50.017 Graphics & Visualization

Group Members: Nguyen Trung Huan (1001704) | Lim Ying Jie (1001753) | Tan Yan You Eiros (1001538)

Procedural World Generation

Abstract
Procedural modelling is a process used to generate models based on a set of rules and parameters. This process has been used in a variety of ways, such as the generation of landscapes and architecture in movies or video games. Our group would like to design a program that performs a 3-dimensional world generation via Procedural Modelling, with the addition of several natural features.

Objectives
The goal of our project is to design a program that can carry out Procedural World Generation using Perlin Noise. The program will be built on the Unity Engine with scripts written in C# code. The final features of this program will be as follows:

- Generation of noise maps using Perlin Noise
    - Height map for displacement mapping of world
    - Heat map for shading of terrain
    - Tree map for placement of trees on world
- World renderer
- Shader for applying textures to world based on heat map

Implementation

Map Generation using Perlin Noise

For our project, both height map and heat maps are generated using Perlin Noise. These maps are used in the generation and shading of our world.

Noise maps are generated with the following variables as input: scale, number of octaves, persistence and lacunarity.

Scale	Scale of the noise map generated
Number of Octaves	Number of distinct noise functions to be combined to give the resulting map
Lacunarity	Scale to increase frequency (for each subsequent noise function)
Persistence	Scale to decrease amplitude (for each subsequent noise function)

Multiple noise functions are used in conjunction to process the perlin noise and give it a more realistic terrain. Each noise function is scaled by an increase in frequency (based on persistence) and a decrease in frequency (based on lacunarity) from the previous function. All functions are then summed together to produce the resulting noise map.

Sample procedure of map generation
Number of octaves: 3
Lacunarity: 2
Persistence: 0.5

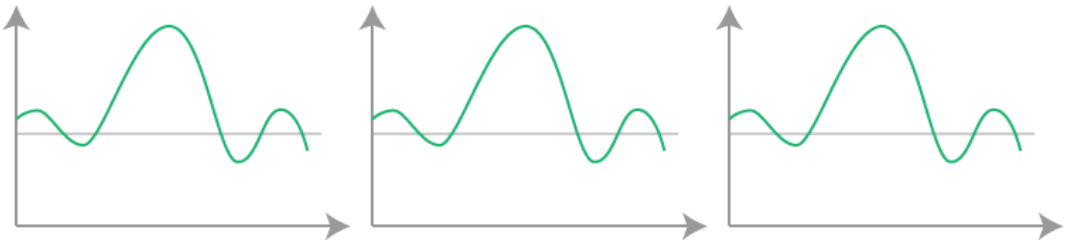

Figure 1: 3 octaves (separate noise function) generated

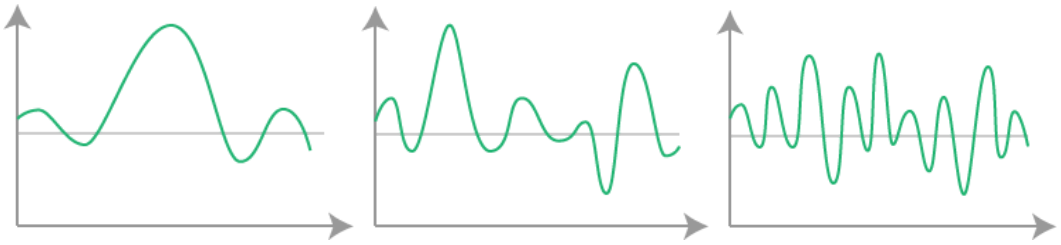

Figure 2: Lacunarity is applied (factor of 2) on each function
Octave 1: Frequency = Lacunarity^0 = 1
Octave 2: Frequency = Lacunarity^1 = 2
Octave 2: Frequency = Lacunarity^2 = 4

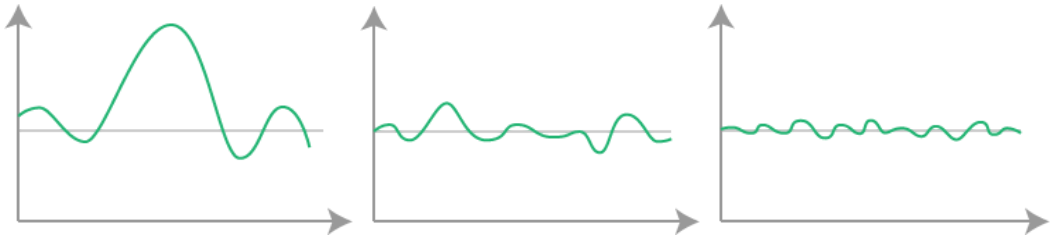

Figure 3: Lacunarity is applied (factor of 0.5) on each function
Octave 1: Amplitude = Persistence^0 = 1
Octave 2: Amplitude = Persistence^1 = 0.5
Octave 3: Amplitude = Persistence^2 = 0.25

