

## Contents

Project Description .....	4
Modifications.....	5
Collaboration .....	6
Student 1: Diana Ali Silva.....	6
Diana's Screenshots.....	7
Student 2: Carlos David Urrea Cabello.....	9
Carlos's Screenshots.....	9
Student 3: Georgii Taisaev .....	11
Student 4: Muslim Muradov .....	14
Muslim's Screenshots .....	15
Planned Improvements for Next Version .....	18
Link to our repository: .....	18
References .....	19

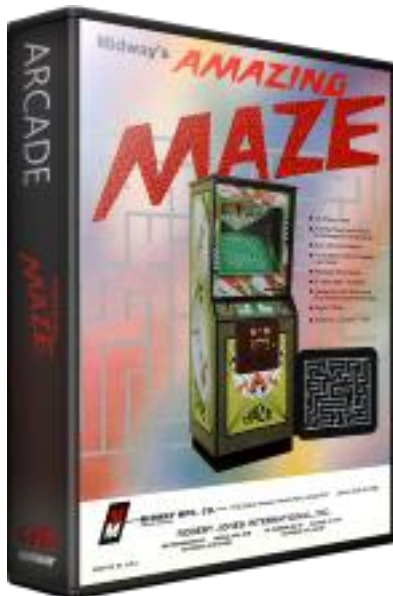


Student name:	Student 1: Diana Ali Silva				
	Student 2: Carlos David Urra Cabello				
	Student 3: Georgii Taisaev				
	Student 4: Muslim Muradov				
Student number:	Student 1: 3123965				
	Student 2: 3125350				
	Student 3: 3119655				
	Student 4: 3128794				
Faculty:	Computing Science				
Course:	BSCH/BSCO/EXCH		Stage/year:	2	
Subject:	Software Development 2				
Study Mode:	Full time	<input checked="" type="checkbox"/>		Part-time	
Lecturer Name:	Gemma Deery				
Assignment Title:	Review 1				
Date due:	03 April 2025				
Date submitted:	03 April 2025				
<b>Plagiarism disclaimer:</b> <i>I understand that plagiarism is a serious offence and have read and understood the college policy on plagiarism. I also understand that I may receive a mark of zero if I have not identified and properly attributed sources which have been used, referred to, or have in any way influenced the preparation of this assignment, or if I have knowingly allowed others to plagiarise my work in this way.</i>  <i>I hereby certify that this assignment is my own work, based on my personal study and/or research, and that I have acknowledged all material and sources used in its preparation. I also certify that the assignment has not previously been submitted for assessment and that I have not copied in part or whole or otherwise plagiarised the work of anyone else, including other students.</i>  <b>Signed:</b> _____ <b>Date:</b> _____					

**Please note:** Students **MUST** retain a hard / soft copy of **ALL** assignments as well as a receipt issued and signed by a member of Faculty as proof of submission.

## Project Description

The 2D Multiplayer Maze Game is a Java application inspired by the classic arcade games *Amazing Maze* (1976) and *Maze Craze* (1980). The game challenges two players to race through a randomly generated maze to reach the exit first. Each maze is built using a recursive backtracking algorithm to ensure it is always solvable and offers a fresh layout in every round.[1,2]

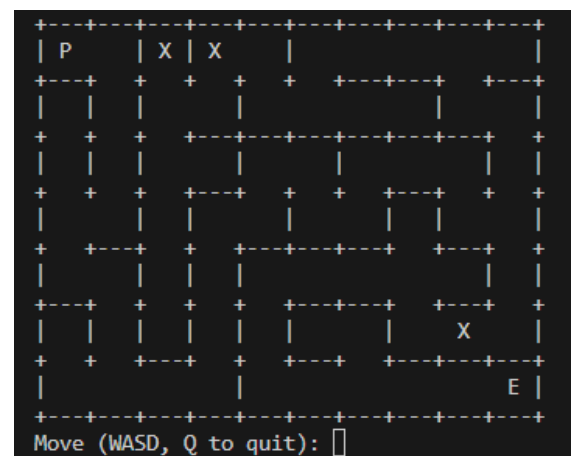


*Amazing Maze* (1976).



