Koneru Lakshmaiah Education Foundation

(Deemed to be University)

FRESHMAN ENGINEERING DEPARTMENT

A Project Based Lab Report

On

Tic-Tac-Toe Game

SUBMITTED BY:

2300030609	S. Ruchita
2300030674	T. Jaswanth
2300030682	T. Yujith Teja
2300030565	P.Nithisha

UNDER THE GUIDANCE OF

Mr. Krishna Suresh

Assistant Professor, CSE.

KL UNIVERSITY

Green fields, Vaddeswaram – 522 502
Guntur Dt., AP, India.

ACKNOWLEDGEMENTS

It is great pleasure for me to express my gratitude to our honorable President **Sri. Koneru Satyanarayana**, for giving the opportunity and platform with facilities in accomplishing the project-based laboratory report.

I express sincere gratitude to our Director **Dr. A. Jagadeesh** for his administration towards our academic growth.

I express sincere gratitude to HOD-BES-1 **Dr. D. Haritha** for her leadership and constant motivation provided in successful completion of our academic semester. I record it as my privilege to deeply thank you for providing us with the efficient faculty and facilities to make our ideas into reality.

I express my sincere thanks to our project supervisor **Mr. Krishna Suresh** for the novel association of ideas, encouragement, appreciation, and intellectual zeal which motivated us to venture this project successfully.

Finally, I am pleased to acknowledge my indebtedness to all those who devoted themselves directly or indirectly to making this project a success.

Project Associates…

2300030609	**S. Ruchita**
2300030674	**T. Jaswanth**
2300030682	**T. Yujith Teja**
2300030565	**P.Nithisha**

ABSTRACT

This C programming project focuses on implementing basic file handling operations to manage records. The project allows users to add, list, modify, and delete records stored in a file. The main functionalities are encapsulated within functions, promoting modularity and code readability. Users interact with the program through a console-based interface, where they can input record data and perform desired operations.

The project utilizes file handling functions such as `fopen()`, `fclose()`, `fwrite()`, `fread()`, `fprintf()`, and `fscanf()` to manipulate records in the file. Key programming concepts such as functions, pointers, files, and arrays are employed to manage data efficiently. Additionally, basic error handling is implemented to handle file I/O errors and invalid user inputs.

The program provides a simple menu-driven interface, prompting users to choose from options like adding a record, listing all records, modifying a record, deleting a record, or exiting the program. Each functionality is implemented with appropriate user prompts and feedback messages to enhance user experience.

This project serves as a practical exercise for students to reinforce their understanding of file handling operations, data structures, and programming concepts in C. It provides a foundation for further enhancements, such as implementing additional features like searching, sorting, or data validation, to create a more robust file management system.

.

INDEX

INTRODUCTION

Tic-Tac-Toe, often called Noughts and Crosses, is a simple yet captivating game that has been enjoyed by generations. In this project, we embark on the creation of a Tic-Tac-Toe game implemented in the C programming language, featuring a thrilling showdown between a human player and a computer opponent.

The essence of Tic-Tac-Toe lies in its simplicity: two players, each represented by either "O" or "X," take turns marking cells on a 3x3 grid. Victory is achieved by forming a row, column, or diagonal of three consecutive marks. However, the game is not merely about placing symbols on a grid; it's about strategy, foresight, and outsmarting the opponent.

Our project encapsulates the spirit of this timeless game while adding a twist: the inclusion of an artificial intelligence (AI) opponent. This computer adversary is programmed to challenge the human player with cunning moves and strategic decisions, providing an immersive and dynamic gaming experience.

Throughout this project, we delve into the intricacies of C programming, from managing data structures and handling user input to implementing sophisticated algorithms for the computer AI. We strive to create an intuitive and responsive interface that allows players to enjoy the game seamlessly while immersing themselves in the thrill of strategic gameplay.

Join us as we embark on a journey to bring the classic game of Tic-Tac-Toe to life in the realm of C programming, where human intellect clashes with machine intelligence in an epic battle for victory.