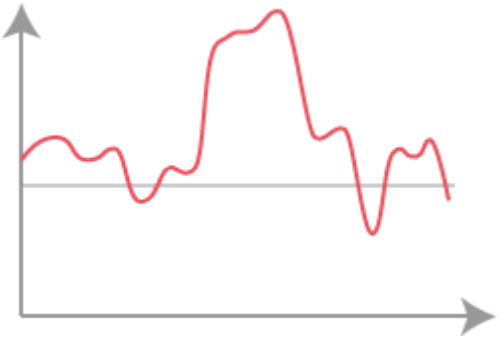

Figure 4: Resulting noise function from summing all 3 octaves

From the above sample, we can thus see how combining several octaves of increasing frequency and decreasing amplitude can help introduce minor features (like boulders, rocks) into a terrain, allowing it to appear more realistic. For a pseudorandom function to be generated, a seed variable is also introduced that offsets the x and y values by a random amount when sampled.

The resulting noise function is then transferred to a HeightMap class in Unity, which stores the values generated in a 2D array along with the minimum and maximum values of the array. A separate TextureGenerator class and MapPreview class is used in order to render and debug the generated noise map, the former generating a texture from a given HeightMap and the latter rendering the resulting texture onto the Unity scene.

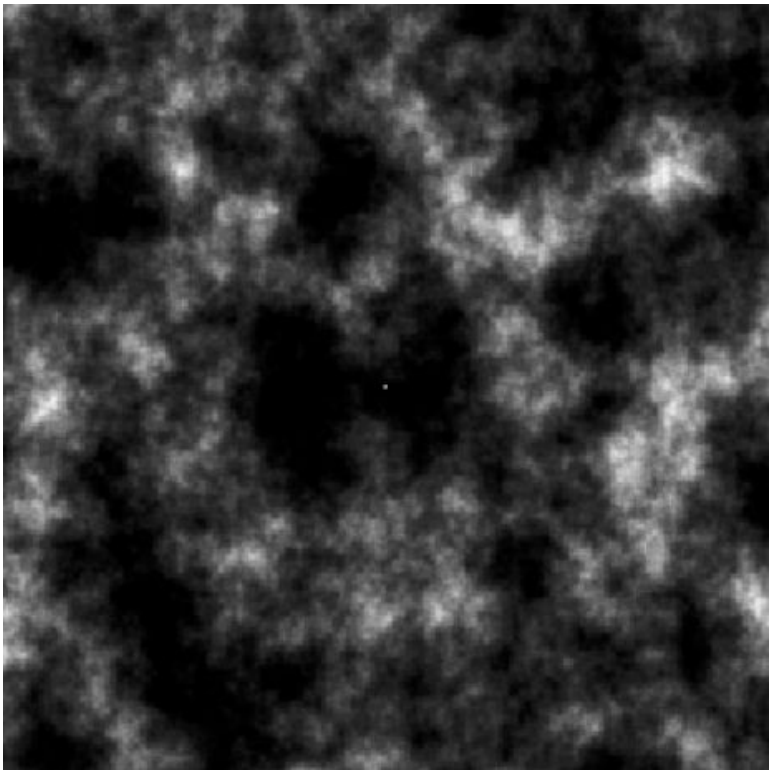

Figure 5: Sample noise map generated by Unity, using scale = 50, octaves = 6, lacunarity = 2 and persistence = 0.65

When rendering the noise map, the values are represented as a linear interpolation from black (0.0f, lowest possible value) to white (1.0f, highest possible value).

Displacement Mapping

To render the world terrain, a 2D array of vertices with triangle meshing is first generated onto the scene. Each vertex is then displaced in the vertical y-direction based on a previously generated height noise map, by reading the value of the noise map at the position of the vertex and multiplying it by a defined height multiplier.

