Procedural Content Generation in Minecraft

Nikki Rademaker (s2641887)

March 2, 2025

1 Introduction

Minecraft, developed by Mojang Studios, is a very popular sandbox 3-dimensional (3D) video game that has a single player and multiplayer mode. It is known for its worlds exclusively made out of blocks. The game itself does not have a specific goal.It is up to the player to decide what they want to do. Depending on the game mode, players can for example unleash their creativity by freely building structures in creative mode, or gather materials and survive against threats in survival mode. The game was fully released in 2011, and continues to still be updated and played by many [1][2].

Procedural Content Generation (PCG) in is a technique that can be used in game development. In this context, it refers to the use of algorithms to automatically create (randomized) game content rather than creating the content manually. PCG can for instance generate landscapes, levels, loot systems, and quests [3][4]. The Minecraft game shows a good example of how PCG can be used to ensure efficiency and randomization in game development. Minecraft relies on PCG to generate its block-based world. By using random seeds and noise functions, the game procedurally generates terrain, biomes, caves, and structures. This ensures that no two worlds are identical. Despite the randomness, PCG ensures that each Minecraft world is both unique and consistent [4].

The Generative Design in Minecraft Challenge (GDMC) is a competition focused on using PCG for architecture in Minecraft. Participants develop AI agents that generate settlements. These settlements adapt to the terrain without manual intervention. It tests how well structures can adjust to their environment while staying consistent in design. The main goal of the competition is to enhance and expand PCG for games, mostly when it comes to making it more adaptive and holistic [5] [6].

Building on the principles of PCG and GDMC, this project focuses on procedural building generation in Minecraft. The goal of this assignment is to procedurally generate a house in a set 100 by 100 build area which needs to be set in Minecraft before generating the building. The building should adapt to different terrains and be logically integrated into different environments. Instead of simply placing a pre-defined structure at a fixed location within the area, the terrain should be evaluated to adjust the buildings placement and design accordingly. The code for this project will be developed in Python using the GDPC and NumPy packages. GDPC (Generative Design Python Client) is a Python framework that interacts with the GDMC HTTP Interface mod for the Java Edition of Minecraft. It allows real-time procedural content generation for modifying Minecraft worlds dynamically [7].
Several challenges have to be addressed to achieve the goal for this project. Terrain adaptation and believability are necessary to ensure the house fits into different environments such as plains, forests, mountains, or hills. It should avoid unrealistic placements, such as standing on top of water, and the terrain itself should only be modified when necessary. The building should blend naturally into the surrounding environment. Furthermore, randomness and variation should be integrated. The building, its interior, and its exterior design should not look identical on every execution. There should be randomness while still maintaining a coherent architectural style. By addressing these challenges, the project demonstrates how PCG can be used to create structures in Minecraft that are both adaptive and varied.

This report discusses related work to further explain PCG in games. Furthermore, it contains information about the process of the project, including what the main inspiration is for the procedural generated buildings. The approach for terrain evaluation and the placement strategy of the houses is explained.

This is followed by the final results, which include examples of generated houses. The report concludes with a discussion of the results.

2 Related Work

Green et al. are the organizers of the GDMC [5]. In 2019, Green et al. wrote a paper about a method for procedurally generating buildings in Minecraft. Their approach uses constrained growth and cellular automata to create functional structures that adapt to terrain. Each room of the building is treated as an individual entity which grows block by block. The system ensures structural integrity while generating diverse floor plans. The main strengths are the speed and adaptability, which allows for large-scale generation in little time [8].
Another interesting project explores declarative procedural generation for architectural design. Van Aanholt and Bidarra (2020) introduce a innovative tile-based PCG method with architectural profiles. They define structures based on function and constraints instead of geometric rules. They indicate this method improves adaptability by considering factors like accessibility and connectivity. This enables more context-aware building generation. Since they use a tile-based approach for PCG, they propose this method could work well for Minecraft-style block-based games [9].

Both works focus on adaptability in procedural content generation with regards to architecture. Similar to Green et al, this project tries to ensure structures are smoothly integrated into the world of Minecraft by adjusting to the terrain. This project also has some common ground with the work of Van Aanholt and Bidarra by using declarative constraints and rules for the building generation. Although the scope of this project is smaller than the aforementioned work, the approach of this project tries to make sure procedural generation stays believable and functional for Minecraft structures.

3 Inspiration

For this project, the chosen architectural style for the house is a simple castle design. The main inspiration comes from quadrangular castles. Figure 1 shows an example of this castle style. These castles typically have a rectangular layout, with four outer walls for strategic defense. They often feature corner towers, and a central courtyard. However, the design of the castle for this project has a more compact castle structure without a courtyard. The design consists of two main cuboid sections, one in the front and one in the back. Each is connected to either one or two towers. This means the castle contains either two, three or four towers in total. Unlike traditional quadrangular castles with open courtyards, this design is more enclosed where the towers are directly integrated into the main structure. This makes the architectural style also have similarities to medieval fortresses (Figure 2).

