NED University of Engineering and Technology

TIC-TAC-TOE

Project Report

Programming Fundamentals (CT-175)

Group Members:

Muhammad Abdullah Ayub CTCY-042

Muhammad Azhan Javed CTCY-046

Muhammad Hasan Khan CTCY-047

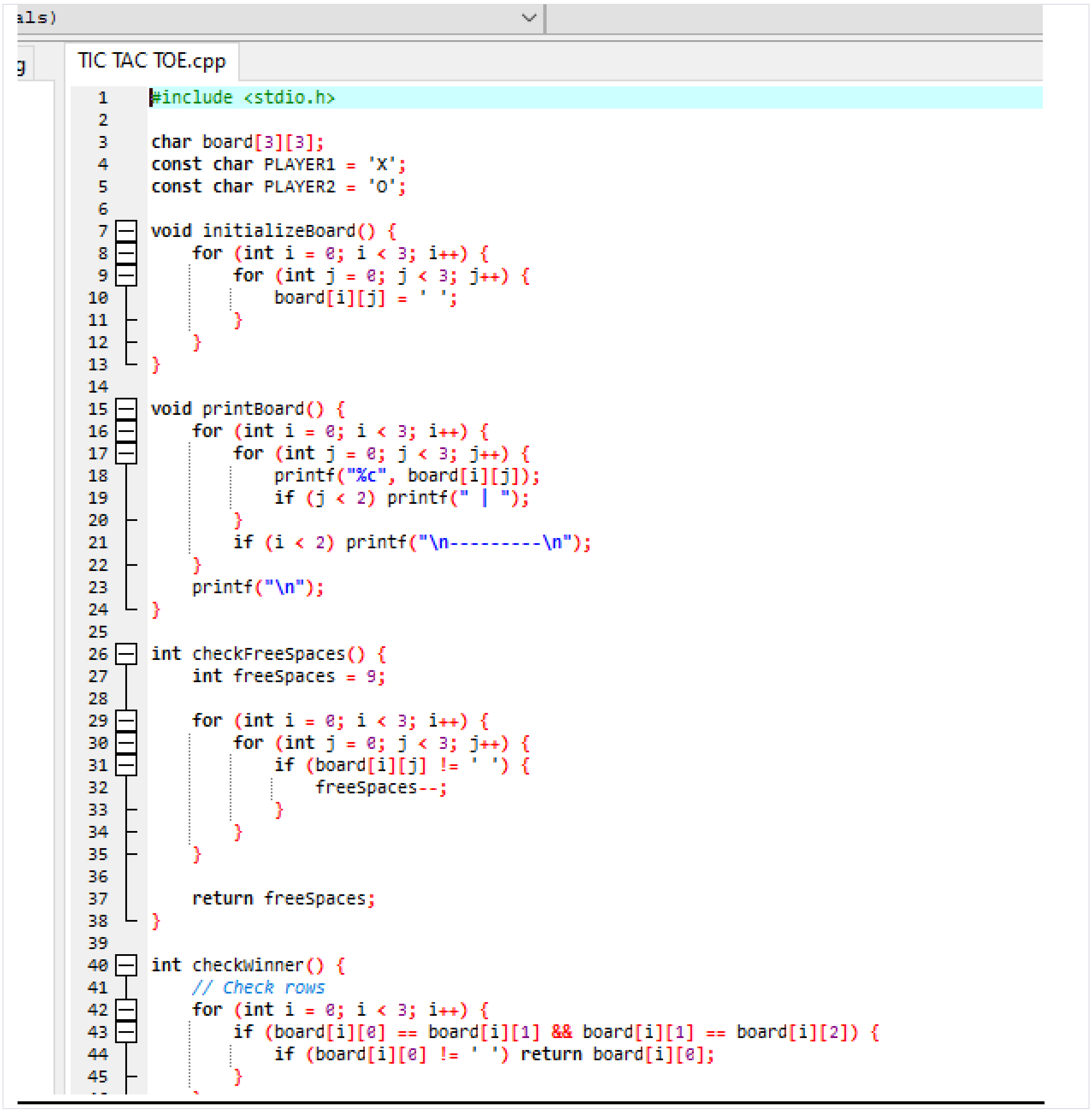
1. Problem Statement:

The problem at hand is to implement a two-player Tic-Tac-Toe game using C programming language. The game should provide a user-friendly interface for players to take turns, make moves, and determine the winner.

