University of Petroleum and Energy Studies **Course Code: CSEG1032**

TIC TAC TOE

Major Project

C programming

Instructor: Dr. Tanu Singh

<u>Submitted By</u>: Krishna Grover

SAP ID: 590023545

B.TECH-CSE

ABSTRACT:

This project implements a console-based **Tic Tac Toe** game in C where a human player (X) plays against the computer (O).

The program supports two difficulty modes and uses rule-based heuristics for the computer; the report explains the code structure, game logic, and algorithmic choices.

Key outcomes: playable game, score tracking, and a “God” mode that prioritizes optimal moves.

3. Problem Definition

Goal: Build a robust, user-friendly Tic Tac Toe program that enforces valid moves, detects wins/draws, tracks scores, and offers two difficulty modes. Constraints: 3×3 board, console I/O, deterministic rules, and responsive AI.

The objective of this project to create a simple TIC-TAC-TOE GAME using C.

4. System Design:

1. Start → Initialize RNG and difficulty → Loop: play_game() → display board → alternate turns → check win/draw → update score → ask replay → End.

Algorithm Overview:

- **Win check: scan rows, columns, diagonals for three identical marks.**
- **Draw check: board full and no winner.**
- **Computer strategy (heuristic):**

  1. **Try immediate winning move.**
  2. **Block opponent immediate win.**
  3. **If God mode, take center.**

4. Take a corner.

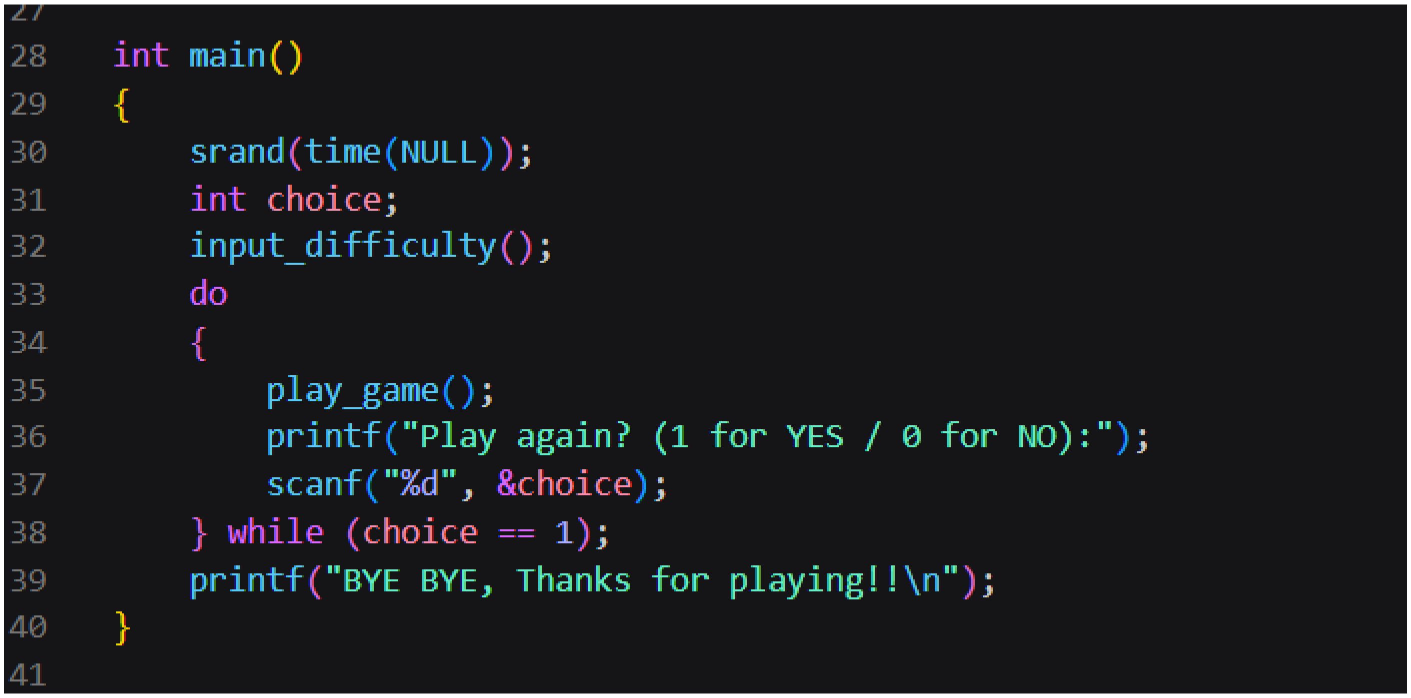
5. Take first available cell.

