Project title	Twisted Trails
Author(s)	Gozman-Pop Maria-Eliza
Group	30424

1. Task Description

This mini-project was primarily developed in IntelliJ IDEA for Java, known for its efficient and developer-friendly environment. However, its adaptable structure allows it to be seamlessly opened and managed in other well-known IDEs, including BlueJ, Eclipse, NetBeans, and Visual Studio Code.

"Twisted Trails" is more than just a game; it's a digital homage to those fond memories of laughter and friendly competition that many of us share with our friends. Picture this: you're at your favorite hangout spot, the Garlic restaurant in Cluj-Napoca, chatting away and enjoying the company while you all dive into the maze puzzles on the placemats, racing to see who can escape the labyrinth first. That's where the heart of "Twisted Trails" lies.

In this delightful maze game, we capture that essence of camaraderie and playful rivalry. It's a cozy virtual corner where you can relive those joyous challenges with your buddies, no matter where you are. You'll navigate through beautifully crafted mazes, each level upping the ante with cleverly placed collectibles that you must gather to progress, adding a sprinkle of strategy to your maze-solving skills. The game wraps you in a fun, engaging soundtrack that sets the perfect mood for adventure. Visually, it's a treat, with elements that catch the eye and make every turn, every solved puzzle, a little celebration of its own.

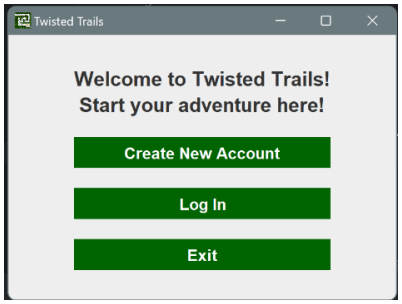
And what's a game without friends to share the fun with? "Twisted Trails" lets you bring in your pals, old and new, with the "Add Friends" feature. Here, every friend you add becomes a part of your journey, as you all can peek at each other's progress and cheer one another on. It's not just about reaching the end; it's about sharing the laughter, the "aha!" moments, and maybe a little bit of bragging rights when you outmaneuver everyone else to the finish line.

And the best part? The fun never ends! Once you've conquered all the levels, a simple restart reshuffles the adventure with randomly generated mazes and obstacles, guaranteeing a fresh challenge every time. So, whether you're the reigning maze champ among your friends or the underdog plotting your comeback, "Twisted Trails" is your new playground for endless fun and new memories. Ready for the challenge?

User Guide for "Twisted Trails"

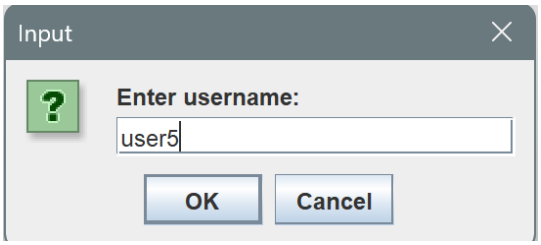
1. Starting the Game

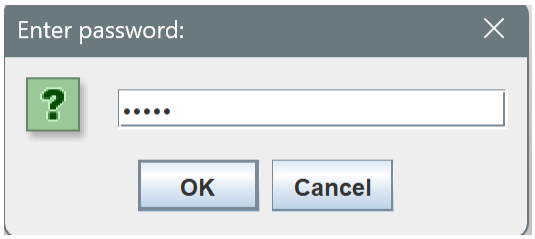
- **Opening Screen:** When you launch "Twisted Trails", you'll be greeted with the main menu.
- **Options Available:** Here you can choose to create a new account, log in with an existing account, or exit the game.

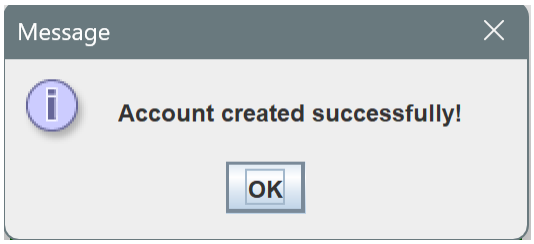


2. Account Creation

- **Creating a New Account**: Click on the "Create New Account" button.
- **Entering Details:** Enter your desired username and password in the prompted dialogs.
- **Account Confirmation:** A message will confirm the successful creation of your account.



