**TANK MANIA**



Session: 2025 – 2029

**Submitted by:**

JAHANZAIB GULL 2025(S)-CS-63

**Supervised by:**

Mr. Laeeq Khan Niazi

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

## Contents

[1. Project Overview 3](#_Toc6556)

[2. Objectives 3](#_Toc6557)

[3. Tools & Technologies Used 3](#_Toc6558)

[4. Game Features 4](#_Toc6559)

[5. Gameplay 5](#_Toc6560)

[6. Technical Highlights 7](#_Toc6561)

[7. Function Prototypes 8](#_Toc6562)

[8. Challenges Faced 8](#_Toc6563)

[9. Future Improvements 8](#_Toc6564)

[10. Codebase 9](#_Toc6565)

[11. Conclusion 9](#_Toc6566)

**Title:** *Tank Mania: A 2D Platformer in C++*

# Project Overview

This project developed a console-based 2D top-down tank shooter in C++. Players control a tank, navigate maze-like environments, avoid obstacles, and engage various enemy tanks across multiple levels with increasing difficulty.

# 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.

# Tools & Technologies Used

|  |  |
| --- | --- |
| **Tool** | **Description** |
| **C++** | Core programming language used. |
| **Windows Console** | Platform for rendering the game. |
| **Version Control** | Git and GitHub. |
| **Visual Studio Code** | (Depending on environment) used for compiling and |

debugging.

# Game Features

* **Player Control**

Arrow keys for movement, spacebar to fire.

* **Dynamic Environments**

Fixed boundary walls and level-specific static obstacles influencing movement and bullet paths.

* **Multiple Enemy Types**

Three distinct enemy tanks (Yellow: vertical, White: horizontal, Purple: diagonal) with varied movement and firing.

* **Bullet Mechanics**

Player and enemy bullets, both affected by obstacle collisions.

* **Health & Scoring**

Player health decreases on collision; score increases by defeating enemies and collecting '$' pills (which also restore health).

* **Multi-Level Progression**

Multiple levels with unique layouts, enemy placements, and adjusted difficulty.

* **Game States**

Main menu, loading, game over, level complete, and win screens.

* **High Score Tracking**

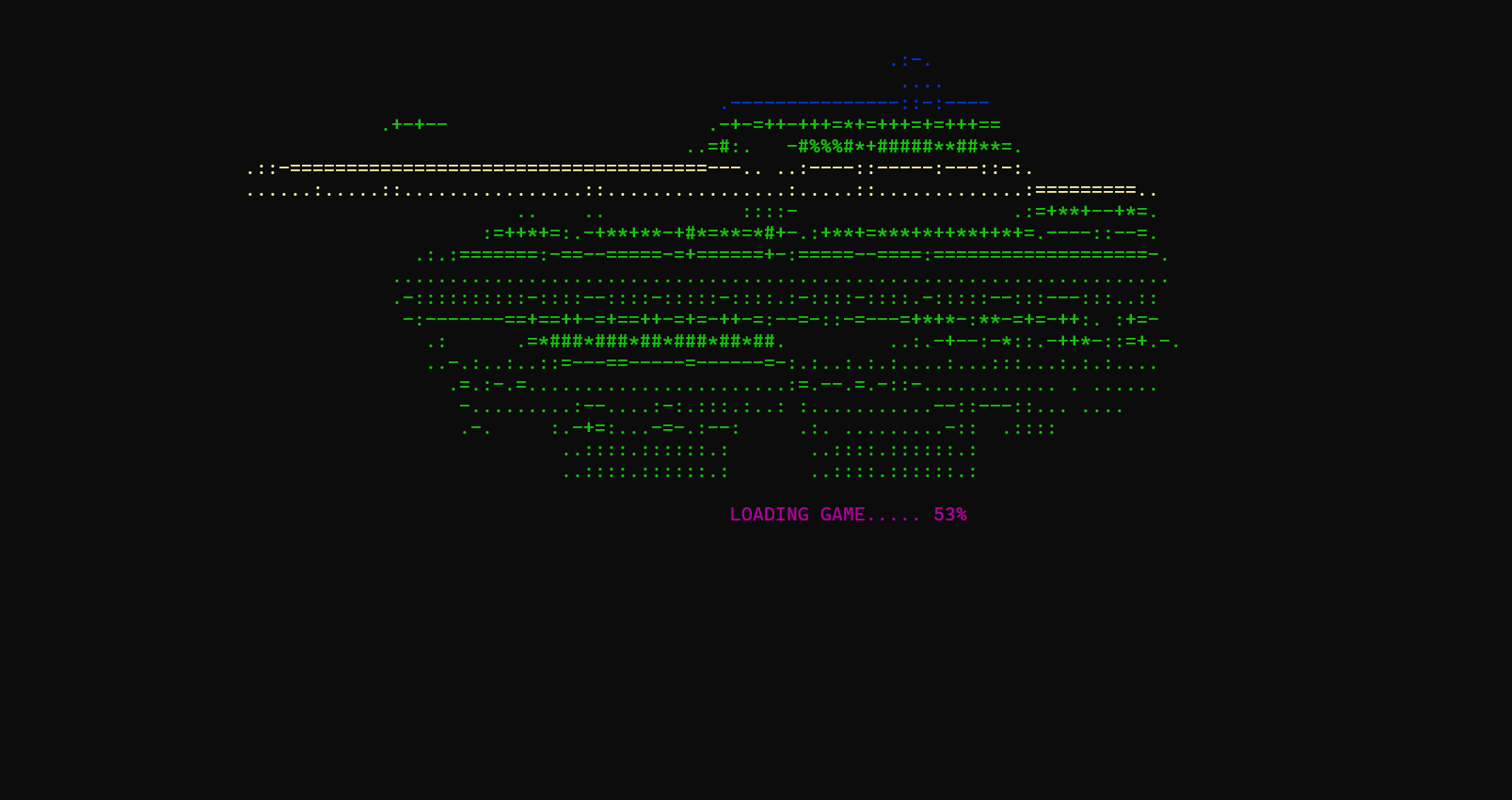
Persistent high score saving/loading using file handling.

