ESCUELA POLITÉCNICA NACIONAL

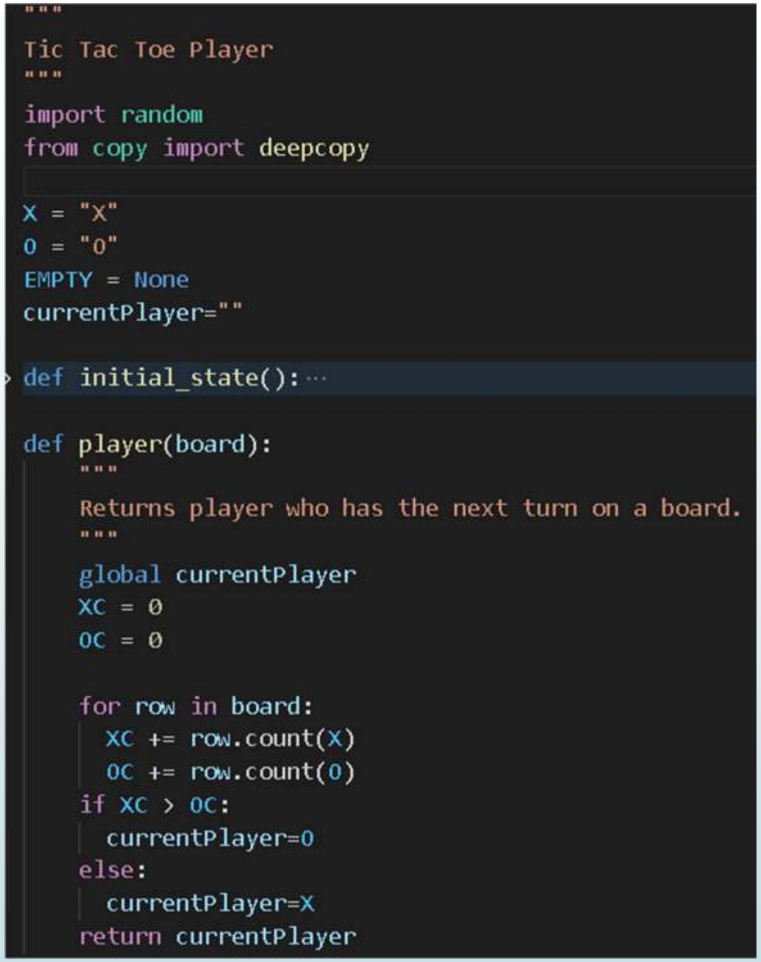
TIC TAC TOE

Planteamiento

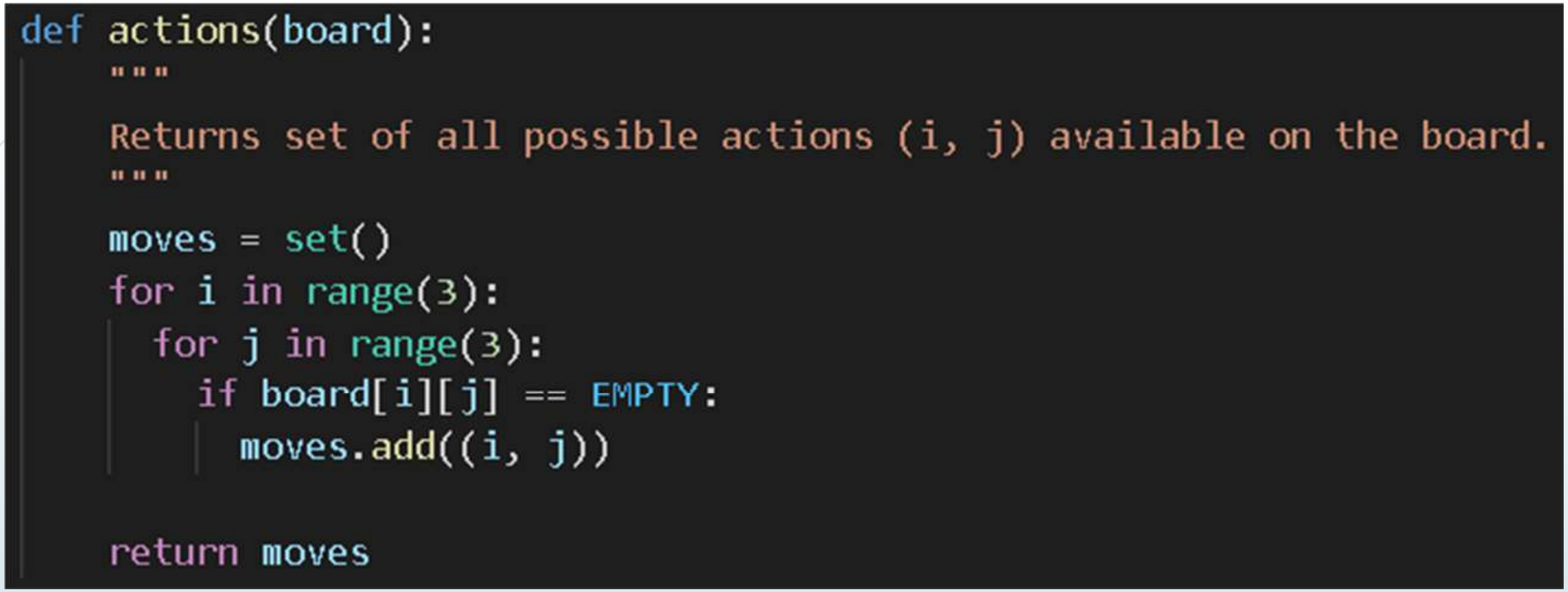
Crear un programa que implemente el juego Tic-Tac-Toe, de tal manera que el mejor de los casos sea un empate y el peor seria perder.

