TicTacX - Classic Tic-Tac-Toe Game in C

CSE115 Programming Language I Project Report

Contents

1 Introduction **3**
1.1 Project Background . . . 3
1.2 Project Objectives . . . 3
1.3 Scope and Limitations . . . 3

2 Literature Review **3**
2.1 Game Theory Background . . . 3
2.2 Previous Implementations . . . 4
2.3 C Programming in Game Development . . . 4

3 System Design and Architecture **4**
3.1 Overall Architecture . . . 4
3.2 Module Structure . . . 4
3.3 Data Flow Design . . . 4

4 Implementation Details **4**
4.1 Development Environment . . . 4
4.2 Programming Paradigms Applied . . . 5
4.3 Key Implementation Features . . . 5

5 Module Analysis **5**
5.1 Main Module Analysis . . . 5
5.2 Board Module Analysis . . . 6
5.3 Game Logic Module Analysis . . . 6
5.4 Artificial Intelligence Module Analysis . . . 6

6 Technical Specifications **6**
6.1 System Requirements . . . 6
6.2 Compilation Process . . . 7
6.3 Memory Management . . . 7

7 User Interface Design **7**
7.1 Visual Design Philosophy . . . 7
7.2 Color Scheme Implementation . . . 7
7.3 Menu System Design . . . 7

8 Game Mechanics and Rules **7**

8.1 Gameplay Flow 7
8.2 Input Format and Validation 8
8.3 Win Condition Detection 8

9 Performance Analysis 8
9.1 Time Complexity Analysis 8
9.2 Space Complexity Analysis 8
9.3 Optimization Techniques 8

10 Testing and Validation 9
10.1 Input Validation Testing 9
10.2 Game Logic Testing 9
10.3 Cross-Platform Testing 9

11 Security Considerations 9
11.1 Input Sanitization 9
11.2 System Command Safety 9
11.3 Memory Safety 9

12 Future Enhancements 10
12.1 Advanced AI Implementation 10
12.2 Enhanced User Interface 10
12.3 Additional Game Modes 10
12.4 Technical Improvements 10

13 Conclusion 10
13.1 Project Success 10
13.2 Learning Outcomes 11
13.3 Technical Competency Demonstration 11
13.4 Educational Value 11

14 References 11

15 Appendices 12
15.1 Appendix A: Project Structure 12
15.2 Appendix B: Compilation Instructions 12
15.3 Appendix C: System Requirements Summary 12

INTRODUCTION

Project Background

Tic-Tac-Toe is a two-player paper-and-pencil game that consists of one or more moving players alternately marking the spaces in a 3 x 3 grid. A player loses when his/her opponent wins by matching three marks horizontally, vertically or diagonally and the game is won when one player can match three elements in one direction, be it vertical, horizontal or diagonal. This straightforward but entertaining game has been carried out in innumerable versions in different programming languages, and it makes an outstanding learning exercise of computer science students.

TicTacX is an advanced program of this ancient game, where the author works with the C programming language. A work produced in the shadow of the CSE115 Programming Language I unit, the project displays the understanding of the fundamentals of programming and the delivery of a simpler user experience through the use of modern terminal interface design principles.

Project Objectives

The primary objectives of the TicTacX project include:

Implementing a fully functional Tic-Tac-Toe game in C programming language

Demonstrating modular programming through multi-file project structure

Applying cross-platform development techniques for compatibility across operating systems

Incorporating user interface enhancements using ANSI escape sequences

Developing both human vs human and human vs AI gameplay modes

Ensuring robust error handling and input validation

Following software engineering best practices in code organization and documentation

Scope and Limitations

This is a project aimed at the development of a terminal-based Tic-Tac-Toe game with basic functionality encompassing duel gameplay, input validation, programmability of wins and multi- platform accessibility. Although the present implementation is a good start, there are opportunities to augment it in the future: by including more elaborated AI algorithms, multiplayer via a network, or even by adding an option to use graphical user interface.

LITERATURE REVIEW

Game Theory Background

Tic-Tac-Toe is a combinatorial game i.e. a game of strategy whose moves are alternated between players, whose play is often, but not always, mathematical. This being a solved game, it is known (assuming both players make the optimal moves) that the game will always be drawn. This mathematic feature is ideal in the application of AI and analysis of game theory.

