**<System Functionality Documentation of Procedural Landmass Generation Plugin within Unity>**

Author: <Zachary Caleb Prole 1464911>

Contents

[Project Summery 2](#_Toc104420884)

[Project Functionality Overview 2](#_Toc104420885)

[C# Script Functionality 2](#_Toc104420886)

[Perlin Noise 2](#_Toc104420887)

[Overview 2](#_Toc104420888)

[Step 1 3](#_Toc104420889)

[Step 2 4](#_Toc104420890)

[Step 3 4](#_Toc104420891)

[Step 4 5](#_Toc104420892)

[Step 5 5](#_Toc104420893)

[Map Generator 6](#_Toc104420894)

[Overview 6](#_Toc104420895)

[Step 1 6](#_Toc104420896)

[Step 2 7](#_Toc104420897)

[Step 3 7](#_Toc104420898)

[Texture Generator 8](#_Toc104420899)

[Overview 8](#_Toc104420900)

[Step 1 8](#_Toc104420901)

[Step 2 8](#_Toc104420902)

[Step 3 8](#_Toc104420903)

[Mesh Generator 9](#_Toc104420904)

[Overview 9](#_Toc104420905)

[Step 1 9](#_Toc104420906)

[Step 2 9](#_Toc104420907)

[Step 3 10](#_Toc104420908)

[Step 4 10](#_Toc104420909)

[Map Display 11](#_Toc104420910)

[Overview 11](#_Toc104420911)

[Step 1 11](#_Toc104420912)

[Step 2 11](#_Toc104420913)

[Final Product Overview 11](#_Toc104420914)

[Bibliography A-Z 12](#_Toc104420915)

# Project Summery

I am producing a plugin for Unity that generates a customisable landscape that can be suited for multiple game ideas for rapid iteration.

# Project Functionality Overview

The overall system functions by the user clicking the generate button which the program then checks to see any pre-saved data is present for each editable variable within the system. If the programs find any of this data, this information is then fed into the equations of this program, if not it will use the default options. After this the program will run through each C# script in order to generate a colligable landmass

# C# Script Functionality

## Perlin Noise

### Overview

Diagram, schematic

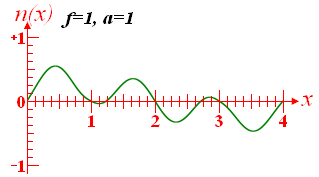
Description automatically generatedThe landmass is generated through a Perlin Noise texture because Unity provides built-in math classes for the formulas. 

Diagram, venn diagram

Description automatically generatedOne problem using this method is that at each integral coordinates of the Perlin Noise are going to be the same colour value. This means that the landmass that will be generated will have similarities with other areas of the landmass. The way I went to get around this is by making the lacunarity, persistence & Octaves (Scale, Frequency & Detail) of the Perlin Noise values editable, so that the user can reduce the number of similarities within their landmass. 

#### Overview Definitions

Perlin noise - A type of coherent noise that is the sum of several coherent-noise functions of ever-increasing frequencies and ever-decreasing amplitudes. (libnoise: Glossary, 2022)



The following graphs show the outputs of a one-dimensional coherent-noise functions n(x).

Lacunarity - A multiplier that determines how quickly the frequency increases for each successive octave in a Perlin-noise function. The frequency of each successive octave is equal to the product of the previous octave's frequency and the lacunarity value. (libnoise: Glossary, 2022)

Persistence - A multiplier that determines how quickly the amplitudes diminish for each successive octave in a Perlin-noise function. The amplitude of each successive octave is equal to the product of the previous octave's amplitude and the persistence value. Increasing the persistence produces "rougher" Perlin noise. (libnoise: Glossary, 2022)

Integral Calculus or Integral - Integrals are the values of the function found by the process of integration. The process of getting f(x) from f'(x) is called integration. Integrals assign numbers to functions in a way that describe displacement and motion problems. (Integral Calculus - Formulas, Methods, Examples | Integrals, 2022) A picture containing text

Description automatically generatedDiagram

Description automatically generated

Octaves - The number of octaves control the amount of detail of Perlin noise. Adding more octaves increases the detail of Perlin noise, with the added drawback of increasing the calculation time. (libnoise: Glossary, 2022)

### Step 1

The first step was to generate the noise map,

This was done by gaining the build in Perlin Noise and looping through each octave and applying the equation (\*2-1). The reason for this is that octaves within a Perlin Nosie will always be a value between 0 and 1. Applying this change the octave values to fall between -1 and 1, an example of this would be (0.1\*2)-1 = -0.8 or (0.7\*2)-1 = 0.4. Now that the entire map has different values between -1 and 1 with gradual differentiation, I can then use this value for the height map on the landmass and because I have the height of all the points on the landmass it also works out the width. This is done by the program knowing that each octave is equal distance from each other, and it know the height, so by using these two measurements, it can work out the width. Text

Description automatically generated

### Step 2

The next step is clamping and normalising all the octaves values as to make the hills and dips realistic looking rather than sharp jagged polygons when it comes to generating the mesh. Text

Description automatically generated

### Step 3

The next step is to randomise noise map locations and set seed numbers.

