

## SRS Format with IEEE Standard

### 1. Introduction

- a. Background
- b. Overall Description
- c. Environmental Characteristics
  - i. Hardware
  - ii. Peripherals
  - iii. People
- d. Interfaces
  - Interface with User
- e. Constraints

### 2. Functional Requirements

### 3. Non-Functional Requirements

### 4. Design Diagrams

### 5. Miscellaneous

## **1. Introduction:**

### **a. Background:**

The idea behind the maze solving project stems from the challenge of navigating through complex structures efficiently. Mazes, with their intricate pathways and dead ends, present a classic problem-solving scenario. The project aims to develop a software solution that can **automatically find the optimal path** through a maze, **employing the backtracking algorithm**. This project is not only a demonstration of algorithmic process but also serves as a practical application of problem-solving techniques in computer science.

### **b. Overall Description:**

The maze solving program is designed to provide users with a platform to interact with and solve mazes digitally. It offers a **single-player experience** where users can input or edit the structure of a maze statically in a **6x6 2D array**. The program then utilizes the backtracking algorithm to determine the correct path(s) from the starting point to the ending point of the maze. The solution path is displayed to the user as the output. The program aims to be user-friendly, allowing individuals to explore the concept of maze solving in a digital environment.

### **c. Environmental Characteristics:**

#### **i. Hardware**

The maze solving program is designed to run on standard computing hardware commonly found in personal computers, laptops, and similar devices. It does not have specific hardware requirements beyond those **necessary for running Java applications**. Therefore, any modern computer with sufficient processing power, memory, and storage capacity should be capable of running the program smoothly.

#### **ii. Peripherals**

The program primarily relies on standard input/output interfaces commonly available on computing devices:

Input: Users interact with the program primarily through the keyboard to input or edit the structure of the maze.

Output: The solution path(s) are displayed on the screen in a textual format.

#### **iii. People**

The intended user base for the maze solving program includes individuals interested in problem-solving, algorithmic exploration, and computer programming. Specifically, the program targets:

Students: Studying computer science or related fields, who can use the program to understand and experiment with maze-solving algorithms.

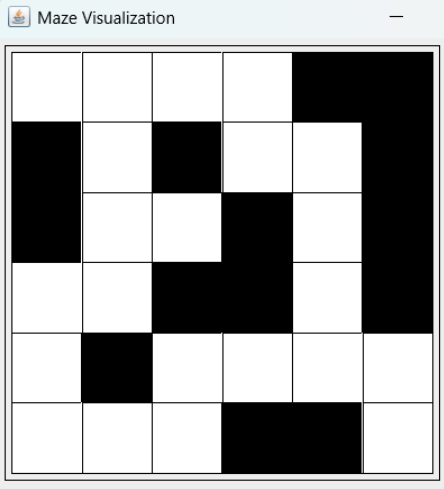
Enthusiasts: Individuals fascinated by puzzles and challenges, who enjoy solving mazes digitally.

Educators: Teachers and instructors who can incorporate the program into their curriculum to demonstrate algorithmic concepts and problem-solving techniques.

#### d. Interfaces:

Interface with User:

```
PS D:\OneDrive\Desktop\UPES\sem 4\SEPM\maze solving program> java Maze_Solver_Improved.java } ; if ($?) { java Maze_Solver_Improved.java
Enter 'start' to execute the program:
start
Choose the difficulty level:
1. Easy
2. Medium
3. Advanced
Enter your choice: 2
Options:
1. Display all paths
2. Display the best path
3. Display both
Enter your choice: 3
All paths:
RDDLDDRRURRRD
RRDRDDDDRD
Best Path:
RRDRDDDDRD
Do you want to display the maze graphically? (yes/no)
yes
```



The user interacts with the program through text prompts on the command line.

**Program Start:** The program starts by displaying a message or menu.

**Maze Difficulty:** The user is prompted to choose a difficulty level for the maze, by typing either 1, 2 or 3 corresponding to difficulty options.