Figure 1: Bolton Castle located in England Wensleydale Yorkshire [10]. This castle has the architectural style of a quadrangular castle.

Figure 2: Nunney Castle is a medieval castle located in the English county of Somerset. [11]

Making sure the castle gets build at a suitable location is a very important part of this project. Inspiration for the evaluation of the terrain in a build area was inspired by how site selection for a construction is done in real life. Factors such as flatness, and accessibility determine where buildings are placed [12]. The placement selection process for this project follows a rule-based heuristic evaluation of the terrain. It is rule-based rather than algorithmic. It uses predefined conditions and rules to assess terrain suitability. Furthermore, it was inspired by a greedy selection approach. The code iterates over possible locations and selects the best-scoring coordinate based on predefined terrain conditions. The next section will further explain how the terrain evaluation of the build area, and the placement strategy of the castles in Minecraft are executed.

4 Terrain Evaluation and Placement Strategy

The code evaluates terrain suitability using a greedy approach. It iterates over the 100x100 build area in steps of 8 blocks along the x and z axes. It also takes the castle's full width and depth into account to ensure the structure fits within the selected site. This means the code checks potential building sites every 8 blocks instead of checking every single block. This is done to speed up the search while still covering the build area.

Each candidate position is assessed based on two key factors that contribute to its evaluation score, which are the flatness of the candidate area, and the obstacles within this area. Additionally, the code accounts for a pre-defined set of block types where the castle is never allowed to build upon. These block ids can be found in the Appendix (A).

4.1 Evaluation Scores

For calculating and determining the evaluation score of a candidate area, two different functions are used (*evaluate_terrain(pos, width)* and *area_search(pos, width)*). The function *evaluate_terrain(pos, width)* gets a region from the heightmap that corresponds to a candidate area. The heightmap stores the ground height at each (x, z) in the build area. In the function the difference between the highest and lowest elevation in this region gets calculated. A lower difference means flatter terrain, which is preferred for building the castle. If the elevation variation exceeds 30 blocks, the area gets penalized with a high evaluation score of 2000, making it unlikely to be selected. This is to avoid super steep height differences that are often caused by cliffs, the edges of mountains, or caves.
The function calls on another function called *area_search(pos, width)* to scan a 3-block height range above the ground in the candidate area. Here, logs of trees, and wooden planks are counted as obstacles. On top of this, *area_search(pos, width)* also accounts for regions with water or lava. Water and lava are given a higher penalty to prevent generating castles on top of or near water or lava.
In the *evaluate_terrain* function, a weighted score gets assigned, where the flatness and obstacle count with a weight of 8 are combined to get the evaluation score of a specific candidate area.

```
score = flatness + 8 * obstacles
```

The lower the score, the more suitable the location seems for building the castle and its exterior design.

4.2 Find Suitable Building Location

The function *find_suitable_location()* calculates an evaluation score for each candidate position (using *evaluate_terrain*). It initializes a variable *best_score* to infinity and *best_position* to None. The best score and best position get updated if a lower position score than the current one is found as the function iterates over candidate locations. At the end, the candidate with the lowest score is selected as the best location for building. This function also checks if the block directly below is forbidden to build on and then skips candidates that are. It is possible for content to me generated near those blocks or at the same location, but it should avoid building on top of these specific block types.

For the exterior design outside the castle, the code selects locations for garden features. It first defines a protected zone around the castle. This prevents the gardens attributes from interfering with the castle by being placed too close or inside the castle. Outside this zone, a random location is chosen within

a 20-block range and withing the building area. The function *garden_location()* checks if the location is suitable by making sure the ground is not one of the pre-defined forbidden blocks. It also uses the *area_search* function to count obstacles. The *garden_location* function returns true if no obstacles were detected. If the location is accepted, a garden feature gets placed. The code repeats this process until a suitable spot is found. This ensures the garden is placed logically. In addition, to make sure the garden features don't get located at the top of a mountain or at the bottom of a hole while the castle is at a different height, they are never allowed to have a large difference compared to the castle placement in starting height.

4.3 Visualizing the Terrain Evaluation

Scatter plots have been created to give a better overview of how the terrain evaluation works. Figure 3, Figure 6, and Figure 9 show the different types of terrains the build area was set at. These images were generated with the help of an example code from the gdpc Github to visualize Minecraft maps [7]. The first terrain (Figure 3) is a common forest biome. Figure 4 shows the evaluation scores of this designated build area. Each dot represents a coordinate in the Minecraft world. The colors indicate the suitability of the terrain for placing the castle. Due to water and lava always being heavily penalized, extreme values for the evaluation score plots have been clipped to the 95th percentile. If these extreme values were included in the visualization, the color scale would be skewed. This makes it more difficult to distinguish normal variations in terrain evaluation. In this figure, it can be seen that water, and trees are seen as obstacles for the content generation. In addition to the evaluation scores, Figure 5 shows the locations of the blocks that are now forbidden to build on. To ensure believability, it should be avoided to build on water, lava, or on trees and other nature blocks, such as flowers, and bamboo. This figure specifically shows that building on trees and their leaves is not allowed which is taken into account for the placement of the castle and its garden.

