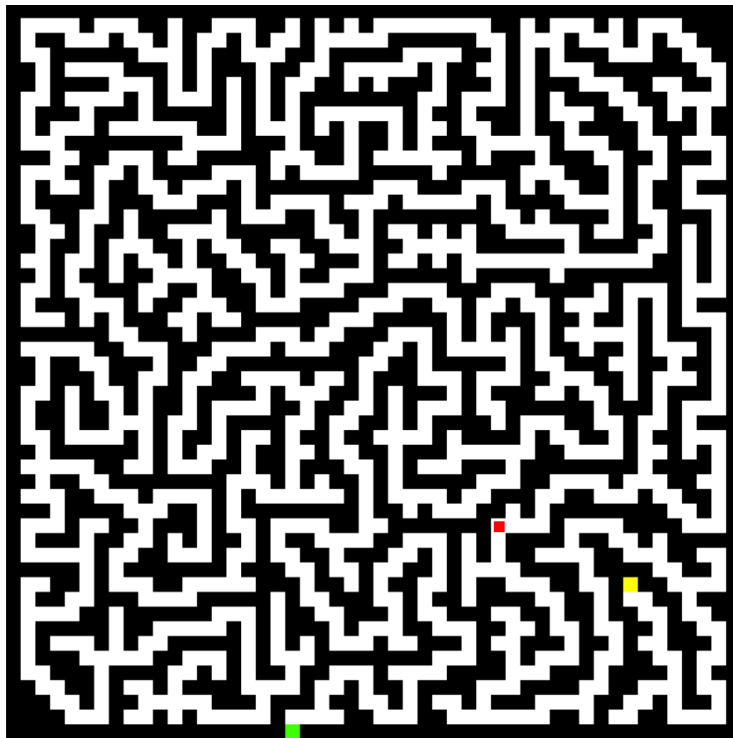


# Maze Game Report

**Author:** Neil SAUVAGE

**Date:** June 11, 2023



# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Maze Creation Algorithm</b>	<b>3</b>
2.1	Algorithm Steps . . . . .	3
2.2	Maze Difficulty Levels . . . . .	4
<b>3</b>	<b>Classes Used in the Python Code</b>	<b>7</b>
3.1	Maze . . . . .	7
3.2	Cells . . . . .	7
3.3	Cell . . . . .	7
3.4	Direction . . . . .	7
3.5	Wall . . . . .	7
<b>4</b>	<b>Game Flow</b>	<b>8</b>
4.1	Initialization . . . . .	8
4.2	Player Movement . . . . .	8
4.3	Collision Detection . . . . .	8
4.4	Victory State . . . . .	8
<b>5</b>	<b>Conclusion</b>	<b>9</b>

# 1 Introduction

The maze game is an engaging and challenging puzzle game where players navigate through intricate mazes to reach the exit. In this report, we present an in-depth analysis of the maze game, including its design, implementation, and gameplay mechanics.

The objective of the game is to guide the player character, represented by a red square, from the starting point to the green tile, which signifies the exit. The mazes are carefully designed to offer varying levels of difficulty, providing players with options to choose their preferred challenge level, such as easy, medium, difficult, and crazy mazes.

In this report, we will discuss the underlying algorithms and data structures used for maze creation, specifically the randomized depth-first search algorithm. We will also explore the different difficulty levels and examine how they affect the maze complexity and gameplay experience.

Additionally, we will delve into the game flow, explaining how players interact with the maze, move the player character, and ultimately achieve victory. Alongside the textual descriptions, relevant images and illustrations will be provided to enhance the understanding of the gameplay mechanics and maze design.

This report aims to provide a comprehensive overview of the maze game, including its development process, gameplay elements, and the strategies employed to create immersive and challenging mazes.

## 2 Maze Creation Algorithm

The maze creation algorithm is a crucial component of the maze game. It determines the layout and structure of the maze, ensuring that each maze offers a unique and challenging experience for the players. In this section, we will explore the randomized depth-first search (DFS) algorithm used for maze generation.

