Lost in the Maze : A Pygame Project

Shivam Singh Yadav

May 20, 2024

Contents

1 Introduction

The objective of this project is to design and implement 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 from the starting point to the end point while avoiding obstacles and traps. The project involves creating various levels of increasing difficulty, implementing collision detection, designing intuitive user interfaces, and integrating sound effects and visual elements to enhance the gaming experience.

Figure 1: MY GAME

My game is Named Labyrinth Leap

2 Basic Requirements

The basic requirements for the project include:

1. Designing an intuitive user interface for the game menu, gameplay screen, and game over screen.
2. Generating random mazes for each level.
3. Implementing player controls for navigating the maze using keyboard inputs.
4. Incorporating collision detection to prevent the player from passing through walls.
5. Introducing features such as score, timer, and lives to enhance gameplay dynamics.

6. Implementing a camera system so that Players should only be able to see some part of the maze in their vicinity.

2.1 User Interface

The game has an interactive user interface, where at the top left the Conner display's Player Name, The interface has has 3 button are center align "PLAY", "LEVELS", "QUIT" where "PLAY" starts the game with Level 1, "LEVELS" Takes you to another screen to chose Level 1,2,3 or the bonus level, "QUIT" exits the game. There are two more buttons left side one for Scoreboard and Right side one for turning music on or Off.

Figure 2: UI

2.2 Maze Generating

The Maze Generating algorithm initializes a 2D array representing the maze with walls. It starts from a cell and maintains a stack of visited cells. In each iteration, it finds possible moves (unvisited adjacent cells) from the current cell. If moves exist, it randomly selects one, knocks down the wall between the cells, marks the new cell as a path, and pushes it to the stack. If no moves are available, it backtracks by popping from the stack. This process continues until the stack is empty, carving out paths through the maze. The random move selection ensures maze randomness, while backtracking ensures connectivity. Thus creating a Maze. Everything related to Maze handling is done in Maze Generator Module(made by me).

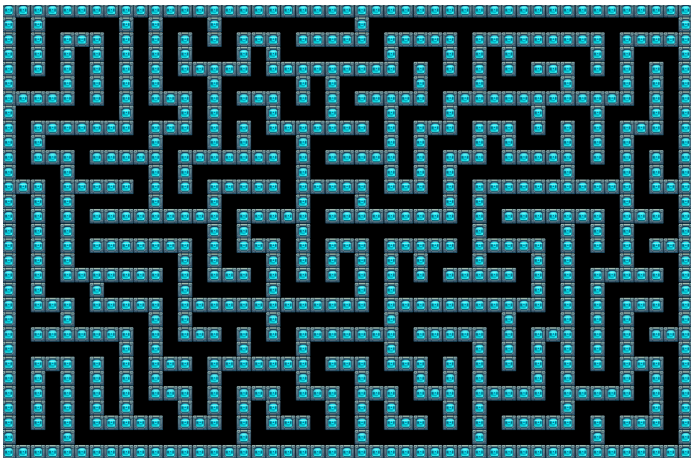

Figure 3: Maze

2.3 Player Controls

Everything Related to the player is created in Player Class where its movements are defined. The player moves Forward when "W" is pressed, Backward when "S" is pressed, Left when "A" is pressed and Right when "D" is pressed. where a const velocity is added/subtracted in x or y axis to move respectively.

2.4 Collision Detection

The collision detection between player and the wall of the maze is handled in the "Check collision" function where the player and the wall bot are assumed as a rectangle and when these rectangles collide the velocity becomes zero, Different collisions in the game are also handled in their respective function using the same logic.

2.5 Features such as score, timer, and lives

Score is your achievement in your game, more you have is better, when you collect Gem while playing the game your score adds by one. Timer is just the time limit in which you have to solve the maze in Oder to go to the next level. Lives are the health of the player when ever you collide with the player your live id decreased by 1, by default you have 3 hearts and at max you can have 4.

Figure 4: status

2.6 Camera System

The game has an focused camera, where all the maze is in Dark only the vicinity around the player is visible, this was done using masking technique. the visible region moves with player keeping the player at the center all the time.

