Procedural Content Generation

Modern Game AI Algorithms – Assignment 1

Jerry Schonenberg (s2041022)

07-03-2021

1 Introduction

The objective for the first assignment of Modern Game AI Algorithms is to generate 2D-maps using procedures. This report contains a description, explanation and my considerations with regards to generating a map which should be realistic and diverse. All maps that are in the report have a resolution of 1280×720 (720p).

2 Procedural Content Generation

This section contains the components with which a map is generated. The components consist of: relief, biomes and populating the map. Every component is discussed in its own subsection and its effect on the map is also shown. For the random seed, the value 42 is chosen in order to showcase the creation of a map, component after component.

But before the components can be discussed, one important foundation has to be addressed. Instead of using the RGB color-space, this project uses the HSV color-space. With HSV-values, it is possible to easily change the color of a pixel by altering its Hue-component. This leaves the Saturation and Value components available to encode information into. The project encodes the height of a pixel in the Value-component in order to create a nice gradient within one biome indicating the height there. Consequently, a dark and light pixel correspond to a low and high height, respectively. The Saturation-component does not contain any information and is used to finetune the colors in the map.

2.1 Relief

The relief is created by generating Perlin/Fractal noise. This is a gradient noise which is suitable for relief-generation. Its implementation is taken from [Vig20]. Fractal noise is specifically used since Perlin on its own generates relief that is too smooth (round corners) such that it became unrealistic. However, Fractal noise can be finetuned with multiple parameters to generate organic relief. Moreover, since the range of Value is in [0, 100], the Fractal noise is scaled to be in the same range. The default settings are used to generate this noise, with the exception of `res` and `octave`: here the values are set to (3,4) and 5, respectively.

The result of the Fractal noise can be seen in Figure 1. Additionally, the relief is saved in a height map, since the Value-component of the map itself can be changed at a later stage of the generation. This way, the height-information is preserved in case the 2D-map is to be converted to a 3D-world.

2.2 Biomes

The second component consists of assigning every pixel to its biome, based on its Value. In other words, the height of a pixel determines to which biome it belongs to. Each biome has its own unique Hue value. This way, we can easily identify to which biome every pixel belongs to. This will be used in the next component, when the map gets populated.

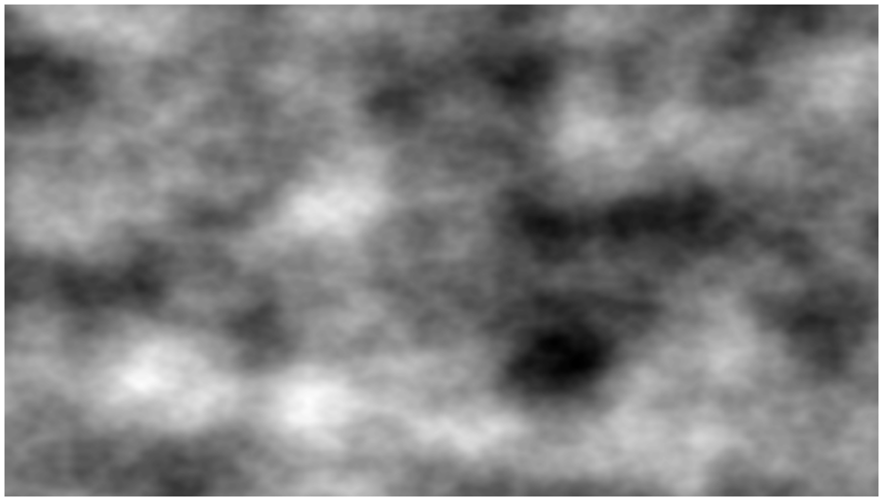

Figure 1: The height map generated with Fractal noise. The dark and light pixels correspond to a low and high height, respectively. The resolution of the image is 1280 × 720.

