Computer Science Extended Essay

Comparison of Procedural Content Generation Algorithms in 2D Terrain Generation

Research Question: How do cellular automata and Perlin noise compare in terms of performance, diversity, and naturality for generating 2D game terrains, specifically caves and islands?

Word Count: 4000

Table of Contents

Introduction

In the fast-paced generation of media production, the video game industry stands out the most with a revenue growth of (in billion USD) 121.50 in 2017 to 294.60 in 2022 [Statista]. Yet, in a world where artificial intelligence and automatically generated content have started to become popular, video games' virtual worlds are mostly handcrafted. For example, in the development of the game Uncharted 4 by the studio Naughty Dog, the "employees worked endless hours, staying at the office as late as 2:00 or 3:00 a.m" [Schreier 31], trying to craft the perfect world for their adventure game. This current approach to video games, though it produces special and unique experiences for gamers, is not capable of supplying the ever-changing demands of modern entertainment. Also, it creates a financial burden on game developers as "one of the main costs of developing a video game is content creation" [Barria 2]. To favor the demand, publishers try to release games earlier than suggested, resulting in games full of bugs[1].

A solution to this problem is procedural content generation (PCG) which is "the automation of media production" [Barria 1]. It is a technique that was used by famous "games like Minecraft, Spelunky, and No Man's Sky" [Schatzeder]. PCG enables an innovative approach to world creation in video games, utilizing algorithms to generate unique worlds for every playthrough. While PCG could be used in games to create "textures, sound effects (...) quests or even game mechanics" [Barria 1], this paper only considers top-down 2D map/terrain generation, specifically caves and islands, to limit the experiment to an approachable one.

[1] "a programming error in the game" [Delgado]

This paper presents and evaluates the difference between PCG algorithms in terms of performance, diversity, and authenticity in terrain creation (explained in “Methodology”). While there are a lot of PCG algorithms and techniques to choose from, the ones that are considered to be evaluated in this essay are Perlin noise and cellular automata as they are the ones that stand out the most [Schatzeder]. They are both evaluated within their own parameters and against each other. The main objective of this research is (a) to identify the more efficient algorithm in terms of generation speed and (b) to identify the more diverse, applicable, and natural-looking generations. The conclusion drawn from this research would inform developers about which algorithm to choose in varying situations.

The paper first introduces the focus and aim of the essay, while providing a brief explanation of the topic. Then, provides detailed information on games that use PCG, experimented terrain types, and the algorithms that are used. Also, introduces the methodology of the experiment, evaluating the data found through the experiment. Lastly, it reflects on the results and the main process, identifying the limitations of the experiment and suggesting further research.

Literature Review

Game Review - Minecraft

Minecraft is the "best-selling video game in history. (...) In Minecraft, players explore a blocky, pixelated, procedurally generated, three-dimensional world with virtually infinite terrain." [Wikipedia]. PCG has enabled Minecraft to give the player a unique experience for each run. The sandbox game using Perlin Noise[2] creates vast terrains with different biomes spanning over the environment.

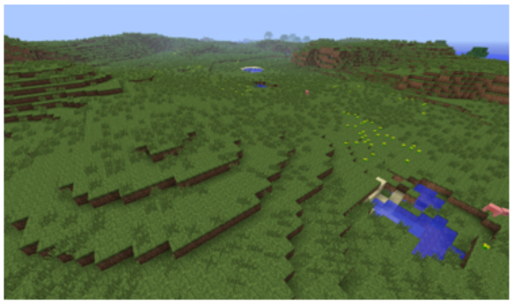


Image 1. Minecraft [Minecraft Help Center]

Game Review - Terraria

"Terraria is a 2D sandbox game that (...) revolves around exploration" and "starts in a procedurally generated world" [Wikipedia] both generating non-player characters and maps. Again PCG enables the player to have a unique experience every run. The terrains spanning from jungles to mountains to dark caves are generated using fractal terrain generation.

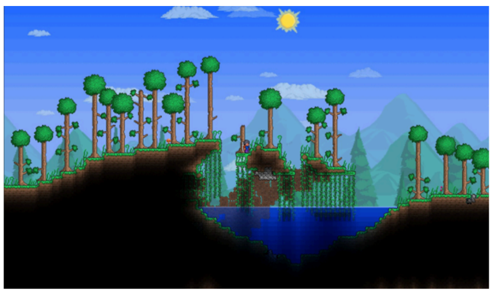


Image 2. Terraria [Alextramblog]

[2] Later in development, Minecraft added more complex algorithms other than Perlin Noise [Zucconi]

Terrain Description[3] - Caves

Caves are enclosed, dark spaces that push the player to explore an unknown and scary environment. They are complex structures with branching pathways, wide chambers, and dead-ends. Some areas might contain chests and other areas may surprise the player with an enemy. Thus, caves should not be too complex with small, inaccessible chambers, or too large with fewer areas to explore. Also, a chunk of a cave would not enable the player to use digging skills.

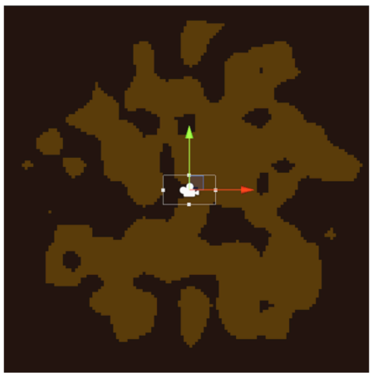


Image 3. Cave Terrain [Jonas Re]

Terrain Description - Islands

Island terrains consist of multiple islands in the ocean connecting to each other through a seaway. Island terrains should not be created as a single island and also should not be really small structures, so the designer can put missions and items inside the generated map. Shortly, islands should be scattered all over the sea but not extremely small.

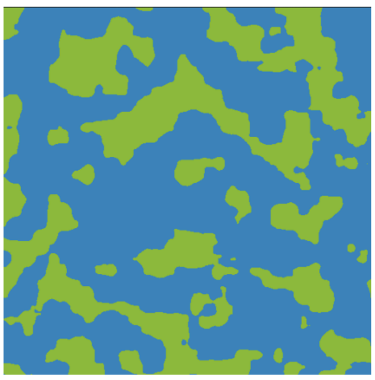


Image 4. Island Terrain [Emmsii]

[3] These brief explanations of the terrains were written from the analysis of Minecraft, Terraria, and their cave/island terrains.

Algorithms - Cellular Automata (CA)

CA’s debut was in the 1940s by the mathematician John Von Neumann. Later, it was popularized by John Conway’s “Game of Life”.

How does CA work?

Essentially, CA is a model which consists of a grid of cells. Each of these cells has an initial state according to a ruleset determined by the creator. Most common states are 0 and 1 but they can be replaced with a finite number of states. Continuing, every square in the grid is updated based on a **list of rules** (based on their and neighbors’ states) by the amount of a predetermined **iteration number**.

CA’s Parameters and Their Effects

The algorithm used in the experiment is taken from YouTube [Sebastian Lague].

There is an initial function named RandomFillMap that gives the value of 1 (wall) to the edges in the map. This function also through a **randomFillPercent parameter** determined before and a value generated between 0 and 100 for each square determines the initial state of blocks when the map is first generated. If the value is below the randomFillPercent the block has the value 1 (wall), else it has the value 0 (ground). The other parameter is the iteration amount (**randomFillMapInt**). For this amount, all squares in the map are checked one by one, applying this ruleset:

- If there are more than 4 neighboring wall tiles, the cell becomes a wall.
- If there are fewer than 4 neighboring wall tiles, the cell becomes ground.

Algorithms - Perlin Noise (PN)

PN was first introduced by Ken Perlin (1983) and is a gradient noise function that establishes itself in the computer world because of the natural visuals it generates.

How does PN work?

Imagine a grid filled with squares inside. This grid represents a 2D space in which the function operates. Every square has 4 corners that are used as points.

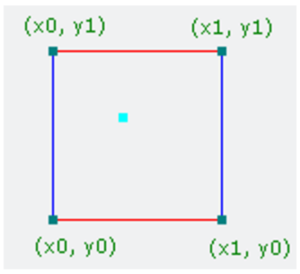


Image 5. PN Step 1 [Adrian]

For each of these corners, there's a **gradient vector** assigned. These vectors are not completely picked at random. They originate from a predetermined set of rules designed to emulate organic and natural patterns. Specifically, they point in one of 8 possible directions like the compass's cardinal and diagonal directions.

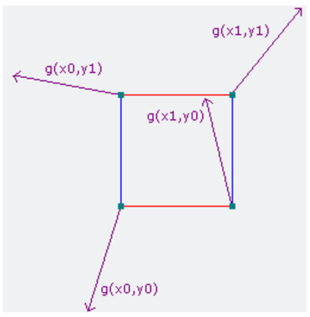


Image 6. PN Step 2 [Adrian]

Next, consider an **input point** (blue point). This point is randomly placed within each square in the grid. For every input point located within a square, **distance vectors** are drawn from the edges of the square to the input point.

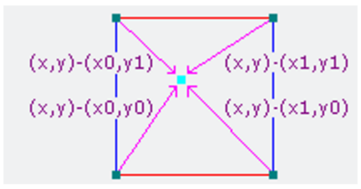


Image 7. PN Step 3 [Adrian]

Then, the dot product of the gradient and the distance vector is taken. The dot product equals the magnitude of both vectors multiplied by the angle between them, leading to either negative or positive values for the final vectors. If the direction of the final vector is the same as the gradient vector the dot product is positive, while an opposite direction makes the dot product negative. This relationship creates a heat map by changing the values of the dot products.

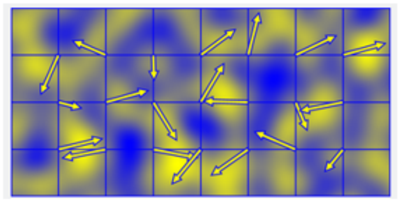


Image 8. PN Step 4 [Adrian]

The last step is called **interpolation**. While it is more computationally demanding, it prevents the map from looking linear and unnatural. The fade function (also called the ease curve) ensures that the transition is smoother as the curve gets closer to an edge. It plays a pivotal role in differentiating Perlin noise from pure random noise, offering the continuous and smooth noise pattern that is desired.

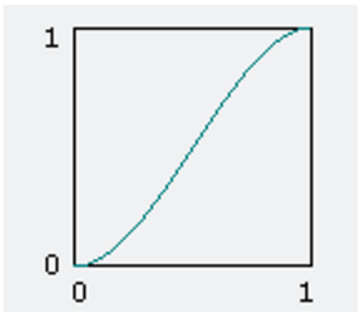


Image 9. PN Step 5 [Adrian]

PN’s Parameters and Their Effects

The algorithm used in the experiment is taken from YouTube [Sebastian Lague]

There are 3 main parameters in PN:

1. **Octaves**: In Perlin noise, "octaves" resemble the number of noise layers that are combined. They come together like layers to produce a more complex noise pattern. More octaves offer more information, but they also put greater strain on the computer. Mostly, octaves are used increasingly to add details to maps.
2. **Lacunarity**: Each succeeding octave’s frequency is determined by lacunarity. Each succeeding octave has a higher frequency if the lacunarity is larger than 1, introducing more detail across a smaller area. As a result, the appearance is "rougher". The first noise pattern (octave) has a frequency equal to $0 \times persistence\ value$ frequency, the second has a frequency equal to $1 \times persistence\ value$ and so on.
3. **Persistence**: Persistence controls the amplitude of each octave. It decides how much each noise layer in octaves contributes to the output. Low persistence indicates a smoother pattern. It becomes rougher and more jagged with high persistence as the influence of other layers increases. The persistence value is between 0 and 1 and is calculated the same as lacunarity.

Methodology

The experiment is done in **Unity 2D Game Engine** on a laptop with **specs**: Intel(R) Core(TM) i7-9750H CPU @2.60GHz, 32GB RAM, NVIDIA GeForce GTX 1650.

Using both CA and PN, 100x100 squared maps will be generated. These maps will be analyzed in the fields of performance, naturality, and diversity and will be using only 2 colors:

- Grey (ground area) and brown (wall area) for cave terrains
- Blue (sea area) and green (land area) for island terrains.[4]

Hypothesis

- In terms of overall performance, PN will be a better choice because it does not depend on iterations like CA.
- In terms of diversity (for generating optimal environments according to their descriptions), CA will be more suitable because it does not change the shape of the map that much when its parameters change, resulting in a more predictable terrain.
- In terms of naturality, CA will be a better choice for both islands and caves because CA only gets better as there are more iterations, unlike PN which can go wrong if some parameters are not selected correctly.

[4] Some example maps are given in **Appendix**

Performance

This criterion refers to the computational efficiency and is measured through runtime. Only the runtime before the frontend map generation is measured to create a fair experiment. Other than comparing these algorithms with each other, the effects of their parameters on their own speeds are also measured. This is done by taking averages on the instances where a single parameter stays constant.

- In CA, generation speeds are processed by iterating the parameters: randomFillMapInt (1 to 6, increasing by 1) and randomFillPercent (40% to 60%, increasing by 1%)
- In PN, generation speeds are processed by iterating the parameters: octaves (2 to 5, increasing by 1), lacunarity (1 to 3 increasing by 1), and persistence (0 to 1 increasing by 0.1)

Diversity

The perception of humans of a game/map is as important as the efficiency since games are considered as art. A poll done by 40 people assess this criterion. Thus, only primary data is used for evaluation. The maps generated in each algorithm are surveyed by a linear scale from 1 to 5. 1 being “big, sized single island/cave (full accessibility[5])” and 5 being “small sized, multiple separate islands/caves (low accessibility). Then, the averages of these data are compared with the explanations from the part “Terrain Descriptions” to determine the best option when using these algorithms and to determine which parameters suit which terrain.

[5] The playable/accessible (land/ground) areas are green/grey and the initially inaccessible or swimmable/digable (sea/wall) areas are blue/brown.

- In CA, three maps for each combination are generated by iterating the parameters: randomFillMapInt (1 to 5, increasing by 1) and randomFillPercent (40% to 60%, increasing by 10%).[6]
- In PN, three maps for each combination are generated by iterating the parameters: octaves (2 to 4, increasing by 1), lacunarity (1 to 3, increasing by 1), and persistence (0 to 1, increasing by 0.5).

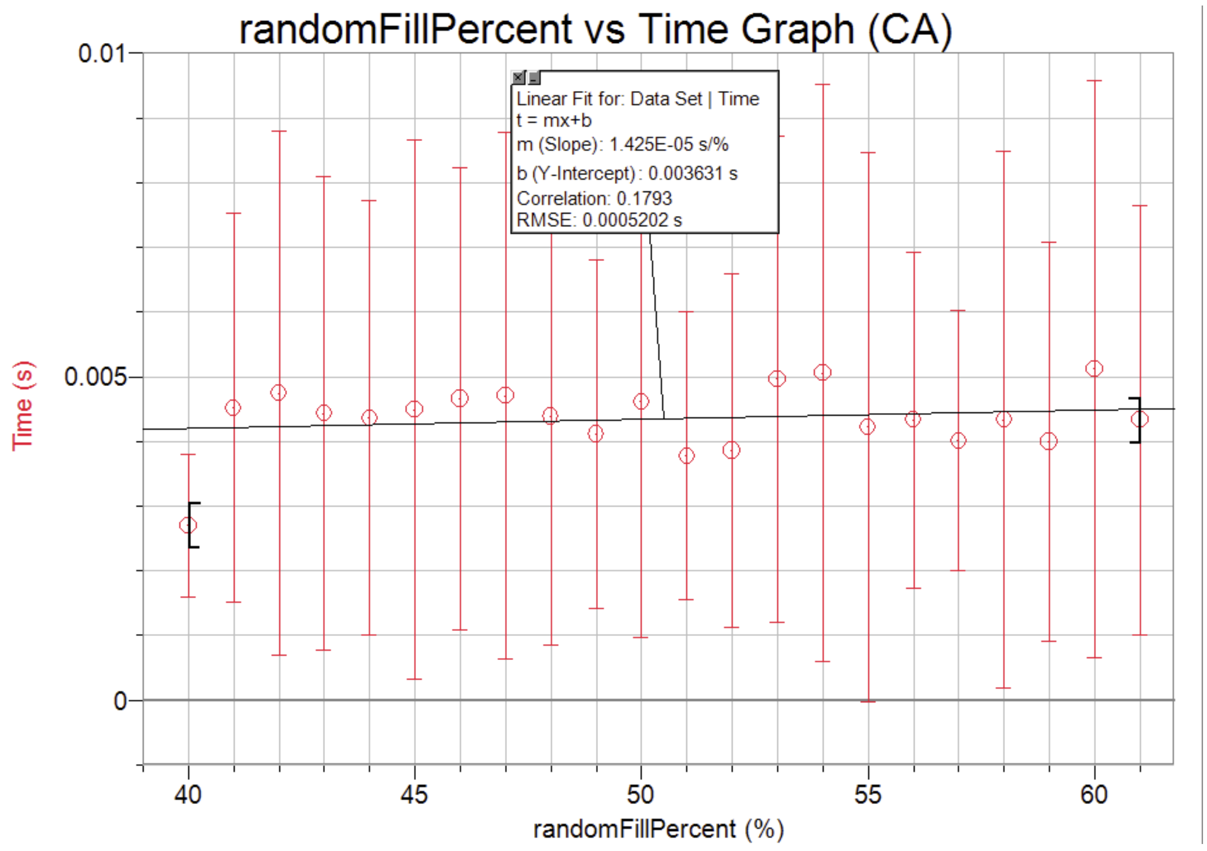
Naturality

Through the same poll, these maps are surveyed again through a linear scale from 1 to 5. 1 being “least natural looking” and 5 being “most natural looking”. The maps are evaluated on how natural and believable they look. This criterion is subjective and aims to find a possible correlation between human perception of the maps, a desired (natural) outcome, and the parameters of algorithms.

[6] The sample size is made to be smaller to create an understandable-sized poll that would not bore people.

