Shell Jumper

Session: 2025 – 2029

Submitted by:

Mirza Talal Ahmed 2025(S)-CS-67

Supervised by:

Mr. Laeeq Khan Niazi

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Contents

Title: *Shell Jumper: A 2D Platformer in C++*

---

1. Project Overview

This project is a **console-based 2D platformer game**, developed using **C++**. Inspired by classics like *Super Mario*, the game runs in the terminal and incorporates basic platforming mechanics such as jumping, collecting coins, defeating enemies, and transitioning through multiple levels. The game makes extensive use of terminal graphics using ASCII characters and ANSI color codes.

---

2. Objectives

- To apply core programming concepts such as loops, arrays, functions, and file handling.
- To gain hands-on experience with object movement, collision detection, and game loops.
- To simulate a real-world programming project with modular code and structured design.
- To create an enjoyable, interactive experience within a terminal environment.

---

3. Tools & Technologies Used

Tool	Description
C++	Core programming language used.
Windows Console	Platform for rendering the game.
ANSI Escape Codes	Used for coloring and styling text in the terminal.
Visual Studio Code	(Depending on environment) used for compiling and debugging.

---

4. Game Features

✅ Multi-Level Gameplay

- The game consists of 3 levels, each stored in a .txt file.
- The player advances to the next level via special transition markers.

✅ Player Mechanics

- Movement: Left and Right using A and D.
- Jumping using W with realistic gravity and physics.
- Health system with 3 lives.

✅ Enemies

- Basic patrolling enemies represented by 'E'.
- Collision with enemies, results in health loss.

✅ Mystery Boxes

- Blocks labeled 'M' that vanish when hit from below.
- Trigger random bonus score points between 10 and 100 when destroyed.

✅ Coins and Score

- Collectibles represented by 'O', increasing the score.
- Displayed in real-time in the HUD.

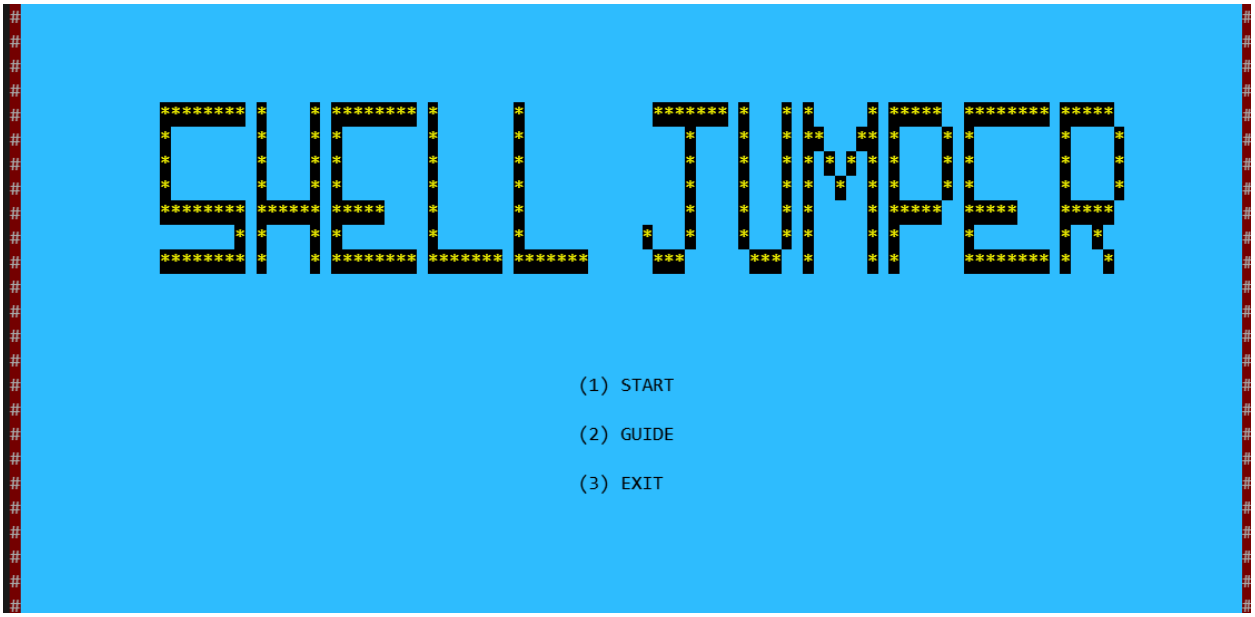
✅ Menu Screens

- A structured main menu, guide, win screen, and loss screen.
- All rendered from text files with color formatting.

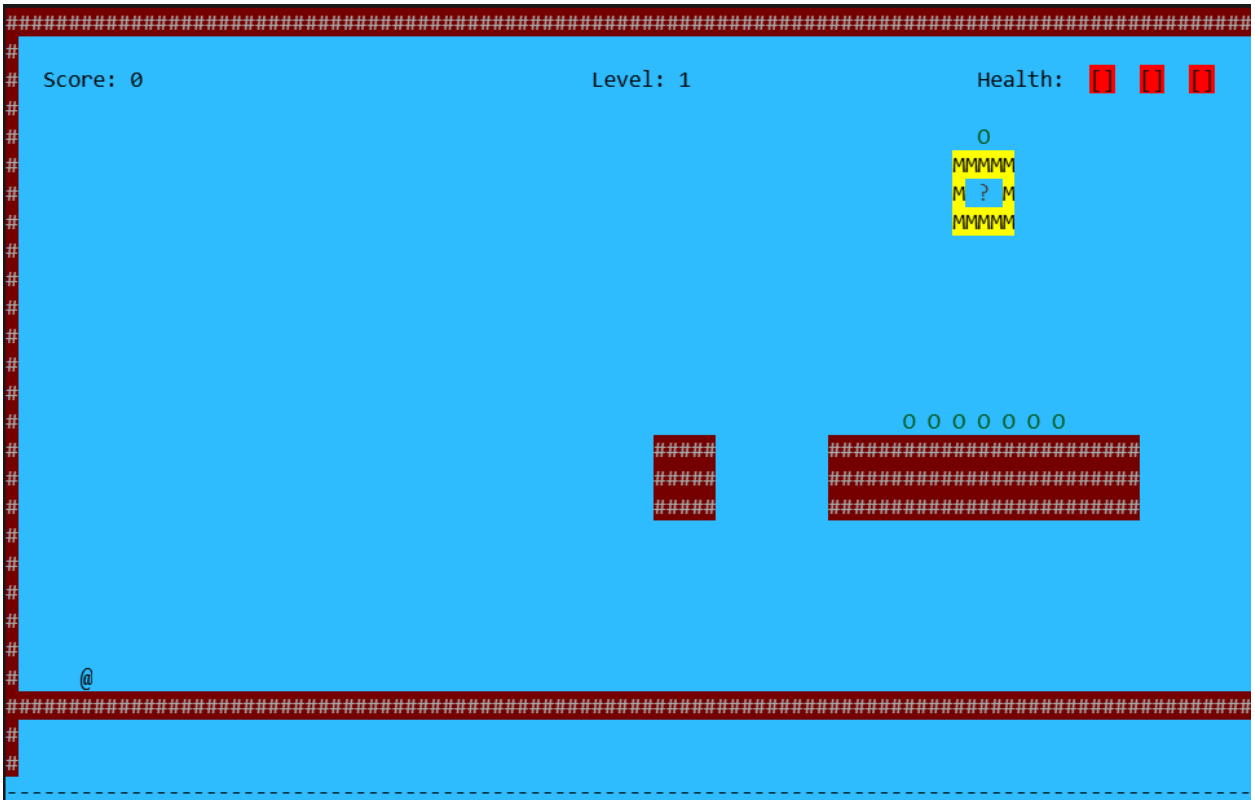
---

5. Gameplay

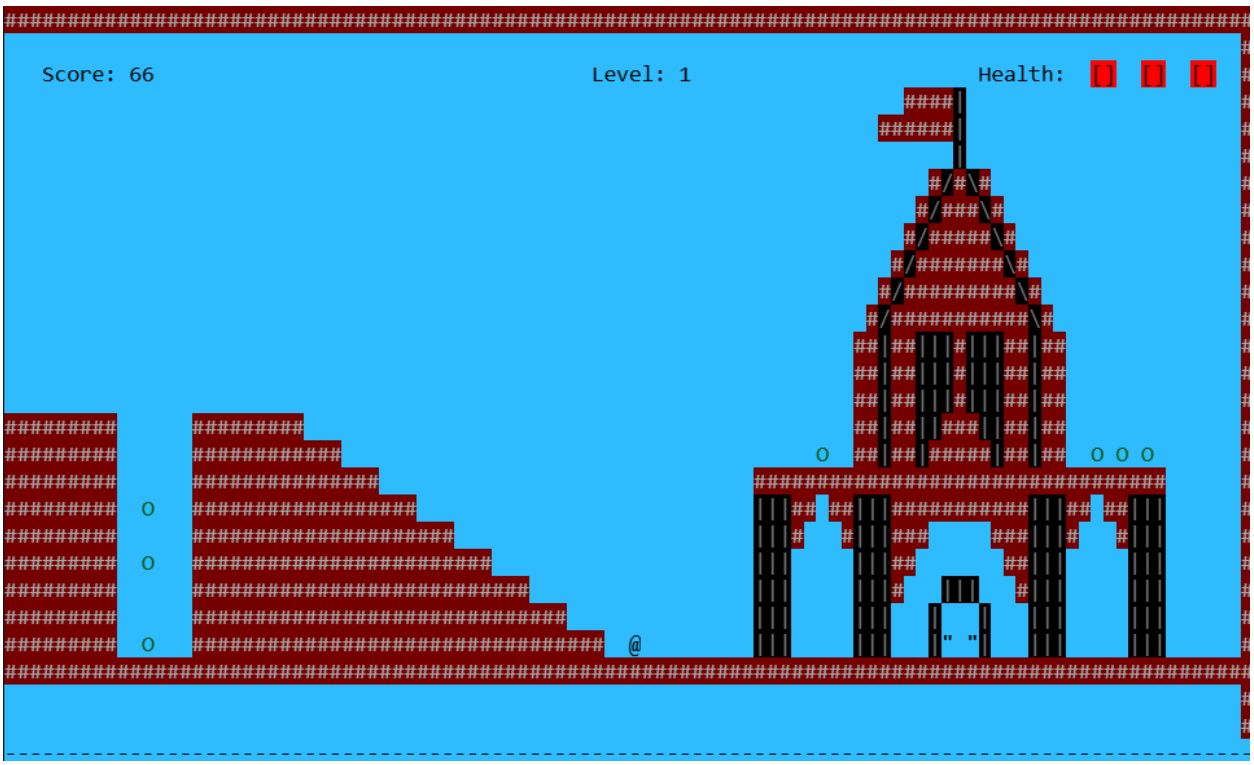
- **Main Menu**



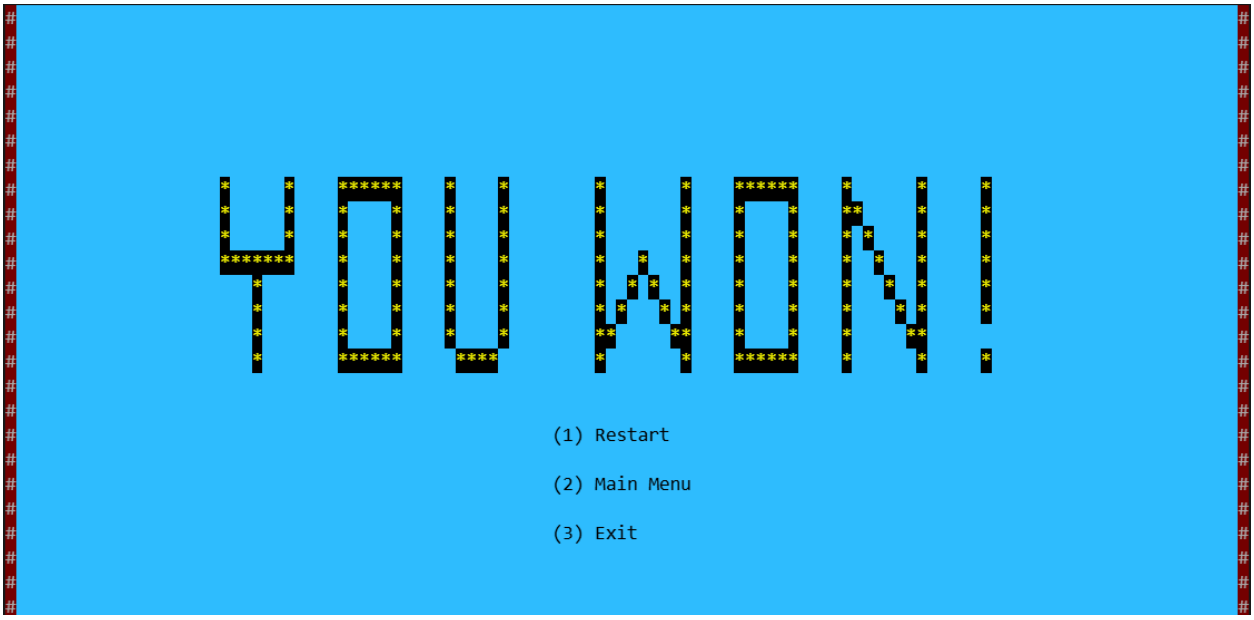
- **Game**



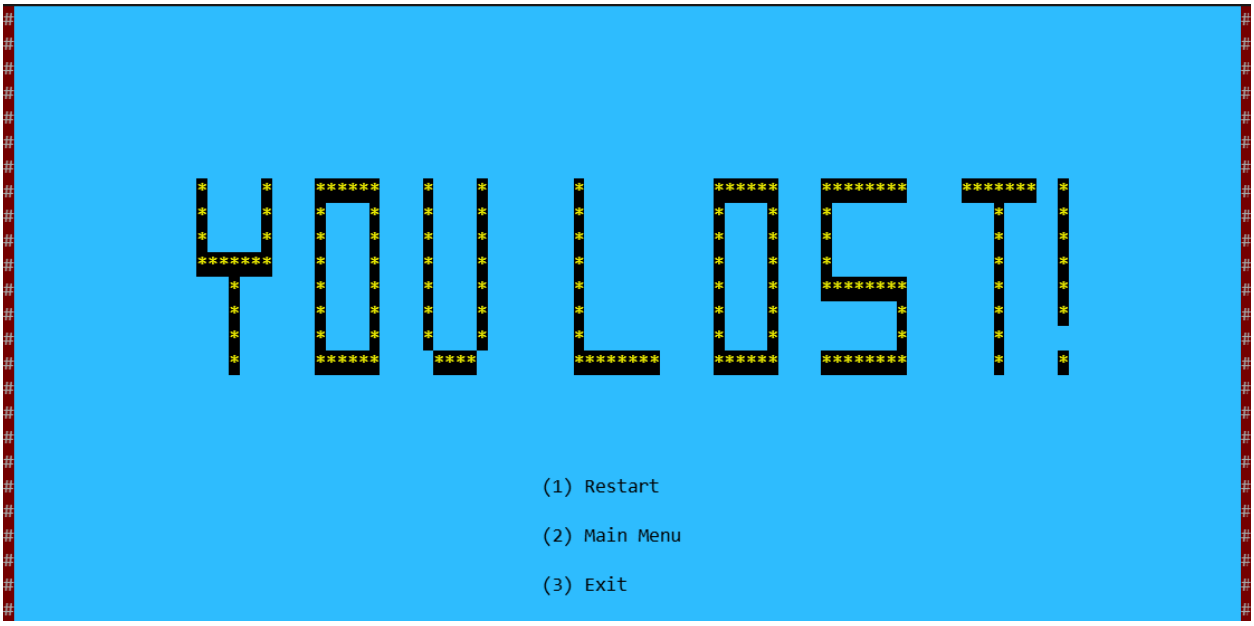
- **End of a level**



- **Win Screen**



- **Loss Screen**



6. Technical Highlights

💡 **File Handling**

- Level and screen layouts are loaded from external .txt files using fstream.

💡 **Game Loop**

- The main() function maintains a continuous loop that listens for input, updates logic, and renders the screen.

💡 **Cursor Management**

- Custom cursor positioning and hiding using windows.h.

💡 **Enemy AI**

- Simple directional patrol logic with obstacle detection and reversal.

💡 **Collision Detection**

- Ground checks, wall collisions, and block destruction handled using getCharAtLevel().

💡 Gravity Simulation

- Smooth vertical motion using float values and physics-based velocity changes.

---

7. Function Prototypes

```
string nonGamePlayScreens(string filename, char nonGamePlayScreen[][NONGAMEPLAYSCREENCOLUMNS]);
