Grammar-Based Procedurally Generated Village Creation Tool

A Senior Project Report

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Engineering

By

Kevin Graves

June 2019

Introduction

This project is a 3D village generator tool for Unity. It consists of three components: a building, mountain, and river generator. All of these generators use grammar-based procedural generation in order to create a unique and logical village and landscape each time the program is run.

The main motivations for creating this tool are to speed up game development and create interesting villages that show the advantages of procedural generation. From my past game development experiences, level design and implementation has always been the most time consuming component. This is one of the reasons there are very few single person or small team produced 3D RPG/Adventure games. To go along with the goal of creating the tool, it is equally, if not more, important that the tool produces a demo village that players will enjoy traversing. The long-term goal of this project is to either provide that tool which will help developers wanting to create these type of games, or provide an example for a more extensive tool to be created.

An interesting part about using procedural generation to create villages and terrain is the vast number of possibilities. Even with a relatively small number of options for each buildings location and building components, there are near infinite possibilities. While a similar number of possibilities would be achievable with random generation, by using a grammar-based procedural scheme, every generation follows a similar pattern so that each village fulfills basic requirements. This means each village will appear to the player as planned out, whereas the developer will not need to do any design work. Also, procedural generation can produce villages and terrains developers would have never imagined.

Background

The initial research for this project came primarily from *Procedural Content Generation in Games* by Noor Shaker, Julian Togelius and Mark J. Nelson. In Chapter 5, they describe how to use grammars and L-systems in applications to vegetation and levels. The section of this chapter most relevant to this project was on the construction of infinite Mario levels. By using a set of 9 different chunks (Platforms, Hills, Gaps, Koopas, etc.) and a ruleset a level is generated by choosing between the options. After the initial generation, a fitness function is applied over the set of chunks in order to eliminate overlapping chunks and measure the content quality.

While not directly related to the goals of this project, I found the paper *Grammar-based Procedural Content Generation from Designer-provided Difficulty Curves* to be helpful in showing how to manipulate threshold values to achieve my desired results. This paper focuses on allowing game designers to guide the procedural generation process by a set of different difficulty curves in relation to the behaviors of AI.

There have been many other programmers working on procedural generation within Unity. However, many of them are for 2D projects. One of the initial example programs I used to learn about procedural generation was a procedural dungeon generator published by Unity's tutorial team.

A similar project to what I aimed to achieve was created by a team of student called TOWN - Tiny prOcedural World geNerator. They created a terrain, urban planning, building, and music

generator for use in Unity. Our projects main difference is that mine is focused for fantasy game development, whereas there would be useful in present day games. Also, their project was completed on a much older version of Unity and is unusable on any version of Unity from the past year.

Tools

All of the development for this tool and demo was done in Unity 2018.3. Unity is a cross-platform game engine written in C++ and C#. For this project, all scripting is done in C# as Unity provides a large scripting API in C#.

Games in Unity are broken up into Scenes. Each of these Scenes contains Game Objects which are created either with the graphical interface by the developer or by scripts.

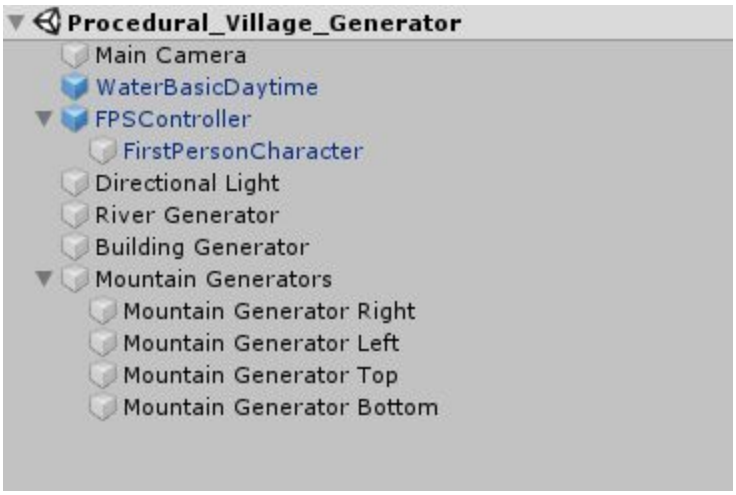


Figure 1: Demo Scene Structure

Figure 1 shows the hierarchical structure of the demo scene provided to playtesters. Every Game Object has its own location in the scene and associated scripts which run on different Engine calls. The main Engine functions used are Start() and Update(). Start() runs once when the game is loaded, and Update() runs every frame. The WaterBasicDaytime and FPSController objects have a blue cube next to them indicating that they are Prefabs. Prefabs are prefabricated objects which either the developer has created for use in multiple scenes, or as in this case, assets acquired from a package.