*Maze Craze* (1980).

Originally developed as a console-based game using ASCII characters, the project has now started transitioning into a graphical version using Java Swing. This GUI aims to enhance the player experience with better visuals and interactive elements. Local multiplayer is a key feature, allowing both players to compete using the same keyboard. Sprite integration and GUI components are currently being developed, and JUnit testing will be implemented to ensure the game runs reliably as more features are added.



Console Output of the project.

### Modifications

Several changes and improvements have been made since we started the project. The general structure of the game, including the folder organization, the maze generation algorithm, and the basic game logic, was set up at the beginning. These parts allowed us to test the game early and gave a strong base to build on.

Diana created a Java Swing window that opens when running the class. For now, it only shows a gray screen, but it's the first step toward moving the game from the console to a graphical interface. Later, this window will be connected to the main game to make the experience smoother and more interactive.[4]

Georgii added some basic sprites to the maze. At the moment, these are shown as the letter 'x' in the console, but they already help make the game more challenging. These placeholders will be replaced by proper graphics once the Swing version is ready.

Muslim worked on the local multiplayer feature. Right now, both players use the same keyboard but cannot move at the same time, since the game runs in the console. However, we plan to support simultaneous gameplay once the game is fully integrated with Java Swing.

### Collaboration

#### Student 1: Diana Ali Silva

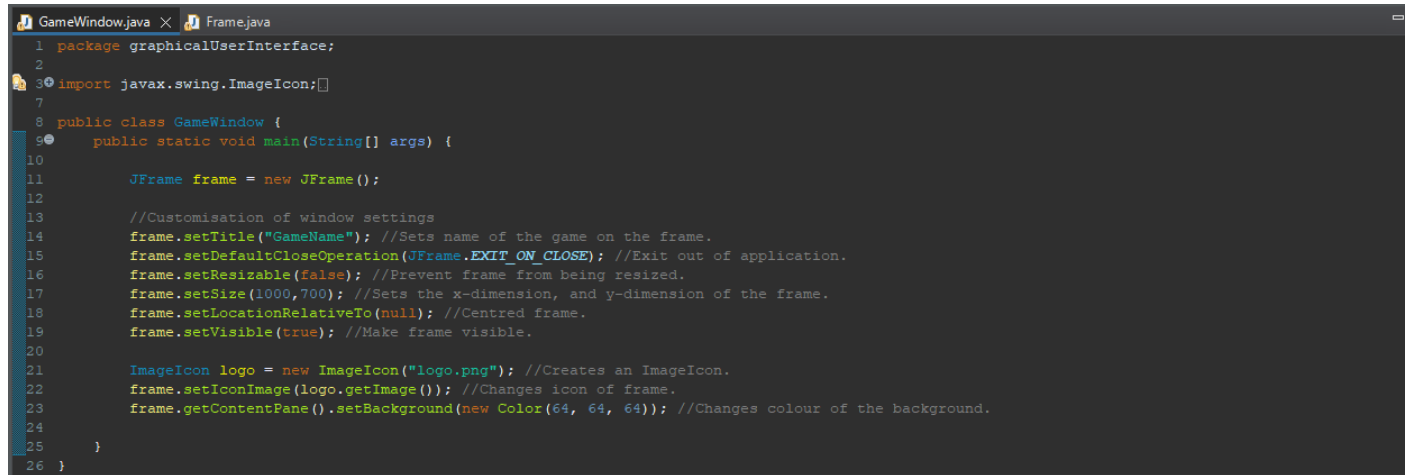
During the development of the maze game, I spent time researching the best way to implement the Graphical User Interface (GUI). After discussing options with my lecturer, Tracey Cassels, we agreed that Java Swing would be the right choice for this stage. It is simpler to work with than JavaFX and better suited for where we are in the learning process. I have since built a basic game window with Swing, setting up the title, icon, size and background colour to create a clean starting point.