**Path Visualization:** The program will ask if the user wants to see all explored paths, the best solution path, or both. The user will respond by typing either 1, 2 or 3.

**Graphical Display:** The program offers an option to display the maze graphically. The user responds with a yes/no prompt.

#### e. Constraints:

The maze solving program operates within certain limitations and boundaries, which are outlined as follows:

##### 1) Fixed Maze Size

The size of the maze is fixed at 6x6, as per the project requirements. This constraint **limits the scalability** of the program to handle mazes of larger dimensions.

##### 2) Static Maze Structure

The arrangement of 0s and 1s in the maze is static. Users can input or edit the maze structure before solving it, but the **structure remains unchanged** during the solving process.

##### 3) Single-Player Interaction

The program is designed for single-player interaction, limiting collaborative or multiplayer maze-solving scenarios.

#### 4) Textual Output

The solution path(s) are displayed in a textual format, which may **limit the visualization of the maze solution** for some users who prefer graphical representations highlighting the path(s) in the maze.

#### 5) Limited Input Methods

Input for editing or creating the maze structure is primarily through the keyboard, **limiting alternative input methods** such as mouse-based editing.

#### 6) Algorithmic Complexity

The backtracking algorithm employed by the program **utilizes a brute-force approach**, which may result in longer processing times for more complex mazes. This constraint may impact the responsiveness of the program, particularly for large or intricate mazes.

#### 7) Platform Dependency

The program is developed in Java, which may **introduce platform dependency issues** across different operating systems. While Java is designed to be platform-independent, minor variations in behaviour may occur on different systems.

## **2. Functional Requirements:**

### Maze Representation:

The program shall represent the maze using a 2D array where:

- **0 represents a wall** or closed cell.
- **1 represents an open cell.**

### Static Input:

- The size of the maze array will be fixed at 6x6.
- The arrangement of 1s and 0s in the array, defining the maze structure, will be static.

### Path Determination:

The program shall determine the correct path(s) from the **starting point (0,0)** to the **ending point (5,5)** within the maze.

### User Interaction:

- The program shall be single-player, requiring user input for static maze configurations.
- Users shall be able to **select the difficulty level of the maze** (easy, medium, hard).
- Users shall have the option to **choose which path(s) they want to be displayed**, considering that mazes could have multiple solutions.
- Users shall have the option to **display a graphical representation** of the selected maze.

### **3. Non-Functional Requirements:**

#### Performance:

- The program should efficiently determine paths in mazes of fixed size (6x6) and static configurations.
- Maze solving should not exceed reasonable time limits based on the selected difficulty level.

#### Usability:

- The user interface should be intuitive, guiding users through maze selection and path display options.
- Clear feedback should be provided on maze solving progress and solution path(s).

#### Reliability:

- The program should handle user inputs gracefully, ensuring no crashes or unexpected errors occur.
- Solution paths provided should always be correct and valid.

#### Portability:

The program should be deployable across different platforms (Windows, Linux, macOS) to ensure accessibility for users.