Additionally, I used the M.E. Town Creator LITE package from the Unity Assets Store. This package contains several prebuilt houses, as well as all of the components which make up the house. An example of one of these buildings and the parts that make up the building can be seen below in Figure 2.

Figure 2: M.E. House 1 Prefab

System Design

The overall system is divided into 3 main generator objects. The River, Building, and Mountain Generators can be moved to any location within a Unity Scene and it will spawn the respective objects at the location where the generator is placed. However, the river generator and building generator must be placed at the same location in order to prevent building from spawning inside the river. However, if the user does not wish to have the river and village near each other, then there will be no issues with placing them in different locations.

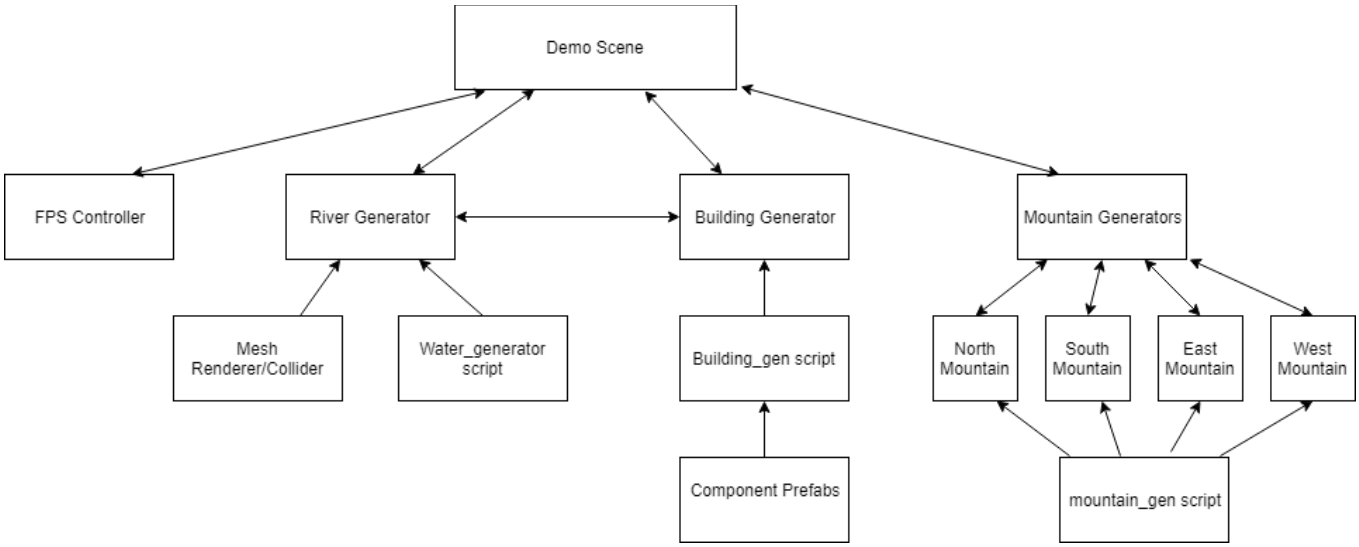


Figure 3: High Level Objects Flowchart

The algorithm used for generating mountains starts by defining a rectangular grid. Each grid location is procedurally assigned a height value. The basis of the height value is first determined by the Mathf.PerlinNoise function. Perlin noise is a psuedo-random pattern of float values generated across a 2D plane. These values gradually increase and decrease making them a simple way to generate terrain. Then that value is multiplied by a constant to increase its max height. Then it is multiplied by a random value between 0.5 and 1.0 in order to make each generation unique. In order to not make the mountain extremely pointed it is then multiplied by the average height of its neighbors and a location value that lowers the values of the edges and increases the values of the more central vertices. Additionally, if the mountains are on the edge of the grid the value is set to 0 to guarantee the mountains do not float. Once the heights have been determined, six points are assigned to each vertice to create two triangles which connect to form a mountainous shape. Finally, it loops through the resulting vertice set and draws the mesh.

