Maze Solver Game – Project Report

Project Title: Maze Solver Game

Course Name: CSE 115

Section: 2

Instructors Name: Mohammad Shifat-E-Rabbi

Group Members:

- Member 1: Arnob Mallik Dhruba
ID: 2512446642

- Member 2: Sumaya Islam Kotha

ID: 2511242642

- Member 3: Samrin Fatema

ID: 2513373042

- Member 4: S.M. Robiul Islam

ID: 2513924642

Submission Date: April 16, 2025

1. Abstract

The Maze Solver Game is a console-based project written in C language, designed to simulate the experience of solving a maze using keyboard controls. The main objective of the game is to navigate a character, represented by P, from the start position at the top-left corner of the maze to the exit located at the bottom-right corner.

This project incorporates fundamental concepts of C programming such asstructures, loops, arrays, conditional statements, functions, and file handling. TheGame emphasizes the understanding of control flow, user input handling, and real-time screen updates using system-level functions. The report covers thedevelopment process, system design, flowchart, implementation, membercontributions, testing, and future enhancements.

By using a minimalistic yet effective approach, the project demonstrates how even basic tools and syntax in C can be combined to build engaging and interactive user experience. This project helps to enhance logical reasoning, debugging skills, and teamwork among students.

2. Introduction

Objective:

The objective of this project is to design and implement a simple interactive game in the C language that simulates the process of solving a maze. The focus is onutilizing basic programming constructs to develop an engaging application.

Background and Significance:

Maze games provide an engaging way to learn programming logic. They combine conditionals, iterations, and data structures to simulate decision-making and pathfinding. This project, tailored for beginners, excludes advanced libraries and focuses purely on fundamentals. In computer science education, practical games like maze solvers serve as the gateway to more complex topics such as graph Theory AI path-finding algorithms, and optimization.

Additionally, by creating a game from scratch, students explore how simple logic can be transformed into a functioning product that mimics real-world decision-making.

3. Literature Review

Overview of Text-Based Games:

Text-based games rely entirely on textual input and output. They emphasize storytelling, decision-making, and logical flow. These games were instrumental in the early days of computer gaming, laying the foundation for modern interactive entertainment.

Evolution of Text-Based Games:

From classics like " Zork" to modern interactive fiction, text-based games have evolved significantly. They have transitioned from simple command-line interfaces to complex narratives with branching storylines, showcasing the versatility and enduring appeal of this genre.

Existing Similar Games (Related Works):

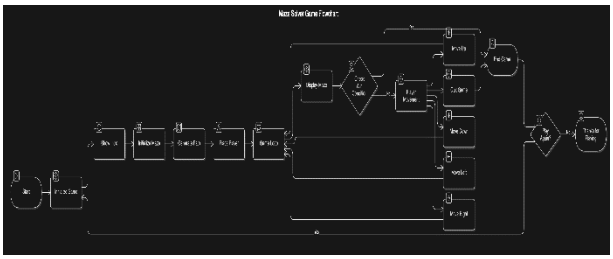
Several projects have explored maze-solving games:

- **Kevin Griggs/Maze Game:** A text-based maze game with an AI component, where the Maze is read from a file, and the player navigates using keyboard inputs.
- **Sarthak Rana/Maze-Solver-Game:** A maze-solving game with three difficulty levels, utilizing data structures, backtracking, and graphics in C.
- **Lucas Martins/Maze-Game-in-C:** A maze game created in C, employing abstract data types like trees, graphs, and stacks to represent the maze and facilitate navigation.

These projects provide valuable insights into different approaches to maze generation and

solving algorithms.

4. Flowchart



The flowchart represents the logical flow of the game. The game begins by initializing the game screen and maze. A random path is generated to connect the start to the end. Once the player is placed at the beginning of the maze, the game enters a loop where user input is captured and used to move the player. The loop continues until the player reaches the end cell or chooses to quit.

5. Application Usage Guidelines

How to Compile and Run the Program

1. Open a C compiler (e.g., Code::Blocks, Dev C++).

2. Paste the code into a new .c file.

3. Compile the file.

4. Run the generated executable.

Input Format

- Keyboard inputs:

o W: Move Up

o S: Move Down

o A: Move Left

o D: Move Right

o Q: Quit Game

Output Format

- Console display of the maze with:

o P: Players' current position.

o E: Endpoint.

o #: Walls.

o: Open paths.

