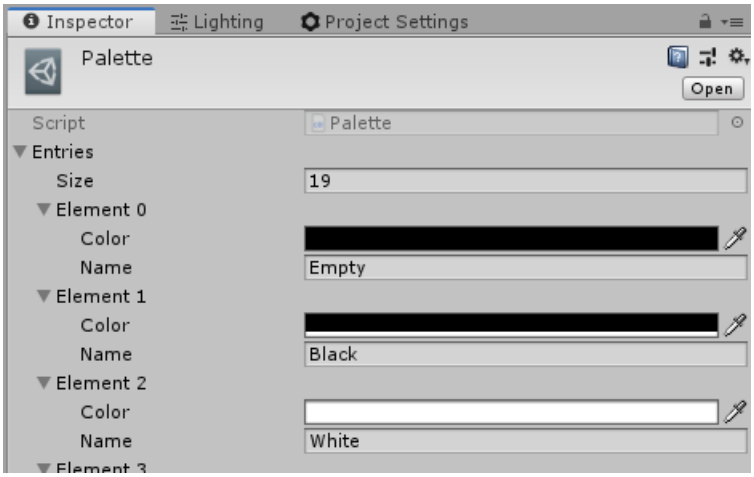
The default palette is configured so that two opposing squares "match" if the colors of those squares are the same, or one is transparent. Recall that two tiles can be placed next to each other if all 9 squares on the face of one tile match with the 9 squares of the opposing face.

Creating a palette

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



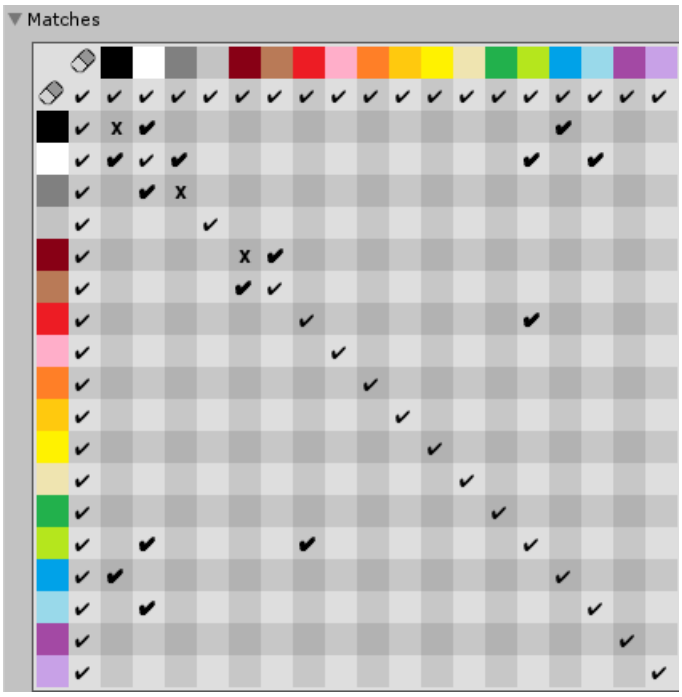
This will create a new asset in your project. You can then select the asset and customize the colors in the inspector.



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.



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.

Animation

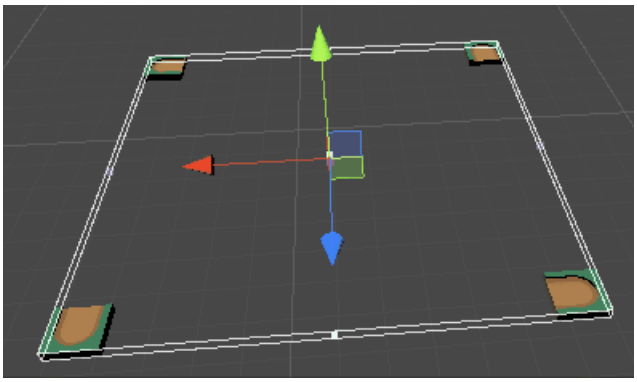
Note

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 that 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. Each step is one of the following:

- Add a tile, and work out all other tiles that are implied by it.
- Backtrack one step, because 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.

Controlling Output

Note

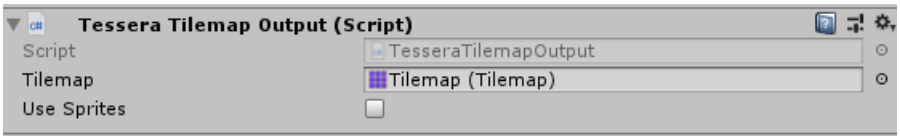
Output control is only available in Tessera Pro

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.