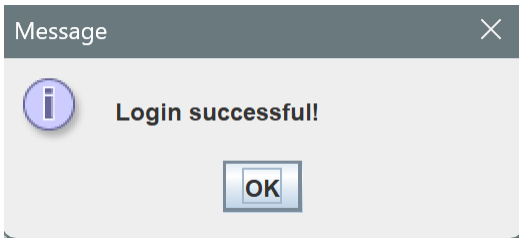




3. Logging In

- **Accessing Your Account**: From the main menu, select "Log In".
- **Login Details**: Enter your username and password.
- **Successful Login:** A success message will appear, and you'll be taken to the game's main interface.

(same as for account creation)

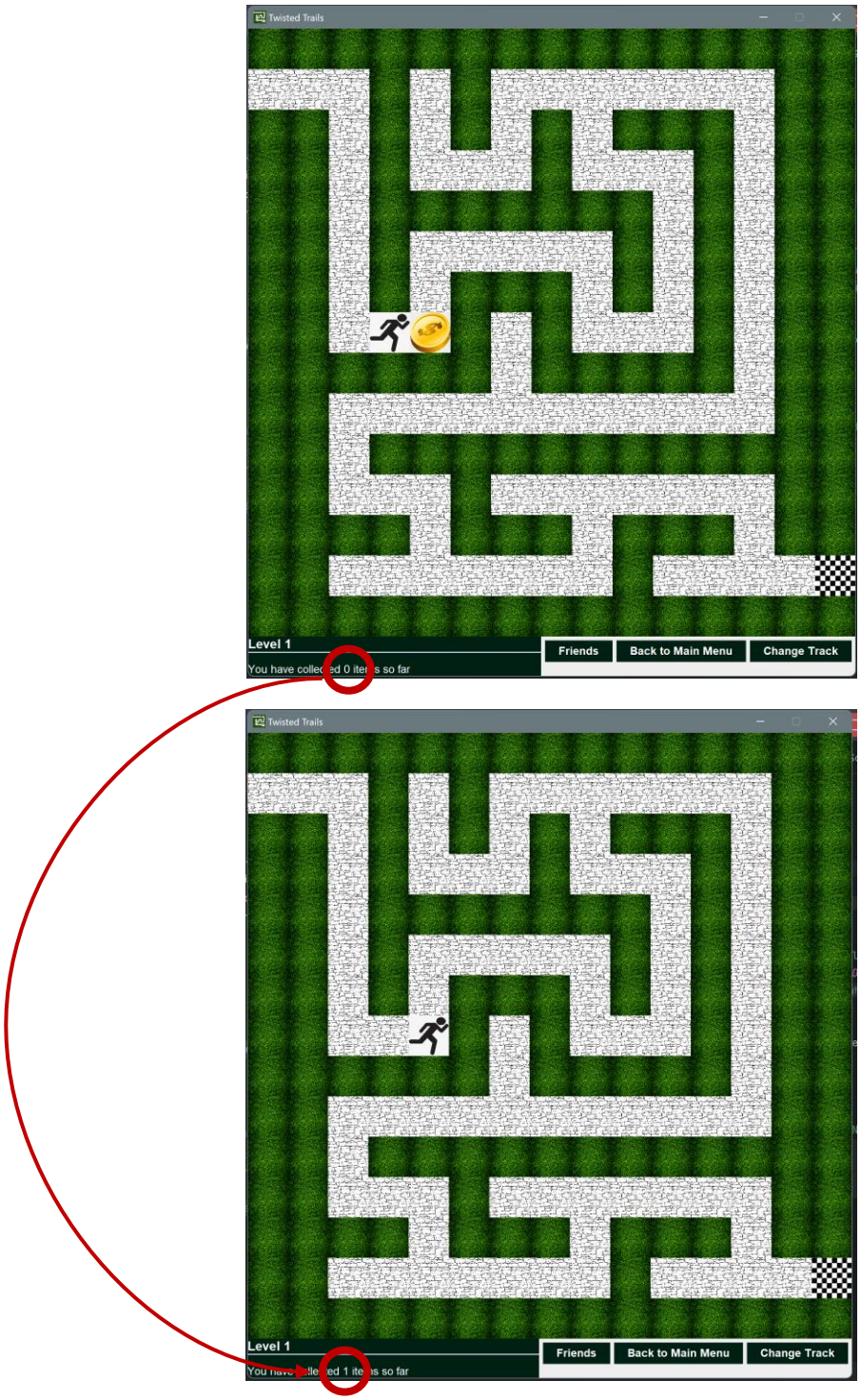


4. Navigating the Main Interface

- **Game Interface:** Once logged in, you'll see the game UI with the current maze level.
- **Controls and Options:** Familiarize yourself with the game controls and any available options like sound settings.