# Gameplay

### • Main Menu



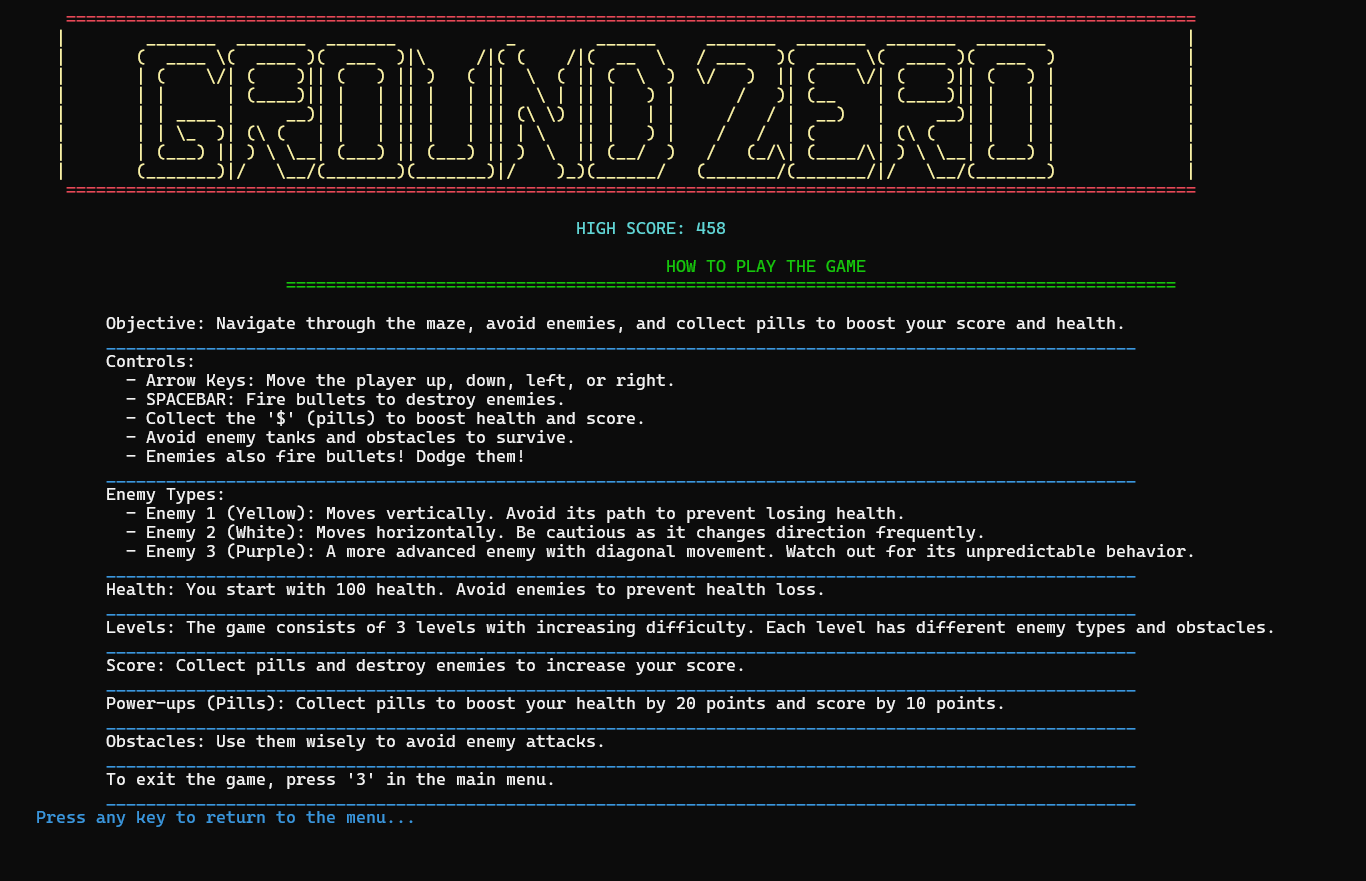
* **Loading Screen**



### • Game Area



### • Instructions



### • Win Screen



### • Loss Screen



* **Level Changing Screen**



# Technical Highlights

* **Console Graphics**

Uses gotoxy() for cursor control and setColor() for colored character-based art, enhancing visual differentiation.