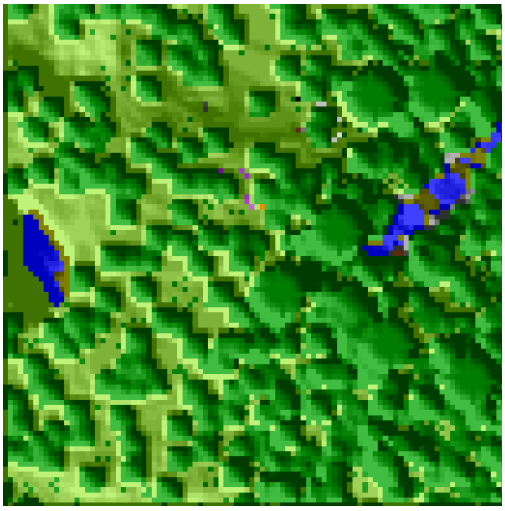

Figure 3: Minecraft 100 by 100 build area of a forest biome.

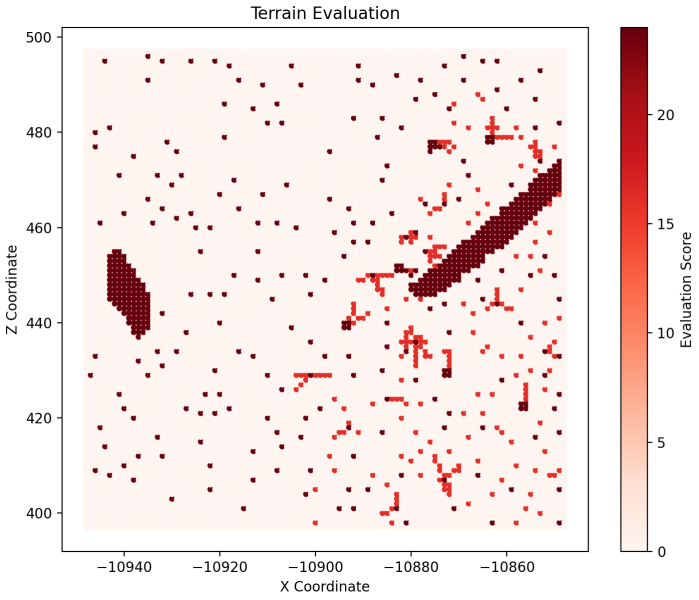

Figure 4: Scatterplot of the build area within the forest biome that visualizes the evaluation score of the terrain. The darker the red, the higher the score, the more unsuitable the location is for building placement.

The second terrain (Figure 6) is a jungle biome with a bamboo field. As can be seen in Figure 7, bamboo is not seen as an obstacle. This is because it is more logical and easier for the castle to be build on a bamboo field rather than in the middle of the jungle that mainly consists of trees with very big logs.

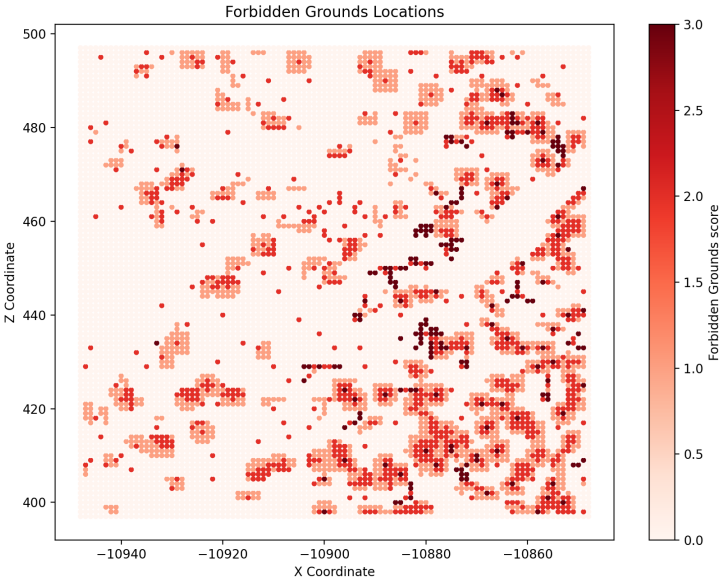

Figure 5: Scatterplot of the positions where forbidden blocks are located within the build area in the forest biome. The castle and exterior should never be built on these types of blocks but are allowed to be built near them. The darker the red, the more forbidden blocks there are in that area.

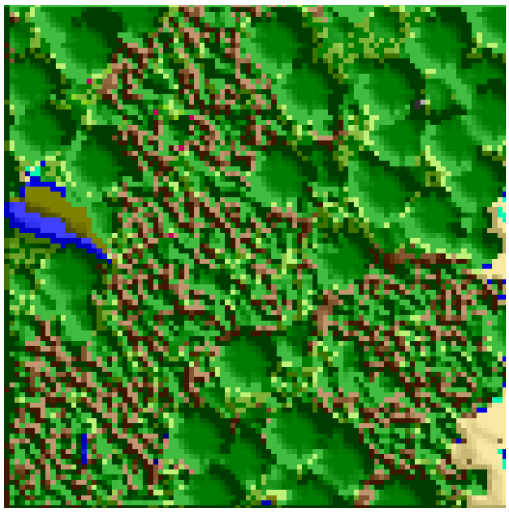

Figure 6: Minecraft 100 by 100 build area of a jungle biome.

