EXPERIMENT 13

Minimax algorithm for gaming

AIM:

To Write the python program to implement Minimax algorithm for gaming.

PROCEDURE:

1. **Initialization**:
    - Initialize the game board with empty cells.
2. **Player Input**:
    - Request input from the player for placing 'X' on the board.
3. **Evaluation and Move Generation**:
    - Evaluate the current state of the board to determine if there's a winner, a tie, or the game continues.
    - Use the minimax algorithm to generate the best move for the computer player ('O'). This involves recursively exploring possible future game states.
4. **Apply Moves**:
    - Place 'X' on the board according to player input.
    - Place 'O' on the board based on the calculated best move.
5. **Loop**:
    - Repeat steps 2-4 for a fixed number of iterations (in this case, 4) to simulate a partial game.
6. **Game End**:
    - Display the final state of the board after the specified number of moves.

PROGRAM:

```
def is_winner(board, player):

    for i in range(3):
        if all(board[i][j] == player for j in range(3)) or all(board[j][i] == player for j in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in
range(3)):
```

```
        return True
    return False


def is_board_full(board):
    return all(board[i][j] != '-' for i in range(3) for j in range(3))


def evaluate(board):
    if is_winner(board, 'X'):
        return -1
    elif is_winner(board, 'O'):
        return 1
    elif is_board_full(board):
        return 0
    else:
        return None

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

    if score is not None:
        return score

    if maximizing_player:
        max_eval = float('-inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == '-':
                    board[i][j] = 'O'
                    eval = minimax(board, depth + 1, False)
                    board[i][j] = '-'
                    max_eval = max(max_eval, eval)
        return max_eval
    else:
        min_eval = float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == '-':
                    board[i][j] = 'X'
                    eval = minimax(board, depth + 1, True)
                    board[i][j] = '-'
                    min_eval = min(min_eval, eval)
        return min_eval
```

```
def find_best_move(board):
    best_val = float('-inf')
    best_move = (-1, -1)

    for i in range(3):
        for j in range(3):
            if board[i][j] == '-':
                board[i][j] = 'O'
                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 print_board(board):
    for row in board:
        print(' '.join(row))
    print()


def main():

    board = [['-' for _ in range(3)] for _ in range(3)]

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

    for _ in range(4):

        x, y = map(int, input("Enter the coordinates for 'X' (row and column): ").split())
        board[x][y] = 'X'
        print_board(board)


        print("Player O's move:")
        best_move = find_best_move(board)
        board[best_move[0]][best_move[1]] = 'O'
        print_board(board)


    print("Final Board:")
    print_board(board)
```

```
if __name__ == "__main__":
    main()
```

OUTPUT:

```
Enter the coordinates for 'X' (row and column): 2 1
O - X
- X -
- X -

Player O's move:
O O X
- X -
- X -

Enter the coordinates for 'X' (row and column): 2 0
O O X
- X -
X X -

Player O's move:
O O X
O X -
X X -

Final Board:
O O X
O X -
X X -
```

RESULT:

Hence the program has been successfully verified.