



Name: Subrato Tapaswi

Class/Roll No: D16AD/60

Grade :

Title of Experiment: Mini Project: Implementation of any Game- Connect Four

Program:

```
def winning_move(board, piece):
    # Check horizontal locations for win
    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT):
            if board[r][c] == piece and board[r][c+1] == piece and board[r][c+2] == piece and board[r][c+3] == piece:
                return True

    # Check vertical locations for win
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT-3):
            if board[r][c] == piece and board[r+1][c] == piece and board[r+2][c] == piece and board[r+3][c] == piece:
                return True

    # Check positively sloped diagonals
    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT-3):
            if board[r][c] == piece and board[r+1][c+1] == piece and board[r+2][c+2] == piece and board[r+3][c+3] == piece:
                return True

    # Check negatively sloped diagonals
    for c in range(COLUMN_COUNT-3):
        for r in range(3, ROW_COUNT):
            if board[r][c] == piece and board[r-1][c+1] == piece and board[r-2][c+2] == piece and board[r-3][c+3] == piece:
                return True

def minimax(board, depth, alpha, beta, maximizingPlayer):
    valid_locations = get_valid_locations(board)
    is_terminal = is_terminal_node(board)
    if depth == 0 or is_terminal:
        if is_terminal:
            if winning_move(board, AI_PIECE):
                return (None, 1000000000000000)
            elif winning_move(board, PLAYER_PIECE):
                return (None, -1000000000000000)
            else: # Game is over, no more valid moves
                return (None, 0)
        else: # Depth is zero
            return (None, score_position(board, AI_PIECE))
    if maximizingPlayer:
        value = -math.inf
        column = random.choice(valid_locations)
        for col in valid_locations:
            row = get_next_open_row(board, col)
            b_copy = board.copy()
            drop_piece(b_copy, row, col, AI_PIECE)
            new_score = minimax(b_copy, depth-1, alpha, beta, False)[1]
            if new_score > value:
                value = new_score
                column = col
            alpha = max(alpha, value)
            if alpha >= beta:
                break
    return column, value
```



```
def get_valid_locations(board):
    valid_locations = []
    for col in range(COLUMN_COUNT):
        if is_valid_location(board, col):
            valid_locations.append(col)
    return valid_locations

def pick_best_move(board, piece):

    valid_locations = get_valid_locations(board)
    best_score = -10000
    best_col = random.choice(valid_locations)
    for col in valid_locations:
        row = get_next_open_row(board, col)
        temp_board = board.copy()
        drop_piece(temp_board, row, col, piece)
        score = score_position(temp_board, piece)
        if score > best_score:
            best_score = score
            best_col = col

    return best_col
```

Output/Game:

