Name : Krish Gupta

Roll no : 29

Div : D

SRN No : 202101521

Title : Implement the Minimax algorithm to solve the Tic Tac Toe problem

Code:

import math

def print\_board(board):

    for row in board:

        print(" | ".join(row))

        print("-" \* 5)

def evaluate(board):

    # Check rows

    for row in board:

        if row.count('X') == 3:

            return 10

        elif row.count('O') == 3:

            return -10

    # Check columns

    for col in range(3):

        if board[0][col] == board[1][col] == board[2][col] == 'X':

            return 10

        elif board[0][col] == board[1][col] == board[2][col] == 'O':

            return -10

    # Check diagonals

    if board[0][0] == board[1][1] == board[2][2] == 'X' or \

       board[0][2] == board[1][1] == board[2][0] == 'X':

        return 10

    elif board[0][0] == board[1][1] == board[2][2] == 'O' or \

         board[0][2] == board[1][1] == board[2][0] == 'O':

        return -10

    return 0

def is\_moves\_left(board):

    for row in board:

        for cell in row:

            if cell == '\_':

                return True

    return False

def minimax(board, depth, is\_maximizing):

    score = evaluate(board)

    if score == 10:

        return score - depth

    if score == -10:

        return score + depth

    if not is\_moves\_left(board):

        return 0

    if is\_maximizing:

        best = -math.inf

        for i in range(3):

            for j in range(3):

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

                    board[i][j] = 'X'

                    best = max(best, minimax(board, depth + 1, not is\_maximizing))

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

        return best

    else:

        best = math.inf

        for i in range(3):

            for j in range(3):

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

                    board[i][j] = 'O'

                    best = min(best, minimax(board, depth + 1, not is\_maximizing))

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

        return best

def find\_best\_move(board):

    best\_val = -math.inf

    best\_move = (-1, -1)

    for i in range(3):

        for j in range(3):

            if board[i][j] == '\_':

                board[i][j] = 'X'

                move\_val = minimax(board, 0, False)

                board[i][j] = '\_'

                if move\_val > best\_val:

                    best\_move = (i, j)

                    best\_val = move\_val

    return best\_move

def play():

    board = [['\_', '\_', '\_'],

             ['\_', '\_', '\_'],

             ['\_', '\_', '\_']]

    print("Let's play Tic Tac Toe!")

    print\_board(board)

    while is\_moves\_left(board):

        player\_move = tuple(map(int, input("Enter your move (row col): ").split()))

        if board[player\_move[0]][player\_move[1]] != '\_':

            print("Invalid move. Try again.")

            continue

        board[player\_move[0]][player\_move[1]] = 'O'

        print\_board(board)

        if evaluate(board) == -10:

            print("You win!")

            return

        if not is\_moves\_left(board):

            print("It's a draw!")

            return

        ai\_move = find\_best\_move(board)

        board[ai\_move[0]][ai\_move[1]] = 'X'

        print("AI's move:")

        print\_board(board)

        if evaluate(board) == 10:

            print("AI wins!")

            return

    print("It's a draw!")

play()

Output:

