```
import math
import copy

X = "X"
O = "O"
EMPTY = None


def initial_state():
    """
    Returns starting state of the board.
    """
    return [[EMPTY, EMPTY, EMPTY],
            [EMPTY, EMPTY, EMPTY],
            [EMPTY, EMPTY, EMPTY]]


def player(board):
    """
    Returns player who has the next turn on a board.
    """
    x_count = sum(row.count(X) for row in board)
    o_count = sum(row.count(O) for row in board)
    return X if x_count <= o_count else O


def actions(board):
    """
    Returns set of all possible actions (i, j) available on the board.
    """
    return {(i, j) for i in range(3) for j in range(3) if board[i][j] == EMPTY}


def result(board, action):
    """
    Returns the board that results from making move (i, j) on the board.
    """
    i, j = action

    # Check if action is valid
    if board[i][j] is not EMPTY:
        raise ValueError("Invalid action: Cell is not empty.")
```

```
    # Deep copy the board
    new_board = copy.deepcopy(board)

    # Apply the move
    new_board[i][j] = player(board)

    return new_board


def winner(board):
    """
    Returns the winner of the game, if there is one.
    """
    # Check rows
    for i in range(3):
        if board[i][0] == board[i][1] == board[i][2] != EMPTY:
            return board[i][0]

    # Check columns
    for i in range(3):
        if board[0][i] == board[1][i] == board[2][i] != EMPTY:
            return board[0][i]

    # Check diagonals
    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 None


def terminal(board):
    """
    Returns True if game is over, False otherwise.
    """
    # If someone has won, the game is over
    if winner(board) is not None:
        return True

    # If there are no EMPTY cells left, it's a draw (also over)
    for row in board:
        if EMPTY in row:
```

```
            return False

    return True


def utility(board):
    """
    Returns 1 if X has won the game, -1 if O has won, 0 otherwise.
    """
    win = winner(board)
    if win == X:
        return 1
    elif win == O:
        return -1
    else:
        return 0


def minimax(board):
    """
    Returns the optimal action for the current player on the board.
    """
    if terminal(board):
        return None  # No moves to make

    current_player = player(board)

    # Maximize for X
    if current_player == X:
        value = -math.inf
        best_move = None
        for action in actions(board):
            move_val = min_value(result(board, action))
            if move_val > value:
                value = move_val
                best_move = action
        return best_move

    # Minimize for O
    else:
        value = math.inf
        best_move = None
        for action in actions(board):
```

```
            move_val = max_value(result(board, action))
            if move_val < value:
                value = move_val
                best_move = action
        return best_move


def max_value(board):
    """
    Returns the maximum utility value for X.
    """
    if terminal(board):
        return utility(board)

    value = -math.inf
    for action in actions(board):
        value = max(value, min_value(result(board, action)))
    return value


def min_value(board):
    """
    Returns the minimum utility value for O.
    """
    if terminal(board):
        return utility(board)

    value = math.inf
    for action in actions(board):
        value = min(value, max_value(result(board, action)))
    return value
```