OOP Final Report: Maze Game

張育瑋
Department of Electrophysics
National Yang Ming Chiao Tung University
Hsinchu City, ROC
brain4416@gmail.com

洪薇欣
Department of Information Management and Finance
National Yang Ming Chiao Tung University
Hsinchu City, ROC
velda.mg10@nycu.edu.tw

***Abstract*—This is a maze game with three modes: easy, normal and hard. You can choose to compete with the computer or not. Also you can choose single player mode or multiplayer mode. There are three methods being used to generating a maze : DFS algorithm, Prim's algorithm and recursive division algorithm. And we also using two method to solve the maze by using depth-first search algorithm and stack-based algorithm. And the game using the pygame draw function to generate everything on the game screen : the player, computer, maze, button, etc.**

I. Introduction

This is a maze game with three modes : easy, normal and hard. You can choose to compete with the computer or not. Also you can choose single player mode or multiplayer mode.

II. OOP design

A. Class Diagram

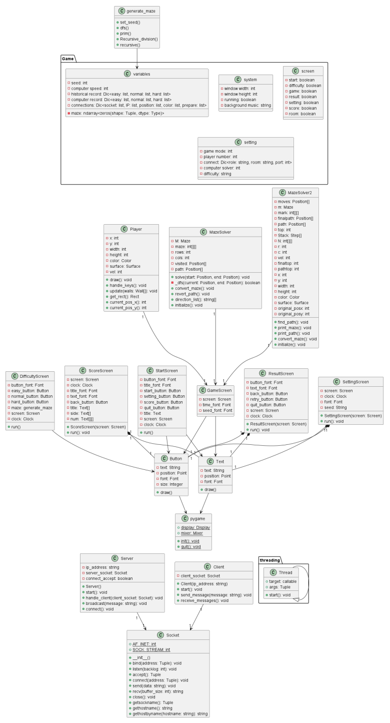


Fig. 1. Class Diagram

B. Code

Below are the explanations of the code for the game :

- The first part of the code provides a structure for organizing and managing the game's system, screens, variables, and settings. It serves as a template or blueprint for creating and controlling the game's behavior and state during its execution.
- Class generate maze:

  This class is for generating the maze.There are three ways to generate the maze. We set the method to generate the maze using the DFS algorithm as the easy mode,the method to generate the maze using the Prim's algorithm as the normal mode, and the method to generate the maze using the recursive division algorithm as the hard mode. The mazes generated by the three methods have similar structure, which are represented by the presence or absence of walls in the four directions (left, up, right, down) and a visited flag.

  All three methods share a common set_seed() method that sets the random seed for generating mazes based on the value of Game['variables']['seed'].The function set_seed ensures that a random seed is set for the game if it hasn't been set previously. The random seed is used to initialize the random number generator, allowing for reproducible random behavior in the game.

  The function dfs implements a depth-first search algorithm to generate a maze represented as a NumPy array. The algorithm explores different paths in the maze, backtracks when necessary, and marks walls and passages accordingly until it has visited all cells.The algorithm starts from the top-left corner and uses a stack-based history to carve passages by randomly selecting unvisited neighbors.

  The function prim implements Prim's algorithm to generate a maze represented as a NumPy array. The algorithm starts with a random position, selects neighboring positions, and adds them to a history list. It continues to explore randomly chosen positions from the history list until all positions have been visited. The walls and passages in the maze are marked based on the chosen directions during the exploration process.It follows a similar approach to dfs() but uses a different strategy for selecting neighbors.The algorithm starts with a single cell and repeatedly adds neighboring cells to the maze while ensuring the maze remains connected.

The function Recursive_division and recursive implements the recursive division algorithm to generate a maze represented as a NumPy array. The algorithm recursively divides the maze into smaller sections by creating walls and passages. It continues the division process until the sections reach a minimum size. The recursive calls ensure that the maze is divided in a fractal-like pattern.It starts with a full grid and recursively divides it into smaller sections by creating walls.The division process is random, and it continues until the desired maze structure is achieved.