The reason for this is that if a user finds an area of the map they liked and want to use, they will need a seed number of that exact location and settings. This was simply done by giving each octave a unique seed number between -100,000 and 100,000.

Then I added a GUI that a user could put in a number and that octave location will be generated. Text

Description automatically generated

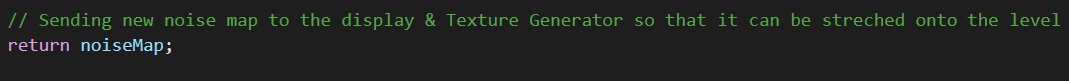
### Step 4

One of the final things to do was to centre the octaves so that when a user uses the seed feature, their seed octave is in the middle and when a user zooms into the map, the zoom to the centre rather than the top right.

Text

Description automatically generated

### Step 5

The final step was to gather all this new information and set it as a new noise map and return it to all the other C# Scrips, so that they can use the new generated values. 

## Map Generator

### Overview

A screenshot of a text message

Description automatically generated with medium confidenceThere are three core tasks the map generator is deigned to do, these are

1.To dictate terrain types of dependent on height values of each octave within the noise map

2. To store all public variables which are Text

Description automatically generated

3. To blend the noise map generated by the Perlin Noise and the colour map, which was generated by the Texture generator

#### Overview Definitions

C# Enumeration (or enum) - Enumeration (or enum) is a value data type in C#. It is mainly used to assign the names or string values to integral constants, that make a program easy to read and maintain. (C# | Enumeration (or enum) - GeeksforGeeks, 2022)

### Step 1

The first is to get the generated noise map and regenerate it alongside a colour map, so when it comes to combing each map, they will be the same. Text

Description automatically generated

### Step 2

The second step is setting up regions.

Text

Description automatically generatedGraphical user interface

Description automatically generatedA map of the world

Description automatically generated with medium confidenceWhat I want from this is the user can declare that certain sections of the map, dependent of height, to be coloured and named differently. What I mean by this is that at the lowest points the user could set a blue colour, making low points of the map blue. This would also work in reverse as the user may want to set high points in the map white for snow. This will work by giving each region a name, height and colour, so the user can give a colour and dictate where it will be coloured.

This will be mechanically functional by looping through each region and then applying the colour to the designated heights.

### Step 3

The third step is combining the noise map , colour map and the mesh map when that is generated.

This was done by making a public enumeration to determin what layer I am drawing in. What I mean by this is that I produced an enumeration called ‘DrawMode’ that kept track of each layer that needed to be generated, which were the noise map , colour map and the mesh map. The reason why I did this was that if I generated them seprently and overlayed them, thy would overwright each other as well as the Perlin Noise map I produced. Doing it this way allows for each map to be blended togther as your only rendering one map. Text

Description automatically generated A picture containing text

Description automatically generated

## Texture Generator

### Overview

A screenshot of a text message

Description automatically generated with medium confidenceThe purpose of the texture generator is to pixelate the colour & noise map then stretch these maps across the world plane.

### Step 1

Pixelization is one of the core ways I got variation for my landmass generation. The reason for this is that going back to the Perlin Noise that was integrated into unity, each integral coordinates of that Perlin Noise is going to be the same colour value. Pixelization completely avoids these problems because it makes all octave values the same, based on the mean octave value per square. How it does this is in each square there is a default 9 octaves, all with different values ranging from -1 to 1. This means that we can interpret these values as gradient colour coordinates. Doing this means that the colour white would equal 1, black would equal 0 and inverted white would equal -1. So, by pixelating the noise map what your doing is adding all 9 octaves values, then dividing them to find the mean value and setting that value to all octaves in that square.

The reason why this works is that each integral coordinates of that Perlin Noise is going to be the same colour value, however all of the alternate coordinates are different colour values, meaning that their octave square mean values will be different than any other, which in turns mean different heights and widths.

Here is an example of a nine-octave square with an overall value of -0.2, which means that it is a slight dip in the map.

### Step 2

Now that the map is pixelated it need to be stretch onto the plane, which is a fairly simple task of getting the current size of the map and setting it into an integer.