However, in attempting to integrate the maze game logic into the GUI, I faced a number of issues with Git. Specifically, some of my commits were present locally but not reflected in the remote repository. To synchronise the local and remote branches, the commits had to be merged together. After addressing these merge conflicts, I encountered another issue in which Git stated that there were no changes to commit despite changes being made to the code. This has momentarily slowed development because I cannot submit changes until the commit history is fully synced.

Moving forward, my immediate plan is to address these issues to ensure a smooth workflow. This involves resolving any discrepancies between local and remote branches, and making sure future commits are properly synchronised. Once these issues are resolved, the next step will be to fully integrate the maze game into the GUI. This includes connecting the game logic to the visual components and ensuring smooth interaction between the two. Additionally, I will review the code for any errors or inefficiencies, particularly in how the game renders with the GUI integration.

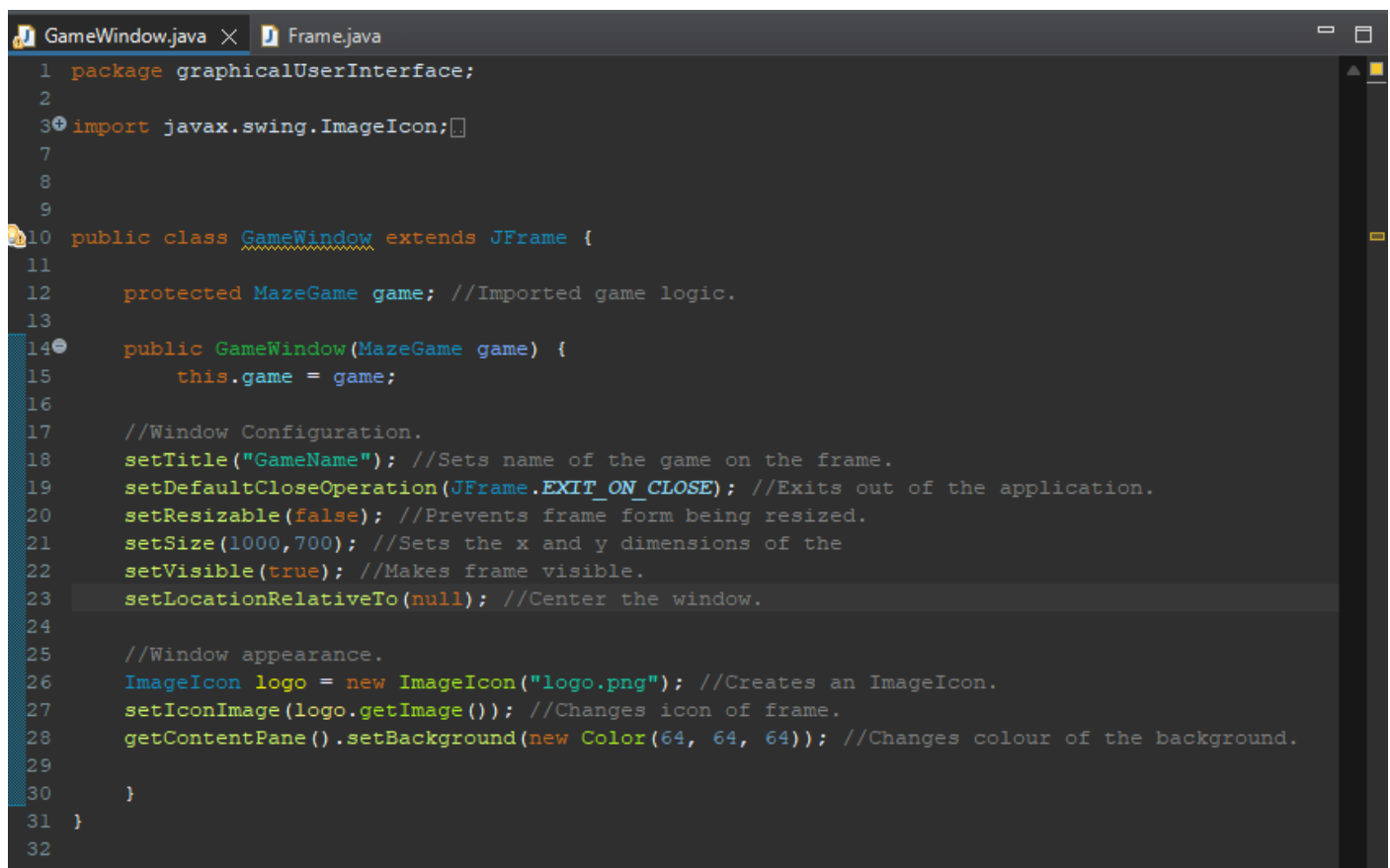
The goal is to have a functional, visually cohesive maze game with a properly working GUI in the next phase of development.

