Maze Game Report

B.Abhinav Chowdary

April 28, 2024

Abstract

This report briefly explains my journey making maze game.

Contents

1 Problem Statement

I believe all of us have played maze games at some point in our lives. I personally have played them a lot, usually in newspapers and magazines. The objective of the game is to navigate through a maze from the starting point to the ending point.
The maze is a complex network of paths and walls, and the player must find the correct path to reach the end point while avoiding obstacles and traps. The game requires strategic thinking,problem-solving skills, and quick reflexes to complete each level successfully. The aim of this project is to design and implement a variant of this 2D maze game using Pygame, a popular Python library for game development. The game will consist of a single player-controlled character navigating through a maze with a restrictive 2D topview from the starting point to the end point.
The project will involve creating various levels of increasing difficulty, implementing collision detection, designing intuitive user interfaces etc.

2 Modules Used

- Pygame
- Sys
- Random

3 How The Game Works

3.1 Running the game

Use python3 game.py or python game.py or click the run button if you are using VS code.
No arguments are expected

3.2 Navigation

The maode of navigation used in this game is **Buttons**.
The navigation is taken care by using a **run_game** function.
The run_game function uses a method similar to system path management as it defines path variable and uses various return values for efficient navigation.
For example: When you run game.py **main_function** is run and your path will be ["main_screen"].
Say you click **Play** button. The **main_screen** function return something called action that is **"level_screen"** here and your path is updated to ["main_screen","level_screen"]
Clicking **Back** removes the last element in path.

Each and every screen has a return value and the values are only returned when something is clicked as only then will the loop break.

3.3 Various Buttons

The main screen consists of 4 buttons namely **Play, Quit, HighScores, Help**.

- Play takes you to the level screen where you can select the level you want to play.
- Quit terminates the game.
- Help opens the help screen where you can read the rules of the game
- HighScores open highscores screen where you can see the **TOP 5** scores in each Level.

Back button is present in various screens and it takes you back to the previous screen.Back button in game over screen takes you to level screen

3.4 Controls and Rules

In each level you have a ninja themed warrior who is struck in a maze. There is one start and one end only.
Controls

- Keyboard UP moves the player up.
- Keyboard DOWN moves the player down.
- Keyboard LEFT moves the player left.
- Keyboard RIGHT moves the player right.
- **Simultaneous pressing of keys is allowed.When you press both UP and LEFT the player moves diagonally if possible.If only one of the movements is possible the player moves along that feasible movement till the other movement becomes feasible after which diagonal movement resumes.**
- Pressing 3 or more is meaning less as opposite movement get cancelled leading to uni directional or no movement at all.

Rules

- There is only one start and one end clearly marked.The goal is to reach the end point before the timer runs out.

- Reaching end point requires collection of all the golden keys in the maze. The amount may differ from level to level.You will know that you are good to go if the KEYS button in the bar above the maze turns **golden from silver**.
- You can also collect gems in the mean time which enhance your score.
- Your Final score is a function of both the amount of time left as well as the number of gems collected.
- Be careful as staying long to collect gems need not necessarily increase your score.
- High score is displayed in game over screen if your score is greater than atleast one of the previous top 5.

4 Basic Features Implemented

1. Design of an intuitive user Interface through buttons.

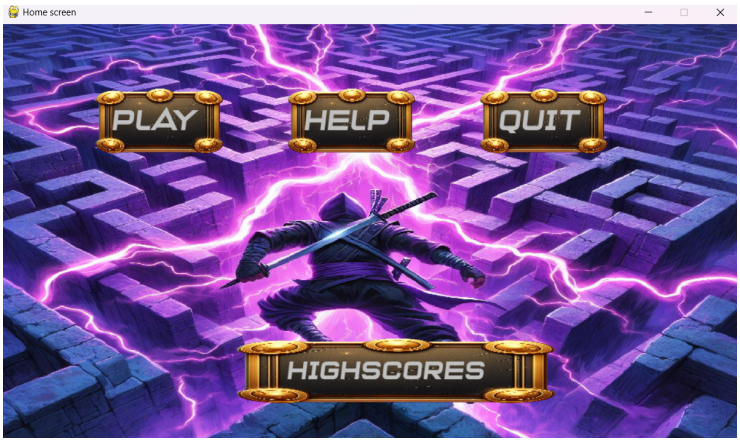

Figure 1: The Main Screen

2. Generation of a random maze using **PRIMS ALGORITHM** that is present in the prims_maze_algo.py file.This Algorithm takes the no of rows and columns of the maze and returns a double array constituting of 'w','c' representing wall and cell respectively.
3. Updating the solution path of the maze in the path files each time a level is played.The solution path is generated through a **Basic Recursive Algorithm** present in maze_solver.py .The solution path comprises of up,down,left,right texts each in a newline.