* **Input Handling**

GetAsyncKeyState for real-time, non-blocking input.

* **Collision Detection**

Bounding box collision for all entities and bullets.

* **Game Loop**

while(true) loop manages updates, input, movement, collisions, and rendering, with Sleep() for frame rate.

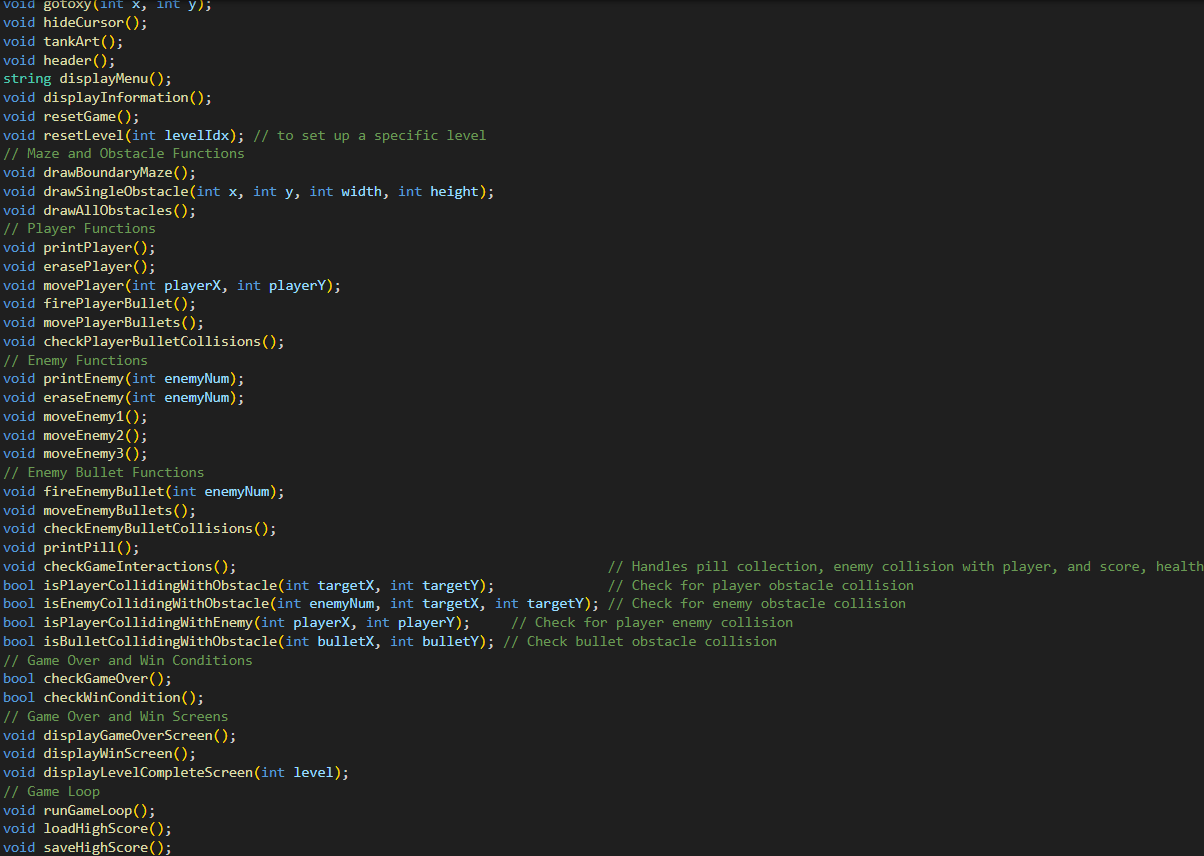
* **File Handling**

fstream for high score persistence.

* **Randomization**

rand() for dynamic enemy fire rates.

# Function Prototypes



# Challenges Faced

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

# 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.
* A refactor to Object-Oriented Programming (OOP) for better modularity.

# Codebase

The full codebase includes:

* **Constants & Global Variables**

Define game parameters, level configurations, and entity states.

* **Main Function**

Manages overall game flow, menu, level progression, and high score.

* **Utility & UI Functions**

Handle console manipulation, art, and menu displays.

* **Game State & Drawing Functions**

Initialize game elements and render them.

* **Movement & Collision Functions**

Update positions and detect interactions.

* **Interaction & Logic Functions**

Manage firing, collisions, scoring, and health.

* **Win/Loss & High Score Functions**

Determine game outcomes and handle persistence.

* **Main Game Loop**

The core logic for continuous gameplay updates.

# 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.