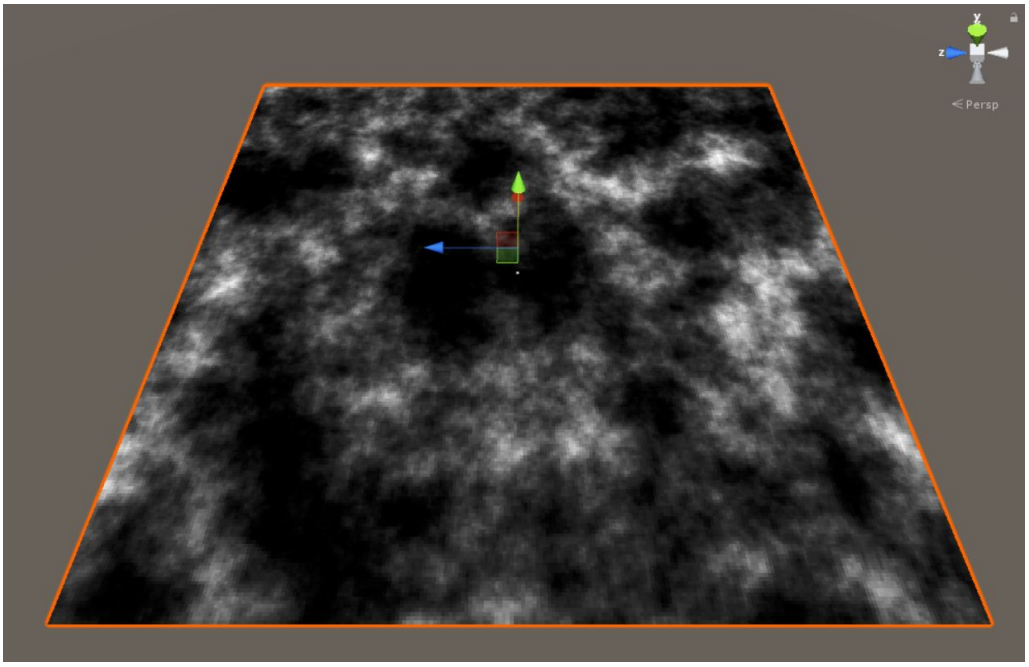

Figure 6: Sample height map of world

Figure 7: Corresponding world generated from map

Heat Mapping

We also used Perlin Noise to generate the heat map for the terrain. In order to obtain a smooth transition between temperatures, we use less octaves to exclude high frequency noise, as well as lower lacunarity and higher persistence. Finally, we increase the scale, so that the hot/cold regions are large enough to be more realistic.

We also accommodate the effect of lower temperatures at higher altitudes by correlating the heat map with the height map, and subtract the temperature according to the height value at each point.

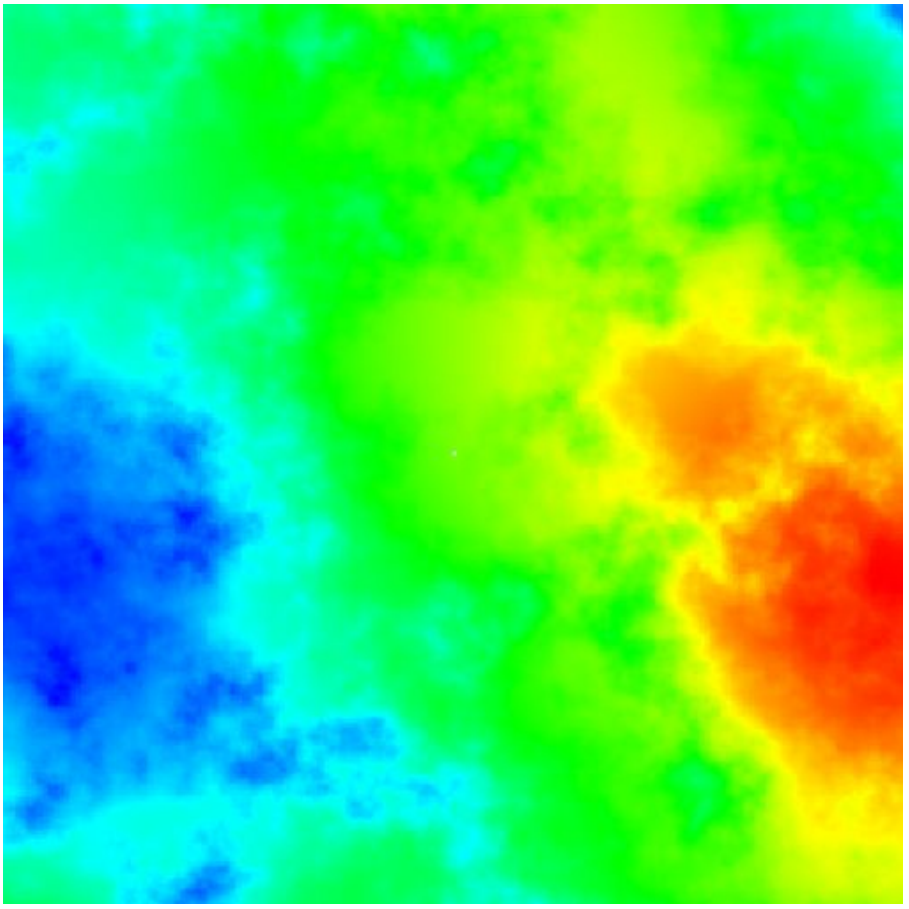

Figure 8: Sample heat map generated by Unity, using scale = 200, octaves = 2, lacunarity = 1 and persistence = 1

When rendering the heat map using TextureGenerator and MapPreview class, the values are represented as a linear interpolation of the hue component of HSV values, starting from blue (0.0f, lowest possible value) to red (1.0f, highest possible value).

Planar Texture Coordinate Mapping

Planar coordinate mapping is used in order to shade the terrain of the world generated. To prevent any noticeable warping of texture when projecting the material onto the mesh at any direction, we instead choose to project the material onto the mesh in all 3 x, y and z-directions. The resulting colour vector is then averaged in order to reduce the increase in brightness and colour of the shaded terrain after summing the texture 3 times, resulting in a blended texture of the terrain.