## Diana's Screenshots



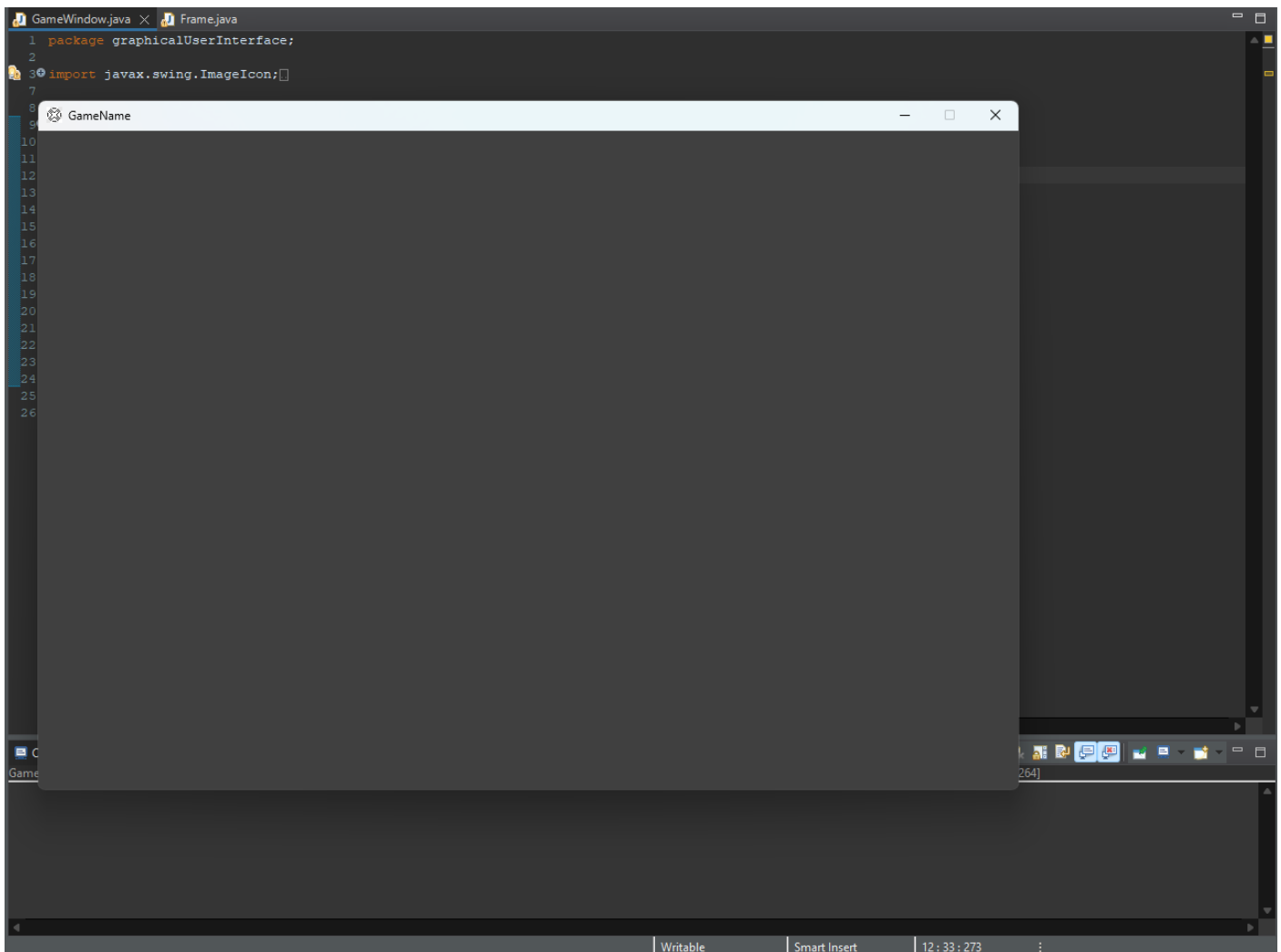
```
1 package graphicalUserInterface;
2
3 import javax.swing.ImageIcon;
4
5 public class GameWindow {
6     public static void main(String[] args) {
7
8         JFrame frame = new JFrame();
9
10        //Customisation of window settings
11        frame.setTitle("GameName"); //Sets name of the game on the frame.
12        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Exit out of application.
13        frame.setResizable(false); //Prevent frame from being resized.
14        frame.setSize(1000,700); //Sets the x-dimension, and y-dimension of the frame.
15        frame.setLocationRelativeTo(null); //Centred frame.
16        frame.setVisible(true); //Make frame visible.
17
18        ImageIcon logo = new ImageIcon("logo.png"); //Creates an ImageIcon.
19        frame.setIconImage(logo.getImage()); //Changes icon of frame.
20        frame.getContentPane().setBackground(new Color(64, 64, 64)); //Changes colour of the background.
21
22    }
23 }
```

