Course Name: CSE115

Section: 02

Instructor's Name: Muhammad Shiffat-E-Rabbi

Group Members:

1. *Noshin Islam (2511819642)*

2. *MD Nafiz Rahman (2511238042)*

3.. *M.M. Rubaiet Ahmed (2512223042)*

4. *Mahin Ahmad (2413417042)*

Abstract

The objective of this project is to design and implement a console-based shooting game in C. This game simulates a spaceship shooting incoming aliens. The project emphasizes basic C programming concepts such as loops, conditionals, functions, arrays, and real-time user interaction using terminal rendering techniques. The report covers game logic, user guide, implementation, testing, and challenges encountered during development.

Introduction

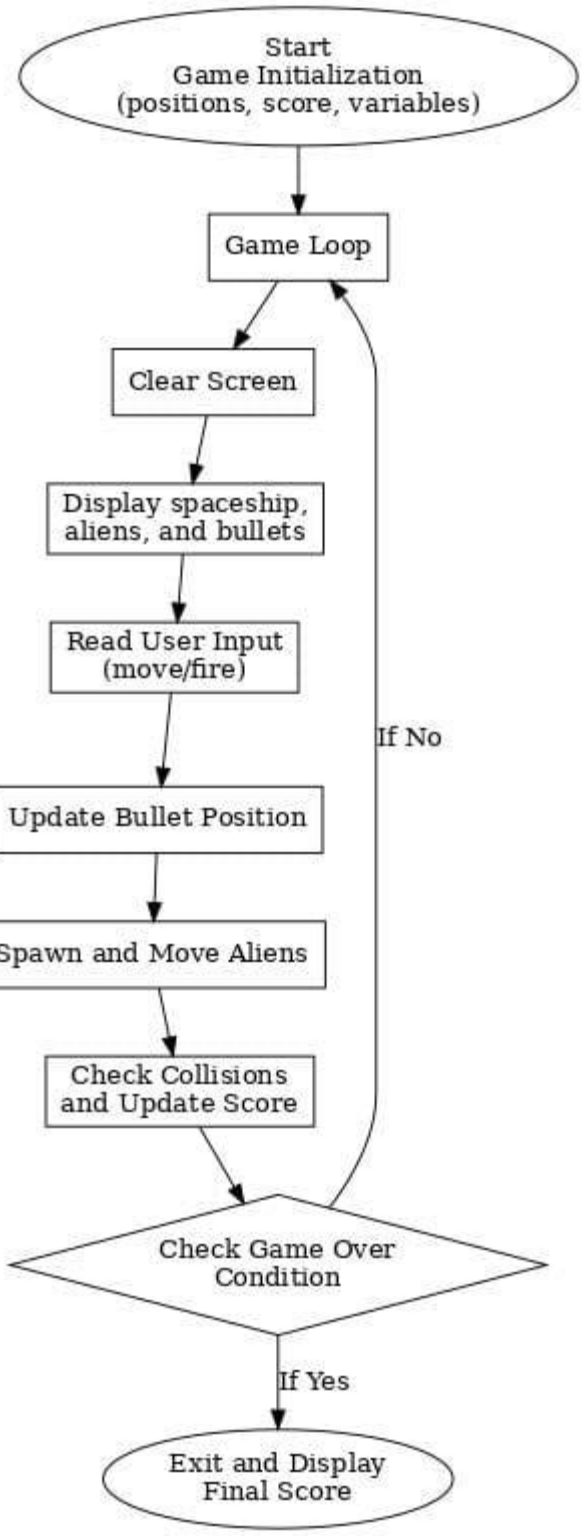
Classic arcade shooters like Space Invaders inspired the design of the Spaceship Invaders game. The player controls a spaceship represented by a character on screen, which can move in four directions and fire bullets to destroy descending aliens. The game is coded entirely in C and runs on Windows using the command line. It provides hands-on experience with structured programming, basic AI behavior, and real-time input handling.

Flowchart

The flow of the program is as follows:

1. Game Initialization (positions, score, variables)
2. Game Loop:
   a. Clear screen
   b. Display spaceship, aliens, and bullets
   c. Read user input (move/fire)
   d. Update bullet position
   e. Spawn and move aliens
   f. Check collisions and update score
   g. Check game-over condition

3. Exit and display final score


Start
Game Initialization
(positions, score, variables)
Game Loop
Clear Screen
Display spaceship,
aliens, and bullets
Read User Input
(move/fire)
Update Bullet Position
Spawn and Move Aliens
Check Collisions
and Update Score
Check Game Over
Condition
If No
If Yes
Exit and Display
Final Score

Application Usage and Guidelines

How to Compile and Run the Program

1. Ensure GCC compiler is installed.
2. Open terminal in the source code directory.
3. Compile using: gcc -o spaceship_invaders spaceship_invaders.c
4. Run the game: ./spaceship_invaders

Input and Output Formats

Input:

Keyboard keys (A, D, W, S, ENTER)

Output:

On-screen updates showing the spaceship ('A'), aliens ('V'), and bullets ('|'). Score is displayed below.
Game ends if an alien reaches the spaceship. Score is displayed as: Score: X

Features and Implementation

- Position Spaceship:
  - Code snippet:

```
void displaySpaceship() {
printf("\033[%d;%dH", spaceshipRow, spaceshipCol);
printf("A");
}
```

  - Explanation:
    Displays the spaceship ‘A’ by using ANSI scape code.

