**18CSC305J- ARTIFICIAL INTELLEGENCE**

**Experiment-6**

**Min Max Algorithm (Tic Tac Toe Problem)**

**Team Ai 4 life:**

Praneet Botke (RA1911031010149)

Sai Mohit Ambekar (RA1911031010137)

Sadekar Adesh H. (RA1911031010141)

Kapuluru Srinivasulu (RA1911031010142)

Ayushi Goenka (RA1911031010151)

Sonia Raja (RA1911031010152)

**Aim:**

To implement Min Max algorithm in Tic – Tac – Toe AI – Finding Optimal Move Problem.

**Code:**

player, opponent = 'x', 'o'

def movesLeft(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 1

elif (b[row][0] == opponent):

return -1

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 1

elif (b[0][col] == opponent):

return -1

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

if (b[0][0] == player):

return 1

elif (b[0][0] == opponent):

return -1

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

if (b[0][2] == player):

return 1

elif (b[0][2] == opponent):

return -1

return 0

# It considers all the possible ways the game can go and returns the value of the board

def minmax(board, depth, isMax):

score = evaluate(board)

# If Maximizer has won the game return his/her

# evaluated score

if (score == 1):

return score

# If Minimizer has won the game return his/her

# evaluated score

if (score == -1):

return score

if (movesLeft(board) == False):

return 0

# If this maximizer's move

if (isMax):

best = -100

for i in range(3):

for j in range(3):

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

# Make the move

board[i][j] = player

# Call minmax recursively and choose

# the maximum value

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

board[i][j] = '\_'

return best

# If this minimizer's move

else:

best = 100

for i in range(3):

for j in range(3):

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

board[i][j] = opponent

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

# Undo the move

board[i][j] = '\_'

return best

def findBestMove(board):

bestVal = -100

bestMove = (-1, -1)

for i in range(3):

for j in range(3):

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

board[i][j] = player

# compute evaluation function for this move.

moveVal = minmax(board, 0, False)

board[i][j] = '\_'

# if moveVal value is more than bestVal then update the bestVal

if (moveVal > bestVal):

bestMove = (i, j)

bestVal = moveVal

print("The value of the best move is:", bestVal)

print()

return bestMove

board = [

[ 'o', 'o', '\_' ],

[ 'o', 'x', 'x' ],

[ 'x', '\_', 'o' ]

]

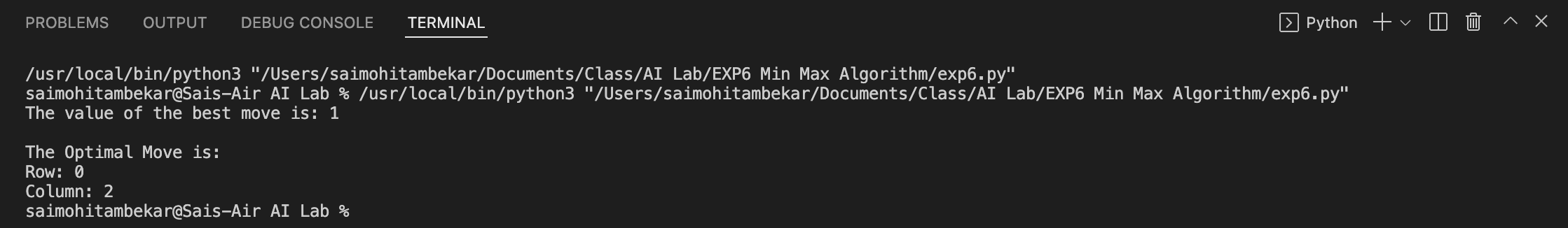
bestMove = findBestMove(board)

print("The Optimal Move is:")

print("Row:", bestMove[0])

print("Column:", bestMove[1])

**Output:**

****

**Result:**

Min Max algorithm for an application – Tic Tac Toe finding optimal move was successfully implemented and verified using python 3.