Results and Analysis

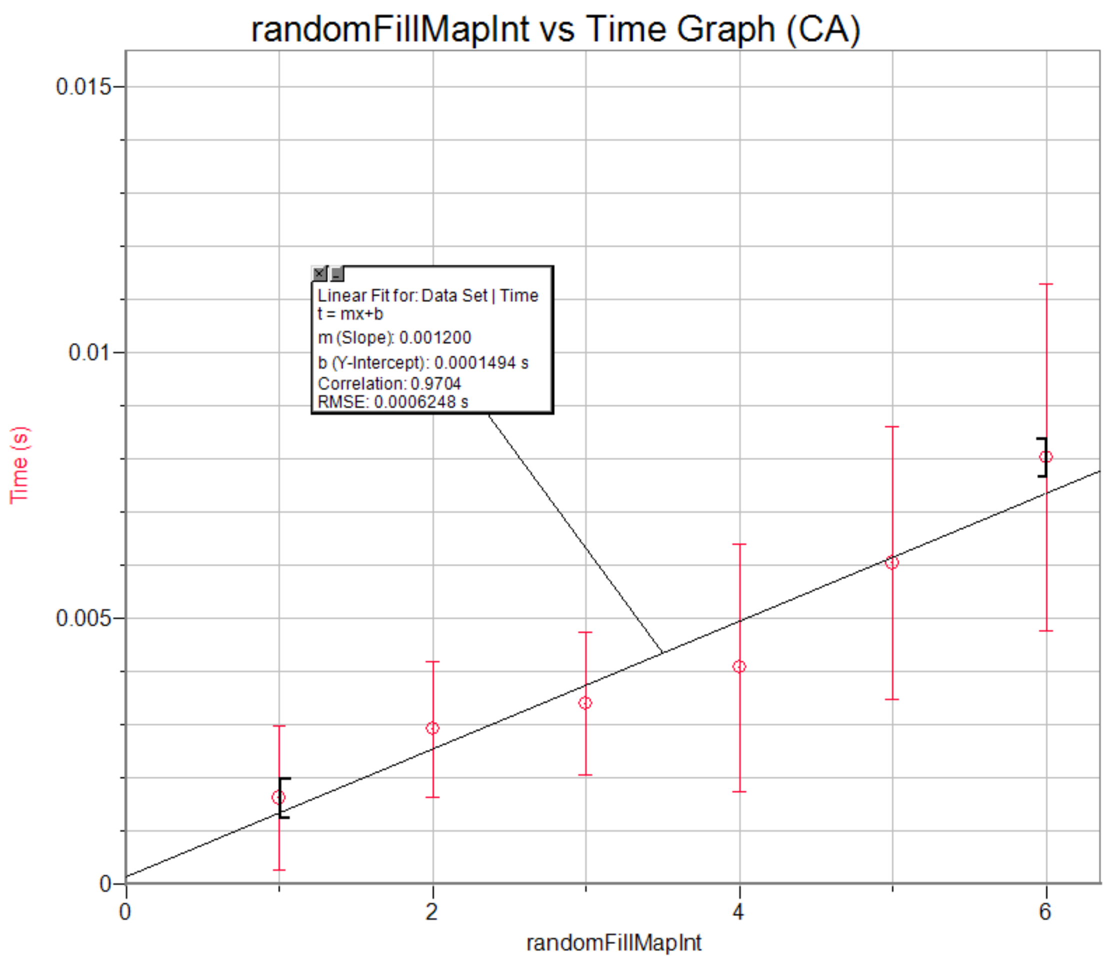
Performance (CA)



Linear Fit for: Data Set | Time
t = mx+b
m (Slope): 1.425E-05 s/%
b (Y-Intercept): 0.003631 s
Correlation: 0.1793
RMSE: 0.0005202 s

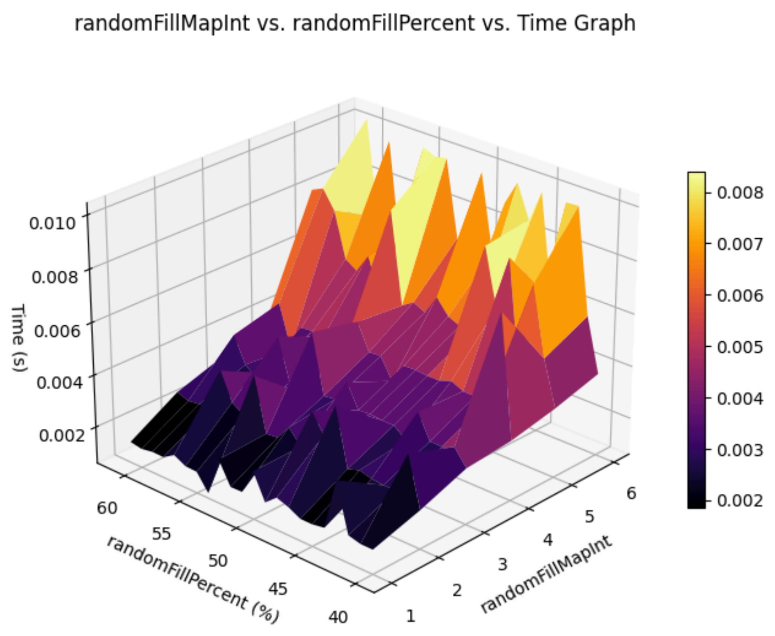
The nearly straight best-fit slope[7], the irrelevant data points on the slope, and the large uncertainty bars suggest that randomFillPercent did not have any effect on the speed of CA's generation. Since randomFillPercent only controls the initial state of the map, this was an expected outcome. Although the cells change according to their initial state, every cell gets checked in every iteration independent of the condition of their neighbors, resulting in no change in speed when changing randomFillPercent.

[7] These graphs were created using Logger Pro App.

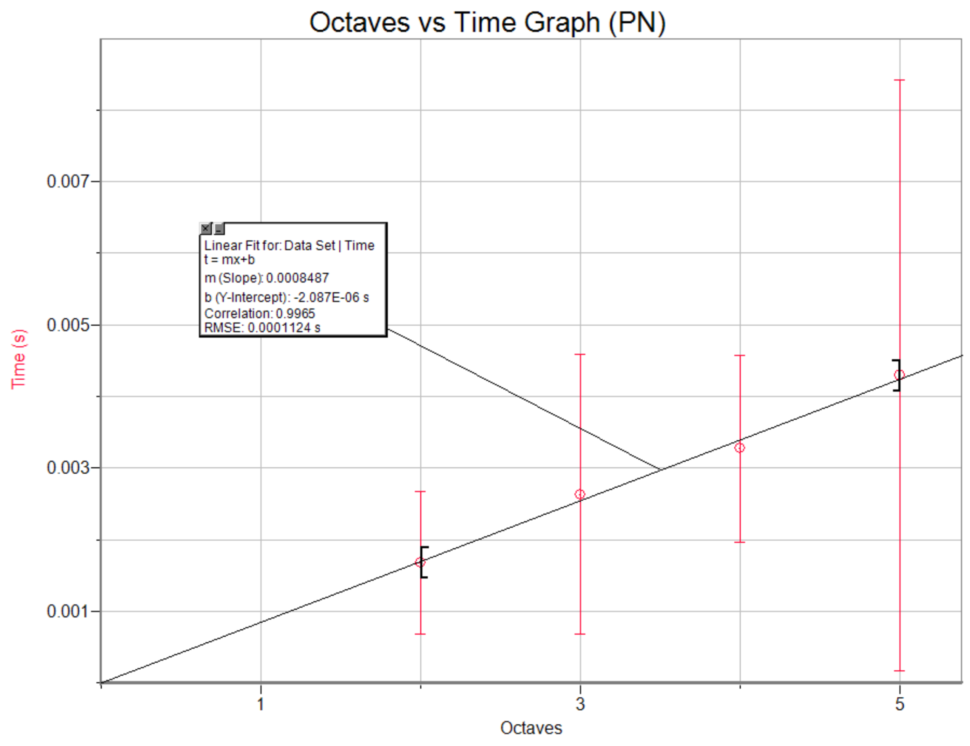


On the contrary, randomFillMapInt had nearly a direct proportion with the performance of the algorithm. The slope of the graph almost passes through the origin and the data points are considerably close to the slope, indicating a great correlation between generation speed and randomFillMapInt. Moreover, the considerably unorganized uncertainty bars indicate some irregularities. Again, this outcome was expected because randomFillMapInt directly affects the time taken to complete the algorithm. Every square in the grid is checked each time the value is increased, increasing computational demand and time taken.

Lastly, here is a 3D graph made using Python, visualizing both variables' effects.

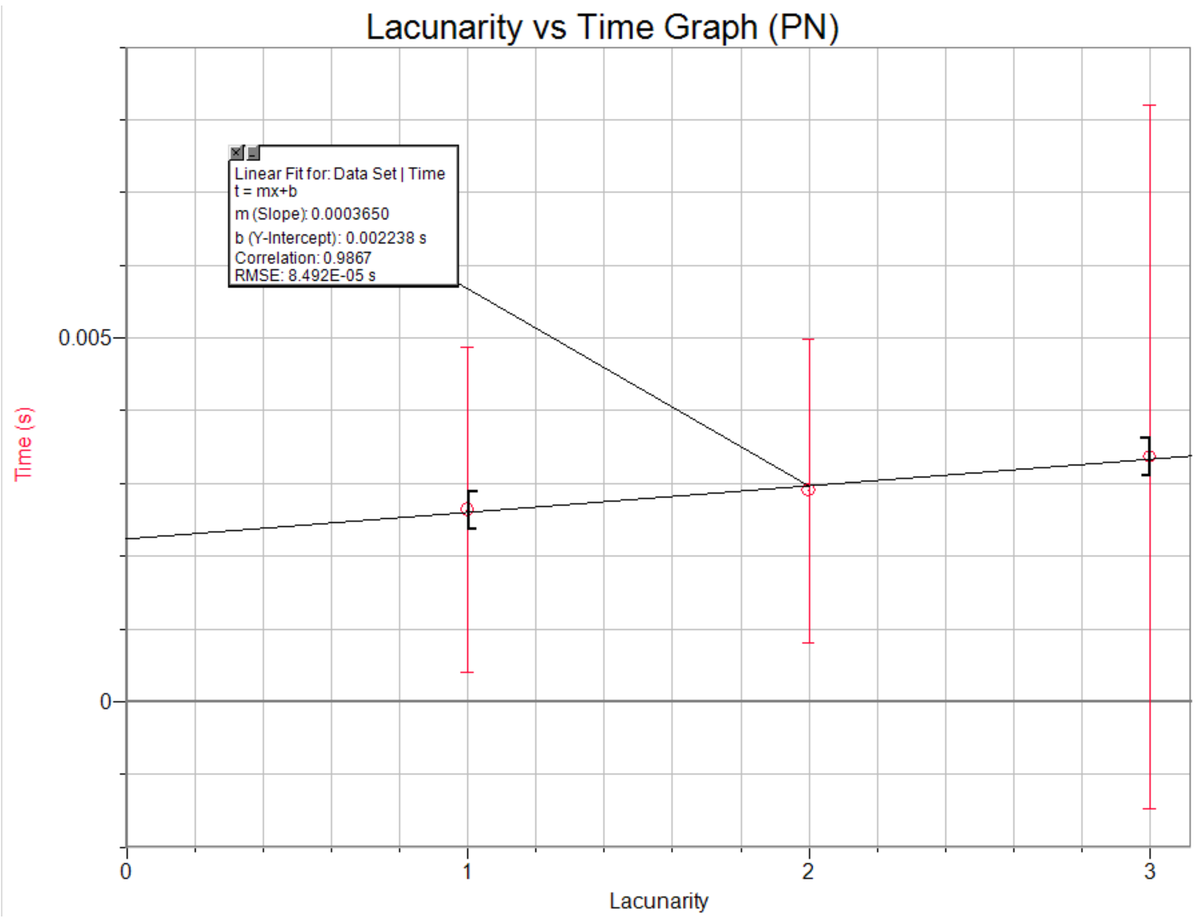


Performance (PN)

