Space Invaders Clone

Course: CSE115. Section:02.

Group Members:

1. **Noshin Islam** (2511819642)

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

3. **Mahin Ahmad** (2413417042)

4. **MD Nafiz Rahman** (2511238042)

Abstract:

In this report, we describe the design and implementation of a console based shooter similar to the arcade game Space Invaders. Would I teach the same way there or in assembly or C, for that matter? The report elaborates on the game logic, user interface, implementation issues, testing and a few enhancement ideas for the future.

1. Introduction

From the Title, the Space Invaders Clone combines certain elements of the original game but the addition being that the Player can control a spaceship in a console and press console buttons to shoot on-screen Aliens. Prepare for some intergalactic fun, as we take you through this adventure game that helps build on programming skills in C [2] and encoding data in arrays.- Given that designing a proper game requires building a game engine, it may become a bit too complex to handle right away. By creating this project, you are learning how to develop games and game logics.

1.1 Objectives

- The main objectives of this project are as follows:
- • Create a functional and interactive game using C.
- • Implement the basic mechanics of games such as movement, shooting and collision detection.
- • To improve comprehension of structured programming and handling in real -time inputs.
- • You want to explore the use of libraries to handle the console and the user input.

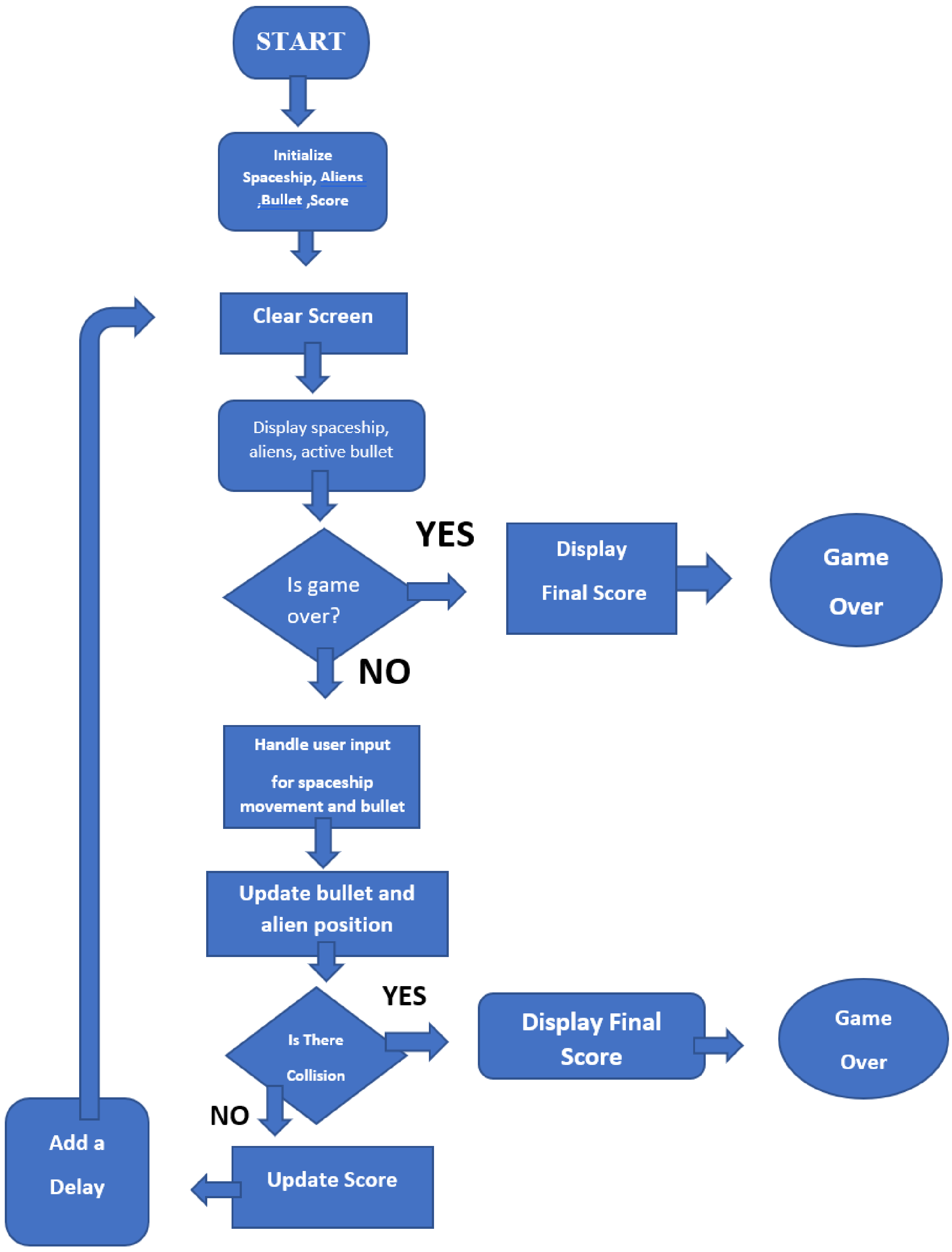
1.2 Scope

The scope of this project includes the development of a simple but engaging game that can be played in the console environment. The game will contain basic graphics using ASCII characters, sound effects and scoring system. Future iterations may include more complex features such as levels, turning on and improved graphics.

2. Game Design

2.1 Game Flow

The game works in a loop that continuously updates the state of the game based on the user's given input and the logic.Here’s the flow of the game:


START
Initialize Spaceship, Aliens ,Bullet ,Score
Clear Screen
Display spaceship, aliens, active bullet
Is game over?
YES
Display Final Score
Game Over
NO
Handle user input for spaceship movement and bullet
Update bullet and alien position
Is There Collision
YES
Display Final Score
Game Over
NO
Update Score
Add a Delay

1. **Game Initialization:** Set initial position , score , and variables.
2. **Game Loop:**
    - Clear the screen.
    - Display spaceship, aliens & bullets.
    - Read user input for the movement and shooting.
    - Update bullet position and aliens movement.
    - Check for collision and update the score.
    - Determine if the game is over or not.
3. Exit & Display the final score.

2.2 User Interface

