Bullet Hell

Session 2025 - 2029

Submitted by:

Muhammad Luqman Butt

2025-CS-350

Supervised by:

Hafsa Nadeem

Course:

CSC-102L Programming Fundamentals A

Department of Computer Science

University of Engineering and Technology Lahore,Pakistan

Table of Contents

1. Overview

Bullet Hell is a 2D terminal-based platformer developed in C++. It utilizes the Windows Console API to create an interactive environment with real-time physics.

2. Game Characters

1. **The Agent (0):** The player-controlled character. Agile but vulnerable. Can move horizontally and perform jumps.
2. **The Sentries [OO]:** Automated AI drones that patrol the platforms. They are programmed to walk back and forth and will reset the Agent's progress if they collide.

3. Rules & Interactions

1. **Gravity:** The Agent is subject to constant gravity and will fall unless standing on a platform (#).
2. **Collection:** Walking through or jumping into an 'X' crystal adds 1 point to the score and removes the item from the screen.
3. **Hazard Collision:** Contact with any part of a Sentry [OO] results in the loss of 1 Life and immediate teleportation back to the starting point.
4. **World Boundary:** Moving outside the safe X-coordinates (1-84) is treated as a hazard.

4. Goal of the Game

The ultimate goal is to achieve a **Score of 50** before your **Lives reach 0**.

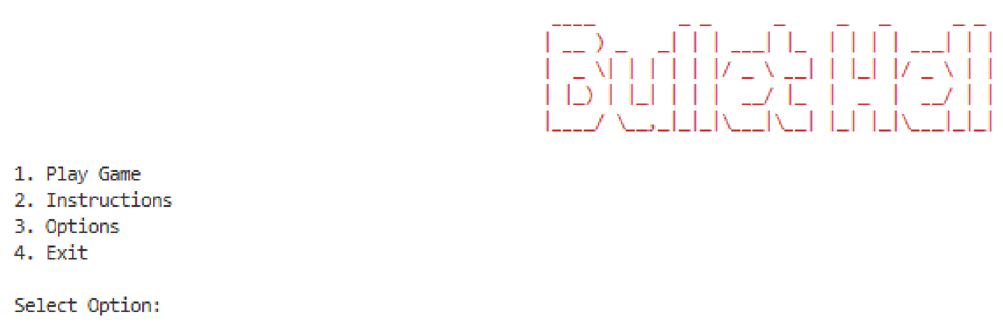
5. Data Structures Used

- **Pass-by-Reference (&):** This is the core architectural choice. By passing variables like player_x, lives, and score by reference, the game avoids global variables while allowing modular functions to update the "Main" game state.
- **COORD Structure:** Part of the windows.h library, used to handle X and Y positions for the console buffer.
- **CHAR_INFO:** Used to "read" the characters currently displayed on the screen to

determine collisions before the player moves.