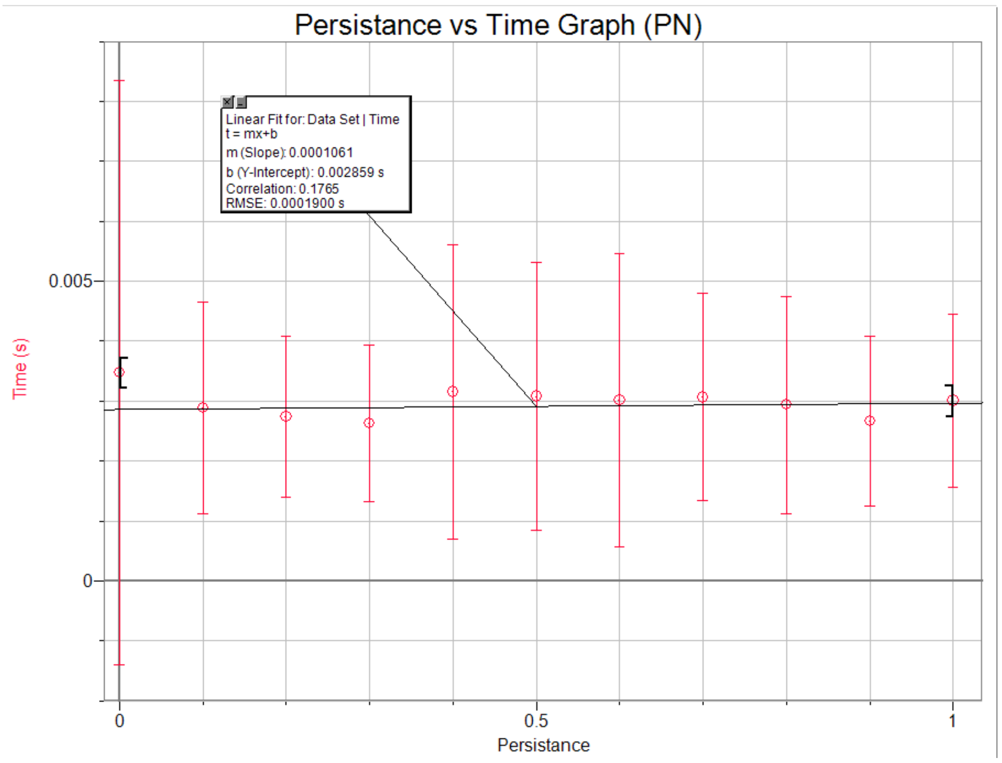


Linear Fit for: Data Set | Time
t = mx+b
m (Slope): 0.0008487
b (Y-Intercept): -2.087E-06 s
Correlation: 0.9965
RMSE: 0.0001124 s

The octave amount in PN directly increases the amount of workload the computer has to complete in order to create a map. The slope perfectly passes through the origin and the data points are all on the slope, indicating direct proportion. Although the uncertainty bars are again quite large this irregularity can be connected to the other parameters having an effect on the outcome of generation speed. Shortly, octaves have a direct correlation with generation speed.

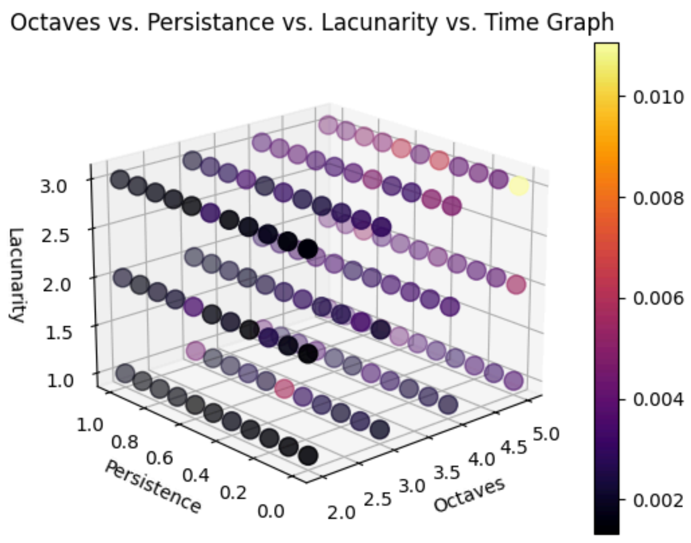


Lacunarity as can be seen from the slightly increasing slope and the suited data points has a smaller effect on the generation speed. Lacunarity changes the frequency of the noise maps generated. As lacunarity increases the frequency of the noise maps increases, leading to more things to calculate by the computer since there are a lot more crests and throughs in the noise maps. On the other hand, the slope does not pass near the origin showing that while lacunarity has an effect on the generation speed, it does not have a linear correlation.



The last parameter persistence, as can be seen from the straight (nearly zero) slope, does not have an effect on the generation speed of the terrain. It does not directly affect the material that needs to be processed by the computer. Persistence only affects the power of octaves on the primary octave but does not change any noise maps' shape.

Lastly, here is a 3D graph, visualizing three of the variables' effects. The color code is about the time spent generating the map.



Performance (Both Algorithms)

To sum up, both algorithms had similar parameters that both affected and did not affect the experiment's outcome. While randomFillPercent in CA and Persistence in PN did not have any effect on the speed of map generation, randomFillMapInt in CA and lacunarity and octaves had an impactful effect on the generation speed.

While these algorithms work differently from each other when generating similar results PN seemed to be much faster compared to CA on the general average as the hypothesis suggested. PN had a general average generation speed of **0.00297 seconds** and CA had a general average generation speed of **0.00435 seconds**. CA nearly took double the amount of time PN took while generating maps. While the near-zero amounts of time may seem unimportant, remember that the experiment was done on a simple 100x100 grid with only 2 colors to work with.

Suppose the scale of a game gets larger (like making it 3D or adding more colors) these generation times would exponentially increase, creating a disadvantage. This result may have been caused by the nature of algorithms. While CA is a more iterative algorithm used for loops within themselves, Perlin Noise is a much more mathematical algorithm that does not change much according to increased parameters. So, PN would have an increasing advantage over CA in terms of performance as the generated maps get larger and more complex. This could have been the reason for Minecraft, a considerably huge game, chose PN when implementing its algorithm. Shortly, the data showcased that PN is a preferable choice in terms of performance.

Diversity (CA)

Caves			
randomFillMapInt	**Average**	**Median**	**Mode**
1	3.25	3.17	3.33
2	3.01	3.00	3.00
3	2.56	2.33	2.00

Islands			
randomFillMapInt	**Average**	**Median**	**Mode**
1	2.64	3.33	3.33
2	3.19	3.00	3.00
3	3.08	3.00	3.00

In all situations the average, median, or mode integer chosen by the survey group gave the value between 2-3, indicating that these maps were neither regarded as having smaller structures and inaccessible nor having a big structure with complete accessibility. While this structure definition is quite suitable for caves[8], it is less suitable for island terrains since they require the generation of more open sea areas. Also, while increasing the randomFillMapInt the value dropped (getting closer to 2) for caves and increased (getting closer to 3) for islands. Thus, increasing the iteration amount is beneficial for both terrains, creating the more desired look according to their descriptions.

[8] Refer to the part "Terrain Descriptions" in Literature Review

Caves			
randomFillPercent	**Average**	**Median**	**Mode**
40	1.96	1.33	1.00
50	3.19	3.17	3.33
60	3.67	4.00	4.00

Islands			
randomFillPercent	**Average**	**Median**	**Mode**
40	1.58	1.00	1.00
50	3.24	3.33	3.33
60	4.09	5.00	5.00

On the other hand, randomFillPercent had a direct effect on the generation of maps. Since the map's initial situation changed in each randomFillpercent value, the maps started turning into smaller structured and less inaccessible maps as the value increased. As the randomFillPercent increased the map's grid filled with ground or land areas decreased, creating fewer opportunities for the ground areas to grow as the algorithm iterated over itself. It could be said that the value between 40-50 is more suitable for cave generations and the value between 50-60 is more beneficial for island generation. Also, compared to the effects of randomFillInt, randomFillPercent was much more impactful in creating the desired terrain.

