**TASK 2**

**TIC-TAC-TOE AI**

Implement an AI agent that plays the classic game of Tic-Tac-Toe

against a human player. You can use algorithms like Minimax with

or without Alpha-Beta Pruning to make the AI player unbeatable.

This project will help you understand game theory and basic search

algorithms.

CODE:-

import java.util.Scanner;

public class TicTacToe

{

static char[][] board = {

{' ',' ',' '},

{' ',' ',' '},

{' ',' ',' '}

};

public static void main(String[] args)

{

Scanner scanner=new Scanner(System.in);

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

printBoard();

while(true)

{

System.out.println("Your move (enter row and column): ");

int row=scanner.nextInt();

int col=scanner.nextInt();

if(board[row][col]==' ')

{

board[row][col]='X';

if(isGameOver())

{

printBoard();

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

break;

}

}

else

{

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

continue;

}

printBoard();

System.out.println("AI's move:");

Move bestMove=findBestMove();

board[bestMove.row][bestMove.col]='O';

printBoard();

if(isGameOver())

{

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

break;

}

if(isDraw())

{

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

break;

}

}

scanner.close();

}

public static void printBoard()

{

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("---------");

}

public static boolean isGameOver()

{

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

{

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

return true;

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

return true;

}

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

return true;

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

return true;

return false;

}

public static boolean isDraw()

{

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 evaluate()

{

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

{

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

{

if(board[i][0]=='O')

return +10;

else if(board[i][0]=='X')

return -10;

}

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

{

if(board[0][i]=='O')

return +10;

else if(board[0][i]=='X')

return -10;

}

}

if(board[0][0]==board[1][1]&&board[1][1]==board[2][2])

{

if(board[0][0]=='O')

return +10;

else if(board[0][0]=='X')

return -10;

}

if(board[0][2]==board[1][1]&&board[1][1]==board[2][0])

{

if(board[0][2]=='O')

return +10;

else if(board[0][2]=='X')

return -10;

}

return 0;

}

public static int minimax(int depth, boolean isMax)

{

int score=evaluate();

if(score==10)

return score - depth;

if(score==-10)

return score + depth;

if(isDraw())

return 0;

if(isMax)

{

int best=Integer.MIN\_VALUE;

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

{

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

{

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

{

board[i][j]='O';

best=Math.max(best,minimax(depth+1,false));

board[i][j]=' ';

}

}

}

return best;

}

else

{

int best=Integer.MAX\_VALUE;

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

{

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

{

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

{

board[i][j]='X';

best=Math.min(best,minimax(depth+1,true));

board[i][j]=' ';

}

}

}

return best;

}

}

public static Move findBestMove()

{

int bestVal=Integer.MIN\_VALUE;

Move bestMove=new Move(-1, -1);

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

{

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

{

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

{

board[i][j]='O';

int moveVal=minimax(0,false);

board[i][j]=' ';

if(moveVal>bestVal)

{

bestMove.row=i;

bestMove.col=j;

bestVal=moveVal;

}

}

}

}

return bestMove;

}

static class Move

{

int row,col;

Move(int r,int c)

{

row=r;

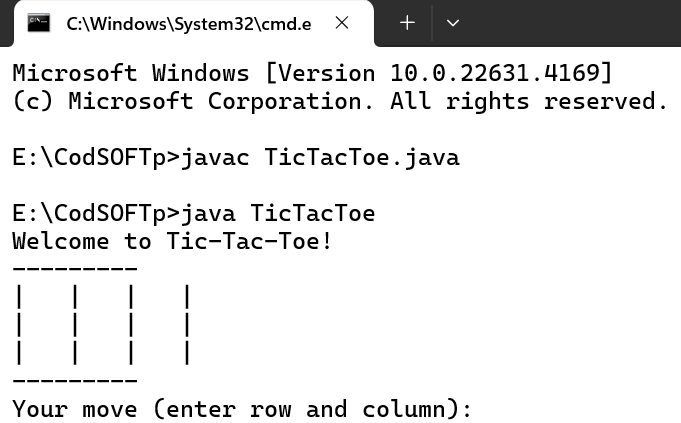
col=c;

}

}

}

**OUTPUT:-**

****