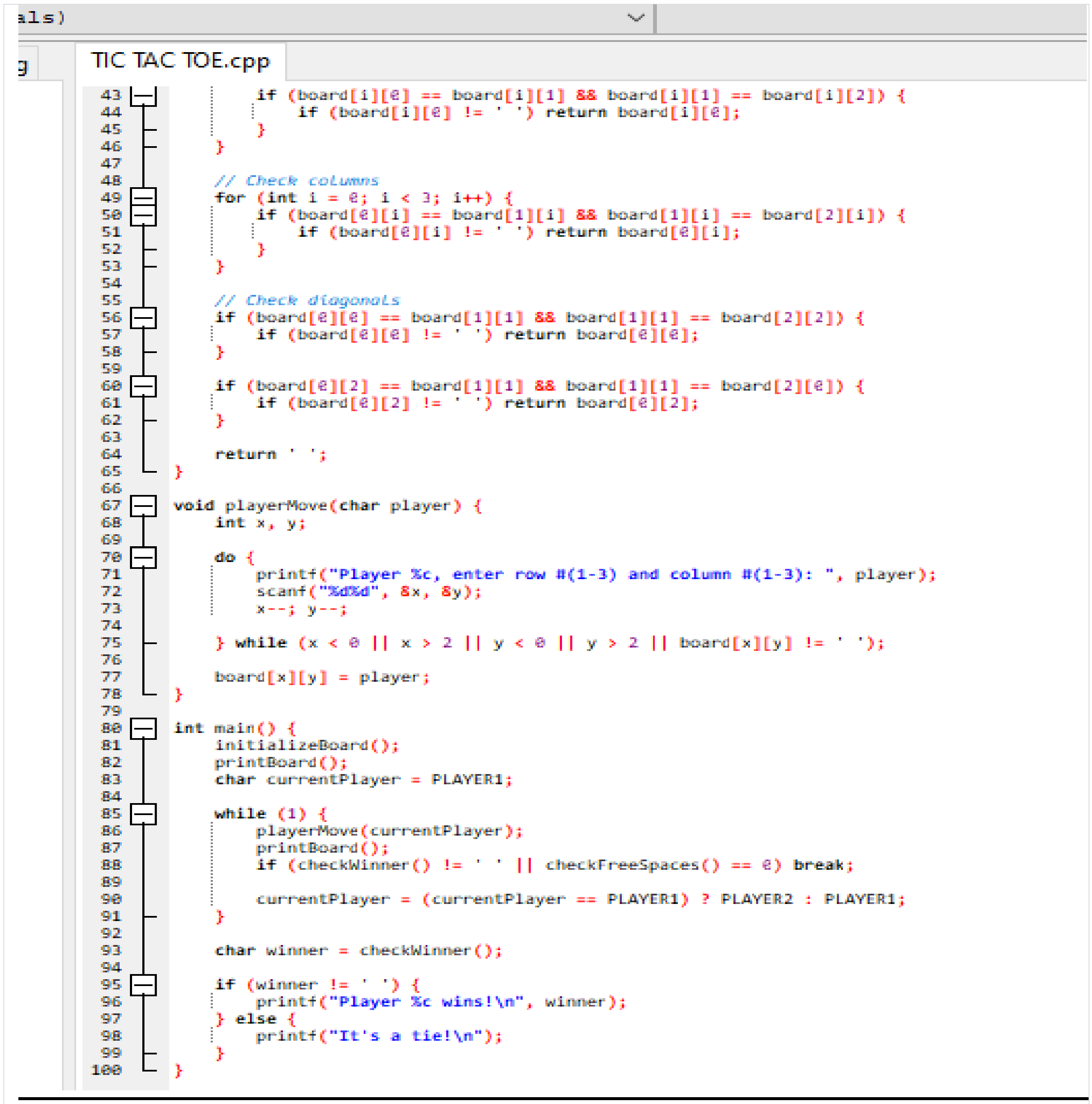
2. Why Did We Choose This Project?