Diversity (PN)

Caves			
Octaves	**Average**	**Median**	**Mode**
2	2.99	3.00	3.00
3	2.97	3.00	3.00
4	3.08	3.22	3.11

Islands			
Octaves	**Average**	**Median**	**Mode**
2	2.53	2.78	2.78
3	2.62	2.56	2.33
4	3.03	3.06	2.89

Increasing the octaves only had a slight effect on the terrain generation. While in islands the average increased by .5, in cave terrains it had almost no effect. According to the description of the terrains increasing the octave amount seemed to contribute to island terrain generation but did not have any effect on creating better cave terrains. This is understandable because octaves only increase the level of detail in terrains. Octaves do not shape the overall look of the maps, resulting in only a small effect on the diversification of the terrains.

Caves			
Lacunarity	**Average**	**Median**	**Mode**
2	2.55	2.44	2.44
3	3.24	3.39	3.44
4	3.26	3.39	3.22

Islands			
Lacunarity	**Average**	**Median**	**Mode**
2	2.76	2.83	2.78
3	2.92	3.11	2.89
4	2.51	2.44	2.33

Lacunarity showed a different trend than other parameter. Rather than increasing/decreasing from one end to the other, it had an increase then decrease in the scale, especially in island terrains. For example, the average went from 2.76 to 2.92 then steeply decreased to 2.51. Lacunarity determines the frequency of octaves, increasing the influence of smaller details gradually as it increases. So, the expected result was the map having smaller structures but it did not occur. The maps got more unified as the lacunarity increased from 3 to 4. This was caused by the increasing level of detail so much that every pixel in the maps seemed like connecting to each other. Shortly, the best value of terrain generation for both terrain types is under the value 4.

Caves			
Persistence	**Average**	**Median**	**Mode**
2	2.64	2.67	2.56
3	2.94	2.89	2.89
4	3.46	3.67	3.67

Islands			
Persistence	**Average**	**Median**	**Mode**
2	2.14	2.17	1.89
3	3.00	3.22	3.33
4	3.04	3.00	2.78

While persistence increases, it only affects the level of influence of details and does not affect the primary octave. So, it would be expected that the maps turned into more and more inaccessible terrains as this value increased. While this hypothesis was true in terms of cave terrains, it was not exactly true for island terrains. In cave terrains, averages, medians, and modes all increased as the persistence increased. On the other hand, in island terrains, the increasing trend from persistence 2 to 3 stopped when changing when going into 4. The trend of the maps getting jagged as persistence increased was similar to the one in lacunarity. Both of these parameters affect the influence of details in maps, so they are most likely to increase inaccessibility in maps to a certain extent. Nevertheless, in both, after some point, the maps start to look more unified because of the excess amount of detail in the maps. It is not optimal to increase the persistence value much when creating game terrains.

Diversity (Both Algorithms)

The results indicated that the average values for CA and PN respectively were **2.97** and **2.73** for islands; and **2.94** and **3.02** for caves. While CA was better (as more suitable) for islands (with 0.24), there was almost no difference in caves. Looking back at the hypothesis suggesting that CA would be better in both terrains, it can be said that it was wrong. The overall performance in terrain generation did not have a significant difference in both of the terrain generations. Rather than both algorithms being compared to each other, the results have shown that they should be only compared within their own parameters when trying to find optimal generation.

Naturality

In contrast to the other two evaluation fields, naturality is a subjective evaluation criterion. So, the averages did not change much (mostly between 2.5 and 3.5) in most fields within each parameter. Again, the hypothesis was wrong, highlighting that the outlook on this topic should be more about specific parameters rather than comparative. To not create an unrelated part in the paper, only significant data that could be found will be discussed. Both the lacunarity of 1 and the persistence of 0 had the average mode of 1.67 in islands (1.22 and 1.78 in caves respectively), indicating that these values directly affected the natural look. The highest values for average naturality were seen in these parameters:

- randomFillMapInt(1) and randomFillPercent(50) for both islands and caves.
- Octaves(3), lacunarity(3) for both and persistence(3) for islands and persistence(0.5) for caves.

Conclusion

In conclusion, the initial hypothesis was right about performance but did not look from the correct perspective about diversity and naturality. The performance of PN has proved to be better compared to CA when working with 2D terrain generation and it will only be better as the games are bigger in scale. On the other hand, both diversity and naturality were mostly dependent on their own parameters rather than the algorithms unlike the hypothesis foresaw. Moreover, diversity was a relatively objective field compared to naturality and it provided more productive output than naturality. The correlation between certain parameters and desired output could be seen, such as lacunarity under 4 being more suitable in both terrains.

This research has shown that when analyzing the field of game development only performance (and maybe diversity) could be used as a comparative field and the relation between parameters is much more important than the comparison between each algorithm.

In addition, there were certain limitations to this experiment. In performance, the generation was done for simple terrain maps and was not a great example for bigger-scale games. Also, the survey was done only by 40 people around the ages of 15 to 20, so it did not provide a full perspective on people's opinions. The age gap was limited to make the survey more understandable to people who are more familiar with games.

Further Research

In the future, it would be best to progress in the field of performance and address the most important limitation: the scale of the game. 3D maps with more colors could be created. Both would have higher computational demand because of the new 3D graphics. In this case, PN would have the upper hand because the heat map could directly be translated into 3D heights by a multiplier, whereas CA would need more colors and new techniques to turn that map into a multicolored 3D map. Also, it would benefit real-life applications as 3D is used more.

Example 2D multicolored map done by a similar code used in this paper:

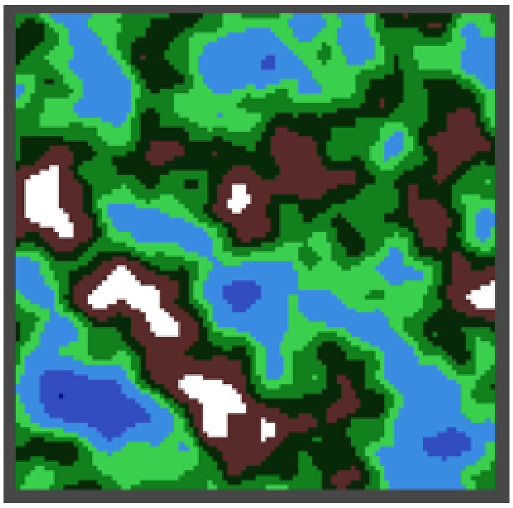


Image 10. PN Multicolor

Bibliography

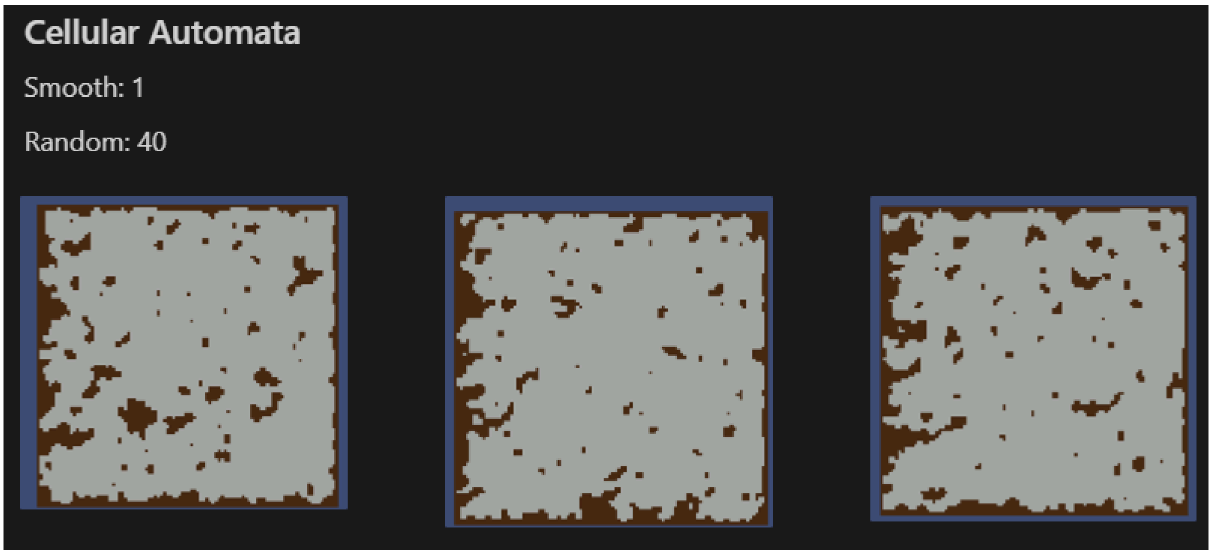
1. Barriga, Nicolas A. “A short introduction to procedural content generation algorithms for Videogames.” *International Journal on Artificial Intelligence Tools*, vol. 28, no. 02, Mar. 2019, p. 1930001, https://doi.org/10.1142/s0218213019300011.
2. Statista. “Video Games - Worldwide: Statista Market Forecast.” Statista, 2023, www.statista.com/outlook/dmo/digital-media/video-games/worldwide#revenue.
3. Schreier, Jason. Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made. Harper, 2017.
4. Delgado, David Velasco y Daniel. “Gaming Dictionary for Newbies.” *MegaInteresting.Com*, 10 Nov. 2019, www.megainteresting.com/techno/gallery/gaming-dictionary-for-newbies-771573460415.
5. Schatzeder, Dan. “The Logic of Procedural Generation.” *Medium*, Medium, 6 May 2021, schatzeder.medium.com/the-logic-of-procedural-generation-3043368a6a06.
6. Biagioli, Adrian. “Understanding Perlin Noise.” *Adrian’s Soapbox*, 9 Aug. 2014, adrianb.io/2014/08/09/perlinnoise.html.
7. “Cellular Automaton.” *From Wolfram MathWorld*, 23 Jan. 2024, mathworld.wolfram.com/CellularAutomaton.html.
8. “Cellular Automaton.” *Wikipedia*, Wikimedia Foundation, 26 Jan. 2024, en.wikipedia.org/wiki/Cellular_automaton.
9. “Perlin Noise.” *Wikipedia*, Wikimedia Foundation, 11 Dec. 2023, en.wikipedia.org/wiki/Perlin_noise.