- Display of the number of steps taken.

6. Features and Implementation

Game Initialization

- Random maze generation using a simple algorithm.
- Ensure there's always a path from start to end.

Movement Handling

- Player movement restricted to open paths and endpoint.
- Invalid movements into walls are ignored.

Game Loop

- Continuously takes input until the player quits or wins.

- Updates the maze display after each move.

Code Snippet – Maze Generation Logic:

c

```c
void generatePath(struct Maze *m)
{
    int x = 0, y = 0;
    m->grid[y][x] = ' ';

    while (x != WIDTH - 1 || y != HEIGHT - 1)
    {
        if (x < WIDTH - 1 && (y == HEIGHT - 1 || rand() % 2 == 0))
        {
            x++;
        }
    else if (y < HEIGHT - 1)
        {
            y++;
        }
        m->grid[y][x] = ' ';
    }

    m->grid[0][0] = 'P';
    m->grid[HEIGHT - 1][WIDTH - 1] = 'E';
}
```

7. Member Contribution

Feature/Task	Implemented By	Description
Maze Generation	Arnob Mallik Dhruba ID: 2512446642	Developed the logic for random maze paths
Player Movement Handling	Sumaya Islam Kotha ID: 2511242642	Implemented controls for player movement
Game Loop and Input	Samrin Fatema ID: 2513373042	Managed the main game loop and input handling
UI and Output Display	S.M. Robiul Islam ID: 2513924642	Designed the console interface and display logic

8. Testing and Results

Test Case 1:

- Input: Player navigates correctly through the maze.
- Expected Output: Reaches endpoint in X steps.
- Result: ✅ Passed

Test Case 2:

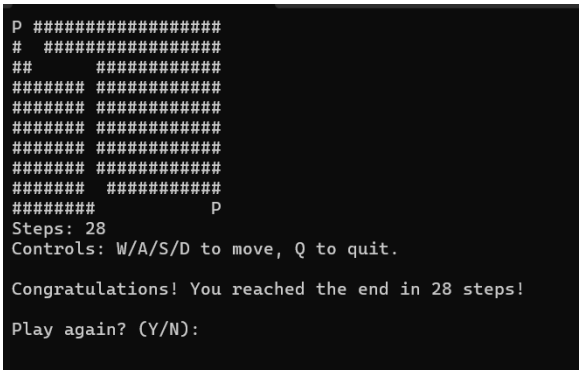
- Input: Player attempts to move into walls.
- Expected Output: Movement denied; position unchanged.
- Result: ✅ Passed

Test Case 3:

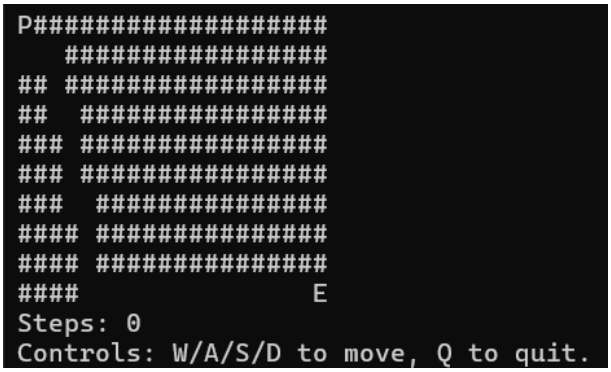
- Input: Player inputs an invalid key.
- Expected Output: No action taken; prompt remains.
- Result: ✅ Passed

Screenshot:

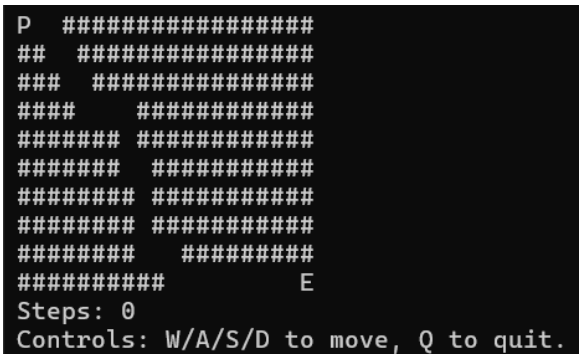
Test Case 1:



Test Case 2:



Test Case 3:



9. Challenges and Solutions

Challenges:

- **Maze Solvability:** Ensuring that the randomly generated maze always has a solvable path.
- **User Input Handling:** Managing unexpected or invalid inputs without crashing the program.
- **Screen Refreshing:** Providing a smooth user experience by effectively clearing and updating the console display.