There are in total seven biomes, which are listed below with their height-range in which they appear:

- **Water**: `[0, 37]`.
- **Beach**: `[38, 42]`.
- **Grass**: `[43, 50]`.
- **Forest**: `[51, 75]`.
- **Dirt**: `[76, 80]`.
- **Mountain**: `[81, 96]`.
- **Snow**: `[97, 100]`.

As can be seen in the enumeration, the ranges are chosen such that there is no intersection between any of the biomes. Consequently, the assignment of biomes to the pixels is deterministic and is achieved by simply iterating over all pixels in the map. This process results in the map visualized in Figure 2.

One disadvantage of using the Value to indicate the height is the fact that not all colors are available to use at every height. For instance, the beaches surrounding the water are too dark (Figure 2) since its height is low, while we would prefer lighter beaches. The same can be said for the mountains, which should be grey. In order to solve this, the Value of these pixels are increased/decreased with a constant after all other components of the generation are completed. This way, the gradient is preserved while the colors are more comparable to the reality (see Figure 4 for the actual colors). Thereby, since a height map is kept (which is not altered by this operation), all height-information is preserved for situations where it might be required (e.g. transforming the 2D-map into a 3D-world). Moreover, it is not possible to change the Value of these pixels when assigning every pixel to a biome, since the height will be used when populating the map.

2.3 Populating the map

This is the final component of the generation, here the map is populated to bring life into the landscape. The step consists of generating the following objects to the map: vegetation, villages, roads, boats, volcano's and flags on top of mountains.

- **Vegetation**: (forest, dirt), $p_{\text{forest}} = 0.05$ and $p_{\text{dirt}} = 0.005$.
- **Villages**: (grass), $p = 0.0003$.

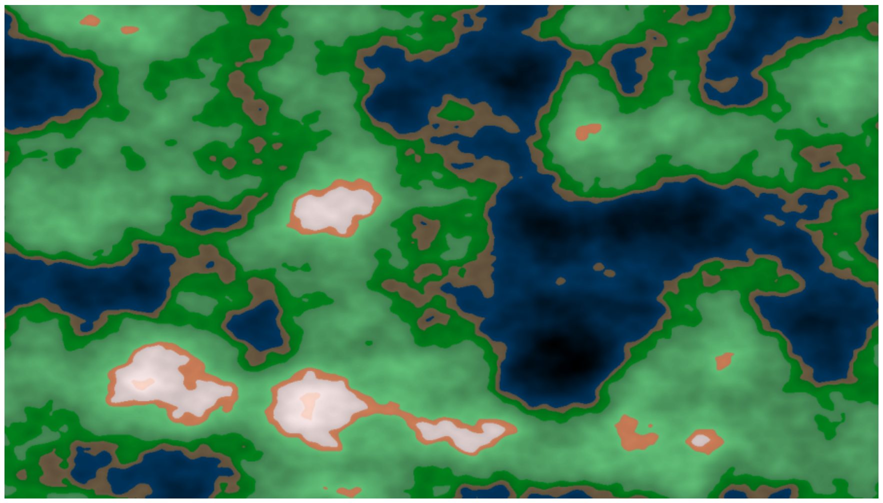

Figure 2: The map with all pixels assigned to one of the seven defined biomes (water, beach, grass, forest, dirt, mountain or snow). The resolution of the image is 1280×720.

- **Roads**: (N.A.), $p = 0.04$.
- **Boats**: (water), $p = 0.00005$.
- **Volcano's**: (snow), $p = 0.3$, unless there is already a volcano, then $p = 0.0$.
- **Flag on top of a mountain**: (snow), $p = 0.001$, unless it is a volcano, then $p = 0.0$.

Every object is assigned to one or more biomes where it might spawn, this set of biomes is denoted in between the brackets (in the enumeration). Moreover, its probability p is also denoted. These probabilities are finetuned for the resolution (720p) with which is experimented, such that every object is not too or less frequently present on the map. This also applies to all other parameters used by the generator. Moreover, each probability denotes the chance that the object is generated per pixel.

