AURORA DEEMED TO BE UNIVERSITY
AURORA HIGHER EDUCATION AND RESEARCH ACADEMY

TASK 2

TIC-TAC-TOE AI

Implement an AI agent that plays the classic game of Tic-Tac-Toe against a human player. You can use algorithms like Minimax with or without Alpha-Beta Pruning to make the AI player unbeatable. This project will help you understand game theory and basic search algorithms.

By
K.Charmi

CONTENT PAGE

INTRODUCTION

Tic Tac Toe is a popular two-player game played on a 3x3 grid. The objective is for one player to align three of their marks ('X' or 'O') in a row, column, or diagonal before the opponent does. This project implements the game using the C programming language, focusing on fundamental programming concepts such as loops, functions, and arrays. The game provides a simple text-based interface where two players can compete interactively.

ABSTRACT

Project Overview:

- A console-based implementation of the classic Tic-Tac-Toe game in C.
- Two players take turns marking a 3x3 grid with 'X' or 'O'.
- The game checks for row, column, or diagonal victories.
- If all spaces are filled without a winner, the game declares a draw.
- Ensures players enter valid moves and prevents overwriting occupied spaces.
- Allows players to restart the game without rerunning the program.

Programming Concepts Used:

- 2D Arrays – To store and update the game board.
- Loops & Conditionals – To control game flow and validate moves.
- Functions – For modularity and better code readability.
- Boolean Logic – To track game status like win, draw, or ongoing.
- Input Handling & Buffer Clearing – Prevents input errors.

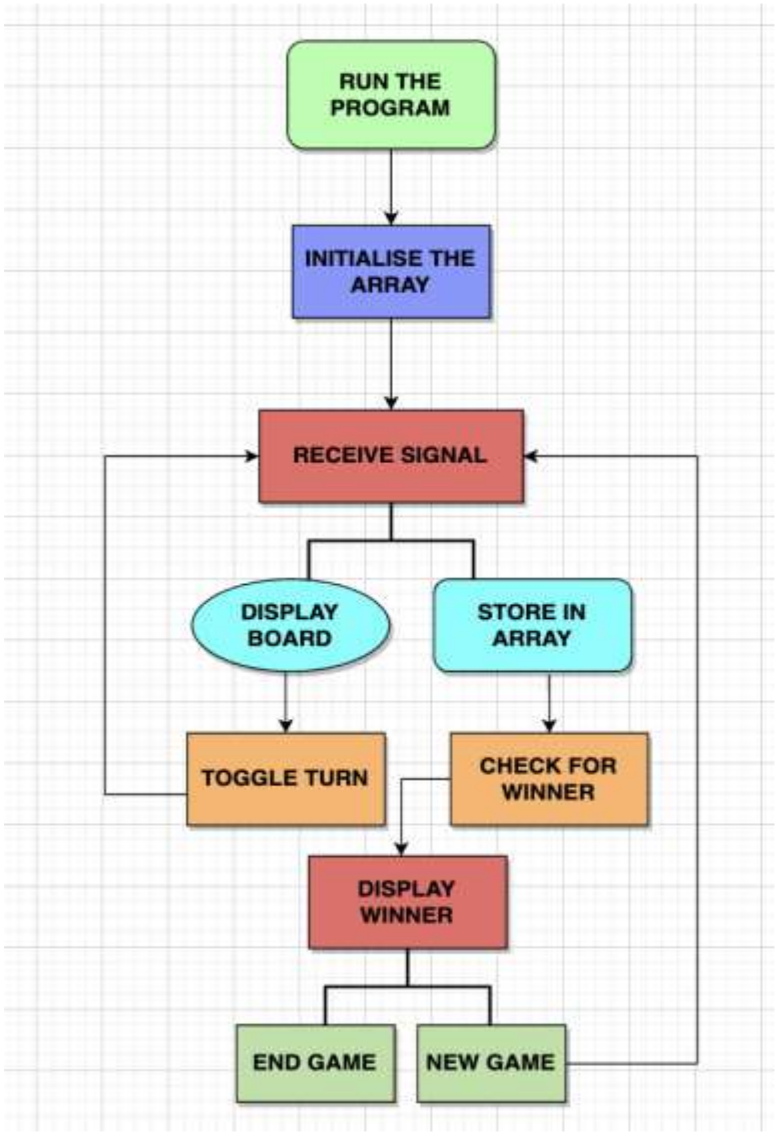
The project focuses on strengthening C programming skills, using arrays, loops, and functions for structured coding. It enhances problem-solving abilities and promotes modular programming. The goal is to develop a user-friendly and efficient Tic Tac Toe game in C.

OBJECTIVES

The primary objectives of this project are:

- To develop an interactive and easy-to-use console-based Tic-Tac-Toe game.
- To practice fundamental C programming concepts, such as loops, conditional statements, functions, and arrays.
- To handle user input effectively and ensure a smooth gaming experience.
- To implement game logic for detecting wins, draws, and invalid moves.

