**Project Report: MINE MAZE**

**1. Project Overview**

**MINE MAZE** is a console-based game inspired by Minesweeper and any typical Maze Game. The objective of the game is for the player to navigate a grid from Top Left to Bottom Right while avoiding hidden mines and reaching the end goal. The game features two difficulty levels (Easy and Hard), and allows movement through the grid using keyboard controls.

**2. Project Objective**

The main goal of the project is to develop a simple but interactive game that incorporates key programming concepts such as grid manipulation, user input validation, random number generation, and game state management. The project follows a modular approach, breaking down the game into manageable file components.

**3. Features**

**3.1 Dynamic Difficulty Levels**

The game offers two difficulty modes:

* **Easy Mode**: 5x5 grid
* **Hard Mode**: 8x8 grid

These modes allow players to choose their preferred level of challenge.

**3.2 Grid Navigation**

Players move across the grid using the following controls:

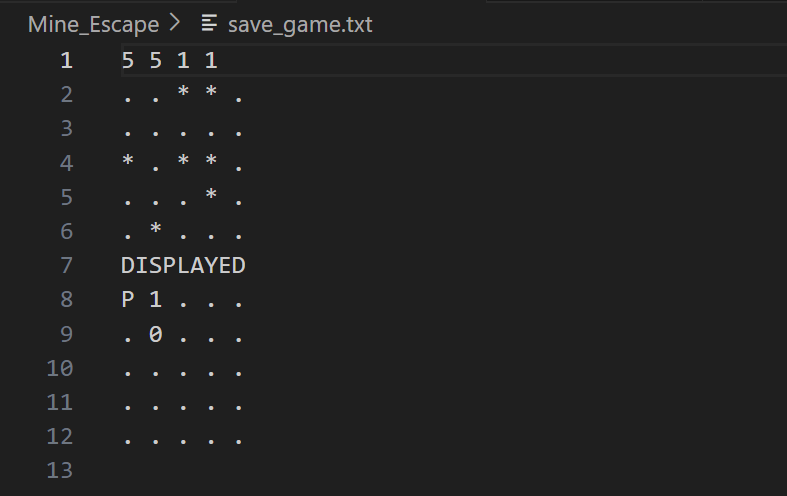
* W: Move up
* A: Move left
* S: Move down
* D: Move right

**3.3 Mine Placement and Detection**

* Mine Layouts are pre-defined and are randomly picked after user makes a choice.
* The number of mines adjacent to each grid cell is displayed when the player moves over it, allowing for strategic decision-making.

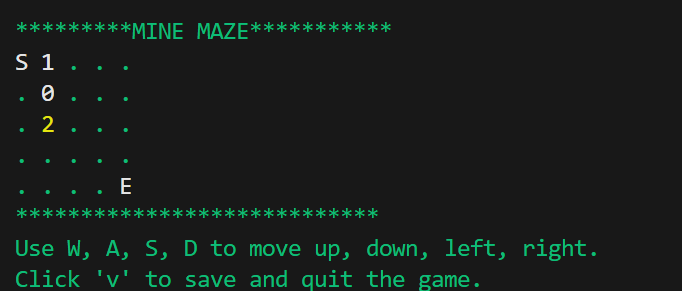
**3.4 Save/ Load Functionality**

* The user can, at any point, decide to save the game at the current grid position and quit the game.
* When playing the game again, user can decide to load the previous game and the exact grid positions will be displayed to them so they can continue the game from there.



**3.5 Modification of the TERMINAL**

* Different Colors are added to the terminal to make the game visually appealing and improve user experience



**3.6 Input Validation**

* Best possible input validation is performed to make sure that there is no anomaly that can affect the user experience

**3.7 Objective**

The goal is to reach the bottom-right corner of the grid, marked as the exit (E), while avoiding mines. Stepping on a mine results in an immediate game over.

