**Name : Tanishq Thuse**

**Branch : CS(AI)**

**Year : SY**

**Div : B**

**Roll no. : 60**

**PRN : 12310237**

**Title : AI Assignment-1 – Tic Tac Toe with and without using AI**

**Q1) Tic Tac Toe without using AI**

**Code :**

*/\*Logic :*

*1) Print board initially in printBoard() function*

*2) Take turn of player in takeTurn() function*

*3) Check if game is over in checkGameOver() function*

*4) If game is over, print the result and end the game*

*5) If game is not over, switch to the other player and continue playing*

*6) Repeat steps 2-5 until the game is over*

*7) Print the result of the game (win/tie) after the game is over*

*8) End the game (exit with code 0) after printing the result (line 5*

*9) Run the game (line 1*

*10) Print the board (line*

*11) Start the game with player X (line 1*

*\*/*

import *java.util.Scanner*;

*public* *class* game {

*// Set up the game board as an array*

*static* String[] board = {"-", "-", "-", "-", "-", "-", "-", "-", "-"};

*// Define a function to print the game board*

*static* void printBoard() {

        System.out.println(board[0] + " | " + board[1] + " | " + board[2]);

        System.out.println(board[3] + " | " + board[4] + " | " + board[5]);

        System.out.println(board[6] + " | " + board[7] + " | " + board[8]);

    }

*// Define a function to handle a player's turn*

*static* void takeTurn(String player) {

        Scanner scanner = new Scanner(System.in);

        System.out.println(player + "'s turn.");

        System.out.print("Choose a position from 1-9: ");

        int position = scanner.nextInt() - 1;

        while (position < 0 || position > 8 || !board[position].equals("-")) {

            System.out.print("Invalid input or position already taken. Choose a different position: ");

            position = scanner.nextInt() - 1;

        }

        board[position] = player;

        printBoard();

    }

*// Define a function to check if the game is over*

*static* String checkGameOver() {

*// Check for a win*

        if ((board[0].equals(board[1]) && board[1].equals(board[2]) && !board[0].equals("-")) ||

                (board[3].equals(board[4]) && board[4].equals(board[5]) && !board[3].equals("-")) ||

                (board[6].equals(board[7]) && board[7].equals(board[8]) && !board[6].equals("-")) ||

                (board[0].equals(board[3]) && board[3].equals(board[6]) && !board[0].equals("-")) ||

                (board[1].equals(board[4]) && board[4].equals(board[7]) && !board[1].equals("-")) ||

                (board[2].equals(board[5]) && board[5].equals(board[8]) && !board[2].equals("-")) ||

                (board[0].equals(board[4]) && board[4].equals(board[8]) && !board[0].equals("-")) ||

                (board[2].equals(board[4]) && board[4].equals(board[6]) && !board[2].equals("-"))) {

            return "win";

        }

*// Check for a tie*

        else if (!String.join("", board).contains("-")) {

            return "tie";

        }

*// Game is not over*

        else {

            return "play";

        }

    }

*// Define the main game loop*

*public* *static* void main(String[] args) {

        printBoard();

        String currentPlayer = "X";

        boolean gameOver = false;

        while (!gameOver) {

            takeTurn(currentPlayer);

            String gameResult = checkGameOver();

            if (gameResult.equals("win")) {

                System.out.println(currentPlayer + " wins!");

                gameOver = true;

            } else if (gameResult.equals("tie")) {

                System.out.println("It's a tie!");

                gameOver = true;

            } else {

*// Switch to the other player*

                currentPlayer = currentPlayer.equals("X") ? "O" : "X";