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 that 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, 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.

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 synchoronously, and return details about the generation. It's easier to use, but can cause noticable 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
        generator.Instantiate(instance);
    }
}
```

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)
            {
                generator.Instantiate(instance);
                // Wait for next frame.
                yield return null;
            }
        }
    }
}
```

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.

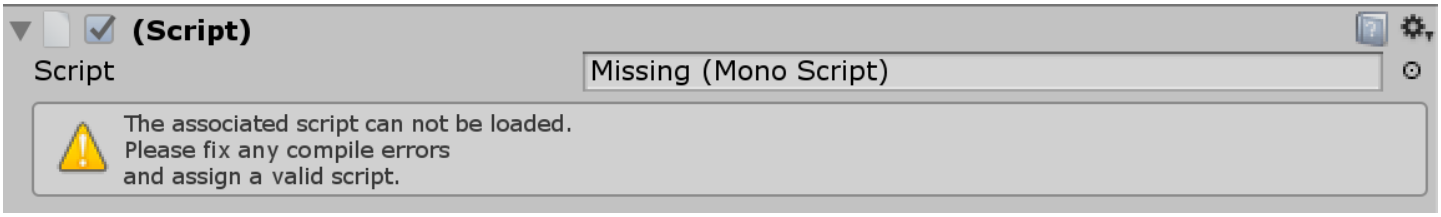
Tessera uses an open source library called DeBroglie for the actual generation process. DeBroglie is initialized in `TesseraGenerator.SetupPropagatorAndRun`. It has many options that are not directly exposed by Tessera, which can be set by changing the code.

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 follow the following instructions.

Backup your project

Always a sensible idea before making any major changes.

Enable Visible Meta Files and Force Text Serialization

This is the default for new projects, but just in case, go to Edit > Project Settings > Editor and select:

- Version Control Mode: Visible Meta Files
- Asset Serialization Mode: Force Text

Replace the Unity references

Close Unity, then open the scene files in any text editor. Then you want to make the following replacements:

Find: `fileID: -65694588, guid: 5ec9deea42ffdf94eae3261973878f98`
Replace: `fileID: 11500000, guid: e3ad2bf01b7a6b7409eb683402aa8669`

Find: `fileID: 2003858105, guid: 5ec9deea42ffdf94eae3261973878f98`
Replace: `fileID: 11500000, guid: 8a3f7e4cbfb5a184b8e397a0175d7112`

Similarly, in any TesseraPalette assets you have:

Find: `fileID: -96226770, guid: 5ec9deea42ffdf94eae3261973878f98`
Replace: `fileID: 11500000, guid: 333e56fb2e5d1ff4bb53c10611586ded`

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

Delete the dummy references

In order to warn users that references have changed, Tessera Pro comes with some "dummy" files that use the old references. These are only used for a warning, so can be safely deleted afterwards. The files are:

- TesseraDummy.dll
- Editor/DummyTesseraTileEditor.cs
- Editor/DummyTesseraGeneratorEditor.cs
- Editor/DummyTesseraPaletteEditor.cs

Downgrading

Follow the same steps for going from Tessera Pro to Tessera, just swapping which strings to find/replace.

2.2.1

- Tessera Palette now serializes correctly
- Fix some Inspector display glitches in Unity 2019.3

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

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

FaceDetails

Records the painted colors for a single face of one cube in a TesseraTile

FaceDirExtensions

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

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

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 (searchInitialConstraints and 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.

TesseraInitialConstraint

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

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

TesseraPalette

TesseraTile

GameObjects with this behaviour record adjacency information for use with a TesseraGenerator.

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

TileEntry

Specifies a tile to be used by TesseraGenerator

Structs

OrientedFace

Records the painted colors and location of single face of one cube in a TesseraTile

Interfaces

ITesseraTileOutput

Enums

FaceDir

Enum of the 6 faces on a cube.