The gaming interface is simple and uses the signs of ASCII to represent a spacecraft, aliens and bullets. The spacecraft is represented "A", aliens "V" and bullets "|" The score is displayed at the bottom of the screen. The user interface is designed to be intuitive, allowing players to understand the controls and destination quickly.

2.3 Controls

The controls for the game are :

- **Movement:**
    - 'A' or 'a': Move left
    - 'D' or 'd': Move right
    - 'W' or 'w': Move up
    - 'S' or 's': Move down
- **Shooting:**
    - 'Enter' or 'Return': Shoot bullet

3. Implementation

3.1 Code Structure

The game is implemented using several functions ,each for specific tasks:

- **displaySpaceship:** Renders the spaceship on the screen.

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

- **clearScreen:** Clears the console for a fresh display.

```
void clearScreen() {
    system("cls");
}
```

- **moveSpaceship:** Updates the spaceship's position based on user input.

```
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;
    }
}
```

- **shootBullet:** Allows the spaceship to shoot a bullet.

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

- **updateBullet:** Moves bullets upward on the screen.

```
void updateBullet() {
    if (bulletRow != -1) {
        bulletRow--; // moves the bullet upward
        if (bulletRow < 1) { // if the bullet goes off the screen
            bulletRow = -1;
            bulletCol = -1;
        }
    }
}
```

- **spawnAliens:** Generates new aliens at random positions.

```
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
        }
    }
}
```

- **moveAliens:** Updates the positions of active aliens.

```
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) { // if the alien goes off the screen
                aliens[i][0] = -1; // deactivates the alien
                aliens[i][1] = -1;
            }
        }
    }
```

- **displayAliens:** Renders active aliens on the screen.

```
void displayAliens() {
    for (int i = 0; i < MAX_ALIENS; i++) {
        if (aliens[i][0] != -1) { // if an alien is active
            printf("\033[%d;%dHV\n", aliens[i][0], aliens[i][1]);
        }
    }
}
```

- **checkCollisions:** Detects collisions between bullets and aliens.

```
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
        }
    }
}
```

- **isGameOver:** Checks if any alien has reached the spaceship.

```
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;
}
```

3.2 Code Snippet

Here is a snippet of the code that handles the movement of the spaceship:

```
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;
    }
}
```

3.3 Game Logic

Game logic is structured to handle different aspects of the game, including movement, shooting and collision detection. The main loop of the game continuously checks the user's input, updates the position of spacecraft and bullets, and manages the movements of extraterrestrials. The logic is designed to ensure that the game runs smoothly and responds to real -time players.

