TIC-TAC-TOE GAME

Prepared by

Name of the Students: ROMIT MUKHERJEE
Enrolment Number: 12022002002224
Section: B
Class Roll Number: 102
Stream: CSE
Subject: Programming for Problem Solving using C
Subject Code: ESC103(Pr)
Department: Basic Science and Humanities

Under the supervision of
Swarnendu Ghosh Sir

Academic Year: 2022-26

PROJECT REPORT SUBMITTED PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE SECOND SEMESTER

DEPARTMENT OF BASIC SCIENCE AND HUMANITIES
INSTITUTE OF ENGINEERING AND MANAGEMENT, KOLKATA

CERTIFICATE OF RECOMMENDATION

We hereby recommend that the project prepared under our supervision by ROMIT MUKHERJEE, entitled **Tic-Tac-Toe Game** be accepted in partial fulfillment of the requirements for the degree of partial fulfillment of the second semester.

Head of the Department
Sciences and Humanities IEM,
Kolkata

Project Supervisor Basic

1. Database Descriptions

1. Players:

2.Columns:

- player_id (integer, primary key): unique identifier for each player.
- name (string): name of the player.

3.Games:

4.Columns:

- game_id (integer, primary key): unique identifier for each game.
- player1_id (integer, foreign key): references the player_id column in the Players table for the first player.
- player2_id (integer, foreign key): references the player_id column in the Players table for the second player.
- winner_id (integer, foreign key): references the player_id column in the Players table for the winner of the game (nullable).
- start_time (datetime): timestamp indicating when the game started.
- end_time (datetime): timestamp indicating when the game ended (nullable).

5.Moves:

6.Columns:

- move_id (integer, primary key): unique identifier for each move.
- game_id (integer, foreign key): references the game_id column in the Games table.
- player_id (integer, foreign key): references the player_id column in the Players table for the player who made the move.
- position (integer): the position on the game board where the move was made (1-9).
- move_time (datetime): timestamp indicating when the move was made.

2. Variables

- **board:** This variable represents the game board. It can be represented as a multidimensional array or a single-dimensional array, depending on your implementation.

- **player:** This variable keeps track of the current player. It can be represented using an enumeration, where each player is assigned a specific value (e.g., Player 1 or Player 2).
- **currentMove:** This variable stores the current move or position chosen by the player. It is typically an integer representing the index or position on the game board.
- **gameOver:** This variable is a flag that indicates whether the game is over or not. It is often a boolean value (0 for not over, 1 for game over).
- **winner:** This variable stores the winner of the game. It can be represented using an enumeration, where each possible outcome (Player 1 wins, Player 2 wins, or a tie) is assigned a specific value.

3. Function

- **drawBoard:** This function is responsible for drawing the Tic-Tac-Toe board on the screen, showing the current state of the game.
- **initializeBoard:** This function initializes the game board, setting all cells to empty or a default value.
- **getPlayerMove:** This function prompts the player to enter their move (row and column) and validates the input.
- **makeMove:** This function updates the game board with the player's move, placing their symbol (X or O) in the specified cell.
- **checkWin:** This function checks if a player has won the game by examining the current state of the board. It checks for winning conditions such as three in a row, column, or diagonal.
- **checkDraw:** This function checks if the game has ended in a draw, meaning all cells on the board are filled, and no player has won
- **switchPlayer:** This function switches the active player between turns (from player X to player O, or vice versa).
- **playAgain:** This function asks the players if they want to play another round and returns the user's choice.

- **main:** The main function controls the flow of the game. It typically calls the other functions and handles the game loop, alternating between player turns until there is a winner or a draw.

4. Features

- **Game Board:** Create a 3x3 grid to represent the game board. You can use a two-dimensional array to store the state of each cell (e.g., empty, X, or O).
- **Player Input:** Allow players to input their moves by specifying the row and column of the cell they want to mark.
- **Alternating Turns:** Implement a mechanism to alternate between players' turns (Player 1: X and Player 2: O).
- **Win Condition:** Check for a win condition after each move. A player wins if they have three marks (X or O) in a row (horizontally, vertically, or diagonally).
- **Draw Condition:** Detect when the game ends in a draw, which occurs when all cells are filled, and no player has won.
- **Game Loop:** Create a loop that continues until a player wins or the game ends in a draw. Prompt the players for their moves in each iteration.
- **User Interface:** Create a simple text-based interface to interact with the players and display the game.

5. Program

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

#define SIZE 3

void displayBoard(char board[][SIZE]);
int checkWinner(char board[][SIZE], char player);

int main()
{
   char board[SIZE][SIZE] = {{' ', ' ', ' '}, {' ', ' ', ' '}, {' ', ' ', ' '}};
   int row, col, moveCount = 0;
   char currentPlayer = 'X';
   printf("Welcome to Tic-Tac-Toe!\n");
```

```
    displayBoard(board);
    while (moveCount < SIZE*SIZE) {
        printf("\nPlayer %c's turn. Enter row and column number to place your move: ", currentPlayer);
        scanf("%d %d", &row, &col);
        if (row >= 0 && row < SIZE && col >= 0 && col < SIZE && board[row][col] == ' ') {
            board[row][col] = currentPlayer;
            displayBoard(board);
            if (checkWinner(board, currentPlayer)) {
                printf("Player %c wins!\n", currentPlayer);
                return 0;
            }
            currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
            moveCount++;
        } else {
            printf("Invalid move. Please try again.\n");
        }
    }
    printf("Game over. It's a draw.\n");
    return 0;
}
void displayBoard(char board[][SIZE])
{
    printf("\n");
    for (int i = 0; i < SIZE; i++) {
        printf(" %c | %c | %c ", board[i][0], board[i][1], board[i][2]);
        if (i != SIZE - 1) {
            printf("\n---|---|---\n");
        }
    }
    printf("\n");
}
int checkWinner(char board[][SIZE], char player)
{
    // Check rows
    for (int i = 0; i < SIZE; i++) {
        if (board[i][0] == player && board[i][1] == player && board[i][2] == player) {
            return 1;
        }
    }
    // Check columns
    for (int j = 0; j < SIZE; j++) {
        if (board[0][j] == player && board[1][j] == player && board[2][j] == player) {
            return 1;
        }
    }
    // Check diagonals
    if (board[0][0] == player && board[1][1] == player && board[2][2] == player) {
        return 1;
    }
    if (board[0][2] == player && board[1][1] == player && board[2][0] == player) {
        return 1;
    }
    return 0;
}
```