AIM

Advantages:-

1. 1. **Programming Skills Development**: This project provides an excellent opportunity to enhance programming skills in C. Developers can practice concepts like data structures, arrays, functions, and conditional statements while implementing the game logic and user interface.
2. 2. **Algorithmic Understanding**: Creating an AI opponent for the computer player requires understanding and implementing algorithms such as minimax, alpha-beta pruning, or heuristic evaluation. This project offers a practical way to delve into these advanced concepts and improve algorithmic problem-solving skills.
3. 3. **User Engagement and Entertainment**: Offering a Human vs. Computer mode adds depth and replay value to the game, attracting players who seek a challenging solo experience. It enhances user engagement by providing an opportunity for players to test their skills against a dynamic AI opponent.
4. 4. **Portfolio Enhancement**: Completing this project adds a tangible product to a developer's portfolio, showcasing their proficiency in C programming, game development, and problem-solving. It demonstrates the ability to create interactive and engaging applications, which can be impressive to potential employers or collaborators.

Disadvantages: -

1. **Limited Complexity**: Tic-Tac-Toe's simplicity might not fully showcase advanced programming techniques or algorithms.
2. **Predictable AI**: Depending on the AI algorithm, the computer opponent may become predictable, reducing challenge.
3. **Limited Interaction**: Solo gameplay lacks the social aspect of multiplayer modes, potentially leading to decreased player interest.
4. **Graphical Interface Complexity**: Implementing a GUI in C can be challenging, especially for beginners.

FUTURE ENCHANCEMENTS

1. **Enhanced AI Algorithms**: Introduce more sophisticated AI techniques for a more dynamic and challenging opponent.

2. **Multiplayer Support**: Add local or online multiplayer modes for increased player engagement.

3. **Improved User Interface**: Enhance the GUI with visuals, animations, and sound effects for a more immersive experience.

4. **Additional Features**: Introduce new game modes, customizable options, and features like scoreboards or achievements to add depth and variety.

SYSTEM REQUIREMENTS

- **SOFTWARE REQUIREMENTS:**

  The major software requirements of the project are as follows:

  Language : Turbo-C

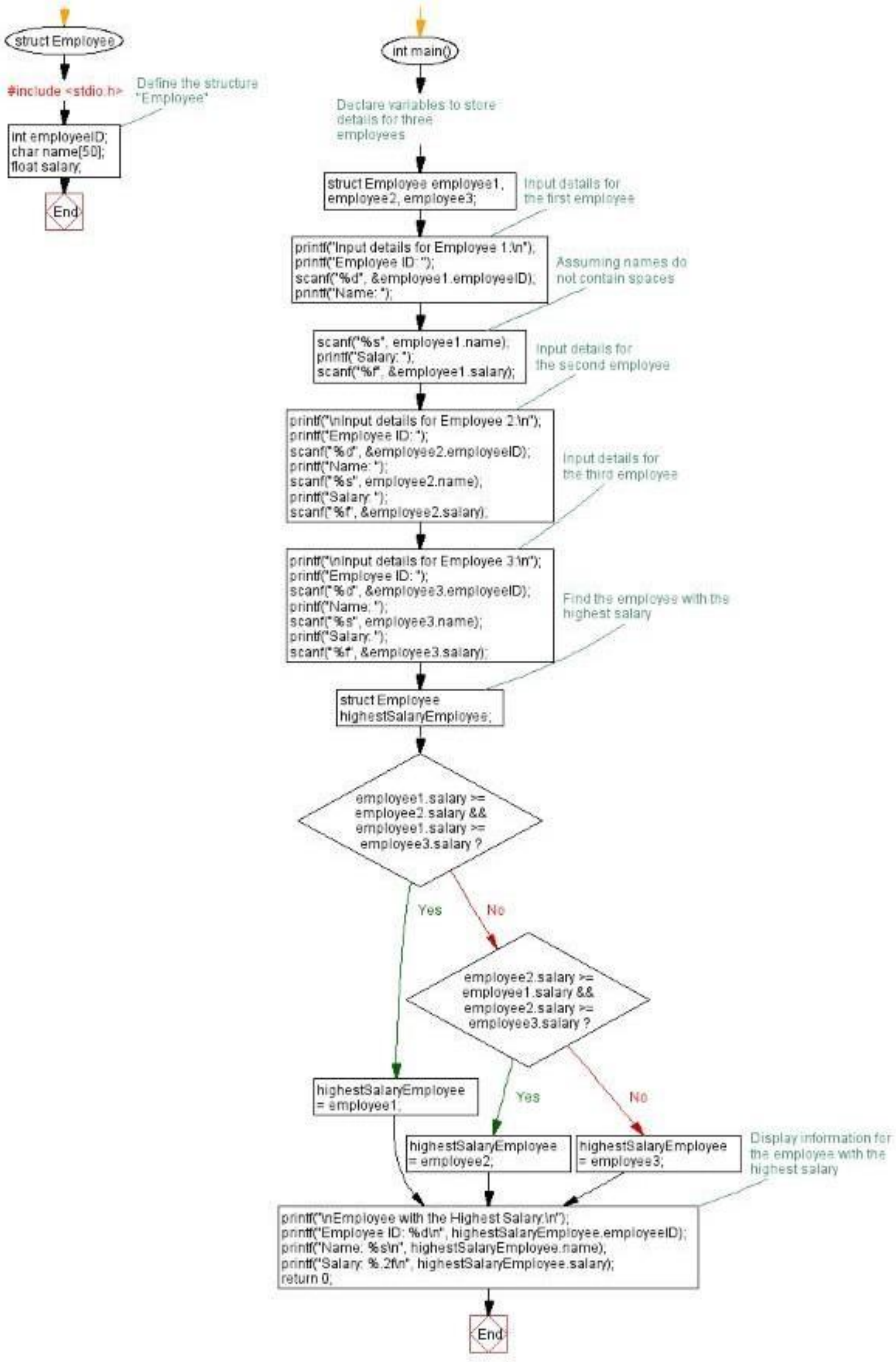
  Operating system**:** Windows Xp or later.
- 1. Operating System: Windows, macOS, Linux.
-
- 2. C Compiler: GCC, Clang, or Microsoft Visual C++ Compiler.
-
- 3. Development Environment: Visual Studio Code, Code::Blocks, or Eclipse.
- 4. Graphics Library Optional: SDL or GTK for GUI implementation.
- These software requirements provide the necessary tools for developing and running the Tic-Tac-Toe game in C.HARDWARE
- REQUIREMENTS:

  The hardware requirements that map towards the software are as follows:

  RAM :

  Processor :

FLOWCHART


struct Employee
#include <stdio.h>
Define the structure "Employee"
End
int main()
Declare variables to store details for three employees
Input details for the first employee
Assuming names do not contain spaces
Input details for the second employee
Input details for the third employee
Find the employee with the highest salary
Yes
No
Yes
No
Display information for the employee with the highest salary
End

```
int employeeID;
char name[50];
float salary;
```

```
struct Employee employee1, employee2, employee3;
```

```
printf("Input details for Employee 1:\n");
printf("Employee ID: ");
scanf("%d", &employee1.employeeID);
printf("Name: ");
```

```
scanf("%s", employee1.name);
printf("Salary: ");
scanf("%f", &employee1.salary);
```

```
printf("\nInput details for Employee 2:\n");
printf("Employee ID: ");
scanf("%d", &employee2.employeeID);
printf("Name: ");
scanf("%s", employee2.name);
printf("Salary: ");
scanf("%f", &employee2.salary);
```

```
printf("\nInput details for Employee 3:\n");
printf("Employee ID: ");
scanf("%d", &employee3.employeeID);
printf("Name: ");
scanf("%s", employee3.name);
printf("Salary: ");
scanf("%f", &employee3.salary);
```

```
struct Employee
highestSalaryEmployee;
```

```
employee1.salary >=
employee2.salary &&
employee1.salary >=
employee3.salary ?
```

```
employee2.salary >=
employee1.salary &&
employee2.salary >=
employee3.salary ?
```

```
highestSalaryEmployee
= employee1;
```

```
highestSalaryEmployee
= employee2;
```

```
highestSalaryEmployee
= employee3;
```

```
printf("\nEmployee with the Highest Salary:\n");
printf("Employee ID: %d\n", highestSalaryEmployee.employeeID);
printf("Name: %s\n", highestSalaryEmployee.name);
printf("Salary: %.2f\n", highestSalaryEmployee.salary);
return 0;
```

ALGORITHM

1. Initialize the Game Board

   - Create a 3x3 grid to represent the game board.