- Class StartScreen :

  The class sets up and runs the starting screen of a maze game. It initializes buttons and text objects, handles user input events, updates the screen accordingly, and controls the frame rate.
- Class DifficultyScreen :

  The class sets up and runs the difficulty selection screen of a maze game. It initializes buttons for different difficulty levels, handles user input events, updates the screen accordingly, and controls the frame rate. It also generates a maze using one of the maze generation algorithms based on the selected difficulty level.
- Class GameScreen :

  The class sets up and runs the game screen of a maze game. It handles user input events, updates the game state, and renders the maze, players, and other elements on the screen. It also manages the timing and movement of the computer solver if activated.
- Class ResultScreen :

  The class manages the display and interaction of the result screen, showing the player's time, minimum time, computer time (if applicable), and providing buttons for navigation and game restart.
- Class SettingScreen :

  The class manages the display and interaction of the settings screen, allowing the player to select game mode, enter a seed value, choose a role (server or client) for multiplayer, and enter a room address. The class handles the transitions between screens and updates the game settings accordingly.
- Class ScoreScreen :

  The class manages the display and interaction of the score screen, showing the scores for different difficulty levels. It calculates and displays the average and standard deviation of the scores and provides a button to return to the start menu.
- Class RoomScreen :

  The class represents the screen for a game room. It initializes various attributes such as the screen itself, clock, code, font, and buttons. The run method is the main loop of the room screen where it checks the role (server or client) and performs actions accordingly. If the role is 0 (server), it starts a server and handles events related to server functionality. If the role is 1 (client), it tries to connect to the server and handles events related to client functionality. The run method continuously updates the screen, handles user input, and displays relevant information.

  The button method initializes the buttons used in the room screen, such as the difficulty selection buttons, ready button, undo button, and start button. The ip_to_id and id_to_ip methods are utility methods used to convert IP addresses to unique IDs and vice versa. The color_check method updates the color of buttons and text based on the selected difficulty and the readiness status of the players.
- Class Player :

  The class handles the movement and collision detection of a player object in the game. It updates the player's position based on key presses, checks for collisions with walls, and provides methods to retrieve the player's position.
- Class MazeSolver :

  This class is for showing the solution of the maze. If the player press the solution button, the solution will be shown. The class provides methods to solve a maze using the depth-first search algorithm, convert the maze representation, retrieve the solved path in simplified coordinates, and initialize the solving process.
- Class MazeSolver2:

  This class is for having a competition with the computer. If the compete with computer setting is on, the computer will start after 30 seconds.The class provides methods to solve a maze using a stack-based algorithm, convert the maze representation, print the maze, print the visited cells, and initialize the solving process.
- Class Button:

  The class provides a convenient way to create and draw buttons in a graphical user interface. It allows you to set the text, position, font, and width of the button, as well as change its color.
- Class Text :

  The class provides a convenient way to create and draw text on a graphical user interface. It allows you to set the text content, position, font, and color, and provides a method to draw the text onto a given surface.
- Class Wall :

  The class provides functionality to create walls with specified positions, dimensions, and colors. It can draw the wall on a surface and check for collisions with other objects, such as a player.
- Class Server :

  The class represents the game server. It has an __init__ method that initializes the server's IP address, server socket, and connection acceptance flag. The start method creates and starts the server socket, binds it to the IP address and port specified in the game settings, and listens for incoming connections. The handle_client method is a thread target that handles individual client connections. It continuously receives messages from the clients and updates the game state accordingly. The broadcast method sends a message to all connected clients except the server socket.

The connect method is another thread target that accepts client connections and adds them to the list of connections. It spawns a separate thread to handle each client connection. The Server class provides the functionality to handle multiple client connections simultaneously.

- Class Client :

  The Client class represents a game client. It has an __init__ method that initializes the client socket and connects it to the server's IP address specified in the game settings. The start method starts a separate thread to receive messages from the server. The send_message method sends information to the server. The receive_messages method continuously receives messages from the server and updates the game state accordingly.
- The last part of the code sets up a Pygame window, initializes the necessary modules, loads and plays background music, and runs a game loop that updates and displays different screens based on the state of the game stored in the Game dictionary.

C. Comparison

- Three methods to generate the maze :

  The choice between DFS, Prim's algorithm, and the recursive division algorithm depends on the desired characteristics of the maze. DFS offers randomness and a more interconnected maze structure, Prim's algorithm generates deterministic mazes with shorter paths, and the recursive division algorithm creates mazes with a structured pattern. The selection can be based on factors like desired difficulty level, aesthetic preferences, or specific gameplay requirements.

