Building a Retro-Style Space Invaders Game: A Beginner-Level Approach

SHOAYEB AHMED NABIL
ID: 2423044042

Project Group: 10
Course: CSE 115
Section: 7
Faculty: Dr. Mohammad Shifat-E-Rabbi [MSRb]

Department of Electrical and Computer Engineering
North South University

GOFRAN F YUSUF
ID: 2422305042
Project Group: 10
Course: CSE 115
Section: 7
Faculty: Dr. Mohammad Shifat-E-Rabbi [MSRb]

Department of Electrical and Computer Engineering
North South University

TAIHAN IBN ALTAF
ID: 2422477642
Project Group: 10
Course: CSE 115
Section: 7
Faculty: Dr. Mohammad Shifat-E-Rabbi [MSRb]

Department of Electrical and Computer Engineering
North South University

Abstract— This project presents the development of a retro-style Space Invaders game, implemented using the C programming language and the 'raylib' library. The objective is to recreate the essence of the classic arcade game while exploring fundamental programming concepts and 2D game development techniques. The game involves a player-controlled spaceship, enemy movements, offering an engaging experience through minimalist graphics and responsive controls. By utilizing raylib, the project demonstrates a beginner-friendly game development framework with C. The report highlights the design process, implementation challenges, and the learning outcomes achieved through the development of this interactive application.

I. INTRODUCTION

Retro arcade games hold a special place in the history of gaming, offering simple yet engaging gameplay that remains popular even today. One such classic is Space Invaders, a game that is simple to play but very fun and challenging. This project aims to recreate the retro-style Space Invaders game while introducing modern programming practices using the C programming language and the raylib library.

The primary motivation behind this project is to explore fundamental programming concepts through the creation of an interactive 2D game. By implementing features like player movement, enemy behavior, collision detection and scoring, the project serves as a practical exercise in logic building, problem solving and software development.

The raylib library was chosen for its simplicity and ease of use, making it an ideal tool for beginners in game development. It provides the necessary functions to handle graphics, input and allowing developers to focus on gameplay mechanics without being overwhelmed by complex configurations.

This report outlines the design and development of the game, covering key aspects such as planning, implementation and challenges faced during the process. The lessons learned from this project will serve as a foundation for future projects in programming and game development.

II. GAME STRUCTURE AND DESIGN

The Space Invaders game was developed using a structured design approach, focusing on simplicity, retro aesthetics, and core gameplay mechanics. The main aspects of the methodology and design are outlined below:

A. *Code Editor and Graphic Library*

The game is implemented in the C programming language using the raylib library. Raylib was selected for its simplicity and ability to handle essential game development tasks such as rendering graphics, detecting collisions, and managing user inputs. The development environment includes a code editor (Visual Studio Code) and a C compiler such as GCC or MinGW.

B. *Game Mechanics*

The game mimics the classic Space Invaders gameplay, incorporating the following elements:

1. **Player Controls**: The player controls a spaceship that moves left or right and fires bullets upward.
2. **Enemies**: A set of enemy ships moves horizontally and randomly shoots bullets downward.
3. **Bullets**: Both the player and the enemies can fire bullets, which interact with other objects.
4. **Collision Detection**: Collisions between bullets and enemies, as well as between enemy bullets and the player are detected to update the game state.
5. **Scoring and Lives**: The score increases when enemies are destroyed. The player starts with three lives and loses one upon being hit by an enemy bullet.

C. Game Components

1. **Player**:
    a. Represented as a triangle-based spaceship drawn using simple geometric shapes.
    b. Moves horizontally based on keyboard input (KEY_LEFT and KEY_RIGHT).
    c. Fires bullets using the KEY_SPACE key.
2. **Enemies**:
    a. Represented as triangular shapes.
    b. Move horizontally and reverse direction upon reaching the screen's edge.
    c. Randomly fire bullets with a low probability.
3. **Bullets**:
    a. Player bullets move upward, while enemy bullets move downward.
    b. Each bullet is represented as a small rectangle and becomes inactive when it leaves the screen or upon collision.
4. **Collision Detection**:
    Utilizes raylib's collision functions to detect interactions between objects, such as bullets hitting enemies or the player.