GAME LOGIC


RUN THE PROGRAM
INITIALISE THE ARRAY
RECEIVE SIGNAL
DISPLAY BOARD
STORE IN ARRAY
TOGGLE TURN
CHECK FOR WINNER
DISPLAY WINNER
END GAME
NEW GAME

METHODOLOGY USED IN THE PROJECT

Problem Definition

- Understanding the rules of Tic-Tac-Toe and translating them into logical conditions.
- A replay option after the game ends.

Algorithm Design

- Initialize the board with numbers 1-9.
- Display the board after every move.
- Take user input and validate the move.
- Update the board with 'X' or 'O' based on the player.
- Check win conditions (rows, columns, diagonals).
- Check for a draw if all spaces are filled.
- Switch players and repeat the process until the game ends.
- Ask for a replay after a win or draw.

Implementation in C

- Use a 2D array (char board[3][3]) to store the board state.
- Define functions to handle different aspects of the game.

Input Handling & Validation

- Ensuring valid moves by checking if the selected position is available.
- Handling invalid inputs (non-numeric characters) by clearing the input buffer.
- Preventing overwriting of existing marks ('X' or 'O’).

Game Logic Implementation

- Checking each row, column, and both diagonals for identical marks.
- If all 9 positions are occupied without a winner, the game ends in a draw.

Testing and Debugging

- Testing each function independently to ensure correct behavior.
- Playing multiple rounds to check for logical errors or unexpected behaviors.
- Fixing any detected input errors, incorrect game states, or infinite loops.

Optimization and Enhancements

- Improving user experience by displaying clear prompts and messages.
- Adding a replay option to allow users to restart without exiting the program.
- Optimizing code for readability and efficiency using functions and loops.

