

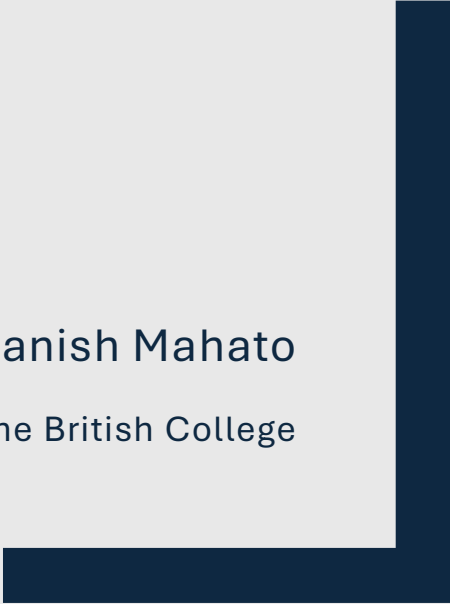


DOCUMENTATION

2D Maze game in C++

Manish Mahato

The British College



Project report: 2D Maze game in C++

1. Project Overview

Title: 2D Maze Game

Language: C++

Platform: Windows (uses `<conio.h>` and `<windows.h>`)

concept: A level-based maze exploration game where the navigates a grid-based maze, collecting items, avoiding enemies, and aiming to reach the exit.

2. Objective

- To develop an interactive console-based game using C++.
- To practice fundamental game logic: movement, collision detection, random generation, levels, and scoring.
- To enhance knowledge of Windows-specific console manipulation.

3. Features

Feature	Description
Maze Generation	Each level generation a new maze with random walls, enemies, and collectibles.
Levels	Total of 4 levels, increasing in difficulty.
Enemies & Obstacles	Dynamic placement of enemies and walls forces the player to strategize.
Collectibles	Players collect '*' to earn points.
Exit Gate	Players must reach 'E' to finish a level.
Limited Moves	Each level has a limited number of moves, increasing slightly each level.
Highest Score Tracking	Keeps track of the highest score across session (within one program run).
Color-coded Elements	Different colors for walls, players, enemies, etc. Using setConsoleTextAttribute() .
Smooth Gameplay	Uses _getch() for real-time key detection without requiring Enter.

4. Game Controls:

- **W:** - Move Up
- **S:** - Move Down
- **A:** - Move Left
- **D:** - Move Right
- **Q:** - Quit current level
- **P:** - Play from main menu
- **E:** - Exit from the main menu

5. Game Design & Flow:

a) Main Menu:

- Display game introduction, controls, and instructions.
- Waits for user game
- **P** to start game
- **E** to exit

b) Maze Generation

- A 10x10 grid.
- Outer walls are fixed.
- Random placement of:
 - Internal walls
 - Collectibles (*)
 - Enemies (X)
 - Exit (E)

c) Gameplay Loop:

- The player starts at (1,1)
- A move counter limits how many moves can be made.
- On each input:
 - New position is calculated.
 - Checks for:

- Walls: No movement
- Enemies: Game over.
- Collectibles: increase score.
- Exit: Complete level.

- Score is calculated cumulatively across levels.
- Highest score updated when a new score exceeds the previous.

d) Win/Loss condition

- **Lose:**
 - Touch an enemy
 - Run out of moves
- **Win:**
 - Complete all levels by reaching the exit.
 - Final congratulatory screen with score displayed.

6. Technical Implementation

a) Data Structures:

- **std::vector<std::vector<char>> maze:** 2D array to store maze structure.
- **pair<int, int> playerPos:** Stores current position of the player.
- Game variables: **score**, **highestScore**, **movesLeft**, **level**.

b) Console Functions

- **SetConsoleTextAttribute():** Sets text color for different elements.
- **SetConsoleCursorPosition():** Moves cursor for smooth screen updates.
- **system("cls"):** Clears screen (Windows-specific).
- **_getch():** Reads input instantly.

c) Randomness

- **srand(time (0)):** Seeds the random number generator.
- **rand ()** used to generate wall, collectible, and enemy positions.

7. Challenges Faced

Challenge	Solution
Handling real-time input	Used _getch() to avoid blocking input
Avoiding infinite loops in placement	Validated position to prevent overlapping walls/object
Smooth rendering	Used goTopLeft and screen clear techniques
Keeping the game fun but fair	Balanced enemy and collectible counts per level.

8. Possible improvements

- Add **enemy movement** logic for more difficulty.
- Introduce **power-ups** or traps.
- Save **high scores** to a file.
- Create a **maze editor** for custom levels.
- Add sound using Windows APIs or external libraries.
- Improve visual UI with **ASCII art** or even shift to **SFML** or **OpenGL** for GUI rendering.

9. Learning Outcomes

- Deepened understanding of game loops, collision detection, and level design.
- Strengthened knowledge of C++ standard libraries.
- Gained experience working with Windows-specific console functions.
- Improved code structure using classes and clean UI design principles.

10. Uses

- File I/O Operations
- 2D array manipulation
- User input handling
- Game state management
- Basic collision detection

11. Conclusion

The 2D Maze Game is a foundational yet feature-rich console game project. It demonstrates solid principles of C++ game development and interactive design. Through multiple levels, randomization, and scoring mechanics, it provides a challenging yet fun gameplay experience.