

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | SPACE INVADER GAME | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | BYFantastic 4 (F4)  * **Anas Norani** * **Hanan Majeed** * **Noor Ul Saba** * **Hamna Shah**   **BSDS-02-A** |  | | | |
|  | | | | DECEMBER 26, 2024—FUNDAMENTALS OF COMPUTER PROGRAMMING—PROF. ARHAM MUSLIM |  | | | |
|  | | |  | | |  | | |

**Spaceship Game Documentation**

**1. Project Overview**

**Title: Spaceship Game**

**Description:**

The **Spaceship Game** is a 2D action-packed shooting game developed using the **SFML** (Simple and Fast Multimedia Library) framework. The player controls a spaceship that navigates across the screen, shooting bullets to destroy waves of alien enemies while avoiding enemy fire. The game features a progression system, immersive sound effects, and shields to absorb damage. Players can enhance their experience by progressing through difficulty levels that introduce more enemies and challenging gameplay. The ultimate goal is to survive as long as possible while earning points.

**2. Features**

**Core Features**

* **Spaceship Control:**
  + The player can move the spaceship horizontally using the arrow keys (left) and (right) for precision control.
  + The spaceship can shoot bullets using the ‘Spacebar’ key, launching projectiles that destroy enemies and interact with other objects in the game.
* **Enemy Behavior:**
  + Enemies are spawned systematically on the screen.
  + Enemies shoot bullets that aim at the player, and these bullets can damage the player’s spaceship.
  + Upon being destroyed, enemies respawn after a brief period, increasing in numbers as the game progresses.
* **Shields:**
  + Players have access to shields in higher levels that can absorb damage from enemy bullets.
  + The shield has a limited number of hits. It slowly degenerates as it continues to being hit by the enemies.
* **Sound Effects and Music:**
  + **Shooting Sound Effects:** A distinct shooting sound plays whenever the player fires a bullet.
  + **Collision Sound Effects:** Various collision sounds trigger when bullets hit enemies, enhancing the gaming experience.
  + **Background Music:** Continuous, atmospheric background music plays throughout the game to immerse the player in the action.
* **Difficulty Levels:**
  + The game starts with an initial difficulty level but increases in difficulty as the player progresses.
  + As the difficulty rises, the number of enemies increases, and the frequency of enemy bullets escalates.
  + A prompt is displayed after specific game milestones, offering the player an option to increase the difficulty for higher rewards.
* **Score Tracking:**
  + The game tracks the player’s score, reflecting the number of enemies destroyed and levels completed.
  + **High Scores** are saved in a file using **ifstream** and **ofstream** for file handling. The high score is loaded and updated whenever a player achieves a new high score.
* **User Interface:**
  + The game features a clean, user-friendly menu system with options to:
    - Start a new game.
    - View the high scores.
    - See game credits.
    - Quit the game.
  + The score display during gameplay shows the player’s current score and the high score achieved.
  + After ending a game the player’s score is displayed along with the instructions to restart the game (press ‘R’) or Quit the game (press ‘Q’) , Level up the game (press ‘L’) or return to main menu (press ‘M’).

**3. MODULES**

**Classes and Their Responsibilities**

**3.1 Spaceship Class**

* **Purpose:** Represents the player’s spaceship.
* **Responsibilities:**
  + Handles spaceship movement (left and right).
  + Manages the shooting mechanism, firing bullets at enemies.
  + Draws the spaceship(body, left and right wings, and nose) to the screen at its current position.

**3.2 Bullet Class**

* **Purpose:** Represents the bullets fired by the spaceship.
* **Responsibilities:**
  + Handles the movement of bullets across the screen.
  + Detects collisions with enemies or shields.
  + Draws the bullet on the screen as it moves.

**3.3 EnemyBullet Class**

* **Purpose:** Represents the bullets fired by enemies.
* **Responsibilities:**
  + Handles the movement of enemy bullets toward the player.
  + Checks for collisions with the player or the spaceship’s shield.

**3.4 Alien Class**

* **Purpose:** Represents the alien enemies.
* **Responsibilities:**
  + Controls enemy behavior, including movement patterns and shooting.
  + Spawns new enemies after destruction.
  + Implements the respawn mechanism and increases the challenge as the game progresses.

**3.5 Shield Class**

* **Purpose:** Represents the player’s shield.
* **Responsibilities:**
  + Manages shield health, allowing it to absorb damage.
  + Draws the shield representation on the screen.

