**Batch:** B1 **Roll Number:** 1914078 **Name:** Devansh Shah

**Experiment Number:** 4

**Aim of the Experiment:** Implementation of Adversarial algorithm-Min-Max for Tic-Tac-Toe Game

**Program/ Steps:**

**Program-**

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

    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("\nThe value of the best Move is :", bestVal)

    print()

    return bestMove

# Driver code

board = []

print("\nEnter space separated rows for 3x3 tic tac toe matrix")

for i in range(3):

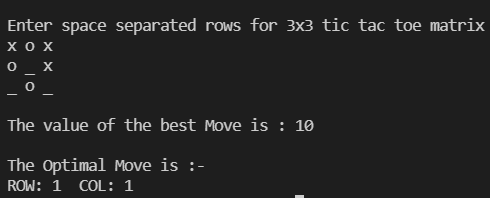
    board.append(input().split(" "))

bestMove = findBestMove(board)

print("The Optimal Move is :-")

print("ROW:", bestMove[0], " COL:", bestMove[1])

**OUTPUT:**



**Questions:**

1. **Game playing is often called as an**

a) Non-adversial search

b) Adversial search

c) Sequential search

d) None of the above

**Answer: b) Adversial Search**

2**. What are the basic requirements or need of AI search methods in game playing?**

a) Initial State of the game

b) Operators defining legal moves

c) Successor functions

d) Goal test

e) Path cost

**Answer: a) Initial state of the game**

**Outcomes:**

**CO2 :**Analyze and formalize the problem (as a state space, graph, etc.) and select the appropriate search method and write the algor

**Conclusion:** We implemented minimax and its significance was understood to be a recursive or backtracking algorithm which helps in decision-making and game theory.

**References:**

* Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach,

Second Edition, Pearson Publication

* Elaine Rich, Kevin Knight, Artificial Intelligence, Tata McGraw Hill, 1999.