In short, to populate the map: iterate over all pixels, determine to which biome it corresponds (on the basis of its Hue), determine with the probabilities that apply to that biome if something should be generated there.

Now, the generation of every component is briefly discussed.

Vegetation

A tree/bush is generated on the map, simply by drawing a 3×3 square which has as color a darker green than its environment. In half of the vegetation, a brown trunk is added below the square. See Figure 3 for both the tree and bush. Due to its relative size, the trunks are not visible unless you zoom in, unfortunately.

Villages and roads

Next, villages and roads are generated. The villages are represented with small red blocks (which on its own represent houses). For every village is its origin (i.e. the pixel from which the village is generated) saved. Moreover, around the origin are houses generated with a probability of 0.05 for every pixel in a radius of 7 pixels around the origin. This results in some houses merging together into a bigger building. As mentioned in the enumeration of all objects, villages can only spawn in a grass biome. This is not entirely true: its origin can only spawn in the grass biome. So, it is possible that some houses may spawn on neighboring biomes, such as the beach, water and dirt. This is allowed, except for the water biome; houses are not able to spawn on the water for obvious reasons.

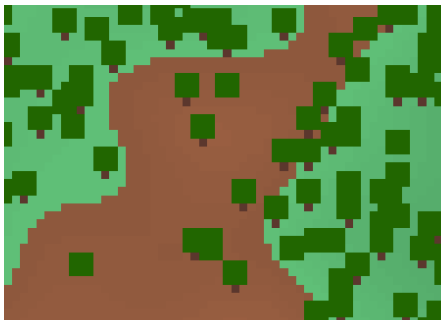

Figure 3: The trees and bushes generated in the grass and dirt biomes. As can be seen, some of the vegetation contains a trunk, representing a tree. The resolution of the image is 1280×720.

Then, when all villages are generated, their origins are used in order to connect some of the villages by road. For every combination of pair of villages is a probability of 0.04 (as denoted in the enumeration) that they will be connected. This road is able to go over all biomes, except for the mountain and snow biome. When the road goes over land and water, its color is black and brown, respectively. This is to differentiate between a road and bridge.

Euclidian distance is used to generate a road, so to avoid large detours. The road ends when it has reached the origin of its destination village. Sometimes, the houses are not equally distributed around the origin, which may result in that the road is not directly next to a house. But that is not something that would make the map unrealistic, since this will be at most 6 pixels.

However, one might say that a bridge across a huge lake is not realistic, when it can also go around the huge lake. Likewise for small ponds, it would be easier to build a road with a tiny detour around the pond instead of going over it. These two situations do hurt the believability of the generation, but is unfortunately not something that we were able to easily fix without significantly slowing down the whole generation process.

Boats and flags

These two objects only appear in their own biome. When one of these objects is to be generated, its definition of which pixels should be overwritten is hardcoded (relative to the pixel at which the object is generated). The flags represent those that mountain climbers place near the top of the mountain.

Volcano's

Lastly, a volcano can be generated on top of a mountain (in the snow biome). If this is the case, the whole snow biome is replaced with a magma/lava texture recursively. This can be done recursively since there is a boolean matrix which denotes which pixels belong to the snow biome. So, the procedure recursively traverses through a snow biome, turns the white pixels into red pixels and flips the corresponding boolean in the matrix. This traverse is done until a different biome is reached, if so then that recursive instance is ended. This procedure does have a disadvantage, namely that when the biome contains too many pixels, its recursive depth becomes too large for Python to handle. In order to avoid this, the snow biome only appears from height 97 and up, resulting in small snow biomes. But if you would increase the resolution of the image, then the recursive depth would still end up too deep. But with 720p as resolution and the current settings, no errors occurred.

