2D Game With Procedurally Generated Levels

Sachin Hegde[1], Satish Badhe[2], Rohit Vijay[3], Shubham Khanzode[4]
Department of Information Technology
[1, 2, 3, 4] Sinhgad Institute of Technology, Lonavala, Pune, India

Abstract-*This paper explores the idea of procedural generation through a 2d game we developed. The idea of procedural generation is touched upon and is explored a bit in the context of world generation. Building huge worlds in video games has grown increasingly difficult for small teams because of the expectation of content among gamers. Procedural generation methods like noise, perlin noise and cellular automata are one of the options to provide content to meet expectations of the gamers with smaller teams. However, some challenges exist when using these methods of procedural content generation (PCG) and this paper also touches upon the various mistakes to be avoided when working with PCG tools.*

Keywords-PCG, procedural generation, content generation, perlin noise, cellular automata

I. INTRODUCTION

Procedural Content Generation or PCG for short is a broad term used to incorporate many similar and/or dissimilar methods to computationally generate content or data. It is long being used in computer games to generate maze levels or big open areas and so on where generating content is more feasible than actually designing it by hand.

More recently however PCG is all rage with new games using it for generating everything in the game from levels to objects, enemies and so on. When thought about, it seems very practical to delegate the job to computers; but, games are meant to be immersive. To be immersive, they must seem believable. To seem believable they must look natural. As any computer scientist will say, that is the hardest thing to achieve.

II. BASICS MECHANICS OF PCG

1.) *Rules Stack:* Procedural generation as the name suggests is procedures or rules to generate something. When generating a simple object like a square, all we need is its width/height and a simple procedure can generate it. Generating Complex structures are a little different because there might be various parameters that affect the outcome. A stack or different rules for each of the parameters is therefore required for such a case.

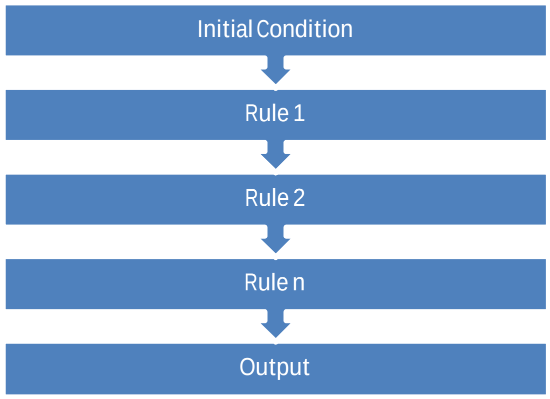

Figure II.1- Rule Stack

2.) *Random Numbers:* As said before, games are designed to look as close to the real world as possible while keeping it more exciting. The real world is not discrete and fixed as computers are. Nature loves entropy and so, it is random or we feel it is random. However, computers struggle with random numbers as there is no way a normal pc can generate true random numbers. So we turn to pseudo-random numbers which are generated using many different methods, the best known of which is linear congruential method[3]. These “random” numbers provide that essential sense of randomness when dealing with content generation.

Figure II.2- Pseudo Random Generation Visualization (khanacademy.org)

3.) *Noise:* Noise as we all know is garbled random set of data. In most cases, noise is not really used for constructive purposes, PCG makes use of noise heavily to generate anything from wind patterns, rain and so many other effects like fog and smoke. Although they are used in most games they are still procedurally generated.

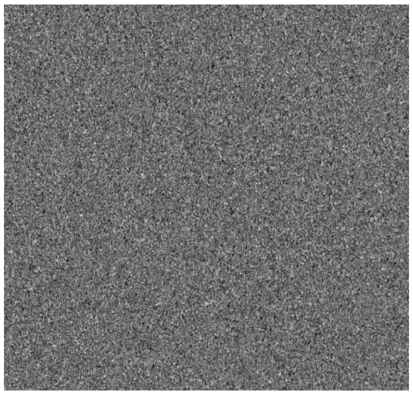
Figure II.3- Sample image of white noise (youtube.com)

III. PERLIN NOISE

What make basic noise undesirable are its abrupt data changes that do not reflect how nature actually works. Nature although random, changes are gradient. To solve this issue, Perlin Noise is the best solution we have.

