```python
# MiniMax - Get score for board

import math
import numpy as np
import time
import copy

# from copy import copy
COUNT = 0  # use the COUNT variable to track number of boards explored


def showBoard(board):
    # displays rows of board
    strings = ["" for i in range(board.shape[0])]
    idx = 0
    for row in board:
        for cell in row:
            if cell == 1:
                s = "X"
            elif cell == -1:
                s = "O"
            else:
                s = "_"

            strings[idx] += s
        idx += 1

    # display final board
    for s in strings:
        print(s)


def get_board_one_line(board):
    # returns one line rep of a board
    import math

    npb_flat = board.ravel()
    stop = int(math.sqrt(len(npb_flat)))

    bstr = ""
    for idx in range(len(npb_flat)):
        bstr += str(npb_flat[idx]) + " "
        if (idx + 1) % (stop) == 0:
            bstr += "|"
    return bstr


def evaluate(board):
    i = 0
    while i <= board.shape[0] - 1:
        if (
            np.sum(board, 0)[i] == board.shape[0]
            or np.sum(board, 1)[i] == board.shape[0]
            or np.sum(np.diagonal(board)) == board.shape[0]
            or np.sum(np.fliplr(board).diagonal()) == board.shape[0]
        ):
            return 1

        elif (
            np.sum(board, 0)[i] == -1 * board.shape[0]
            or np.sum(board, 1)[i] == -1 * board.shape[0]
            or np.sum(np.diagonal(board)) == -1 * board.shape[0]
            or np.sum(np.fliplr(board).diagonal())
            == -1 * board.shape[0]
        ):
            return -1
        i += 1

    return 0


def is_terminal_node(board):

    board_val = evaluate(board)

    if board_val == 1 or board_val == -1:
        return True
    elif board_val == 0 and np.argwhere(board == 0).size == 0:
        return True
    else:
        return False


def get_child_boards(board, char):
    """numpy version"""
    if not char in ["X", "O"]:
        raise ValueError("get_child_boards: expecting char='X' or 'O' ")

    newval = -1
    if char == "X":
        newval = 1

    child_list = []
    zero_values = np.argwhere(board == 0)  # Determine indeces of zeros
    temp_arr = []

    for indice in zero_values:
        temp_arr = copy.deepcopy(board)
        temp_arr[indice[0]][indice[1]] = newval
        child_list.append(temp_arr)

    return child_list


def minimax(board, depth, maximizingPlayer):
    """returns the value of the board
    0 (draw) 1 (win for X) -1 (win for O)
    Explores all child boards for this position and returns
    the best score given that all players play optimally
    """
    global COUNT
    COUNT += 1
    # print(board) # Debug line
    if depth == 0 or is_terminal_node(board):
        return evaluate(board)

    if maximizingPlayer:  # max player plays X
        maxEva = -math.inf
        child_list = get_child_boards(board, "X")
        # print(f"Max child lists:\n",child_list, "\n")
        for child_board in child_list:
            eva = minimax(child_board, depth - 1, False)
            maxEva = max(maxEva, eva)
        return maxEva

    else:  # minimizing player
        minEva = math.inf
        child_list = get_child_boards(board, "O")
        # print(f"Min child lists:\n",child_list, "\n")
        for child_board in child_list:
            eva = minimax(child_board, depth - 1, True)
            minEva = min(minEva, eva)
        return minEva


def run_code_tests():
    """
    b1 : expect win for X (1)  < 200 boards explored
    b1 = np.array([[1, 0, -1], [1, 0, 0], [-1, 0, 0]])

    In addtion to the board b1, run tests on the following
    boards:
       b2:  expect win for O (-1)  > 1000 boards explored
       b2 = np.array([[0, 0, 0], [1, -1, 1], [0, 0, 0]])

       b3: expect TIE (0)  > 500,000 boards explored; time around 20secs
       b3 = np.array([[0, 0, 0], [0, 0, 0], [0, 0, 0]])

       b4: expect TIE(0) > 7,000,000 boards;  time around 4-5 mins
       b4 = np.array(
        [[1, 0, 0, 0], [0, 1, 0, -1], [0, -1, 1, 0], [0, 0, 0, -1]])

    """
    # Minimax for a board: evaluate the board
    b1 = np.array([[1, 0, -1], [1, 0, 0], [-1, 0, 0]])
    b2 = np.array([[0, 0, 0], [1, -1, 1], [0, 0, 0]])
    b3 = np.array([[0, 0, 0], [0, 0, 0], [0, 0, 0]])
    b4 = np.array(
        [[1, 0, 0, 0], [0, 1, 0, -1], [0, -1, 1, 0], [0, 0, 0, -1]]
    )

    # Making it easier to switch boards:
    board = b4
    max_depth = np.count_nonzero(board == 0)
    print(f"Running minimax w/ max depth {max_depth} for:\n", board)

    if np.sum(board) == 0:
        is_x_to_move = True
    elif np.sum(board) == 1:
        is_x_to_move = False
    else:
        print("illegal board")
        exit

    # read time before and after call to minimax
    print(time.ctime())
    score = minimax(board, max_depth, is_x_to_move)
    print(time.ctime())

    print(f"score : {score}")
    print(f"board count is", COUNT)


if __name__ == "__main__":
    run_code_tests()

```