Lost in Maze: A Pygame Adventure

ENIGMA ESCAPE

Abhilasha Sharma Suman

23B1011

Contents

1 Problem Statement

The aim of this project was to design and implement a variant of a 2D maze game using Pygame, a popular Python library for game development. The game consists of a single player-controlled character navigating through a maze with a restrictive 2D top view while avoiding obstacles and traps.

2 Modules Used

- Pygame
- Sys
- Random

3 Directory Structure

- game.py
- Player.py
- BackGTheme.mp3
- gameLoopDisplay.py
- Wilson.py
- Path.txt
- scores.txt
- utils.py
- screenChange.py
- Images

— AvatarsSelSize

— BackGrounds

— Characters

— Collectibles

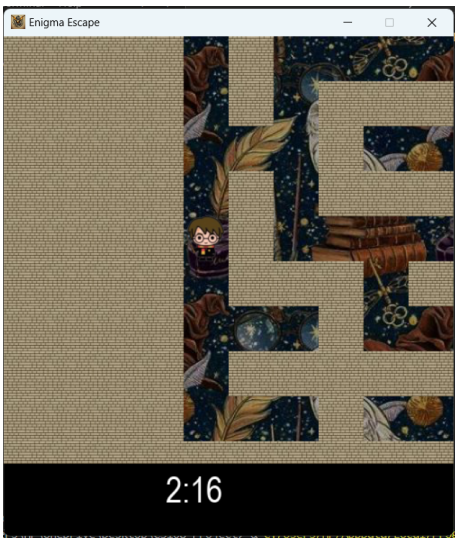
— wallRed.jpg

— logoreduced.jpg

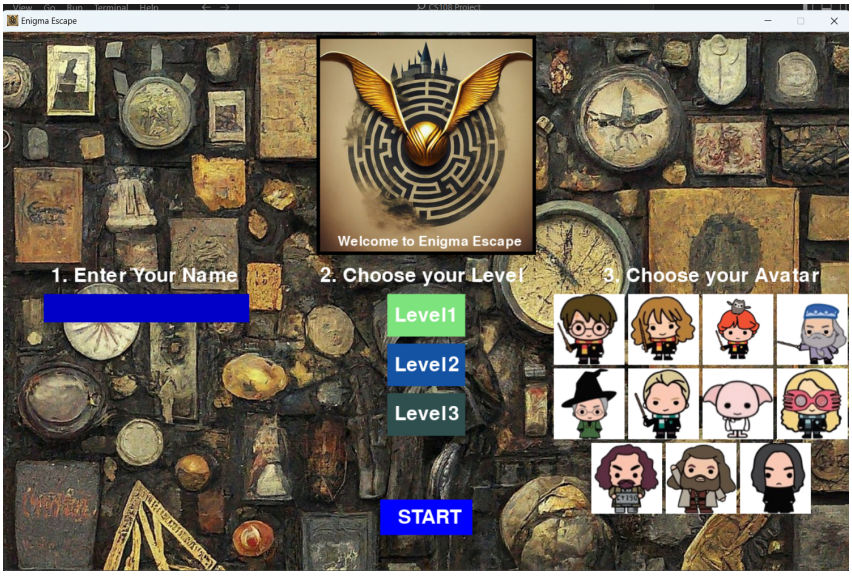
4 Game Features

4.1 Basic Features

1. Restrictive 2D Top View:
   Implemented by first creating a padded maze corresponding to the maze grid and then sliced a part of that array depending on the player's position in the original maze grid