  1) DFS Algorithm :
     a) Approach: DFS is a graph traversal algorithm that starts from an initial cell and explores as far as possible along each branch before backtracking. In the context of maze generation, DFS creates a maze by carving out passages between cells.
     b) Characteristics:
        i) Randomness: DFS can generate random mazes as it explores cells in a random order.
        ii) Path length: DFS-generated mazes tend to have long and winding passages, which can result in longer paths between two points.
        iii) Difficulty: DFS-generated mazes can be relatively easier to solve since there are often multiple paths to reach the goal.
        iv) Maze structure: DFS-generated mazes often have a more interconnected structure with loops and dead-ends.
  2) Prim's Algorithm :
     a) Approach: Prim's algorithm starts with a single cell and repeatedly adds neighboring cells to the maze, creating a minimum spanning tree. This process continues until all cells are connected.
     b) Characteristics:
        i) Deterministic: Prim's algorithm generates deterministic mazes with a unique solution.
        ii) Path length: Prim's algorithm tends to create mazes with shorter and more direct paths between two points.
        iii) Difficulty: Prim's algorithm-generated mazes can be relatively harder to solve as there is often a single optimal path.
        iv) Maze structure: Prim's algorithm-generated mazes often have a more tree-like structure with fewer loops and dead-ends.
  3) Recursive Division Algorithm :
     a) Approach: The recursive division algorithm starts with a full grid and recursively divides it into smaller sections by creating walls. This process continues until the desired maze structure is achieved.
     b) Characteristics:
        i) Deterministic: Recursive division algorithm generates deterministic mazes.
        ii) Path length: The path length in recursive division algorithm-generated mazes can vary depending on the placement of the dividing walls.
        iii) Difficulty: The difficulty of solving recursive division algorithm-generated mazes can vary based on the maze structure and the positions of walls.
        iv) Maze structure: Recursive division algorithm-generated mazes often have a more structured pattern with horizontal and vertical dividing walls.
- Two methods to solve the maze:

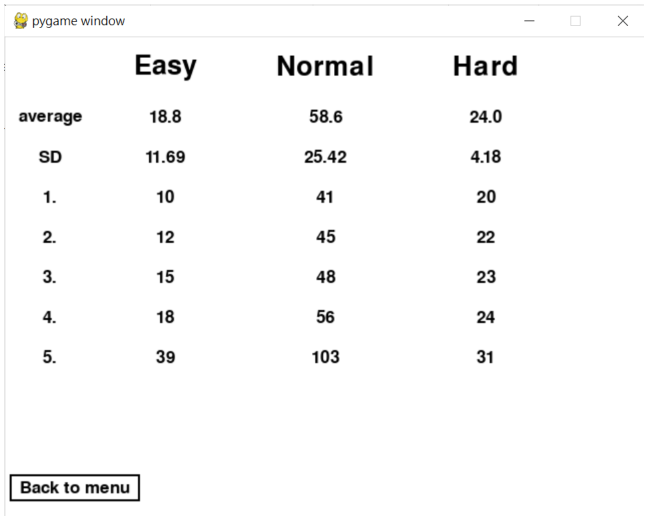
  DFS is a simple and intuitive algorithm implemented recursively, but it may have limitations in terms of efficiency and memory usage. A stack-based algorithm provides more control and avoids recursive call stack overflow but requires explicit management of the stack and additional memory. The choice between these approaches depends on the specific requirements of the maze-solving problem and the characteristics of the maze itself.

  1) DFS Algorithm :
     a) DFS is a recursive algorithm that explores paths as deeply as possible before backtracking.
     b) In the context of maze solving, DFS starts at the initial cell and recursively explores neighboring cells until it reaches the goal cell or encounters a dead end.

c) DFS maintains a stack implicitly through the call stack of recursive function calls.
d) Advantages :
   i) Simple to implement recursively.
   ii) It can find a solution quickly if the path to the goal is relatively short.
e) Disadvantages :
   i) May get stuck in an infinite loop if there are cycles in the maze.
   ii) Not guaranteed to find the shortest path.
   iii) Can consume a large amount of memory in deeply nested recursive calls for large mazes.

2) Stack-Based Algorithm :
   a) A stack-based algorithm uses an explicit stack data structure to track the cells visited and explore the maze.
   b) Instead of using recursive function calls, the algorithm explicitly maintains a stack to keep track of the current cell and its neighbors.
   c) The algorithm pops a cell from the stack, explores its neighbors, and pushes unvisited neighbors onto the stack.
   d) The process continues until the goal cell is reached or the stack becomes empty.
   e) Advantages :
      i) Allows more control over the exploration process.
      ii) Avoids the risk of recursive call stack overflow for very large mazes.
   f) Disadvantages :
      i) Requires additional memory to maintain the stack.
      ii) Still subject to the same limitations as DFS, such as potentially getting stuck in infinite loops or not guaranteeing the shortest path.