3.4 Random Alien Spawning

Foreigners are deployed on random positions on the top of the screen. The friction mechanism ensures that the new aliens appear at different intervals and contribute to the game of unpredictability. The following code of code illustrates the logic of aliens:

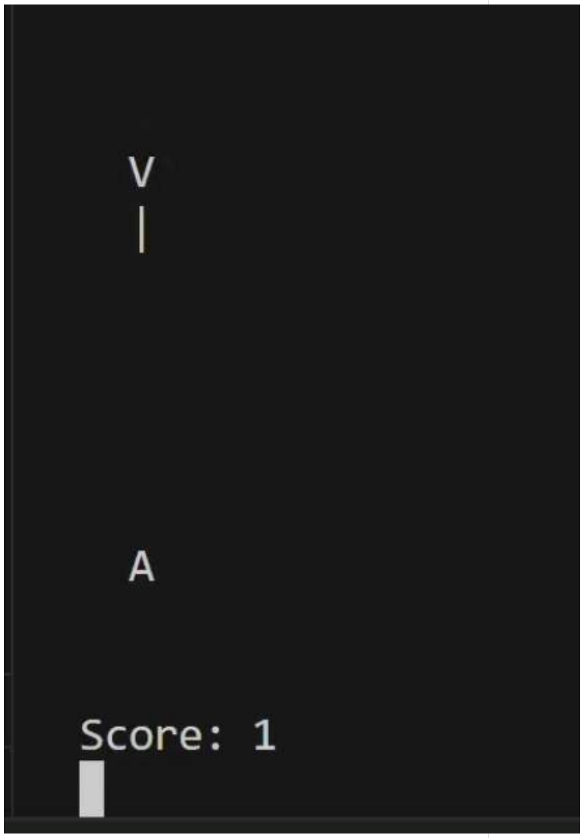
```
void spawnAliens() {
    for (int i = 0; i < MAX_ALIENS;
i++) {
        if (aliens[i][0] == -1) { //
if an alien is not active
            aliens[i][0] =
1;              // spawn at the top
row
            aliens[i][1] = 1 + rand()
% 30;  // random columnn
        }
    }
}
```

4. Testing and Results

4.1 Test Cases

- The game was subjected to various test cases to ensure functionality and performance. The following test cases have been made:

- • **Test case 1:** Verify that the alien with a bullet increases the score.



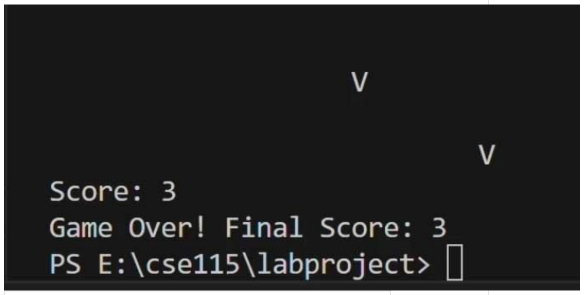
```
V
|



A

Score: 1
```

- **Test Case 2:** Ensure that an alien collides with the spaceship, the game triggers "Game Over" state.

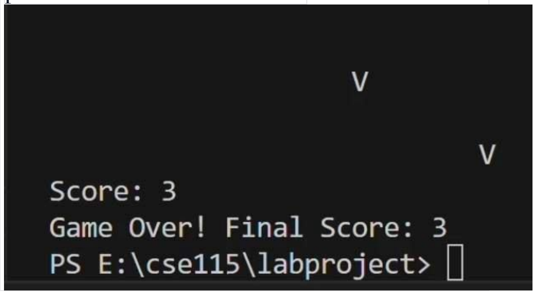


```
                 V
                        V
Score: 3
Game Over! Final Score: 3
PS E:\cse115\labproject>
```

• **Test case 3:** Test the spacecraft movement using A/D keys to confirm smooth navigation.

• **Test case 4:** Quick input commands to check the game sensitivity.