However, it is forbidden for the castle and its garden to build on bamboo itself (Figure 8). The code does allow for the castle to build on the bamboo field, replacing the bamboo.
Figure 9 is a map of an ice spikes biome located near a snowy mountain. When the area mostly consists of water, the plot does show the true evaluation values of the locations within the build area (10). It can be seen that land, even if there are height differences, is preferred over building on water and ice. This is also to ensure believability of the placement of the castle. Although it would be easier to build on a flat ice surface, it is not as believable. In figure 11, the location is displayed where the castle gets built within this build area based on the terrain evaluation.

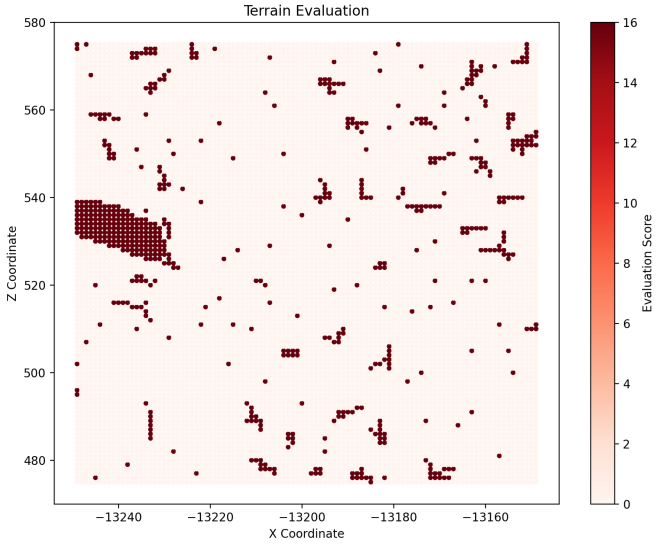

Figure 7: Scatterplot of the build area within the jungle biome that visualizes the evaluation score of the terrain. The darker the red, the higher the score, the more unsuitable the location is for building placement.

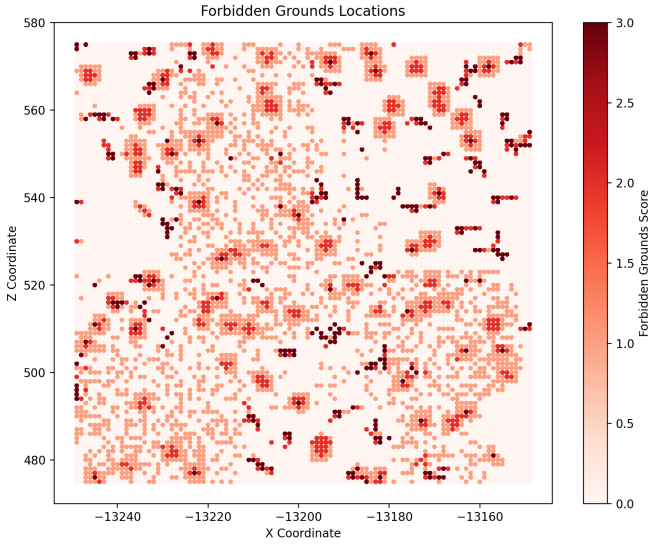

Figure 8: Scatterplot of the positions where forbidden blocks are located within the build area in the jungle biome. The castle and exterior should never be built on these types of blocks but are allowed to be built near them. The darker the red, the more forbidden blocks there are in that area. For example, a castle should never be built on top of bamboo.

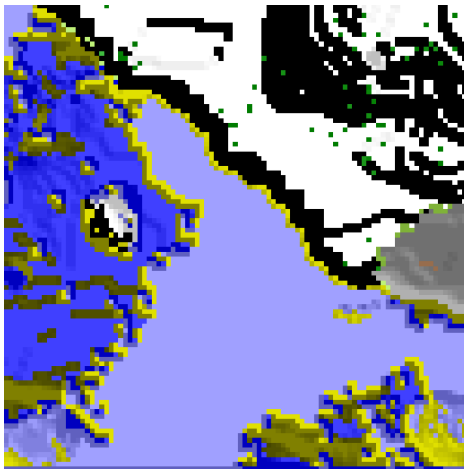

Figure 9: Minecraft 100 by 100 build area of a ice spikes biome.

