## Sprint 3 – Finishing the Perlin Noise method and research in terrain synthesis techniques 21/02/19 – 26/02/19

### Abstract

The goal of this sprint is to finish the Perlin Noise class implementation, research ways to improve the terrain synthesis techniques, implementing some program design like the UML class diagram that describes the structure of the class.

### Research

During the research for improving the Perlin Noise algorithm a journal named Improving Noise by Ken Perlin (Perlin, 2002) explained two deficiencies present on the algorithm; one of those was the fade method, this method is responsible for easing the curve and the other one was the gradient calculation.

#### Fade function

The first thing was improving the fade function by changing the formula, so the author replaced the method present in [Figure 1] to the new method present in [Figure 2].



Figure - Original Fade Function

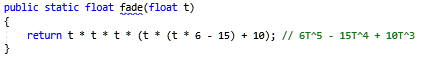


Figure - Fade function improved

Just by simple changing the fade function it is possible to perceive the increased level of detail on the right image, the sand is creating a better pattern and it is possible to perceive that now the water is present in places that before was just sand. This is due to the fact that now the values get even closer to integral coordinates, in other words closer to 1 and 0 [Figure 3].

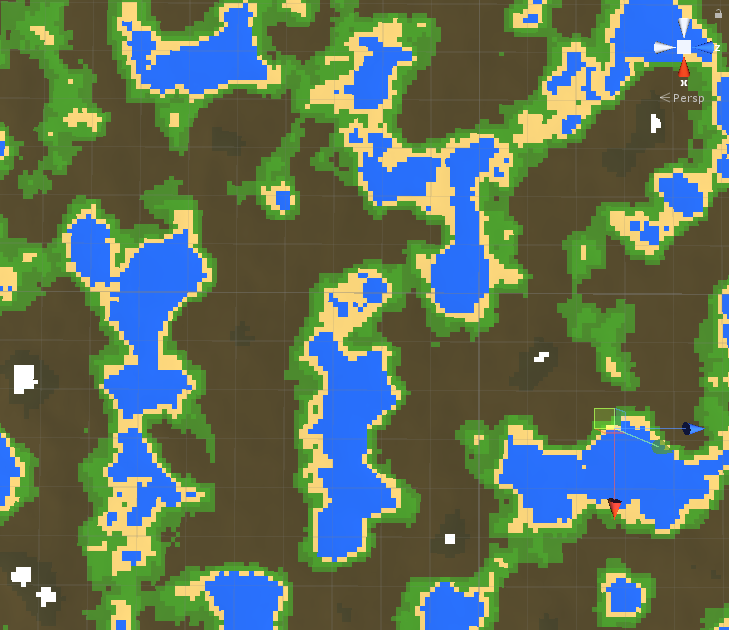
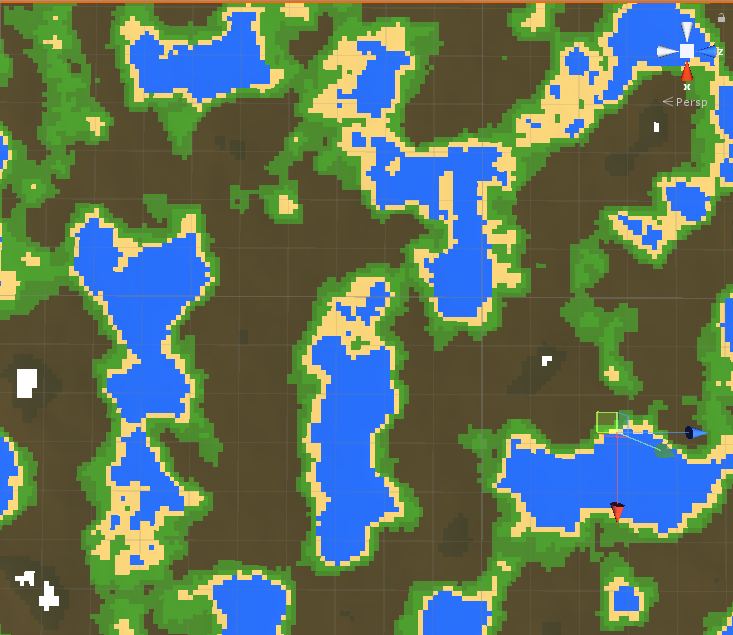


Figure - Original Fade Function left, Improved fade function on the right.