- **For provably optimal play use Minimax which evaluates all possible moves recursively and selects the best outcome for the maximizer.**

<u>Flowchart</u>

Algorithm for Main:

1. Seed random number generator using the current time so starting player is randomized: srand(time(NULL));.
2. Call input_difficulty() and loop inside that function until the user selects a valid difficulty (1 or 2).
3. Initialize local control variable choice (e.g., int choice;).
4. Enter replay loop: do { ... } while (choice == 1);. Inside the loop:
    a. Call play_game() which runs one full game, updates score, and returns when the game ends.
    b. Prompt user for replay: print "Play again? (1 for YES / 0 for NO):" and read choice with scanf("%d", &choice);.
    c. Validate replay input: if input is not 0 or 1, re-prompt until valid (recommended to clear stdin on invalid reads).
5. After loop ends, print a farewell message and optionally final scores.
6. Return from main (e.g., return 0;).

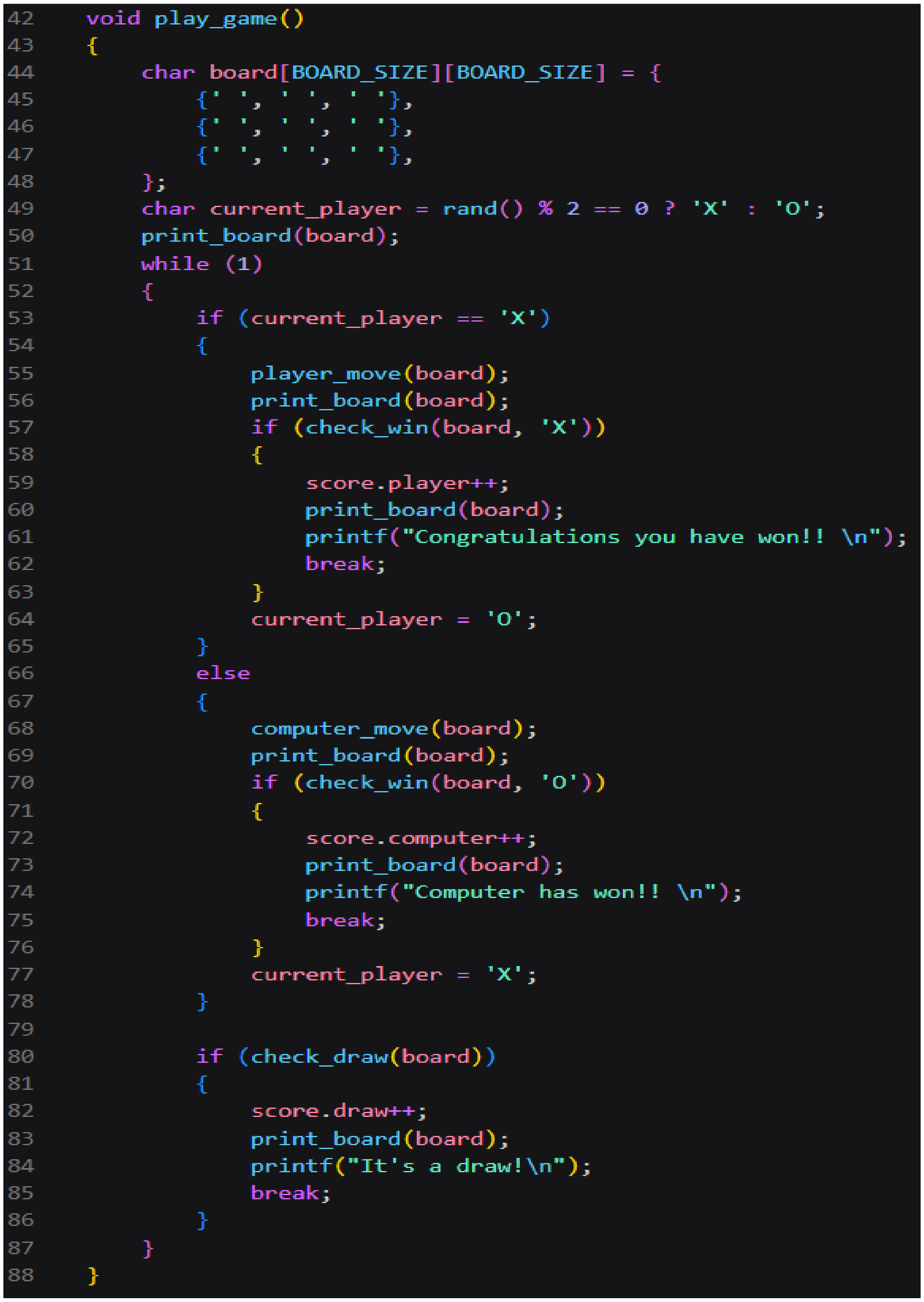


```c
int main()
{
    srand(time(NULL));
    int choice;
    input_difficulty();
    do
    {
        play_game();
        printf("Play again? (1 for YES / 0 for NO):");
        scanf("%d", &choice);
    } while (choice == 1);
    printf("BYE BYE, Thanks for playing!!\n");
}
```

