

```
# 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):
```

```
        for col in range(3):
            if board[row][col] == EMPTY:
                board[row][col] = PLAYER_X
                best = max(best, minimax(board, False))
                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, True))
                    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]
```

```
[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)
```

IDLE Shell 3.11.9

File Edit Shell Debug Options Window Help

Python 3.11.9 (tags/v3.11.9:de54cf5, Apr 2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

```
>>>  
= RESTART: C:/Users/DELL/AI.py  
Current Board:  
X O X  
O X .  
. O X  
Best Move: (1, 2)
```

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

```
>>> |
```