D. Game Flow and Logic

The game is divided into key states, such as:

1. **Gameplay**: The main loop continuously updates the positions of the player, enemies, and bullets. Collision checks determine if enemies are hit or if the player loses a life.
2. **Game Over**: Triggered when the player loses all lives. A "Game Over" message is displayed and the game ends.

E. Visual Design

The graphics are kept minimal to reflect a retro aesthetic:

1. The player's spaceship and enemies are designed using simple geometric shapes, including triangles and rectangles.
2. The game uses a grey background with color-coded elements: blue for the player, red for enemies, yellow for player bullets, and green for enemy bullets.

F. Development Process

The game was developed in following steps:

1. Initializing the game window and setting up basic elements (player, enemies and bullets).
2. Adding player movement and shooting mechanics.
3. Implementing enemy movement and random shooting.
4. Introducing collision detection for bullets and objects.
5. Refining the visual design and adding score and lives display.

III. IMPLEMENTATION

The implementation of the Space Invaders game involved creating a structured program in C using the raylib library. The key aspects of the implementation are described below.

a) Game Initialization

The game begins by setting up the window and initializing variables:

1. **Window Setup**:
    a. A window of size 750×700 pixels is created using InitWindow().
    b. The frame rate is capped at 60 FPS using SetTargetFPS().
    c. A custom grey background color is defined.
2. **Player Initialization**:
    a. The player is represented by a Player structure containing position, speed, and lives.
    b. The initial position is centered horizontally near the bottom of the screen, with three lives and a speed of 4.0 units.
3. **Enemy Initialization**:
    a. An array of Enemy structures is used to represent multiple enemies.
    b. Each enemy is placed randomly along the horizontal axis and staggered vertically.
    c. Enemies move horizontally and reverse direction upon hitting the screen edge.
4. **Bullet Initialization**:
    a. Two arrays of Bullet structures are used for player and enemy bullets.
    b. Bullets have fixed speeds (upward for the player and downward for enemies) and are initially inactive.

B. Player Mechanics

1. **Movement**:
    a. The player's spaceship moves left or right in response to KEY_LEFT and KEY_RIGHT inputs.
    b. The position is updated directly by modifying the position.x value.
2. **Shooting**:
    a. Pressing KEY_SPACE fires a bullet.
    b. The program iterates through the player bullet array to find an inactive bullet and activates it at the spaceship's current position.

C. Enemy Mechanics

1. **Movement**:
    a. Enemies move horizontally, and their direction reverses when they hit the screen edges.
    b. This is implemented by checking if an enemy's position exceeds the screen boundaries.
2. **Shooting**:
    a. Enemies randomly fire bullets with a 2% chance on each frame.
    b. The program iterates through the enemy bullet array to find an inactive bullet and activates it at the enemy's current position.

D. Collision Detection

1. **Player Bullets and Enemies**:
    a. Each active player bullet is checked against all active enemies.
    b. A collision is detected using CheckCollisionCircleRec(), which checks if a bullet intersects an enemy's bounding rectangle.
    c. Upon collision, the bullet is deactivated, the enemy is destroyed, the score increases and the enemy is respawned.
2. **Enemy Bullets and Player**:
    a. Each active enemy bullet is checked against the player's bounding rectangle.
    b. If a collision is detected, the bullet is deactivated and the player loses a life.

c. If the player's lives reach zero, the game ends with a "Game Over" message.

E. Game Rendering

1. **Player Drawing**:
   a. The player's spaceship is drawn using triangles and rectangles to form a retro-styled ship.
   b. The design includes wings, a body, and additional details like exhaust flames.
2. **Enemy Drawing**:
   Enemies are drawn as triangular shapes with layered components to give a 2D geometric appearance.
3. **Bullets Drawing**:
   Active bullets are drawn as small rectangles, yellow for the player and green for enemies.
4. **Score and Lives Display**:
   The score and the player's remaining lives are displayed at the top of the screen using DrawText().