**3.6 Button Class**

* **Purpose:** Represents interactive buttons in the UI.
* **Responsibilities:**
  + Handles drawing of buttons for the main menu (Start, Credits, High Score and Quit) and other UI components.
  + Detects user interaction, such as key presses.

**3.7 Game Logic Functions**

* **Purpose:** Manages the core logic of the game.
* **Responsibilities:**
  + Collision detection between bullets, shields, and enemies.
  + Updates game state such as score and enemy positions.
  + Manages transitions between different game states, including game-over conditions and restart.

**4. Difficulty Levels Implementation**

**4.1 Game Progression System**

The game starts at an easy difficulty level but gradually increases the challenge as the player advances.

**Functionality:**

* + **Difficulty Upgrade Mechanism:** After every few levels, the player is prompted to increase the difficulty level. The difficulty upgrade introduces more enemies and faster gameplay.
  + The player can choose to stay at the current level or opt for a more challenging one for higher rewards.

**5. Sound Effects and Music**

**5.1 Shooting Sound:**

* A shooting sound is played whenever the player presses the fire button (Spacebar). This adds a satisfying auditory cue to the player's actions.

**5.2 Collision Sounds:**

* Different sounds are played depending on the collision type. When bullets hit enemies or shields, a distinct sound is triggered to signify damage.

**5.3 Background Music:**

* The game plays continuous, non-distracting background music throughout the session, enhancing the gaming experience and maintaining immersion.

**6. Key Functions**

**6.1 loadHighScore()**

* **Purpose:** Loads the saved high score from a file to track player progress.
* **Implementation:** Reads from a file using ifstream to retrieve and display the stored high score.

**6.2 saveHighScore()**

* **Purpose:** Saves the current score as the new high score if it exceeds the previous one.
* **Implementation:** Writes the new high score to a file using ofstream to ensure it is persistent.

**6.3 checkBulletShieldCollisions()**

* **Purpose:** Detects if a bullet has collided with the shield.
* **Implementation:** Checks if the bullet’s bounding box overlaps with the shield’s area, and reduces shield health upon collision.

**6.4 checkShieldEnemyBulletCollisions()**

* **Purpose:** Detects if an enemy bullet has collided with the player’s shield.
* **Implementation:** Similar to the bullet-shield collision detection but for enemy bullets.

**6.5 restartGame()**

* **Purpose:** Resets the game state to its initial conditions, such as resetting the spaceship, bullets, enemies, and score.
* **Implementation:** Reinitializes all game variables and objects, allowing the player to start a fresh session.

**7. User Interface**

**7.1 Menu System:**

The main menu includes options to start the game, view high scores, check credits, and exit the game. Buttons are drawn on the screen, and users can interact with them using the keyboard.

**7.2 Score Display:**

During gameplay, the current score and high Score is displayed at the top of the screen, while the highest score is also visible at the start of game. This gives the player feedback and encourages competition.

**8. Game Loop**

**8.1 Event Handling:**

The game loop constantly checks for user input, including spaceship movement and bullet firing. Events such as key presses and mouse clicks are captured and processed accordingly.

**8.2 Game State Updates:**

The game state is updated at regular intervals. Positions of bullets, enemies, and the spaceship are recalculated, and collisions are checked.

**8.3 Rendering:**

The game screen is cleared and updated each frame, drawing all objects (spaceship, bullets, enemies, etc.) to their new positions. After rendering, the frame is displayed to the player.

**9. Conclusion**

The **Spaceship Game** is designed to deliver an engaging and challenging experience for players. The combination of smooth gameplay mechanics, dynamic difficulty progression, immersive sound design, and intuitive controls creates a game that is both exciting and addictive. As the player progresses, they face increasingly difficult challenges, all while being encouraged to beat their own high scores.

**10. Future Improvements**

**10.1 Additional Levels:**

Introduce new environments with unique obstacles and challenges, providing a sense of progression and diversity.

**10.2 Dynamic Enemy Behaviors:**

Incorporate random enemy movements and fire patterns to make the gameplay more unpredictable and exciting.

**10.3 Power-Ups:**

Add power-ups that grant temporary boosts, such as rapid fire, shield regeneration, or invincibility, for a more varied gameplay experience.