A screenshot of a computer

Description automatically generated with medium confidence

### Step 3

The final step is to reapply the colours and saving the values into a colour map.

Text

Description automatically generated

## Mesh Generator

### Overview

A screenshot of a text message

Description automatically generated with medium confidenceThe purpose of the mesh generator is to make a collidable mesh for other objects the user puts in interact with.

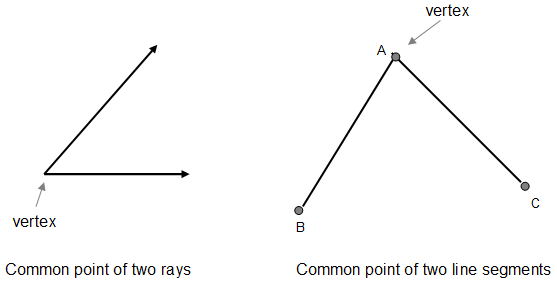
### Step 1

The first step is to generate a plane then get the size of the plane and the vertices. Then when I have gotten this information, put it into the mesh data. Text

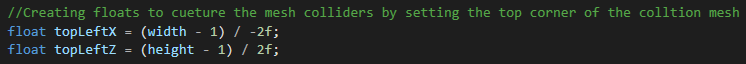
Description automatically generatedText

Description automatically generated Text

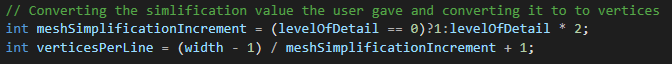
Description automatically generated

Vertices are the corners of any shape and from these vertices I was able to draw in collidable triangles.

### Step 2

Before I started generating triangles, I needed to find the top left vertices because the generation code works by starting at the top left point to make a triangle at, otherwise there would be overlapping meshes, which would crash the whole mesh. 

### Step 3

Next, I needed to sort out the editable value of mesh simplification increment (M.S.I), followed my re-applying the mesh data to the plane. The reason for this is that the M.S.I changes the triangle generation by skipping over vertices, meaning that this is vital to be done before the mesh generation rather than afterwards. 

### Step 4

Now the next part of the mesh generation is drawing in the tringles by looping thought the vertices and vertexes.

Image that I map a map size of 3 by 3, meaning their will be 9 vertices. The reason for this is one in each corner, middle of a side and the centre. Then these vertices are stored into a one-dimensional array that are labelled 0 to 8.

To generate a triangle within Unity, you need to supply an array of 3 integers points to the vertices that make up that triangle. So, one of these tringles will start at 0, as it is the top left, followed by 4 then 3. Then I want to complete the other half of the triangle, which would look like 4 to 0, 0 to 1 then 1 to 4. To sum this up, I am drawing a line from points 0,4,3,4,0,1. This will continue indefinitely to each mini section, so the next piece would look like 1,5,4,5,1, 2.

Before this I need to work out how many elements my arrays are going to contain these include the number of vertices and triangles, so the number of vertices is calculated by…

…and the number of triangles is calculated by.

Map

Description automatically generated

Now that I have all of this information, I can now draw each triangle into the scene, by setting the draw mode to mesh in the map generator.

Text

Description automatically generated

## Map Display

### Overview

Diagram

Description automatically generatedThe purpose of the map display is to mix and render the texture and mesh maps together.

### Step 1

The first step is to get all of the items you will be rendering.

A screenshot of a computer

Description automatically generated with medium confidence

### Step 2

Text

Description automatically generatedThe next step is to set the right draw mode on i.e., mesh and texture and then just render it out into the scene.

## Final Product Overview

A screenshot of a computer

Description automatically generated with medium confidence

In the end I am happy with the results of this project. One of the highlights of this project for me was the fact of customizability of the terrain itself, as it supports 9 different customisable options for landscape shape, and it can be coloured in any way the user wants. Another feature I am happy to achieve is the seed mechanics, which allow designers to generate from any section of the Mathf.Noise patten. The amount of different options a user can generate, excluding colouring is equal to 200000^60 or (1.152921504606846976e+318), which means that a user can a truly unique map every time they use this generator.

# Bibliography A-Z

Cuemath. 2022. *Integral Calculus - Formulas, Methods, Examples | Integrals*. [online] Available at: <https://www.cuemath.com/calculus/integral/> [Accessed 24 May 2022].

Libnoise.sourceforge.net. 2022. *libnoise: Glossary*. [online] Available at: <http://libnoise.sourceforge.net/glossary/#persistence> [Accessed 23 May 2022].

GeeksforGeeks. 2022. *C# | Enumeration (or enum) - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/c-sharp-enumeration-or-enum/> [Accessed 24 May 2022].