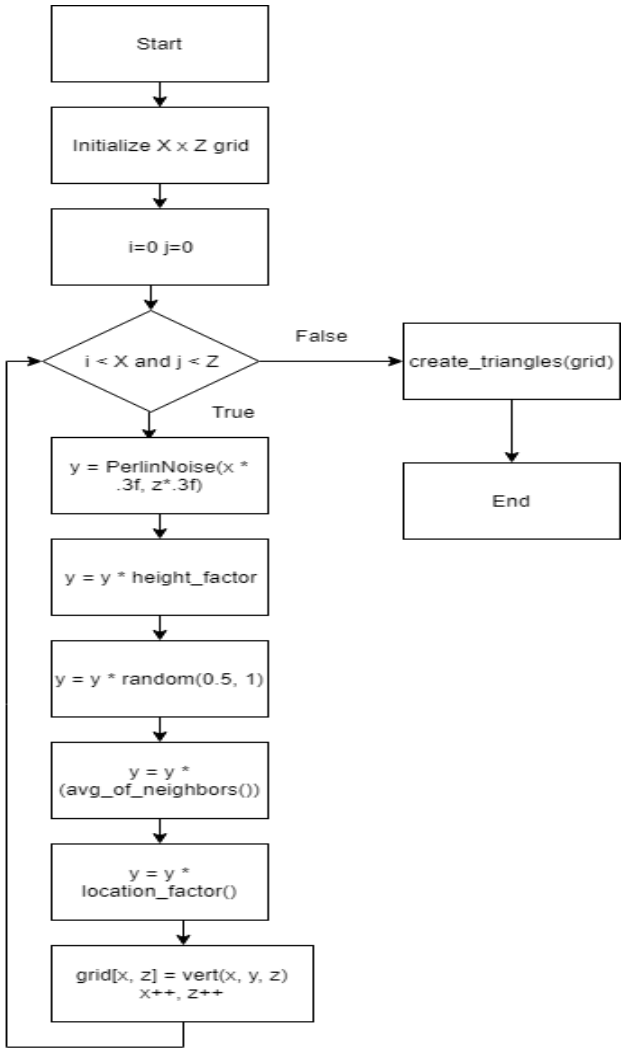


Figure 4: Height Selection Algorithm for Mountain Generator

The main difficulties in creating this generator were determining how to weight the different constants and optimizing the number of vertices for performance and visual appeal. Originally I had hoped to use a 1000x200 grid for the starting vertices. However, after testing on a system with a gtx 770 gpu and 4GB of RAM, the initial loading of the mountains took around 5-10 seconds. By rescaling to a 100x20 grid, loading of the mountains was instant and the visual appeal of the mountains did not suffer too greatly. However, users of this generator can easily modify these values as they are public variables.

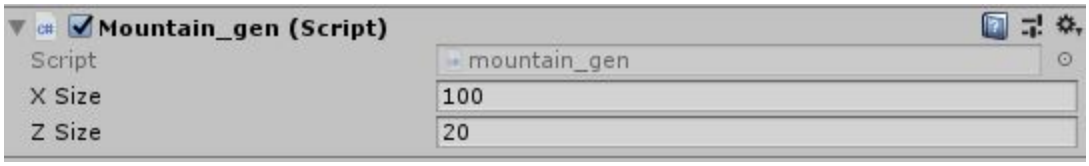


Figure 5: User interface for mountain generator script

The first part of the village generator is deciding the locations of houses. The user can select the max number of houses, number of grid spots, and the size of each spot (the user will need to make sure the size of each spot is big enough for their assets as there is no check for this). The script also takes in the prefabs of either complete houses to spawn, or components to build houses so that each house is unique. The script does check that the prefabs exist before using them so the user can remove or add as many assets as they desire.

Figure 6: User interface for building generator script

In order to choose the location of houses, the algorithm first randomly selects a starting point within the grid. In this first spot it assigns the H1 house which is representative of a town inn.

Then the next grid location is chosen in one of three directions, one direction is blocked so that the village seems to grow out from the town center. Every time a location and building style is chosen, an array of weights is created. In the example using 5 building styles, an array of 5 elements represents the weights of type each house. These weights are used so that houses of similar styles tend to be near each other representing neighborhoods or similar economic status. Also, the rotation of each building is decided by looking at neighbors rotation and attempting to have houses face one another by adding or subtracting 90 degrees.

Originally, the goal of the project was to build these houses from simple components. However, I ran into issues in doing this with the assets I could acquire. Every wall, floor, door, and stairs was sized entirely differently. In Unity there is no easy way to change all prefabs to a universal size. You would need to manually modify the scale of each object which is incredibly tedious and also would distort the texture. A better solution would be to create your own assets with modularity in mind. The aspect of this goal achieved in this project was roof placement. When a building of style 3 is chosen, it will then choose 1 of 4 roofs.