**10.4 Multiplayer Mode:**

Introduce a local multiplayer mode where two players can cooperate or compete against each other, either cooperatively fighting off enemies or versus mode.

**10.5 Enhanced Graphics:**

Upgrade the visual assets and animations to provide a more polished and visually appealing gaming experience.

**How to Play the Spaceship Game**

TO START THE GAME FIRST COPY CODE AND PASTE IT IN VISUAL STUDIO(PREFFERED)

THEN CHANGE THE PATH OF BACKGROUND IMAGE, BACKGROUNG SOUND, BULLET FIRING SOUND, COLLOSION SOUND AND FONT (PROVIDED) ACCORDING TO YOUR PC

WHERE YOU PASTE THESE FILES (SOUND, FONT ETC) LIKE IN OUR CASE IT IS   
C:/Users/H.M/Desktop/hit.wav

C:/Users/H.M/Desktop/gunfire.wav

C:/Users/H.M/Desktop/background.wav

"C:/Users/H.M/Desktop/Roboto-Regular.ttf

C:/Users/H.M/Desktop/space.jpg

AS I PASTED IT ON DESKTOP SO IF YOU PASTE IT ON DESKTOP CHANGE THE PATH ACCORDINGLY TO START THE GAME.  
YOU HAVE TO CHANGE THE PATH OF 5 FILES STARTING FROM LINE NO 590 OF CODE TO ONWARD.

**Game Start: Main Menu**

When the game launches, the **Main Menu** is displayed, featuring four interactive buttons:

1. **Start** - Begins the game at the basic difficulty level.
2. **High Scores** - Displays the highest scores achieved by players.
3. **Credits** - Shows the names of the development team.
4. **Quit** - Exits the game.

Navigate through the menu using the keyboard, and click the respective buttons to make a selection.

**Basic Gameplay**

Upon selecting **Start**, the game begins at the **basic difficulty level** with the following setup:

* **Enemies:** A single row of alien enemies appears at the top of the screen. These enemies periodically fire bullets at your spaceship.
* **Spaceship Controls:**
  + Use the **left** and **right arrow keys** to move the spaceship horizontally across the screen.
  + Press the **Spacebar** to fire bullets at the enemies above.

**On-Screen Information**

During gameplay, the following stats are always displayed:

* **Current Score:** Reflects the points earned by destroying enemies.
* **High Score:** Tracks the highest score ever achieved.
* **Lives:** Shows the number of remaining lives. Each time the spaceship is hit by an enemy bullet, a life is lost.

**Game Over**

The game ends when all lives are lost. After the game over, a **Game Over Screen** appears, displaying:

* **Player’s Score:** The total score achieved in the session.
* **Instructions for Next Actions:**
  + **Press ‘Q’** to quit the game.
  + **Press ‘R’** to restart the game from the basic level.
  + **Press ‘M’** to return to the Main Menu.
  + **Press ‘L’** to progress to the next level if eligible.

**Level Progression**

Players can opt for a level upgrade by pressing **‘L’** from the Game Over Screen. However, this option is available only if they meet a **pre-defined condition** (e.g., achieving a specific score or destroying a minimum number of enemies).

Upon progressing to the next level, the game introduces:

* **Increased Difficulty:**
  + Higher numbers of enemies.
  + Faster enemy bullets.
  + More frequent enemy fire.
* **Shields:** Players can use shields to absorb a limited amount of enemy damage.

The gameplay loop continues with the increased challenge, allowing players to push their skills and achieve higher scores.