Previous Implementations

There are many implementations of Tic-Tac-Toe in many programming languages, as simple as a console program to as complex as graphical user interfaces. They differ in the complexity of user interface and AI, however, the majority demonstrates identical basic game mechanics.

C Programming in Game Development

C programming Language has found extensive applications in the development of games and has been very efficient, low control and portable. C was also used on the creation of classic games such as Doom, Quake and many others which indicates the ability of C in dealing with complex game logic and real time processing needs.

SYSTEM DESIGN AND ARCHITECTURE

Overall Architecture

In the TicTacX project a modular design strategy is pursued where concerns are divided into separate modules that take care of distinct functionality. It is an architectural decision that could help increase maintainability, readability and even scalability of the code.

Module Structure

The project is organized into the following modules:

Main Module: Serves as the entry point and controller for the entire application, managing the main menu system and user interaction flow.

Board Module: Responsible for game board representation, visualization, and rendering using ANSI escape sequences for enhanced visual appeal.

Game Logic Module: Implements core game mechanics including move validation, win condition checking, turn management, and game state control.

Artificial Intelligence Module: Provides computer opponent functionality with random move generation for single-player mode.

Data Flow Design

Application goes through a data flow type of organization with user input going through validation layers before being applied to the game state. The visual representation is constantly updating, and changing depending upon the game state, giving us a responsive user experience.

IMPLEMENTATION DETAILS

Development Environment

The project was developed using standard C programming tools and follows the C11 standard. The development environment includes:

GCC compiler with Wall and Wextra flags for enhanced warning detection

Cross-platform development approach supporting Windows, Linux, and macOS

Standard C library functions for core functionality

Programming Paradigms Applied

The implementation utilizes procedural programming paradigms with strong emphasis on:

Modular design through function decomposition

Data encapsulation using header files and include guards

Preprocessor directives for cross-platform compatibility

Memory-efficient stack-based allocation

Key Implementation Features

Cross-Platform Compatibility: The project also applies sophisticated cross-platform that is evangelized by preprocessor directions that identify the operating system during compilation. Windows, Linux, and macOS use different code paths to make sure that UTF-8 encoding works and so execute the console commands.

Unicode Support: Enhanced Unicode support gives it the capacity correctly to represent special characters and ANSI escape sequences between operating systems. Console encoding is configured automatically on windows systems via use of command `chcp 65001`.

ANSI Escape Sequences: the visual design takes advantage of ANSI escape-codes to output it in color, making an enjoyable user interface with red X symbols, blue O symbols and well-aligned game-board borders.

Input Validation: Input validation results in robust input, which eliminates the common user error, as well as game integrity. Invalid numeric input, out of range coordinates, duplicate move and non-numeric entry are all handled by the system in such a way that it does not affect the flow of the game.

MODULE ANALYSIS

Main Module Analysis

The primary module is a central dashboard of the application since it uses a sophisticated menu infrastructure with five options. In use, it makes use of advanced input processing algorithms such as flushing buffers and protecting the streams against corruption of input as well as error recovery.

Menu System Design: The dynamic menu system enables the simplicity of navigation of simple options descriptions. Each menu item triggers the right functionality and solidifying common user behaviour throughout the application.

Cross-Platform Console Management: Platform-specific console management ensures platform consistent behavior no matter which platform is being run on. Automatic detection of the operating system in order to execute the appropriate command (`cls` on Windows, `clear` on systems based on Unix) is a feature common to clear screens.

Board Module Analysis

The board module aims at visualization and improvement of the user interface. The application produces a professional appearance of a game board because the borders and separators are handled by Unicode box-drawing characters.

Visual Design Elements: The visual design they employ uses the color theory to contrast various symbols of the players. The red color is applied to the letter X thus depicting action and aggression whilst the blue color is tacked onto the letter O thus signifying peace and tactics.

ANSI Escape Sequence Implementation: The module provides proper processing of ANSI escape sequences with reset codes in order to prevent intermixture of colors and terminal consistent state following each rendering task.

Game Logic Module Analysis

Game logic module is the intelligence region of the application, enhanced algorithms to recognize win condition besides the game having state control.

Win Condition Detection: The implementation applies optimized algorithm to check all combination of win such as three rows, three columns and two diagonals. It is also admission-check of O(1) time complex processing per move.