III. Experiment Result



	Easy	Normal	Hard
average	18.8	58.6	24.0
SD	11.69	25.42	4.18
1.	10	41	20
2.	12	45	22
3.	15	48	23
4.	18	56	24
5.	39	103	31

Fig. 2. Experiment Result

IV. Appendix

A. Outlook of the game

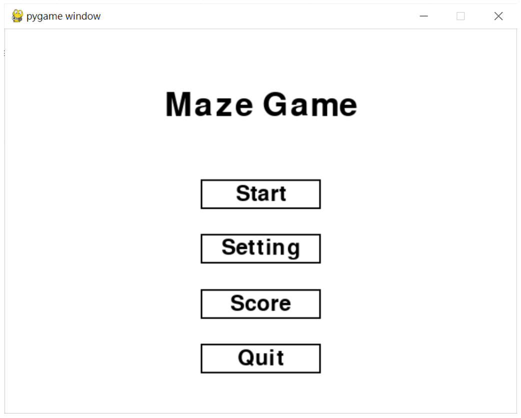


Fig. 3. Main Menu

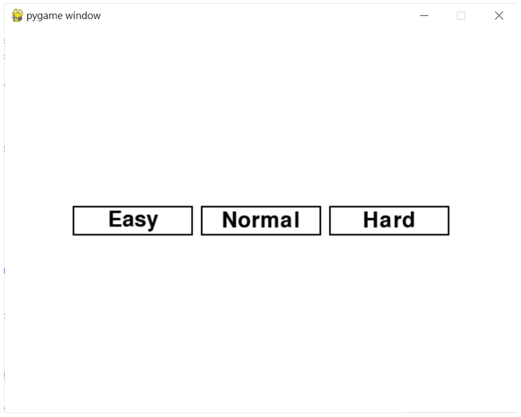
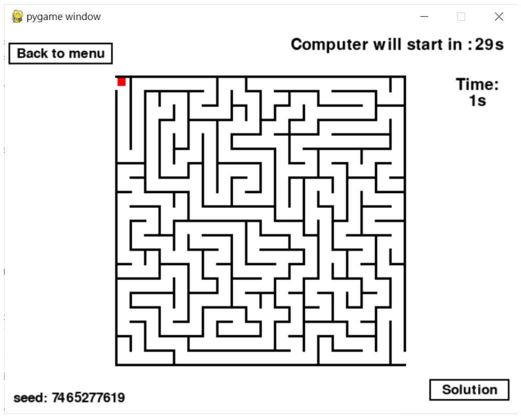


