CSE 115 | Section: 04 | Project Group No. 02

Team Members:

- Lead Developer: Soumik Halder (ID: 2531413042)
- Board & Display: Mostafia Al Jannati (ID: 2532532642)
- Player Input: Shreyosi Mohanta (ID: 2532176042)
- Game Logic: Md.Kaif Khan (ID: 2532389642)
- Documentation: Sarah Tabassum (ID: 2531479042)

Development of a Console-Based Tic-Tac-Toe Game Using Modular C Programming

Abstract

This project presents the design and implementation of a console-based Tic-Tac-Toe game, developed as a requirement for the CSE 115 course at North South University. The primary objective was to create a robust, two-player interactive game that demonstrates core concepts of the C programming language, including arrays, pointers, modular programming, and input validation. The system allows two users to play on a shared terminal, featuring a dynamic 3x3 grid, real-time input parsing, and automated win/draw detection logic. This report details the software development lifecycle, the modular architecture adopted, the algorithmic logic for game state evaluation, and the testing methodologies employed. The resulting software is a stable, user-friendly application that successfully meets all specified project requirements.

1. Introduction

1.1 Background

Tic-Tac-Toe, also known as Noughts and Crosses, is a classic paper-and-pencil game for two players who take turns marking the spaces in a 3×3 grid with X or O. Despite its simplicity, the game serves as an excellent case study for introductory computer science projects. It requires the implementation of state management, turn-based logic, and condition checking, which are foundational concepts in game development and software engineering.

In the context of the CSE 115 curriculum, developing this game in the C programming language challenges students to manage memory manually, handle standard input/output (I/O) streams, and structure code effectively without the aid of high-level object-oriented features found in languages like Java or Python.

1.2 Objective

The primary goal of this project was to develop a functioning software application that allows two human players to compete against each other. Key objectives included:

- **Modular Design:** Splitting the codebase into separate header and source files (board.c, game_logic.c, input.c) to promote code reusability and maintainability.
- **Robust Input Handling:** Implementing safeguards against invalid inputs (e.g., entering letters instead of numbers, selecting occupied cells).
- **Game State Management:** Accurately detecting win conditions (rows, columns, diagonals) and draw conditions (full board).

1.3 Scope

The current version of the project is a local multiplayer game executed in a command-line interface (CLI). It does not currently support network play or an AI opponent, though the modular design allows for these extensions in future updates.

2. Methodology and System Design

The development process followed a structured approach, beginning with requirement analysis and moving through architectural design, implementation, and testing.

2.1 Software Architecture

To ensure maintainability and collaborative efficiency, the team adopted a modular architecture. The monolithic approach (writing all code in main.c) was rejected in favor of separating concerns. The system is divided into four distinct modules:

1. **Main Module (main.c):** Serves as the entry point, orchestrating the game loop and managing player turns.
2. **Board Module (board.c, board.h):** Responsible for initializing the data structure and rendering the visual grid to the console.
3. **Input Module (input.c, input.h):** Handles user interactions, parses raw input, and sanitizes data before passing it to the game logic.
4. **Logic Module (game_logic.c, game_logic.h):** Contains the core algorithms for determining game outcomes (Win/Loss/Draw).

2.2 Data Structures

The game board is represented by a 2D character array:

```
char board[3][3];
```

