Procedural Content Generation for Video Game Development

2024

Student Name: Zhenxin Hu

Student ID: 10872814

Programme Title: BSc (Hons) Artificial Intelligence

Supervisor: Ke Chen

Department of Computer Science

Title Page: xxxx

**Contents**

[**Abbreviations** 6](#_Toc163520265)

[**Abstract** 7](#_Toc163520266)

[**Declaration** 8](#_Toc163520267)

[**Copyright** 8](#_Toc163520268)

[**Acknowledgements** 9](#_Toc163520269)

[**1 Introduction** 10](#_Toc163520270)

[**1.1 What is PCG?** 11](#_Toc163520271)

[**1.2 Why is PCG?** 12](#_Toc163520272)

[**1.2.1 Freshness** 12](#_Toc163520273)

[**1.2.2 Variety** 12](#_Toc163520274)

[**1.2.3 Cost** 12](#_Toc163520275)

[**1.3 Aims & Objectives** 13](#_Toc163520276)

[**1.4 Structure** 13](#_Toc163520277)

[**2 Background** 14](#_Toc163520278)

[2.1 Random Noise & Perlin Noise 14](#_Toc163520279)

[**2.2 Prim Algorithm** 15](#_Toc163520280)

[**2.3 Cellular Automaton and Flood fill** 16](#_Toc163520281)

**List of Tables**

**List of Figures**

[Figure 1:Random Noise (left) and Perlin Noise(right) 15](#_Toc163417203)

## **Abbreviations**

PCG: Procedural Content Generation

USD: United States dollar

CAGR: Compound Annual Growth Rate

## **Abstract**

## **Declaration**

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

## **Copyright**

1. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.
2. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.
3. The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the “Intellectual Property”) and any reproductions of copy right works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.
4. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see [*http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420*](http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420)) in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see <http://www.library.manchester.ac.uk/about/regulations/>) and in The University’s policy on presentation of Theses.

## **Acknowledgements**

## **1 Introduction**

Nowadays, video games have come into our lives and more and more people are playing video games, with millions of people playing games every day. Mordor Intelligence (2023) state that the gaming market size is expected to be United States dollar (USD) 272.86 billion in 2024 and is projected to reach USD 426.02 billion by 2029, growing at a Compound Annual Growth Rate

(CAGR) of 9.32% during the forecast period (2024-2029). The expansion of the game market of course cannot be separated from the body of the game, the body of the game is rightly more and more sophisticated, which is also the players concerned about the place. However, players' requirements for games are also rising. The rise in the number of players and the high demand for games has made it increasingly difficult to make game content manually (Iosup, 2010). Compared to the disadvantages of handmade games as mentioned above, the advantage of Procedural Content Generation (PCG) is that by using algorithms, it is possible to automatically generate the relevant game content, thus making the whole game less expensive (Risi et al., 2014).

## **1.1 What is PCG?**

The full name of PCG is Procedural Content Generation, which is defined as an algorithm generating relevant game content through some inputs that are not direct (Togelius, J. et al., 2011) and what is generated is like what would be written to mimic a human being (Smith, G.E. et al., 2012). Thus, PCG is also, in a sense, a game that players and designers work on together. PCG isn't a brand-new concept, it's been used on top of many games and their commercialisation has been a major success. Some of their more recognisable games that the *Binding of Isaac*, which creates 10-20 rooms from 200 existing styles; *Civilization* series that generate game worlds based on attributes like in-game land area, temperature, climate, time and period. And other typical examples such as Minecraft, No Man's Sky etc. The above games can be divided into two types in terms of how the world is generated (Short, T.X. and Adams, T., p.99, 2017):

1. Creating a sparse game environment to enable players to discover and modify the landscape at their will.
2. Craft a world where nearly all elements are pre-established for player interaction.

In other words, category 1 refers to the fact that, for example, Minecraft generates different terrains on which the player is able to build all kinds of tools and constructions through player’s own ability and creativity, while category 2 refers to the fact that the player is able to interact with the buildings or planets, but these buildings or planets are already generated and fixed, and the player is unable to change them.