Game State Management: Games with more advanced game states track player turn, move count and game state. The system will automatically interchange between players and draw when there occurs no winner on the board and all the positions are set.

Artificial Intelligence Module Analysis

The AI module provides computer opponent functionality with random move generation. While simple in approach, the implementation demonstrates proper random number generation techniques and move validation.

Random Move Generation: The AI uses standard library random number functions seeded with current time to ensure varied gameplay experiences. The implementation includes proper validation to prevent placement on occupied board positions.

User Experience Enhancement: The AI implementation includes artificial delays to simulate thinking time, creating a more realistic and engaging gaming experience. Visual feedback shows the AI's chosen move coordinates for transparency.

TECHNICAL SPECIFICATIONS

System Requirements

Hardware Requirements:

Standard PC or laptop with basic processing capabilities

Minimum 512MB RAM

Available storage space for executable (less than 1MB)

Software Requirements:

C compiler supporting C11 standard

Terminal or command prompt with ANSI escape sequence support

Operating system: Windows 10/11, Linux distributions, or macOS

Compilation Process

The project utilizes a sophisticated Makefile-based build system that automates the compilation process. The build system creates a separate build directory for object files and the final executable, maintaining clean project organization.

Build Targets:

Main executable target with dependencies on all object files

Individual object file targets with proper dependency tracking

Clean target for removing compiled objects

Directory creation target for build organization

Memory Management

The implementation uses efficient stack-based memory allocation with fixed-size data structures. The 3×3 game board requires minimal memory footprint, making the application lightweight and fast.

USER INTERFACE DESIGN

Visual Design Philosophy

The user interface design focuses on creating an engaging yet intuitive gaming experience through terminal-based interaction. The implementation balances visual appeal with functional clarity to ensure users can easily understand game state and available actions.

Color Scheme Implementation

The color scheme utilizes contrasting colors for different game elements:

Red for player 'X' symbols representing energy and action

Blue for player 'O' symbols representing strategy and calmness

Black background for enhanced contrast and visual focus

White for board borders and structural elements

Menu System Design

The menu system provides clear navigation options with descriptive labels. The implementation includes proper error handling for invalid menu choices and provides helpful feedback to guide users.

GAME MECHANICS AND RULES

Gameplay Flow

The game follows standard Tic-Tac-Toe rules with enhanced user experience features:

1. Players take turns placing their symbols on the 3×3 grid
2. Player 'X' always moves first

3. Win condition requires three consecutive symbols in any row, column, or diagonal
4. Game ends when a player wins or all positions are filled (draw)

Input Format and Validation

Players enter moves using "row column" format where both values range from 1 to 3. The system validates inputs to prevent:

Non-numeric entries

Values outside the valid range

Placement on already occupied positions

Incomplete move specifications

Win Condition Detection

The implementation checks for win conditions after each move using optimized algorithms that examine:

All three rows for consecutive symbols

All three columns for consecutive symbols

Both diagonals for consecutive symbols

PERFORMANCE ANALYSIS

Time Complexity Analysis

Win Detection: O(1) - Fixed number of comparisons regardless of game state
Move Validation: O(1) - Single position check for validity
AI Move Generation: O(1) average case with potential for multiple attempts
Overall Game Loop: O(n) where n represents the number of moves

Space Complexity Analysis

Memory Usage: O(1) - Constant memory usage with fixed 3×3 board
Stack Allocation: Efficient use of automatic storage duration
No Dynamic Memory: Eliminates memory leaks and allocation overhead

Optimization Techniques

The implementation employs several optimization strategies:

Minimal function call overhead through efficient modular design

Early termination in win condition checking

Buffer flushing for responsive input handling

Preprocessor optimization for platform-specific code paths

TESTING AND VALIDATION

Input Validation Testing

Comprehensive testing was conducted to ensure robust input handling:

Valid numeric inputs within range (1-3)

Invalid character inputs and special symbols

Out-of-range numeric values

Duplicate move attempts on occupied positions

Partial input entries and incomplete data

Game Logic Testing

Extensive testing of game mechanics confirmed proper functionality:

All eight possible win conditions (3 rows, 3 columns, 2 diagonals)

Draw game scenarios with complete board filling

Proper player turn switching and symbol assignment

AI move validation and position selection

Cross-Platform Testing

The application was tested across multiple platforms:

Windows 10/11 PowerShell environments

Various Linux distributions including Ubuntu and Fedora

macOS terminal compatibility verification

Different terminal emulators and console applications

SECURITY CONSIDERATIONS

Input Sanitization

The implementation includes robust input sanitization to prevent buffer overflow vulnerabilities and ensure application stability. All user inputs undergo validation before processing.