10. Angin, Murat. “Perlin Noise Nedir?” *Medium*, Medium, 14 May 2020, medium.com/@muratangin187/perlin-noise-nedir-203c2bee9596.
11. Mount, Dave, and Roger Eastman. “Procedural Generation: 2D Perlin Noise.” CS UMD. https://www.cs.umd.edu/class/spring2018/cmsc425/Lects/lect13-2d-perlin.pdf.
12. Shen, Zhenyuan. “Procedural generation in games: Focusing on dungeons.” *SHS Web of Conferences*, vol. 144, 2022, p. 02005, https://doi.org/10.1051/shsconf/202214402005.
13. “Procedural Terrain Generation.” YouTube, uploaded by Sebastian Lague, 8 April 2019, https://www.youtube.com/playlist?list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3
14. “Procedural Cave Generation.” YouTube, uploaded by Sebastian Lague, 26 June 2015, https://www.youtube.com/playlist?list=PLFt_AvWsXl0eZgMK_DT5_biRkWXftAOf9
15. “Minecraft.” *Wikipedia*, Wikimedia Foundation, 5 Feb. 2024, en.wikipedia.org/wiki/Minecraft.
16. “Terraria.” *Wikipedia*, Wikimedia Foundation, 24 Jan. 2024, en.wikipedia.org/wiki/Terraria.
17. “How Minecraft ACTUALLY Works.” YouTube, uploaded by Alan Zucconi, 27 May 2022, https://www.youtube.com/watch?v=YyVAaJqYAfE
18. Zollner, Amelia. “Major Publishers Report AAA Franchises Can Cost over a Billion to Make.” *IGN*, IGN, 1 May 2023,

www.ign.com/articles/major-publishers-report-aaa-franchises-can-cost-over-a-billion-to-make.

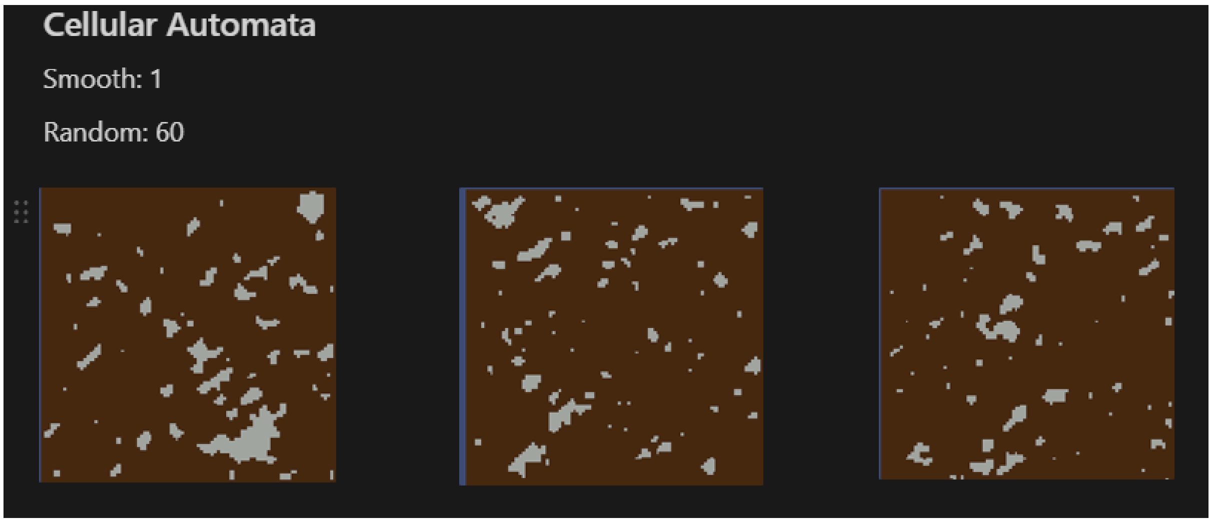
19. "Minecraft - Types of Biomes." *Minecraft Help Center*, help.minecraft.net/hc/en-us/articles/360046470431-Minecraft-Types-of-Biomes.
20. Alextramblog. "Terraria: Forest Biome." *Alex's Blog*, 22 Apr. 2016, alextramblog.wordpress.com/2016/03/08/terraria-forest-biome/.
21. Jonas Re. "Find Largest Room in 2D World." *Game Development Stack Exchange*, 19 Aug. 2021, gamedev.stackexchange.com/questions/195329/find-largest-room-in-2d-world.
22. Emmsii. "Detecting Islands on a 2D Map?" *JGO*, 18 Sept. 2014, jvm-gaming.org/t/detecting-islands-on-a-2d-map/51030.
23. Martin, Edwin. "Play John Conway's Game of Life." *Play John*, playgameoflife.com/.

