**SPACE SHOOTER GAME**

Session: 2025 – 2029

**Submitted by:**

Faizan Shakeel 2025S-CS-09

**Supervised by:**

Sir Laeeq

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**SHORT DESCRIPTION**

The Space Shooter Game is a console-based arcade-style game where players control a spaceship to defend against enemy attacks. Players can move in four directions, shoot projectiles, collect power-ups, and progress through increasingly difficult levels. The game features health management, scoring, and dynamic enemy behavior.

**Game Objects Description**

* **Player Bullets (^):**
  + Projectiles fired upwards by the player
  + Maximum of 10 bullets can be on screen at once
  + Destroy enemy ships on contact
* **Enemy Bullets (.):**
  + Projectiles fired downwards by enemies
  + Infinite number possible (though code limits to 100)
  + Reduce player health on contact
* **Fruits (F):**
  + **Power-ups** that appear after enemies fire 10 shots
  + Fall from the top of the screen
  + Award 50 points when collected
  + Restore 1 health point if health is not full

**Rules & Interactions**

1. **Movement Rules:**
   * Player can move in all four directions within maze boundaries
   * Cannot pass through walls (#)
2. **Combat Rules:**
   * Player fires with spacebar
   * Enemies automatically fire at increasing rates each level
   * Enemy bullets move downward, player bullets move upward
3. **Collision Rules:**
   * Player bullet + Enemy = Enemy destroyed (+10 points)
   * Enemy bullet + Player = Health reduced by 1
   * Player + Fruit = +50 points and possible health restore
4. **Level Progression:**
   * Level completes when all enemies are destroyed
   * Each new level increases enemy fire rate
   * Bonus points awarded for level completion (100 × level)

**Goal of the Game**

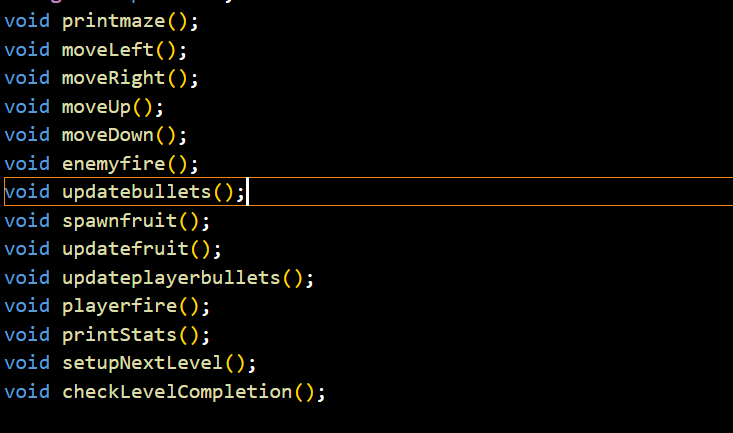
1. **Primary Objective:**
   * Survive through all 5 levels while destroying enemy ships
2. **Secondary Objectives:**
   * Maximize score by destroying enemies and collecting fruits
   * Maintain health by avoiding enemy fire and collecting fruits
3. **Winning Condition:**
   * Complete all 5 levels without losing all health
4. **Losing Condition:**
   * Player health reaches 0

**FUNCTIONAL REQUIREMENTS:**

* **Player Movement**
  + The player can move up, down, left, and right using arrow keys.
  + Movement is constrained within game boundaries.
* **Combat System**
  + Players can shoot bullets using the spacebar.
  + Enemies fire projectiles at increasing rates as levels progress.
* **Health Management**
  + The player starts with 3 health points.
  + Health decreases when hit by enemy bullets.
  + Health can be restored by collecting fruits (bonus items).
* **Scoring System**
  + Points are awarded for defeating enemies (+10) and collecting fruits (+50).
  + Bonus points (+100 × level) are given upon completing a level.
* **Level Progression**
  + The game has 5 levels with increasing difficulty.
  + Enemy fire rate increases with each level.
* **Power-ups**
  + Fruits appear periodically and restore health when collected.
* **Game States**
  + Tracks win/loss conditions (health depletion or level completion).
  + Displays "GAME OVER" or victory messages with final scores.
* **Real-time Updates**
  + Continuously refreshes the game screen to show player/enemy positions and stats.