The randomized DFS algorithm begins by initializing a grid of cells, where each cell represents a possible location in the maze. The algorithm starts with a single cell as the starting point and marks it as visited. It then randomly chooses an unvisited neighboring cell and connects it to the current cell by removing the wall between them. This process continues recursively until all cells have been visited.

To create more complex and intricate mazes, the algorithm incorporates randomness during the selection of neighboring cells. When choosing the next cell to visit, it shuffles the list of neighboring cells to introduce unpredictability. This randomization ensures that the maze paths take unexpected turns and vary in length, adding to the challenge and excitement for the players.

The randomized DFS algorithm guarantees that every cell in the maze is reachable from any other cell, creating a fully connected maze without isolated areas or unreachable paths. It also ensures that there is only one possible path between any two cells, preventing the creation of loops or multiple routes to the exit.

By utilizing the randomized DFS algorithm, the maze game offers almost an infinite possibilities of maze configurations, each with its own unique layout and difficulty level. Players can expect a diverse and challenging experience as they navigate through these meticulously crafted mazes.

### 2.1 Algorithm Steps

The randomized DFS algorithm for maze creation can be summarized into the following steps:

1. Initialize a grid of cells representing the maze.
2. Choose a starting cell and mark it as visited.
3. Randomly select an unvisited neighboring cell.
4. Connect the current cell with the chosen neighboring cell by removing the wall between them.
5. Recursively repeat steps 3-4 for the chosen neighboring cell.
6. If there are no unvisited neighboring cells, backtrack to the previous cell and continue the process.
7. Repeat steps 3-6 until all cells have been visited.

The randomization and recursion in the algorithm result in the creation of intricate and challenging mazes, ensuring a captivating gameplay experience for the players.

## 2.2 Maze Difficulty Levels

The maze game offers multiple difficulty levels for players with varying skill levels and preferences. Each difficulty level affects the maze's complexity and navigational challenges. Here are the difficulty levels available in the game:

- **Easy:** Easy mazes feature simpler layouts with fewer dead ends and shorter paths. They serve as an ideal starting point for players new to the game or those seeking a more relaxed and straightforward gameplay experience.



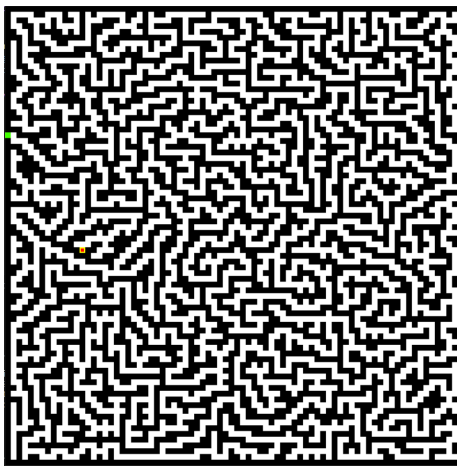
- **Medium:** Medium mazes provide a balanced challenge, offering a mix of longer paths, dead ends, and occasional twists and turns. Players can expect a moderate level of difficulty that requires strategic decision-making and navigation skills.



- **Difficult:** Difficult mazes present players with intricate layouts, longer paths, and numerous dead ends. They demand careful planning, patience, and precise navigation to reach the exit successfully. These mazes are recommended for experienced players looking for a more demanding and mentally stimulating gameplay experience.



- **Crazy:** Crazy mazes push the boundaries of complexity and challenge. They feature convoluted paths, multiple loops, deceptive dead ends, and unpredictable twists. Only the most skilled and persistent players can conquer these mind-boggling mazes. They provide the ultimate test of problem-solving abilities and determination.



The availability of multiple difficulty levels ensures that players can have their gameplay experience adapted to their skill level and desired challenge. Whether you are a novice seeking a gentle introduction or a seasoned maze master craving an extreme test, the maze game offers the perfect difficulty level for everyone.

In the next sections, we will go deeper into the game development with the expansion of the different classes used.