5. Playing the Game

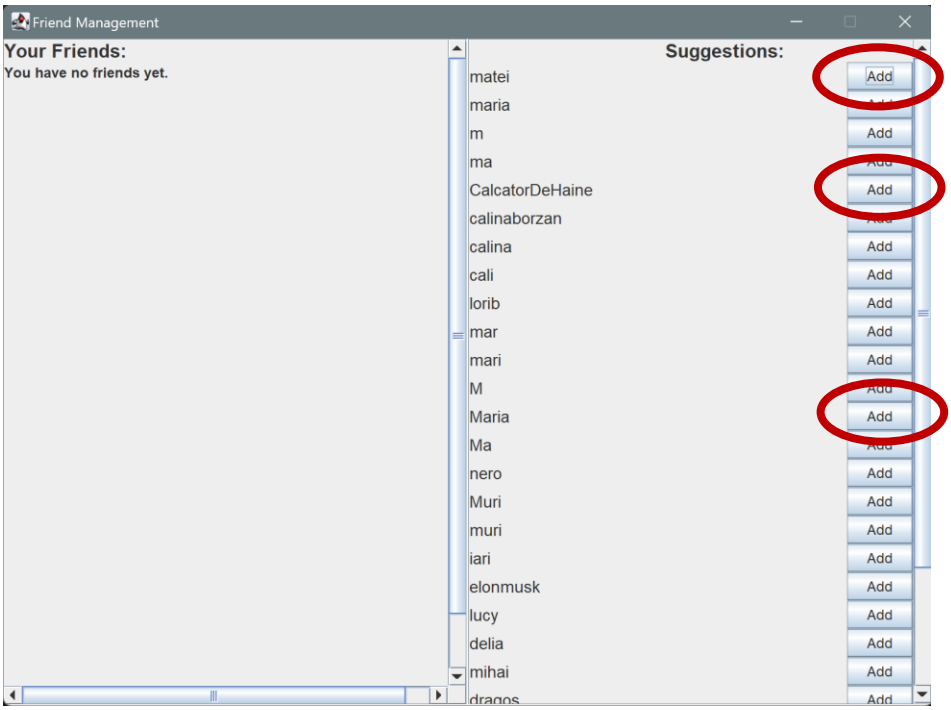
- **Maze Navigation:** Use the keyboard or on-screen controls to navigate through the maze.
- **Collecting Items:** Ensure to collect all required items in the maze to progress.
- **Level Completion:** Reach the end of the maze to complete the level, and you will automatically be taken to the next one.