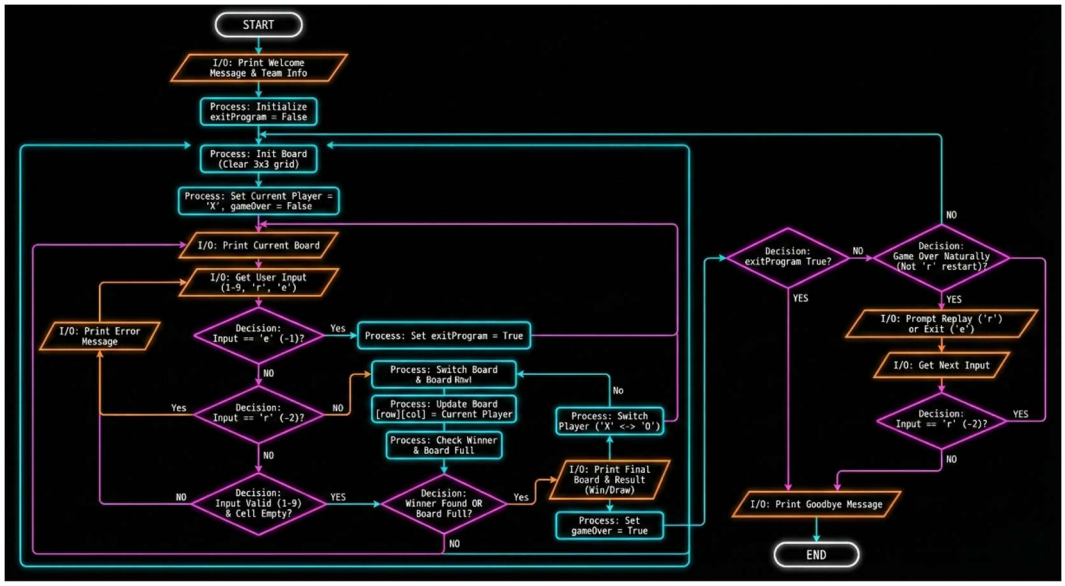
This data structure was chosen for its direct mapping to the visual representation of the grid. Each cell holds one of three states:

- ' ' (Space): Represents an empty cell.
- 'X': Represents Player 1's move.
- 'O': Represents Player 2's move.

2.3 Flowchart Description

The game operates on a continuous while loop controlled by boolean flags (gameOver, exitProgram).

1. **Initialization:** The board is cleared, and Player 'X' is set as the starting player.
2. **Render:** The current state of the board is printed.
3. **Input:** The system prompts the current player for a move (1-9).
4. **Validation:**
    - Is the input a number?
    - Is the number between 1-9?
    - Is the chosen cell empty?
5. **Update:** The board array is updated with the player's symbol.
6. **Check:** The system checks for a winner or a draw.
7. **Switch/End:** If no terminal state is reached, the turn switches. If the game ends, the user is prompted to restart or exit.



3. Implementation Details

This section delves into the specific coding techniques and algorithms used in the project components.