2. Implemented options to choose levels and start a new game



3. Generated random maze and gave it's solution path in Path.txt by using Wilson's Algorithm discussed ahead (5)

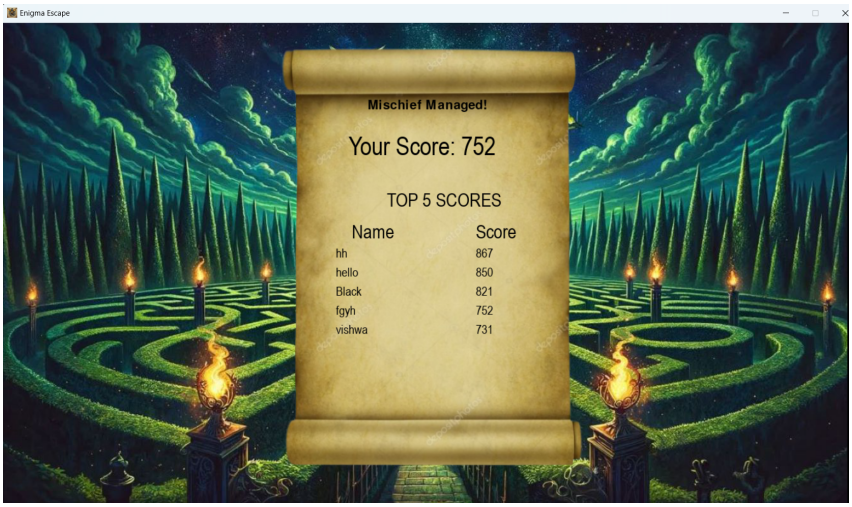
4. Implemented different levels of difficulty by changing maze size and time limit to solve each maze

5. Implemented collision detection by marking wall cells as 0 and path cells as 1 and now allowing the player to go in a direction if the destination cell is 0.

6. Introduced reverse timer and score calculation see 6.4 3

4.2 Advanced Features

1. Maintained a text file from which the 5 highest scores are being extracted

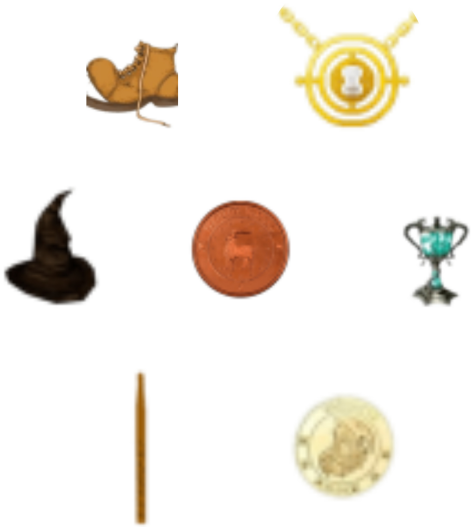


2. Integrated background music which plays while the player is on the playscreen and can be muted by using the volume button

3. Allowed player to select an avatar of their choice out of 11 possible characters

4. Integrated power ups like time-turner (increases time limit by 10s), portKeys (act like vents, aren't on path tiles themselves but teleport the character to a tile in the solution path)

5. Integrated collectibles like galleons(30pts), sickles (20pts), knuts(10pts) and magic wands(100pts).



6. Integrated custom backgrounds and eye-catching maze-design(subject to the vieweer's perception)

4.3 Playing the Game

- Run python game.py
- First enter the player name, then the level and then the character you wish to choose
- Click start on the wait screen whenever you are ready to start the game
- Player movements are controlled using arrow keys, find the end point of the maze before time runs out. If you wish to quit while time is left, press X.
- When time runs out, the results are displayed on the endScreen. If you wish to play again, press R key. For main menu press M key and during the entire game if you wish to quit (as in close the game) press Q.

1. Restrictive 2D Top View:
   Implemented by first creating a padded maze corresponding to the maze grid and then sliced a part of that array depending on the player's position in the original maze grid
2. Implemented options to choose levels and start a new game
3. Generated random maze and gave it's solution path in Path.txt by using Wilson's Algorithm discussed ahead (5)
4. Implemented different levels of difficulty by changing maze size and time limit to solve each maze
5. Implemented collision detection by marking wall cells as 0 and path cells as 1 and now allowing the player to go in a direction if the destination cell is 0.
6. Introduced reverse timer and score calculation see 6.4 3

5 Algorithm

For the maze generation and solving Wilson's Algorithm has been used. The explanation is given below

1. Wilson's Maze Generation Algorithm:
   The algorithm starts by choosing two cells in the maze randomly. A random walk is started from one, which continues until it reaches the other cell. Then, every loop in the walk is removed and the resulting path is cut through the maze. Another random cell is chosen and the random walk done until it reaches a visited cell, after which all the loops are removed and the path is cut. This continues until every cell has been visited. Finally cells which act as walls are represent in the cellGrid as 0 and those acting as free path are represented by 1
2. Wilson Loop Erased Random Walk Algorithm to Solve the Maze:
   The algorithm starts by initializing an empty path and setting the current cell to the starting cell of the maze and then performs a random walk until it reaches the ending cell. During this walk, at each step, it randomly chooses a valid direction to move in while ensuring that it doesn't go out of the maze. As the walk progresses, the visited cells and the direction taken at each step is recorded in the path dictionary.Once the path gets to the ending cell, the solution path is generated by following the recorded path backward from the ending cell to the starting cell and deleting any loops found in the path.