bool loadLevel(char level[][levelColumns], string levelFileName);
void handleLevelTransition(string &levelNo, int &cameraX, int &playerX, int &playerY,
    bool &gameRunning, char nonGamePlayScreen[][NONGAMEPLAYSCREENCOLUMNS]);
void updateRender(char level[][levelColumns], char render[][renderWidth], int &cameraX,
    int &score, int &health, string &levelNo);
void drawFrame(char render[][renderWidth]);
char movePlayer(char level[][levelColumns], string direction, string &levelNo, int &playerX,
    int &playerY, int &cameraX);
void gotoxy(int x, int y);
void hideCursor();
bool isGrounded(char level[][levelColumns], int &playerX, int &playerY);
void clearScreen();
char getCharAtLevel(char level[][levelColumns], int x, int y);
void moveEnemy(char level[][levelColumns], int row, int &col, int &direction, char enemyChar);
void clearInputBuffer();
```

8. Challenges Faced

- Managing screen flicker in the terminal during frame redraws.
- Handling real-time keyboard input without blocking the game loop.
- Designing large ASCII-based levels while maintaining readability and balance.
- Debugging transition logic between multiple levels.

---

9. Future Improvements

- Add sound using system beeps or a cross-platform audio library.
- Implement save/load functionality.
- Introduce boss fights or power-ups.
- Optimize rendering to reduce flickering further.
- Add Linux/macOS compatibility by abstracting platform-dependent code.

- Adding momentum in the physics system.

---

10. Codebase

The full codebase includes:

- main.cpp – Game logic and rendering
- level1.txt, level2.txt, level3.txt – ASCII maps
- MenuScreen.txt, Guide.txt, WinScreen.txt, LostScreen.txt – UI screens

---

11. Conclusion

This project has been a valuable learning experience, reinforcing fundamental programming concepts and introducing game logic development. Creating a full-fledged platformer in the terminal environment has demonstrated how powerful even basic C++ concepts can be when applied creatively.