   - Initially, set all cells of the grid to empty (' ').

2. Display the Game Board

   - Write a function to display the current state of the game board on the screen.

   - The function should print the grid with the current marks ('X', 'O', or empty).

3. Get Player Input

   - Prompt the current player to enter their move.

   - Accept input for the row and column where the player wants to place their mark.

   - Ensure the input is within the valid range (0-2) and the selected cell is empty.

4. Update the Game Board

   - Place the current player's mark ('X' or 'O') at the specified row and column on the game board.

   - Update the grid with the player's move.

5. Check for Win

   - After each move, check if the current player has won the game.

   - Check if any row, column, or diagonal has all three cells filled with the same mark.

   - If a win condition is met, declare the current player as the winner and end the game.

6. Check for Draw

   - If no win condition is met and the game board is full, declare the game as a draw.

- Check if all cells on the game board are filled with marks.

7. Switch Players

- After each turn, switch the current player to the other player ('X' to 'O' or vice versa).

- Alternate between players until the game ends.

8. Repeat

- Repeat steps 2-7 until the game is won by a player or ends in a draw.

9. End of Game

- Once a player wins or the game ends in a draw, display the final state of the game board.

- Print a message indicating the winner or declaring a draw.

- Terminate the program or allow players to restart the game if desired.

IMPLEMENTATION

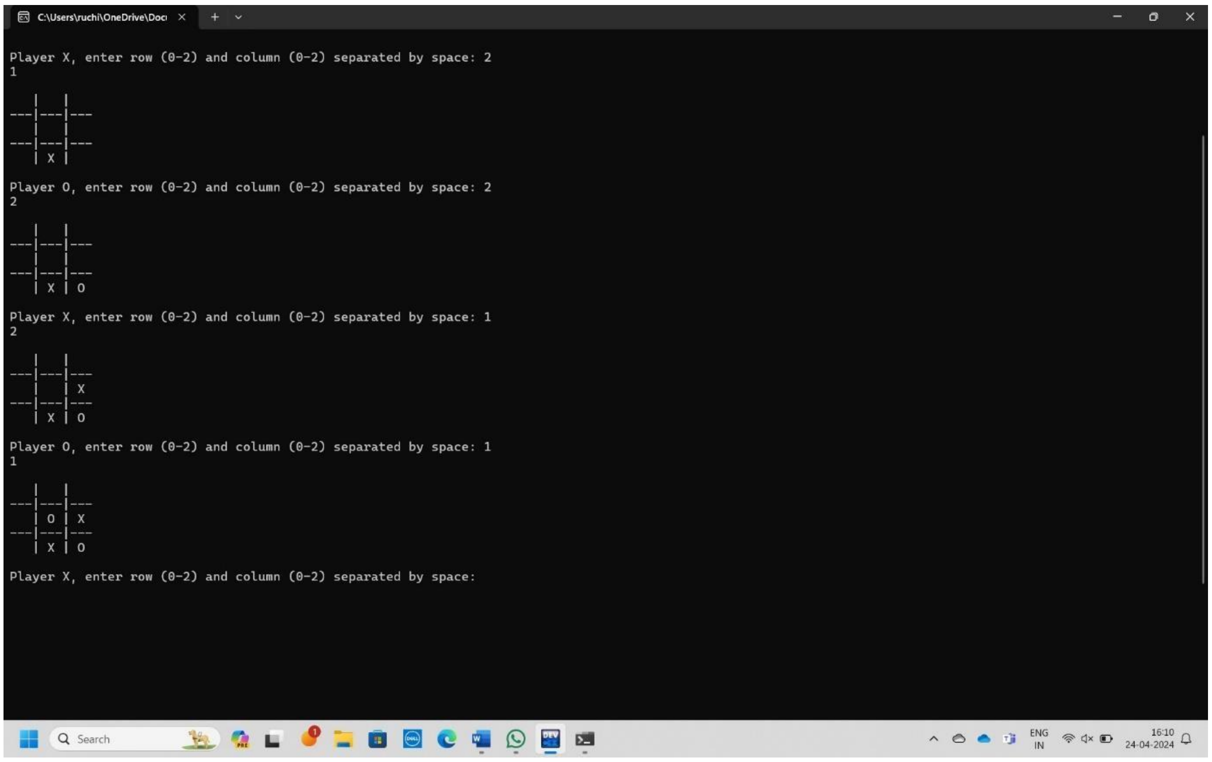
```
#include <stdio.h>
#include <stdbool.h>

void initializeBoard(char board[3][3]);
void displayBoard(char board[3][3]);
bool checkWin(char board[3][3], char player);
bool checkDraw(char board[3][3]);

int main() {
    char board[3][3];
    char currentPlayer = 'X';
    int row, col;

    initializeBoard(board);

    while (true) {
        displayBoard(board);

        printf("Player %c, enter row (0-2) and column (0-2) separated by space: ",
currentPlayer);
        scanf("%d %d", &row, &col);

        if (row < 0 || row > 2 || col < 0 || col > 2 || board[row][col] != ' ') {
            printf("Invalid move! Try again.\n");
            continue;
        }
        board[row][col] = currentPlayer;

        if (checkWin(board, currentPlayer)) {
            displayBoard(board);
            printf("Player %c wins!\n", currentPlayer);
            break;
        }

        if (checkDraw(board)) {
            displayBoard(board);
            printf("The game is a draw!\n");
            break;
        }

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

    return 0;
}

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