Implementar los métodos definidos para el funcionamiento eficiente del programa requerido.

Player

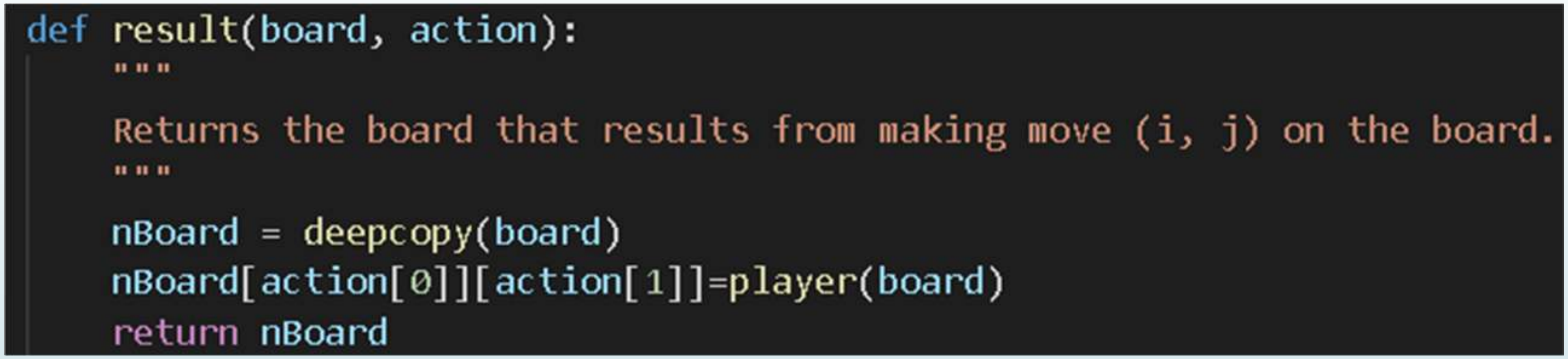


```
"""
Tic Tac Toe Player
"""

import random
from copy import deepcopy

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

def initial_state():···

def player(board):
    """
    Returns player who has the next turn on a board.
    """
    global currentPlayer
    XC = 0
    OC = 0

    for row in board:
        XC += row.count(X)
        OC += row.count(O)
    if XC > OC:
        currentPlayer=O
    else:
        currentPlayer=X
    return currentPlayer
```