**WIREFRAMES**

**FUNCTION PROTOTYPES:**

****

**FUNCTION DATA STRUCTURES:**

**A screen shot of a computer screen

AI-generated content may be incorrect.**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

DATA STRUCTURES

**GAME MAZE:**

**A screen shot of a black background

AI-generated content may be incorrect.**

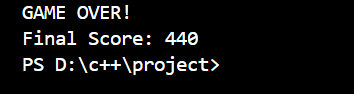
GAME MAZE

**SCORING MENU:**

**A black background with white text

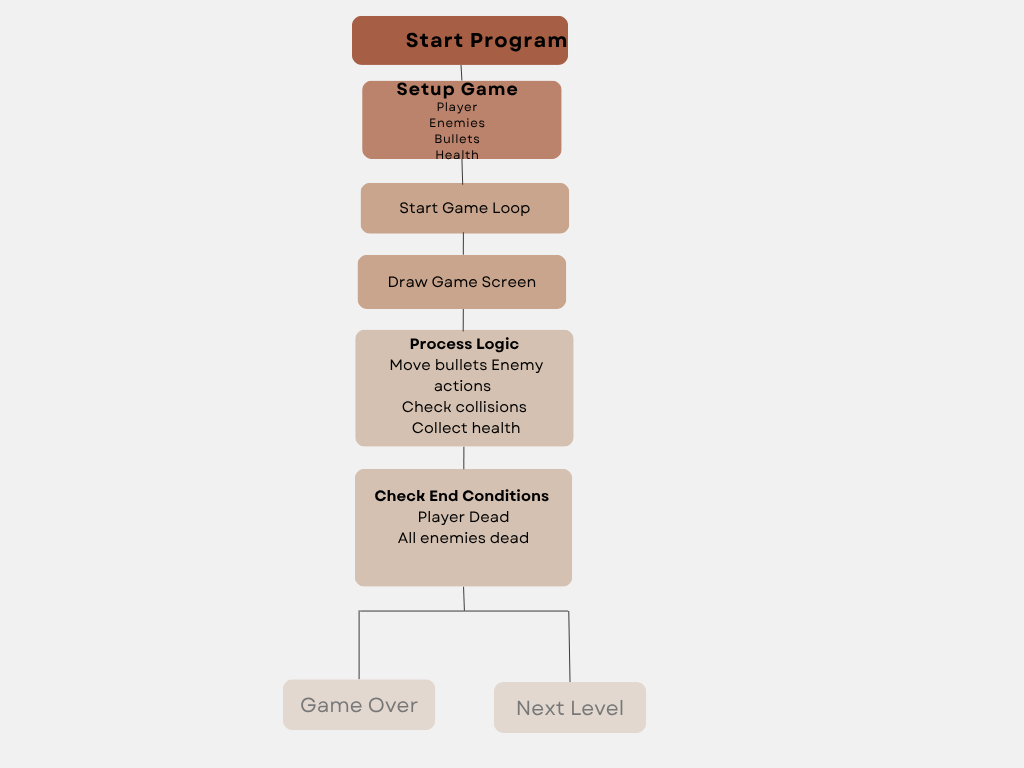
AI-generated content may be incorrect.**   
SCORE, HEALTH MENU

**END OF THE GAME:**

****

AFTER LOSING THE GAME

**FLOWCHART**



**GAME CODE**

#include <iostream>

#include <windows.h>

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD cursorPosition;

using namespace std;

void printmaze();

void moveLeft();

void moveRight();

void moveUp();

void moveDown();

void enemyfire();

void updatebullets();

void spawnfruit();

void updatefruit();

void updateplayerbullets();

void playerfire();

void printStats();

void setupNextLevel();

void checkLevelCompletion();

const int rows = 20;

const int column = 20;

const int maxBullets = 100;

const int maxPlayerBullets = 10;

char maze[rows][column] = {

    {'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'},

    {'#', 'V', 'V', 'V', 'V', 'V', 'V', 'V', 'V', 'V', 'V', 'V', 'V', 'V', 'V', 'V', 'V', 'V', 'V', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'},

    {'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'}};

int planeX = 18, planeY = 10;

int bulletX[maxBullets], bulletY[maxBullets];

bool bulletactive[maxBullets];

int bulletcount = 0;

int framecounter = 0;

int lastfiredenemy = 0;

int enemyShotsFired = 0;

int health = 3;

int score = 0;

int playerBulletX[maxPlayerBullets], playerBulletY[maxPlayerBullets];

bool playerBulletActive[maxPlayerBullets];

int playerBulletCount = 0;

bool fruitActive = false;

int fruitX, fruitY;

const int fruitSpawnThreshold = 10;

int level = 1;

int enemiesRemaining = 38;

int currentFireDelay = 20;

bool levelComplete = false;

int main() {

    hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

    CONSOLE\_CURSOR\_INFO cursorInfo;

    GetConsoleCursorInfo(hConsole, &cursorInfo);

    cursorInfo.bVisible = false;

    SetConsoleCursorInfo(hConsole, &cursorInfo);

    while (health > 0) {

        cursorPosition.X = 0;

        cursorPosition.Y = 0;