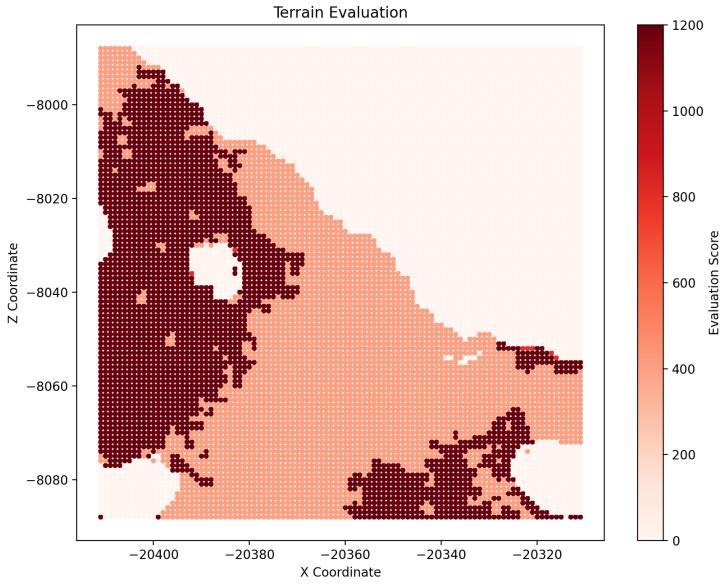

Figure 10: Scatterplot of the build area within the ice spikes biome that visualizes the evaluation score of the terrain. The darker the red, the higher the score, the more unsuitable the location is for building placement.

Figure 11: Location where the castle got built within the 100 by 100 build area of the forest biome.

5 Implementation and Results

The project was developed for Minecraft 1.19.2 (Java Edition) in Python using GDPC, NumPy, and the Minecraft Forge mod. Before running the code, the build area needs to be set withing minecraft through a command (*/setbuildarea* $\sim$ $\sim$ $\sim$ $\sim$ 100 $\sim$ 100 $\sim$ 100). The code can be run by pressing run in the code editor or by using a command to run it in the terminal. This section describes the development process and shows the results of this project.

5.1 Castle Design and its Randomness

The castle is procedurally generated using a combination of structured placement and controlled randomness. Once a suitable location is selected, the castle is built in a structured way while still incorporating randomized elements to ensure variation. The main structure of the castle consists of two cuboids that are connected to each other. The back cuboid is always wider and higher than the front cuboid and is centered behind it. The sides of the cuboids contain towers (cylinders) higher than their respective cuboids, and the diameter of the towers is equal to the depth of the cuboids. The back towers of the castle are hollow and contain a door at the back. Through there, there is a ladder going all the way up to the roof of the back tower. Examples of the front of the castle can be seen in Figures 12 and 13. Figure 14 shows an example of the back of a castle, and Figure 15 shows the ladder going up to the roof of the back tower.

Figure 12: Front view example of a generated castle.

Figure 13: Front view example of a generated castle.

The number of towers is randomized, with either one or two on both the front and back sides. This means the castle has either two, three or four towers. If the castle has two towers (one for the front cuboid and one for the back cuboid), the towers are on opposite sides of each other. The height of the towers is randomized within a predefined range. Similarly, the depths and widths of the front cuboid and the back cuboid are also randomized within set boundaries. To ensure the castle continuously has a logical

Figure 14: Behind view example of a generated castle.

Figure 15: Example of the inside of a tower from a generated castle.

structure, the randomness has pre-defined constraints.
For more randomness, the code ensures random selection of several materials. The blocks or materials for the towers, the doors, the windows, and the base structure are all randomly picked. Despite this randomness, materials are chosen from python lists to maintain a medieval or fantasy aesthetic. The selection for the towers and base includes different types of stone, wood, and metal blocks to ensure visual variety.

5.2 Interior Design and its Randomness

The interior of the castle is also different every time, but it always follows a structured layout. The front cube connects to the back cube through a centered doorway. Inside the front cuboid, there are random decorations. Sometimes a lantern hangs from the ceiling, and sometimes a spore blossom is placed instead. If the front cube has its smallest possible width, it always gets a lantern for proper and cozy lighting. If the front cube is wider than that, there are torches on the walls near the entrance and a blossom hanging from the ceiling. If the front cuboid has enough space, a chest gets placed against the back wall. The chest is either on the left or right side. If there is one, the block above it will be cleared when needed so it can open. Small decorations like potted plants are placed near the windows, either one or two depending on the random selection. If there is only one plant, a different decoration is placed on the other side. The back cuboid is bigger and has furniture. Banners of random colors are placed above the doorway, and a carpet of various sizes is placed in the middle of the cube with random colors. The placement of the carpet avoids the towers so the carpet does not overlap with them. A bed is placed along the back wall, and its color is almost always random. However, if the castle is built with purpur, the bed is always pink, and if oxidized copper is used, the bed is always cyan to match the building blocks of the castle. The bed placement adjusts to avoid towers, and if needed, the space above it is cleared. Next to the doorway between the cubes, a crafting table and furnace are placed. They appear on either the left or right side, chosen randomly. The ceiling of the back cube always has a glowstone layer for lighting. The way everything is placed makes sure each castle looks different while still keeping a functional and structured layout. Through code, all decorations and furniture are placed in a way it will never interfere with each other or with the walls of the castle and towers. Example images can be found in Figures 16 and 17. More figures can be found in the Appendix (A).