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

secondsPerStep

Declaration

```
public float secondsPerStep
```

Field Value

TYPE	DESCRIPTION
Single	

uncertaintyTile

Declaration

```
public GameObject uncertaintyTile
```

Field Value

TYPE	DESCRIPTION
GameObject	

Properties

IsRunning

Declaration

```
public bool IsRunning { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsStarted

Declaration

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

Property Value

TYPE	DESCRIPTION
Boolean	

Methods

PauseGeneration()

Declaration

```
public void PauseGeneration()
```

ResumeGeneration()

Declaration

```
public void ResumeGeneration()
```

StartGeneration()

Declaration

```
public void StartGeneration()
```

StopGeneration()

Declaration

```
public void StopGeneration()
```

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<TesseraTile> tiles
```

Field Value

TYPE	DESCRIPTION
List<TesseraTile>	

Class FaceDetails

Records the painted colors for a single face of one cube in a TesseraTile

Inheritance

Object

FaceDetails

Implements

IEnumerable<ValueTuple<Vector2Int, Int32>>

IEnumerable

Namespace: Tessera

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class FaceDetails : IEnumerable<(Vector2Int, int)>, IEnumerable
```

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	

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

Item[Vector2Int]

Declaration

```
public int this[Vector2Int p] { get; set; }
```

Parameters

TYPE	NAME	DESCRIPTION
Vector2Int	p	

Property Value

TYPE	DESCRIPTION
Int32	

Methods

GetEnumerator()

Returns an enumerator of length 9 with the position and color index

Declaration

```
public IEnumerator<(Vector2Int, int)> GetEnumerator()
```

Returns

TYPE	DESCRIPTION
IEnumerator<ValueTuple<Vector2Int, Int32>>	

RotateBy(Rotation)

Returns a new FaceDetails with the paint shuffled around. Assumes the rotation is about the normal of the face

Declaration

```
public FaceDetails RotateBy(Rotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
Rotation	r	

Returns

TYPE	DESCRIPTION
FaceDetails	

RotateBy(FaceDir, Rotation)

Returns a new FaceDetails with the paint shuffled around. Assumes the rotation is about the y-axis, and the this face has the given facing.

Declaration

```
public FaceDetails RotateBy(FaceDir detailsFaceDir, Rotation rot)
```

Parameters

TYPE	NAME	DESCRIPTION
FaceDir	detailsFaceDir	
Rotation	rot	

Returns

TYPE	DESCRIPTION
FaceDetails	

ToString()

Declaration

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

Returns

TYPE	DESCRIPTION
String	

Overrides

Object.ToString()

Explicit Interface Implementations

IEnumerable.GetEnumerator()

Declaration

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

Returns

TYPE	DESCRIPTION
IEnumerator	

Implements

System.Collections.Generic.IEnumerable<T>
System.Collections.IEnumerable

Enum FaceDir

Enum of the 6 faces on a cube.

Namespace: Tessera

Assembly: cs.temp.dll.dll

Syntax

```
public enum FaceDir
```

Fields

NAME	DESCRIPTION
Back	
Down	
Forward	
Left	
Right	
Up	

Class FaceDirExtensions

Inheritance

Object

FaceDirExtensions

Namespace: Tessera

Assembly: cs.temp.dll.dll

Syntax

```
public static class FaceDirExtensions
```

Methods

Forward(FaceDir)

Declaration

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

Parameters

TYPE	NAME	DESCRIPTION
FaceDir	faceDir	

Returns

TYPE	DESCRIPTION
Vector3Int	The normal vector for a given face.

Inverted(FaceDir)

Declaration

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

Parameters

TYPE	NAME	DESCRIPTION
FaceDir	faceDir	

Returns

TYPE	DESCRIPTION
FaceDir	Returns the face dir with the opposite normal vector.

Up(FaceDir)

Declaration

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

Parameters

TYPE	NAME	DESCRIPTION
FaceDir	faceDir	

Returns

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

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()

Clear the output

Declaration

```
void ClearTiles()
```

UpdateTiles(IEnumerable<TesseraTileInstance>)

Update a chunk of tiles. If inremental 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(IEnumerable<TesseraTileInstance> tileInstances)
```

Parameters

TYPE	NAME	DESCRIPTION
IEnumerable<TesseraTileInstance>	tileInstances	

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

hasSymmetricTiles

If set, symmetricTilesX and symmetricTilesZ is used to determine symmetric tiles. Otherwise, they are automatically detected.

Declaration

```
public bool hasSymmetricTiles
```

Field Value

TYPE	DESCRIPTION
Boolean	

symmetricTilesX

If hasSymmetricTiles, this set specifies tiles that look the same before and after x-reflection. If hasSymmetricTiles is not set, this list is automatically inferred by inspecting the tile's paint.

Declaration