        SetConsoleCursorPosition(hConsole, cursorPosition);

        maze[planeX][planeY] = 'A';

        if (!levelComplete) {

            enemyfire();

            updatebullets();

            checkLevelCompletion();

        } else {

            if (level < 5) {

                setupNextLevel();

                levelComplete = false;

            } else {

                system("cls");

        cursorPosition.X = 0;

        cursorPosition.Y = 0;

        SetConsoleCursorPosition(hConsole, cursorPosition);

                cout << "CONGRATULATIONS! YOU WON!\n";

                cout << "Final Score: " << score << endl;

                return 0;

            }

        }

        playerfire();

        updateplayerbullets();

        if (enemyShotsFired >= fruitSpawnThreshold && !fruitActive) {

            spawnfruit();

            enemyShotsFired = 0;

        }

        updatefruit();

        printmaze();

        printStats();

        if (GetAsyncKeyState(VK\_LEFT)) {

            moveLeft();

        }

        else if (GetAsyncKeyState(VK\_RIGHT)) {

            moveRight();

        }

        else if (GetAsyncKeyState(VK\_UP)) {

            moveUp();

        }

        else if (GetAsyncKeyState(VK\_DOWN)) {

            moveDown();

        }

        framecounter++;

        Sleep(50);

    }

    system("cls");

        cursorPosition.X = 0;

        cursorPosition.Y = 0;

        SetConsoleCursorPosition(hConsole, cursorPosition);

    cout << "GAME OVER!\n";

    cout << "Final Score: " << score << endl;

    return 0;

}

void enemyfire() {

    if (framecounter % currentFireDelay == 0) {

        int enemiesToFire = 2 + level;

        if (level > 2) {

            for (int i = 0; i < enemiesToFire && bulletcount < maxBullets; i++) {

                int col = 1 + (framecounter + i) % (column - 2);

                if (maze[1][col] == 'V') {

                    bulletX[bulletcount] = 2;

                    bulletY[bulletcount] = col;

                    bulletactive[bulletcount] = true;

                    bulletcount++;

                    enemyShotsFired++;

                }

            }

        } else {

            int start = (lastfiredenemy + 1) % (column - 2);

            if (start == 0) start = 1;

            for (int j = start, fired = 0; j < column - 1 && fired < enemiesToFire; j++) {

                if (maze[1][j] == 'V' && bulletcount < maxBullets) {

                    bulletX[bulletcount] = 2;

                    bulletY[bulletcount] = j;

                    bulletactive[bulletcount] = true;

                    bulletcount++;

                    lastfiredenemy = j;

                    fired++;

                    enemyShotsFired++;

                }

            }

        }

    }

}

void printmaze() {

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < column; j++) {

            cout << maze[i][j];

        }

        cout << endl;

    }

}

void moveLeft() {

    if (planeY > 1 && maze[planeX][planeY - 1] != '#') {

        maze[planeX][planeY] = ' ';

        planeY--;

        maze[planeX][planeY] = 'A';

    }

}

void moveRight() {

    if (planeY < column - 2 && maze[planeX][planeY + 1] != '#') {

        maze[planeX][planeY] = ' ';

        planeY++;

        maze[planeX][planeY] = 'A';

    }

}

void moveUp() {

    if (planeX > 1 && maze[planeX - 1][planeY] != '#') {

        maze[planeX][planeY] = ' ';

        planeX--;

        maze[planeX][planeY] = 'A';

    }

}

void moveDown() {

    if (planeX < rows - 2 && maze[planeX + 1][planeY] != '#') {

        maze[planeX][planeY] = ' ';

        planeX++;

        maze[planeX][planeY] = 'A';

    }

}

void updatebullets() {

    for (int i = 0; i < bulletcount; i++) {

        if (bulletactive[i]) {

            if (maze[bulletX[i]][bulletY[i]] == '.') {

                maze[bulletX[i]][bulletY[i]] = ' ';

            }

            bulletX[i]++;

            if (bulletX[i] < rows - 1) {

                if (bulletX[i] == planeX && bulletY[i] == planeY) {

                    health--;

                    bulletactive[i] = false;

                }

                else {

                    maze[bulletX[i]][bulletY[i]] = '.';

                }

            }

            else {

                bulletactive[i] = false;

            }

        }

    }

}

void printStats() {

    cout << "Health: " << health << " | Score: " << score << " | Level: " << level << endl;

    cout << "Enemies Remaining: " << enemiesRemaining << endl;

}

void playerfire() {

    if (GetAsyncKeyState(VK\_SPACE) && playerBulletCount < maxPlayerBullets) {

        for (int i = 0; i < maxPlayerBullets; i++) {

            if (!playerBulletActive[i]) {

                playerBulletX[i] = planeX - 1;

                playerBulletY[i] = planeY;

                playerBulletActive[i] = true;

                playerBulletCount++;

                break;