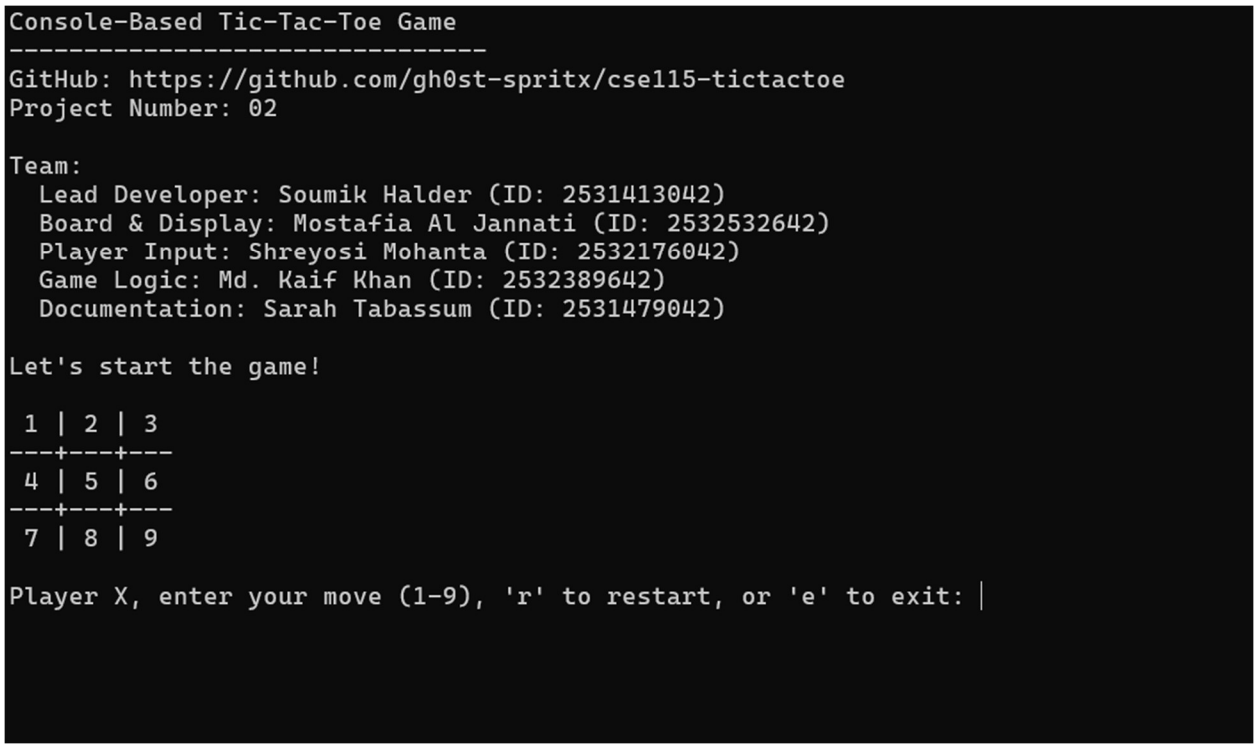
3.1 The Main Loop (main.c)

The main.c file acts as the controller. It introduces the game, displays team information, and manages the primary while loop. A significant feature here is the use of stdbool.h for semantic clarity (using true/false instead of 1/0).

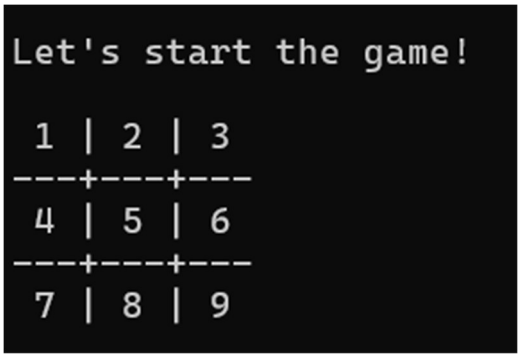
The loop structure handles the replayability factor. After a game concludes, the user can input r to restart immediately without restarting the program, or e to terminate.

// Snippet demonstrating the turn switch logic

currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';



3.2 Dynamic Board Rendering (board.c)



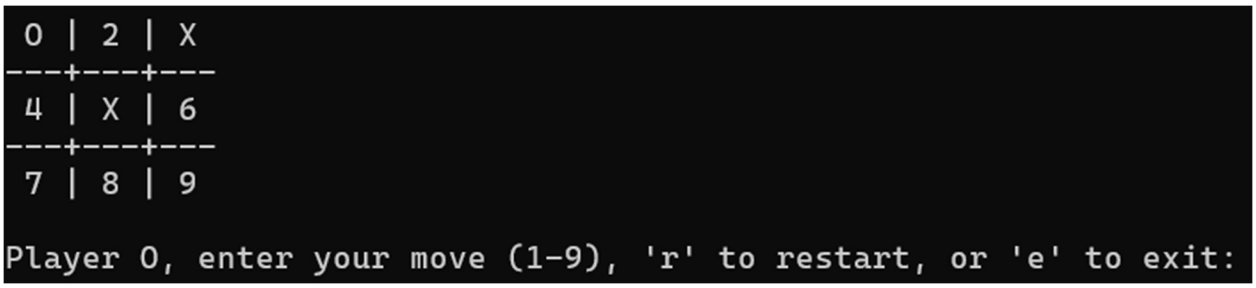
The printBoard function iterates through the 3x3 array. To enhance user experience, the system prints a visual grid using ASCII characters (|, ---). A key usability feature is the numeric mapping. If a cell is empty (' '), the renderer prints the corresponding number (1-9) calculated by i*3 + j + 1. This allows players to intuitively select cells using their numpad.