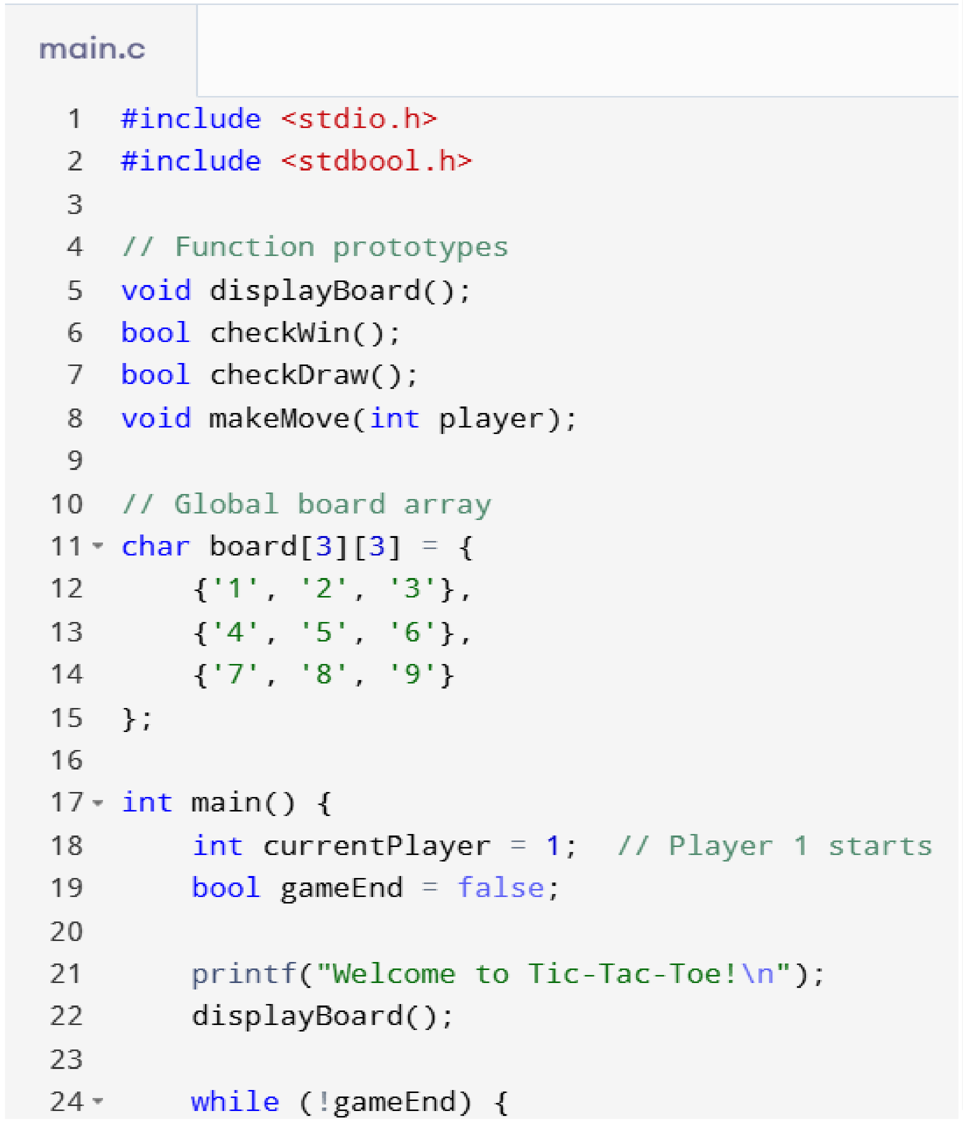
SOFTWARE REQUIREMENTS

1. **Operating System**
   - Windows / Linux / macOS
2. **Programming Language**
   - C Language
3. **Compiler**
   - Programiz (Online Compiler)
4. **Additional Libraries (if required)**
   - ncurses – If you want to enhance the user interface with a terminal-based graphical layout.
5. **System Requirements**
   - Processor: Any modern CPU (Intel, AMD, ARM)
   - RAM: At least 512MB
   - Storage: Less than 10MB

FUNCTIONAL REQUIREMENTS

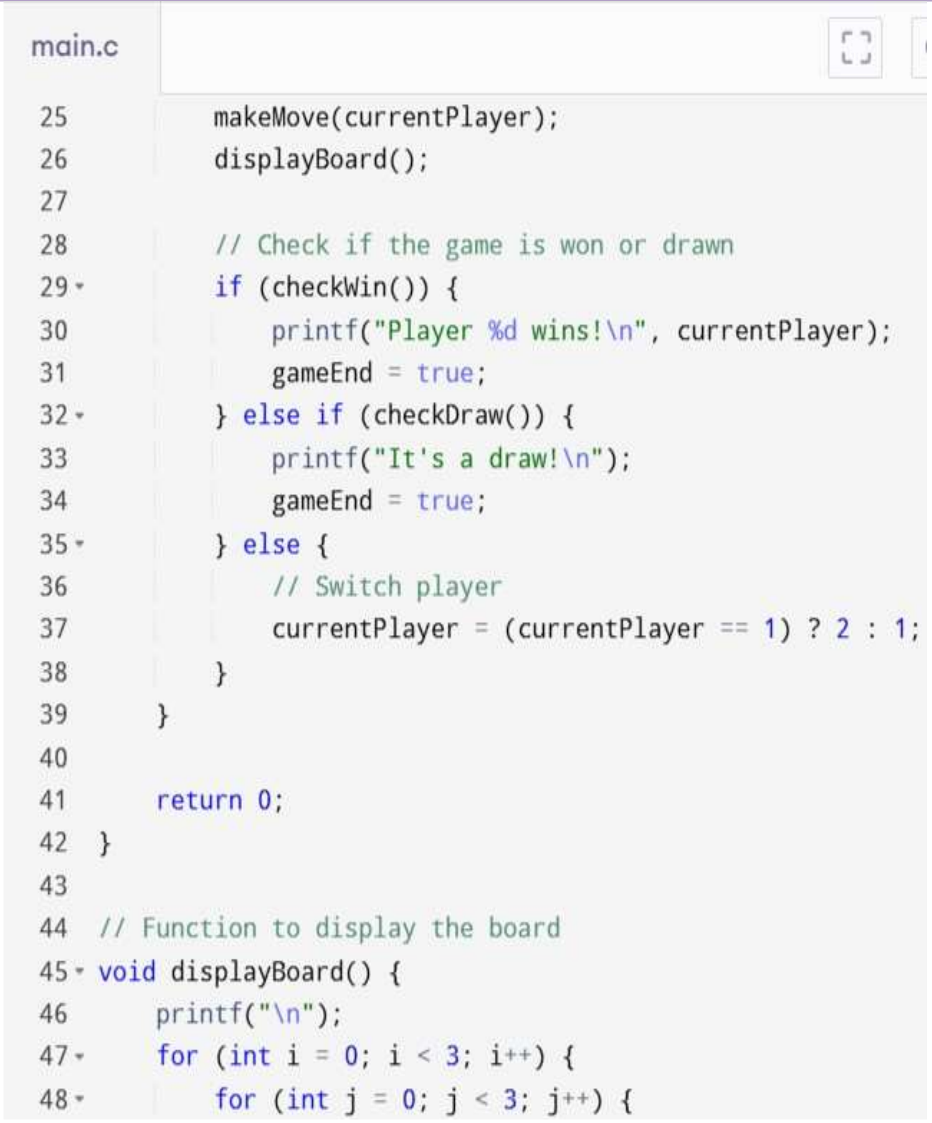
- Game Initialization
- Player Input Handling
- Game Mechanics and Logic
- Display Features
- Game Restart and Exit
- Error Handling
- Performance and Efficiency

