# Minimax Algorithm for Tic-Tac-Toe

This is a simple Minimax algorithm for a Tic-Tac-Toe game. It helps find the best move for the 'X' player.

## Functions:

1. is\_moves\_left(board): Checks if there are empty spaces left on the board.

2. evaluate(board): Evaluates the board and assigns scores (10 for 'X' win, -10 for 'O' win, 0 for draw).

3. minimax(board, is\_max): Recursively determines the best move for 'X' and 'O'.

4. find\_best\_move(board): Finds the best move for 'X' using Minimax.

## Python Code:

def is\_moves\_left(board):  
 """Checks if there are any empty spaces left on the board."""  
 for row in board:  
 if " " in row:  
 return True  
 return False  
  
def evaluate(board):  
 """Evaluates the board and returns a score based on the game state."""  
 for row in board:  
 if row[0] == row[1] == row[2] and row[0] != " ":  
 return 10 if row[0] == 'X' else -10  
   
 for col in range(3):  
 if board[0][col] == board[1][col] == board[2][col] and board[0][col] != " ":  
 return 10 if board[0][col] == 'X' else -10  
   
 if board[0][0] == board[1][1] == board[2][2] and board[0][0] != " ":  
 return 10 if board[0][0] == 'X' else -10  
   
 if board[0][2] == board[1][1] == board[2][0] and board[0][2] != " ":  
 return 10 if board[0][2] == 'X' else -10  
   
 return 0  
  
def minimax(board, is\_max):  
 """Finds the best move using recursion."""  
 score = evaluate(board)  
 if score in [10, -10]:  
 return score  
 if not is\_moves\_left(board):  
 return 0  
   
 best = -1000 if is\_max else 1000  
 player = 'X' if is\_max else 'O'  
   
 for i in range(3):  
 for j in range(3):  
 if board[i][j] == " ":  
 board[i][j] = player  
 best = max(best, minimax(board, not is\_max)) if is\_max else min(best, minimax(board, not is\_max))  
 board[i][j] = " "  
 return best  
  
def find\_best\_move(board):  
 """Finds the best possible move for 'X'."""  
 best\_val = -1000  
 best\_move = (-1, -1)  
   
 for i in range(3):  
 for j in range(3):  
 if board[i][j] == " ":  
 board[i][j] = 'X'  
 move\_val = minimax(board, False)  
 board[i][j] = " "  
 if move\_val > best\_val:  
 best\_move = (i, j)  
 best\_val = move\_val  
   
 return best\_move  
  
  
board = [['X', 'O', 'X'],  
 ['O', 'O', 'X'],  
 [' ', ' ', ' ']]  
  
print("Best move:", find\_best\_move(board))