An interesting challenge was figuring out how to prevent houses from spawning inside the river. The scripts for these two generators were kept separate so that users could use both independently or in tangent. The river generator is similar to mountain in that it is modifying a mesh. The scene contains a water object at height -0.1. When tiles are selected as river tiles, the height of vertices on those tiles drops to -1 by default. Since the plane is much larger than the village generator grid, the tiles of these two generators are entirely independent. Therefore, to create communication between the two generators I used two static variables to keep track of the maximum and minimum x positions occupied by the rivers. These will always work as long as the river generator and building generator objects are placed at the same location.

```
if ((transform.position.x + i / 10 + i % 10 *grid_width < (water_generator.River_Min * 10) - buffer) |
    ((transform.position.x + i / 10 + i % 10 *grid_width) > (water_generator.River_Max * 10) + buffer))
```

Figure 7: Building_generator code snippet for river avoidance

Results and Evaluation

In order to test this tool, I had 7 peers playtest a demo scene. They were asked to generate at least 3 villages and spend 2-3 minutes walking through each generation. In the demo, players can spawn as many villages as they like by pressing 'G' which will delete the previous village. When the game starts the river and mountain are generated. The river for the demo was modified so that it would always start at the center of the map, causing a split in the village. After playtesting, they were then asked to complete a short survey.

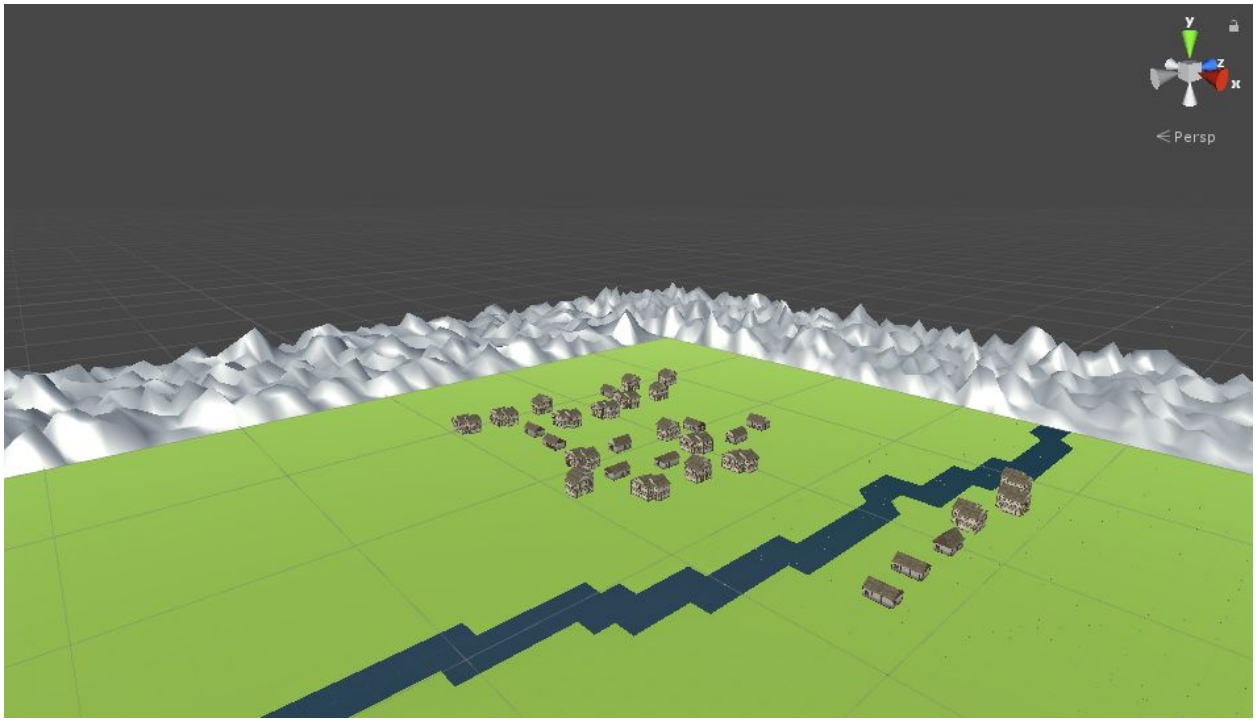


Figure 8: Village Created in Playtest Demo

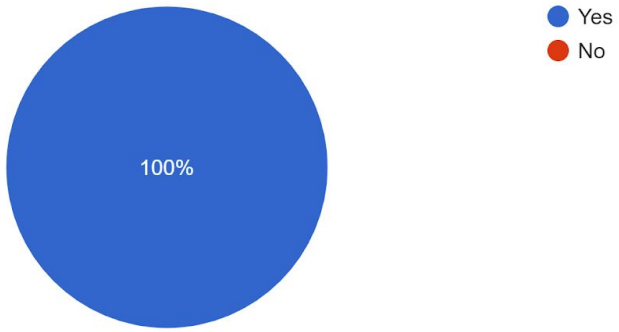
Survey Results

The first few questions were aimed to get an idea of the type of player testing the demo.