## **1.2 Why is PCG?**

### **1.2.1 Freshness**

If a map, doesn't have a PCG system, then what the player sees every time they enter that map is exactly the same. It is undeniable that the first time the player sees is certainly a sense of freshness, but if it has been continued to use the same map, the player's sense of freshness will certainly continue to fade, especially the kind of exploration for the purpose of the game, even if the mechanism is more complex, the player will also get tired of playing the day, and this also limits the life of a game. Of course, this does not mean that all game maps should apply PCG technology, such as L4D2 (published by Valve in 2009, a first-person shooter through the killing of zombies to reach the destination), although it also has elements of exploration, but it is more fun in the teamwork, and a fixed map will be more in line with the needs.

### **1.2.2 Variety**

With the world generated by the PCG, players can play through the game in different ways, and that's where some of the appeal of Roguelike comes in. For example, in the Roguelike game " The Binding of Isaac ", players use their own "builds" to get through the game. The different combinations of props and the fact that each time the map is brand new makes the player enjoy playing game after game. In addition, since the maps are new every time, even if the build is the ideal one, the player may make a mistake due to the different maps, which may lead to failure. Therefore, a good roguelike player will modify his strategy according to the current situation.

### **1.2.3 Cost**

An example is the classic PCG game "Elite", which uses PCG technology to generate a universe of eight galaxies, each of which has about 256 solar systems. In each solar system, there are about one to twelve planets. Each planet has a space station in orbit above it, name, terrain, price and details. And limited by the 8-bit computers of the time, these were generated by programs that determined the combination of planets by fixed algorithms (Amato, A., 2017). Assuming each star system's data needs only 1 KB, without procedural content generation, the entire system would need over 400 TB of storage space (Aversa, D., 2015). And by now, the use of PCG technology not only saves in terms of storage space, but also saves a lot of time for the designer, while it also generates the world according to certain rules that have already been laid down to make the whole world look more natural.

## **1.3 Aims & Objectives**

The aim of this project is to produce a video game capable of generating content dynamically as it is played, which can be deployed on PCs. The core logic of the game is to be able to automatically generate relevant game content through algorithms written in-house. And the objectives of this project are following:

1. Pygame was used as the engine for this project, which adds the main, pause, end and game interfaces to the game.
2. Generate a map in the main game using the Perlin Noise function, use this algorithm to distinguish between different terrains (mountains, land, rivers, and the intersection of mountains and land), use this map as the map for the main game, and add enemies and players to this map.
3. In Subgame 1, the Prim algorithm was used as the base algorithm for the random maze, which ensures the connectivity of the maze without making it too simple.
4. In Subgame 2, the core algorithm was a Cellular Automaton and using flood fill to guarantee the size of individual caves, which is used to generate the desired cave map.

## **1.4 Structure**

The report will contain () chapters:

1. **Introduction**: An introduction to the project
2. **Background**:

ddd

## **2 Background**

## **2.1 Random Noise & Perlin Noise**

Noise has different definitions in different fields and applications, and we focus here on noise in computer graphics. Noise is utilized in procedural texturing for a wide variety of applications, such as generating cloud patterns, wave textures, tornado effects, rocket exhaust trails, heat distortion ripples, and the subtle movements of animated characters, among others (Lagae, A. *et al.* 2010). A primary use of this method is in creating terrain, employing various seeds to produce different landscapes each time (Tuomo Hyttinen, Erkki Mäkinen and Timo Poranen, 2017). Whether the array is generated by random noise or by the Perlin noise function, they are just a 2D or 3D array, which is meaningless. Therefore, with the idea of volume up contour map, the numbers obtained above are scaled to the range of grey scale map (to make it easier to understand, only 2 dimensions are used), which is from 0 to 255, and we can identify the numbers as different terrains according to the size of the numbers (like the contour lines). For example, for values larger than 153, we identify them as mountain ranges; 127 to 153, we identify it as the intersection of mountains and land; 102 to 127.5, we identify it as land; and values less than 102 are identified as rivers. These values can be manually tuned to make it closer to the real world.