Algorithm for play_game():

1. Initialize board: create a 3×3 board filled with blank spaces ' '.
2. Choose starting player: set current_player to 'X' or 'O' at random.
3. Display board: call print_board(board) to show initial state and scores.
4. Main loop: repeat until a terminal condition is reached:
    a. If current_player is 'X':
        i. Call player_move(board) to read and apply a validated player move.
        ii. Call print_board(board) to update display.
        iii. If check_win(board, 'X') returns true:
            1. Increment score.player.

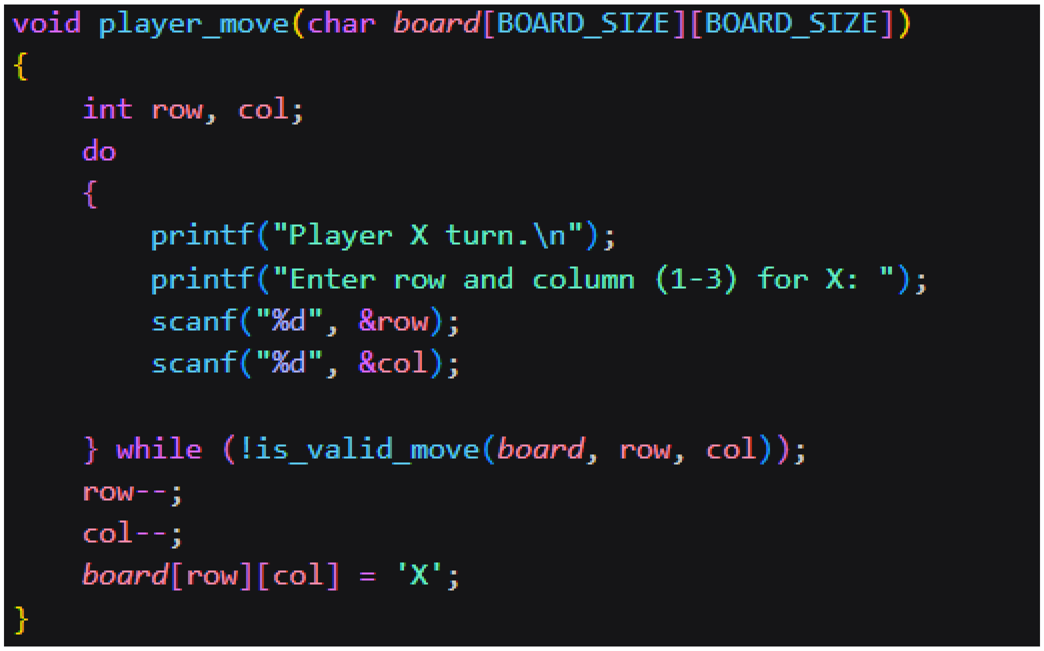
```
            2. Optionally call print_board(board) again.
            3. Print a player-win message and break the loop.
        iv. Set current_player = 'O'.
    b. Else (current_player is 'O'):
        i. Call computer_move(board) to compute and apply the
           computer’s move.
        ii. Call print_board(board) to update display.
        iii. If check_win(board, 'O') returns true:
            1. Increment score.computer.
            2. Optionally call print_board(board) again.
            3. Print a computer-win message and break the loop.
        iv. Set current_player = 'X'.
5. After either move, check for draw:
    a. If check_draw(board) returns true:
    b. Increment score.draw.
    c. Call print_board(board).
    d. Print a draw message and break the loop.
6. End: return control to caller (scores persist in global score).
```



```c
void play_game()
{
    char board[BOARD_SIZE][BOARD_SIZE] = {
        {' ', ' ', ' '},
        {' ', ' ', ' '},
        {' ', ' ', ' '},
    };
    char current_player = rand() % 2 == 0 ? 'X' : 'O';
    print_board(board);
    while (1)
    {
        if (current_player == 'X')
        {
            player_move(board);
            print_board(board);
            if (check_win(board, 'X'))
            {
                score.player++;
                print_board(board);
                printf("Congratulations you have won!! \n");
                break;
            }
            current_player = 'O';
        }
        else
        {
            computer_move(board);
            print_board(board);
            if (check_win(board, 'O'))
            {
                score.computer++;
                print_board(board);
                printf("Computer has won!! \n");
                break;
            }
            current_player = 'X';
        }

        if (check_draw(board))
        {
            score.draw++;
            print_board(board);
            printf("It's a draw!\n");
            break;
        }
    }
}
```