5.3 Exterior Design and its Randomness

The exterior of the castle includes a garden area that is placed outside of the protection zone of the castle mentioned before (Section 4.2). This is important so the exterior decorations do not interfere with the main structure of the castle. If the ground is grass, flowers are planted randomly around the area. A small table with two chairs are also placed in the garden. The table is made from a fence post with a

Figure 16: Example of interior of a castle.

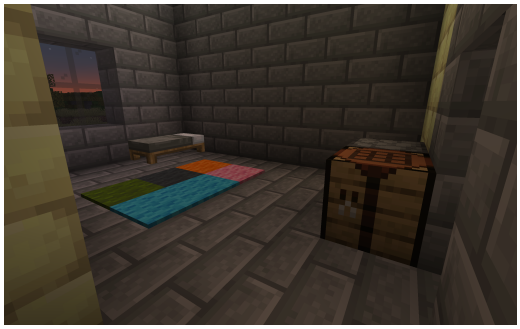

Figure 17: Example of interior of a castle.

pressure plate on top, and the chairs are stairs placed on opposite sides. The orientation of the chairs are random for more variety. The garden also includes one of two constructions: either a small shed made of bookshelves with an enchantment table inside, or a horse with a chest containing useful items. The shed has an open space in the middle, and a doorway is placed on the most suitable wall. The horse, when present, spawns near a chest that contains a saddle, leads, golden apples, and other horse-related items. Both of these features fit the fantasy theme of a castle. The enchantment table ties into the idea of magic, while the horse adds to the medieval aesthetic. These elements fit the fantasy aesthetic castles often have in the real world. Examples images can be found in Figures 18 and 19

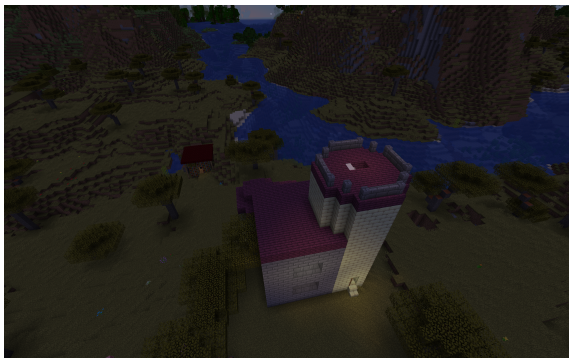

Figure 18: Example of the above view of a castle with its garden.

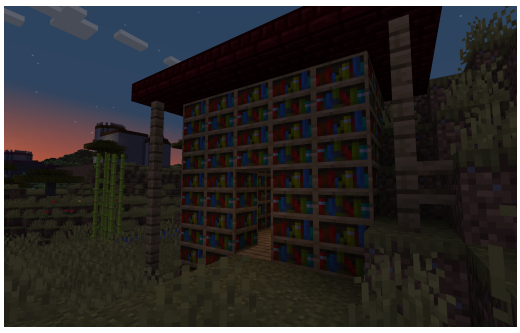

Figure 19: Example of bookshelves shed with enchantment table.

5.4 Adaptability and Believability

The castle and its exterior adapts to different terrains through several mechanisms. Its size adjusts based on terrain density. If there is a lot of space, the cuboids of the castle will always have their largest possible width and depth. On the other hand, if the area is extremely dense, the castle will have the smallest possible width and depth. This is based on the evaluation score of the chosen build position. There is logic to the placement of the exterior elements (as explained in Section 4.2) to avoid overwriting the castle and to ensure believable placement. The bookshelf shed adapts by elongating to the ground when necessary and its doorway is placed on the most suitable side based on available space. Pathways are cleared when needed to provide access to doors of the castle, and ladders are placed when the terrain is too steep to navigate naturally after clearing those pathways. To maintain a believable integration into the Minecraft world, certain specific obstacles are removed that are directly above the towers or base of the castle. To further improve the believability, floating trees above the pathway are prevented by clearing tree longs above those pathways. Meaning, if the pathway is fully cleared and is not underground, there will be no floating trees above it. In certain environments when there is no other option, such as the jungle, trees get deleted to clear space for building the castle. Additionally, for when the castle is placed on uneven terrain, a platform is constructed beneath the protection zone. This is to prevent parts of the castle from floating. The platform fills any gaps beneath the castle with the same block as the terrain to make the transition

more believable. All of this combined, was programmed to allow for integration of the castle into various terrains and environments while trying to still maintaining a logical, usable and believable structure. Example images of the pathways and its ladder can be found in Figures 20, 21, and 22. Furthermore, Figure 23 shows another example of a castle in a different biome.

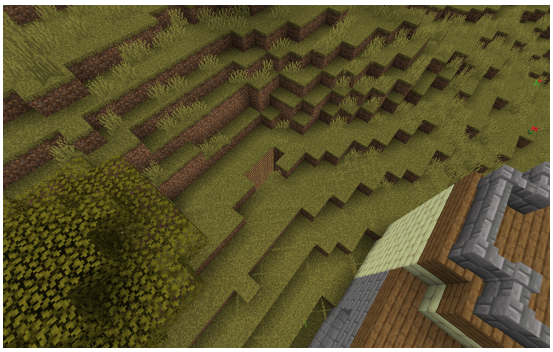

Figure 20: Example of pathway entrance.