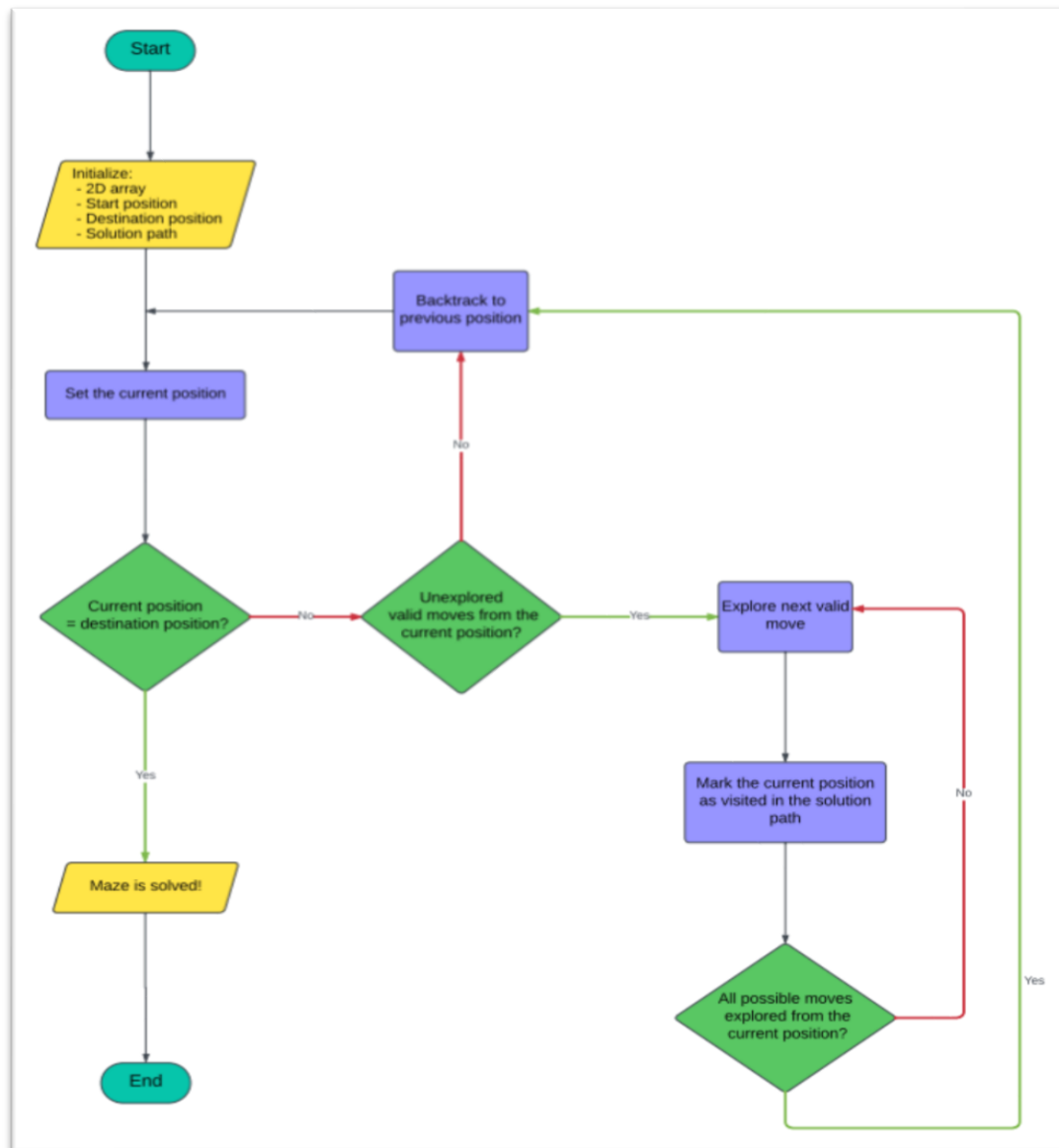
#### Maintainability:

- Codebase should be well-organized and documented to facilitate future updates or modifications.
- The system should be designed with modular components to ease maintenance tasks.

