BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2024
http://excel.fit.vutbr.cz

Maze-based 2D game

Kateřina Čepelková*

Abstract
This project focuses on leveraging the cellular automaton to procedurally generate the mazes. The objective is to develop an algorithm capable of generating diverse mazes of varying sizes. The created algorithm is then used in designing a 2D game (in Godot) where the player's goal is to reach the highest level possible while navigating through dynamically generated mazes. Along the way, they will encounter various threats strategically spawned within the maze.

*xcepel03@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Procedural terrain/content generation is a widely utilized method in game design, used, for example, in No Man's Sky, Terraria, or Minecraft. It can save memory, because there is no need to save pre-designed maps, and more importantly human labor, as it can be exhausting and repetitive, so designers can focus on more critical design problems [1].

This project swiftly generates 2D maze-like maps usable in the base game itself, in additional puzzles or levels. It brings an element of randomness, guaranteeing the creation of dynamic and unpredictable 2D maps of all sizes, ensuring each playthrough offers a unique experience and challenge.

When generating maps, randomness is essential, but it needs to be controlled to achieve optimal outcomes. That is why this project tackles generating using a few different methods and algorithms resulting in effective mazes. These are presented in the implemented game, developed in the game engine Godot 4 using GDScript. It also addresses the challenge of integrating objects into the game environment, as the procedural generation process makes it difficult to distribute them within the maze.

2. Automation of maze generation

The algorithm chooses the best way to approach the generation of labyrinths based on their size. Tests were performed on various algorithm combinations to determine the most optimal method to generate mazes of different sizes, based on factors such as the number of rooms generated or the length of the longest path within the maze. The tests resulted in the discovery that groups of mazes of different sizes needed some different starting matrix or approach with CA. This commentary focuses on describing the general approach to the creation.

2.1 General map generation

For achieving diversity in generated mazes the starting base is a matrix of a given size filled with 1 (wall tile) and 0 (path tile). This initial setup is facilitated by Godot native (pseudo)random number generator which uses *PCG32*, the result is illustrated in **Figure 1**. The distribution of the ratio of generated 1 and 0 is controlled by modulo operation.

This matrix is then put through a single iteration of the cave CA, the result is seen in **Figure 2**, which creates clusters of 1s (representing walls) which helps to achieve better outcomes for the second CA. The second CA utilized is *OCA:Maze/Mazectric* with rule 1234/3 [2]. This CA is explosive (most randomly generated starting patterns will explode in all directions). Therefore, the clusters formed by the cave CA serve as optimal seeds for its growth. Although the resulting maze CA pattern appears maze-like at first glance, upon closer inspection it reveals corridors that are not all interconnected, as demonstrated in **Figure 3**.

This algorithm resolves this issue by systematically identifying all groups of 0s, representing paths within the maze. It then iterates through each group, seamlessly connecting them into a single cohesive structure

by strategically breaking one random wall that separates them. This process not only resolves the problem, it also introduces another element of diversity to the generation. An example of the resulting maze is in **Figure 4**.

2.2 Path finder

The idea to find the start and finish is to find the two furthest points from each other, but this cannot be easily achieved when trying to generate quickly. That is why this project uses the method breadth-first search (BFS). Firstly it gets the first available path tile in the matrix from the left-hand corner of the matrix and finds its furthest path tile. This is the start of the maze – on this tile BFS is applied again resulting in the path tile that is set as the finish.

3. Placement of game entities

Ensuring the diverse placement of game entities, such as enemies and items, requires a specialized algorithm. The maze layout, start and end path, and difficulty levels must be considered.

The implemented solution to this problem is rejection sampling. It creates a matrix of the same size as the maze with random 1 and 0 (similar to **Figure 1**) – this is the sample function/matrix. This matrix with samples is "placed" on the generated map matrix and all samples (tiles with value 1 from the sample matrix) that share coordinates with the path tile in the maze matrix are taken as potential spawn points for entities. The list of potential spawn points is then filtered to remove tiles too close to the start so that the player won't have issues with running into the enemy too soon, and finish tile. This implementation of finding possible entity spawn points is visualized on **Figure 6**. Now the list is shuffled and can be passed on to the generating function, which chooses the needed number of tiles to assign to all the required entities.

4. Game and its mechanics

The game is based on simple, skill-based arcade games, where the player's goal is to reach the highest level with the highest score, such as Pac-Man or Galaga. The levels are increasing in difficulty and challenge thanks to the rising size of the generated maze and the addition of enemies and items (some of which are seen in **Figure 5**.

The game offers many types of enemies with diverse mechanics. Some of them are for example "demon" enemy, which spawns fire, "goblin" with bow and arrows, or slime, which slows the player upon contact. The player's goal is to survive and reach the end of the maze in the shortest time, which gets him into the new level/maze. The score from each level is based on the level difficulty, the time it took to complete the level, discarded enemies, and the items collected. Upon losing all health player loses the game. Game screenshots are in **Figure 7** and **Figure 8**.

5. Conclusions

The topic of the thesis was to develop a method of automatic maze generation using cellular automaton and then implement it into a game. Maze generation has been developed, tested, and optimized. Thanks to the usage of CA the results are generated very quickly which is important when implemented in the game.

The game was created in Godot and its alpha version has been tested on volunteers, which brought important feedback not only for the game but for generating itself.

Acknowledgements

I would like to thank my supervisor Ing. Michal Vlnas, for his time and guidance throughout this project. I am also thankful to all who volunteered to test my game; their input was invaluable.

References

[1] Zhixuan Wu, Yuwei Mao, and Qiyu Li. Procedural game map generation using multi-leveled cellular automata by machine learning. In *Proceedings of the 2nd International Symposium on Artificial Intelligence for Medicine Sciences*, ISAIMS '21, page 168–172, New York, NY, USA, 2021. Association for Computing Machinery.

[2] LifeWiki. Oca:maze. online, 2022.