Terrain Shading

The terrain of the world is shaded based on a function that takes in two noise maps as inputs: the height map and the heat map. This is based off the classification of biomes by Whittaker, who identified the formation of different biomes based on annual precipitation and average temperature of a given area.

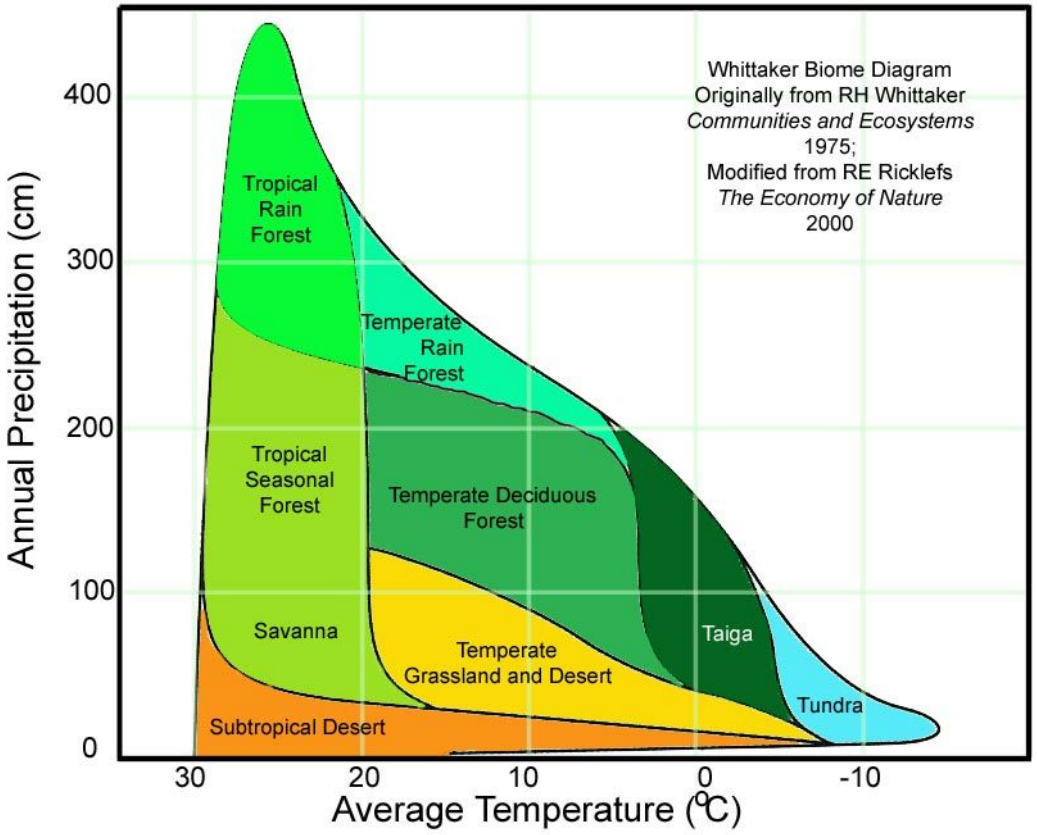

Figure 9: Whittaker Diagram

To simplify the Whittaker Diagram for our use, we have narrowed down the types of terrain we wish to use into 5 types: snow, rock, grass, sand and water. Furthermore, we remove the use of precipitation and instead replace it with height as a factor in the terrain type, with a lower altitude being a factor for water bodies, while higher altitude is more associated with snowy terrain. Above the water bodies and below the snowy terrain, we have sandy and rocky terrains respectively. For areas of the map where the altitude in neither high nor low, we have the terrain to be either sandy, grassy or snowy based on the temperature assigned to the area by the heat map.