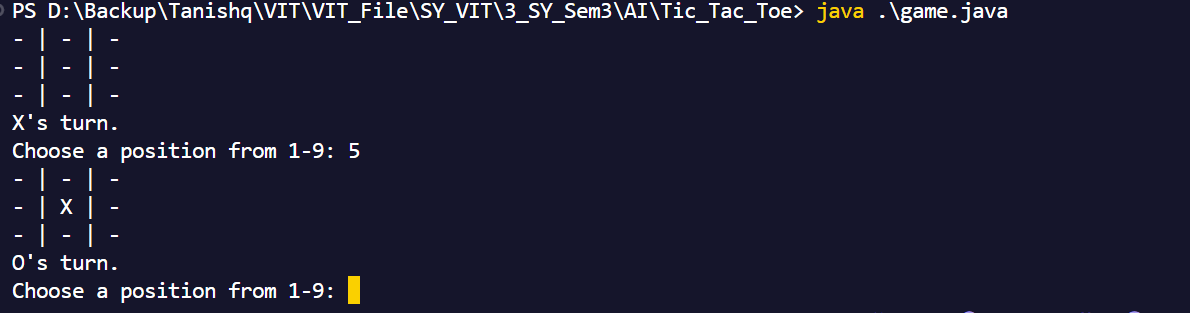
            }

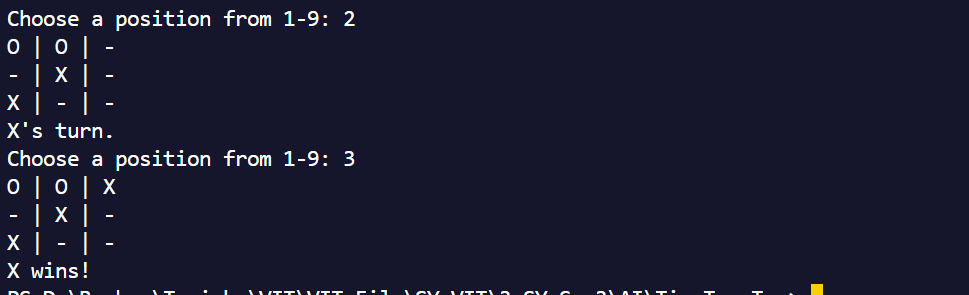
        }

    }

}

**Output :**





**Q2) Tic Tac Toe using AI**

**Code :**

import *java.util.Scanner*;

*/\* Logic:*

*1. Print the initial board using printBoard() function.*

*2. Check if there's a winner or the board is full using checkWinner() and isBoardFull() functions.*

*3. If the game is over, print the result (win/tie) and end the game.*

*4. If the game is not over, switch to the other player.*

*5. Repeat steps 2-4 until the game ends.*

*6. Print the final result of the game (win/tie) after the game is over.*

*7. End the game (exit with code 0) after printing the result.*

*8. Run the game starting with player X.*

*9. Print the board initially and after each turn.*

*10. Handle player input and AI moves until the game concludes.*

*\*/*

*public* *class* TicTacToe {

*static* char[][] board = {

        {' ', ' ', ' '},

        {' ', ' ', ' '},

        {' ', ' ', ' '}

    };

*public* *static* void printBoard() {

        for (int i = 0; i < 3; i++) {

            for (int j = 0; j < 3; j++) {

                System.out.print(board[i][j]);

                if (j < 2) System.out.print(" | ");

            }

            System.out.println();

            if (i < 2) System.out.println("---------");

        }

    }

*public* *static* boolean checkWinner(char player) {

        for (int i = 0; i < 3; i++) {

            if (board[i][0] == player && board[i][1] == player && board[i][2] == player) return true;

            if (board[0][i] == player && board[1][i] == player && board[2][i] == player) return true;

        }

        if (board[0][0] == player && board[1][1] == player && board[2][2] == player) return true;

        if (board[0][2] == player && board[1][1] == player && board[2][0] == player) return true;

        return false;

    }

*public* *static* boolean isBoardFull() {

        for (int i = 0; i < 3; i++) {

            for (int j = 0; j < 3; j++) {

                if (board[i][j] == ' ') return false;

            }

        }

        return true;

    }

*public* *static* int[] minimax(char[][] board, int depth, boolean isMaximizing) {

        if (checkWinner('O')) return new int[]{1, -1, -1};

        if (checkWinner('X')) return new int[]{-1, -1, -1};

        if (isBoardFull()) return new int[]{0, -1, -1};

        int bestScore = isMaximizing ? Integer.MIN\_VALUE : Integer.MAX\_VALUE;

        int[] bestMove = new int[]{-1, -1, -1};

        for (int i = 0; i < 3; i++) {

            for (int j = 0; j < 3; j++) {

                if (board[i][j] == ' ') {

                    board[i][j] = isMaximizing ? 'O' : 'X';

                    int[] currentScore = minimax(board, depth + 1, !isMaximizing);

                    board[i][j] = ' ';

                    currentScore[0] += isMaximizing ? -depth : depth;

                    if (isMaximizing && currentScore[0] > bestScore) {

                        bestScore = currentScore[0];

                        bestMove = new int[]{bestScore, i, j};

                    } else if (!isMaximizing && currentScore[0] < bestScore) {

                        bestScore = currentScore[0];

                        bestMove = new int[]{bestScore, i, j};

                    }

                }

            }

        }

        return bestMove;

    }

*public* *static* void aiMove() {

        int[] result = minimax(board, 0, true);

        if (result[1] != -1 && result[2] != -1) {

            board[result[1]][result[2]] = 'O';

        }

    }

*public* *static* void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Welcome to Tic-Tac-Toe!");

        while (true) {

            printBoard();

            if (checkWinner('X')) {

*// System.out.println("Player X wins!");*

                System.out.println("Human wins!");

                break;

            }

            if (checkWinner('O')) {

*// System.out.println("Player O wins!");*

                System.out.println("AI wins!");

                break;

            }

            if (isBoardFull()) {

                System.out.println("It's a tie!");

                break;

            }

            System.out.println("Enter your move (row and column): ");

            int row = scanner.nextInt();

            int col = scanner.nextInt();

            if (row < 0 || row >= 3 || col < 0 || col >= 3 || board[row][col] != ' ') {

                System.out.println("Invalid move. Try again.");

                continue;

            }

            board[row][col] = 'X';

            if (!checkWinner('X') && !isBoardFull()) {

                aiMove();

            }

        }

        printBoard();

        scanner.close();

    }

}

Output :