EXECUTION SCREENSHOTS



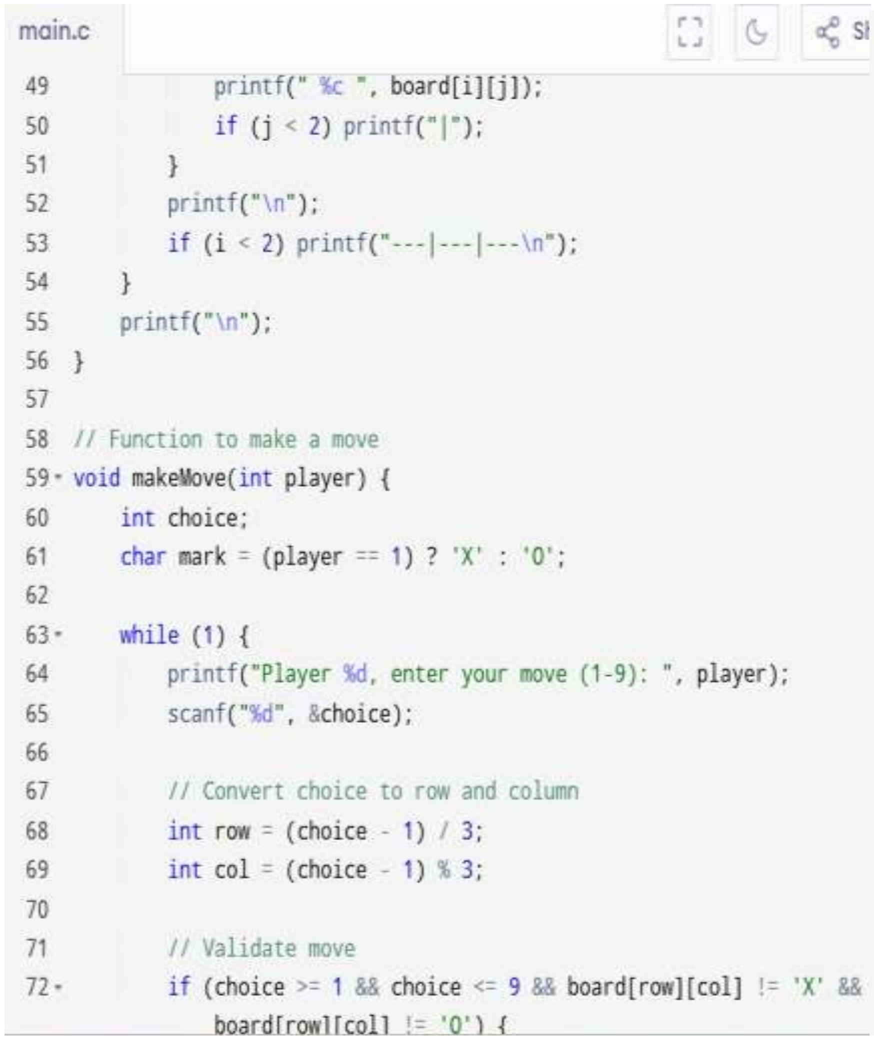
main.c

```c
#include <stdio.h>
#include <stdbool.h>

// Function prototypes
void displayBoard();
bool checkWin();
bool checkDraw();
void makeMove(int player);

// Global board array
char board[3][3] = {
    {'1', '2', '3'},
    {'4', '5', '6'},
    {'7', '8', '9'}
};

int main() {
    int currentPlayer = 1;  // Player 1 starts
    bool gameEnd = false;

    printf("Welcome to Tic-Tac-Toe!\n");
    displayBoard();

    while (!gameEnd) {
```