**4. Project Structure**

The project is divided into multiple C++ files, each responsible for different aspects of the game:

1. **main.cpp**: This is the core file containing the main game loop and overall functionality. It handles the game's initialization, menu selection, and the game loop.
2. **grid.cpp**: Contains functions related to grid management, including:
   * Initializing the grid.
   * Displaying the grid to the user.
   * Counting adjacent mines for each grid cell.
3. **mine\_layouts.cpp**: Responsible for randomly placing mines on the grid at the start of the game.
4. **player\_movements.cpp**: Manages the player's movement on the grid. It ensures that the movement is within bounds and updates the player's position accordingly.
5. **file\_operations.cpp**: This file will manage functionality for saving and loading the game state, allowing users to pause and resume their game at a later time.
6. **misc.cpp**: Contains utility functions, such as input validation to ensure the player enters valid commands during gameplay.
7. **compile.bat**: A batch file designed to streamline the compilation process. It compiles all source files and runs the program with a single command.

**5. Methodology**

The game was developed using a **modular approach**, breaking down the tasks into separate functions and classes to handle specific features of the game. This approach allows for better organization of the code and easier maintenance.

**5.1 Grid Setup**

The grid is a 2D array that stores information about each cell, including whether it contains a mine and the number of adjacent mines. The grid is dynamically created using pointers based on the selected difficulty.

There exists another grid called the displayed grid which is the one that is shown to the player and contains ‘S’ and ‘E’ at the starting and ending positions and hints on the cells that are revealed.

**5.2 Random Mine Placement**

The “srand”, “time” and “rand” from the “ctime” and “cstdlib” are used to pick a random number which corresponds to the index of an array that stores different mine layouts and picks one.

The original grid is then cloned with the picked layout.

**5.3 Input Handling**

User input is captured via the keyboard using the W, A, S, and D keys for movement. The program validates input to ensure that only valid commands (i.e. the move doesn’t take the player out of bounds) are processed.

**5.4 Game Loop**

The game runs in a loop until the player either reaches the goal or steps on a mine. The displayed grid is compared with the original grid to check what the player stepped on. During each iteration of the loop, the grid is displayed, the player's position is updated, and the number of adjacent mines is shown.

**5.5 Save/Load Functionality**

File handling operations are used to save and load game. The current player column and rows along with both grids are written on the file when the player wants to save the game. The information is then read and the grids are cloned when the user wants to load the game.

**5.6 Terminal Modification**

Different colors are added to the terminal using a header file called “color.h” which used ANSI escape codes and key messages are highlighted using message boxes.

**6. Challenges and Solutions**

**6.1 Random Mine Placement**

Placing mines randomly using “rand” alone for co-ordinates of mines on the grid while ensuring that the starting and goal positions are not affected and the path is not completely blocked posed a challenge. This was resolved by pre-defining different layouts and using the “rand” function then to pick a layout. This approach was considered because otherwise, different algorithms that go out of our knowledge domain, were to be used.

**6.2 Input Validation**

Another challenge was ensuring that the player inputs valid commands and does not exceed the grid boundaries. This was addressed by implementing a function to validate input and prevent the player from moving out of bounds or entering invalid keys.

Also, during the course of the game, multiple character inputs were required, so an appropriate function for input validation was created with the help of AI that not only ensures that the input is within the allowed options but also accounts for different data types and longer inputs.

**6.3 Difficulty Adjustment**

Managing different grid sizes for Easy and Hard modes required dynamic allocation of memory for the grid. This was accomplished by adjusting the grid size based on the selected difficulty.

**7. Future Enhancements**

Several features could be added to enhance the game further:

* **Automated random mine placement**: Implementing harder algorithms to avoid hard-coding the mine layouts.
* **Graphical User Interface (GUI)**: A GUI could be implemented to replace the text-based interface, providing a more engaging experience.
* **Custom Difficulty**: Allow players to set their own grid size and number of mines for a fully customizable difficulty level.

**8. Conclusion**

In conclusion, **MINE ESCAPE** effectively demonstrates the application of key programming concepts such as grid manipulation, input validation, and modular design. Although the game is simple in design, it provides a solid foundation for future enhancements. Adding features like a graphical interface, and customizable difficulty settings would improve the overall experience. This project has been a great way to apply theoretical knowledge into a practical, functional application, with clear opportunities for further development.