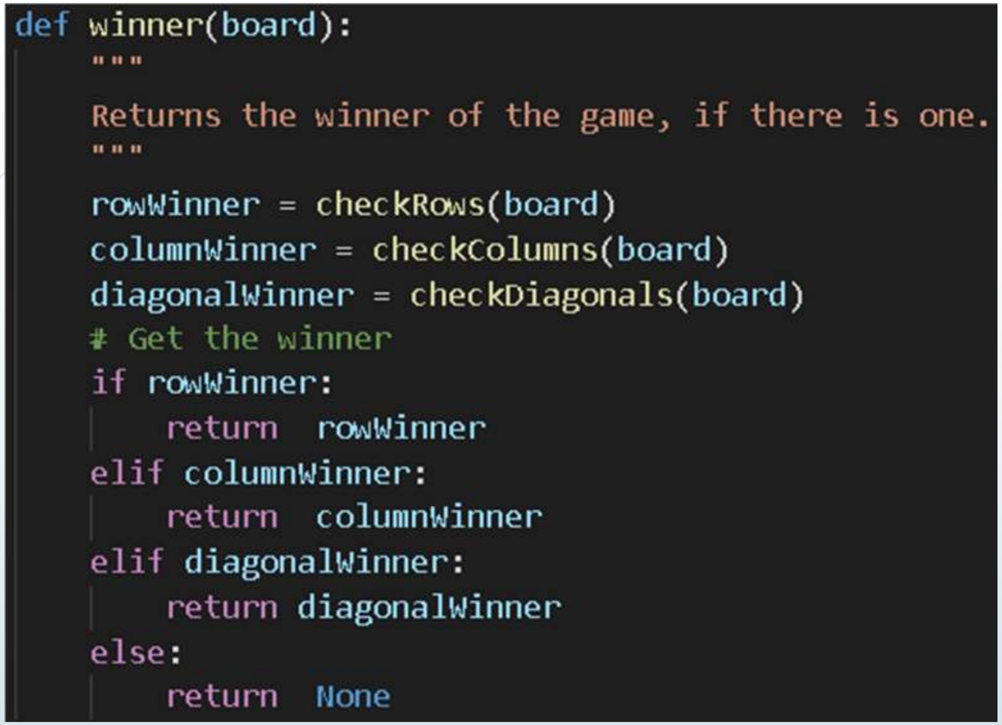
```python
def actions(board):
    """
    Returns set of all possible actions (i, j) available on the board.
    """
    moves = set()
    for i in range(3):
        for j in range(3):
            if board[i][j] == EMPTY:
                moves.add((i, j))

    return moves
```

Actions



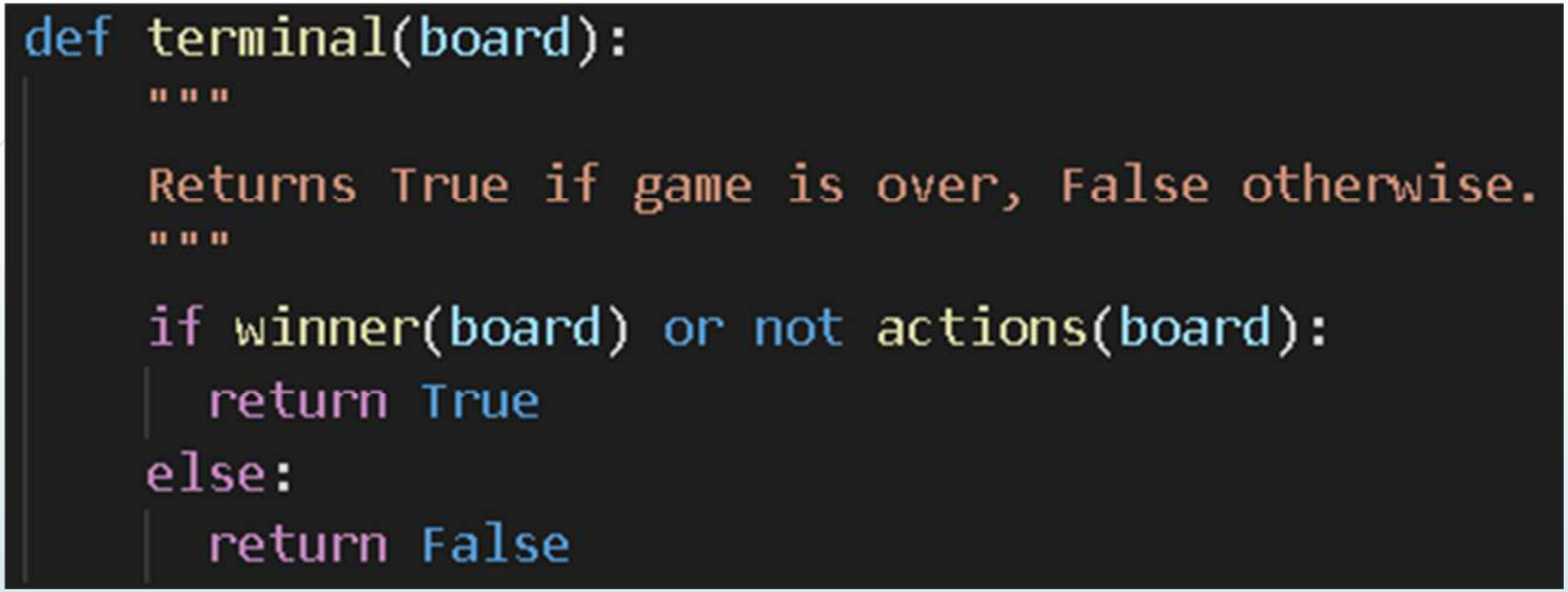
```
def result(board, action):
    """
    Returns the board that results from making move (i, j) on the board.
    """
    nBoard = deepcopy(board)
    nBoard[action[0]][action[1]]=player(board)
    return nBoard
```

