MGAIA Assignment 1

Evan Meltz s3817911

March 2024

1 Introduction

In this assignment, the goal was to work with Procedural Content Generation (PCG) in order to build houses in Minecraft. This involved incorporating techniques into the code, which uses the Generative Design Python Client (GDPC) library, that ensure randomness amongst each house and well as the believability and adaptability of where in the environment the house gets placed.

2 Inspiration

Before beginning on the algorithm, the first goal was to decide on the style of house to build. Considering the environment and terrain of Minecraft is, for the most part, secluded, this gave me the idea of an old rustic log cabin which would typically be found in the woods or in the snow.

Figure 1: Log Cabin Inspiration.

Figure 1 is an example of a typical log cabin with its curved roof, minimalist structure and the abundance of nature surrounding it. While I took inspiration from pictures like this, my final design

was based on experimenting with different materials and finding the ones which suited it best.

3 Believability & Adaptability

One of the primary concerns of implementing this algorithm was scanning the environment to ensure the most suitable sub-area within the random build area is found to build the best. The primary factors which were taken into consideration were the flatness of the terrain and whether or not there was water in the area. This was to ensure the house would integrate believably into the environment without placing in trees of in the water or anywhere unsuitable for a house. Also it ensures the house can adapt to the environment and find the flattest area to then make the least amount of altercations necessary to be able to have a completely flat surface to build on. To implement this two functions were created: `find_optimal_building_spot()` and `flatten_build_area()`.

Firstly using the functions of the GDPC library, the buildarea was retrieved and used to get the heightmap of the area. This refers to the height of the topmost solid (or fluid) block at each x,z co-ordinate within the buildarea. Using this, the function `find_optimal_building_spot()` would take in this information and search over all areas of size 15x15 within the 100x100 build area and measuring the variance in height for these areas. The idea was to find the area with the lowest variance as that would mean the difference in height is the lowest possible meaning it is the most suitable area to build in and will require the least amount of modification to the environment possible. In addition to this, the function will also scan each block in the heightmap to ensure its not water and if water is found the area is disregarded and the next one is looked at. This to ensure believability. Note that the algorithm scans areas of size 15x15 and moves in step sizes of a default amount = 15. This can be easily changed by calling the method and passing in a different step size. However through experimentation it was determined this was the best balance between efficiency and precision. The value can be set lower, which will result in increased accuracy in terms of the most optimal area which is found however it will come at a trade-off of increased computing power required. Once the function has run through all areas, the co-ordinates for the most optimal area are returned along with the corresponding variance. It is checked that an area without water was actually found and if the area that was found has a variance below the acceptable threshold of 10.0(This value was chosen as any higher would require too much modification to the environment) and if either of the requirements are not met the program will end and encourage the user to set a new build area. A visual plot showing the Terrain Variance Evaluation can be seen below in Figure 2

At this point The second function flatten_build_area(), will then take as input the co-ordinates of most optimal area and then work out the average height of all blocks in this area(again making use of the heightmaps). Once the average is found, the function removes all blocks above the average height and fills in all blocks below the average to ensure a stable and flat foundation. Lastly a wooden platform is placed along this area to ensure realism as it would make sense to have a foundation for a house. The average height is then returned. Now, using the optimal co-ordinates found earlier along with the average height found, a new build area is created which will be used to construct the house.

Figure 3 shows the implementation of this by showing how the houses ignore the water and find the flattest land available, even in an area primarily surrounded by water.

