Usama Mehboob **Roll No 225** **Section E**

LAB # 12

GAME THOERY IN ARTIFICIAL INTELLIGENCE

Objective:-

Implement Tic-Tac-Toe game using Minimax Algorithm in Game Theory

Lab Task:-

Finding optimal move in Tic-Tac-Toe using Minimax Algorithm in Game Theory?

Code:

```
import math
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 5)

def is_moves_left(board):
    for row in board:
        if "_" in row:
            return True
    return False

def evaluate(board):
    for row in board:
        if row[0] == row[1] == row[2]:
            if row[0] == 'X':
                return 10
            elif row[0] == 'O':
                return -10

    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col]:
            if board[0][col] == 'X':
                return 10
            elif board[0][col] == 'O':
                return -10
```

```
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col]:
            if board[0][col] == 'X':
                return 10
            elif board[0][col] == 'O':
                return -10

    if board[0][0] == board[1][1] == board[2][2]:
        if board[0][0] == 'X':
            return 10
        elif board[0][0] == 'O':
            return -10

    if board[0][2] == board[1][1] == board[2][0]:
        if board[0][2] == 'X':
            return 10
        elif board[0][2] == 'O':
            return -10

    return 0

def minimax(board, depth, is_max):
    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_max:
        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, False))
                    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, True))
                    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, True))
                    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_val = move_val
                    best_move = (i, j)
```

```
    return best_move

if __name__ == "__main__":
    board = [
        ["X", "O", "X"],
        ["_", "O", "_"],
        ["_", "_", "_"]
    ]

    print("Initial Board:")
    print_board(board)

    best_move = find_best_move(board)
    print(f"\nThe best move is at position: {best_move}")
```

Output:

```
Initial Board:
X | O | X
-----
_ | O | _
-----
_ | _ | _
-----

The best move is at position: (2, 1)
```