Base code for the GUI.



```
1 package graphicalUserInterface;
2
3 import javax.swing.ImageIcon;
4
5 public class GameWindow extends JFrame {
6     protected MazeGame game; //Imported game logic.
7
8     public GameWindow(MazeGame game) {
9         this.game = game;
10
11        //Window Configuration.
12        setTitle("GameName"); //Sets name of the game on the frame.
13        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Exits out of the application.
14        setResizable(false); //Prevents frame from being resized.
15        setSize(1000,700); //Sets the x and y dimensions of the
16        setVisible(true); //Makes frame visible.
17        setLocationRelativeTo(null); //Center the window.
18
19        //Window appearance.
20        ImageIcon logo = new ImageIcon("logo.png"); //Creates an ImageIcon.
21        setIconImage(logo.getImage()); //Changes icon of frame.
22        getContentPane().setBackground(new Color(64, 64, 64)); //Changes colour of the background.
23
24    }
25 }
```

Code in attempt to integrate game logic.



Simple GUI window with gray background, logo and generic game name.

```
diana@DESKTOP-410C6F7 MINGW64 ~/projects/Software_Development_Project/Game_Proje
ct/src/graphicalUserInterface (main)
$ git commit -m "Reverted codes for both GameWindow.java and Frame.java without
game integration."
On branch main
Your branch is up to date with 'origin/main'.
nothing to commit, working tree clean
```

Unable to commit due to git not recognising any changes made.



Merge pull request to synch local and remote commits.



## Student 2: Carlos David Urrea Cabello

At the start of the project, I created a GitHub repository to help the team collaborate more easily and work remotely. This also made it easier to track changes and manage versions.

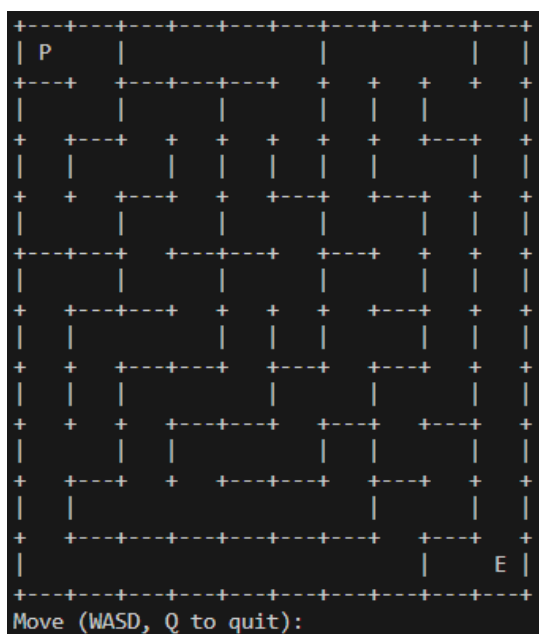
I developed the **random maze generator algorithm** using recursive backtracking, which creates a new and solvable maze each time the game runs. I also implemented basic game logic, including single-player movement and a message that appears when the player reaches the exit.[3]

Lastly, I organized the folder structure to follow the layout from the example shared on Moodle. This helped make the project easier to manage and prepare for future updates like the GUI and multiplayer features.