Figure 10: Simplified Whittaker diagram

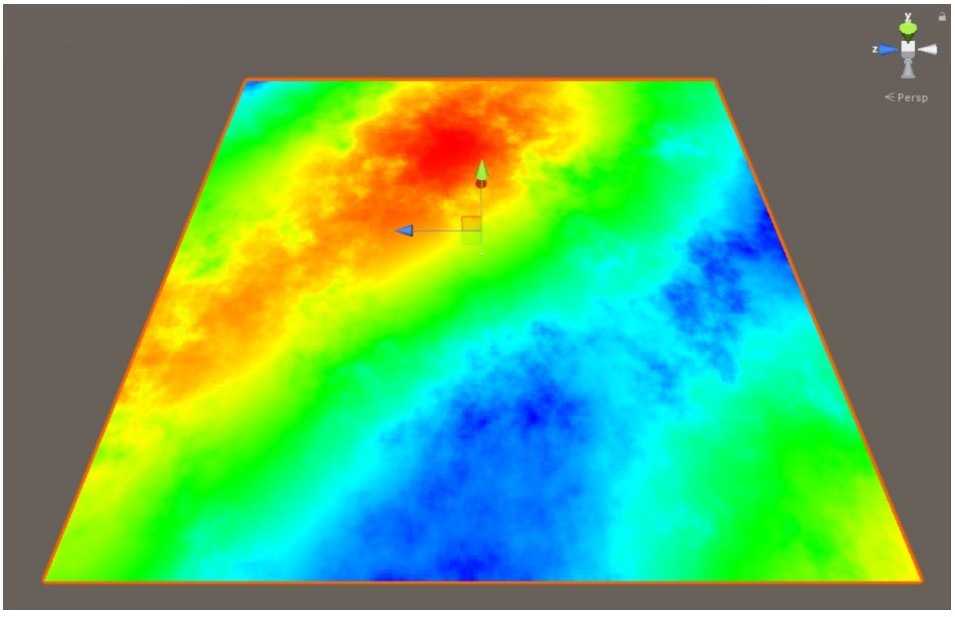

Figure 11: Sample heat map

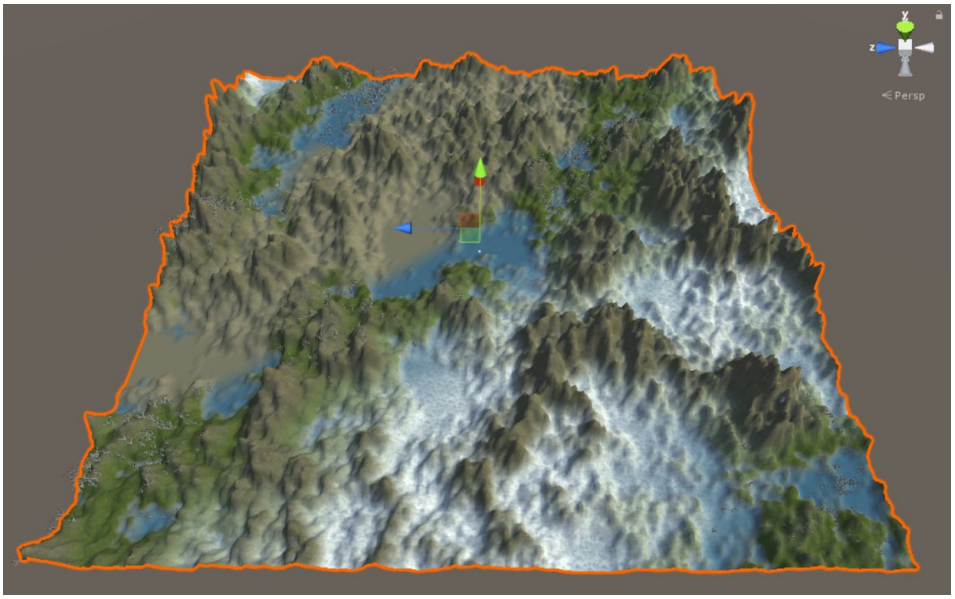

Figure 12: World map with terrain shaded based on above heat map (Figure 11)

As seen from above, we can see how the world (Figure 12) is affected by its corresponding heat map (Figure 11). In the regions of the map with middling altitudes, cold areas (painted blue on the heat map) are shaded with the snow texture, while hot areas (painted red on the heat map) are shaded with the sand texture. Areas which are neither cold nor hot (painted green on the heat map) are shaded with the grass texture.

Tree Mapping and Placement

In determining the placement of trees on the world generated, considerations are made to its altitude on the world map as well as the temperature of its surroundings. The trees can neither be in areas too high (mountain peaks) or too low (water bodies), nor can it be in regions too cold (snow) or too hot (sand).

To generate the tree map, both height and heat maps are passed into the TreeMapGenerator class. For each vertex, we check whether both the height and heat values fall within a certain range. If these criteria are fulfilled, then a tree has a chance to spawn at that location. The probability is then calculated from the normalized heightValue and heatValue according to the formula below:

$$heightValue,\ heatValue \in [0, 1]$$

$$pHeight = \frac{cos(\ 2\pi\ (\ heightValue - 0.5\ )\ ) + 1}{2}$$

$$pHeat = \frac{cos(\ 2\pi\ (\ heatValue - 0.5\ )\ ) + 1}{2}$$

$$p = pHeight \times pHeat \times 2$$

We also scale the size of the tree being spawned according to the probability, so that areas with a high probability of spawning a tree can have bigger trees, thus creating the appearance of a forest.

Before rendering the tree map, values are assigned to the 2D array based on the altitude at which the trees are to be positioned. The trees are then represented as white points against a black plane on the tree map. These points are a linear interpolation from the lowest tree (dark grey) to the highest tree (white). Any region where there are no trees is left as black.

