Introduction

Thanks for purchasing our package!
TriThis is a triangulation package that allows you to generate meshes of randomly generated 2D level grids and visualize them.
It reads a grid with cell types, typically generated by a procedural level generation algorithm, and creates a mesh representation of the data.

Contents

Included in this package are:

- Scripts for triangulating gridbased data.
- A prefab that manages the triangulation process.
- A prefab needed to create meshes from the triangulation data.
- An example that reads level data from a file and visualizes it with a mesh.
- An example that shows how to use it in combination with ProD, a package designed for generating random grid-based levels.

Quickstart

To use this package, add the CellEdgeContourFinder script to a GameObject in the scene (or use the prefab included in the package).

1) In the inspector, define what characters make up your "solid" types (ie. "x" or "#"). Only those are traced and get converted to meshes. You can also define colors for them. Note that these vertex colors only work if your material / shader uses them.

2) Set the CellEdgeContour finder what kind of prefab gets instantiated when creating a chunk. You can create your own, or you can use the prefab in the package. The prefab in the package already has a nice material with a shader that utilizes the vertex colors.

3) Call the GenerateMap method of the CellEdgeContourFinder by using "CellEdgeContourFinder.Instance.GenerateMap(typeMap);"
The typemap is a grid of strings that defines what type of cell is in which location.

In the example scene, we show how to do all of this. Including creating a TypeMap from a text file (In our case from Resources/level.txt).

More Options

There are several more options that can be set in the CellEdgeContourFinder (You can play around with them in our Sample scene):

[Scale]
scales the whole map by this scale. Normally, 1 gridcell equals 1 unity unit.

[Seperate by type]
When this is turned off, only edges are created between solid and non solid cells, so the different type of solid cells bled together.
Turning this on makes multiple passes through the grid and gives each cell type its own mesh.

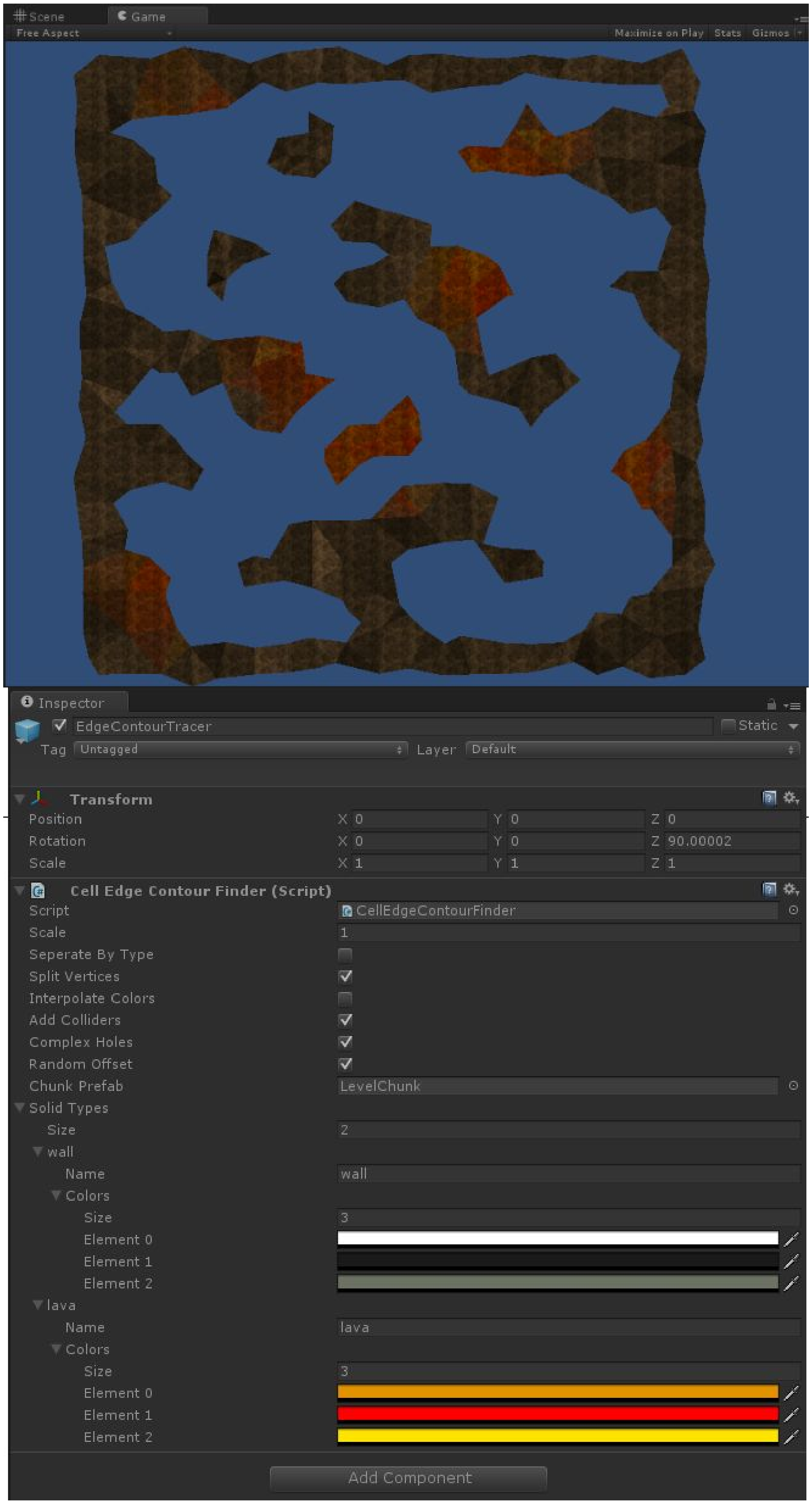
[Split vertices]
When turned on, this gives each triangle its own set of vertices so that colors don't interpolate between triangles, but there are visible edges.