#### 4. Design Diagrams:

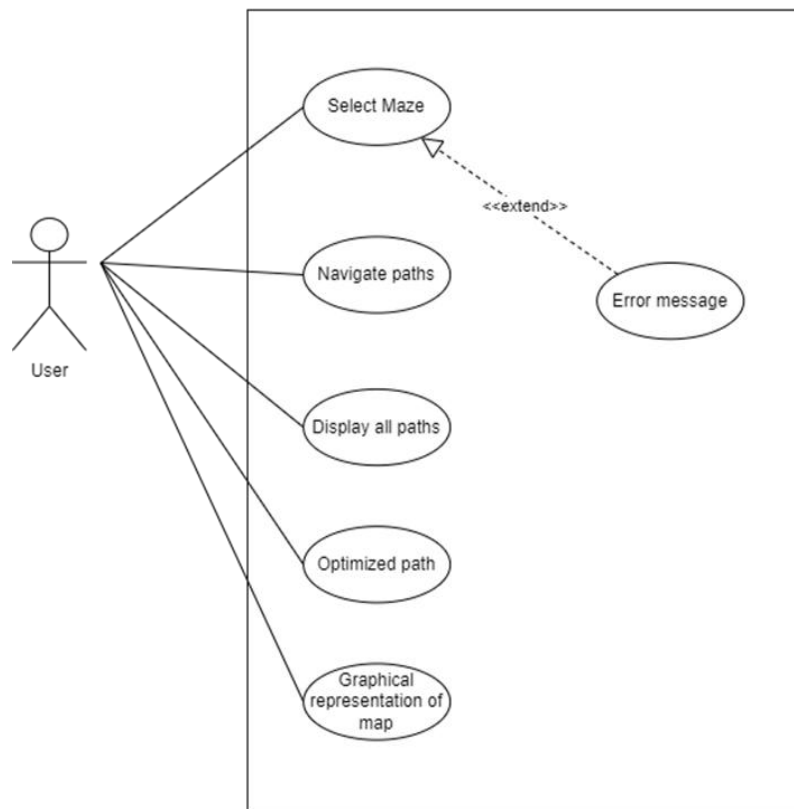
Flowchart:

- Illustrates the sequential flow of operations within the maze-solving algorithm.
- Provides a high-level overview of the decision-making process involved in finding the solution path.



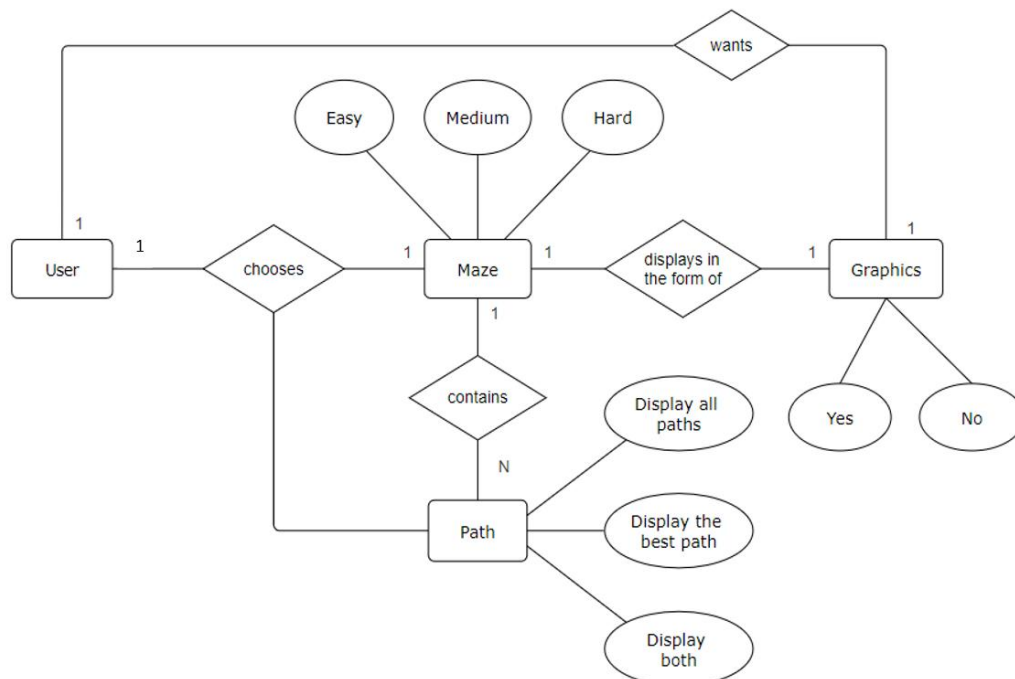
Use Case Diagram:

- Identifies the interactions between the user and the system.
- Demonstrates the various actions users can perform within the maze-solving program.