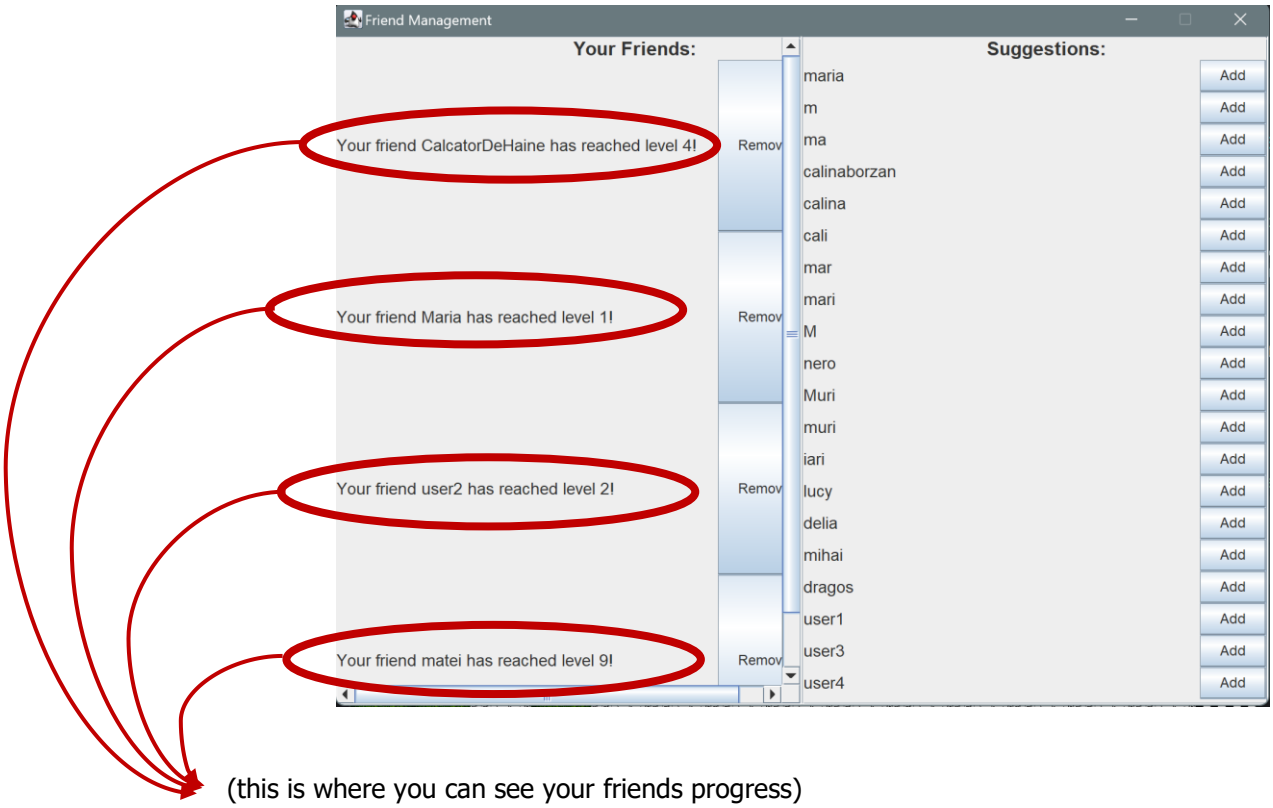



Twisted Trails
Message
Congratulations! You reached the end of the maze!
OK
Level 1
Friends
Back to Main Menu
Change Track
Twisted Trails
Level 2
Friends
Back to Main Menu
Change Track

6. Adding and Viewing Friends

- **Adding Friends:** Access the friends feature from the main menu or in-game menu.
- **Search and Add:** Search for your friends by username and add them to your friend list.
- **Viewing Progress:** View your friends' progress in their maze journey.

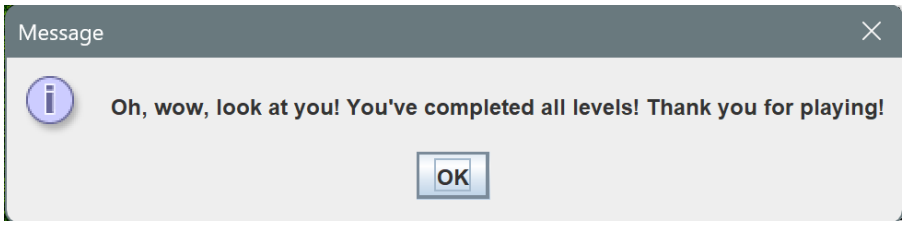


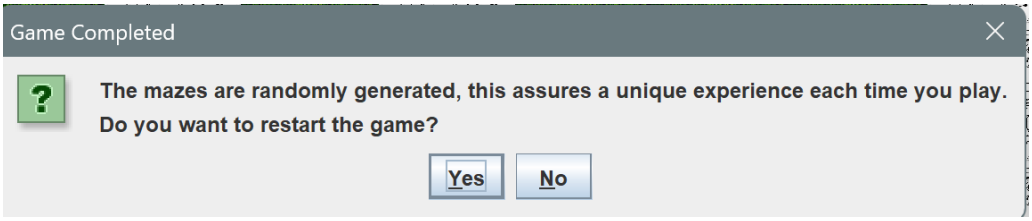


(this is where you can see your friends progress)