From this response it is clear that everyone who took this survey spends much of their week playing video games. Therefore, the results will be from a dedicated gamer perspective.

Have you played any 3rd or 1st person Fantasy RPG/Adventure games? (Examples: Elder Scrolls Series, Fable, Gothic, Zelda, etc.)

6 responses



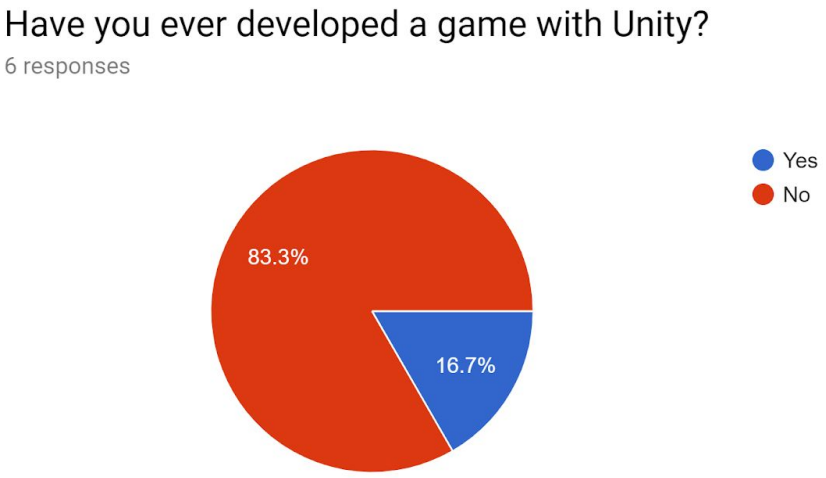
Every person surveyed had at least some experience playing games in the genre this project aims to adhere to.

If yes to previous question, please list the 3rd or 1st person Fantansy RPG/Adventure games you have played.

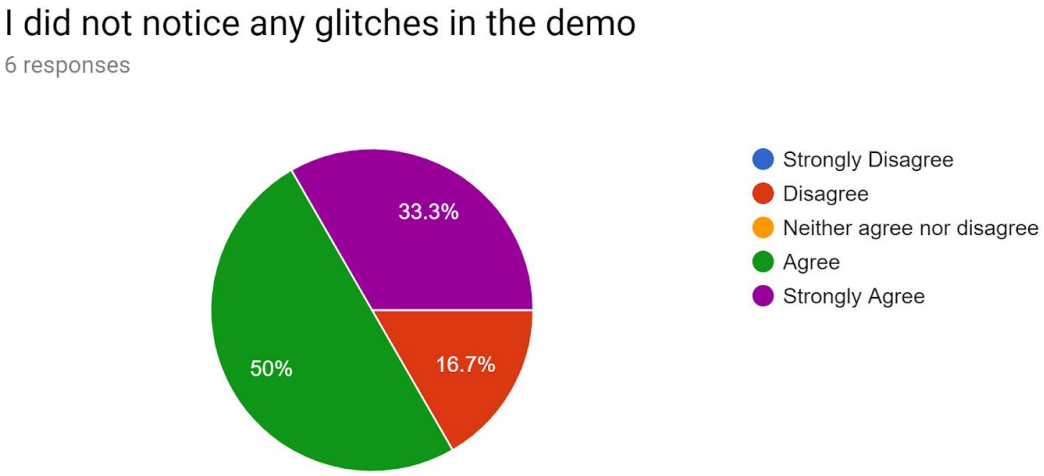
6 responses

Zelda, Fable, World of Warcraft, Elder Scrolls
Borderlands, TESO Fable, Most Zelda's
Elder Scrolls, The Witcher, and Dragon Age
Minecraft, Zelda, TESO
elder scrolls, zelda, fallout, minecraft
Skyrim, Zelda, Witcher, Dark Souls

The testers have all played a wide variety of games I had hoped this village generator would seem to mimic. However, for the series such as Fable, Zelda, and TES this tool coincides with more of their past titles rather than those released in the last 5 years. Despite our audience being composed of all dedicated gamers, none of them listed an indie game rpg/adventure game. This shows that there is either a lack of their interest in indie game rpg/adventure games, or that there is in fact a significant shortage in small team development of those titles.