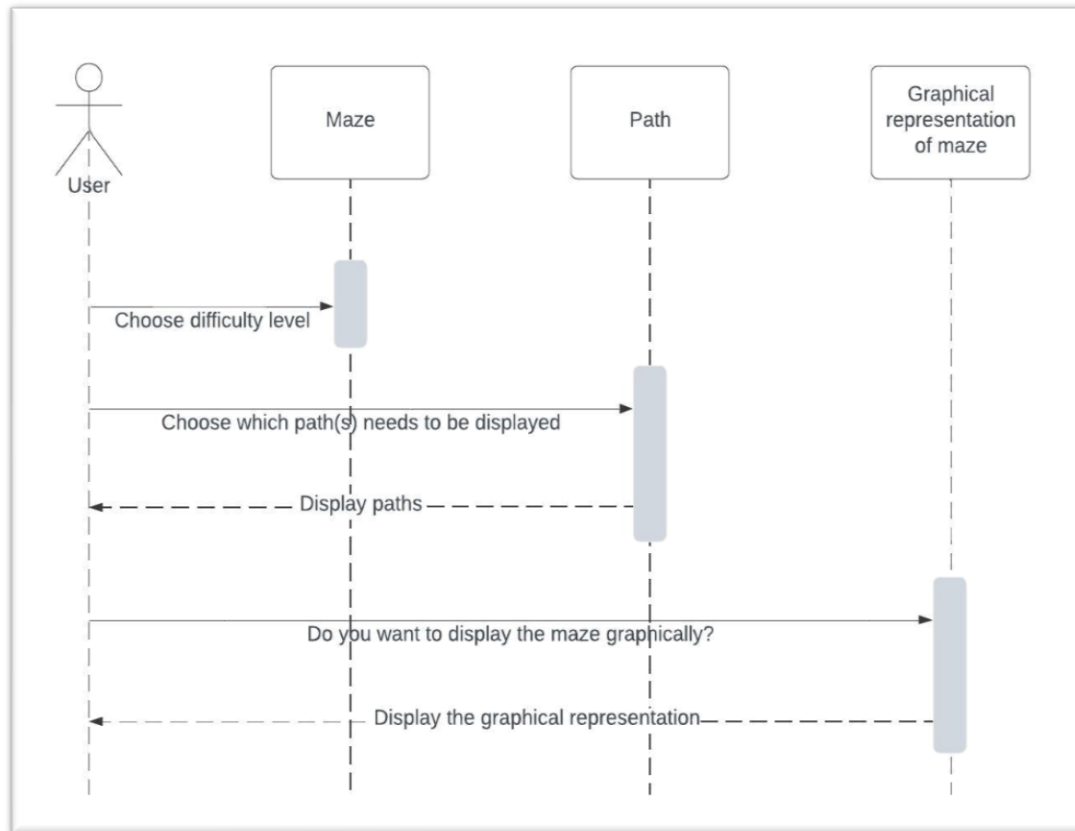
#### Entity-Relationship (ER) Diagram:

- Models the **relationships between different entities** such as maze, solution path, and user preferences.
- Clarifies how different components of the system are interconnected and interact with each other.



Sequence Diagram:

- Depicts the **sequence of interactions between system components** during maze solving.
- Illustrates the order of messages exchanged between objects and how they collaborate to achieve maze solution.



## 5. Miscellaneous:

Abbreviations:

- JVM: Java Virtual Machine
- GPU: Graphics Processing Unit
- API: Application programming interface
- GUI: Graphical User Interface
- CLI: Command-Line Interface

References:

- **GeeksForGeeks: Rat in a Maze**

Date Published: 26 Mar, 2022

<https://www.geeksforgeeks.org/rat-in-a-maze/amp/>



- **YouTube: Rat in A Maze | Backtracking**

Channel Name: take U forward

Date published: 17 May 2021

<https://youtu.be/bLGZhJlt4y0?si=U32Z6Av74HAX-BHm>

- **JavatPoint: Rat in a Maze Problem in Java**

Date published: 29 April, 2020

<https://www.javatpoint.com/rat-in-a-maze-problem-in-java>