6. Screenshots

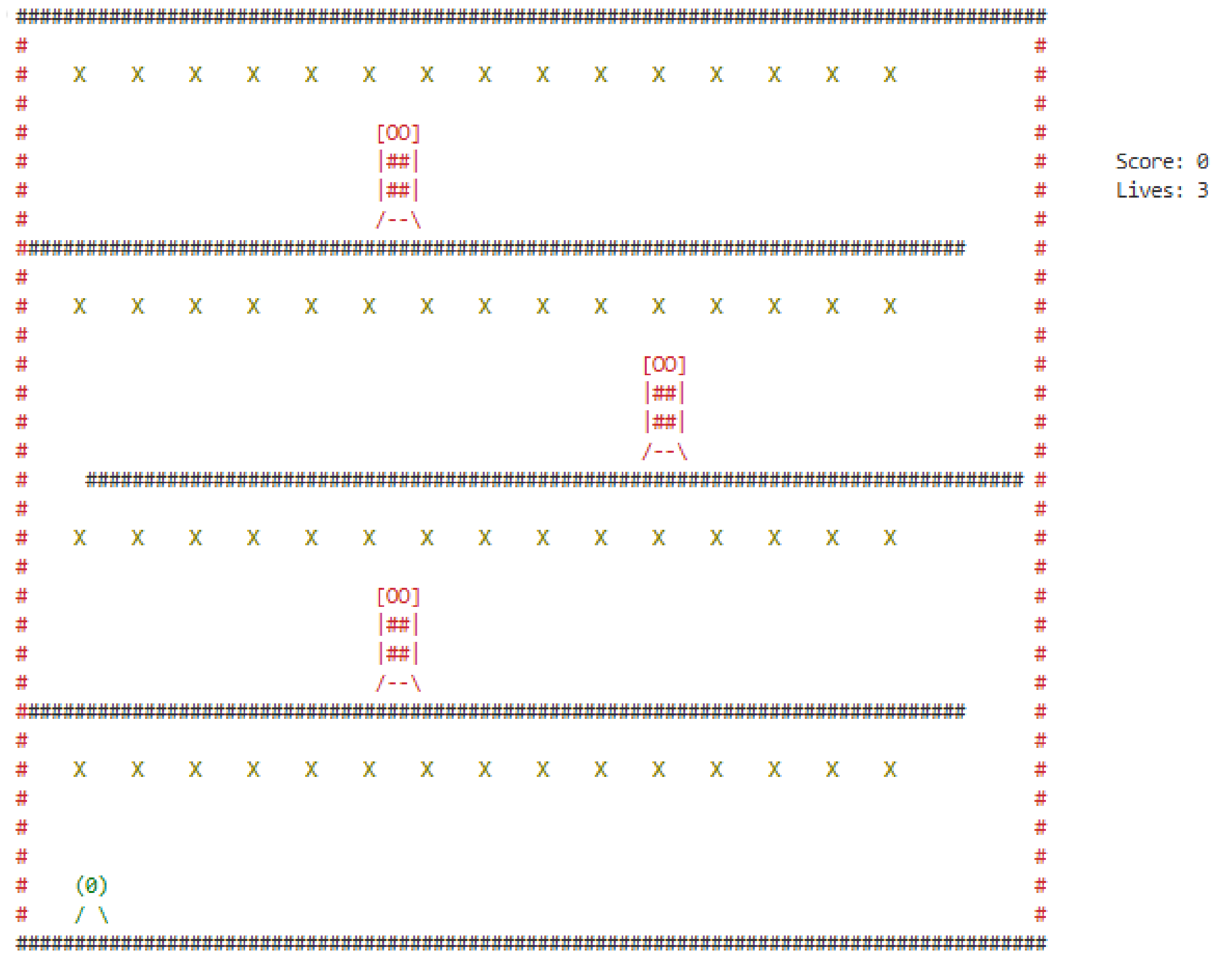
Interface:



```
1. Play Game
2. Instructions
3. Options
4. Exit

Select Option:
```

Maze:



```
Score: 0
Lives: 3
```

Player:

```
(0)
/ \
```

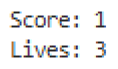
Enemy:

```
[OO]
|##|
|##|
/--\
```

Bonus Points:

```
X    X    X    X    X    X    X    X    X    X    X    X    X    X    X
```

Score and Live:



```
Score: 1
Lives: 3
```

7. Function Prototypes

The following prototypes are categorized by their role in the game's engine:

A. System & Console Utilities

Manages the Windows console environment and cursor state.

- void hideCursor();
- void gotoxy(int x, int y);
- char getCharAtxy(int x, int y);

B. Player Movement & Physics

Handles input, horizontal translation, jumping arcs, and gravity.

- void movePlayerRight(int &player_x, int player_y, int &score, int &lives);
- void movePlayerLeft(int &player_x, int player_y, int &score, int &lives);
- void handleJumpInput(int player_x, int player_y, int &jumpSteps);
- void applyJump(int &player_x, int &player_y, int &jumpSteps, int &score, int &lives);
- void applyGravity(int &player_x, int &player_y, int &score, int &lives, int jumpSteps);

C. Enemy AI & Hazard Detection

Manages automated patrol paths and character-hazard intersections.

- void moveEnemies(int &enemy1_x, int enemy1_y, int &d_enemy1, int &enemy2_x, int enemy2_y, int &d_enemy2, int &enemy3_x, int enemy3_y, int &d_enemy3);
- void checkPassiveHit(int &player_x, int &player_y, int &lives);

D. Rendering & Visuals

Prints maze boundaries, collectibles, and character sprites.

- void print_title();
- void printMaze();
- void print_bonus();

- void draw_player(int x, int y);
- void remove_player(int x, int y);
- void draw_enemy1(int x, int y);
- void draw_enemy2(int x, int y);
- void draw_enemy3(int x, int y);
- void remove_enemy_1(int x, int y);
- void remove_enemy_2(int x, int y);
- void remove_enemy_3(int x, int y);

E. Game State Management

Handles level resets, menus, and win/loss logic.

- void handleReset(int &player_x, int &player_y, int &lives);
- void display_instructions();
- bool collision(char nextHead, char nextFeet);
- bool checkGameStatus(int score, int lives);

8. Conclusion

The development of the **Bullet Hell** project has been a significant milestone in strengthening my technical programming and architectural logic skills. By engineering real-time gravity physics, implementing precise collision detection using the Windows API, and designing modular enemy patrol systems, this project has deeply enhanced my understanding of game loops and state management.

Applying pass-by-reference techniques to maintain a clean architecture without global variables provided invaluable experience in memory management and modular code design. Overall, this project has not only solidified my proficiency in C++ but has also provided a comprehensive foundation in problem-solving and logic formulation essential for advanced software development.