7. Restarting and Randomization

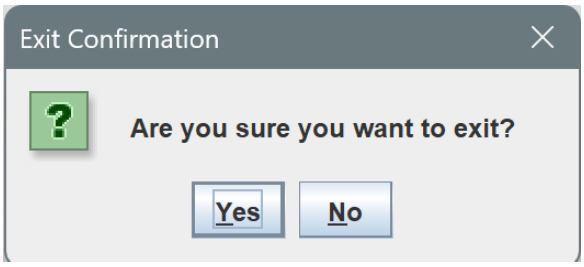
- **Restarting the Game:** Once you complete all levels, you can choose to restart the game.
- **Experiencing New Mazes:** Each restart generates new mazes and challenges.





8. Exiting the Game

- **Exiting:** You can exit the game from the main menu or the in-game menu.
- **Saving Progress:** Your progress is automatically saved when you exit.



2. Class Discovery

MainMenu	
Initialize the game window	• **JFrame** • **JPanel**
Handle user interactions for account creation	• **JOptionPane**
Handle user interactions for login	• **JOptionPane**
Navigate to the game interface when starting the game	• **UIManager**
Update user progress after level completion	• **UserAccount** • **CredentialsManager**

UIManager	
Manage the user interface for the game	• **JFrame** • **CardLayout** • **JPanel**
Handle level loading	• **Level**
Handle player movements	• **KeyListener**
Draw the maze and player	• **Graphics**
Play sounds on certain actions	• **BackgroundSound** • **WinningSounds**
Update the game state	• **Level** • **UserAccount**

Level	
Generate maze structure	• **MazeGenerator**
Place collectible items in the maze	
Check for valid player movements	
Determine when the level is completed	• **UserAccount**

CredentialsManager	
Save new user credentials to file	• **FileWriter**
Load user credentials from file	• **FileReader**
Update user level in credentials	• **FileWriter**
Check if username exists in the credentials	
Delete a user account from credentials	• **FileWriter**

UserAccount	
Store user credentials	
Manage user level progression	
Load user's friend list from file	• **FileReader**
Save user's friend list to file	• **FileWriter**
Add a friend to the friend list	
Remove a friend from the friend list	

FriendManagementUI	
Initialize Friend Management Window	• **JFrame** • **JPanel**
Manage User's Friends List	• **JLabel** • **UserAccount**
Display Friends Suggestions	• **JFrame** • **JPanel**

Sound	
Play a sound from a file	• **AudioSystem** • **Clip**
Stop playing a sound	
Handle playback completion	• **LineListener**

BackgroundSound	
Play background music continuously	• **Clip**
Change the background music track	
-	• **Sound** (inherits responsibilities)

WinningSounds	
Play sound when an item is collected	
Play sound when a level is finished	
	• **Sound** (inherits responsibilities)

- **MainMenu Class:** This is the entry point for the game's user interface. It creates a window where players can create a new account, log in to an existing account, or exit the game. The class is responsible for setting up the main graphical interface, including buttons for different actions and customizing their appearance. It also manages the transition from the main menu to the game's main user interface upon successful login or account creation.

- **UIManager Class:** This class is crucial for the game's visual and interactive elements. It renders the game's levels, handles player movements, and updates the user interface based on game events. It is responsible for drawing the maze, player, collectibles, and other game elements on the screen. Additionally, it manages transitions between different levels, checks for game completion, and includes UI components like a side menu for extra functionalities.

- **Level Class:** Each instance of this class represents an individual level in the game. It manages the maze's layout, including the placement of walls, passages, and collectibles. The class handles the logic for item collection and checks if the player has reached the finish line. A key functionality is the generation of the maze structure, which is done using the MazeGenerator class. This ensures each level of the game is a distinct challenge for the player.

- **CredentialsManager Class:** This class manages user data, particularly for login credentials. It handles saving new user credentials, loading existing credentials, updating user progress, and deleting user accounts. This is essential for maintaining user progress and personalization of the game experience.

- **UserAccount Class:** Represents a user account in the game. It stores user-specific information like username, level progress, and friends list. This class is important for personalizing the game experience, tracking progress, and integrating social features like a friends list.