```
}


void displayBoard(char board[3][3]) {
    printf("\n");
    printf(" %c | %c | %c \n", board[0][0], board[0][1], board[0][2]);
    printf("---|---|---\n");
    printf(" %c | %c | %c \n", board[1][0], board[1][1], board[1][2]);
    printf("---|---|---\n");
    printf(" %c | %c | %c \n", board[2][0], board[2][1], board[2][2]);
    printf("\n");
}


bool checkWin(char board[3][3], char player) {
    for (int i = 0; i < 3; i++) {
        if (board[i][0] == player && board[i][1] == player && board[i][2] ==
player) {
            return true;
        }
    }
    for (int j = 0; j < 3; j++) {
        if (board[0][j] == player && board[1][j] == player && board[2][j] ==
player) {
            return true;
        }
    }
    if ((board[0][0] == player && board[1][1] == player && board[2][2] ==
player) ||
        (board[0][2] == player && board[1][1] == player && board[2][0] ==
player)) {
        return true;
    }
    return false;
}

bool checkDraw(char board[3][3]) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ') {
                return false;
            }
        }
    }
    return true;
}
```

INTEGRATION AND SYSTEM TESTING

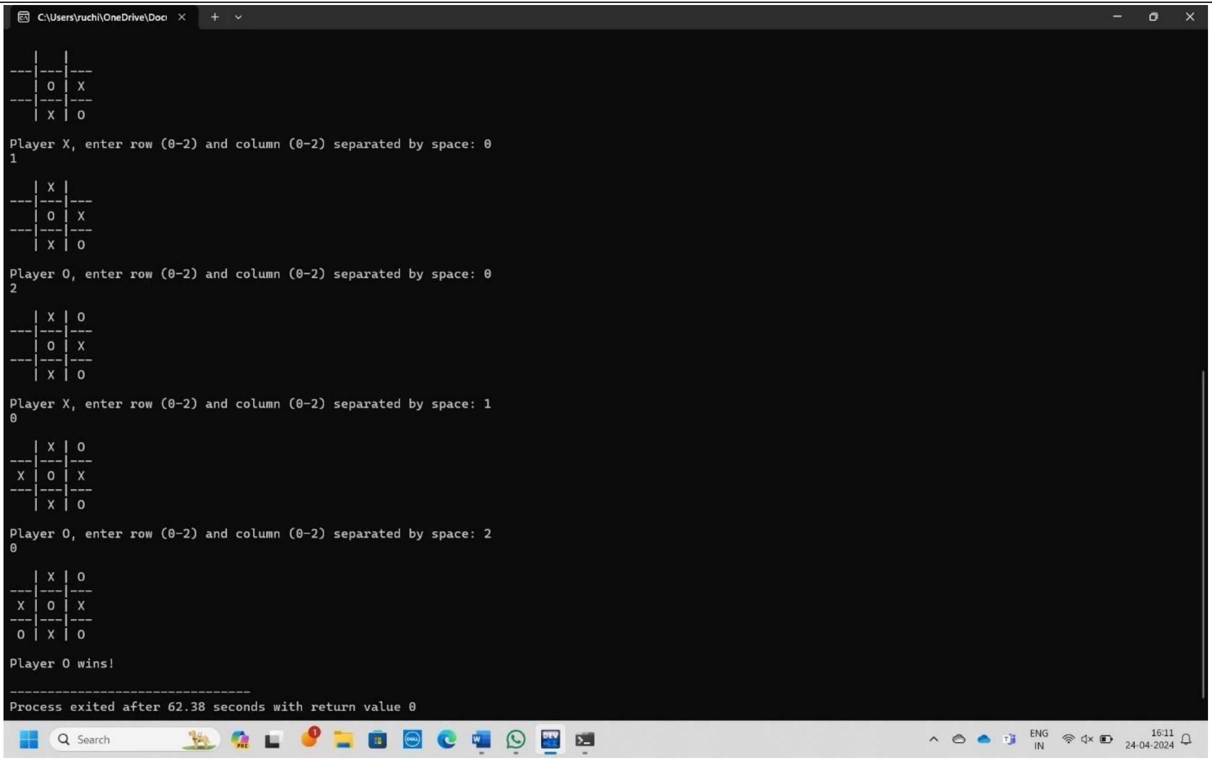
OUTPUTS

Screen Shot:



C:\Users\ruchi\OneDrive\Doc

```
Player X, enter row (0-2) and column (0-2) separated by space: 2
1

   |   |
---|---|---
   |   |
---|---|---
   | X |

Player O, enter row (0-2) and column (0-2) separated by space: 2
2

   |   |
---|---|---
   |   |
---|---|---
   | X | O

Player X, enter row (0-2) and column (0-2) separated by space: 1
2

   |   |
---|---|---
   |   | X
---|---|---
   | X | O

Player O, enter row (0-2) and column (0-2) separated by space: 1
1

   |   |
---|---|---
   | O | X
---|---|---
   | X | O

Player X, enter row (0-2) and column (0-2) separated by space:
```



```
   |   |
---|---|---
   | O | X
---|---|---
   | X | O

Player X, enter row (0-2) and column (0-2) separated by space: 0