Figure 21: Example of pathway to back tower door.

Figure 22: Example of ladder in pathway.

Figure 23: Example of castle blending in with the biome.

6 Conclusion and Discussion

The procedural castle generation works well overall. It adapts to different terrains while keeping the design structured and varied. The code adjusts the size of the castle based on how much space is available. The placement logic helps avoid major obstacles and tries to ensure believability. The pathways and ladders makes sure the castle and its rooms are accessible. Furthermore, the randomization in material choices and structure variations adds variety and randomness to the content generation. The exterior and interior are programmed in a way it does not interfere with each other or the castle, while also mostly remaining random in placement. All these elements help integrate the castle into the Minecraft world in a way that often feels natural.

However, there are some clear limitations. The placement is not always ideal since the evaluation mostly focuses on terrain flatness and certain obstacles. It is hard to determine which obstacles to look out for since it is important to avoid too many limitations for the generation, but also to ensure the placement makes sense without needing to change the environment too much. It also does not look for the best-looking or most convenient spot for players. Another important aspect to note is the platform that prevents floating because in some cases it makes the structure look unnatural—especially when it sticks out from hills. Since the platform is always a square, it can sometimes make the castle feel out of place rather than blending into the environment. Additionally, while variation is important, the randomization could be smarter. Right now, the castle size and features are mostly chosen randomly within a range, but they could be more influenced by the terrain. Adjusting shapes based and the number of towers on surroundings would be a good next step for this project. In addition to this, it would also be possible to use different materials depending on the biome. The main challenges throughout this project were

making sure placements are correct and do not overwrite each other while also allowing for randomness in placement and sizes. This took a lot of time to get correct. Due to this, the terrain evaluation and adaptation to the terrain still contain some issues.
To conclude, there are multiple ways this project could be improved. The placement algorithm could be refined to consider more terrain details, such as slopes and landscape aesthetics, to ensure better integration. As mentioned, the randomness could also be more advanced. The platform could also be improved by shaping it to better match the terrain, rather than always creating a flat square. Despite the limitations, the system still does a good job of generating various castles that mostly fit their environments. With refinements, and implementing advanced algorithms such as Perlin noise for better terrain-adaptive features, the castle generation could become more natural, dynamic, and better integrated into different Minecraft landscapes.

References

[1] Landin, P. (2023, June 5). What is Minecraft. Minecraft.net. Retrieved from: https://www.minecraft.net/en-us/article/what-minecraft?utm_source=chatgpt.com. Retrieved on: Feb 2025.

[2] Zucconi, A. (2022, June 5). The world generation of minecraft. Alan Zucconi. Retrieved from: https://www.alanzucconi.com/2022/06/05/minecraft-world-generation/. Retrieved on: Feb 2025.

[3] Shaker, N., Togelius, J., & Nelson, M. J. (2016). Procedural content generation in games. In Computational synthesis and creative systems. https://doi.org/10.1007/978-3-319-42716-4.

[4] Kenny. (2021, December 31). Procedural Generation: An Overview. Medium. Retrieved from: https://kentpawson123.medium.com/procedural-generation-an-overview-1b054a0f8d41. Retrieved on: Feb 2025.

[5] Generative Design in Minecraft (GDMC) - Generative Design in Minecraft. Retrieved from: https://gendesignmc.wikidot.com/. Retrieved on: February 2025.

[6] Salge, C., Green, M. C., Canaan, R., & Togelius, J. (2018, August). Generative design in Minecraft (GDMC): Settlement generation competition. Proceedings of the 13th International Conference on the Foundations of Digital Games. ACM. http://dx.doi.org/10.1145/3235765.3235814.

[7] avdstaaij/gdpc: A python framework for procedural generation in Minecraft with the GDMC HTTP Interface mod. (n.d.). GitHub. https://github.com/avdstaaij/gdpc.

[8] Green, M. C.,Salge, C., Togelius, J. (2019, June 11). Organic Building Generation in Minecraft. arXiv. Retrieved from: https://arxiv.org/abs/1906.05094.

[9] van Aanholt, L., Bidarra, R. (2020). Declarative Procedural Generation of Architecture with Semantic Architectural Profiles. In 2020 IEEE Conference on Games (CoG) (pp. 351–358). Osaka, Japan. https://ieeexplore.ieee.org/document/9231561. doi: 10.1109/CoG47356.2020.9231561.

[10] Ooms, S. (2019, September 11). Bolton Castle near Leyburn, North Yorkshire, England. https://flic.kr/p/2qbA9j3

[11] Alexander, I. (2018, May 22). Nunney Castle Somerset, England. https://commons.wikimedia.org/w/index.php?curid=69371748.