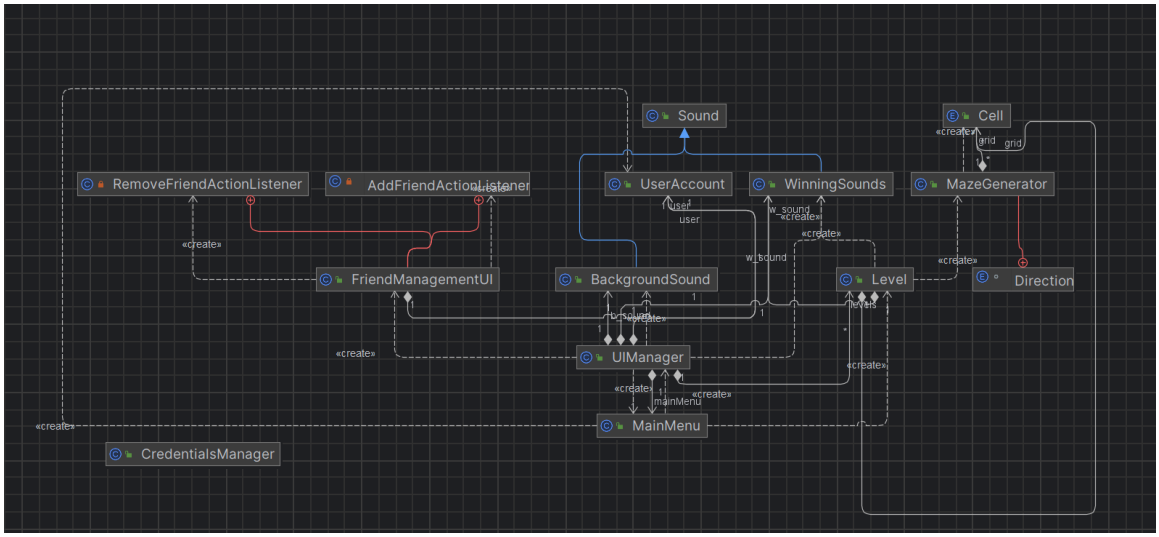
- **FriendManagementUI Class:** This class is integral to the social aspect of the game, providing a specialized interface for managing in-game friendships. It facilitates viewing and managing the player's network of friends within the game. The class constructs a user-friendly window that includes separate panels for current friends and friend suggestions.

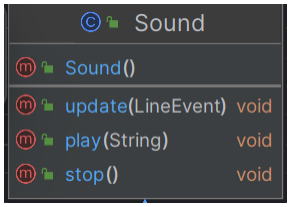
- **Sound Class:** This is a base class designed for general audio functionalities within the game. It can play, stop, and manage audio files. The class also implements the LineListener interface, enabling it to monitor and respond to different stages of audio playback.

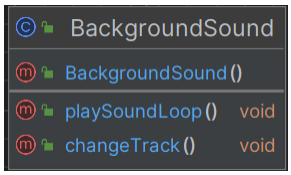
- **BackgroundSound Class:** As the name suggests, this class is dedicated to managing the background music that plays during gameplay. It can loop background music and allows the player to switch between different tracks, enhancing the game's ambiance.

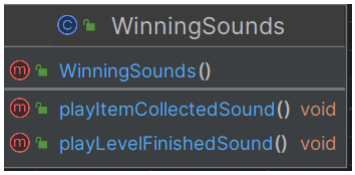
- **WinningSounds Class**: This class handles the sound effects in the game, specifically for moments of achievement like collecting an item or finishing a level. It extends a base Sound class and provides methods to play specific audio files, adding an auditory dimension to the player's accomplishments.

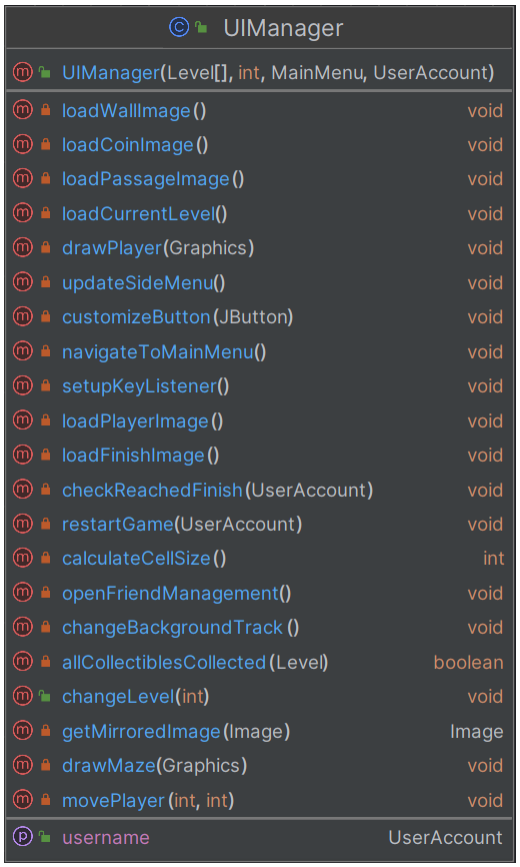
3. Class Diagram

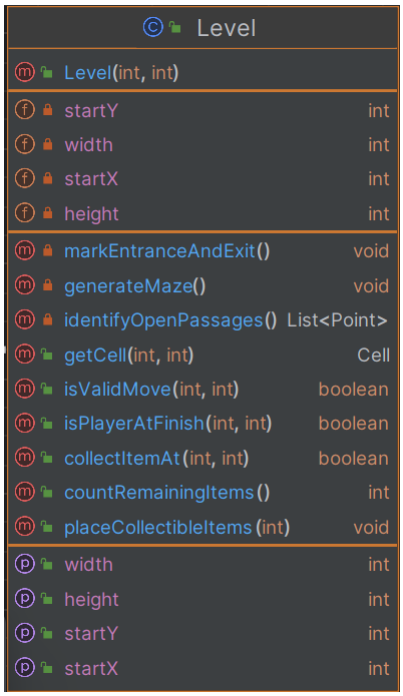


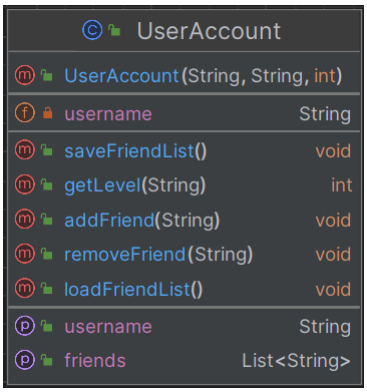


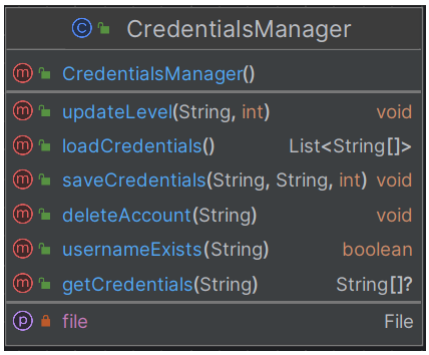


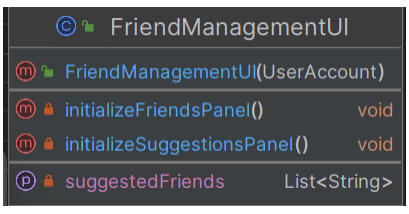


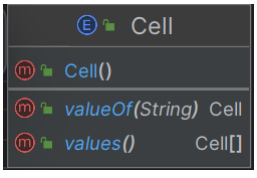

UIManager
UIManager(Level[], int, MainMenu, UserAccount)
loadWallImage() void
loadCoinImage() void
loadPassageImage() void
loadCurrentLevel() void
drawPlayer(Graphics) void
updateSideMenu() void
customizeButton(JButton) void
navigateToMainMenu() void
setupKeyListener() void
loadPlayerImage() void
loadFinishImage() void
checkReachedFinish(UserAccount) void
restartGame(UserAccount) void
calculateCellSize() int
openFriendManagement() void
changeBackgroundTrack() void
allCollectiblesCollected(Level) boolean
changeLevel(int) void
getMirroredImage(Image) Image
drawMaze(Graphics) void
movePlayer(int, int) void
username UserAccount