Ken Perlin came up with this algorithm[1] to generate textures for a movie instead of actually drawing them and that won him an Academy Award. All the disadvantages of noise is resolved with Perlin Noise. The data we get can be adjusted to whatever type of dataset we want. The gradient can be smooth or discrete based on the scale of the noise.

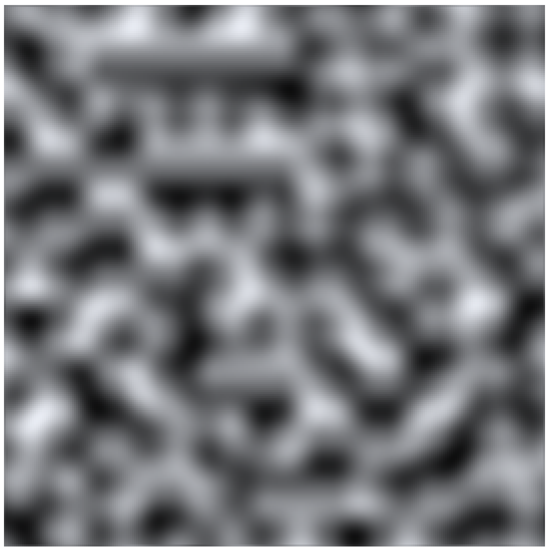
Figure III.1- Gray- Scale 2D Perlin Noise

Unity engine which we used to develop our game has an inbuilt function to get values of a Perlin Noise dataset. As Perlin() function[2] is already defined, it requires a x and a y value to return a float noise value for that map. Dividing x and y value with scale will give the value according to the scale assigned. Higher scale means more gradient while less scale means fewer gradients.

For our project we used Perlin Noise for generating the structure of the world itself. What point on the map will be water and what will be land is decided using Perlin noise. As we are developing a ship warfare game, generating huge landmass is quite convenient for the project.

A Script will accept values like the height and width of the map on an arbitrary scale. The scale of the Perlin Noise and seed value is the initial value of x and y. The initial value will determine what section of Perlin Noise will be returned. To get a new set of values, all we need to do is generate a “random” seed value or ask user for the same.

IV. SOME MISTAKES TO AVOID

Through some trial and error, we have had some experience with PCG tools. Although by no means experts, some things we did find were quite difficult with current techniques. At the end of the day, making the game was our goal, to make it immersive and playable we had to discard many of the things we decided we would do with PCG.

1. *Do not try to generate everything:* while it can be tempting to make a game fully procedural, it is quite challenging. It somehow takes away the immersion and game play smoothness from the player. While we did try hard by adding more parameters or adjusting all the rules, we fell flat.

2. *Know what you want with PCG:* Before starting the project, always make sure you know what content to generate and what to design and handcraft. Because making a game is more important than writing a thousand line procedure to generate a map.

3. *Know before code:* Before coding a procedure to generate something, one must know how that something works roughly. For example, our game has a wind system where wind will push the player’s ship in some direction; the direction of the wind is procedurally generated and updated every now and then. As wind does not blow all over the place, understanding the system is the key.

4. *Trial and error is fine:* No matter how much you plan, with PCG, you never know what you are going t o get. With trillions of possible outcomes, the developer is also an explorer of his own creation.

Tinkering with values every now and then to get the desired results is the only way to achieve the goal.

V. CONCLUSION

Our little project can surely serve as a tutorial for PCG in video games as it is working game with pretty simple mechanics. But at the same time uses all the major techniques used in mainstream PCG games.

The main goal of the project was to develop a game that is immersive and enjoyable which uses PCG as its basic building block. We think we were fairly successful in doing that although hoped we could have done more.

ACKNOWLEDGEMENTS

We take this opportunity to thank our project guide Prof. A.P Kulkarni and Head of the Department Prof. N.A.Dhawas for their valuable guidance and for giving all the necessary facilities and resources which were indispensable for the completion of this project report. Further, we are thankful to our parents who have supported us throughout our lives. We are also thankful to all the staff members of the Department of Information Technology of Sinhgad Institute of Technology for their valuable time, support, comments, suggestions and persuasion. We would also like to thank the Institute for providing the required facilities, Internet access and required books.

REFERENCES

[1] Ken Perlin, Improving Noise, SIGGRAPH, 2002
[2] Unity Scripting Reference, https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html
[3] Hui-Chin Tang, Modulus of Linear Congruential Random Number Generator, Springer, 2005