System Command Safety

System commands are carefully managed through controlled execution paths. The clear screen functionality uses predefined commands without user input injection possibilities.

Memory Safety

Stack-based allocation eliminates dynamic memory allocation risks. The fixed-size data structures prevent memory-related vulnerabilities common in C applications.

FUTURE ENHANCEMENTS

Advanced AI Implementation

Minimax Algorithm: Implementation of the minimax algorithm would create an unbeatable AI opponent, providing maximum challenge for experienced players.

Difficulty Levels: Multiple difficulty settings could cater to players of different skill levels, from beginner to expert.

Strategic Move Planning: Advanced AI could analyze board positions and implement opening strategies and endgame techniques.

Enhanced User Interface

Animated Transitions: Smooth animations for move placements and win celebrations would enhance visual appeal.

Score Tracking System: Persistent score tracking could maintain win/loss records across multiple gaming sessions.

Customizable Themes: User-selectable color schemes and visual themes would personalize the gaming experience.

Additional Game Modes

AI vs AI Mode: Demonstration mode where two AI opponents play against each other for educational purposes.

Tournament System: Bracket-based tournament functionality for multiple players or AI configurations.

Timed Gameplay: Time-limited moves to add pressure and strategic depth to gameplay.

Technical Improvements

Configuration Files: External configuration files could store user preferences and game settings.

Save/Load Functionality: Game state persistence would allow players to resume interrupted games.

Network Multiplayer: Online multiplayer capabilities would enable remote player competition.

CONCLUSION

Project Success

The TicTacX project successfully demonstrates the application of fundamental C programming concepts in creating a sophisticated terminal-based game. The implementation showcases professional software engineering practices through modular design, comprehensive documentation, and robust error handling.

Learning Outcomes

This project provided valuable experience in:

Multi-file project organization and dependency management

Cross-platform development techniques and compatibility solutions

User interface design using ANSI escape sequences

Input validation and error recovery mechanisms

Game logic implementation and state management

Technical Competency Demonstration

The implementation demonstrates proficiency in:

C programming language fundamentals and advanced features

Makefile-based build systems and compilation process management

Memory-efficient programming with stack-based allocation

Platform-specific code optimization and conditional compilation

Educational Value

As an educational project, TicTacX serves multiple purposes:

Practical application of theoretical programming concepts

Introduction to software engineering principles and best practices

Exposure to cross-platform development challenges

Foundation for more complex game development projects

The TicTacX project represents a successful completion of the CSE115 course requirements while providing a solid foundation for future enhancements and learning opportunities. The clean code structure, comprehensive feature set, and professional implementation approach make it an excellent example of academic software development.

REFERENCES

1. Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language* (2nd ed.). Prentice Hall.

2. ISO/IEC 9899:2011. *Information technology - Programming languages - C.*

3. Stallman, R. M. (2019). *Using the GNU Compiler Collection (GCC).* Free Software Foundation.

4. GNU Make Manual. *GNU Make: A Program for Directing Recompilation.*

5. ECMA-48. *Control Functions for Coded Character Sets.*

6. Microsoft Developer Network. *Console Virtual Terminal Sequences.*

7. POSIX.1-2017. *Portable Operating System Interface.*

APPENDICES

Appendix A: Project Structure

```
tictacx/
 main.c
 support/
    board.c
    board.h
    game.c
    game.h
    GameAI.c
    GameAI.h
 Makefile
 README.md
```

Appendix B: Compilation Instructions

Using Makefile:

```
make
./build/tictacx
```

Manual Compilation:

```
mkdir build
gcc main.c support/game.c support/board.c support/GameAI.c -o build/tictacx
./build/tictacx
```

Appendix C: System Requirements Summary

Language: C (C11 standard)

Compiler: GCC or compatible C compiler

Platforms: Windows, Linux, macOS

Dependencies: Standard C library

Terminal: ANSI escape sequence support required

This comprehensive report was generated based on the TicTacX project implementation by Group 2 for CSE115 Summer 2025, providing detailed analysis of the system architecture, implementation details, and technical specifications.