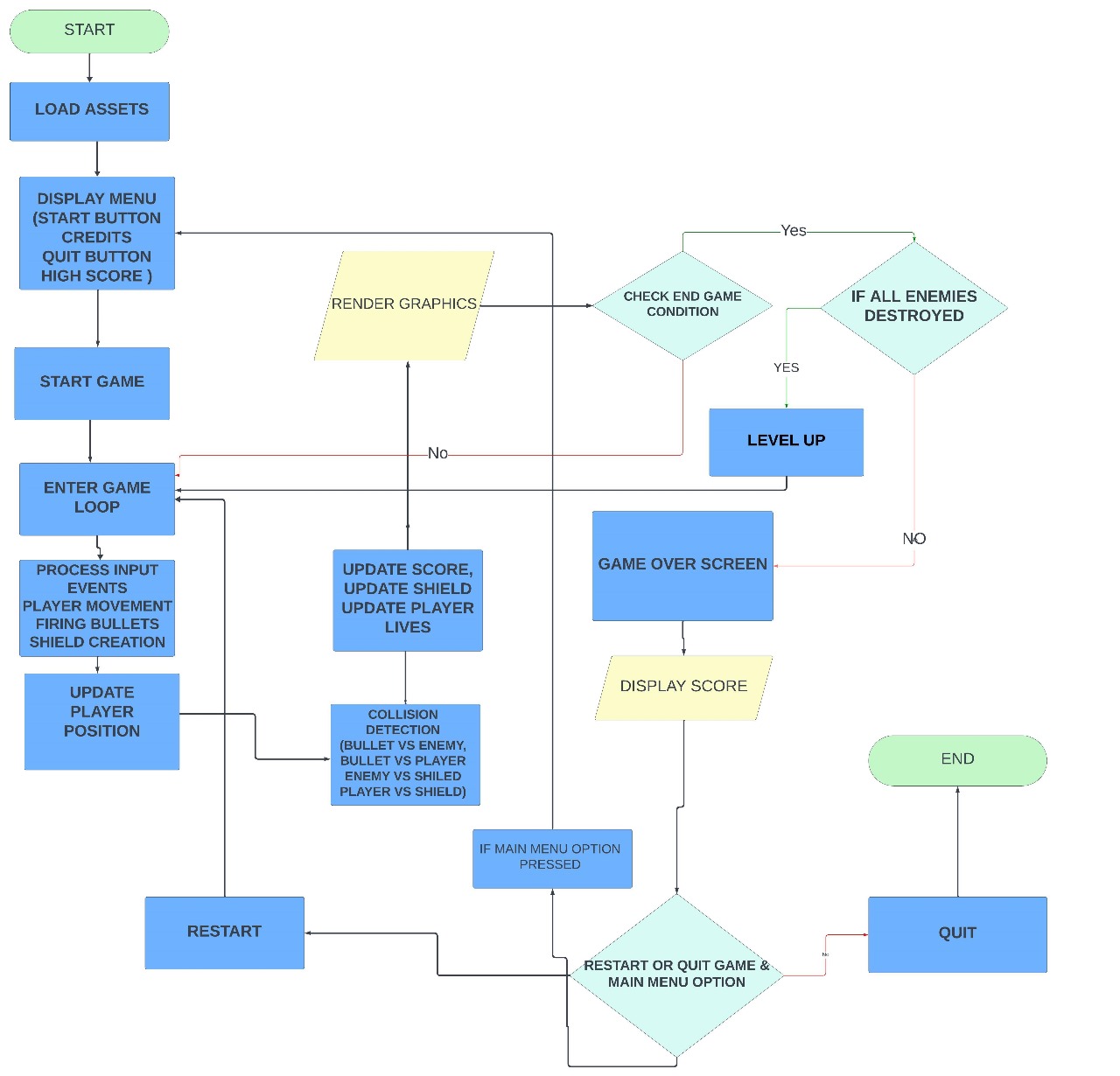
**Winning Strategy**

* **Movement Precision:** Master the spaceship’s controls to dodge incoming bullets effectively.
* **Timing and Accuracy:** Fire bullets strategically to maximize enemy hits and conserve time.
* **Shield Management:** Use shields wisely to absorb damage in critical situations.
* **Level Selection:** Only progress to higher levels if you’re confident in your skills to handle increased difficulty.

#### Diagram of a diagram of a bullet management system Description automatically generated****Architecture Diagram:****

Below is the abstract project architecture diagram:

**3.Detailed Flowchart:**

****

**Flowchart Steps with Technical Details:**

**1. Game Initialization:**

* **Purpose**: Initialize all required game elements and prepare the game loop.
* **Details**:
  + Set up the window using sf::RenderWindow with desired width and height.
  + Load resources (textures, fonts, sounds) using sf::Texture, sf::Font, and sf::SoundBuffer.
  + Initialize entities: player spaceship, enemies, bullets, boundaries, score, and lives.

**2. Input Handling:**

* **Purpose**: Process user input to control the player spaceship and shooting.
* **Details**:
  + Use sf::Keyboard::isKeyPressed() to detect key presses:
    - Arrow keys (Up, Down, Left, Right) for movement.
    - Spacebar (Space) to fire bullets.
  + Store bullets fired by the player in a **vector** (e.g., std::vector<sf::CircleShape>).