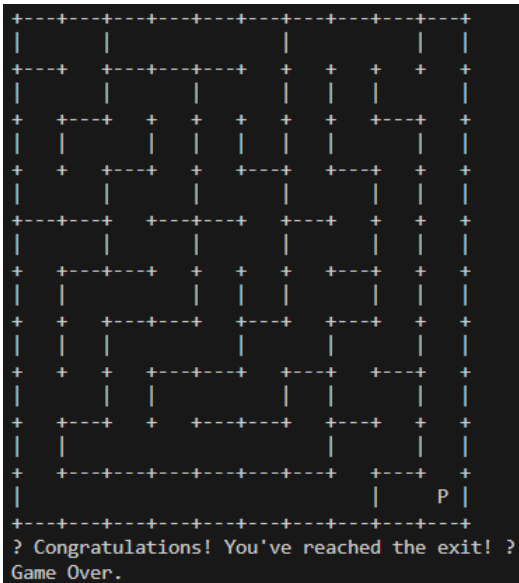
## Carlos's Screenshots

[Software\\_Development\\_Project](#) PublicJava Updated 3 hours ago

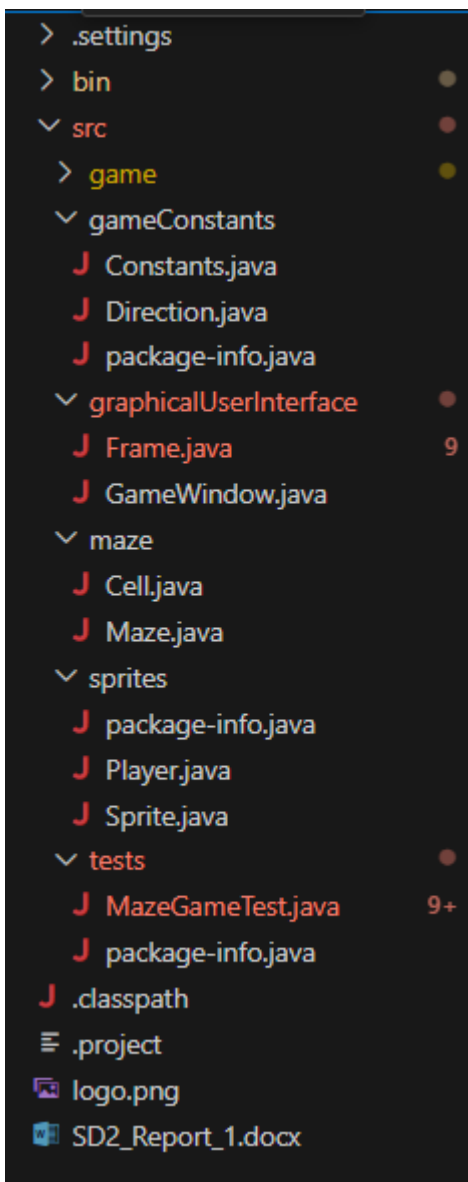
## Github repository



randomly generated maze



*Exit message shown when the player successfully reaches the end of the maze.*



Organized folder structure based on the example provided on Moodle.

