ASSIGNMENT 1: PROCEDURAL CONTENT GENERATION

Suju Li
4024354

1 INTRODUCTION

Procedural Content Generation (PCG) in the context of games involves creating game content algorithmically rather than manually designing every aspect. In Minecraft, the game uses algorithms to generate the vast and diverse world that players can explore and interact with.

For example, the terrain shown in Figure 1, including mountains, caves, rivers, and biomes, is procedurally generated. Instead of a designer creating each mountain or cave by hand, the game uses algorithms to create them based on certain rules and parameters. This allows for a practically infinite world with unique landscapes in every new game.

Figure 1: The landscape is automatically generated. Every time after creating a new world, the terrain would be different.

In this assignment, a house would be generated in Minecraft within a defined building area. The algorithm is designed to scan the selected environment, and places its structures in a fitting way, i.e., believability. Moreover, it will build a house that has probably floors of 2, 3 or other integer, doors facing south/north/east/west and so on, i.e., variation.

2 PREREQUISITE

2.1 HARDWARE ENVIRONMENT

The PCG program runs in the following hardware environment:

- Chip: Apple M3 Pro
- Memory: 18 GB

Minimum requirements should be satisfied to run Minecraft and conduct my PCG algorithm:

- CPU: Intel Core i3-3210 / AMD A8-7600 APU or equivalent
- RAM: 8GB
- GPU: GeForce 700 Series or AMD Radeon Rx 200 Series with OpenGL 4.5
- Storage: SSD with at least 4GB of available space

2.2 Software Environment

The software requirements are presented in the following list:

- Operating System: Windows 10 or higher, macOS 10.12 Sierra or higher, or Linux
- Minecraft 1.19.2 (Java Edition)
- Up-to-date Java version: openjdk version 17+ should be sufficient.
- Minecraft Forge - MC 1.19.2
- GDMC HTTP interface mod (v1.0.0)
- GDPC: a Python framework for use in conjunction with the GDMC-HTTP mod for Minecraft Java edition.
- IDE: vscode could be an option.

3 Implementation

3.1 Where to build

Given a $100 * 100$ building area, it would be reasonable to scan the area and get familiar with the terrain: is there an ocean? Are there any flat places which is big enough to build the house?

My implementation will build a house having an area of $10 * 10$ on a land of $16 * 16$. Thus, for each $16 * 16$ grid in the $100 * 100$ building are, we will iterate every block on the grid to see if there is any "ocean" block. If yes, this grid will be marked as invalid. If not, the degree of flatness would then be calculated, which is defined as the average of the difference between the height of each block and the average height of the grid. After an iteration process, it could be know that if there is a suitable area to build the house. If yes, a coordination implies the starting point of the construction would be returned, shown in Figure 2. If not, a message shown in the Figure 3 will be given in the terminal. The heightmap of a sample selected area could be illustrated in Figure 4, which can evaluate the scanning and selecting process.

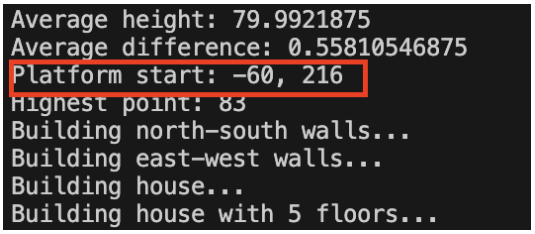


Figure 2: A message within the red box shows the coordination of a start point for the future building.

3.2 House Platform

The terrain selected for the house is highly probably not a flat place, though it is the most flat area in the build area. So it would be a good practice that a platform that is adaptive to the terrain could be built before any house construction. The idea is to find the highest point in the building grid and than we build a base, as shown in Figure 5, starting from the ground to this point.

```
Building platform...
Scanning terrain...
Starting coordinates of scans: (50, 100, 50)
Ending coordinates of scans:(150, 200, 150)
7056 grids scanned, 0 grids are valid and 7056 grids contain water surfaces
No suitable subgrid found within the given build area.
No suitable area for building platform found. Please use another area.
Done!
```

Figure 3: A message shows there is no suitable area for building.