F. Game Flow

The main game loop handles the following tasks in order:

1. **Input Handling**:
   Reads user input for player movement and shooting.
2. **Game Updates**:
   a. Updates the positions of the player, enemies, and bullets.
   b. Checks for collisions and updates the game state accordingly.
3. **Rendering**:
   a. Clears the screen and redraws all active objects (player, enemies, bullets).
   b. Displays the score and lives counter.
4. **Game Over**:
   If the player's lives reach zero, the game displays a "Game Over" message and exits.

G. Development Challenges

1. **Enemy Bullet Timing**:
   a. Ensuring a balanced firing rate for enemy bullets to avoid overwhelming the player.
2. **Collision Detection**:
   Adjusting the areas where objects can collide to make sure the game detects hits accurately without slowing down performance.
3. **Object Respawning**:
   a. Enemies respawn at new random positions after being destroyed, maintaining gameplay dynamics.

IV. RESULTS

A. Game Functionality

1. **Player Controls**:

   The player's spaceship moves smoothly left and right with the arrow keys. Pressing the spacebar fires bullets, and the controls are easy to use.

2. **Enemy Behavior**:

Enemies move from left to right and reverse direction when they reach the edge of the screen. They also randomly fire bullets at the player, making the game challenging.

3. **Collisions**:

The collision detection works as expected:

a. Player bullets hit enemies and destroy them, increasing the score.
b. Enemy bullets hit the player's ship, causing the player to lose lives. If

the player loses all lives, the game ends with a "Game Over" message.

4. **Score and Lives**:

The score increases each time an enemy is destroyed. The player starts with three lives, and the remaining lives are displayed at the top of the screen. When all lives are lost, the game ends.

B. Visual Design

1. **Graphics**:

   The game has a retro look, with simple shapes representing the player and enemies. The player's ship is made up of triangles and rectangles, while enemies are drawn using triangles. Bullets are small rectangles, yellow for the player and green for the enemies.

2. **Background and Display**:

The background is dark grey, which makes the brightly colored objects stand out. The score and lives are clearly shown at the top of the screen.

C. Performance

The game runs smoothly at 60 frames per second, with no noticeable lag or slowdowns. The simple graphics and efficient collision detection help the game run well, even with many enemies and bullets on the screen.

D. Challenges and Limitations

1. **Enemy Behavior**:

   The enemy behavior is simple, with enemies just moving back and forth and randomly shooting. Adding more complex behaviors in the future could make the game more interesting.

2. **Bullet Overlap**:

Sometimes, multiple bullets can overlap with enemies, which could cause a small error in detecting collisions. This was solved by making sure only the first bullet to hit an enemy counts.

3. **Graphics and Animation**:

The graphics are clear but basic. In the future, more detailed animations and effects could be added to improve the game's look.

E. Future Improvements

1. **Better Enemy Behavior**:

   Future updates could include more interesting enemy movements such as flying in formations or making coordinated attacks.

2. **Power-ups**:

Adding power-ups, like faster shooting or extra lives, could make the game more exciting.

3. **Sound Effects**:

Adding background music and sound effects for shooting, explosions, and other actions would improve the overall experience.

4. **Level Progression**:

The game could include different levels with harder enemies and faster gameplay to keep it more challenging.

V. LEARNING OUTCOMES

This project taught our team a lot about both programming and game development:

Programming Skills: As a team, we gained hands-on experience using C and raylib, improving our understanding of game logic, object management, and event handling. We worked together to structure the game with multiple entities such as players, enemies and bullets and collaborated on implementing collision detection and scoring systems.

Game Development: Working on this project gave us a deeper understanding of how games are created from the ground up. We learned how to balance game mechanics like speed, difficulty and player engagement. Additionally, the project emphasized the importance of debugging and testing, which we all contributed to, ensuring smooth gameplay.

VI. CONCLUSION

The Space Invaders project successfully implemented a classic retro-style game using C and raylib. The basic gameplay mechanics were fully functional, including player movement, shooting, enemy behavior, and collision detection. Despite the simplicity of the graphics, the game offers a fun and engaging experience, staying true to the original concept of Space Invaders.

ACKNOWLEDGMENTS

1. Raylib, "raylib: A simple and user-friendly library for game programming," Available: https://www.raylib.com
2. W3Schools, "C Programming Language" Available: https://www.w3schools.com/c/
3. E. Balagurusamy, *Programming in ANSI C*, 8th edition