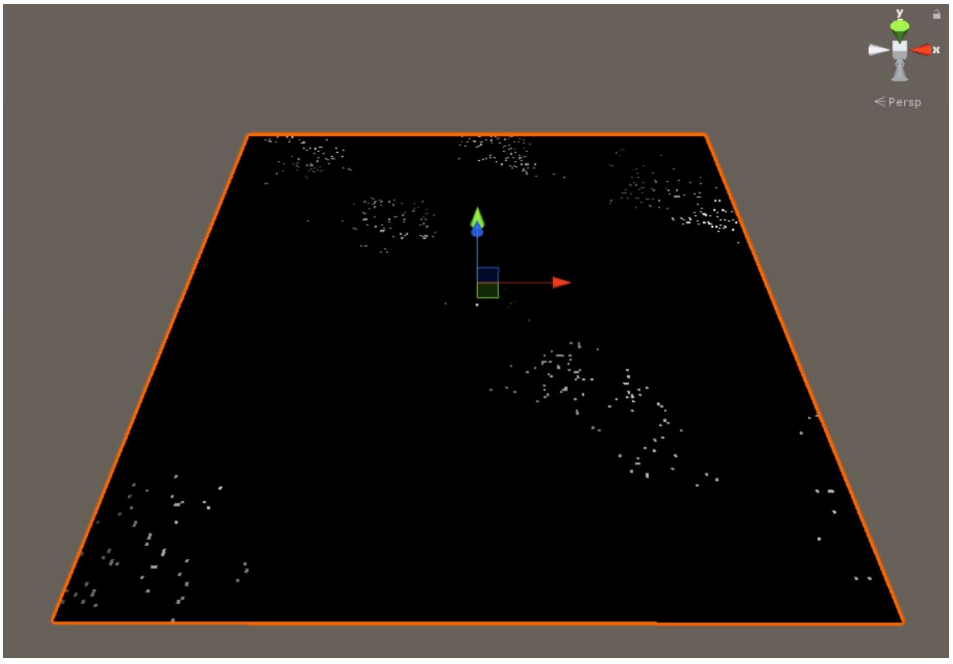

Figure 13: Sample tree map.

Figure 14: Trees rendered on a generated world

Results

By the end of the project, we have developed a Procedural World Generator that is capable of rendering a 3-dimensional world using Perlin Noise. Users are able to change the value of the scale, number of octaves, lacunarity and persistence of the height, heat and tree maps generated in the settings (under Assets > Terrain Assets), which then affects the terrain rendered in the scene.

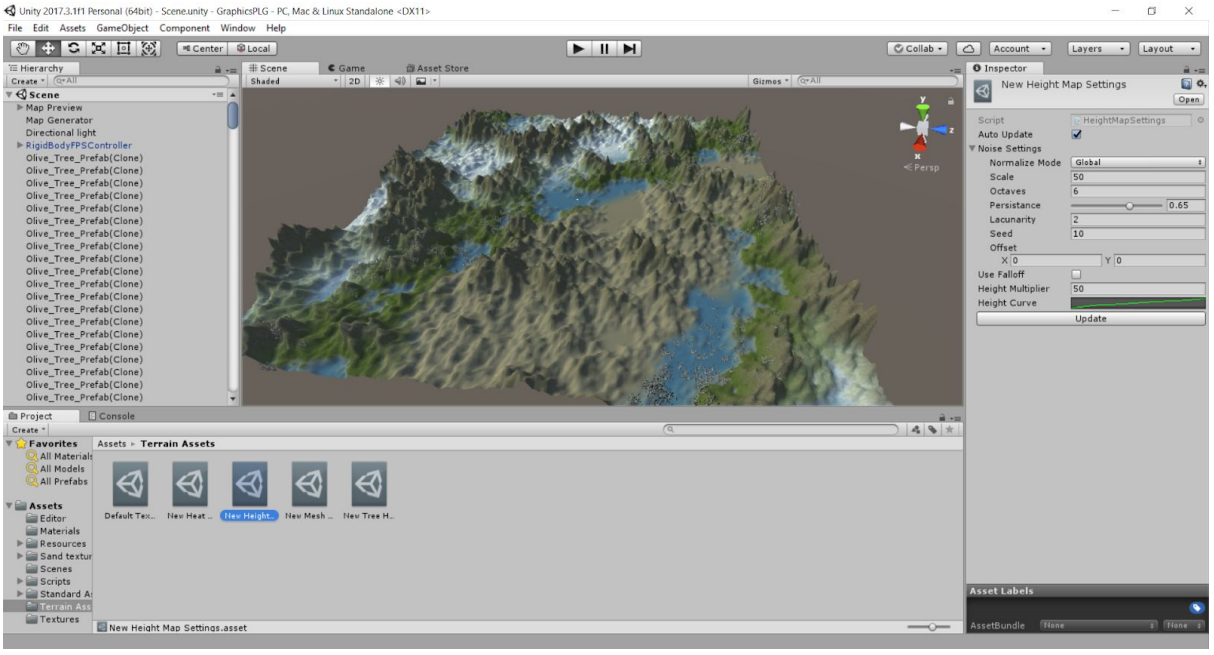

Figure 15: Unity interface of our project (Settings)

Users are also able to switch between viewing the height, heat and tree maps generated as well as the resulting mesh, by clicking on the Map Preview object in the scene and choosing the desired Draw Mode in the dropdown.