3.3 Input Sanitization (input.c)

One of the most challenging aspects of C programming is handling scanf securely. Our implementation reads inputs as strings first (char buf[10]) rather than integers. This prevents the infinite loop bug that occurs when scanf encounters a non-integer character while expecting %d.

The getUserInput function performs three checks:

1. **Command check:** Checks if the user entered 'e' (exit) or 'r' (restart).
2. **Conversion:** Uses atoi() to convert the string to an integer.
3. **Range check:** Ensures the integer is within the valid range [1, 9].



3.4 Win Detection Algorithm (game_logic.c)

The checkWinner function evaluates the board state after every move. Instead of iterating blindly, it checks specific vectors:

- **Rows:** Three iterations checking board[i][0] == board[i][1] == board[i][2].
- **Columns:** Three iterations checking board[0][j] == board[1][j] == board[2][j].

- **Diagonals:** Two hardcoded checks for [0][0]...[2][2] and [0][2]...[2][0].

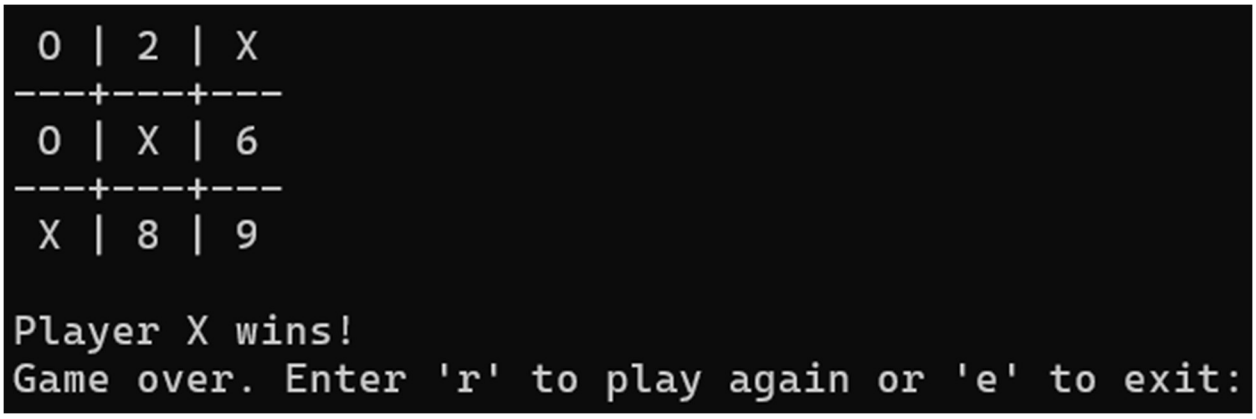
This function returns 1 if X wins, 2 if O wins, and 0 otherwise. This return value is seamlessly used by main.c to declare the victor.