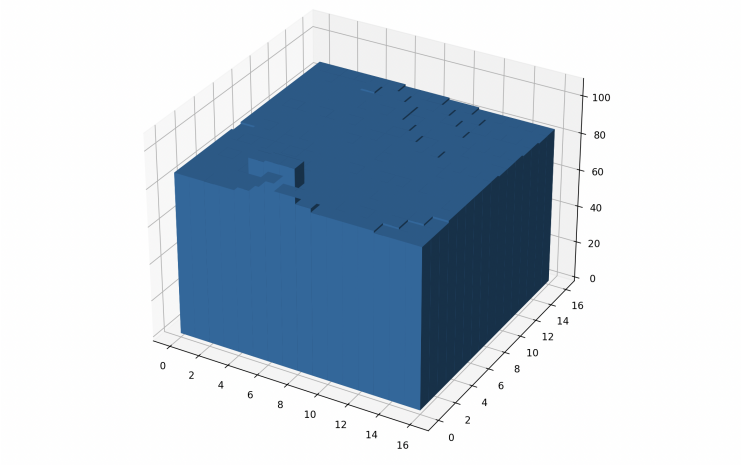


Figure 4

3.3 HOUSE

As shown in the following code, a class *House* is designed to describe how the house will look like. The *nfloors* describes how many floors the house has. The *isRoof* and *isCampfire* show if the house will have a pointed roof or a flat one and if campfires will be inside, which is randomly assigned with *Ture* or *False*.

```
class House:
   nfloors = randint(1, 5)
   isRoof = random.choice([True, False])
   isCampfire = random.choice([True, False])
```

Next, we start to build the house from the ground floor to the top recursively. In each iteration process, a class *floor*, shown in the following code, will be created. The *nthFloor* records on which floor we are building and *floorHeight* describe how high of this floor will be. The *windowWidth*, *windowHeight*, *windowStart* and *stairsEntranceStart* shows the width, height, start point of the window and the start point of the stair entrance. It can be noticed that the window properties could be assigned with a random value.

```
class Floor:
   def __init__(self) :
      self.nthFloor= i+1
      self.floorHeight= 10
      self.windowWidth=randint(1, 3)
      self.windowHeight= randint(2, 4)
      self.windowStart= (random.choice([houseSTARTX, houseENDX]),
          randint(1,10),random.choice([houseSTARTZhouseENDZ]))
      self.stairsEntranceStart= (0,0,0)
      pass
floor= Floor()
```

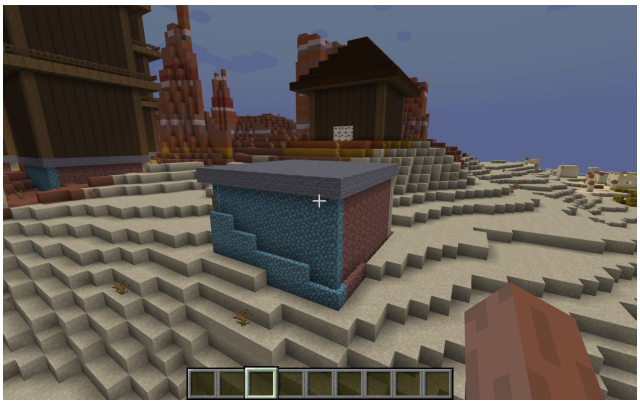


Figure 5: The center of the figure shows the platform.

Next, we use the following customized methods to build on each floor. The functionality of them could be implied from the comments on the codes.

```
#build walls
buildWalls(ED, houseSTARTX, houseSTARTZ, houseENDX, houseENDZ, zfloor)
# build fence
buildFence(ED, house, houseSTARTX, houseSTARTZ, houseENDXhouseENDZ, z, i)
# build door
# build door in the first floor
door_side = buildDoor(ED, houseSTARTX, houseSTARTZ, houseENDXhouseENDZ,
    z, floor, i)
# build ceiling
buildCeiling(ED, houseSTARTX, houseSTARTZ, houseENDX, houseENDZz, floor)
#build windows
buildWindow(ED, houseSTARTX, houseSTARTZ, houseENDX, houseENDZz, floor,
    door_side, i)
# build carpet
buildCarpet(ED, houseSTARTX, houseSTARTZ, houseENDX, houseENDZz, floor)
# build campfire
buildCampfire(ED, houseSTARTX, houseSTARTZ, z, floor)
#build stairs
buildStairs(ED, house, floor, i)
# build bed
buildBed(ED, houseSTARTX, houseSTARTZ, houseENDX, houseENDZ, zfloor)
# build roof
buildRoof(ED, house, houseSTARTX, houseSTARTZ, houseENDXhouseENDZ, z, i)
```

