AI Lab-1 Gaurav Mishra – 9557 Batch B

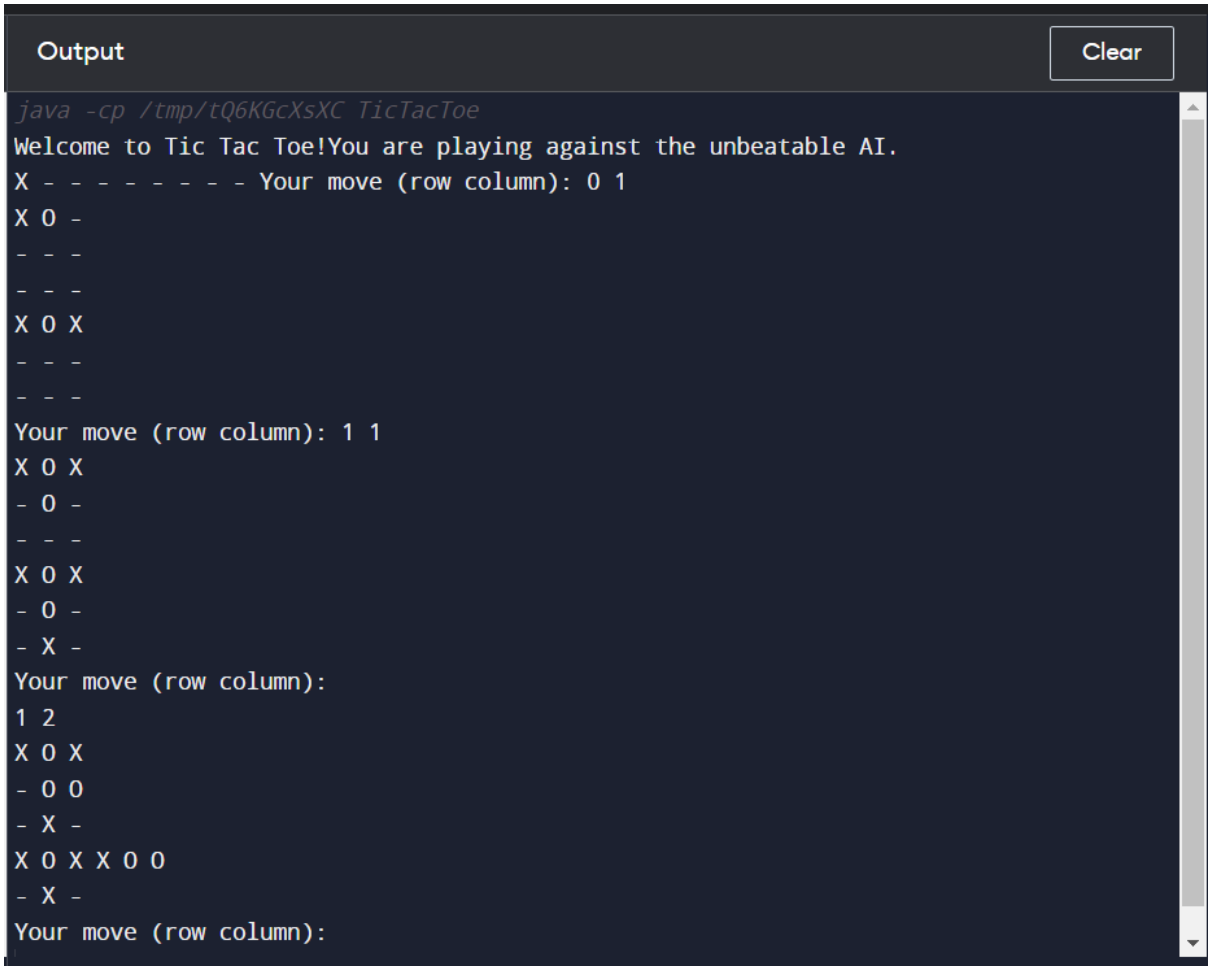
1. Tic tac toe by Brute Force Method :

```
import java.util.Scanner;

public class TicTacToe {
    private static final char EMPTY = '-';
    private static final char PLAYER_X = 'X';
    private static final char PLAYER_O = 'O';
    private static final int BOARD_SIZE = 3;

    private char[][] board;
    private char currentPlayer;

    public TicTacToe() {
        board = new char[BOARD_SIZE][BOARD_SIZE];
        currentPlayer = PLAYER_X;
        initializeBoard();
    }

    private void initializeBoard() {
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                board[i][j] = EMPTY;
            }
        }
    }

    private void printBoard() {
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                System.out.print(board[i][j] + " ");
            }
            System.out.println();
        }
    }

    private boolean isBoardFull() {
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                if (board[i][j] == EMPTY) {
                    return false;
                }
            }
```

```
        }
        return true;
    }

    private boolean hasWon(char player) {
        // Check rows and columns
        for (int i = 0; i < BOARD_SIZE; i++) {
            if ((board[i][0] == player && board[i][1] == player && board[i][2] == player) ||
                (board[0][i] == player && board[1][i] == player && board[2][i] == player)) {
                return true;
            }
        }
        // Check diagonals
        return (board[0][0] == player && board[1][1] == player && board[2][2] == player) ||
               (board[0][2] == player && board[1][1] == player && board[2][0] == player);
    }

    private boolean isValidMove(int row, int col) {
        return row >= 0 && row < BOARD_SIZE && col >= 0 && col < BOARD_SIZE &&
board[row][col] == EMPTY;
    }

    private void makeMove(int row, int col, char player) {
        board[row][col] = player;
    }

    private void switchPlayer() {
        currentPlayer = (currentPlayer == PLAYER_X) ? PLAYER_O : PLAYER_X;
    }

    private int evaluate() {
        if (hasWon(PLAYER_X)) {
            return 1; // AI wins
        } else if (hasWon(PLAYER_O)) {
            return -1; // Human wins
        } else {
            return 0; // Draw
        }
    }

    private int minimax(int depth, boolean isMaximizing) {
        int score = evaluate();

        // Base case: terminal state reached
        if (score != 0) {
            return score;
        }
```

```
    // If it's AI's turn
    if (isMaximizing) {
        int bestScore = Integer.MIN_VALUE;
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                if (board[i][j] == EMPTY) {
                    board[i][j] = PLAYER_X;
                    int currentScore = minimax(depth + 1, false);
                    board[i][j] = EMPTY;
                    bestScore = Math.max(bestScore, currentScore);
                }
            }
        }
        return bestScore;