Fig. 4. Menu For Mode

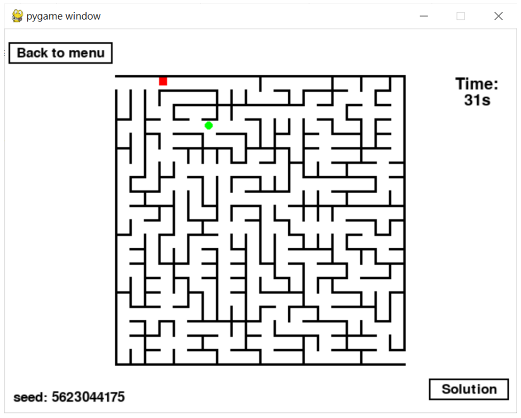


Computer will start in :29s

Time:
1s

seed: 7465277619

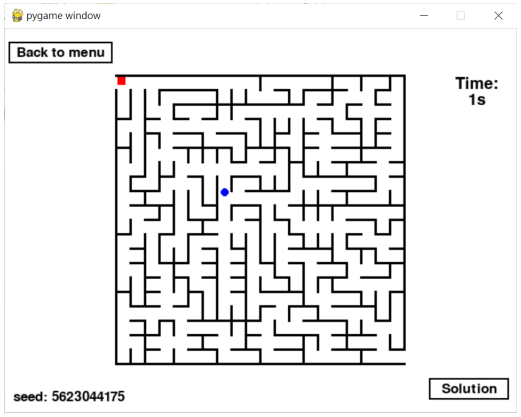
Fig. 5. Main Game



Time:
31s

seed: 5623044175

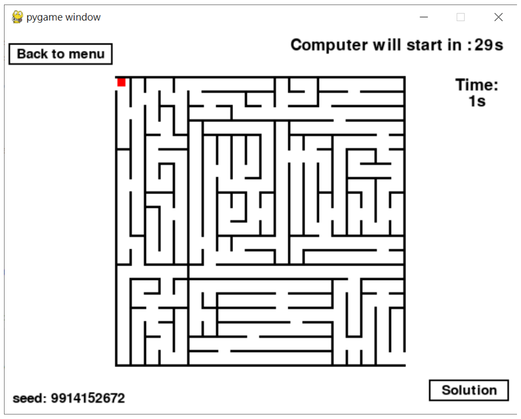
Fig. 6. Compete With the Computer



Time:
1s

seed: 5623044175

Fig. 7. Show the Solution

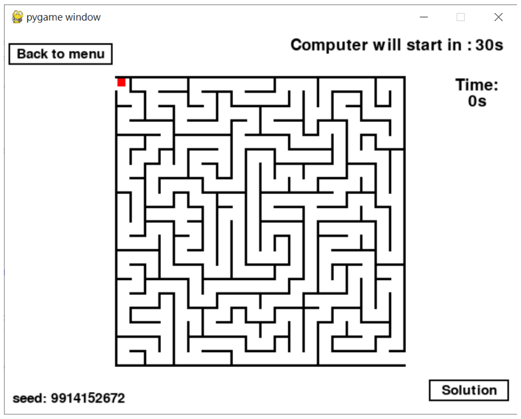


Computer will start in :29s

Time:
1s

seed: 9914152672

Fig. 8. Easy Mode

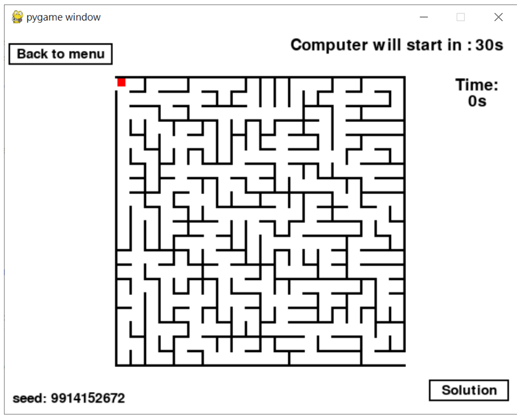


Computer will start in :30s

Time:
0s

seed: 9914152672

Fig. 9. Normal Mode

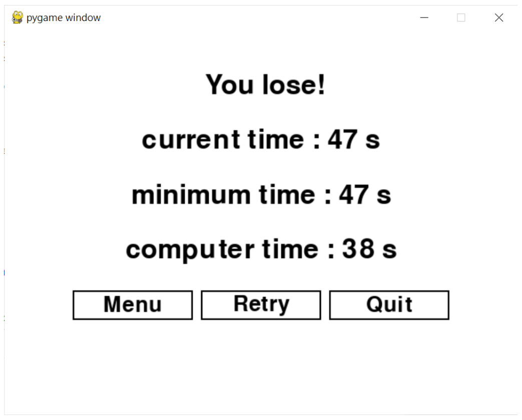


Computer will start in :30s

Time:
0s

seed: 9914152672

Fig. 10. Hard Mode


pygame window

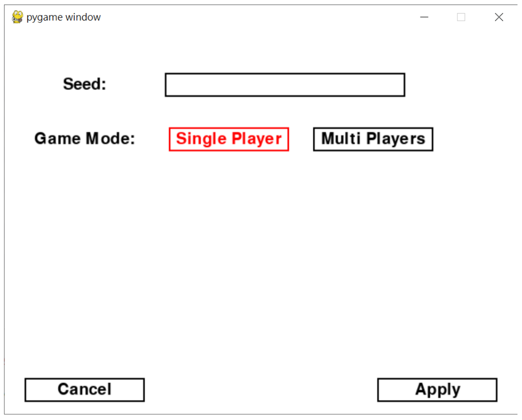
You lose!