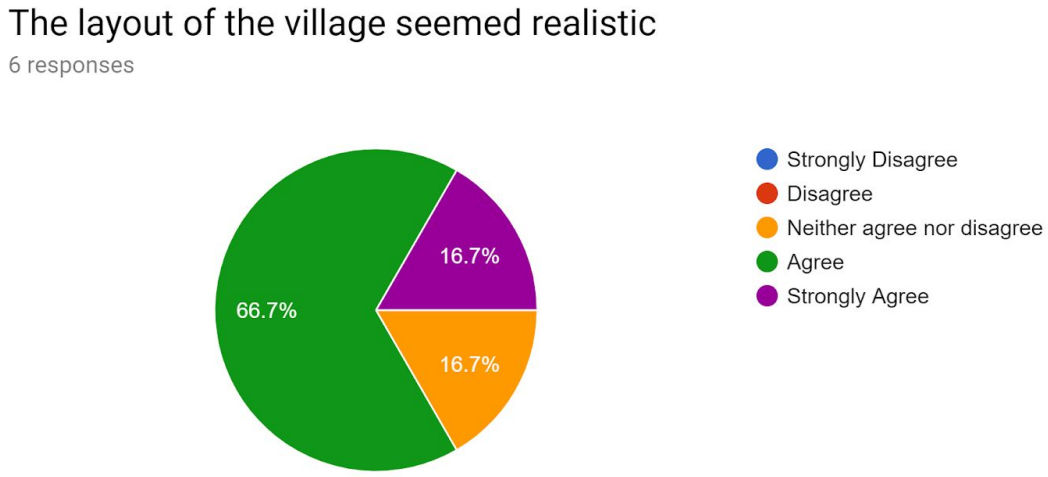


This playtesting was focused on having players rather than developers tests. I wanted to get the opinion of people who would be playing games developed by the tool, rather than only developers. However, it is nice to get the opinion of at least a few developers.

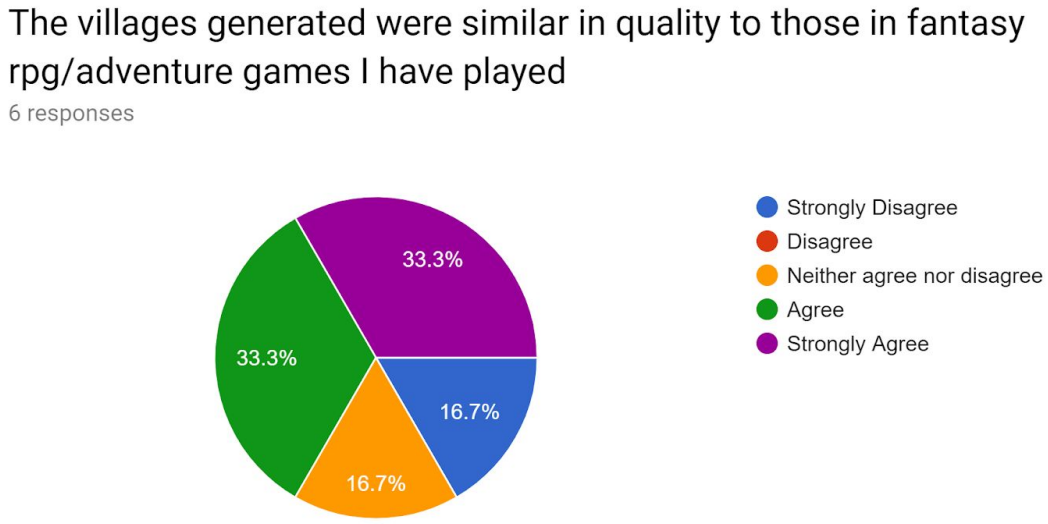


Only 1 person noticed a significant glitch in the demo. However, as will be seen in future responses, all glitches noticed were due to either the Unity Player configuration or assets that are not part of this project's tool, but were added to make the demo more playable.