#### Gradient Function

This function is used to calculate the dot product of a randomly selected gradient vector in 8 different locations. The original function [Figure 4] made by Ken Perlin’s uses complicated and confusing bit-flipping code to calculate the dot product and due to that fact, the author was not satisfied, he then decided to begin a research for an easy and more understandable solution.

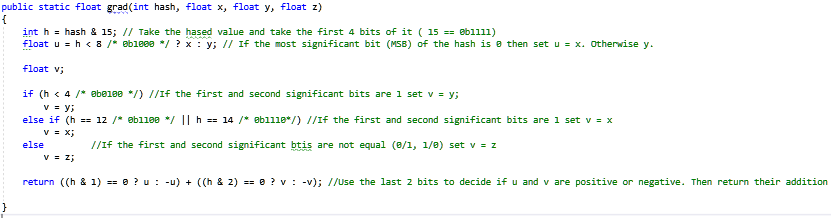


Figure - Gradient Original Function

After a few hours of researching the author found a great solution that seems to have the same effect from the original one (Riven 2010); not only is easier to understand but is twice as fast from the original due to the fact that uses a switch statement and gets all the 16 possible outcomes from the gradient in a simple way instead of using bit manipulation [Figure 5].

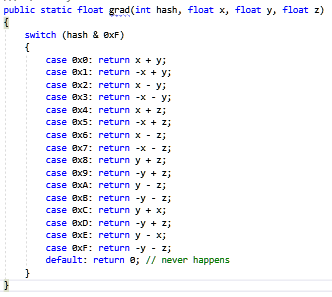


Figure - Simple Gradient Function

To prove that the function gets the same result, in the [Figure 6] is possible to see the two different functions side by side, and that the result is totally identical.

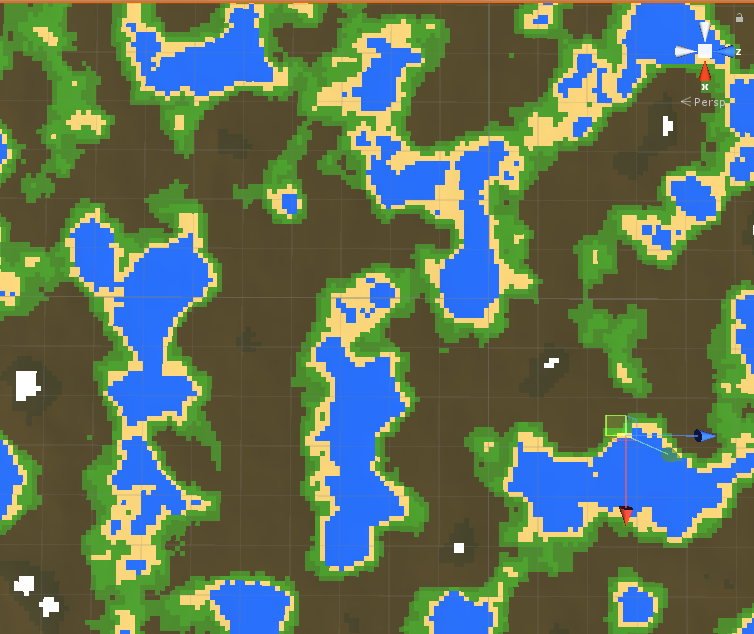


Figure - Original Gradient function | Simple Gradient Function

#### Colour Map

##### Overview

For creating the colour map the author needed to find a way to pass some values using the user interface in unity. So, the first thing was to create a public struct with the three essential attributes; the name, noise height and the colour [Figure 7].

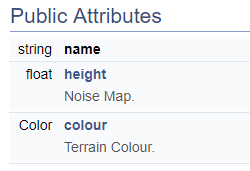


Figure 7 - Struct Attributes

Each of the terrains now can be defined in the unity interface, where the water colour is given for every part of the noise map that has the value between 0.00f - 0.35f while the sand colour is given to the values of between 0.36f - 0.40f. Due to this fact the grass is the last one, the green colour is given the noise values between 0.40 and 0.45f. [Figure 8] [Figure 9].