The Tic-Tac-Toe game is a classic and widely recognized concept. Choosing this project allows for the application of fundamental programming concepts, such as arrays, loops, conditionals, and user input handling. Additionally, it provides an opportunity to implement a simple algorithm for checking the winner and handling the game flow.

CODE SNIPPETS:



```
#include <stdio.h>

char board[3][3];
const char PLAYER1 = 'X';
const char PLAYER2 = 'O';

void initializeBoard() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            board[i][j] = ' ';
        }
    }
}

void printBoard() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%c", board[i][j]);
            if (j < 2) printf(" | ");
        }
        if (i < 2) printf("\n---------\n");
    }
    printf("\n");
}

int checkFreeSpaces() {
    int freeSpaces = 9;

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] != ' ') {
                freeSpaces--;
            }
        }
    }

    return freeSpaces;
}

int checkWinner() {
    // Check rows
    for (int i = 0; i < 3; i++) {
        if (board[i][0] == board[i][1] && board[i][1] == board[i][2]) {
            if (board[i][0] != ' ') return board[i][0];
        }
```



```
        if (board[i][0] == board[i][1] && board[i][1] == board[i][2]) {
            if (board[i][0] != ' ') return board[i][0];
        }
    }

    // Check columns
    for (int i = 0; i < 3; i++) {
        if (board[0][i] == board[1][i] && board[1][i] == board[2][i]) {
            if (board[0][i] != ' ') return board[0][i];
        }
    }

    // Check diagonals
    if (board[0][0] == board[1][1] && board[1][1] == board[2][2]) {
        if (board[0][0] != ' ') return board[0][0];
    }

    if (board[0][2] == board[1][1] && board[1][1] == board[2][0]) {
        if (board[0][2] != ' ') return board[0][2];
    }

    return ' ';
}

void playerMove(char player) {
    int x, y;

    do {
        printf("Player %c, enter row #(1-3) and column #(1-3): ", player);
        scanf("%d%d", &x, &y);
        x--; y--;

    } while (x < 0 || x > 2 || y < 0 || y > 2 || board[x][y] != ' ');

    board[x][y] = player;
}

int main() {
    initializeBoard();
    printBoard();
    char currentPlayer = PLAYER1;

    while (1) {
        playerMove(currentPlayer);
        printBoard();
        if (checkWinner() != ' ' || checkFreeSpaces() == 0) break;

        currentPlayer = (currentPlayer == PLAYER1) ? PLAYER2 : PLAYER1;
    }

    char winner = checkWinner();

    if (winner != ' ') {
        printf("Player %c wins!\n", winner);
    } else {
        printf("It's a tie!\n");
    }
}
```

3. Algorithms and Methods Used:

- **Board Initialization (initializeBoard):**
  The board is a 3x3 grid represented by a 2D array. The **initializeBoard** function initializes the board with empty spaces.

- **Printing the Board (printBoard):**
  This function prints the current state of the board, separating cells with '|' and rows with dashes for a clear visual representation.
- **Checking Free Spaces (checkFreeSpaces):**
  Counts the number of remaining empty spaces on the board to determine if the game is a tie.
- **Checking Winner (checkWinner):**
  The function checks for a winning condition by examining rows, columns, and diagonals. If a player has three in a row, the game declares that player as the winner.
- **Player Move (playerMove):**
  Takes player input for the row and column, validates the input, and updates the board with the player's symbol.
- **Main Loop (main):**
  The main function orchestrates the game by initializing the board, printing it, and entering a loop where players take turns making moves until there is a winner or a tie.