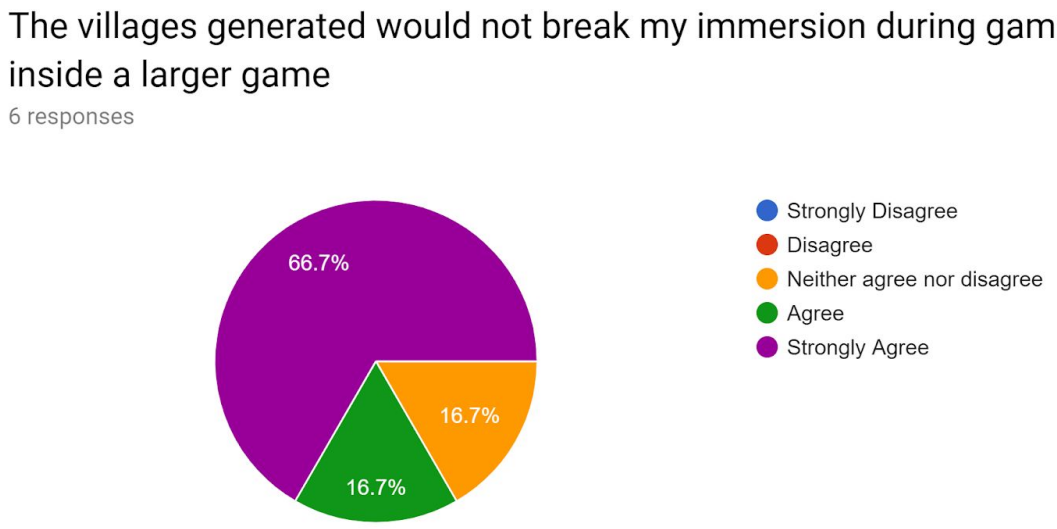
The following questions were focused on the player's experience with the village generator.



Most people were very positive that the village layout seemed realistic which is a huge confirmation that this tool was successful. While the initial plan of this project was the building of unique houses, the main focus instead turned into achieving the realistic layout and this confirms that part was well received.



Considering the massive planning and development time spent on modern AAA RPG/adventure games, I did not expect to receive a positive response to this question. Surprisingly, it was mixed. This shows that there are some games where this tool would be useful, while others where manual placement would be necessary.

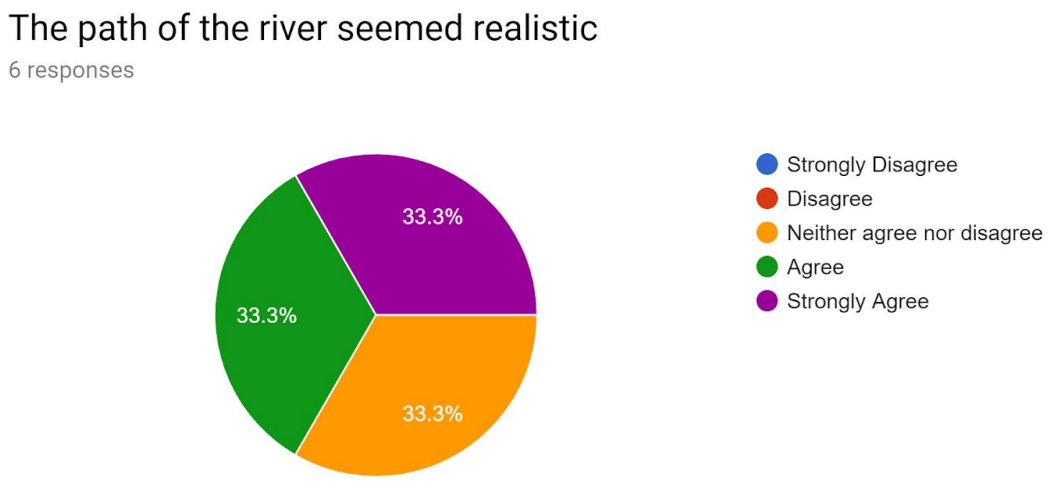


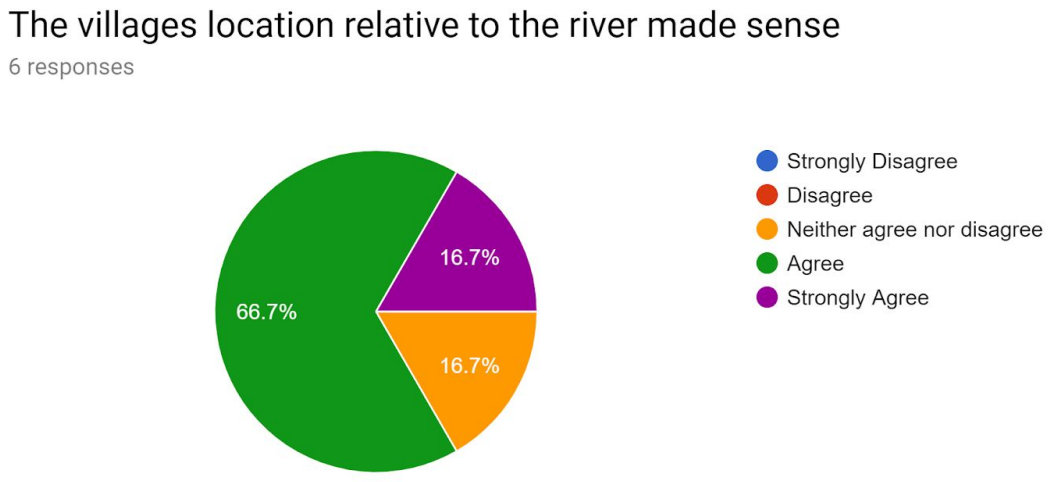
Achieving complete realism or complete similarity to current RPG/Adventure game standards would be a near impossible goal. However, immersion was the baseline goal I hoped to achieve. The responses shows that the tool does achieve this goal.