ALGORITHM FOR player_move():

1. Prompt the player that it is their turn and request row and column numbers in the range 1–3.
2. Read two integers from standard input for row and col.
3. Validate the move by checking:
    a. row and col are within 1 and BOARD_SIZE (inclusive).
    b. the target cell board[row-1][col-1] is empty (' ').
4. If invalid, repeat the prompt and read until a valid move is provided.
5. When valid, convert to zero-based indices (row--, col--) and set board[row][col] = 'X'.
6. Return to the caller with the board updated.



```c
void player_move(char board[BOARD_SIZE][BOARD_SIZE])
{
    int row, col;
    do
    {
        printf("Player X turn.\n");
        printf("Enter row and column (1-3) for X: ");
        scanf("%d", &row);
        scanf("%d", &col);

    } while (!is_valid_move(board, row, col));
    row--;
    col--;
    board[row][col] = 'X';
}
```

ALGORITHM FOR computer_move():

High level steps:

1. **Scan for immediate winning move for O**.
2. **Scan to block immediate winning move for X**.
3. **If God difficulty take center**.
4. **Take any available corner**.
5. **Take the first available cell**.

Detailed stepwise algorithm:.

1. **For each cell (i, j) on the board** in row-major order:
    - If the cell is empty:
    - Place 'O' temporarily in the cell.
    - Call check_win(board, 'O').
    - If check_win returns true:
        - **Keep 'O' in that cell** and return from the function (immediate win found).
    - Otherwise undo the temporary move (set cell back to empty) and continue scanning.

2. **For each cell (i, j) on the board** in row-major order:
    - If the cell is empty:
    - Place 'X' temporarily in the cell.
    - Call check_win(board, 'X').
    - If check_win returns true:
        - **Place 'O' in that cell** (block the opponent) and return.
    - Otherwise undo the temporary move and continue scanning.
3. **If difficulty equals 2 (God mode)**:
    - If the center cell board[1][1] is empty:
    - Place 'O' in the center and return.
4. **Try corners in this order**: (0,0), (0,2), (2,0), (2,2).
    - For each corner, if it is empty:
    - Place 'O' there and return.
5. **Fallback**:

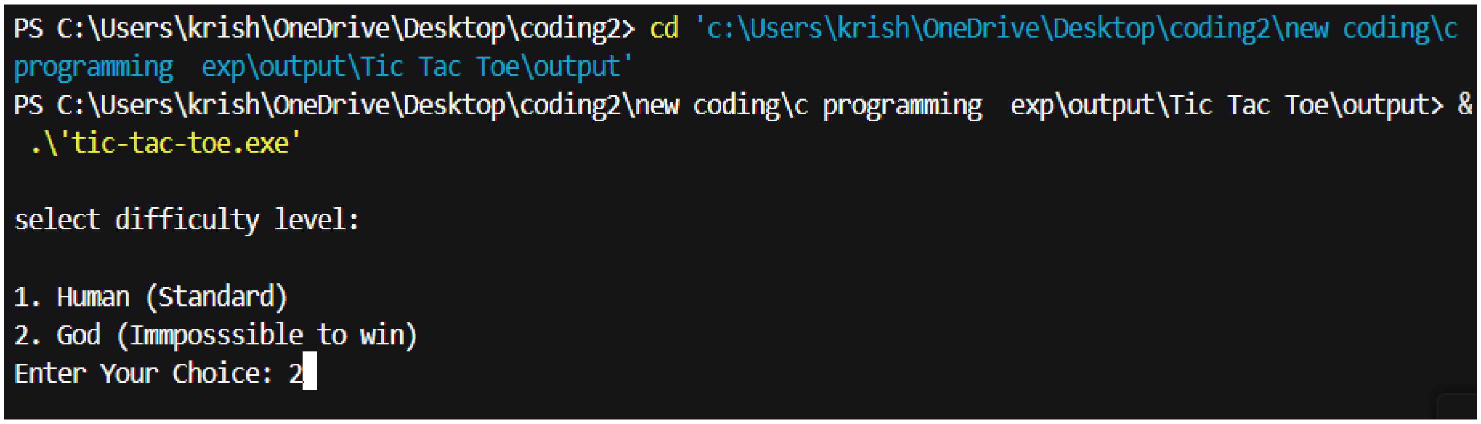
- Scan the board in row-major order and place 'O' in the first empty cell found, then return.

Algorithm for print_board():

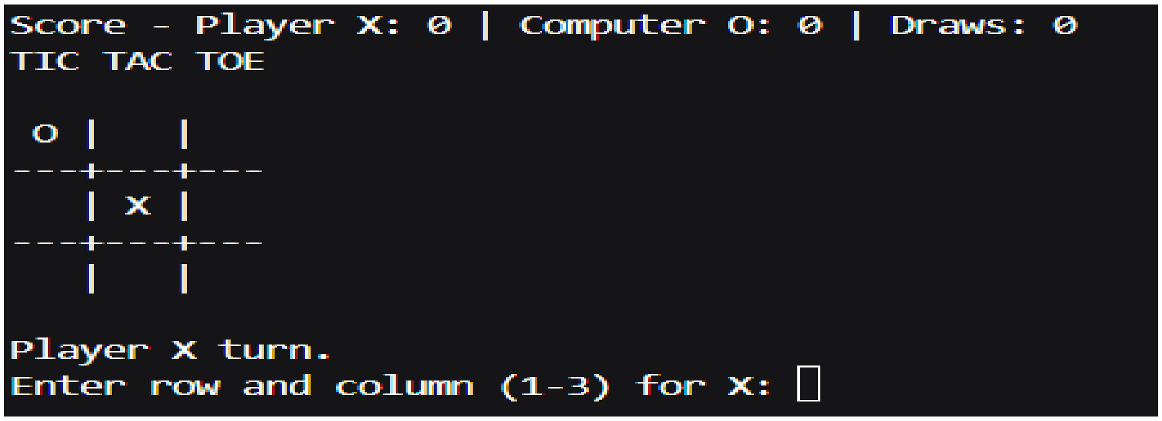
Clear screen using platform-specific command (cls on Windows, clear on Unix-like systems).

- **Print score line** showing **Player X**, **Computer O**, and **Draws** using the global score struct.
- **Print title** such as TIC TAC TOE.
- **For each row** from 0 to 2:
    - Print a newline to separate rows visually.
    - **For each column** from 0 to 2:
        - Print a space, the board cell character, and a space (" %c ").
        - If the column is not the last, print a vertical separator |.
    - After finishing the row, if the row is not the last, print a horizontal separator line ---+---+---.
- **Print trailing newlines** to separate the board from subsequent prompts or message.