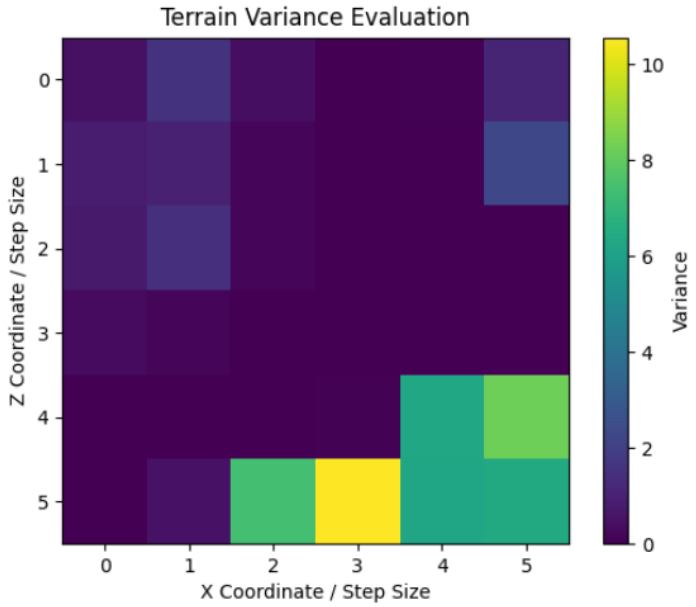

Figure 2: Plot showing variance in terrain for each area according to step size.

4 Randomness

The other main concern dealt with in this algorithm was the essence of PCG itself which is randomness. This means that each house is not of the same size with the same configurations, but instead, while following a structure, has random elements incorporated into it. This is done throughout the algorithm in the construction process to ensure each house is as unique and random as possible. The first thing done, was the dimensions of the house. This was randomised to between 6-7 by 9. This was chosen to ensure the houses were of an adequate size, while still being random and that they can also have even or odd dimensions. Figure 4 shows houses of even and odd dimensions.

Note that for symmetry, even dimension have two doors while odd ones have one. Also notable is the difference in height which is randomised from 5,7. This value can easily be extended for further experimentation, how lower or higher seemed architecturally unpleasing. Adding to the design of the house, log pillars were used for each corner of the house and glass was put in the curve of the roof. These features are consistent among all houses are they contribute to the overall design of a log cabin. However the roof is not consistent and changes which can be seen in Figure 5.

Another main element to incorporate randomness was the orientation of the house and its placement within the build area found. The orientation can be vertical or horizontal and the house can be anywhere within the build area. This can be seen below in Figure 5.

The last significant incorporation of randomness is the interior of the house. All houses will contain 3 hanging lamps for a light source, while the one on the highest part of the roof is random, the other two will always appear in random but opposite corners to ensure light is even spread throughout

Figure 3: Figure showing the adaptability and believability of the house placement

the house. This is seen in Figure 6 below.

In addition, the house will always consist of: a bed; a chest; a crafting table; and a furnace (all the essentials needed for a proper starting house in Minecraft). However the randomness comes in as the bed can spawn in a variety of colours and in either corners, the other items always spawning randomly along the opposite wall to ensure randomness but not having everything cluttered. An example of this can be seen below in Figure 6.

5 Conclusion

In conclusion, this algorithm successfully incorporates the main ideas behind PCG and builds a house, by first choosing the most suitable and realistically feasible area to build it in. Then incorporates randomness into each house to almost be guaranteed that no two houses will generate the exact same. A key insight from this project is the careful balance between achieving optimal solutions and maintaining computational efficiency. Although limited by available computational resources, this exploration highlights the necessity of finding a middle ground where the algorithm remains practical yet effective. There is significant scope for advancing PCG techniques to develop even more advanced and varied algorithms, as has been done many times before.