**3. Update Game Entities:**

* **Purpose**: Update the position and states of all game elements.
* **Details**:
  + **Player**: Move based on input and ensure it stays within the window boundary.
    - Use an sf::FloatRect for boundaries to restrict movement.
  + **Enemies**: Move randomly or in a predefined pattern. If near the player, fire bullets straight down.
    - Store bullets fired by enemies in a separate **vector**.
  + **Bullets**:
    - Update positions (player bullets move upward, enemy bullets downward).
    - Remove bullets if they exit the screen using erase-remove idiom.
    - Data Structures: **Vectors** are used to store bullets for both the player and enemies.
  + **Collision Detection**:
    - Check if:
      * Player bullets collide with enemies.
      * Enemy bullets collide with the player.
    - Use sf::FloatRect::intersects() for collision detection.
    - On collision:
      * Remove the collided entity (e.g., enemy or bullet).
      * Update score or lives.

**4. Rendering:**

* **Purpose**: Draw all updated game entities onto the screen.
* **Details**:
  + Use sf::RenderWindow::clear() to clear the window at the beginning of each frame.
  + Draw each entity:
    - Player spaceship (sf::Sprite or sf::ConvexShape).
    - Enemies (sf::RectangleShape).
    - Bullets (sf::CircleShape).
    - Score and lives (sf::Text).
  + Use sf::RenderWindow::display() to present the updated frame.

**5. Collision Checking:**

* **Purpose**: Manage interactions between game elements.
* **Details**:
  + Player bullet vs. enemy: Increment score and remove enemy/bullet.
  + Enemy bullet vs. player: Decrease player lives and remove bullet.
  + Use algorithms optimized for collision detection:
    - Iterate through vectors of bullets and enemies.
    - Check intersections using bounding boxes (sf::FloatRect).

**6. Game State Updates:**

* **Purpose**: Manage game progression, score, and lives.
* **Details**:
  + Update **score**:
    - Increment by predefined points (e.g., 10 points per enemy).
  + Update High Score
* As the score progress
  + Update **lives**:
    - Decrease when the player gets hit.
    - If lives reach zero, set the **Game Over** flag.

**7.level up:**

* **Purpose:** to increase user experience and difficulty
* **Details:**
* When user clear the first level

**8.Game Over Handling:**

* **Purpose**: Stop the game and display the final score.
* **Details**:
  + Check if player lives reach 0.
  + Pause the game loop.
  + Display "Game Over" and final score using sf::Text.

**Algorithms and Data Structures:**

1. **Data Structures**:
   * **Vectors**:
     + Store bullets for both player and enemies (std::vector<sf::CircleShape>).
     + Maintain dynamic entities (e.g., active enemies or bullets).
   * **sf::FloatRect**:
     + Used for boundary enforcement and collision detection.
   * **sf::Text**:
     + Used to display the score, lives, and "Game Over" message.
2. **Collision Detection Algorithm**:
   * Iterate through each player bullet and check for collisions with all active enemies using bounding box checks (sf::FloatRect::intersects()).
   * Iterate through each enemy bullet and check for collisions with the player.
3. **Random Enemy Movement**:
   * Use std::rand() or std::uniform\_int\_distribution to generate random directions or speeds for enemies.
   * Ensure they stay within boundaries.
4. **Game Loop Logic**:
   * Standard SFML game loop:

while (window.isOpen()) {

handleInput(); // Process player input

updateEntities(); // Update positions, collisions, score

render(); // Draw everything on screen

}

#### ****4. Breakdown of Group Work (4 Members)****

**Task Distribution:**

1. **Input and Rendering Module & Audio (Hamna Shah):**
   * Develop the Input Module to handle keyboard inputs for player movement and shooting.
   * Create and render player and enemy sprites using SFML.
   * Add sound effects for shooting, explosions, and other events using SFML.
   * Implement background music for the game.
2. **Game Logic Module (Anas Norani & Hanan Majeed):**
   * Implement enemy movement patterns, level progression, and difficulty scaling.
   * Manage collision detection between shields, enemies, and the player.
   * Define win/loss conditions and ensure proper game flow.
3. **Data Module and High Score Management (Noor UL Saba):**
   * Create the system to track and save high scores.
   * Manage persistent data for settings and configurations.
   * Display UI components, including score and player lives.
   * Test and debug the overall system for performance optimization.