SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF NETWORKING & COMMUNICATIONS

18CSC305J-ARTIFICIAL INTELLIGENCE

SEMESTER – 6

BATCH-1

REGISTRATION NUMBER	**RA1911003011013**
NAME	**Shivank Arora**

B.Tech- CSE / CC, Third Year (Section: H2)

Faculty In charge: Dr. S. Prabakeran, B.Tech, M.E, PhD
Assistant Professor
School of Computing - Department of Networking and Communications

Year 2021-2022 / Even Semester

INDEX

Ex No	DATE	Title	Page No	Marks
1b	14/01/2022	Toy Problem: Tic Tac Toe		

Exercise: 1

Date : 21-01-2021

TOY PROBLEM

Problem Statement :

Two players, named ‘player1’ and ‘player2’, play a tic-tac-toe game on a grid of size ‘3 x 3’. Given an array ‘moves’ of size ‘n’, where each element of the array is a tuple of the form (row, column) representing a position on the grid. Players place their characters alternatively in the sequence of positions given in ‘moves’. Consider that ‘player1’ makes the first move. Your task is to return the winner of the game, i.e., the winning player’s name. If there is no winner and some positions remain unmarked, return ‘uncertain’. Otherwise, the game ends in a draw, i.e., when all positions are marked without any winner, return ‘draw’.

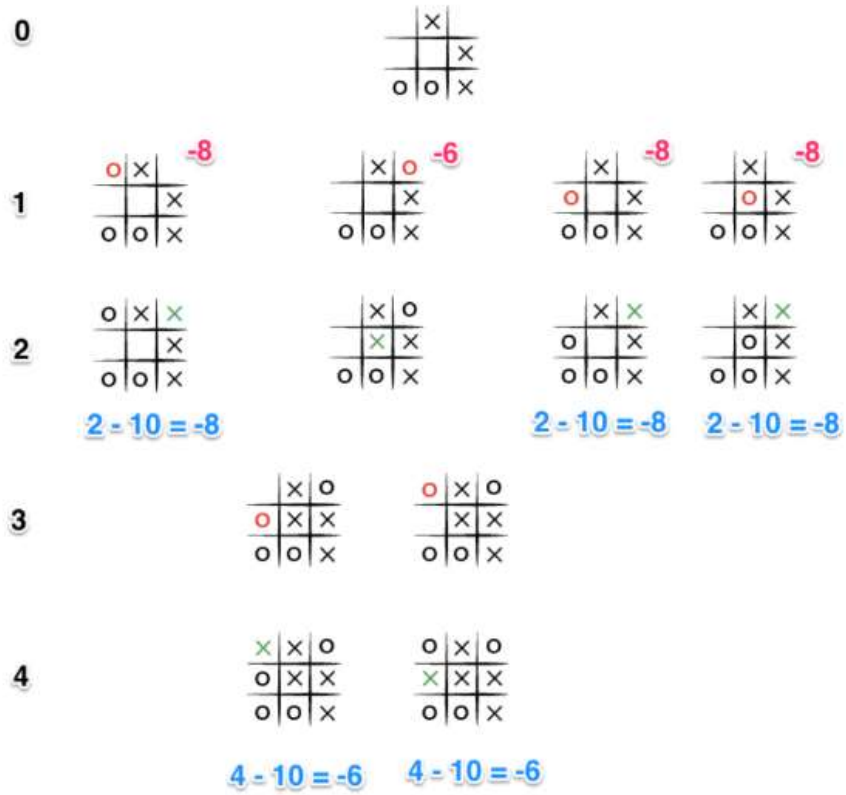
Algorithm :

- The game is to be played between two people (in this program between HUMAN and COMPUTER).
- One of the player chooses ‘O’ and the other ‘X’ to mark their respective cells.
- The game starts with one of the players and the game ends when one of the players has one whole row/ column/ diagonal filled with his/her respective character (‘O’ or ‘X’).
- If no one wins, then the game is said to be draw.

Optimization technique : The key is to use Minimax algorithm . A back and forth between the two players, where the player whose "turn it is" desires to pick the move with the maximum score. In turn, the scores for each of the available moves are determined by the opposing player deciding which of its available moves has the minimum score. And the scores for the opposing players moves are again determined by the turn-taking player trying to maximize its score and so on all the way down the move tree to an end state.

A description for the algorithm, assuming X is the "turn taking player,"

- If the game is over, return the score from X's perspective.
- Otherwise get a list of new game states for every possible move
- Create a scores list
- For each of these states add the minimax result of that state to the scores list
- If it's X's turn, return the maximum score from the scores list
- If it's O's turn, return the minimum score from the scores list



Tool : Jupyter Notebook and Python 3.9.0

Programming code :

```
player, opponent = 'x', 'o'

def isMovesLeft(board) :

    for i in range(3) :
        for j in range(3) :
            if (board[i][j] == '_') :
                return True
    return False

def evaluate(b) :


    for row in range(3) :
        if (b[row][0] == b[row][1] and b[row][1] == b[row][2]) :
            if (b[row][0] == player) :
```

```
            return 10
        elif (b[row][0] == opponent) :
            return -10

    for col in range(3) :

        if (b[0][col] == b[1][col] and b[1][col] == b[2][col]) :

            if (b[0][col] == player) :
                return 10
            elif (b[0][col] == opponent) :
                return -10

    # Checking for Diagonals for X or O victory.
    if (b[0][0] == b[1][1] and b[1][1] == b[2][2]) :

        if (b[0][0] == player) :
            return 10
        elif (b[0][0] == opponent) :
            return -10

    if (b[0][2] == b[1][1] and b[1][1] == b[2][0]) :

        if (b[0][2] == player) :
            return 10
        elif (b[0][2] == opponent) :
            return -10


    return 0

def minimax(board, depth, isMax) :
    score = evaluate(board)

    if (score == 10) :
        return score
    if (score == -10) :
        return score
    if (isMovesLeft(board) == False) :
        return 0

    if (isMax) :
        best = -1000
```

```
        for i in range(3) :
            for j in range(3) :


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


                    board[i][j] = player


                    best = max( best, minimax(board,
                                      depth + 1,
                                      not isMax) )

                    board[i][j] = '_'
            return best
        else :
            best = 1000


            for i in range(3) :
                for j in range(3) :


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


                        board[i][j] = opponent

                        best = min(best, minimax(board, depth + 1, not isMax))


                        board[i][j] = '_'
            return best

    def findBestMove(board) :
        bestVal = -1000
        bestMove = (-1, -1)


        for i in range(3) :
            for j in range(3) :


                if (board[i][j] == '_') :
```

```
            board[i][j] = player

            moveVal = minimax(board, 0, False)

            board[i][j] = '_'

            if (moveVal > bestVal) :
                bestMove = (i, j)
                bestVal = moveVal

    print("The value of the best Move is :", bestVal)
    print()
    return bestMove

board = [
    [ 'x', 'o', 'x' ],
    [ 'o', 'x', 'o' ],
    [ '_', '_', '_' ]
]

bestMove = findBestMove(board)

print("The Optimal Move is :")
print("ROW:", bestMove[0], " COL:", bestMove[1])
```

Output screen shots :

```
The value of the best Move is : 10

Optimal Move  :
ROW: 2  COL: 0
```

Result : The Tic Tac Toe problem was implemented successfully using minmax algorithm to evaluate the best moves with the highest score.