Then, when the top of the mountain is transformed into a volcano, a stream on the side is generated randomly. Here, a random traverse is executed where it is not allowed to walk backwards (relative to the previous movement), and if the stream reaches the forest biome, it stops. Moreover, in order to stop the stream getting too large, a small probability of 0.001 is added which determines the chance for every step to be the last.

However, the generation of volcano's does not take the other snow biomes into account. For instance, it would not be believable to have a volcano next to a snow biome. But this is unlikely since snow is very rare, so the chance of having two separate snow biomes next to each other does not happen often. Other than that, the generation of the volcano is realistic with regards to its size, shape and how the lava flows out of it.

3 Resulting map

After all objects are added to populate the landscape, the generation process is completed. The resulting map is visualised in Figure 4. Here, the color corrections, as mentioned in Section 2.2, are also applied, resulting in the output of the generator.

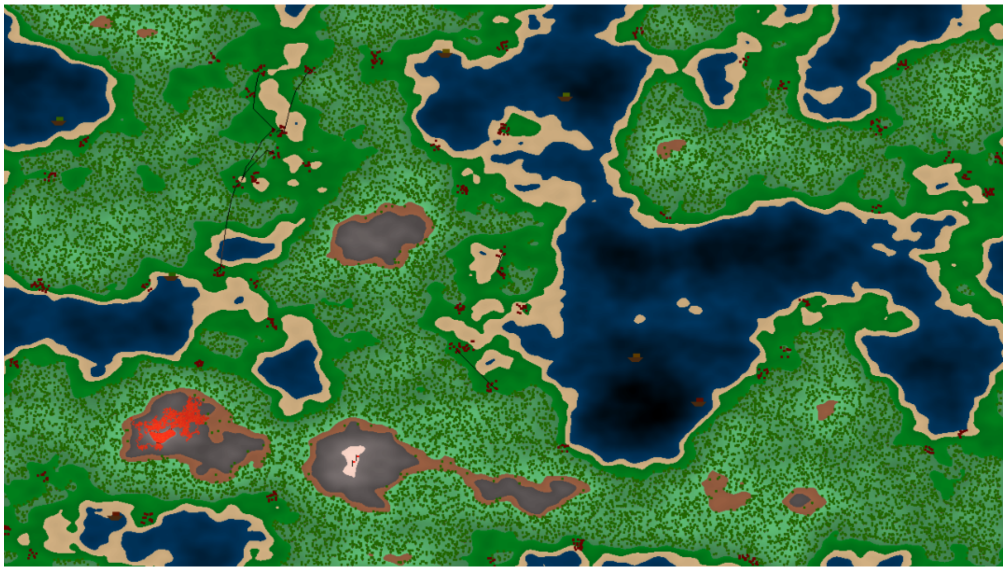

Figure 4: The resulting map after all objects are added into the landscape. This is also the output of the generator. The random seed is set to 42 and it has a resolution of 1280×720.

4 Conclusion

This project delivers a 2D-map generator which generates realistic landscapes with the use of procedures. It generates a believable relief-map with the use of Perlin/Fractal noise, and based on this map are the biomes defined. Moreover, HSV-values are used in such a way that the Value represents the height of a pixel, resulting in that a gradient occurs within a biome. This gradient indicates the height differences among pixels (light and dark pixels are for high and low heights respectively).

However, the way in which roads are generated is not always believable. This is since it follows a path according to the Euclidian distance, resulting in the fact that sometimes the road might go over a pond or huge lake, while it would have been easier to go around the water (even though it is a small detour). Moreover, it is possible that a volcano can be generated next to a snow biome, which is not realistic. But other than that, the project is able to generate a realistic landscape with biomes, boats, trees, villages, beaches and more.

References

[Vig20] Pierre Vigier. Perlin numpy. github, github repository. https://github.com/pvigier/perlin-numpy/, commit: 6f077f811f5708e504732c26dee8f2015b95da0c, August 31, 2020.