Solutions:

- **Maze Generation Algorithm:** Implemented a controlled randomization technique that guarantees at least one path from start to end.
- **Input Validation:** Incorporated checks to handle unexpected inputs gracefully.
- **Console Management:** Utilized `system

10. Methodology

Problem Definition

Game Concept & Storyline

The Maze Solver Game is a simple text-based game where the player must navigate a randomly generated maze from the top-left corner to the bottom-right corner. The concept focuses on exploration, decision-making, and logical reasoning to solve the maze efficiently. There is no traditional story: instead, the gameplay experience is the central theme.

Key Features

- Random maze generation
- User-controlled player movement (W, A, S, D keys)
- Step counter to track player efficiency
- Victory detection on reaching the endpoint

Code Architecture

NPC

The game does not include Non-Playable Characters (NPCs), as it a single-player maze-solving experience. However, the maze itself acts as a dynamic obstacle course that reacts to player inputs.

Decision Tree

While no AI decision tree is implemented, the player's movement and game logic form a conditional structure resembling a decision tree for handling input and movement consequences.

Handling Text (Input/Output)

Text handling includes:

- **Input:** Keyboard control using getch() to capture real-time inputs
- **Output:** Console display of the maze, player position, and game status using printf()

Game Loop & Logic Handling

The core game loops continuously:

1. Displays the maze

2. Takes user input

3. Validates input and updates player position

4. Checks for winning condition or exit

Save & Load

Save and load features are not implemented in this version. Future iterations may store progress using file handling.

Mini Games

This version does not feature additional mini games. Future expansions may include timed

challenges, collectibles, or puzzle-solving sections within the maze.

Problem Faced & Debugging

- **Screen Flicker:** Solved using system ("cls") to refresh display.
- **Maze Path Issues:** Ensured at least one path from start to end using guided randomization.

- **Input Crashes:** Prevented using getch() and conditional checks.

11. System Requirements

Libraries Used

- stdio.h: Input/output operations
- stdlib.h: Utility functions like rand() and srand()
- time.h: Seeding random number generation
- conio.h: Handling console input
- windows.h: Delay using Sleep() and screen management

Font

- Default terminal font (e.g., Consolas, Courier New) ensures readability in the console window

12. Challenges & Limitations

Challenges

- Random generation balancing randomness and solvability
- Efficiently managing game state without advanced data structures
- Limitations
- Fixed maze size (10x20)
- No graphical interface
- No AI elements or multiplayer support
- No save/load functionality

13. Future Works

GUI Interface

A graphical version of the game can be built using libraries like SDL, OpenGL, or Unity (in C#).

Multiplayer Mode

Enable two players to compete in solving the maze, either side-by-side or in turns.

Smart NPC Interactions

Introduce AI elements such as chase characters or friendly guides to add complexity.

14. Appendices

Code Snippets

Key functions like maze generation, game loop, and movement logic are already provided in the main document.

Walkthrough

1. Launch the program.

2. Read the instructions.

3. Navigate using WASD keys.

4. Reach the end to win.

5. Repeat or exit as preferred.

15. Summary & Conclusion

This project reinforced essential programming skills, including arrays, structures, conditionals, and user input

handling. The Maze Solver Game demonstrates how basic logic can create engaging interactive experiences. With potential expansion, it can evolve into a more comprehensive educational game.

16. References & Biography

Books

- The C Programming Language by Brian W. Kernighan and Dennis M. Ritchie

Articles

- “Text-Based Games and Their Evolution” – Tech Journal Monthly, 2023

Journals

- International Journal of Game-Based Learning (IJGBL), 2021 Issue on Game Mechanics

Websites

- GeeksforGeeks.org
- Tutorialspoint.com

Online Publications

- Various open-source GitHub repositories listed in the Literature Review