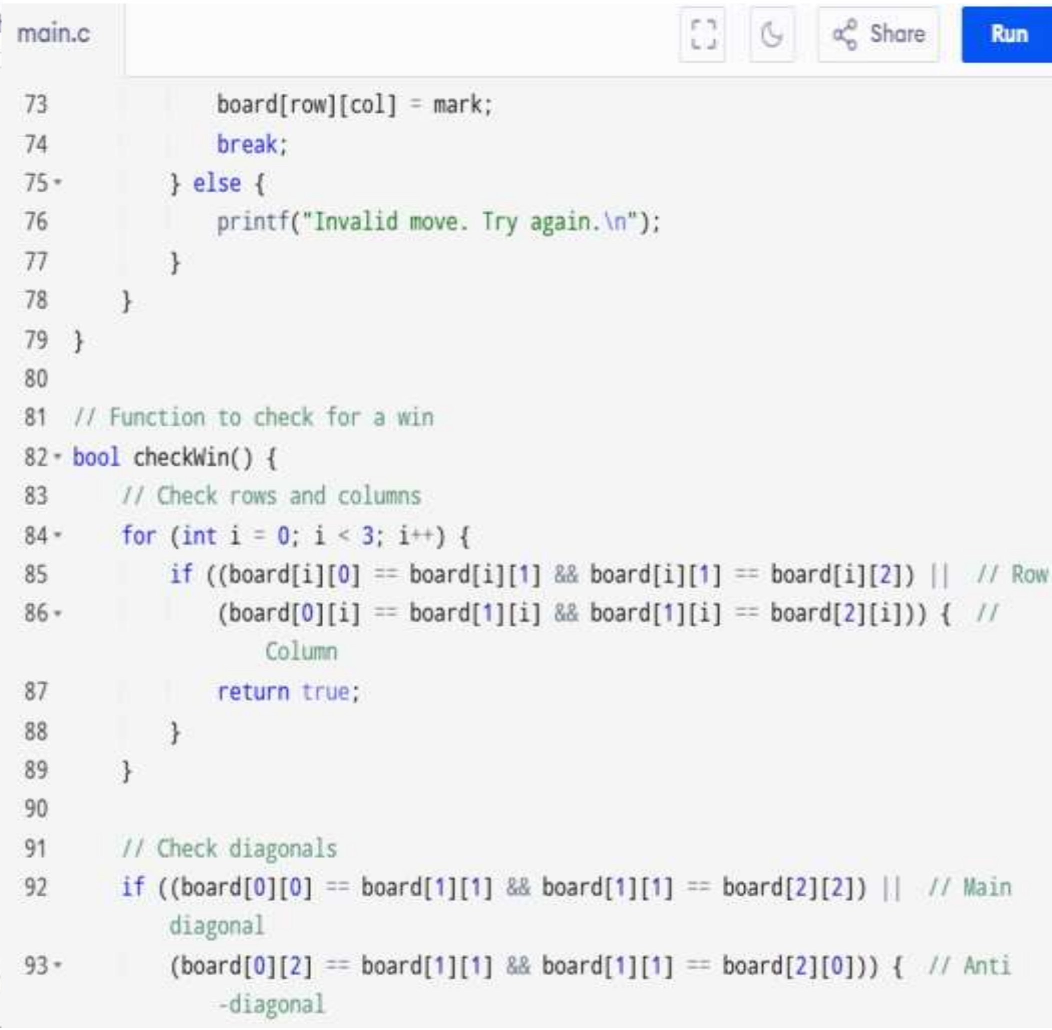


main.c

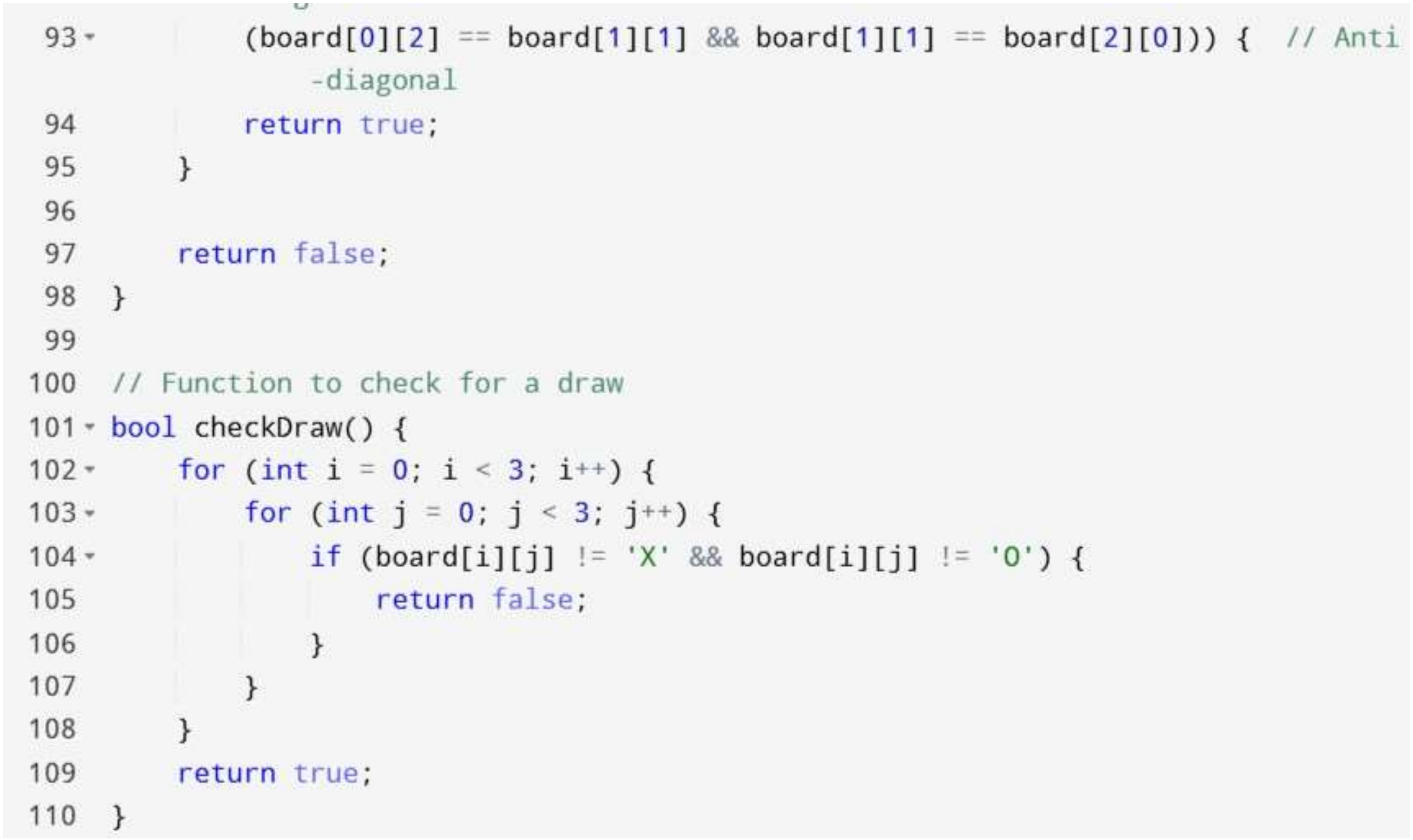
```c
        makeMove(currentPlayer);
        displayBoard();

        // Check if the game is won or drawn
        if (checkWin()) {
            printf("Player %d wins!\n", currentPlayer);
            gameEnd = true;
        } else if (checkDraw()) {
            printf("It's a draw!\n");
            gameEnd = true;
        } else {
            // Switch player
            currentPlayer = (currentPlayer == 1) ? 2 : 1;
        }
    }

    return 0;
}

// Function to display the board
void displayBoard() {
    printf("\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
```



```c
            printf(" %c ", board[i][j]);
            if (j < 2) printf("|");
        }
        printf("\n");
        if (i < 2) printf("---|---|---\n");
    }
    printf("\n");
}

// Function to make a move
void makeMove(int player) {
    int choice;
    char mark = (player == 1) ? 'X' : 'O';

    while (1) {
        printf("Player %d, enter your move (1-9): ", player);
        scanf("%d", &choice);

        // Convert choice to row and column
        int row = (choice - 1) / 3;
        int col = (choice - 1) % 3;

        // Validate move
        if (choice >= 1 && choice <= 9 && board[row][col] != 'X' && board[row][col] != 'O') {
```