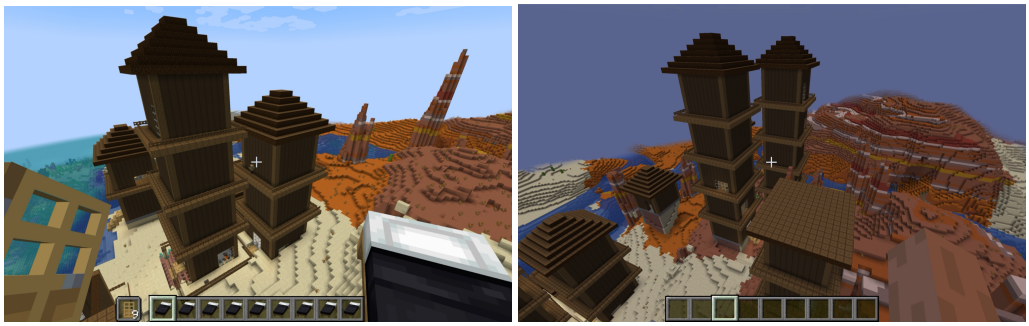
4 Results

4.1 Diverse Houses on the Terrain

After three runs of the PCG program, it could be seen from the Figure 6 that an appropriate area is selected and the house building is adaptive to the terrain. Moreover, the houses could have different numbers of floors and a pointed roof or a flat roof.

4.2 Diverse Windows Facing and Sizing

Except for ground floor, window could face to south, north, east, or west. On the ground floor, window could be on the side different from that of the door. Besides, the size of the windows could range from $2 * 3$ to $4 * 5$. The diversity could be illustrated in Figure 7.



(a) (b)

Figure 6: House diversity design.

Figure 7: We can see two windows having different facing and sizing.

4.3 Door

The door could have four different facing as well. It is framed with light color stones and decorated with a black wall sign, as shown in Figure 8. Moreover, there are two torches and a door step in front of the house, which is convenient for anyone who lives or visits here.

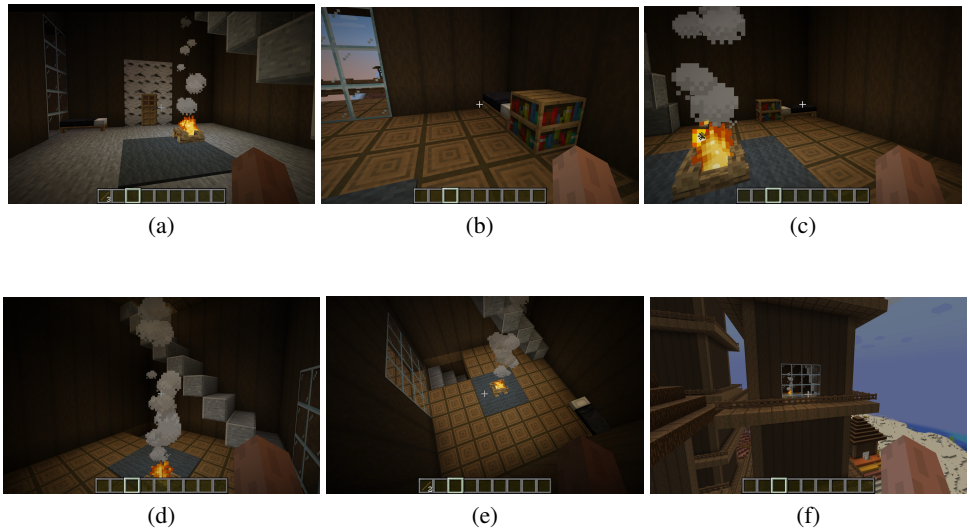
4.4 Interior Decoration

On each floor, a bed, bookshelf, carpet, stairs are placed in default. A campfire could possibly be on somewhere center of this floor. And the location of bed and bookshelf could be slightly different from the other floors. Interior design is illustrated in Figure 9.

5 Conclusion

Believablity/Adaptability In this assignment I design a house that will not be built over water and will choose a relatively flat area to build on. I designed the door way and placed torches so that players can easily find and walk to my house, even at night.

Figure 8: The design of the door.



(a) (b) (c)

(d) (e) (f)

Figure 9: Interior design

Variation For the goal of variance, I used a $random$ module to simulate the process of building a random house, which can exhibit a certain amount of randomness in the number of floors, the roof design, the size and orientation of windows and doors, and the interior decorations.

The project repository for this assignment is here.