However, just like Noor Shaker (2016, p.71) said in his book “*Maps can be generated very simply by randomizing these numbers, though this leads to unnatural and ugly maps*.” The problem with random noise is that it is simply too random, and it is difficult to generate terrain that is similar to the real nature, which is random but has some patterns implied in it. So, Perlin (1985) suggests that an ideal noise requires the following properties:

1. Statistical invariance under rotation
2. A narrow bandpass limit in frequency
3. Statistical invariance under translation

And Perlin also presents its own improved function, the Perlin Noise function, in this article.

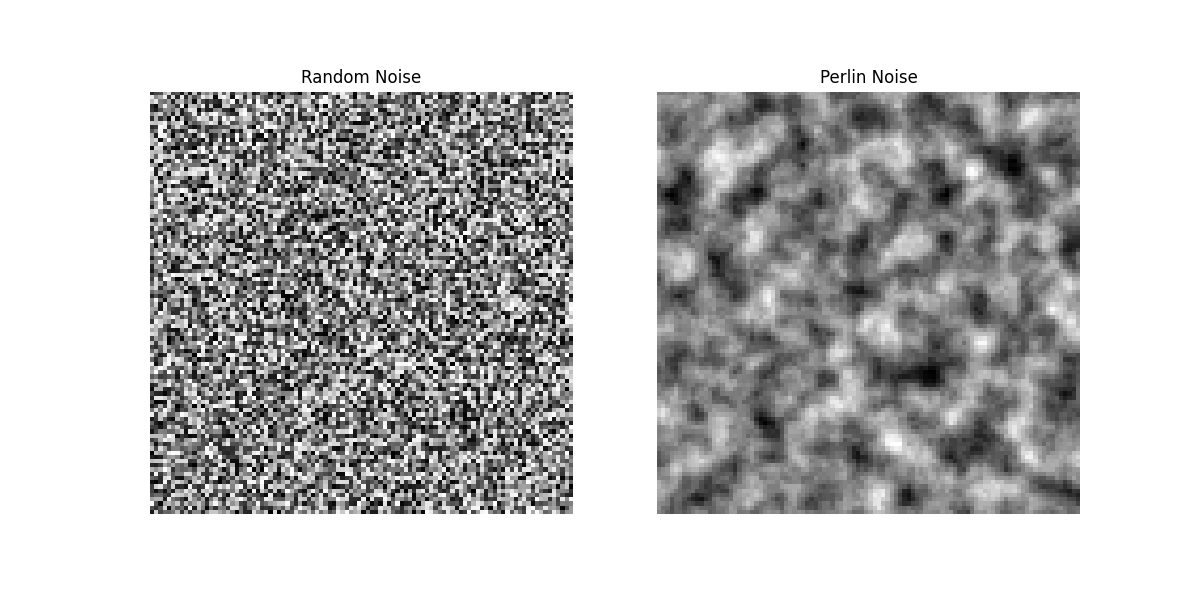


Figure 1:Random Noise (left) and Perlin Noise(right)

Perlin noise is also a type of noise, but it is different from random noise in that Perlin noise looks more continuous, that's all. In terms of Figure 1 (left), the change in random noise is very abrupt. And it's obvious that the noise in Figure 1 (right) looks smoother, with less abrupt changes, which is the main principle of how we generate mountains, land, and rivers: using a random number that has a less abrupt tendency to change. To generate random numbers "smoothly", one-dimensional noise is interpolated to make it smooth, but this results in a flawed noise, with high undulations in some places and gentle undulations in others. Therefore, the "gradient" of the noise is chosen to be interpolated so that the result is not too abrupt. It is often used in video games to make program-generated terrain look natural. This success is due in part to the layered structure of Perlin noise, which mimics the naturally occurring layered structure that (Etherington, T.R. 2022).

