# Elabyrinth Game Application
# High Level Design Document

**History :**

| Elabyrinth Game Application | | | | |
|---|---|---|---|---|
| **Date** | **Version** | **Author** | **Brief Description of Changes** | **Approver** |
| | 1.0 | | Initial draft | |
| | | | | |
| | | | | |

# Introduction

## 1. Purpose:

The purpose of this project is to implement a maze game that can be played by a single user, the game involves walls and bombs that will restrict/guide the player. The user can exit a game only when he finishes the game or he quits.

## 2. Intended Audience:

There is no such specific audience, it could be a student or employee or an organization also. Its a genric game and doesn't requires any technical background to play.

## 3. Project Purpose:

The purpose of this project is to implement a maze game that involves multiple mazes to play from. The game involves varied walls that restrict player movement and bombs that kills the player. The main purpose of such puzzles is not only entertainment but also teaching by increasing our knowledge and stimulating curiosity.

## 4. Key Project Objectives:

1. Ask the player to start/exit the game.

2. Allow users to select the mazes which he wants to play
3. Tell the user when he is about to come in contact with bombs.
4. Create walls that restricts the player movement.
5. Warn user about not moving out of maze.
6. Allow users to track back the movement.

## 5. Project Scope and Limitation

New users can be able to register and register users can be able to login with the limitation of maximum three attempts of incorrect password. And search history can be done up to seven days.

## 6. In Scope:

The games is made for entertainment, idle for players looking out for strategy based games. The game is created in C and used 2-D array concepts. Clients do not need to install any other software on their machine. Nor any network connections are required. The games restricts movement via walls

## 7. Design Overview:

The game comprises of the following modules:

| Name of the Module | main(); |
|---|---|
| Handled by | Bhaskarla Sandeep Sai Kiran |
| Description | The main function serves as the starting point for program execution.<br>It controls program execution by directing the calls to other functions in the program. |

| Name of the Module | report(); |
|---|---|
| Handled by | Bhaskarla Sandeep Sai Kiran |
| Description | This report is used to store the contents that printed on terminal/console. |

| Name of the Module | mazetype(); |
|---|---|
| Handled by | Bhaskarla Sandeep Sai Kiran |
| Description | It asks the dimension of maze the user wants to play and shows the options the user can select. |

| Name of the Module | options(); |
|---|---|
| Handled by | Shivant Kamboj |
| Description | In options function different types of mazes . Player choose any maze which want to play like easy, medium, hard. |

| Name of the Module | rectangle(); |
|---|---|
| Handled by | Shivant Kamboj |
| Description | Rectangle function is used for printing the Rectangle maze. |

| Name of the Module | square(); |
|---|---|
| Handled by | Shivant Kamboj |
| Description | Square function is used for printing the Square maze. |

| Name of the Module | readFile(); |
|---|---|
| Handled by | Shivant Kamboj |
| Description | In readFile switch case is running if player choose square then case 1 is running<br>If player choose rectangle then case 2 is running. |

| Name of the Module | readMazeCSV(char* fileName); |
|---|---|
| Handled by | Anadi Mishra |
| Description | This function is the core part of our game it is responsible for reading all the different maze file and maintaining the records of it. |

| Name of the Module | rungame(); |
|---|---|

| | |
|---|---|
| Handled by | Anadi Mishra |
| Description | The function rungame() is responsible for running the program and it also contain other functions like : key(), bombsuggets(), showdirection() etc that help to run our program smoothly. |

| | |
|---|---|
| Name of the Module | key(); |
| Handled by | Anadi Mishra |
| Description | The key() function is basic and small function that describes that player have got the keys or not from the mazes. |

| | |
|---|---|
| Name of the Module | showdirections(); |
| Handled by | Harsha Vardhan |
| Description | This function is responsible to guide the player how to navigate in our Elabyrinth |

| | |
|---|---|
| Name of the Module | printdirections(); |
| Handled by | Harsha Vardhan |
| Description | Print direction is the most important function from the players perspective because this is the function which shows the player which directions he can possibly make a move and which directions he cannot move. |

| | |
|---|---|
| Name of the Module | bombsuggest(); |
| Handled by | Harsha Vardhan |
| Description | This function warns the player whenever the player is nearby a bomb to make the player choose his/her next move with caution |

| | |
|---|---|
| Name of the Module | move(); |
| Handled by | Aditya Srivastava |
| Description | This function is responsible for movement, we use switch case along with if statements to control the movement of player, the if |

| | |
|---|---|
| | walls are present on any sides we restrict the movement in that directions. |

| | |
|---|---|
| Name of the Module | printmaze(); |
| Handled by | Aditya Srivastava |
| Description | This function prints out the maze for the player. |

| | |
|---|---|
| Name of the Module | hint(); |
| Handled by | Aditya Srivastava |
| Description | This function invokes another function as hint for the user. If the user is stuck at any point and want to see the maze then he can use this hint function. |

| | |
|---|---|
| Name of the Module | dead(); |
| Handled by | Aditya Srivastava |
| Description | This function is called when a player steps into the block with bomb. The game exists as soon as this function is called. |

| | |
|---|---|
| Name of the Module | score(); |
| Handled by | Aditya Srivastava |
| Description | This prints out the number of moves that the user took to complete the maze. |

| | |
|---|---|
| Name of the Module | win(); |
| Handled by | Aditya Srivastava |
| Description | Once you reach the last block the game finishes and so the player wins the challenge and the game exits. |

| | |
|---|---|
| Name of the Module | fclose(); |
| Handled by | Aditya Srivastava |
| Description | Finally we close the report file and save the report. |

## 8. Design Objectives:

1. The maze has to be covered with block walls on edges so the player doesn't go out of the maze.
2. Maze had to be challenging and not have any loopholes.
3. The maximum number of moves a player can make is 1 lakh.
4. Obstacles had to fit into the maze and not inhibit game play more than necessary.
5. Code has to be functional in order to allow gameplay to flow well.
6. The game had to be engaging, balanced and enjoyable for the player to want to continue until the end.

## Design Alternative:

Logic puzzles are an integral part of entertainment which find place in various newspapers, magazines and even mobile applications or web pages. These types of tasks are to find a solution or answer by using reasoning based on knowledge or intuition. Depending on the specific objectives of the puzzle, the solution may require time and patience. The main purpose of such puzzles is not only entertainment but also teaching by increasing our knowledge and stimulating curiosity.

### 2.1 User Interface Paradigms:

In user interface three levels are present -

1. Easy level
2. Medium level
3. Hard level

### 2.2 Maintenance:

Very little maintenance should be required for this setup. An initial configuration will be the only system required interaction after system is put together. The only other user maintenance would be any changes to settings after setup, upgrading software and/or hardware and various

forms of maintenance to keep things running, such as clearing out old data that's not needed anymore, which is generally easier to do checking files. assessing hard disk space. examining folder permissions etc.

## 3. Integration Requirements:

1. Language:  C
2. Tools:  splint, Valgrind, make
3. Compiler:  gcc
4. Linux Environment

## 3.1 Configuration:

**3.1.1: Operating System**: Linux environment Ubuntu 20.04
**3.2.2: Compiler**: gcc