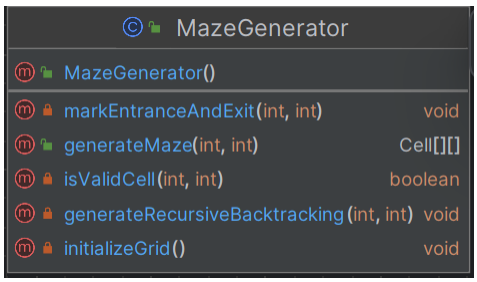

Level
Level(int, int)
startY int
width int
startX int
height int
markEntranceAndExit() void
generateMaze() void
identifyOpenPassages() List<Point>
getCell(int, int) Cell
isValidMove(int, int) boolean
isPlayerAtFinish(int, int) boolean
collectItemAt(int, int) boolean
countRemainingItems() int
placeCollectibleItems(int) void
width int
height int
startY int
startX int

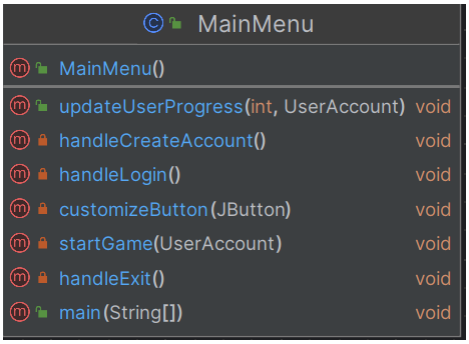

UserAccount
UserAccount(String, String, int)
username String
saveFriendList() void
getLevel(String) int
addFriend(String) void
removeFriend(String) void
loadFriendList() void
username String
friends List<String>


CredentialsManager
CredentialsManager()
updateLevel(String, int) void
loadCredentials() List<String[]>
saveCredentials(String, String, int) void
deleteAccount(String) void
usernameExists(String) boolean
getCredentials(String) String[]?
file File


FriendManagementUI
FriendManagementUI(UserAccount)
initializeFriendsPanel() void
initializeSuggestionsPanel() void
suggestedFriends List<String>


Cell
Cell()
valueOf(String) Cell
values() Cell[]





4.Conclusion

The Java code provided demonstrates several key concepts of Object-Oriented Programming (OOP):

Classes and Objects: Classes like MainMenu, UIManager, Level, UserAccount, CredentialsManager, Sound, BackgroundSound, WinningSounds are blueprints for creating objects. Each class defines state and behavior that the objects of the class can have.

Inheritance: There is a clear inheritance structure where BackgroundSound and WinningSounds classes extend the Sound class. This allows these subclasses to inherit methods and properties from the Sound class.

Encapsulation: Encapsulation is seen in the use of private fields (like private int userProgress in MainMenu and private Cell[][] grid in Level) and public methods (like public void play(String audioFilePath) in Sound). This ensures that the internal representation of the object is hidden from the outside and is accessed only through the methods.

Polymorphism: Polymorphism is used, though not extensively. It's more implicit, seen in how objects of BackgroundSound and WinningSounds are treated as Sound objects. They override the play method from the Sound class.

Abstraction: Abstraction is achieved by using complex classes like MainMenu, UIManager, and Level, which hide complex logic behind simple interfaces.

Association: There are several associations:

- MainMenu has a UserAccount.
- UIManager has a BackgroundSound, WinningSounds, MainMenu, UserAccount, and an array of Level.

Aggregation: Aggregation is a special form of association where the child can exist independently of the parent. For example, UIManager contains Level objects, but Level objects can exist independently of UIManager.

Composition: Composition is a strong form of aggregation. MainMenu, for instance, is composed of JPanel, JButton, etc. If MainMenu is destroyed, its components are also destroyed.

In conclusion, this Java project, leveraging the Swing framework, represents a comprehensive and interactive gaming application that combines graphical user interface elements with sound management and user account handling. The modular design, as reflected in the distinct responsibilities of each class, allows for easy maintenance and scalability of the application. Overall, these classes work together to create a comprehensive gaming experience. The MainMenu class serves as the entry point, UIManager and Level classes create the game environment, MazeGenerator adds complexity to the gameplay, CredentialsManager and UserAccount handle personalization and progress tracking, FriendManagementUI offers a comprehensive and socially interactive gaming environment, and the sound-related classes (WinningSounds, BackgroundSound, Sound) enhance the game's auditory appeal. Overall, this project exemplifies a well-rounded application that combines functional complexity with user-friendly design. Its modular structure and object-oriented approach facilitate easy maintenance and future enhancements.