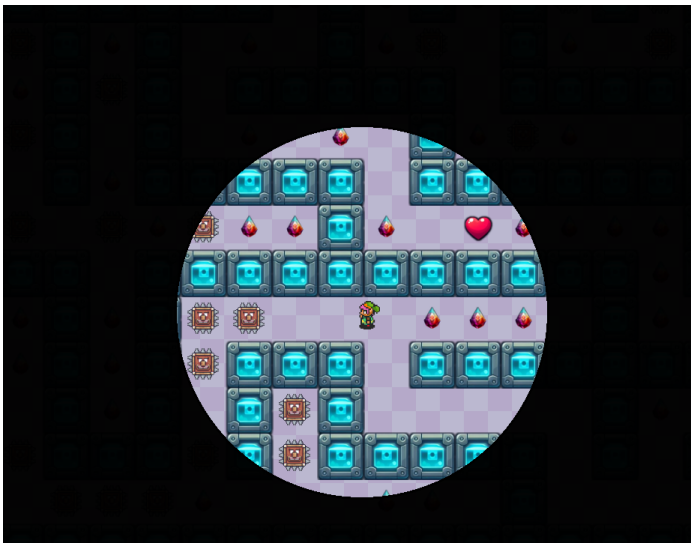

Figure 5: focus camera

3 Modules

The project utilizes the following external modules:

- **Pygame**: A popular Python library for game development, used for creating and managing the game window, handling user input, rendering graphics, and playing sounds.
- **NumPy**: A library for scientific computing in Python, used for generating and manipulating the maze data structures.
- **OS**: A built-in Python module for interacting with the operating system, used for file operations and path handling.
- **Random**: A module used to generate random numbers.

4 Directory Structure

The project directory contains the following files:

- **game.py**: The main script that handles the game logic, user interface, and game loop and runs the game.
- **Maze_genrator.py**: A module containing functions for generating random mazes, solving mazes, and manipulating maze data structures.
- **pg_buttons.py**: A module containing a custom class for creating and managing buttons in the user interface.
- **Assets**: A directory containing various assets such as images, fonts, and sound files used in the game.
- **Solution.txt**: A file generated during runtime, containing the solution path for the current maze.

5 Running Instructions

To run the game, follow these steps:

1. Install the required dependencies (Pygame and NumPy) using pip or another package manager.
2. Navigate to the project directory in the terminal or command prompt.
3. Run the `game.py` script using the Python interpreter.
4. The game will start with the start screen, enter you name and press "ENTER".
5. Use the mouse to navigate through the menus and select options.
6. During gameplay, use the W,A,S,D keys to move the player character through the maze.
7. Press the ESC key to return to the main menu from any level.
8. Press the ENTER key when the player reaches the end point to proceed to the next level.
9. Press the backtick (`) key to display the solution path (cheat code).

6 Customization

In addition to the basic requirements, the following advanced features have been implemented in the game:

- **Graphical User Interface**: The user interface includes animated elements, background music, and hover effects on buttons, Amazing Graphics enhancing the overall aesthetic and user experience.

- **Starting screen**: When the game is started a starting screen shows up where you have to enter your player name and by default the player name is jasper stone.

Figure 6: Starting Screen

- **Scoreboard System**: The scoreboard screen display's the name and score of different players, in descending Oder of the score.
- **Player Animation**:Player movement and all the actions are animated.
- **Graphics**: The game is filled with amazing graphics to make the game more appealing.
- **Particle Effects**:When ever the player collides with the obstacle's a exploding effect is shown.
- **Music**: Background Music is Added to the game, and custom Music can also be played by passing an argument while executing the game, (python game.py Music_name.mp3)
- **PowerUPs**: There is one power up in the game named "HEART" which increases your health by 1 and a max up to 4.
- **Cheat Code**:There is a cheat code in the game which when pressed shows the path for a fraction of a second. The Cheat Code is (‘) or the tilda button.

7 Project Journey

This project has been an enriching experience, allowing me to enhance my skills in Python programming, game development using Pygame, and algorithm implementation. One of the notable challenges I encountered was generating random mazes that were solvable and adjusting the maze difficulty based on the level. I overcame this challenge by implementing a depth-first search algorithm to generate the maze and ensure the presence of a solution path.

Another challenge was designing an intuitive and visually appealing user interface. I tackled this by incorporating animations, background music, and hover effects, creating an immersive gaming experience.

Overall, this project has deepened my understanding of game development, problem-solving skills, and the application of algorithms in practical scenarios.