(a) House of even dimensions

(b) House of odd dimensions

Figure 4: Houses of different dimensions

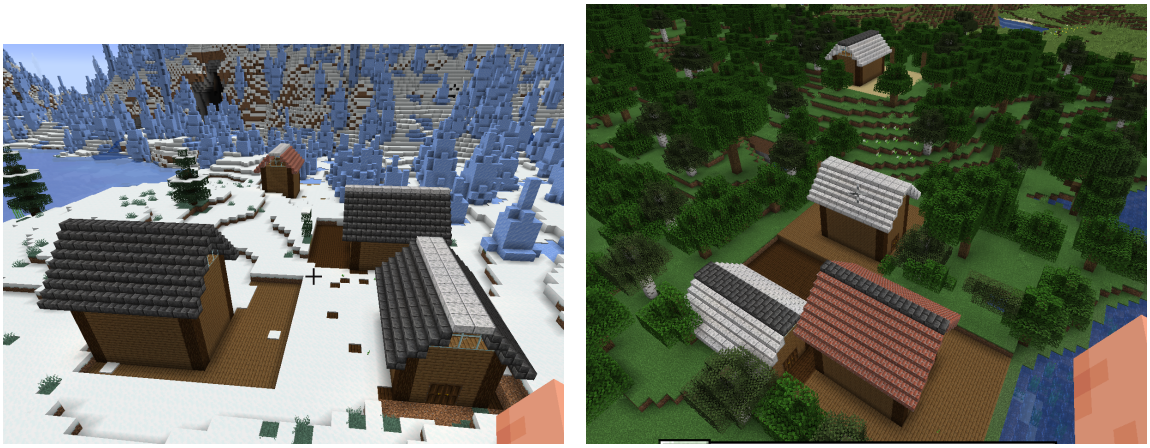

Figure 5: Houses of different orientations and positions

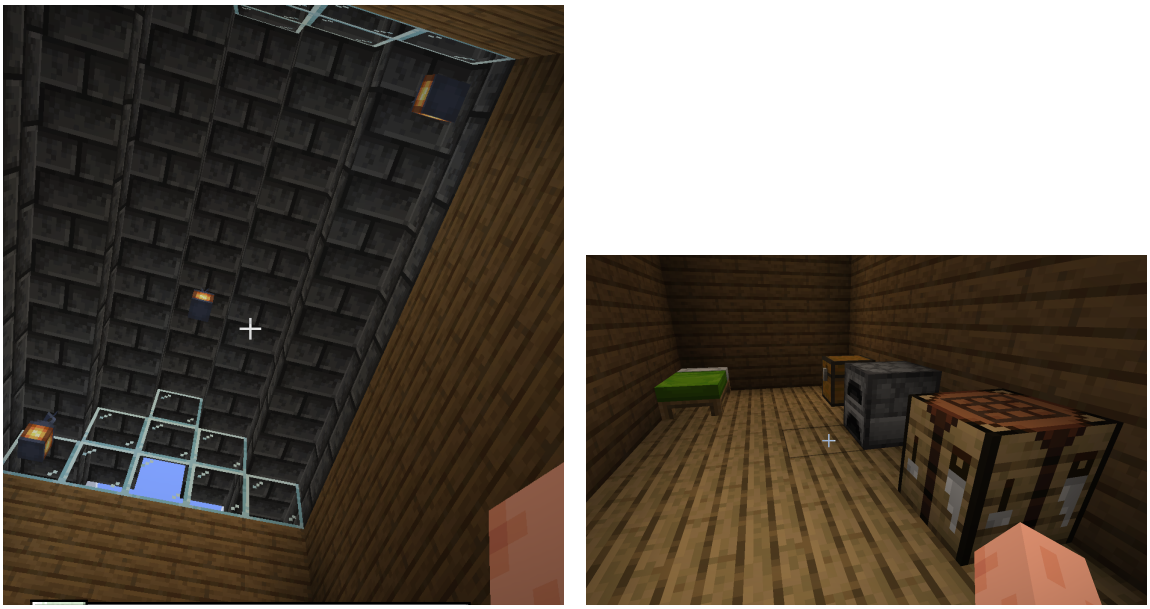

Figure 6: Houses interior

References

[1] avdstaaij. *GDPC (Generative Design Python Client)*. [Software]. Available at: `https://github.com/avdstaaij/gdpc`. [Accessed 1 March 2024].

[2] Rustic Log Cabins. (n.d.). *Image of rustic cabin*. Retrieved from `https://l.icdbcdn.com/oh/60907f50-c4d6-4044-9422-b536a7fdabfa.jpg`.