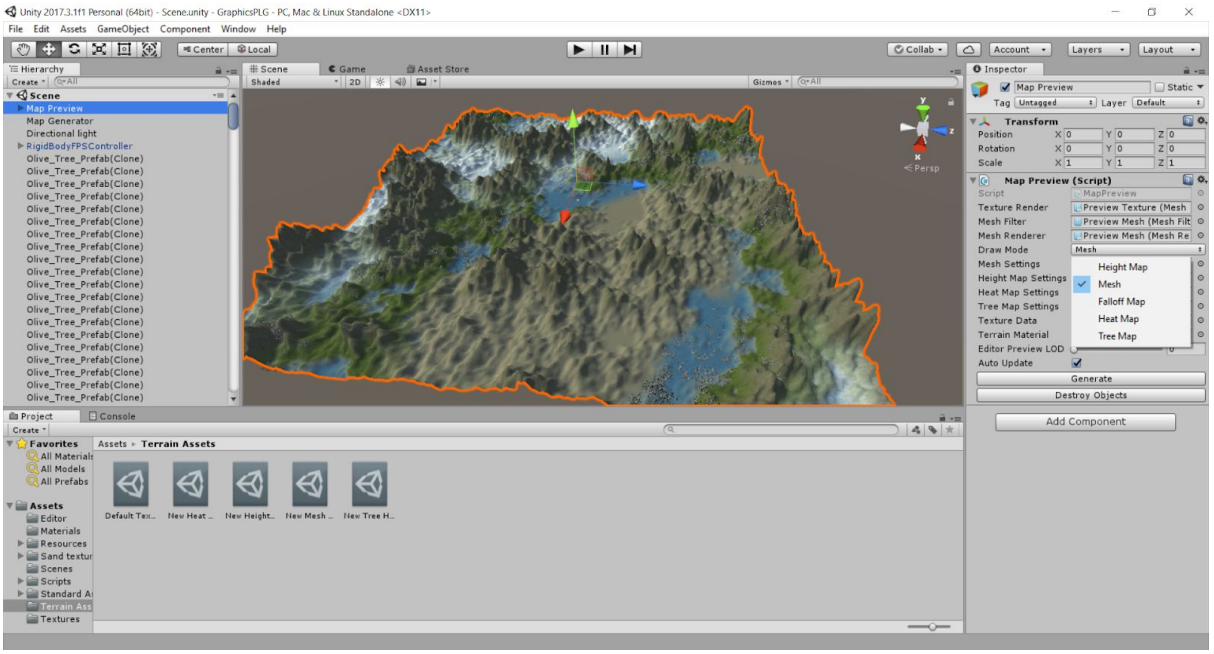

Figure 16: Unity Interface (Map Preview object)

Figure 17: Map Settings

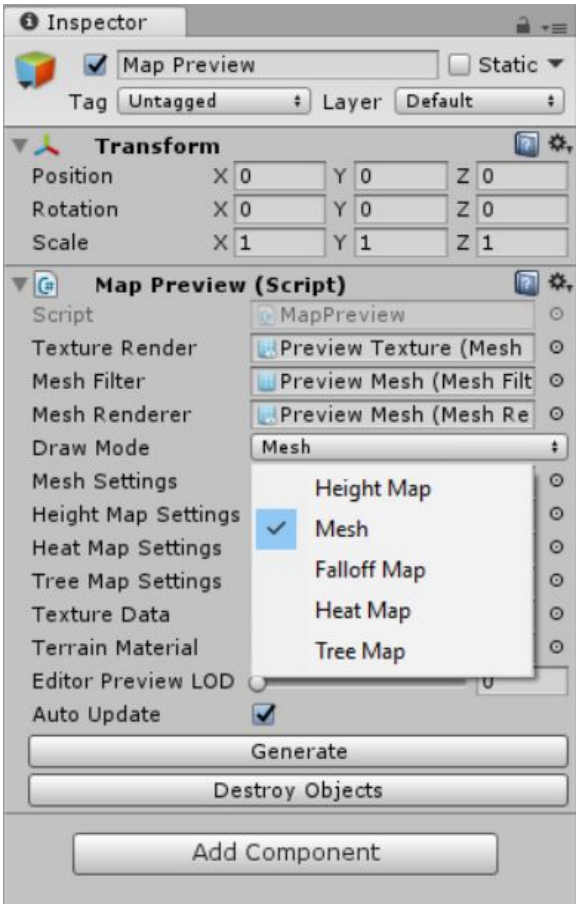

Figure 18: Dropdown (Draw Mode setting of Map Preview)