## Student 3: Georgii Taisaev

1) Add spirite logic moves position

## sprites file update



TgTstion • 33 minutes ago

```
package sprites;
import java.util.Random;
import maze.Maze;

public class Sprite {
    private int row;
    private int col;
    private final Random random = new Random();

    public Sprite(int row, int col) {
        this.row = row;
        this.col = col;
    }

    public int getRow() {
        return row;
    }

    public int getCol() {
        return col;
    }

    public void moveRandomly(Maze maze) {
        String[] directions = {"W", "S", "A", "D"};
        for (int i = 0; i < 4; i++) {
            String direction = directions[random.nextInt(directions.length)];
            if (maze.canMoveSprite(this, direction)) {
                switch (direction) {
                    case "W" -> row--;
                    case "S" -> row++;
                    case "A" -> col--;
                    case "D" -> col++;
                }
                break;
            }
        }
    }
}
```

2) Add spirites printing with maze

## sprites printing in maze and can move



TgTstion • 32 minutes ago

```
public void printMaze(Player player, List<Sprite> spirits) {
    System.out.println("+" + "---+".repeat(width));

    for (int row = 0; row < height; row++) {
        StringBuilder top = new StringBuilder(str:"|");
        StringBuilder bottom = new StringBuilder(str:"+");

        for (int col = 0; col < width; col++) {
            Cell cell = grid[row][col];
            boolean isPlayerHere = (player.getRow() == row && player.getCol() == col);
            boolean isExit = (row == exitRow && col == exitCol);
            boolean isSpiritHere = false;

            for (Sprite spirit : spirits) {
                if (spirit.getRow() == row && spirit.getCol() == col) {
                    isSpiritHere = true;
                    break;
                }
            }

            String body;
            if (isPlayerHere) {
                body = " P ";
            } else if (isSpiritHere) {
                body = " X ";
            } else if (isExit) {
                body = " E ";
            } else {
                body = "   ";
            }
            top.append(body);
            top.append(cell.east ? "|" : " ");

            bottom.append(cell.south ? "---" : " ");
            bottom.append(str:"+");
        }
    }
}
```

```

public boolean canMoveSprite(Sprite sprite, String direction) {
    int row = sprite.getRow();
    int col = sprite.getCol();
    Cell cell = grid[row][col];

    return switch (direction.toUpperCase()) {
        case "W" -> row > 0 && !cell.north;
        case "S" -> row < height - 1 && !cell.south;
        case "A" -> col > 0 && !cell.west;
        case "D" -> col < width - 1 && !cell.east;
        default -> false;
    };
}

```

3) When sprites caught player game ends, sprites not avoiding walls

end game when spirit caught, save spir...



TgTstion • 30 minutes ago

```

List<Sprite> spirits = new ArrayList<>();
for (int i = 0; i < 3; i++) {
    int randomRow = random.nextInt(maze.getHeight());
    int randomCol = random.nextInt(maze.getWidth());
    spirits.add(new Sprite(randomRow, randomCol));
}
while (true) {
    maze.printMaze(player, spirits);

    if (maze.isAtExit(player)) {
        System.out.println(x: "🎉 Congratulations! You've reached the exit! 🎉");
        break;
    }
    for (Sprite spirit : spirits) {
        if (spirit.getRow() == player.getRow() && spirit.getCol() == player.getCol()) {
            System.out.println(x: "💀 A spirit caught you! Game Over! 💀");
            return;
        }
    }

    System.out.print(s: "Move (WASD, Q to quit): ");
    input = scanner.nextLine().toUpperCase();
}

```

```

}
for (Sprite spirit : spirits) {
    spirit.moveRandomly(maze);
}
}

```

### Student 4: Muslim Muradov

Adding local multiplayer functionality for the game was the main goal of my part of the project. By adding a second player object and changing the input handling logic appropriately, I was able to adapt the single player movement to include two players.

I modified the code so that player 1 spawns in the top-left corner of a 10x10 maze (coordinates (0,0)), and player 2 spawns in the bottom-right corner (coordinates (9,9)). This arrangement is intended as a precursor to a mirrored maze design, ensuring a fair playing field for both players. Simultaneous input is not possible yet because the game is console based. Consequently, the game currently handles moves in a sequential manner: Player 1 uses the WASD keys to move (and Q to quit), while Player 2 uses the IJKL keys (and T to quit). In order to allow both players to input simultaneously, we are going to use java swing.

In order to expand the gameplay to support local multiplayer, I made sure that the core game logic remained consistent by modifying the existing functions, which were initially written for a single player, to handle two players.