Appendix


Cellular Automata

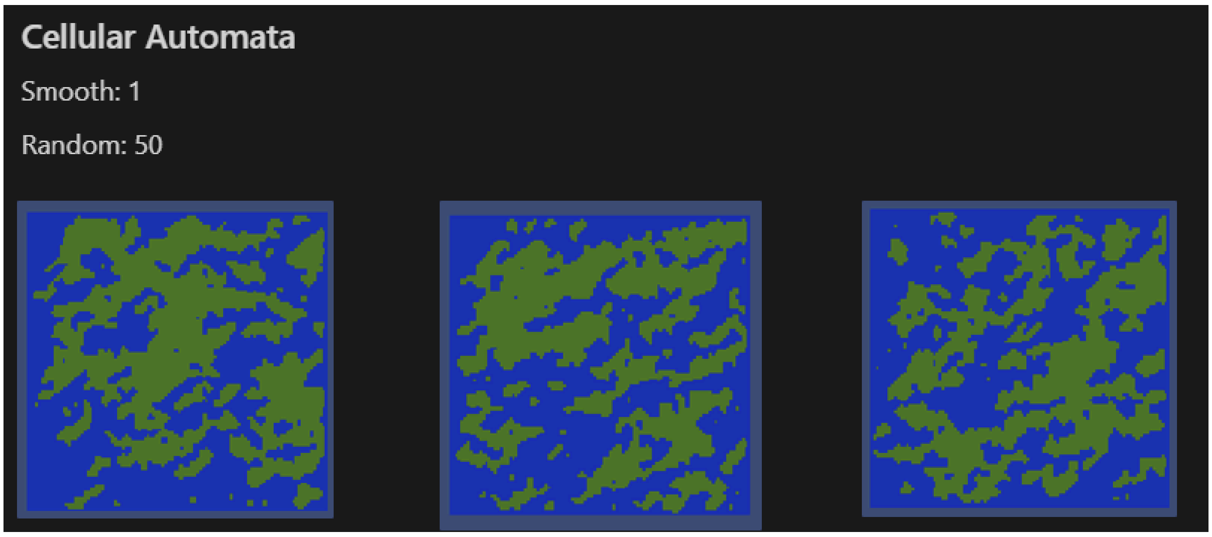
Smooth: 1

Random: 40


Cellular Automata

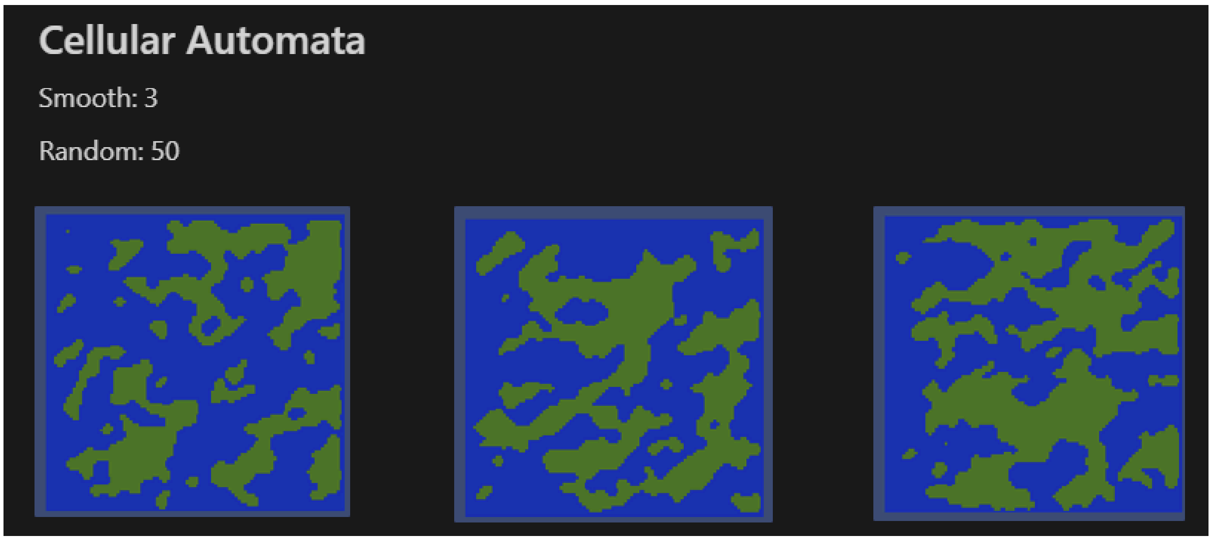
Smooth: 1

Random: 60


Cellular Automata

Smooth: 1

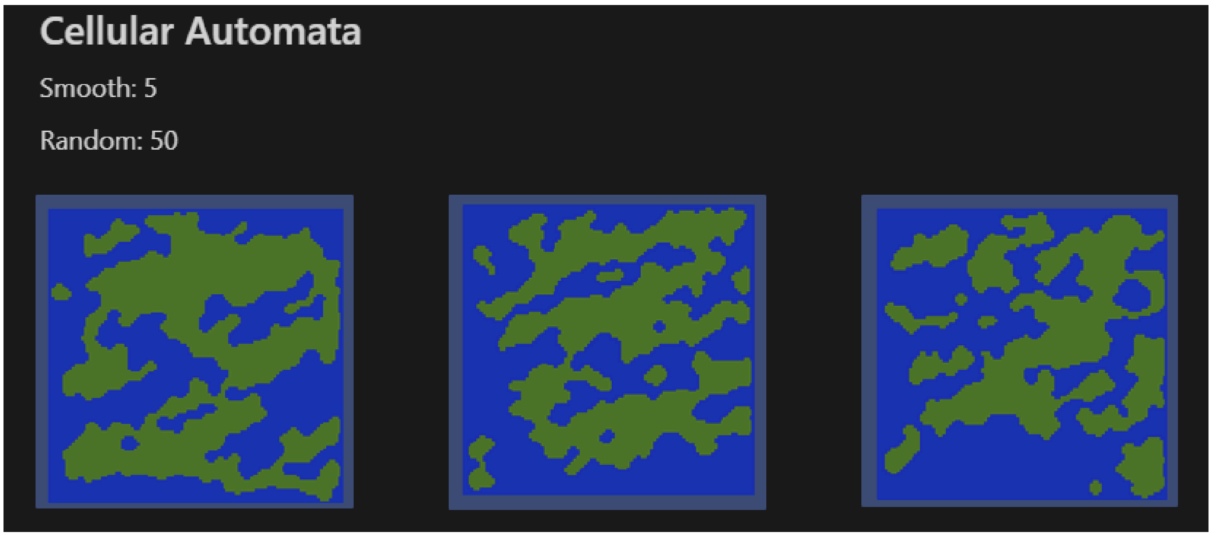
Random: 50



Cellular Automata

Smooth: 3

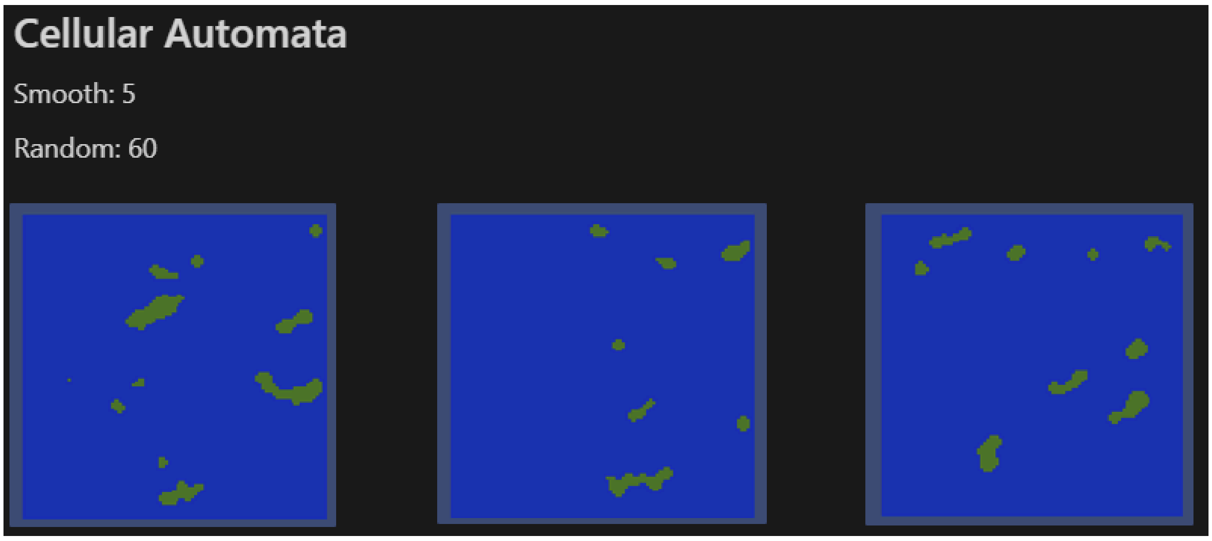
Random: 50



Cellular Automata

Smooth: 5

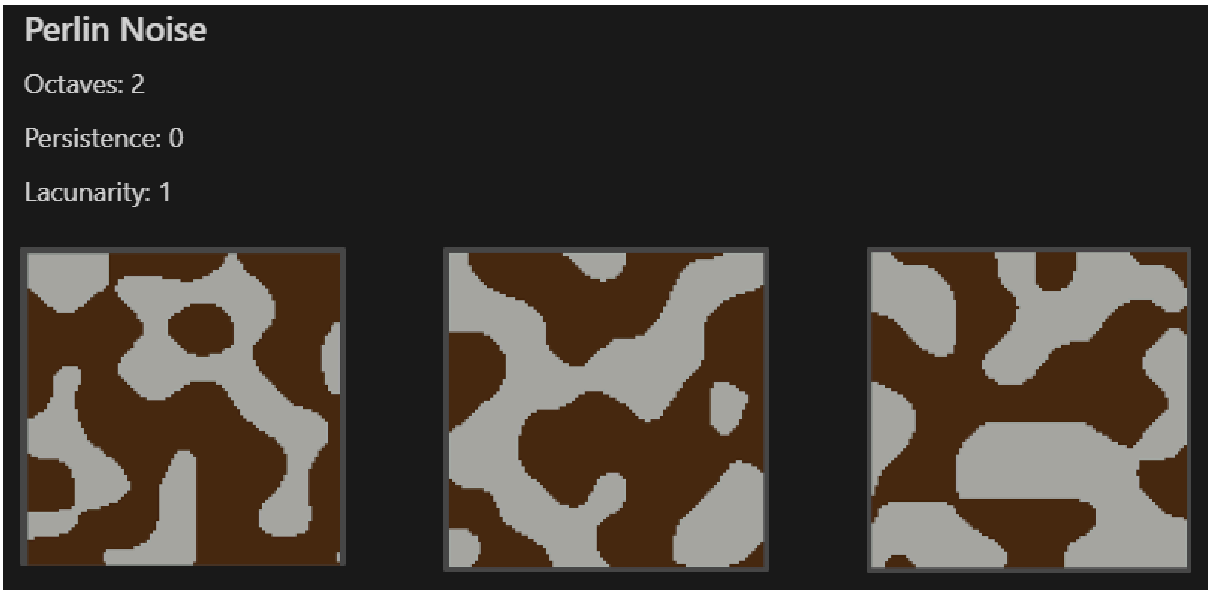
Random: 50



Cellular Automata

Smooth: 5

Random: 60

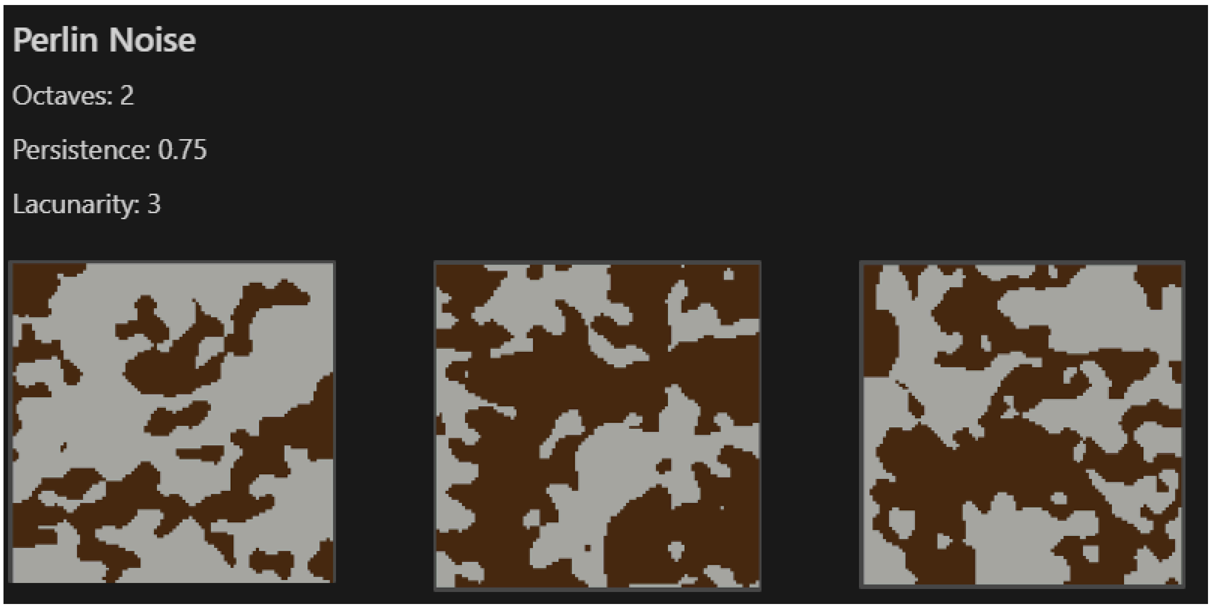