current time : 47 s

minimum time : 47 s

computer time : 38 s

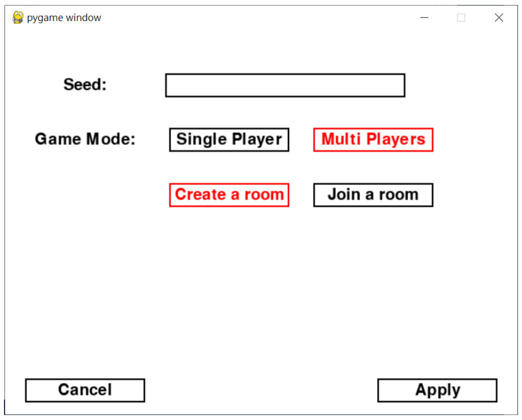
Fig. 11. End Game - Score


pygame window

Seed:

Game Mode: [✓ Single Player] Multi Players

Fig. 12. Setting

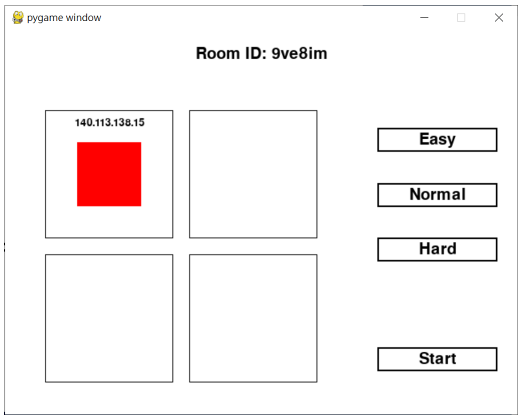

pygame window

Seed:

Game Mode: Single Player [✓ Multi Players]

[✓ Create a room] Join a room

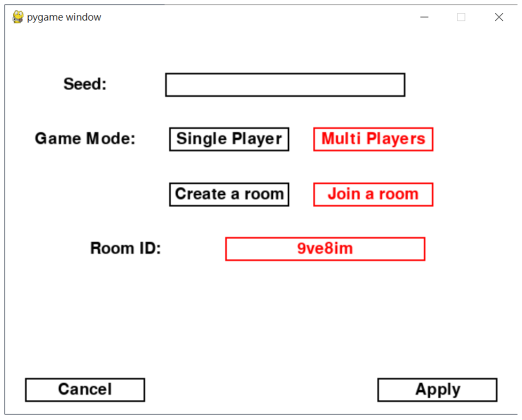
Fig. 13. Multiplayer: create a room


pygame window

Room ID: 9ve8im

140.113.138.15

Fig. 14. Room: server


pygame window

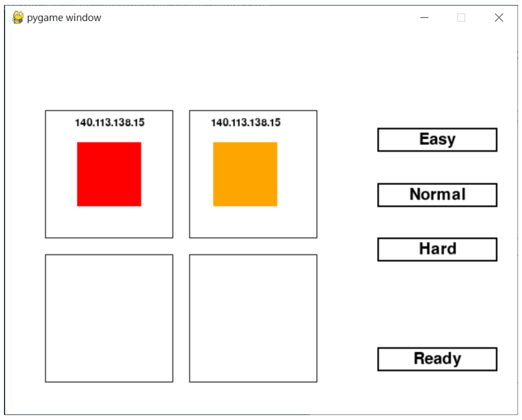
Seed:

Game Mode: Single Player [✓ Multi Players]

Create a room [✓ Join a room]

Room ID: 9ve8im

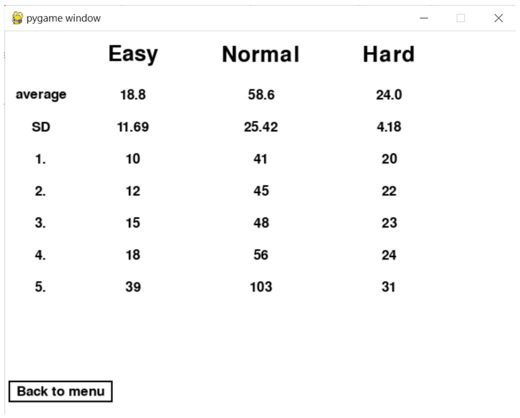
Fig. 15. Multiplayer: join a room


pygame window

140.113.138.15

140.113.138.15

Fig. 16. A room include multi players



	Easy	Normal	Hard
average	18.8	58.6	24.0
SD	11.69	25.42	4.18
1.	10	41	20
2.	12	45	22
3.	15	48	23
4.	18	56	24
5.	39	103	31

Fig. 17. Score

B. GitHub link

Maze Game - https://github.com/veldahung/group21_project

References

[1] ChatGPT