- Spaceship Movement:
  - Code snippet:

```
void moveSpaceship(char key) {
switch (key) {
case 'A': case 'a': // moves left
if (spaceshipCol > 1) spaceshipCol--;
break;
case 'D': case 'd': // moves right
if (spaceshipCol < SCREEN_WIDTH) spaceshipCol++;
break;
case 'W': case 'w': // moves up
if (spaceshipRow > 1) spaceshipRow--;
break;
case 'S': case 's': // moves down
if (spaceshipRow < SCREEN_HEIGHT) spaceshipRow++;
break;
}
}
```

  - Explanation:
    Uses switch and ‘if’ condition to make spaceship movement.
- Bullet Shooting:
  - Code snippet:

```
void shootBullet() {
if (bulletRow == -1) { // only one bullet can exist at a time
bulletRow = spaceshipRow - 1;
bulletCol = spaceshipCol;
}
}
```

- Spawn Alien:
  - Code snippet:

```
void spawnAliens() {
for (int i = 0; i < MAX_ALIENS; i++) {
if (aliens[i][0] == -1) { // if an alien is not active
aliens[i][0] = 1;              // spawn at the top row
aliens[i][1] = 1 + rand() % 30;  // random columnn
}
}
}
```

- Random Alien Movement:
  - Code snippet:

```
void moveAliens() {
for (int i = 0; i < MAX_ALIENS; i++) {
if (aliens[i][0] != -1) { // if an alien is active
aliens[i][0]++; // moves downward

if (aliens[i][0] > SCREEN_HEIGHT) {
aliens[i][0] = -1; // deactivates the alien
aliens[i][1] = -1;
}
```

```
            }
        }

        // Randomly spawn new aliens
        if (rand() % 10 < 3) { // 30% chance to
    spawn a new alien
            spawnAliens();
        }
    }
```

- Update Bullet:
  - Code snippet:

```
void updateBullet() {

if (bulletRow != -1) {

bulletRow--; // moves the bullet upward

if (bulletRow < 1) { // if the bullet goes off the
screen

bulletRow = -1;

bulletCol = -1;

}}}
```

  - Explanation:
    The 'bulletRow--' decrement moves the bullet upward and the if condition checks if the bullet goes off the screen.

- Check Collision:
  - Code snippet:

```
void checkCollisions() {

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

// checks if bullet and alien occupy the same
position

if (aliens[i][0] != -1 && bulletRow == aliens[i][0]
&& bulletCol == aliens[i][1]) {

// collision detected

aliens[i][0] = -1; // deactivates the alien

aliens[i][1] = -1;

bulletRow = -1;    // removes the bullet

bulletCol = -1;

score++;          // increments the score

}

}

}
```

  - Explanation:

    Uses if condition to detect if the alien and bullet come into contact, and doing so deactivates them both.

- Game Over:
  - Code snippet:

```
int isGameOver() {

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

if (aliens[i][0] == spaceshipRow &&
aliens[i][1] == spaceshipCol) {

return 1; // Game over

}

}

return 0;

}
```

  - Explanation:

    Uses a for loop and if condition to declare game over.

Testing and Results

Test Case 1: Alien hit by bullet -> Score increased.

Test Case 2: Alien collides with spaceship -> Game Over triggered.

Test Case 3: Moving spaceship with A/D keys -> Works smoothly.

Test Case 4: Rapid input -> Game responds correctly.

Test Case 5: Multiple aliens and frame delay -> Game remains playable.

Challenges and Solutions

1. Console Screen Manipulation

Challenge: Using system("cls") to clear the screen and printf("\033[%d;%dH") for positioning was hard.

Solution:

- On Windows, using an additional library windows.h and SetConsoleCursorPosition for more reliable control over cursor positioning.

2. Random Column for Aliens

Challenge: The column calculations 1 + rand() % 30 created aliens outside the screen width or overlap with other aliens.

Solution:

- Maintaining aliens[i][1] falls strictly within the bounds of 1 to SCREEN_WIDTH.
- Used logic to prevent aliens from spawning in the same column at once.

3. Screen Flickering

Challenge: The frequent use of clearScreen() (via system("cls")) can caused flickering, making the game visually uncomfortable.

Solution:

- Used a double-buffering approach. Updated the game state in memory and render it all at once.
- Alternatively, only update the parts of the screen that have changed instead of clearing and redrawing the entire screen.

4. Alien Movement and Spawning

Challenge: Aliens spawned too frequently or inconsistently, leading to chaotic gameplay.

Solution:

- Adjusted the probability of spawning (rand() % 10 < 3) to fine-tune alien appearance frequency.
- Introduced logic to limit the number of aliens on the screen at any given time.

Conclusion

This project provided hands-on experience with C programming and basic game development. It helped reinforce the use of functions, conditionals, arrays, and coordinate manipulation. The final product runs smoothly and mimics arcade behavior within a command-line environment. Potential future improvements include adding sound effects, multiple bullet support, levels, and GUI integration.