    } else { // If it's human's turn
        int bestScore = Integer.MAX_VALUE;
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                if (board[i][j] == EMPTY) {
                    board[i][j] = PLAYER_O;
                    int currentScore = minimax(depth + 1, true);
                    board[i][j] = EMPTY;
                    bestScore = Math.min(bestScore, currentScore);
                }
            }
        }
        return bestScore;
    }
}

private void aiMove() {
    int bestScore = Integer.MIN_VALUE;
    int bestRow = -1;
    int bestCol = -1;

    // Find the best move
    for (int i = 0; i < BOARD_SIZE; i++) {
        for (int j = 0; j < BOARD_SIZE; j++) {
            if (board[i][j] == EMPTY) {
                board[i][j] = PLAYER_X;
                int currentScore = minimax(0, false);
                board[i][j] = EMPTY;
                if (currentScore > bestScore) {
                    bestScore = currentScore;
                    bestRow = i;
                    bestCol = j;
                }
            }
```

```
            }
        }

        // Make the best move
        makeMove(bestRow, bestCol, PLAYER_X);
    }

    public void play() {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Welcome to Tic Tac Toe!");
        System.out.println("You are playing against the unbeatable AI.");

        while (!isBoardFull() && !hasWon(PLAYER_X) && !hasWon(PLAYER_O)) {
            if (currentPlayer == PLAYER_X) {
                aiMove();
            } else {
                System.out.println("Your move (row column): ");
                int row = scanner.nextInt();
                int col = scanner.nextInt();
                if (!isValidMove(row, col)) {
                    System.out.println("Invalid move! Try again.");
                    continue;
                }
                makeMove(row, col, PLAYER_O);
            }
            printBoard();
            switchPlayer();
        }

        if (hasWon(PLAYER_X)) {
            System.out.println("AI wins! Better luck next time.");
        } else if (hasWon(PLAYER_O)) {
            System.out.println("Congratulations! You win!");
        } else {
            System.out.println("It's a draw!");
        }

        scanner.close();
    }

    public static void main(String[] args) {
        TicTacToe game = new TicTacToe();
        game.play();
    }
}
```



2. Heuristic approach

```
import java.util.Scanner;

public class TicTacToeHeuristic {

    private static final char EMPTY = '-';
    private static final char X = 'X';
    private static final char O = 'O';

    private char[][] board;
    private char currentPlayer;

    public TicTacToeHeuristic() {
        board = new char[3][3];
        currentPlayer = X;
        initializeBoard();
    }

    private void initializeBoard() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                board[i][j] = EMPTY;
```

```
        }
      }
    }

    public void printBoard() {
      for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
