CSE 115.4 – Group 8 – Radit Rifah Rahman(2521985042) , Seendeed Islam(2524611042), Afrida Hossain Khan(2522096642)

Space Invaders - CLI Game in C

CSE115 Programming Language I Project Report

1. Introduction

Background

Space Invaders is a classic arcade shooting game where the player controls a ship at the bottom of the screen, shooting at waves of descending alien invaders. Our project is a Command-Line Interface (CLI) implementation of this game using the C programming language. The game runs entirely in the console and uses ASCII characters to represent the player, bullets, and invaders.

Instead of flashy graphics, our game uses a simple grid of characters to represent the player, bullets, and enemies. The player controls a spaceship located at the bottom of the grid and attempts to shoot down all enemies before they descend far enough to collide with the player's position.

Objectives

The Space Invaders project was designed to meet the following objectives:

1. Create a playable text-based version of Space Invaders that runs entirely in a terminal environment without external graphics libraries.
2. Demonstrate modular C programming by separating game functions into logical parts for readability and maintainability.
3. Implement real-time gameplay mechanics with smooth player movement, bullet firing, and enemy movement patterns.
4. Include a scoring system to provide feedback and encourage competitive play.
5. Design a user-friendly menu allowing players to choose between starting the game, viewing controls, learning about the project, or exiting.
6. Practice code optimization to ensure the game runs smoothly even with frequent updates to the screen.

Scope and Limitations

Scope:

- The game features a 20-row by 40-column grid.
- The player can move left and right and shoot bullets upward.
- Enemies (invaders) move horizontally and drop down one row when they hit the screen's edge.
- The game ends in one of two scenarios:
  - Player destroys all invaders (win).
  - Invaders reach the player's row (loss).

Limitations:

- The game is entirely text-based, with no sound except a simple Beep() when an enemy is destroyed.
- The grid size is fixed.
- The game supports only single-player mode.
- Enemy AI is simple — all invaders move together without advanced pathfinding.

2. Background Review

Game Theory Background

From a game theory perspective, Space Invaders belongs to a category of deterministic reaction-based games. While randomness plays little to no role in enemy movement (in our version), the challenge lies in anticipating enemy behavior and timing actions accordingly. The game can be compared to a dynamic puzzle that changes every second. Players must balance offense (destroying enemies quickly) and defense (avoiding having enemies reach the player's row).

C Programming in Game Development

C has been used extensively in professional game development for its speed, low-level memory control, and portability. Many early PC and arcade games were written entirely in C or assembly language. While modern game engines often use higher-level languages, C still forms the foundation of performance-critical systems such as physics engines, rendering pipelines, and embedded systems in game consoles.

Our project uses C in a lightweight, procedural manner, allowing for easy understanding of game loops, event handling, and collision detection.

3. Structural Design of the Game

Overall Architecture

The Space Invaders game follows a loop-based architecture. Once the game starts, it repeatedly processes player input, updates the game state, renders the grid, and checks for win/loss conditions.

Module Structure

Although contained in a single .c file, the program is organized into distinct functions:

- initGrid(): Sets up the grid with empty spaces, places invaders at the top, and the player at the bottom.
- drawGrid(): Prints the grid to the screen with the current score.
- movePlayer():Handles left/right movement and shooting.
- updateBullets(): Moves bullets upward and checks for collisions.
- updateInvaders(): Moves invaders horizontally, and down if they reach the screen edge.
- invadersLeft(): Checks if any invaders remain alive.

Data Flow Design

1. Main Menu: User selects Play, Controls, About, or Exit.
2. Initialization: The grid is reset and the player/invaders are positioned.
3. Game Loop:
    - Read input (if any).
    - Update positions of player, bullets, and invaders.
    - Redraw grid.
    - Check game state.
4. End State: Display win/loss message and return to menu.

4.Implementation Details

Development Environment

- Language: C (C11 Standard)
- Compiler: GCC
- OS: Windows 10/11
- Libraries Used:
    - <stdio.h> for I/O.
    - <conio.h> for _kbhit() and _getch() functions.
    - <windows.h> for Sleep() and Beep().
    - <stdlib.h> and <time.h> for randomization and utility functions.

Programming Paradigms Applied

- Procedural Programming: All code organized into functions for specific tasks.
- Modular Thinking: Logical separation of gameplay elements.
- Event-Driven Input: Immediate reaction to player keystrokes.

Key Implementation Features

- Real-Time Input Handling: _kbhit() detects keystrokes without blocking game updates.
- Collision Detection: Simple comparisons check if bullets hit invaders.
- Dynamic Invader Speed: Controlled by a moveThreshold variable.
- Score System: Points added on enemy kill.
- Sound Feedback: Beep() on enemy hit.

