TicTacX - Classic Tic-Tac-Toe Game in C

CSE115 Programming Language I Project Report

Avishek Dutta (2524093042)
Md Ishrak Mashroor (2524709042)
Ashab Mahmud Raseen (2524767042)
Sneha Nandy (2524508042)

Section 4 - Group 2 - Summer 2025

Summer 2025

INTRODUCTION

Project Overview

TicTacX is a terminal-based implementation of the classic Tic-Tac-Toe game developed in C programming language for the CSE115 course. The project demonstrates fundamental programming concepts through a complete game application featuring human vs human and human vs AI gameplay modes.

Project Objectives

The primary objectives include implementing a fully functional Tic-Tac-Toe game, demonstrating modular programming through multi-file structure, applying cross-platform development techniques, incorporating ANSI escape sequences for enhanced user interface, and ensuring robust error handling with comprehensive input validation.

SYSTEM DESIGN AND ARCHITECTURE

Overall Architecture

TicTacX follows a modular design approach with clear separation of concerns. The architecture consists of four main modules: Main Module (application controller and menu system), Board Module (game visualization and rendering), Game Logic Module (core mechanics and rules), and AI Module (computer opponent functionality).

Project Structure

Listing 1: Project Directory Structure

```
tictacx/
          main.c              // Entry point and menu system
          support/
                board.c         // Board visualization
                board.h         // Board function declarations
```

```
                game.c          // Core game logic
                game.h          // Game function declarations
                GameAI.c        // AI implementation
                GameAI.h        // AI function declarations
          Makefile            // Build automation
          README.md           // Documentation
```

Data Flow Design

User input flows through validation layers before affecting game state. The visual representation updates dynamically based on game state changes, providing responsive user feedback throughout gameplay.

IMPLEMENTATION DETAILS

Cross-Platform Compatibility

The implementation uses preprocessor directives for platform detection and appropriate system commands:

Listing 2: Cross-Platform Console Management

```
void clear() {
    system("clear || cls"); // Works on both Unix and Windows
}

#ifdef _WIN32
    system("chcp 65001");  // Enable UTF-8 on Windows
#elif __linux__
    char *lang = getenv("LANG");
    if (!(lang && strstr(lang, "UTF-8"))) {
        printf("Linux: Unicode (UTF-8) not enabled\n");
        return 0;
    }
#endif
```

Enhanced User Interface

The board visualization uses ANSI escape sequences and Unicode box-drawing characters for professional appearance:

Listing 3: Board Rendering with ANSI Colors

```
void print_cell(char c) {
    if (c == X) printf(RED BLACK_BG " %c " RESET, c);
    else if (c == O) printf(BLUE BLACK_BG " %c " RESET, c);
    else printf(BLACK_BG "   " RESET);
}

void print_board(char board[3][3]) {
    printf("    1   2   3\n");
    printf("  " BLACK_BG WHITE "                                    \n"
        RESET);
    // Board rendering logic with colored cells
}
```

Input Validation System

Comprehensive input validation prevents common user errors:

Listing 4: Robust Input Validation

```
if (scanf("%d␣%d", &row, &col) != 2) {
    while (getchar() != '\n'); // Buffer flush
    clear();
    printf("Invalid␣input.␣Please␣enter␣two␣numbers.\n");
    continue;
}

if (row < 1 || row > 3 || col < 1 || col > 3) {
    clear();
    printf("Row␣and␣column␣must␣be␣between␣1␣and␣3.\n");
    continue;
}

if (board[row][col] != EMPTY) {
    clear();
    printf("Cell␣is␣already␣taken!\n");
    continue;
}
```

GAME LOGIC IMPLEMENTATION

Win Condition Detection

The game implements optimized O(1) win detection checking all possible winning combinations:

Listing 5: Win Detection Algorithm

```
// Row and column checks
for (int i = 0; i < 3; i++) {
    if (board[i][0] != EMPTY && board[i][0] == board[i][1]
        && board[i][1] == board[i][2]) {
        win_message(board, current_player);
        return;
    }
    if (board[0][i] != EMPTY && board[0][i] == board[1][i]
        && board[1][i] == board[2][i]) {
        win_message(board, current_player);
        return;
    }
}

// Diagonal checks
if (board[0][0] != EMPTY && board[0][0] == board[1][1]
    && board[1][1] == board[2][2]) {
    win_message(board, current_player);
    return;
}
```

Game Modes

The application supports two distinct gameplay modes managed through the `run_game()` function with mode parameters: Mode 1 (Human vs Human) and Mode 2 (Human vs AI). The AI uses random move generation with proper validation to ensure legal moves.

TECHNICAL SPECIFICATIONS

System Requirements

Hardware: Standard PC with minimum 512MB RAM
Software: C compiler supporting C11 standard, terminal with ANSI escape sequence support, Windows 10/11, Linux distributions, or macOS

Build System

The project uses Makefile-based compilation with organized build directory structure:

Listing 6: Makefile Build System

```
CC = gcc
CFLAGS = -Wall -Wextra -std=c11
SRCDIR = support
BUILDDIR = build
SOURCES = main.c $(SRCDIR)/game.c $(SRCDIR)/board.c $(SRCDIR)/GameAI.c
TARGET = $(BUILDDIR)/tictacx

$(TARGET): $(SOURCES) | $(BUILDDIR)
        $(CC) $(CFLAGS) $(SOURCES) -o $(TARGET)

$(BUILDDIR):
        mkdir -p $(BUILDDIR)

clean:
        rm -rf $(BUILDDIR)
```

Performance Analysis

Time Complexity: Win Detection: O(1), Move Validation: O(1), AI Move Generation: O(1) average
Space Complexity: O(1) constant memory usage with fixed 3×3 board
Memory Management: Efficient stack-based allocation eliminates memory leaks

USER EXPERIENCE AND TESTING

Menu System

The main menu provides intuitive navigation with five options: Human vs Human gameplay, Human vs AI gameplay, Game instructions, Project information, and Exit application. Each option includes proper error handling and user feedback.