[12] Sahoo, P. (2024, February 22). 20 Key factors and bonus insights for site selection. The Architects Diary. Retrieved from: https://thearchitectsdiary.com/20-key-factors-and-bonus-insights-for-site-selection/. Retrieved on: February 2025.

A Appendix

IDs of forbidden blocks to build on:

```
tree_logs = ["minecraft:oak_log", "minecraft:spruce_log", "minecraft:birch_log",
    "minecraft:jungle_log", "minecraft:acacia_log", "minecraft:dark_oak_log",
    "minecraft:mangrove_log"]

dont_build_on = ["minecraft:scaffolding", "minecraft:oak_fence_gate", "minecraft:
    spruce_fence_gate", "minecraft:birch_fence_gate", "minecraft:jungle_fence_gate
    ", "minecraft:acacia_fence_gate", "minecraft:dark_oak_fence_gate", "minecraft:
    mangrove_fence_gate", "minecraft:crimson_fence_gate", "minecraft:
    warped_fence_gate", "minecraft:twisting_vines", "minecraft:oak_fence", "
    minecraft:spruce_fence", "minecraft:birch_fence", "minecraft:jungle_fence", "
    minecraft:acacia_fence", "minecraft:dark_oak_fence", "minecraft:mangrove_fence
    ",     "minecraft:crimson_fence", "minecraft:warped_fence", "minecraft:
    cobblestone_wall", "minecraft:mossy_cobblestone_wall", "minecraft:granite_wall
    ", "minecraft:diorite_wall", "minecraft:andesite_wall", "minecraft:melon", "
    minecraft:pumpkin", "minecraft:carved_pumpkin", "minecraft:lit_pumpkin", "
    minecraft:azalea", "minecraft:flowering_azalea", "minecraft:mushroom_stem", "
    minecraft:oak_leaves", "minecraft:spruce_leaves", "minecraft:birch_leaves", "
    minecraft:jungle_leaves", "minecraft:acacia_leaves", "minecraft:
    dark_oak_leaves", "minecraft:mangrove_leaves", "minecraft:azalea_leaves", "
    minecraft:flowering_azalea_leaves", "minecraft:bee_nest", "minecraft:bamboo",
" minecraft:brown_mushroom_block", "minecraft:red_mushroom_block", "minecraft:
    big_dripleaf", "minecraft:cactus", "minecraft:dandelion", "minecraft:poppy", "
    minecraft:blue_orchid", "minecraft:allium", "minecraft:azure_bluet", "
    minecraft:red_tulip", "minecraft:orange_tulip", "minecraft:white_tulip", "
    minecraft:pink_tulip", "minecraft:oxeye_daisy", "minecraft:cornflower", "
    minecraft:lily_of_the_valley", "minecraft:sunflower", "minecraft:lilac", "
    minecraft:rose_bush", "minecraft:peony"]

forbidden_ground = set(dont_build_on + tree_logs)
```

IDs of obstacle blocks:

```
obstacles_ids = ["minecraft:oak_log", "minecraft:spruce_log", "minecraft:
    birch_log", "minecraft:jungle_log", "minecraft:acacia_log", "minecraft:
    dark_oak_log", "minecraft:mangrove_log", "minecraft:oak_planks", "minecraft:
    spruce_planks", "minecraft:birch_planks", "minecraft:jungle_planks", "
    minecraft:acacia_planks", "minecraft:dark_oak_planks", "minecraft:
    crimson_planks", "minecraft:warped_planks", "minecraft:mushroom_stem", "
    minecraft:brown_mushroom_block", "minecraft:red_mushroom_block"]
```

More example figures:

Figure 24: Example of interior of a castle.

Figure 25: Example of interior of a castle.

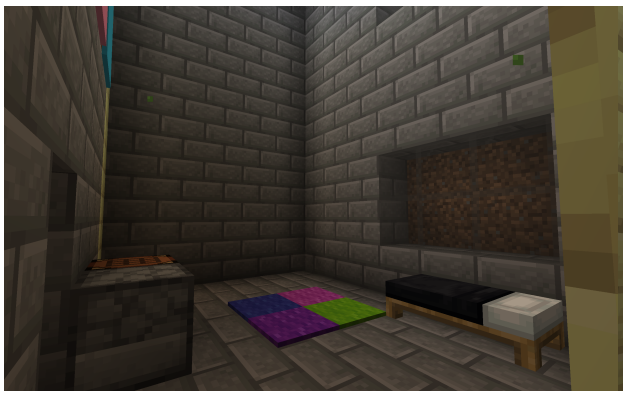

Figure 26: Example of interior of a castle.

Figure 27: Example of interior of a castle.

Figure 28: Example of garden of a castle.

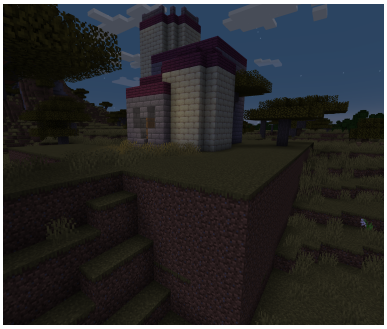

Figure 29: Example of platform that looks out of place