```
public List<TesseraTile> symmetricTilesX
```

Field Value

TYPE	DESCRIPTION
List<TesseraTile>	

symmetricTilesZ

If hasSymmetricTiles, this set specifies tiles that look the same before and after z-reflection. If hasSymmetricTiles is not set, this list is automatically inferred by inspecting the tile's paint.

Declaration

```
public List<TesseraTile> symmetricTilesZ
```

Field Value

TYPE	DESCRIPTION
List<TesseraTile>	

Methods

ReflectedEquals(FaceDetails, FaceDetails)

Declaration

```
public static bool ReflectedEquals(FaceDetails a, FaceDetails b)
```

Parameters

TYPE	NAME	DESCRIPTION
FaceDetails	a	
FaceDetails	b	

Returns

TYPE	DESCRIPTION
Boolean	

SetSymmetricTiles()

Declaration

```
public void SetSymmetricTiles()
```

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, FaceDir, FaceDetails)

Declaration

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

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
FaceDir	faceDir	
FaceDetails	faceDetails	

Fields

faceDetails

Declaration

```
public FaceDetails faceDetails
```

Field Value

TYPE	DESCRIPTION
FaceDetails	

faceDir

Declaration

```
public FaceDir faceDir
```

Field Value

TYPE	DESCRIPTION
FaceDir	

offset

Declaration

```
public Vector3Int offset
```

Field Value

TYPE	DESCRIPTION
Vector3Int	

Methods

Deconstruct(out Vector3Int, out FaceDir, out FaceDetails)

Declaration

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

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
FaceDir	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

hasPathColors

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

Declaration

```
public bool hasPathColors
```

Field Value

TYPE	DESCRIPTION
Boolean	

hasPathTiles

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

Declaration

```
public bool hasPathTiles
```

Field Value

TYPE	DESCRIPTION
Boolean	

pathColors

If hasPathColors, this set filters tiles that the path can connect through.

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<TesseraTile> pathTiles
```

Field Value

TYPE	DESCRIPTION
List<TesseraTile>	

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>	

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	

tileInstances

The list of tiles to create.

Declaration

```
public IList<TesseraTileInstance> tileInstances { get; set; }
```

Property Value

TYPE	DESCRIPTION
IList<TesseraTileInstance>	

Class TesseraConstraint

Abstract class for all generator constraint components.

Note

This class is available only in Tessera Pro

Inheritance

Object
TesseraConstraint
CountConstraint
MirrorConstraint
PathConstraint

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
CancellationToken	

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) 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>	

Properties

multithreaded

If set, then generation is offloaded to another thread stopping Unity from freezing. Requires you to use StartGenerate in a coroutine.

Declaration

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

Property 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 { get; set; }
```

Property Value

TYPE	DESCRIPTION
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 (searchInitialConstraints and 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

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
Boolean	

initialConstraints

The initial constraints to be used, if searchInitialConstraints is false. This can be filled with objects returned from the GetInitialConstraint methods.

Declaration

```
public List<TesseraInitialConstraint> initialConstraints
```

Field Value

TYPE	DESCRIPTION
List<TesseraInitialConstraint>	

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	

searchInitialConstraints

If true, then active tiles in the scene will be taken as initial constraints. If false, then initialConstraints is used instead.

Declaration

```
public bool searchInitialConstraints
```

Field Value

TYPE	DESCRIPTION
Boolean	

skyBox

If set, this tile is used to define extra initial constraints for the boundary.

Declaration

```
public TesseraTile skyBox
```

Field Value

TYPE	DESCRIPTION
TesseraTile	

tiles

The list of tiles eligable 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	

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	

center

The local position of the center of the area to generate.

Declaration

```
public Vector3 center { 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

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	

GetInitialConstraint(TesseraTile)

Utility function that gets the initial constraint from a given tile. The tile should be aligned with the grid defined by this generator.

Declaration

```
public TesseraInitialConstraint GetInitialConstraint(TesseraTile tile)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTile	tile	The tile to inspect

Returns

TYPE	DESCRIPTION
TesseraInitialConstraint	Initial constraint for use with initialConstraints

GetInitialConstraint(TesseraTile, Matrix4x4)

Utility function that 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(TesseraTile tile, Matrix4x4 localToWorldMatrix)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTile	tile	The tile to inspect
Matrix4x4	localToWorldMatrix	The matrix indicating the position and rotation of the tile

Returns

TYPE	DESCRIPTION
TesseraInitialConstraint	Initial constraint for use with initialConstraints

GetInitialConstraints()

Utility function that represents what searchInitialConstraints does.

Declaration

```
public List<TesseraInitialConstraint> GetInitialConstraints()
```

Returns

TYPE	DESCRIPTION
List<TesseraInitialConstraint>	Initial constraints for use with initialConstraints

Instantiate(TesseraTileInstance)

Declaration

```
public GameObject[] Instantiate(TesseraTileInstance instance)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileInstance	instance	

Returns

TYPE	DESCRIPTION
GameObject[]	

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 iessentially the same as Unity's normal Instantiate method, but it respects instantiateChildrenOnly.