Testing Methodology

Comprehensive testing covered input validation (valid/invalid numeric inputs, out-of-range values, duplicate moves), game logic (all win conditions, draw scenarios, player switching), and cross-platform compatibility (Windows PowerShell, Linux terminals, macOS compatibility).

Error Handling

The system handles various error conditions gracefully: invalid character inputs, out-of-range coordinates, occupied cell selection, incomplete input entries, and buffer overflow prevention through proper input flushing.

Practical Application of C Programming Concepts

This project provided hands-on experience with fundamental C programming concepts taught in CSE115. Students gained practical understanding of variable declarations, function definitions, array manipulation, and control flow statements through real-world application.

Problem-Solving Skills Development

Implementing the game logic required breaking down complex problems into smaller, manageable functions. This approach demonstrates structured problem-solving techniques essential for programming.

Code Organization and Documentation

The multi-file project structure taught the importance of code organization, proper commenting, and documentation. Students learned to write maintainable code that others can understand and modify.

Debugging and Testing Experience

Throughout development, students encountered and resolved various programming errors, providing valuable debugging experience. This included logical errors, syntax errors, and runtime issues.

CHALLENGES AND SOLUTIONS

Cross-Platform Compatibility Issues

Challenge: Different operating systems handle console commands and character encoding differently.

Solution: Used preprocessor directives to detect the operating system and execute appropriate commands for each platform.

Listing 7: Platform-Specific Solutions

```
#ifdef _WIN32
    system("chcp 65001");  // Enable UTF-8 on Windows
#elif __linux__
    char *lang = getenv("LANG");
    if (!(lang && strstr(lang, "UTF-8"))) {
        printf("Linux: Unicode (UTF-8) not enabled\n");
        return 0;
    }
#endif
```

Input Buffer Management

Challenge: Residual characters in the input buffer caused unexpected behavior during user input.

Solution: Implemented proper buffer flushing using `while (getchar() != '\n')` after each input operation.

ANSI Escape Sequence Compatibility

Challenge: Some terminals don't support ANSI escape sequences for colors and formatting.

Solution: Ensured UTF-8 support is available before using special characters and provided fallback options where necessary.

Memory Management

Challenge: Ensuring efficient memory usage without memory leaks.

Solution: Used stack-based allocation with fixed-size arrays, eliminating the need for dynamic memory management and preventing memory-related issues.

PROJECT DEVELOPMENT PROCESS

Planning and Design Phase

The team began with requirement analysis and system design. This included defining the game rules, user interface requirements, and technical specifications. The modular approach was decided early to facilitate parallel development.

Implementation Phase

Development was divided among team members with clear responsibilities:

- Board visualization and user interface design
- Core game logic and rule implementation
- AI functionality and random move generation
- Cross-platform compatibility and testing

Testing and Integration

Each module was tested individually before integration. The team conducted comprehensive testing across different platforms and user scenarios to ensure robust functionality.

Documentation and Finalization

Code documentation, README file creation, and this comprehensive report were completed to provide clear understanding of the project implementation and usage.

CODE QUALITY AND BEST PRACTICES

Coding Standards

The project follows consistent coding standards including:

- Meaningful variable and function names
- Proper indentation and spacing
- Comprehensive comments explaining complex logic
- Consistent bracket placement and style

Error Prevention

The implementation includes defensive programming practices such as input validation, boundary checking, and proper error handling to prevent common programming mistakes.

Code Reusability

Functions are designed to be reusable and modular. The board printing function, for example, can be called from any part of the program without modification.

Maintainability

The modular structure makes the code easy to maintain and extend. New features can be added without affecting existing functionality.

RESULTS AND FUTURE ENHANCEMENTS

Project Achievements

TicTacX successfully demonstrates modular C programming with cross-platform compatibility, professional terminal interface using ANSI escape sequences, robust input validation and error handling, and educational value for game development concepts.

Future Improvements

Potential enhancements include implementing smarter AI algorithms, adding difficulty levels and score tracking system, incorporating sound effects and animations, developing network multiplayer capabilities, creating save/load functionality for game persistence, and adding customizable player symbols and colors.

Educational Extensions

The project serves as a foundation for learning advanced programming concepts such as data structures, algorithms, and software design patterns. Students can extend this project to explore more complex programming topics.

CONCLUSION

TicTacX successfully fulfills the CSE115 course requirements while demonstrating professional software development practices. The modular architecture, comprehensive feature set, and cross-platform compatibility showcase the team's understanding of fundamental programming concepts and their practical application in game development.

The project effectively combines theoretical knowledge with practical implementation, resulting in a polished, user-friendly application that serves both as an entertaining game and an educational programming example. The clean code structure and thorough documentation make it an excellent reference for future C programming projects.

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

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

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