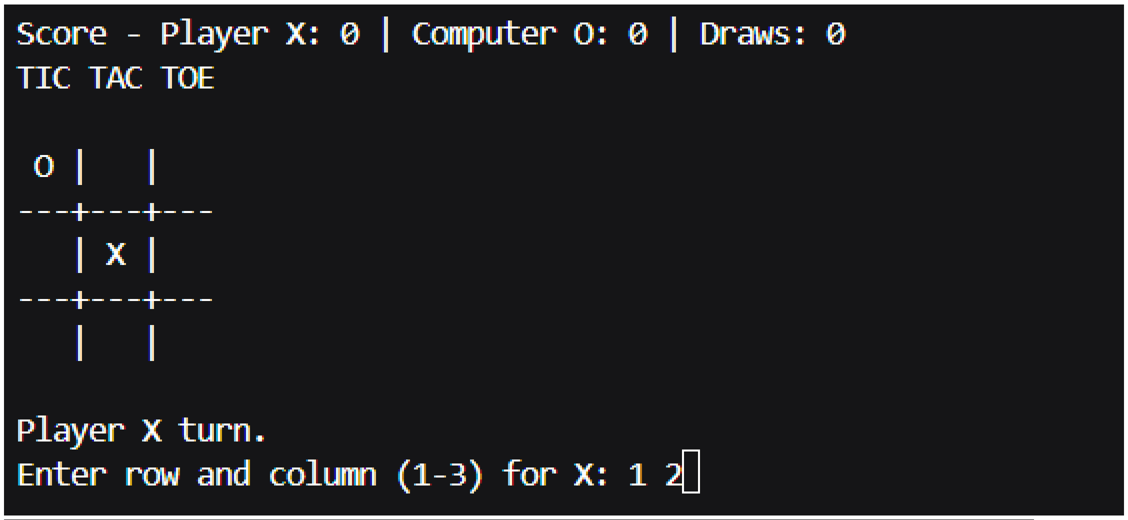
TESTING AND RESULTS:



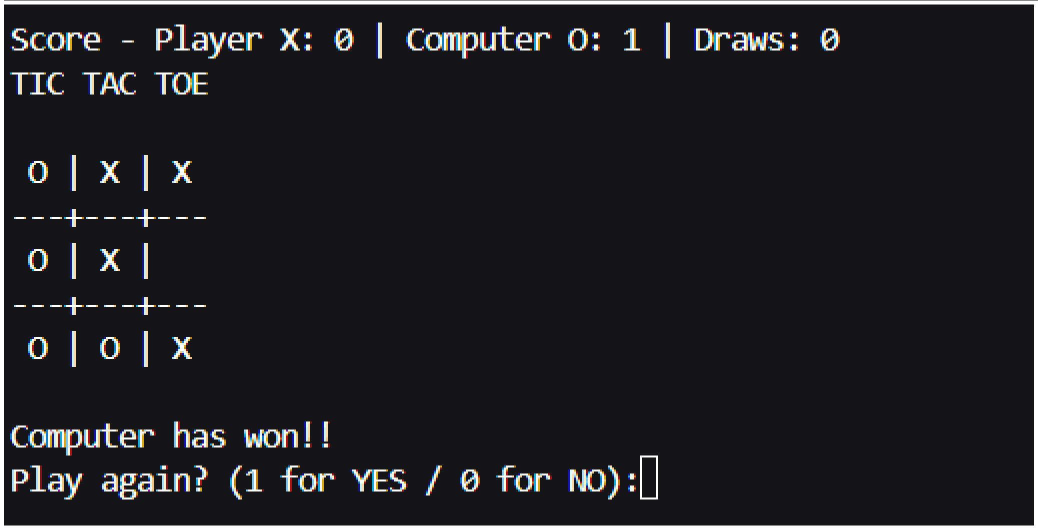
```
PS C:\Users\krish\OneDrive\Desktop\coding2> cd 'c:\Users\krish\OneDrive\Desktop\coding2\new coding\c programming  exp\output\Tic Tac Toe\output'
PS C:\Users\krish\OneDrive\Desktop\coding2\new coding\c programming  exp\output\Tic Tac Toe\output> & .\'tic-tac-toe.exe'

select difficulty level:

1. Human (Standard)
2. God (Immposssible to win)
Enter Your Choice: 2
```