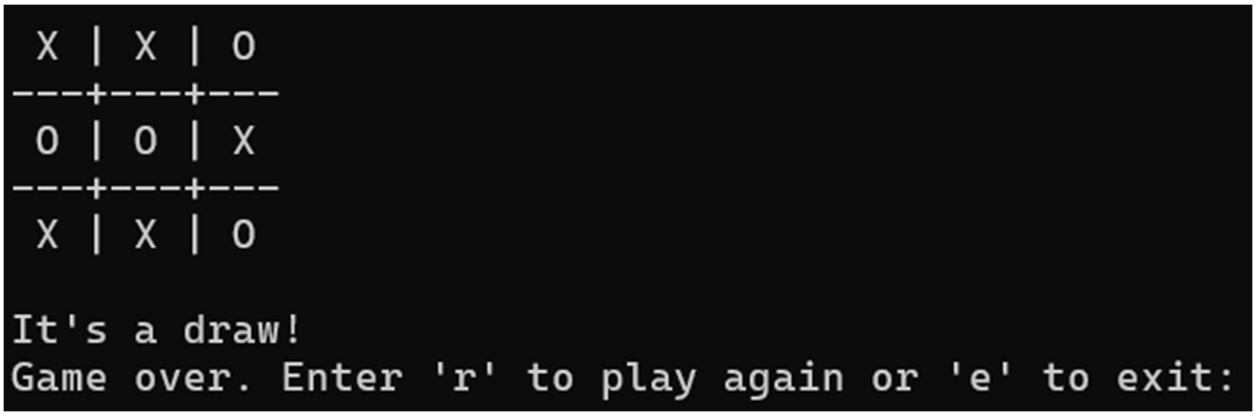
4. Testing and Results

The application underwent rigorous black-box testing to ensure stability.

4.1 Test Cases

Test Case ID	Input	Expected Output	Actual Output	Status
TC-01	Input 1 (Empty Board)	Cell 1 marked with 'X'	Cell 1 marked with 'X'	**Pass**
TC-02	Input 1 (Occupied Cell)	Error: "Cell already taken"	Error: "Cell already taken"	**Pass**
TC-03	Input a (Invalid Char)	Error: "Invalid input"	Error: "Invalid input"	**Pass**
TC-04	Input 10 (Out of Bounds)	Error: "Invalid input"	Error: "Invalid input"	**Pass**
TC-05	Player X completes Row 1	"Player X wins!"	"Player X wins!"	**Pass**
TC-06	Board Full, no winner	"It's a draw!"	"It's a draw!"	**Pass**
TC-07	Input r mid-game	Game restarts	Game restarts	**Pass**





4.2 Performance

The game consumes negligible system resources, operating in O(1) time complexity for win checking (since the board size is fixed at 3x3). Memory usage is minimal, involving only a few stack-allocated variables.

5. Discussion

5.1 Challenges Faced

During development, the team encountered issues with input buffering. Initially, pressing 'Enter' would sometimes leave a newline character in the buffer, causing subsequent scanf calls to behave erratically. This was resolved by switching to string-based input reading and manual parsing. Another challenge was ensuring the board array updated correctly across function calls. By passing the array to functions (which decays to a pointer), we ensured that modifications in board.c reflected in main.c without needing global variables.

5.2 Educational Outcomes

This project reinforced several critical CSE 115 learning outcomes:

- **Pointers and Arrays:** Understanding how 2D arrays are stored in memory and accessed.
- **Header Files:** Learning how to link multiple .c files using #include and header guards (#ifndef).
- **Team Collaboration:** Utilizing GitHub for version control allowed the team (Soumik, Mostafia, Shreyosi, Md.Kaif, Sarah) to work on different modules simultaneously without conflict.

6. Conclusion and Future Work

6.1 Conclusion

The Group 02 Tic-Tac-Toe project is a fully functional, console-based application that meets all academic requirements. It demonstrates a strong grasp of C programming fundamentals, modular software design, and user-centric development. The code is clean, well-documented, and free of common runtime errors.

6.2 Future Improvements

Future iterations of this project could include:

1. **AI Opponent:** Implementing a Minimax algorithm to allow single-player mode against the computer.
2. **GUI:** Porting the logic to a graphical interface using libraries like SDL or Raylib.
3. **Save System:** Using file I/O to save game states and leaderboards to a text file.

References

[1] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1988.

[2] North South University, "CSE 115: Programming Language I Course Outline," Department of Electrical and Computer Engineering, 2025.

[3] "Tic-Tac-Toe Game Logic," *GeeksforGeeks*, [Online]. Available: https://www.geeksforgeeks.org. [Accessed: Dec. 12, 2025].

[4] GitHub Repository, "cse115-tictactoe," *GitHub*: https://github.com/gh0st-spritx/cse115-tictactoe.