          System.out.print(board[i][j] + " ");
        }
        System.out.println();
      }
    }

    public boolean makeMove(int row, int col) {
      if (row < 0 || row >= 3 || col < 0 || col >= 3 || board[row][col] != EMPTY) {
        return false;
      }
      board[row][col] = currentPlayer;
      currentPlayer = (currentPlayer == X) ? O : X;
      return true;
    }

    public char checkWinner() {
      for (int i = 0; i < 3; i++) {
        if (board[i][0] != EMPTY && board[i][0] == board[i][1] && board[i][0] ==
board[i][2]) {
          return board[i][0];
        }
        if (board[0][i] != EMPTY && board[0][i] == board[1][i] && board[0][i] ==
board[2][i]) {
          return board[0][i];
        }
      }
      if (board[0][0] != EMPTY && board[0][0] == board[1][1] && board[0][0] ==
board[2][2]) {
        return board[0][0];
      }
      if (board[0][2] != EMPTY && board[0][2] == board[1][1] && board[0][2] ==
board[2][0]) {
        return board[0][2];
      }
      return EMPTY;
    }

    public boolean isBoardFull() {
      for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
          if (board[i][j] == EMPTY) {
```

```
            return false;
          }
        }
      }
    return true;
}

public boolean isGameOver() {
    return checkWinner() != EMPTY || isBoardFull();
}

public void play() {
    Scanner scanner = new Scanner(System.in);

    while (!isGameOver()) {
        System.out.println("Current Board:");
        printBoard();
        System.out.println("Player " + currentPlayer + "'s turn.");

        if (currentPlayer == X) {
            System.out.print("Enter row (0-2): ");
            int row = scanner.nextInt();
            System.out.print("Enter column (0-2): ");
            int col = scanner.nextInt();
            if (!makeMove(row, col)) {
                System.out.println("Invalid move. Try again.");
            }
        } else {
            makeAIMove();
        }

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

    System.out.println("Final Board:");
    printBoard();
    char winner = checkWinner();
    if (winner == EMPTY) {
        System.out.println("It's a draw!");
    } else {
        System.out.println("Player " + winner + " wins!");
    }
}

public void makeAIMove() {
    // Simple AI strategy: choose the first available empty cell
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
```

```
                if (board[i][j] == EMPTY) {
                    board[i][j] = currentPlayer;
                    return;
                }
            }
        }
    }

    public static void main(String[] args) {
        TicTacToeHeuristic game = new TicTacToeHeuristic();
        game.play();
    }
}
```

Output

Clear

```
java -cp /tmp/tQ6KGcXsXC TicTacToe
Welcome to Tic Tac Toe!You are playing against the unbeatable AI.
X - - - - - - - - Your move (row column): 0 1
X O -
- - -
- - -
X O X
- - -
- - -
Your move (row column): 1 1
X O X
- O -
- - -
X O X
- O -
- X -
Your move (row column):
1 2
X O X
- O O
- X -
X O X X O O
- X -
Your move (row column):
```