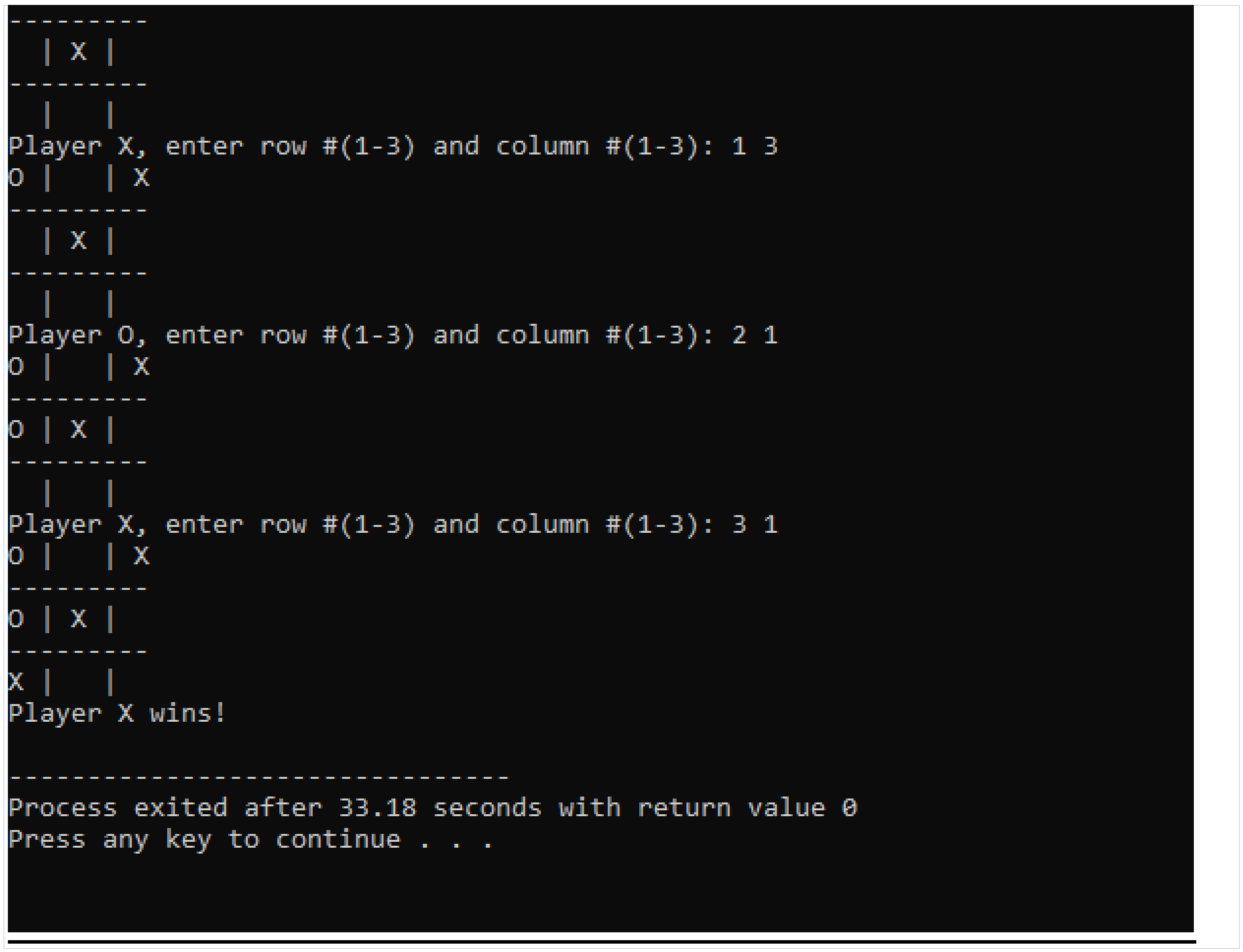
4. Features:

- 2-player Tic-Tac-Toe game.
- Clear visual representation of the game board.
- Input validation to ensure a valid move from players.
- Dynamic switching between Player 1 (X) and Player 2 (O).
- Detection of a winning player or a tie game.