Perlin Noise

Octaves: 2

Persistence: 0

Lacunarity: 1

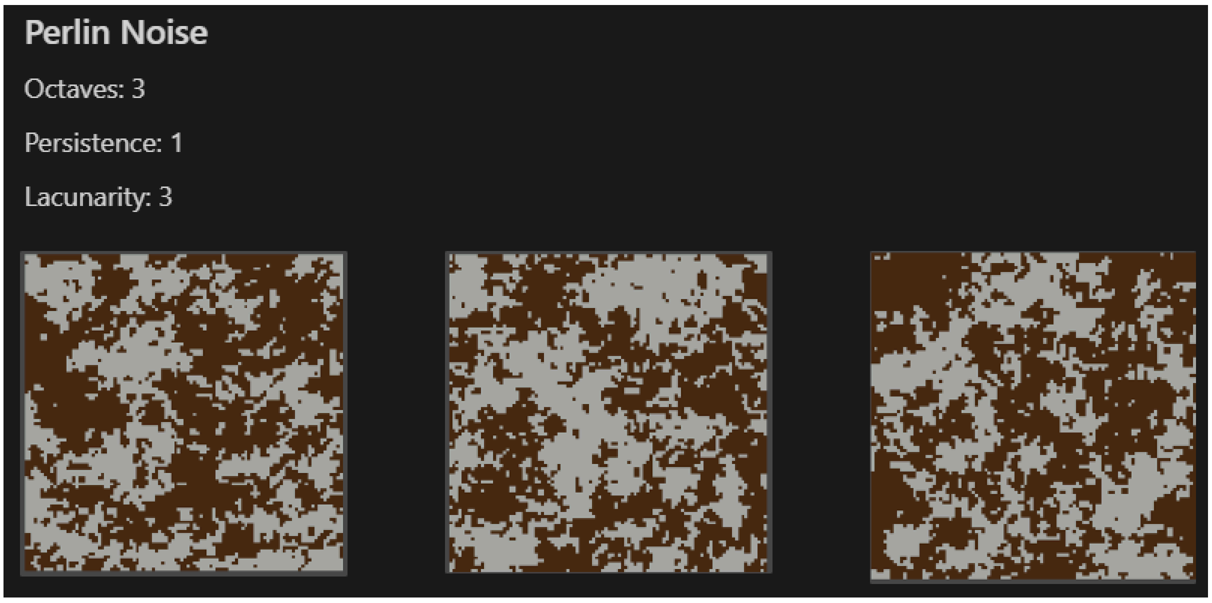


Perlin Noise

Octaves: 2

Persistence: 0.75

Lacunarity: 3

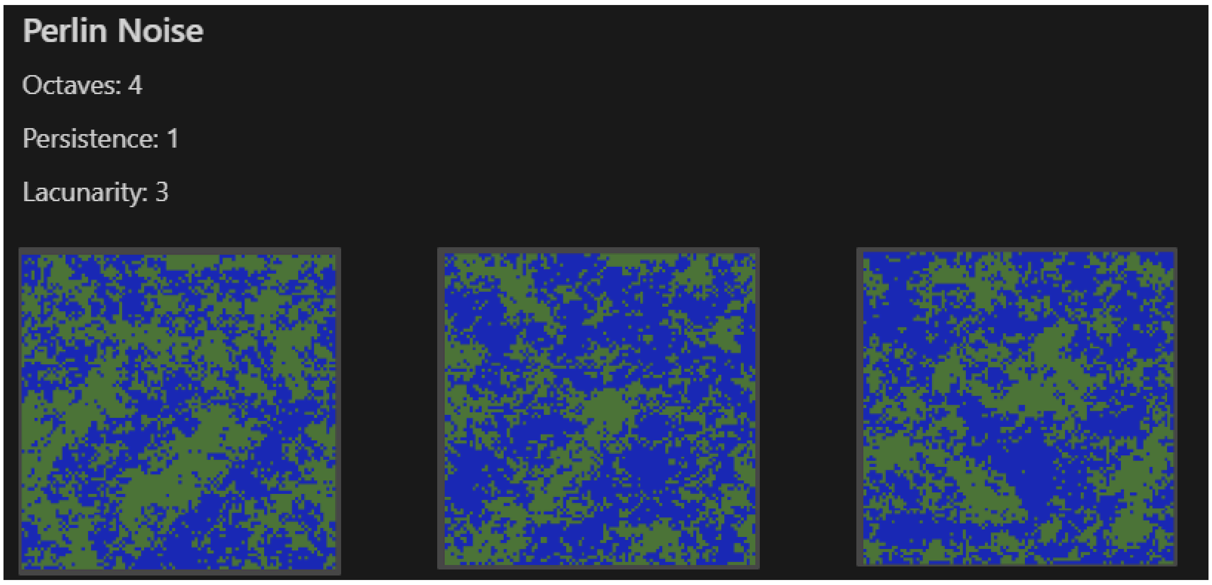


Perlin Noise

Octaves: 3

Persistence: 1

Lacunarity: 3

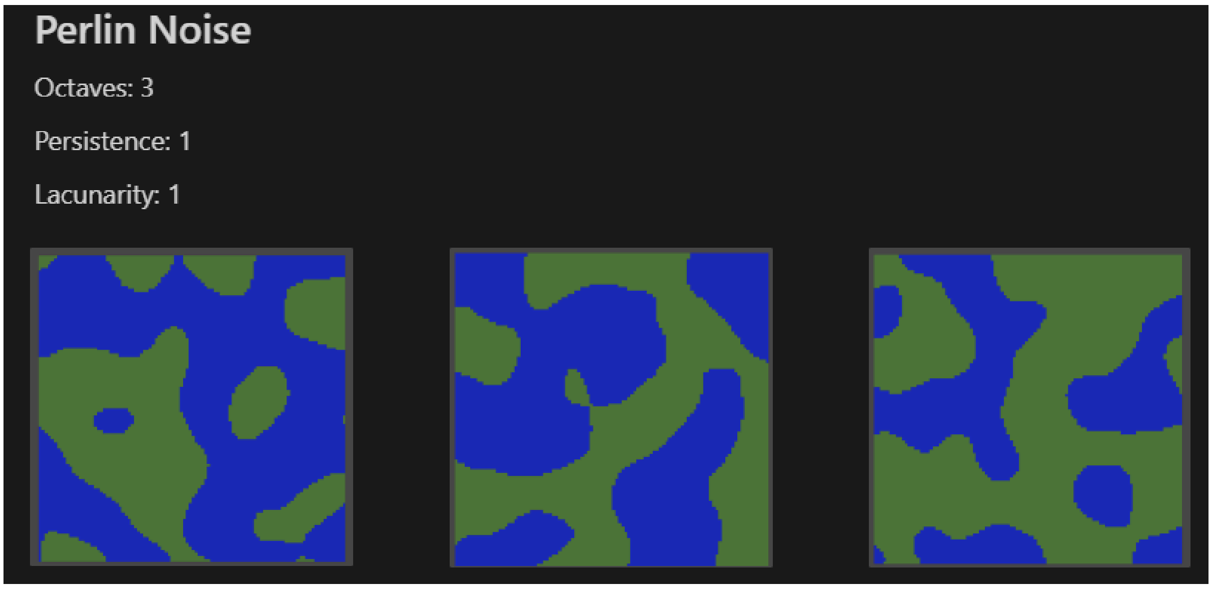


Perlin Noise

Octaves: 4

Persistence: 1

Lacunarity: 3

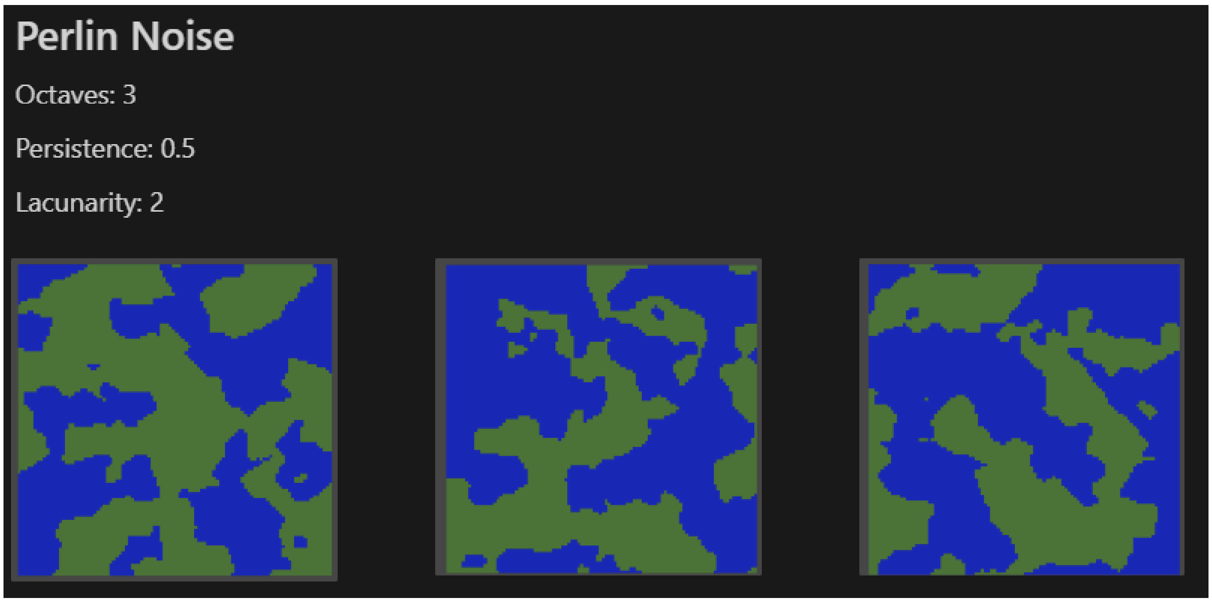


Perlin Noise

Octaves: 3

Persistence: 1

Lacunarity: 1



Perlin Noise

Octaves: 3

Persistence: 0.5

Lacunarity: 2