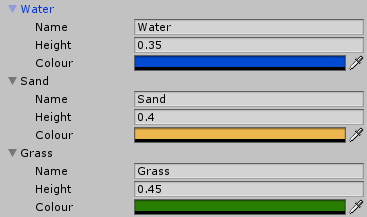


Figure 8 - Unity Array Type of Terrains

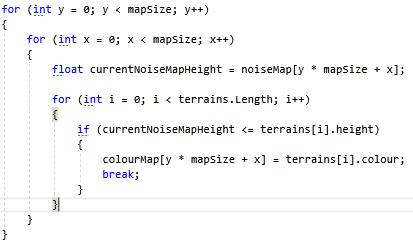


Figure 9 - Adding the colours to the colour map

##### Definition

In [Figure 10] it is possible to see how the ColourMap function is defined.

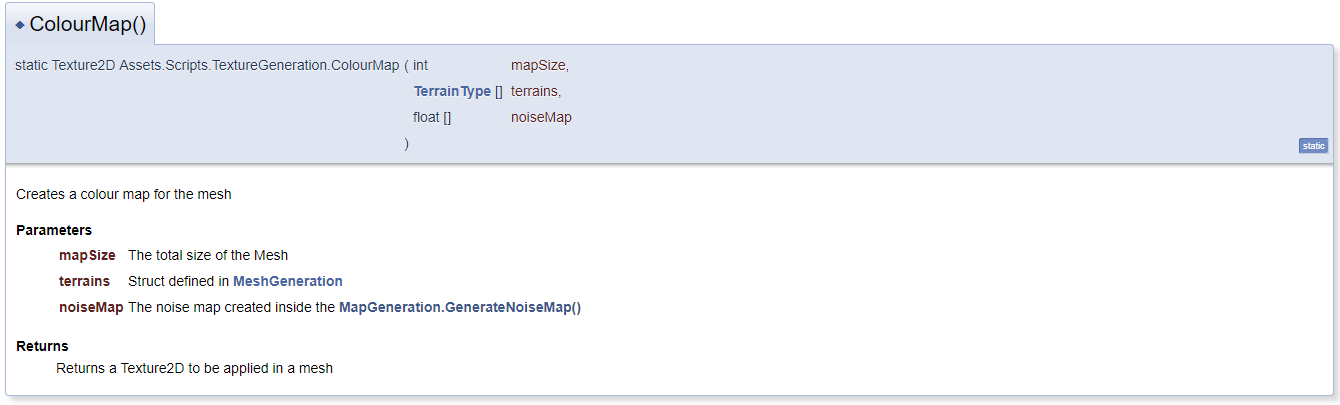


Figure 10 - TextureGeneration.ColourMap()

###### Map Size

The reason why the author passes the mapSize is for creating the total size of the array [Figure 11], and to be able to loop through all the vertices in the map, in this case 65 025 vertices.



Figure 11 – Colour Map array

###### Terrain Type

The TerrainType is essential for knowing how many terrains are defined by the user, going through the array by using the Length variable [Figure 12]. The other use is by getting the height defined by each of the terrains and check if the noise map is smaller or equals to the height, if this is true then apply the selected colour to the colourmap [Figure 13].



Figure 12 - Terrain Length

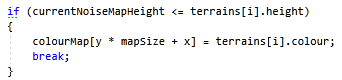


Figure 13 - Noise Map Height

###### Noise Map

The reason why the noiseMap[] is passed in the parameter list is due to the fact that the author needs to know what are the height values for each vertices position. It is possible to retrieve the noise value in that exact vertices location creating two for loops less than mapSize and then current noise map value for each iteration [Figure 14].



Figure 14 - Noise Map current value

###### Return

It returns a Texture2D with the size of mapSize in X and mapSize in the Y where the parameter in the function SetPixels() is the colourmap that was created in the code above.

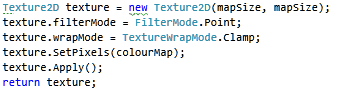


Figure 15 - Return Colour Map

### Sprint Review

The sprint was an overall success, not only the author learned how to improve the noise function and the two deficiencies that were present in the first algorithm made by Ken Perlin, but also learned how to use the noise values and convert them into a Textured2D for later use in any object or mesh.

### WBS

1. Work on procedural Map Generation (50%) (21 hours)
   1. Improve the Perlin Noise method (40%) (8 hours)
   2. Implement a Colour Map using the noise map (60%) (13 hours)
2. Research (50%) (21 hours)
   1. Research about Perlin Noise improvements (90%) (19 hours)
   2. Create a Class Diagram (5%) (1 hours)
   3. Create a Flow Diagram (5%) (1 hours)