            }

        }

    }

}

void updateplayerbullets() {

    for (int i = 0; i < maxPlayerBullets; i++) {

        if (playerBulletActive[i]) {

            if (maze[playerBulletX[i]][playerBulletY[i]] == '^') {

                maze[playerBulletX[i]][playerBulletY[i]] = ' ';

            }

            playerBulletX[i]--;

            if (playerBulletX[i] > 0) {

                if (maze[playerBulletX[i]][playerBulletY[i]] == 'V') {

                    maze[playerBulletX[i]][playerBulletY[i]] = ' ';

                    playerBulletActive[i] = false;

                    playerBulletCount--;

                    score += 10;

                    enemiesRemaining--;

                }

                else {

                    maze[playerBulletX[i]][playerBulletY[i]] = '^';

                }

            }

            else {

                playerBulletActive[i] = false;

                playerBulletCount--;

            }

        }

    }

}

void spawnfruit() {

    fruitY = (framecounter % (column - 2)) + 1;

    fruitX = 2;

    while (maze[fruitX][fruitY] == 'V' && fruitY < column - 2) {

        fruitY++;

    }

    fruitActive = true;

    maze[fruitX][fruitY] = 'F';

}

void updatefruit() {

    if (fruitActive) {

        if (planeX == fruitX && planeY == fruitY) {

            score += 50;

            if (health < 3) {

    health++;

}

            fruitActive = false;

            maze[fruitX][fruitY] = ' ';

        }

        else {

            maze[fruitX][fruitY] = ' ';

            fruitX++;

            if (fruitX < rows - 1) {

                maze[fruitX][fruitY] = 'F';

            }

            else {

                fruitActive = false;

            }

        }

    }

}

void checkLevelCompletion() {

    enemiesRemaining = 0;

    for (int j = 1; j < column - 1; j++) {

        if (maze[1][j] == 'V') enemiesRemaining++;

    }

    if (enemiesRemaining == 0) {

        levelComplete = true;

        score += 100 \* level;

    }

}

void setupNextLevel() {

    level++;

    currentFireDelay = 20 - (level \* 2);

    if (currentFireDelay < 5) currentFireDelay = 5;

    for (int j = 1; j < column - 1; j++) {

        maze[1][j] = 'V';

    }

    enemiesRemaining = column - 2;

    for (int i = 0; i < maxBullets; i++) bulletactive[i] = false;

    for (int i = 0; i < maxPlayerBullets; i++) playerBulletActive[i] = false;

    bulletcount = 0;

    playerBulletCount = 0;

    fruitActive = false;

    enemyShotsFired = 0;

    planeX = 18;

    planeY = 10;

}

**FUTURE DEVELOPMENT**

* **Enhanced Visuals**
  + Implement graphical elements using libraries like SFML or Unity for a more engaging experience.
* **Sound Effects**
  + Add background music and sound effects for shooting, collisions, and power-ups.
* **Multiplayer Mode**
  + Allow cooperative or competitive gameplay with multiple players.
* **High Score System**
  + Save and display top scores persistently.
* **Advanced Enemies**
  + Introduce enemy types with unique behaviors (e.g., zigzag movement, shields).

**WEAKNESSES**

1. **Limited Visual Appeal**
   * The console-based interface lacks graphical polish.
2. **No Persistent Data**
   * Scores and progress are not saved after the game ends.
3. **Basic Enemy** 
   * Enemies follow simple movement and firing patterns.

### **CONCLUSION:**

The Space Shooter Game is a functional and entertaining console application that demonstrates core game development principles, including real-time input handling, collision detection, and state management. With future enhancements, it could evolve into a more immersive experience.

**RUBRICS:**

|  |  |  |
| --- | --- | --- |
|  | **A-Extensive Evidence** | **DONE OR NOT DONE** |
| Documentation Formatting **Grade:** | All the documentation meets all the criteria. | Done |
| Documentation Contents  **Grade:** | Documentation includes all of the criteria. | Done |
| Code Style  **Grade:** | All Code style criteria is followed | Done |
| Data Structure (Arrays)  **Grade:** | Data structure is sufficient for the project requirements | Done |
| Modularity  **Grade:** | Meet all Modularity criteria | Done |
|  |  |  |
| Recommendation Feature | Proper meaning full recommendation is present into system | Done |
| Presentation and Demo  **Grade:** | Presentation and Demo was 100% working | Done |
| Student Understanding with the Code.  **Grade:** | Student has complete understanding how the code is working and knows the concept. | Done |

|  |  |
| --- | --- |
| **Checked by:** | Click or tap here to enter text. |