## **2.2 Prim Algorithm**

The Prim algorithm is a greedy algorithm that finds the minimum spanning tree for a weighted undirected graph. This means that it finds subsets of edges to form a tree containing every vertex where the total weight of all edges in the tree is minimized. The Prim algorithm is also one of the algorithms used by PCG to generate mazes (Agnesia and Istiono, W., 2023). And there are some slight changes in the prim algorithm for generating mazes. Starting with all cells initialized as walls, a cell is randomly acquired, and its neighboring cells (front layers) are detected, one of which is randomly added to the maze. As cells are added to the maze, new boundaries are detected. The algorithm continues until all cells are in the maze (Kozlova, A., Brown, J.A. and Reading, E., 2015).

## **2.3 Cellular Automaton and Flood fill**

### **2.3.1 Cellular Automaton**

A cellular automaton is a mathematical model for simulating the dynamic behavior of a system consisting of many simple components, called "cells". In the cellular automaton model, each cell is in a state in a finite set of states and updates its state according to defined rules based on the states of itself and its neighboring cells. These updates usually occur simultaneously in discrete time steps. A cellular automaton consists of the following components (Rollier, M. et al., 2024):

1. Lattice: a collection of cells, usually arranged in a regular grid in one, two or three dimensions.
2. State: the state that each cell can take, usually limited, e.g. "alive" or "dead".
3. Neighborhood: for each cell in the grid, defines a set of surrounding cells whose state affects the update of that cell's state.
4. Transition rules: a set of rules that define how the next state of a cell is calculated based on the current state of the cell and its neighbors.

Here using one of the examples of cellular automata, "The game of Life". It is a zero-player game. The concept is based on an infinite two-dimensional lattice where cells can be either 'active' (alive) or 'inactive' (dead). The evolution of each cell is dictated by specific rules that consider the states of its neighboring cells. Starting from an initial pattern, the configuration transforms over time autonomously, with no additional input from users, hence it is known as a 'zero-player' game (Izhikevich, E., Conway, J. and Seth, A., 2015). Martin Gardner (1970) made the rule as the Game of Life operates on an endless two-dimensional grid where each cell can be in one of two states: 'on/alive' or 'off/dead'. This cellular automaton progresses in discrete time steps. The future state of each cell at time t is influenced by its current state and the states of its eight surrounding cells at time t-1 (this surrounding area is known as the Moore neighborhood with a radius of 1). The evolution of these cells is governed by a set of straightforward rules:

1. A cell that is 'alive' at time t-1 and is surrounded by fewer than two alive neighbors at the same time will turn 'dead' at time t.

2. An 'alive' cell at time t-1 will stay 'alive' at time t if it has two or three alive neighbors at time t-1.

3. An 'alive' cell at time t-1 will become 'dead' at time t if it has more than three alive neighbors at time t-1.

4. A cell that is 'dead' at time t-1 will become 'alive' at time t if it has exactly three alive neighbors at time t-1.

### **2.3.2 Flood fill**

Flood fill is defined as every pixel within the specified region holds the value old\_value, while none of the pixels on the region's edge share this value. The objective of these algorithms is to modify the value of each pixel within the specified region to a new value, known as new\_value. New\_value means the user actively selects areas of contiguous pixels that are then populated with a specified pixel value, here referred to as new\_value (Foley, J.D. and Andries Van Dam, 1982, pp.446-447). Since in the game, the player can only operate up, down, left and right, the choice of the Four-way flood filling, which is starting from the seed point, the algorithm spreads up, down, left, and right to find the pixels that need to be filled. Although the recursive version of this algorithm is probably better known, a non-recursive implementation is used to avoid the stack overflow problem that recursion can cause. In this method, when a pixel satisfies the fill condition, the pixels around it are added to the stack and the process continues until the stack is empty (Heckbert, P.S., 1990).