The free responses to the village varied quite a bit. Some were very positive, while others had some very good constructive criticism. It seems that one person had trouble getting the spawning to work. I do not know if that was a glitch with the tool, the Unity player, or user-error. Others had trouble with moving around the village with the Unity FPS controller. I looked into this more and

when I distributed the demo via installer, the installer did not always correctly assign buttons. This would cause some people to not be able to jump or look up and down. However, stairs should work even without jumping. Having more functional stairs would definitely be a high priority for future development.

The next two questions focused on the river generator.





The river was generated on a flat surface which makes having a realistic layout difficult. Luckily, people were not overtly negative to the path that it took. More importantly, people were happy with how the river and village interacted with one another.

What was your overall impression of the river?

Wet

Wet, Blue

water looked okay but when walking in the walker the sounds of the footsteps didnt change at all.

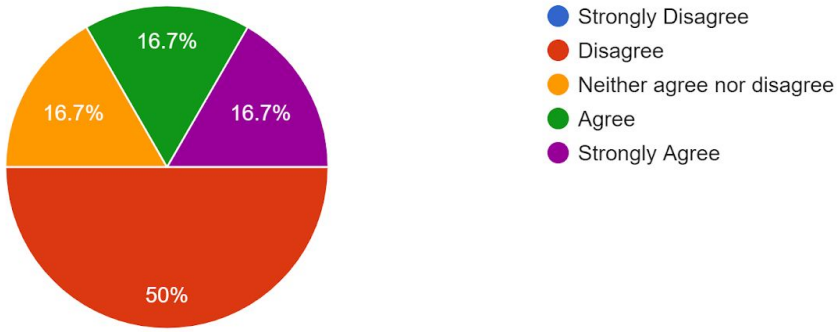
Blocky, the small bends feel random. no depth

One issue with the demo was that the water was very shallow and there was no change in sound when the player stepped into the water. This can be taken care of by changing the height value on the river generator, and the sound issue can be handled by modifying the FPS controller.

The next questions regarded the mountain generator.

The terrain of the background mountains seemed realistic

6 responses



As described in the System Design section, there were performance issues limiting the quality of the mountains which is reflected in the responses to this question. To fix this, either a more efficient algorithm for terrain generation or a rendering technique could be used. Also, the texture of the mountain was relatively simple. A good solution would be a procedural terrain mesh.

What was your overall impression of the mountains?

Fun to jump off of

Rocky, Tall

White, Tall
they didnt look good
Fine for the purpose. seemed basic

Overall people weren't too disappointed with the mountains. However, it is clear they are most likely the main candidate for improvement in this tool. In the current state, they are a decent background, but not a playable area.

What games, if any, could this village be included into without breaking immersion?

4 responses

any fantasy/medieval game
Minecraft
elder scrolls :)
Minecraft etc. games that already have a procedural vibe, would feel out of place in a fully planned game.

Conclusion

Overall, this project was successful in providing a basic procedural village, river, and mountain generator which can be used in the development of 3D RPG/Adventure games. There is much room for improvement to make it more useful in larger projects. Primarily, the generation of individual building so that each are unique is the main goal that this project was not able to achieve. Also, the mountain generator could use performance optimization to allow for better quality. The aspects where the river generator fell flat in playtesting are responsibilities of the developer, not the tool. However, it could be an extra feature of the tool to further ease constraints on the developer. Also, this project shows that procedural generation can be extremely useful and provide interesting results when implemented in a 3D environment. While development in 3D is much harder than 2D, 3D gaming has been the standard of the 21st century and speeding up development in this area will greatly improve the quantity and quality of the indie games of the future.