• **Test case 5:** Assess gameplay with more aliens on the screen to ensure that the performance remains stable.



```
                    V

                            V
Score: 3
Game Over! Final Score: 3
PS E:\cse115\labproject>
```

4.2 Performance evaluation

The game works well under different conditions and maintains a smooth frame frequency and responsive controls. Implementation of **Sleep delay (100)** ensures that the game takes place at a manageable speed, allowing players to respond appropriately to the coming alien. Metrics of power have been collected during testing, showing an average frame rate of 10 frames per second, which is reasonable for this type of game.

4.3 User Feedback

The user feedback was collected from a small group of testers. The following points were emphasized:

• Players enjoyed the simplicity and nostalgic feeling of the game.

• Proposals for improvement included the addition of sound effects and more diverse extraterrestrial types.

• Some players asked for the ability to pause the game and restore later.

5. Calls and solutions

5.1 Handling the console screen

Challenge: Deleting the screen and the location of the cursor was difficult using standard methods.

Solution: Windows.H library used for better control over cursor positioning. This allowed more accurate rendering of game elements without flashing.

5.2 Movement and friction of aliens

Challenge: The aliens emerged too often, leading to chaotic play.

Solution: Adjusts the likelihood of rubbing new aliens and reduced the number of active aliens on the screen. This balance ensures that the game remains demanding without being stunning.

5.3 Flashing of the screen

Challenge: Frequent screen cleaning caused flashing.

Solution: Implected Double Buffering Access to update the status of games in memory before rendering. This technique significantly reduced flashing and improved the overall visual experience.

5.4 Detection of collision

Challenge: The exact detection of the collision between bullets and aliens was complex.

Solution: He developed a systematic approach to checking whether the coordinates of the bullets corresponded to the coordinates of any active alien. This method ensured that the collisions were detected reliably and increase the game.

6. Future Work

6.1 Enhancements

The future iteration of the game could include several improvements:

• **Sound effects**: Adding sound effects for shooting, aliens and playing over the script to improve the game experience.

• **Multiple levels**: introducing different levels with increasing difficulties, including faster aliens and more complex patterns.

• **Power-UPS**: Power-UPS implementation that players can collect to gain temporary benefits such as fast fire or shields.

• **GUI Graphical interface (GUI):** Transition from console -based console to graphical use by libraries such as SDL or SFML, for a more modern look.

6.2 Community into operation

To improve the quality and sustainability of the game project, it is necessary to active involvement in the game community. It allows developers to acquire diverse perspectives, identify areas for improvement and support players' satisfaction. The key points include:

• Collection of feedback:

Determination of structured channels - for example feedback, forums or surveys - allows players to report problems, design features and share opinions.

• Development of cooperation:

Encouraging players to participate in the development process promotes a sense of ownership and contributes to better design decisions.

• Improved Game Quality:

Regular interaction with users can help reveal errors, gentle game mechanics and increase the overall user experience.

• Community platforms:

Creating and maintenance of premises on social media or dedicated websites allows communication and announcement in real time.

• Posts with open source code:

Permitting community contributions, especially in academic or non -commercial projects, can lead to innovative improvements and educational value.

• **Public opinion testing and surveys:**

Connecting the community in Beta Beta testing and collecting feedback through public opinion surveys can lead future updates and implementation of functions.

By accepting these practices, developers can ensure that the game develops through access to cooperation and focused on players, which eventually leads to a more refined and more successful product.

7. Conclusion

The development of Space Invaders Clone offered valuable practical experience with C programming and introduced concepts of basic games. It has strengthened key programming structures such as functions, conditional, fields and logic based on coordinates.

Overall, the project shows the practical use of structured programming in a creative context. Potential future enhancements include the integration of sound effects, allowing multiple shooting functions, adding game levels, and implementing a graphical user interface to improve the user interface.

Link

1. **Kernigh, B. W., & Ritchie, D. M. (1988**). Programming language C (2. Edition). Prentice Hall.

- Basic text for understanding syntax, structure and concepts of programming.

2. **Malik, D. S. (2012). C Programming**: from problems analysis to program design (7. Edition). CENGAGE LEARNING.

- offers a structured approach to programing in C, suitable for beginners to middle students.

3. **Geeksforgeeks. (n.d.). Development of the game using C/C ++**. Cited from https://www.geeksforgeeks.org/game-development-using-c-c/

- Online source explaining how to use C/C ++ to easily create games, including examples of logic and basic graphics.