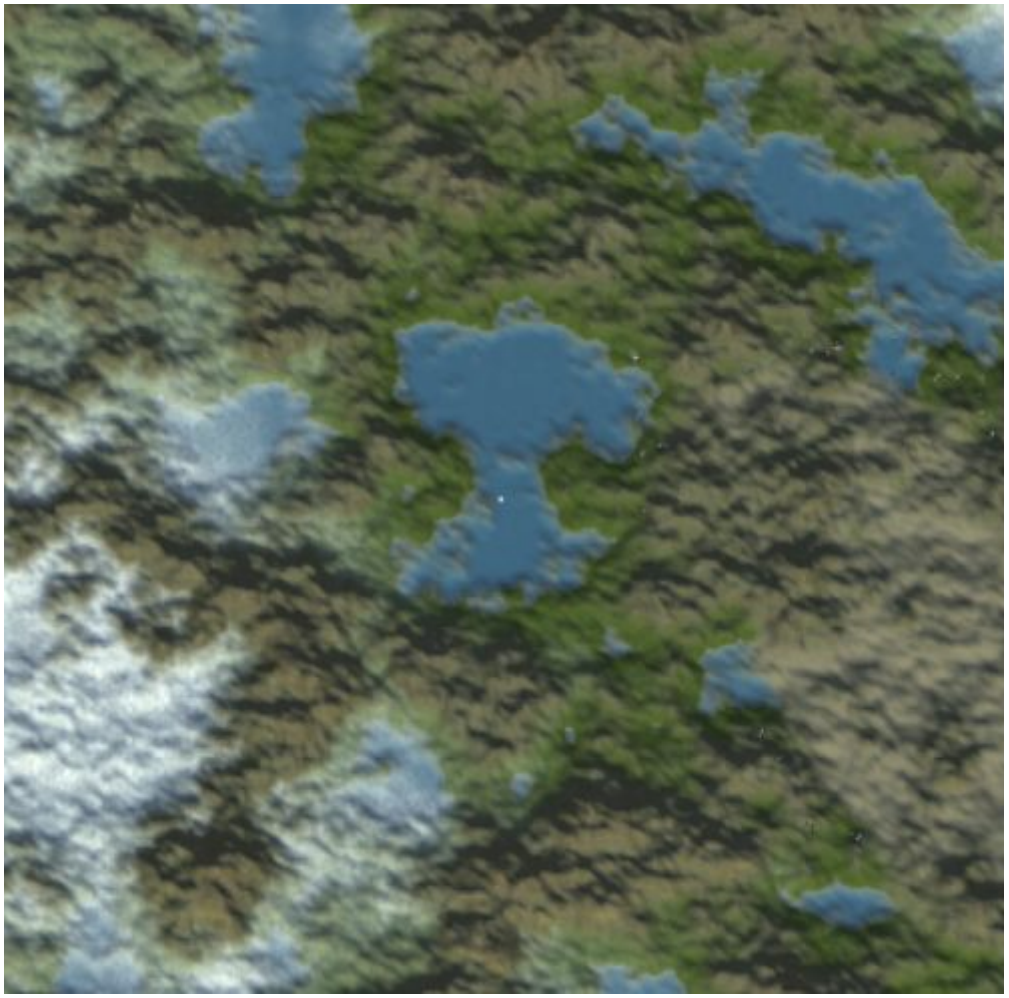

Figure 19: Top view of a generated world

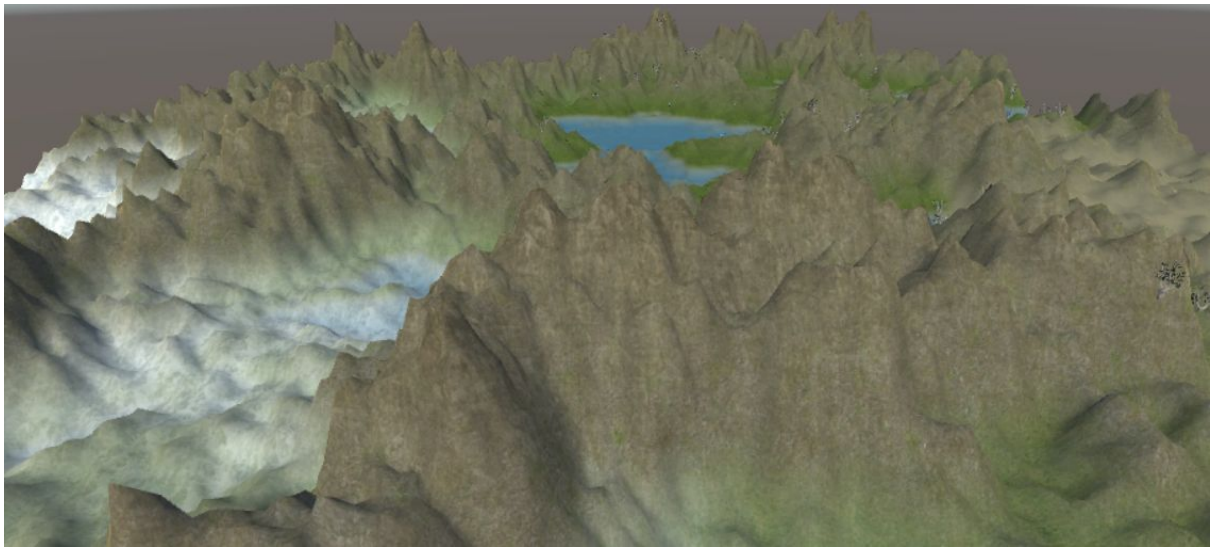

Figure 20: Side view of a generated world