30/4/25

Implementation of MIN MAX Algorithm.

Aim: To implement min max algorithm

Scenario: AI vs Human player - winning move situation.

Context: here The AI is playing as X and human is playing as O. Its AI's turn, and there is a possible winning move.

given Board state (Before AI 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 Constraints: Player_X = 1, Player_O = -1 empty = 0.

2 Create evaluate(board) to check for a winner by scanning rows and columns or diagonals.

```
return 0

# check if moves are left.
 def is moveleft (board):
      for row in range (3):
           for col in range (3):
                if board[row][col] == EMPTY
                   return True
   return False.


def  minimax (board, is Max):
    score = evaluate (board)
    if score == player_X return score
    if core == PLAYER_O return score
    if not is Movesleft (board): return 0
    if is Max:
          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 is Max)
                        board [row][col] = Empty
        return Best
```

3 Create isMovesLeft(Board) to check for empty spaces, return true if moves are available, else false.

4 Implement minimax(Board, isMax):

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

5 Implement find Best Move (board):

- Initialise bestVal = -∞ and best move = (-1, -1)
- loop through emptycells, place X call min max (board, False), undo move update best move, if a better move is found

6 Implement printboard (board) to display board state using "X", "O", "." for empty

7) Initialise a sample board, print its state, call find bestmove (board), update the board with AI's move, and print its final state.

```
# Find the best move for PLAYER_X
def find bestMove(board):
    best val = - float('inf)
    best Move = (-1,-1)
    for row in range(3):
        for col in range(3):
            if board[row][col] == EMPTY
            board[row][col] = PLAYER_X
            move Val = minimax(board, False)
            board[row][col] = Empty
            if move Val > best Val:
                best_Move = (row, col)
                best Val = move Val

# 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]))


Board = [
    [PLAYER-X; PLAYER_O, PLAYER_X],
    [PLAYER_O, PLAYER_X , 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
```