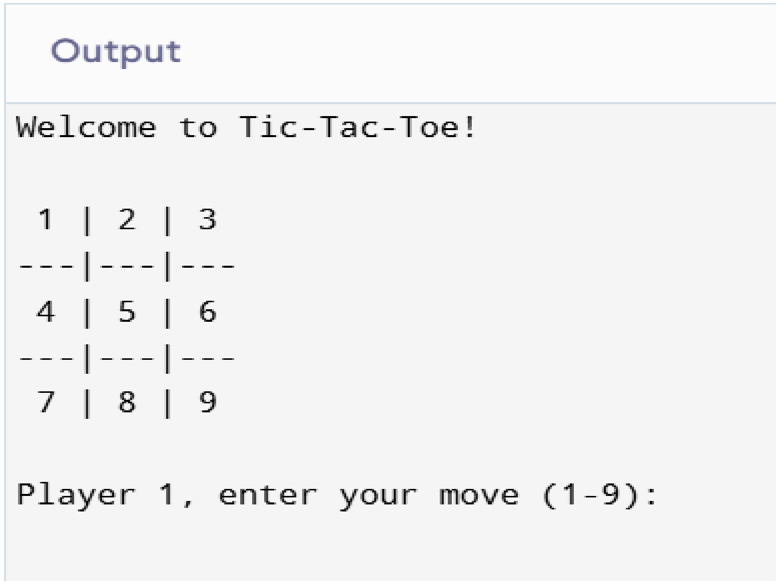
```c
            board[row][col] = mark;
            break;
        } else {
            printf("Invalid move. Try again.\n");
        }
    }
}

// Function to check for a win
bool checkWin() {
    // Check rows and columns
    for (int i = 0; i < 3; i++) {
        if ((board[i][0] == board[i][1] && board[i][1] == board[i][2]) ||  // Row
            (board[0][i] == board[1][i] && board[1][i] == board[2][i])) {  // Column
            return true;
        }
    }

    // Check diagonals
    if ((board[0][0] == board[1][1] && board[1][1] == board[2][2]) ||  // Main diagonal
        (board[0][2] == board[1][1] && board[1][1] == board[2][0])) {  // Anti-diagonal
```



```
        (board[0][2] == board[1][1] && board[1][1] == board[2][0])) {  // Anti-diagonal
        return true;
    }

    return false;
}

// Function to check for a draw
bool checkDraw() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] != 'X' && board[i][j] != 'O') {
                return false;
            }
        }
    }
    return true;
}
```