## Muslim's Screenshots

```
public class MazeGame { no usages Muslim Muradov +1

    public static void main(String[] args) { no usages Muslim Muradov +1
        // Maze creation placeholder
        Maze maze = new Maze(10, 10);
        maze.generateMaze();

        // Creating 2 players:
        // Player 1 starts at (0,0), player 2 – (9,9) (10x10 maze)
        Player player1 = new Player(0, 0);
        Player player2 = new Player(9, 9);

        Scanner scanner = new Scanner(System.in);
        String input;
```

Added spawn logic for 2 players, so they spawn in two opposite sides

```
while (true) {
    // Checking if player 1 reached the exit
    if (maze.isAtExit(player1)) {
        System.out.println("Congratulations Player 1! You've reached the exit! \uD83C\uDFC6");
        break;
    }

    // Player 1 turn
    System.out.print("(Move WASD, Q to quit): ");
    input = scanner.nextLine().toUpperCase();
    if (input.equals("Q")) {
        break; // Player 1 wants to quit
    }

    // Movemonet logic (if canMove == true, than move)
    if (maze.canMove(player1, input)) {
        switch (input) {
            case "W": player1.move(-1, 0); break;
            case "S": player1.move(1, 0); break;
            case "A": player1.move(0, -1); break;
            case "D": player1.move(0, 1); break;
            default:
                // Invalid input
                System.out.println("Invalid command for Player 1!");
        }
    } else {
        System.out.println("You hit a wall!");
    }
}
```

Updated logic for player 1



```
// Checking if player 2 reached the exit
if (maze.isAtExit(player2)) {
    System.out.println("Congratulations Player 2! You've reached the exit! \uD83C\uDFC6");
    break;
}

// Player 2 turn
System.out.print("(Move IJKL, T to quit): ");
input = scanner.nextLine().toUpperCase();
if (input.equals("T")) {
    break; // Player 2 wants to quit
}

// Movement logic (if canMove == true, then move)
if (maze.canMove(player2, input)) {
    switch (input) {
        case "I": player2.move(-1, 0); break;
        case "K": player2.move(1, 0); break;
        case "J": player2.move(0, -1); break;
        case "L": player2.move(0, 1); break;
        default:
            System.out.println("Invalid command for Player 2!");
    }
} else {
    System.out.println("You hit a wall!");
}
}

scanner.close();
System.out.println("Game Over.");
```

Adapted code for player 2

### Planned Improvements for Next Version

For the next stage of the project, our goal is to move from a console-based experience to a more complete and interactive graphical game. One of the main updates will be to fully integrate the Java Swing interface with the game logic, allowing the maze and player movement to be displayed in a window with basic visual elements.[4]

We plan to replace the current console output with a 2D view of the maze, where walls, paths, and players are drawn using simple graphics or placeholder sprites. This will improve the overall user experience and make the game easier to understand and play.

Another important update is to support simultaneous local multiplayer. Right now, players take turns moving through the maze, but in the next version, both players should be able to move at the same time using different keys on the same keyboard. This will make the gameplay more dynamic and competitive.

Other improvements may include adding a simple menu or game start screen, using Swing buttons or key bindings, and possibly updating the placeholder sprites to better represent obstacles or player characters.

These updates will bring the project closer to a fully playable and visually interactive game.

**Link to our repository:**

[https://github.com/Carlos-o620/Software\\_Development\\_Project.git](https://github.com/Carlos-o620/Software_Development_Project.git)

## References

1. Midway Manufacturing Co. (1976). Amazing Maze [Arcade Game]. International Arcade Museum. Retrieved from <https://www.arcade-museum.com/Videogame/amazing-maze>
2. Atari. (1980). Maze Craze [Atari 2600 Game]. Atari Official Website. Retrieved from <https://atari.com/pages/mazecraze>
3. Buck, J. (2015). Mazes for Programmers: Code Your Own Twisty Little Passages. The Pragmatic Bookshelf. ISBN: 978-1-68050-055-4
4. GeeksforGeeks. (n.d.). Introduction to Java Swing. Retrieved from <https://www.geeksforgeeks.org/introduction-to-java-swing/>