5. Component Analysis

Main Module Analysis

The main() function is the central hub of the program. It handles the main menu loop, reading the player's choice, and launching the appropriate section of the game. The menu options are:

1. Play: Starts a new game by resetting the score, player position, invader positions, and variables such as movement direction.
2. Controls: Displays the available controls in a simple format: A to move left, D to move right, and Spacebar to shoot.
3. About: Shows the project credits, instructor name, and contributors.
4. Exit: Terminates the program.

The main module also ensures that once the game ends (win or loss), the user is returned to the menu rather than exiting automatically. This improves the user experience by allowing continuous play without restarting the application.

Grid Module Analysis

The grid module is represented by the grid[Rows][Colm] array. This array is used to store and visualize the entire game state in a character-based format:

- Player ('A') — Always positioned in the bottom row, moves horizontally.
- Invader ('M') — Starts at the top rows, moves horizontally and down.
- Bullet ('.') — Travels upward from the player's position.
- Empty (' ') — Represents an empty space.

The grid module ensures that any changes in the game state (e.g., bullet movement, invader movement) are reflected visually on the next redraw.

Player Control Module Analysis

The movePlayer() function is responsible for reading the user's key input and translating it into movement or shooting actions:

- If A is pressed and the player is not at the leftmost column, the player moves one step left.
- If D is pressed and the player is not at the rightmost column, the player moves one step right.
- If Spacebar is pressed, a bullet is created just above the player's position (if the space is empty).

The player's position is stored in the playerPos variable, and the bottom row of the grid is updated accordingly.

Bullet Module Analysis

The updateBullets() function iterates through the entire grid and updates any bullet positions:

- If a bullet is in the top row, it disappears (missed shot).
- If a bullet collides with an invader, both are removed and the player's score increases by 10 points. A short beep sound is played for feedback.
- If a bullet moves upward into an empty cell, it is moved one row higher.
- Bullets cannot pass through invaders — they are removed upon impact.

Invader Module Analysis

The updateInvaders() function is key to creating challenge in the game:

- Invaders move horizontally in the direction stored in invaderDirection.
- When any invader hits the left or right wall, all invaders reverse direction and move one row down.
- If any invader reaches the player's row, gameOver is set to 1 and the loop ends.
- The movement speed is controlled by moveThreshold, which acts as a delay mechanism so that invaders don't move every single cycle.

6. User Interface Design

Visual Design Philosophy

Since the game is text-based, we focus on clarity and immediate feedback:

- The score is always displayed at the top.
- The grid is redrawn every cycle to reflect the updated state.
- Distinct characters are chosen for player, invaders, and bullets to avoid confusion.

Menu System Design

- Numbered menu items for intuitive navigation.
- Each option leads directly to its function.
- Invalid inputs display an "Invalid" message and return to the menu.

7. Game Mechanics and Rules

Gameplay Flow

1. Player chooses "Play" from the menu.
2. Grid is initialized with invaders at the top and the player at the bottom.
3. Player moves and shoots using keys.
4. Bullets move upward and can destroy invaders.
5. Invaders move left/right and down over time.
6. Game ends in win or loss condition.

Input Format and Validation

- Inputs are single key presses (A, D, Spacebar).
- Out-of-bound moves are ignored.
- Only one bullet can occupy a position at a time.

Win Condition Detection

- Game checks if any cell contains an invader.
- If none found → Player wins.

8. Performance Analysis

Time Complexity

- Bullet update: O(Rows × Colm).
- Invader update: O(Rows × Colm).
- Total operations per loop are minimal for a fixed grid size.

Space Complexity

- Fixed O(Rows × Colm) memory usage.
- No additional memory required during gameplay.

Optimization Techniques

- moveThreshold prevents invaders from moving every cycle, reducing unnecessary redraws.
- Early exits when detecting wall collisions or win/loss conditions.

9. Functionality Testing

Input Validation Testing

- Pressing A at the left edge does nothing (tested).
- Pressing D at the right edge does nothing (tested).
- Pressing Spacebar fires bullet only if above cell is empty.

Game Logic Testing

- Bullets correctly destroy invaders.
- Invaders move in correct pattern and reverse direction at wall.
- Game ends instantly if invader reaches player row.

Stress Testing

- Game runs smoothly at default refresh rate on low-end hardware.
- Multiple bullets and invaders handled without lag.

10.Security Considerations

Input Sanitization

- Since only key presses are used, risk of injection attacks is negligible.

System Command Safety

- Only safe system commands like cls are used.

Memory Safety

- Stack-based allocation avoids manual memory management risks.

11.Future Improvements

- Add multiple difficulty levels.
- Include power-ups and different weapons.
- Implement pause/resume function.
- Add different enemy types with varied speeds.
- Display a leaderboard of high scores.

12. Conclusion

The Space Invaders CLI game meets its objectives of creating a functional, fun, and educational programming project. It provides a solid example of applying C programming concepts to an interactive application.

13. References

1. **Teach Yourself C- Herbert Schildt**
2. **Original Space Invaders mechanics.**
3. **OpenAI**
4. **GitHub Repository: https://github.com/limewolf6/Retro-spaceship-game**