TEST CASES

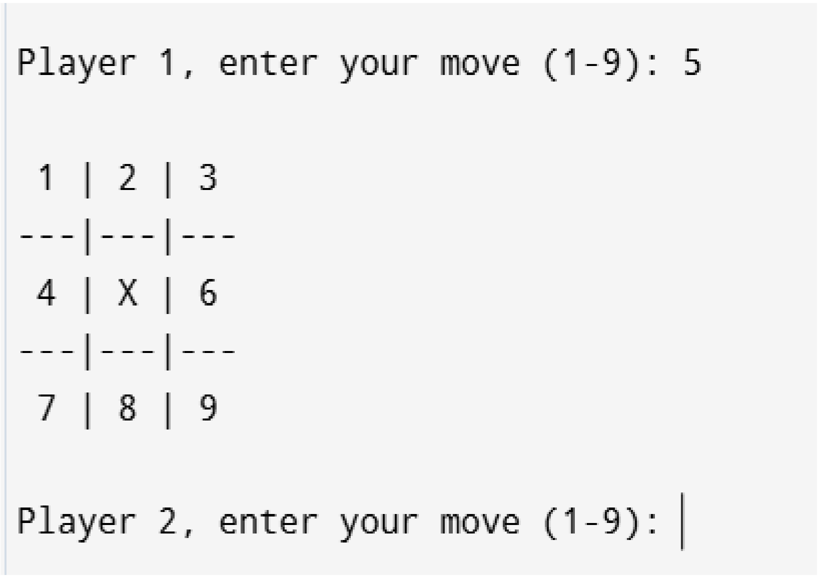
1



Output

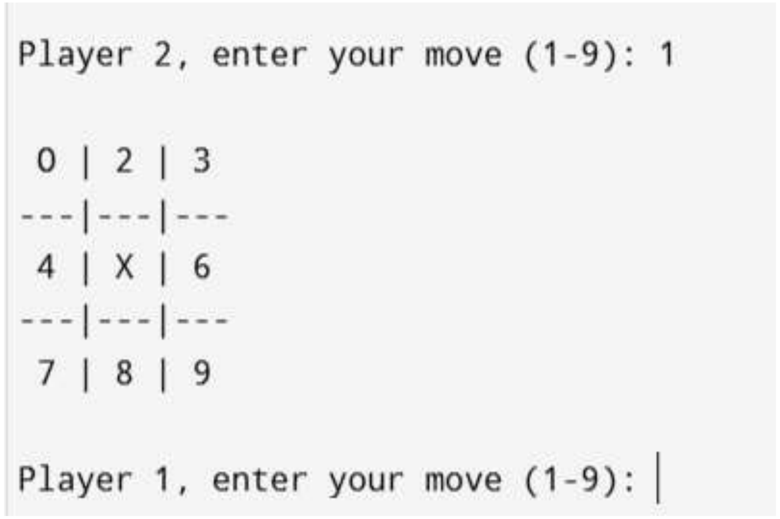
```
Welcome to Tic-Tac-Toe!

 1 | 2 | 3
---|---|---
 4 | 5 | 6
---|---|---
 7 | 8 | 9

Player 1, enter your move (1-9):
```

2



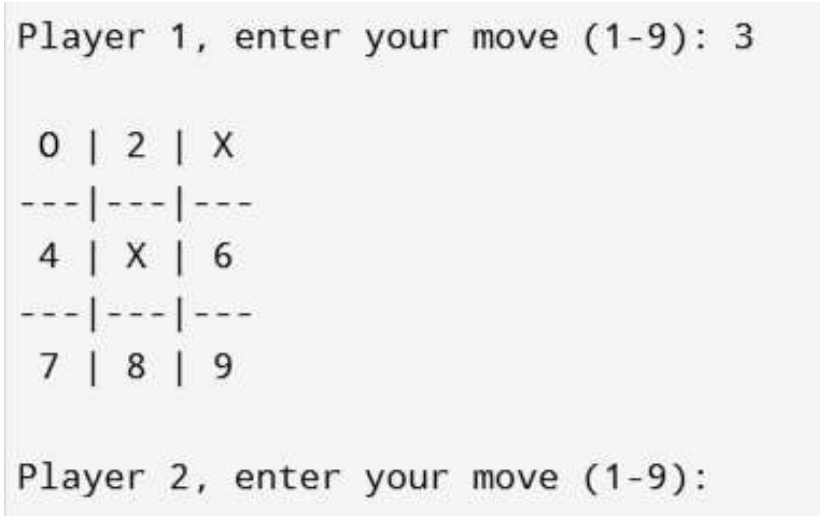
```
Player 1, enter your move (1-9): 5

 1 | 2 | 3
---|---|---
 4 | X | 6
---|---|---
 7 | 8 | 9

Player 2, enter your move (1-9):
```

3



```
Player 2, enter your move (1-9): 1

 O | 2 | 3
---|---|---
 4 | X | 6
---|---|---
 7 | 8 | 9

Player 1, enter your move (1-9):
```

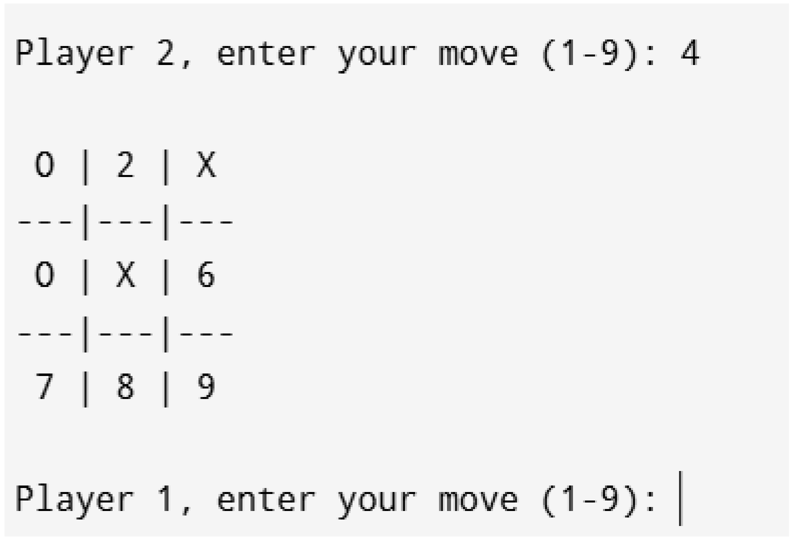
4



```
Player 1, enter your move (1-9): 3

 O | 2 | X
---|---|---
 4 | X | 6
---|---|---
 7 | 8 | 9

Player 2, enter your move (1-9):
```

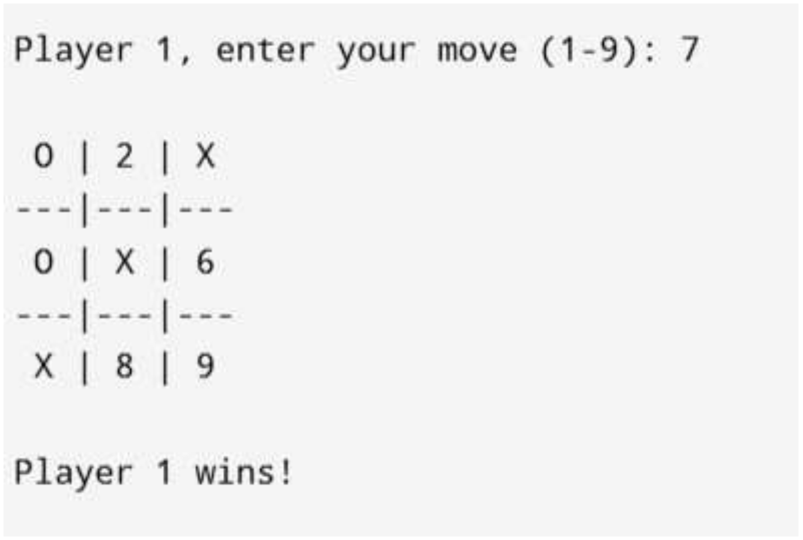
Output/Results

5



```
Player 2, enter your move (1-9): 4


 O | 2 | X
---|---|---
 O | X | 6
---|---|---
 7 | 8 | 9

Player 1, enter your move (1-9): 
```

6



```
Player 1, enter your move (1-9): 7


 O | 2 | X
---|---|---
 O | X | 6
---|---|---
 X | 8 | 9

Player 1 wins!
```

CREATIVITY AND INNOVATION

1. AI-Powered Single-Player Mode – Implement an AI opponent using random moves or the Minimax algorithm.
2. Score Tracking and Leaderboard – Track wins, losses, and draws across multiple rounds with file storage.
3. Voice-Controlled Moves – Implement speech recognition for hands-free gameplay.
4. Animated or Interactive Console – Add animations, hints, and real-time move suggestions.

CONTRIBUTION TO SOCIETY

Enhancing Programming Skills

- Helps beginners learn C programming, logic, and problem-solving.

Entertainment & Mental Stimulation

- Improves strategic thinking and provides fun gameplay.

Educational Tool

- Used in schools and STEM education to teach game logic and AI concepts.

CONCLUSION

- This C-based Tic-Tac-Toe game successfully implements a two-player turn-based strategy game using fundamental programming concepts like arrays, loops, conditional statements, and functions.
- The project effectively demonstrates the logic behind board games, user input handling, and win-condition detection.
- This project serves as an excellent learning tool for beginners in C programming and can be further enhanced by integrating AI (Minimax Algorithm) for single-player mode, a graphical user interface (GUI), or an online multiplayer feature.

THANK
YOU