1

   | X |
---|---|---
   | O | X
---|---|---
   | X | O

Player O, enter row (0-2) and column (0-2) separated by space: 0
2

   | X | O
---|---|---
   | O | X
---|---|---
   | X | O

Player X, enter row (0-2) and column (0-2) separated by space: 1
0

   | X | O
---|---|---
 X | O | X
---|---|---
   | X | O

Player O, enter row (0-2) and column (0-2) separated by space: 2
0

   | X | O
---|---|---
 X | O | X
---|---|---
 O | X | O

Player O wins!

--------------------------------
Process exited after 62.38 seconds with return value 0
```

CONCLUSION

In conclusion, the Tic-Tac-Toe game project has been successfully implemented to provide an engaging and interactive experience for two players. The project achieves its objectives of allowing players to choose their symbols ('O' or 'X'), marking cells on the game board, and determining a winner based on completing a row, column, or diagonal with their respective symbols.

Throughout the development process, various programming concepts and techniques have been applied, including data structures, conditional statements, loops, and user input/output handling. The game's logic has been carefully designed to ensure fair gameplay and to handle different scenarios, such as detecting a win or a draw condition.

Additionally, error handling mechanisms have been incorporated to address invalid user inputs and ensure the robustness of the game. The user interface provides clear instructions and prompts to guide players through the gameplay process.

In future iterations of the project, enhancements such as implementing different game modes (e.g., player vs. computer), adding graphical elements for a more visually appealing interface, and optimizing the code for efficiency could be considered. Overall, this project serves as a valuable learning experience in programming fundamentals and game development concepts