Ex No: 1c

IMPLEMENTATION OF MINIMAX algorithm

Aim:

To implement MINIMAX algorithm.

Scenario: AI vs. Human Player – Winning Move Situation

Context: **The AI is playing as** Player X **and the human is playing as** Player O**. It's** AI's turn**, and there is a possible** winning move.

Given Board State (Before AI's Move)**:**

X O X
O X .
. O X

Expected AI Move (Best Move using Minimax):

X O X
O X X
. O X

Procedure:

1. Define constants: `PLAYER_X = 1`, `PLAYER_O = -1`, `EMPTY = 0`.
2. Create `evaluate(board)` to check for a winner by scanning rows, columns, and diagonals. Return `1` if AI wins, `-1` if human wins, or `0` if no winner.
3. Create `isMovesLeft(board)` to check for empty spaces; return `True` if moves are available, otherwise `False`.
4. Implement `minimax(board, isMax)`:

- If `evaluate(board)` returns a winner, return the corresponding score.
- If no moves are left, return `0`.
- If `isMax` is `True` (AI's turn), initialize `best = -∞`, loop through empty cells, place `X`, call `minimax(board, False)`, undo move, update `best` with maximum value, and return `best`.
- If `isMax` is `False` (Human's turn), initialize `best = +∞`, loop through empty cells, place `O`, call `minimax(board, True)`, undo move, update `best` with minimum value, and return `best`.

5. Implement `findBestMove(board)`:

- Initialize `bestVal = -∞` and `bestMove = (-1, -1)`.
- Loop through empty cells, place `X`, call `minimax(board, False)`, undo move, update `bestMove` if a better move is found.
- Return `bestMove`.

6. Implement `printBoard(board)` to display board state using `"X"`, `"O"`, and `"."` for empty spaces.
7. Initialize a sample `board`, print its state, call `findBestMove(board)`, update the board with AI's move, and print the final state.

Program:

```
# Constants for players
PLAYER_X = 1
PLAYER_O = -1
EMPTY = 0
# Evaluate the board
def evaluate(board):
    for row in range(3):
        if board[row][0] == board[row][1] == board[row][2] != EMPTY:
            return board[row][0]
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != EMPTY:
            return board[0][col]
    if board[0][0] == board[1][1] == board[2][2] != EMPTY:
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] != EMPTY:
        return board[0][2]
    return 0
# Check if moves are left
def isMovesLeft(board):
    for row in range(3):
        for col in range(3):
            if board[row][col] == EMPTY:
                return True
    return False
# Minimax function
def minimax(board, isMax):
    score = evaluate(board)
```

```
    if score == PLAYER_X: return score
    if score == PLAYER_O: return score
    if not isMovesLeft(board): return 0
    if isMax:
        best = -float('inf')
        for row in range(3):
            for col in range(3):
                if board[row][col] == EMPTY:
                    board[row][col] = PLAYER_X
                    best = max(best, minimax(board, not isMax))
                    board[row][col] = EMPTY
        return best
    else:
        best = float('inf')
        for row in range(3):
            for col in range(3):
                if board[row][col] == EMPTY:
                    board[row][col] = PLAYER_O
                    best = min(best, minimax(board, not isMax))
                    board[row][col] = EMPTY
        return best
# Find the best move for PLAYER_X
def findBestMove(board):
    bestVal = -float('inf')
    bestMove = (-1, -1)
    for row in range(3):
        for col in range(3):
            if board[row][col] == EMPTY:
                board[row][col] = PLAYER_X
                moveVal = minimax(board, False)
                board[row][col] = EMPTY
                if moveVal > bestVal:
```

```
                bestMove = (row, col)
                bestVal = moveVal
    return bestMove
# Print the board
def printBoard(board):
    for row in board:
        print(" ".join(["X" if x == PLAYER_X else "O" if x == PLAYER_O else "." for x in
row]))
# Example game
board = [
    [PLAYER_X, PLAYER_O, PLAYER_X],
    [PLAYER_O, PLAYER_X, EMPTY],
    [EMPTY, PLAYER_O, PLAYER_X]
]
print("Current Board:")
printBoard(board)
move = findBestMove(board)
print(f"Best Move: {move}")
board[move[0]][move[1]] = PLAYER_X
print("\nBoard after best move:")
printBoard(board)
```

Output:

```
Current Board:
X O X
O X .
. O X

Best Move: (2, 0)

Board after best move:
X O X
O X .
X O X
```