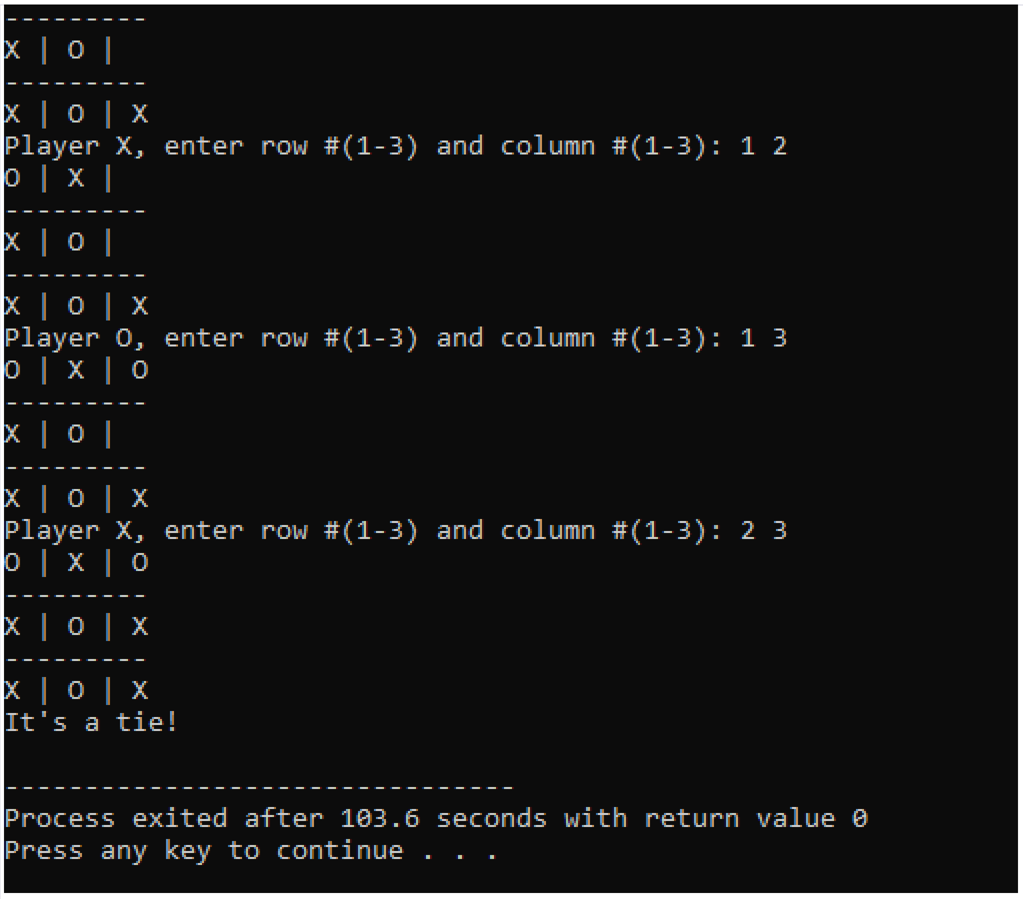
5. Limitations of Project:

- The current implementation doesn't include error handling for invalid input types (non-integer input).
- The game does not have a graphical user interface (GUI) and relies on the console for input and output.
- Limited scalability: The game is specifically designed for a 3x3 grid, and modifications are needed for larger board sizes.
- Lack of an AI opponent: The game is limited to two human players; adding an AI opponent would enhance the gaming experience.

OUTPUT SCREENSHOTS:



```
---------
  | X |
---------
  |   |
Player X, enter row #(1-3) and column #(1-3): 1 3
O |   | X
---------
  | X |
---------
  |   |
Player O, enter row #(1-3) and column #(1-3): 2 1
O |   | X
---------
O | X |
---------
  |   |
Player X, enter row #(1-3) and column #(1-3): 3 1
O |   | X
---------
O | X |
---------
X |   |
Player X wins!

--------------------------------
Process exited after 33.18 seconds with return value 0
Press any key to continue . . .
```



```
---------
X | O |
---------
X | O | X
Player X, enter row #(1-3) and column #(1-3): 1 2
O | X |
---------
X | O |
---------
X | O | X
Player O, enter row #(1-3) and column #(1-3): 1 3
O | X | O
---------
X | O |
---------
X | O | X
Player X, enter row #(1-3) and column #(1-3): 2 3
O | X | O
---------
X | O | X
---------
X | O | X
It's a tie!

--------------------------------
Process exited after 103.6 seconds with return value 0
Press any key to continue . . .
```

CONCLUSION:

In conclusion, the Tic-Tac-Toe game serves as a practical project to reinforce basic programming concepts and logic. Enhancements can be made to address limitations and expand the game's features, making it an excellent starting point for further development and learning.