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.

Returns

TYPE	DESCRIPTION
GameObject[]	The game objects created.

StartGenerate(TesseraGenerateOptions)

Asynchronously runs the generation process described in the class docs, for use with StartCoroutine.

Declaration

```
public IEnumerator StartGenerate(TesseraGenerateOptions options = null)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraGenerateOptions	options	

Returns

TYPE	DESCRIPTION
IEnumerator	

Remarks

The default instantiation is still synchronous, so this can still cause frame glitches unless you override onCreate.

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

Namespace: Tessera

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class TesseraInitialConstraint
```

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()

Declaration

```
public void ClearTiles()
```

UpdateTiles(IEnumerable<TesseraTileInstance>)

Declaration

```
public void UpdateTiles(IEnumerable<TesseraTileInstance> tileInstances)
```

Parameters

TYPE	NAME	DESCRIPTION
IEnumerable<TesseraTileInstance>	tileInstances	

Implements

ITesseraTileOutput

Class TesseraPalette

Inheritance

Object

TesseraPalette

Implements

ISerializationCallbackReceiver

Namespace: Tessera

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraPalette : ScriptableObject, ISerializationCallbackReceiver
```

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<ValueTuple<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	

Match(Int32, Int32)

Declaration

```
public bool Match(int a, int b)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	a	
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 TesseraTile

GameObjects with this behaviour record adjacency information for use with a TesseraGenerator.

Inheritance

Object

TesseraTile

Namespace: Tessera

Assembly: cs.temp.dll.dll

Syntax

```
public class TesseraTile : MonoBehaviour
```

Fields

center

Where the center of tile is. For big tils that occupy more than one cell, it's the center of the cell with offset (0, 0, 0).

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, FaceDir) 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

TYPE	DESCRIPTION
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

TYPE	DESCRIPTION
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

TYPE	DESCRIPTION
TesseraPalette	

reflectable

If true, when generating, reflections in the x-axis will be used.

Declaration

```
public bool reflectable
```

Field Value

TYPE	DESCRIPTION
Boolean	

rotatable

If true, when generating, all 4 rotations of the tile will be used.

Declaration

```
public bool rotatable
```

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	

Methods

AddOffset(Vector3Int)

Configures the tile as a "big" tile that occupies several cells. Keeps offsets and faceDetails in sync.

Declaration

```
public void AddOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Get(Vector3Int, FaceDir)

Finds the face details for a cell with a given offeset.

Declaration

```
public FaceDetails Get(Vector3Int offset, FaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
FaceDir	faceDir	

Returns

TYPE	DESCRIPTION
FaceDetails	

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 void RemoveOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

TryGet(Vector3Int, FaceDir, out FaceDetails)

Finds the face details for a cell with a given offeset.

Declaration

```
public bool TryGet(Vector3Int offset, FaceDir faceDir, out FaceDetails details)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
FaceDir	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

Cells

Declaration

```
public Vector3Int[] Cells { get; }
```

Property Value

TYPE	DESCRIPTION
Vector3Int[]	

IntPosition

Declaration

```
[Obsolete("Use Cells")]
public Vector3Int IntPosition { 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	

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 TesseraTile Tile { get; }
```

Property Value

TYPE	DESCRIPTION
TesseraTile	

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()

Declaration

```
public void ClearTiles()
```

UpdateTiles(IEnumerable<TesseraTileInstance>)

Declaration

```
public void UpdateTiles(IEnumerable<TesseraTileInstance> tileInstances)
```

Parameters

TYPE	NAME	DESCRIPTION
IEnumerable<TesseraTileInstance>	tileInstances	

Implements

ITesseraTileOutput

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 TesseraTile tile
```

Field Value

TYPE	DESCRIPTION
TesseraTile	

weight

The weight controls the relative probability of this tile being selected. I.e. tile wiht 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
Single	