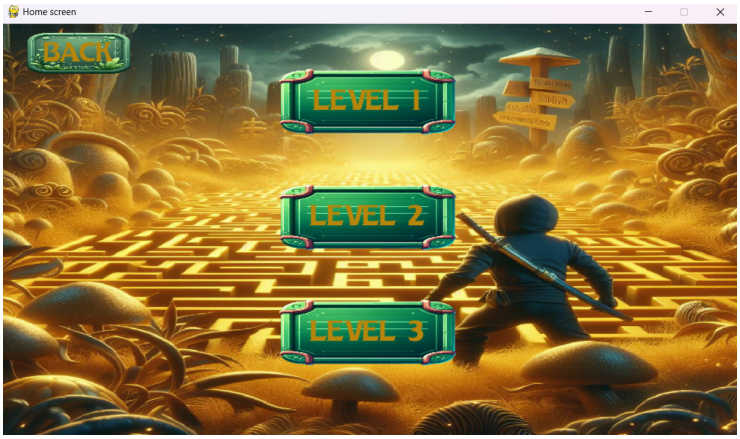

Figure 2: The Level Screen

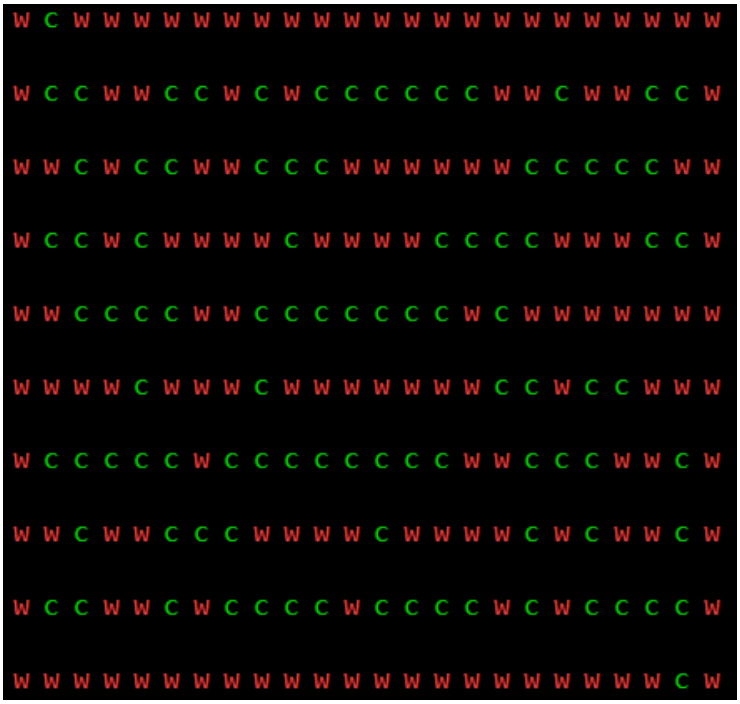

Figure 3: Prims Maze

Figure 4: GamePlay Screen

4. Implementing **Restricted View**.Fraction of the Maze displayed depends on level but is always less than or equal to 1/4.This is done by creating a maze not on the screen but on a surface and then selectively cutting part of this maze to blit on the screen ensuring a restricted view.
5. Increasing difficulty levels based on complexity,time and more keys to be collected.
6. Implemented Key Board Player controls through Up,Down,Left,Right keys.
7. Implemented feautures such as **Score, Time, Keys Collected** .

5 Folders and Files Structure

- **Classes:** Consists of the button class in button.py and player class in Player.py .
    1. **Button:** consists of all the information regarding a button and provides various methods to check mouse_clicking,mouse_hovering and to also draw the button.We can give smooth hovering effects using these functions.
    2. **Player:** consists of all the information regarding the player(ninja) such as speed,coordinates,images used etc.
       It contains various methods such as **collision_checker to implement Collision detection** , **image_update** to update the image based on the input keyboard keys, **draw_player** to draw him.
       It also includes methods to check key collection and gem collection.

- **Fonts:** Consists of the various gaming fonts that were used in the making of this game.
- **highscores:** Consists of files that keep track of the high scores in various files.
- **images:** Consists of all the images and templates used in the making of this game.
- **Ninja_sprites:** Consists of the sprites used to animate motion of the player.
- **paths:** Consists of 3 files which keep track of the current running levels solution path.
- **highscore_func.py:** Consists of functions to read and update high score files.
- **game_screen_maker.py:** Consists of a level building function which takes various arguments and generates the game screen . These arguments can be controlled to change difficulty.
- **prims_maze_algo.py:** Consists of the Prims algorithm for maze building.
- **maze_solver.py:** Consists of a recursive algorithm to find solution path and update it in the path files.
- **game.py:** Consists of the main game running code as well as various other screen design.

6 Advanced Feature Implemention

1. **A High Score Leaderboard** to track players perfomances across multiple game sessions.
2. **A background music** during gameplay to enhance the gaming experience.
3. **Button Hovering effects** to enhance the user interface.
4. **Animated player** to enhance the gaming experience.
5. **Collectibles** such as keys and gems to make the game more engaging.

Figure 5: Highs Score Leaderboard

7 References

1. Prims algorithm taken from `https://github.com/OrWestSide/python-scripts/blob/master/maze.py`
2. Maze solving algorithm taken from `https://thecleverprogrammer.com/2021/01/26/maze-solver-with-python/`
3. Images are generated using **Microsoft Bing**
4. Ninja sprites are taken from `https://www.gameart2d.com/ninja-adventure---free-sprites.html`
5. Fonts are taken from `https://www.fontspace.com/category/gaming`
6. The Background music is taken from `https://archive.org/details/tvtunes_5700`
7. Wall and cell images are taken from `https://timberwolfgames.itch.io/amazing-maze-sprites`