Name: Saurav Gujar

Class : TY CS D

Roll No: 14

Topic : Code of Tic Tac Toe using AI

Code :

import java.util.Scanner;

public class TicTacToeAI {

    private static final char EMPTY = ' ';

    private static final char PLAYER\_X = 'X';

    private static final char PLAYER\_O = 'O';

    public static void main(String[] args) {

        char[][] board = {

                {EMPTY, EMPTY, EMPTY},

                {EMPTY, EMPTY, EMPTY},

                {EMPTY, EMPTY, EMPTY}

        };

        printBoard(board);

        while (!isGameOver(board)) {

            playerMove(board);

            if (isGameOver(board)) {

                break;

            }

            AIMove(board);

            printBoard(board);

        }

        char winner = checkWinner(board);

        if (winner == PLAYER\_X) {

            System.out.println("Congratulations! You win!");

        } else if (winner == PLAYER\_O) {

            System.out.println("AI wins! Better luck next time.");

        } else {

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

        }

    }

    private static void playerMove(char[][] board) {

        Scanner scanner = new Scanner(System.in);

        int row, col;

        do {

            System.out.print("Enter your move (row [1-3] and column [1-3]): ");

            row = scanner.nextInt() - 1;

            col = scanner.nextInt() - 1;

        } while (!isValidMove(board, row, col));

        board[row][col] = PLAYER\_X;

    }

    private static void AIMove(char[][] board) {

        int[] bestMove = getBestMove(board);

        board[bestMove[0]][bestMove[1]] = PLAYER\_O;

        System.out.println("AI chooses row: " + (bestMove[0] + 1) + ", column: " + (bestMove[1] + 1));

    }

    private static int[] getBestMove(char[][] board) {

        int[] bestMove = {-1, -1};

        int bestScore = Integer.MIN\_VALUE;

        for (int i = 0; i < 3; i++) {

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

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

                    board[i][j] = PLAYER\_O;

                    int score = minimax(board, 0, false);

                    board[i][j] = EMPTY;

                    if (score > bestScore) {

                        bestScore = score;

                        bestMove[0] = i;

                        bestMove[1] = j;

                    }

                }

            }

        }

        return bestMove;

    }

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

        if (checkWinner(board) == PLAYER\_X) {

            return -10 + depth;

        }

        if (checkWinner(board) == PLAYER\_O) {

            return 10 - depth;

        }

        if (isBoardFull(board)) {

            return 0;

        }

        int bestScore;

        if (isMaximizing) {

            bestScore = Integer.MIN\_VALUE;

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

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

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

                        board[i][j] = PLAYER\_O;

                        int score = minimax(board, depth + 1, false);

                        board[i][j] = EMPTY;

                        bestScore = Math.max(bestScore, score);

                    }

                }

            }

        } else {

            bestScore = Integer.MAX\_VALUE;

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

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

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

                        board[i][j] = PLAYER\_X;

                        int score = minimax(board, depth + 1, true);

                        board[i][j] = EMPTY;

                        bestScore = Math.min(bestScore, score);

                    }

                }

            }

        }

        return bestScore;

    }

    private static boolean isValidMove(char[][] board, int row, int col) {

        if (row < 0 || row >= 3 || col < 0 || col >= 3) {

            return false;

        }

        return board[row][col] == EMPTY;

    }

    private static boolean isBoardFull(char[][] board) {

        for (int i = 0; i < 3; i++) {

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

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

                    return false;

                }

            }

        }

        return true;

    }

    private static boolean isGameOver(char[][] board) {

        return isBoardFull(board) || checkWinner(board) != EMPTY;

    }

    private static char checkWinner(char[][] board) {

        // Check rows

        for (int i = 0; i < 3; i++) {

            if (board[i][0] != EMPTY && board[i][0] == board[i][1] && board[i][1] == board[i][2]) {

                return board[i][0];

            }

        }

        // Check columns

        for (int j = 0; j < 3; j++) {

            if (board[0][j] != EMPTY && board[0][j] == board[1][j] && board[1][j] == board[2][j]) {

                return board[0][j];

            }

        }

        // Check diagonals

        if (board[0][0] != EMPTY && board[0][0] == board[1][1] && board[1][1] == board[2][2]) {

            return board[0][0];

        }

        if (board[0][2] != EMPTY && board[0][2] == board[1][1] && board[1][1] == board[2][0]) {

            return board[0][2];

        }

        return EMPTY;

    }

    private static void printBoard(char[][] board) {

        System.out.println("-------------");

        for (int i = 0; i < 3; i++) {

            System.out.print("| ");

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

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

            }

            System.out.println();

            System.out.println("-------------");

        }

    }

}