[Interpolate colors]
When turned off, each vertex in a triangle has the same color. When turned on, each vertex can have a different color, and these get interpolated along the edges.

[Add colliders]
When turned on, this creates 2D colliders for the generated meshes.

[Complex holes]
Normally a chunk can have holes. When these holes have chunks in them, and those chunks have again holes in them, these holes wont be generated if "Complex holes" is turned off.
This option makes the generation process more computationally expensive.

[Random offset]
Randomly offsets each vertex that is being created in the contour. This creates a more organic feeling. Turning it off creates a more "blocky" feeling.



EdgeContourTracer Static
Tag Untagged Layer Default

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 90.00002
Scale	X 1	Y 1	Z 1

Cell Edge Contour Finder (Script)

Script	CellEdgeContourFinder
Scale	1
Seperate By Type	☐
Split Vertices	✓
Interpolate Colors	☐
Add Colliders	✓
Complex Holes	✓
Random Offset	✓
Chunk Prefab	LevelChunk
Solid Types	
Size	2
wall	
Name	wall
Colors	
Size	3
Element 0	⬜
Element 1	⬛
Element 2	▨
lava	
Name	lava
Colors	
Size	3
Element 0	🟧
Element 1	🟥
Element 2	🟨

Short Explanation of the inner workings

The most important script of this package is the CellEdgeContourFinder.
The CellEdgeContourFinder is the starting point of the triangulation process which is Initiated by a call to GenerateMap.

These are the steps that the CellEdgeContourFinder goes through:

- It first creates EdgeData; the collection of all the edges on the map. Where an edge is defined as the border between two different type of (solid) cells, and the edge data is .
- The EdgeData is converted into ContourData; a collection of vertices that make up the contour of something we call a chunk. It is created by walking along the edges of EdgeData. Then, the algorithm continues to find new chunks by starting at an not yet visited edge.
- When all the chunks (with contours) are found, a chunk prefab is created. This chunk prefab will create it's own mesh with the chunk data (containing it's contour) it is given. This is done by using Delaunay triangluation used in the Triangle.NET software package, also included in this package.

More on development on this package can be read on http://triangulatorunity.blogspot.nl