6 Code Implementation

The code has been divided into (*******) files each with their own functions. A brief description of each file is given below

6.1 game.py

This is the file that controls the flow of the program between different functions. It contains the primary game loop

The list of functions in game.py are as follows

1. gameRunner(): Controls flow of the program; calls different display, maze generator, maze solving and various other functions.

6.2 screenChange.py

This is the file that controls the flow of the program between different functions. It contains the primary game loop

The list of functions in screenChange.py are as follows

1. screenChange: Sets the display as per the screen the game is currently on (Start/ Wait & Controls/ Play/ End).
2. topFive: Returns the top five scores to display when the game has ended
3. scoreCalculation: Calculates the score of the player when the game has ended depending on the time taken, collectibles acquired and the level of maze solved

6.3 Wilson.py

This file contains the functions that generates the maze, solves it, create the display grid, allot the collectibles and auxiliary functions that "help" the aforementioned functions. Code was referenced from CaptainFlint's Log Book

The class in Wilson.py are as follows

1. WilsonMazeGenerator: This contains all the functions that generate the maze, solve it, give a grid apt for using in display function. The maze thus generated has ()

6.4 gameLoopDisplay.py

This file contains the function that updates the display as the player mover i.e. it implements the limited top view feature and reverse timer. It also contains the functions which display the collectibles, walls, chooses a random background for each game to bring variety.

The functions in this file are are as follows

1. playLoop: This function that updates the display as the player mover i.e. it implements the limited top view feature. It takes a padded grid and blits it onto the play screen.
2. chooseBG: This function chooses a random background out of 5 possible images for each game to bring variety.
3. collectibles, makeWalls: These functions control the display of collectibles and the walls respectively taking in the padded grid as the argument.

6.5 Players.py

This file contains the functions that control player movement and the collectible boosts' functions.

The class and functions in this file are are as follows

1. class Player: This initialises the player object. It has functions for the string representation and to return the ending information of the game.
2. turnTime: Encodes the functionality of a time turner collectible by increasing the time left to find the end point by 10 seconds.
3. teleport: Encodes the functionality of portkey (hat/ boot/ triwizard cup) collectibles by teleporting the player to a point which is in somewhere in the middle of the solution path ordered list of the maze. Note; The portkeys themselves do not exist on cells that are a part of the solution path.
4. playerMove: This function detects collisions while the player is moving, updates the player's position and controls acquisition of collectibles.

6.6 utils.py

This file contains the constants such as screen sizes, colours and paths to images.

6.7 Path.txt

The solution path is over written in this file for each run of the game.

6.8 scores.txt

The scores of all past players are stored in this file to later find out the highest five scores.

7 Project Experience

I would first like to thank our instructor, Prof. Chebrolu for teaching a somewhat non-enaging subject in a way that pushed us to attend the lectures. I would also like to that the project in-charge TA and all other SSL TAs, for always being there to solve our doubts no matter how trivial they were. I would also like to thanks my peers for their feedback during the testing phase of the game.

All that being said, this project was somewhat of a challenging project to begin with considering that I had no prior experience in pygame module but nonetheless, it was a fun experience learning it from scratch and building a game from it.

When the project started I kept thinking of good it would be when it ended but now that I am at the end, I keep thinking of all the other things that I could do with some more time. Thanks for such an interesting assignment!

8 References

1. Wilson's Algorithm Wikipedia Page
2. Project In-Charge TA's Past SSL Project
3. GeeksforGeeks Pygame Tutorial
4. Pygame Documentation
5. CaptainFlint's Log Book - Wilson's Algorithm Code Reference
6. Images generated using Gemini AI