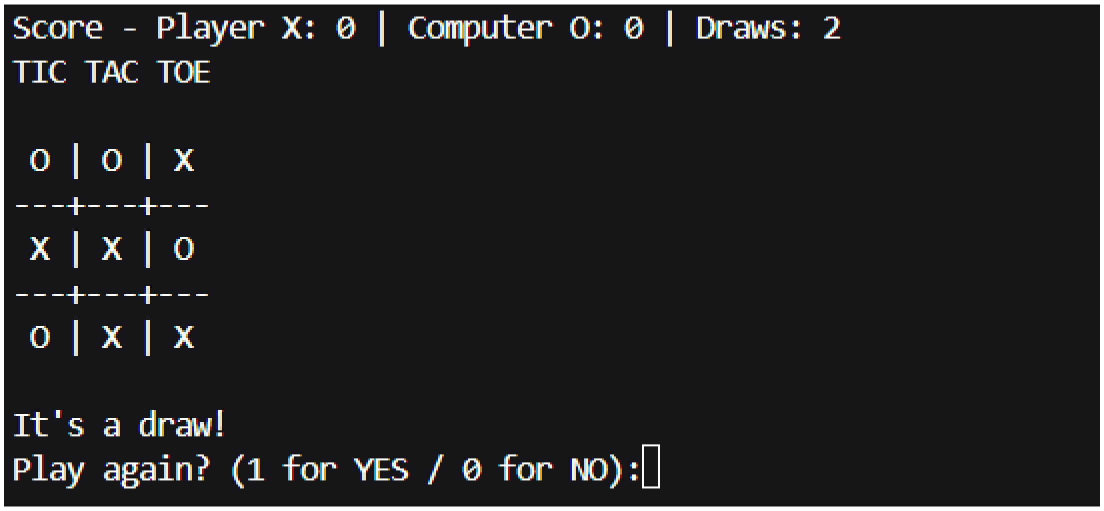
```
Score - Player X: 0 | Computer O: 0 | Draws: 0
TIC TAC TOE

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

Player X turn.
Enter row and column (1-3) for X:
```

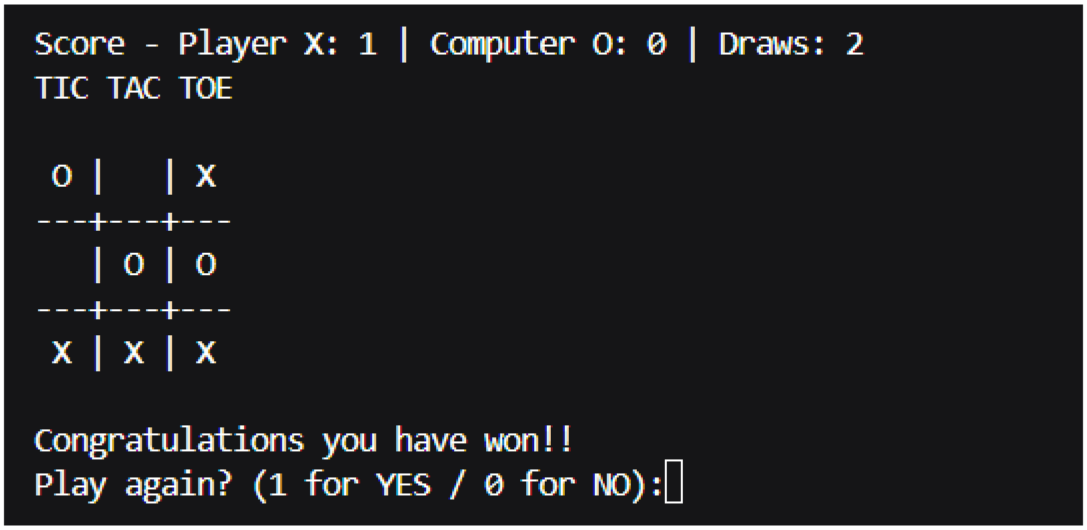


```
Score - Player X: 0 | Computer O: 0 | Draws: 0
TIC TAC TOE

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

Player X turn.
Enter row and column (1-3) for X: 1 2
```



```
Score - Player X: 0 | Computer O: 1 | Draws: 0
TIC TAC TOE

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

Computer has won!!
Play again? (1 for YES / 0 for NO):
```


```
Score - Player X: 0 | Computer O: 0 | Draws: 2
TIC TAC TOE

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

It's a draw!
Play again? (1 for YES / 0 for NO):
```


```
Score - Player X: 1 | Computer O: 0 | Draws: 2
TIC TAC TOE

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

Congratulations you have won!!
Play again? (1 for YES / 0 for NO):
```

Conclusion & Future Work

Conclusion: The project meets objectives: valid gameplay, score tracking, and two AI modes.

Future work: **implement Minimax with alpha-beta pruning for optimal play, add GUI, and support networked two-player mode**

References

- **Let Us C** – Yashavant Kanetkar
- GeeksforGeeks
- Class notes by DR. TANU SINGH
- YouTube: Complete coding by Prashant sir