## 3 Classes Used in the Python Code

In the implementation of the maze game, several classes are utilized to represent different elements and functionalities. These classes encapsulate the behavior and properties of various components, enabling the creation and interaction within the game. Let's explore the classes used in the Python code:

### 3.1 Maze

The **Maze** class represents the overall maze structure. It is responsible for initializing the maze's size, managing the cells within the maze, and implementing the maze creation algorithm. The **Maze** class also tracks the starting and spawn cells within the maze, which are essential for gameplay.

### 3.2 Cells

The **Cells** class is responsible for managing the individual cells within the maze. It maintains a grid of cells, with each cell being aware of its position in the maze and its visited status. The **Cells** class provides methods to access neighboring cells, check for unvisited cells, and display the maze's current state.

### 3.3 Cell

The **Cell** class represents an individual cell within the maze. It stores the cell's position, visited status, and whether it is a wall or not. The **Cell** class provides methods to check the visited status, mark a cell as visited, and retrieve its position.

### 3.4 Direction

The **Direction** class is a utility class that defines the different directions used within the maze. It represents the possible movement directions, including North, East, South, and West. The **Direction** class also provides methods to invert a direction and obtain diagonal directions.

### 3.5 Wall

The **Wall** class represents a wall object within the maze. It is responsible for defining the position and appearance of a wall. The **Wall** class uses the Pygame library's **Rect** class to create a rectangular shape representing the wall on the game screen.

These classes work together to create and manage the maze, track the player's position, and provide the necessary functionalities for gameplay. By encapsulating the game elements within classes, the code follows object-oriented principles, promoting modularity, reusability, and ease of maintenance.

In the next sections, we will explore the game flow of the maze.

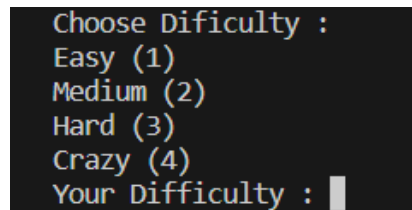


## 4 Game Flow

The maze game follows a specific flow that governs the player's interaction and progress within the maze. Understanding this flow is essential for a seamless and engaging gaming experience. The following outlines the typical game flow:

### 4.1 Initialization

Before the maze appears on the screen, the player is prompted to choose a difficulty level. The available difficulty levels include easy, medium, difficult, and crazy. The player selects the desired difficulty level through the console. Once the difficulty level is chosen, the maze is generated accordingly, and the player's starting position is determined.



```
Choose Dificulty :  
Easy (1)  
Medium (2)  
Hard (3)  
Crazy (4)  
Your Dificculty : █
```

### 4.2 Player Movement

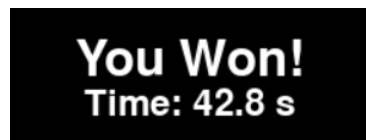
Once the game is initialized and the maze is displayed on the screen, the player can begin moving within the maze. The player interacts with the game using keyboard inputs to navigate through the corridors and rooms. The player can move up, down, left, or right to explore different paths and find the exit.

### 4.3 Collision Detection

During player movement, collision detection is performed to check if the player has encountered a wall or reached the exit. If the player collides with a wall, their movement is blocked in that direction, and they must choose an alternate path. If the player reaches the exit tile, the game transitions to the victory state.

### 4.4 Victory State

When the player successfully reaches the exit tile, a victory message is displayed on the screen, congratulating the player for completing the maze. The message may also include the time taken to finish the maze, providing a sense of accomplishment.



```
You Won!  
Time: 42.8 s
```

## 5 Conclusion

In conclusion, the maze game provides an exciting and challenging experience for players. Through the implementation of maze generation algorithms, player movement mechanics, and victory conditions, the game offers a dynamic and immersive gameplay environment. The project showcases the use of Python programming language and the Pygame library to create a maze-based game with various levels of difficulty.

Future enhancements to the game could include additional features such as enemy characters, or multiple levels with increasing difficulty. These additions could further enhance the gameplay and provide additional depth to the overall gaming experience.

In conclusion, the maze game project showcases the potential for creativity and innovation within game development. By combining algorithmic complexity, user interaction, and visual aesthetics, the game captivates players and offers an enjoyable and immersive journey through intricate mazes.