Result



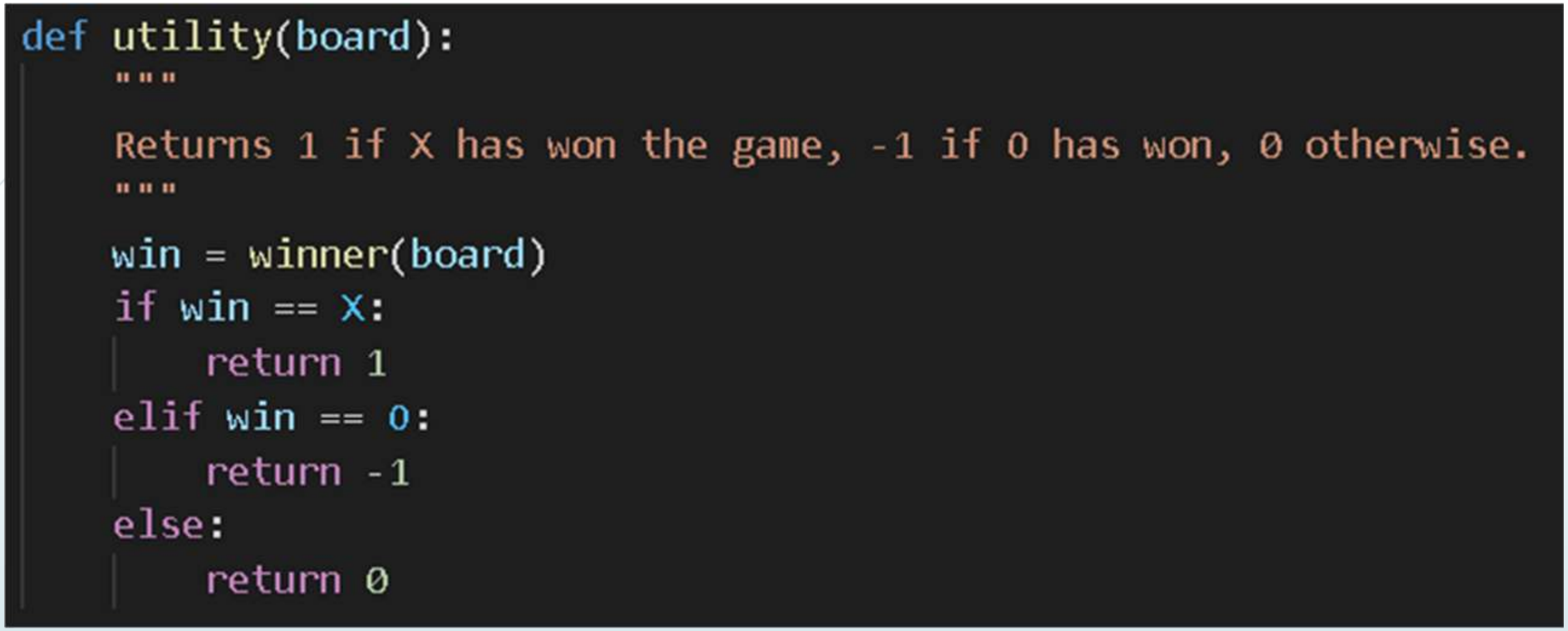
```python
def winner(board):
    """
    Returns the winner of the game, if there is one.
    """
    rowWinner = checkRows(board)
    columnWinner = checkColumns(board)
    diagonalWinner = checkDiagonals(board)
    # Get the winner
    if rowWinner:
        return  rowWinner
    elif columnWinner:
        return  columnWinner
    elif diagonalWinner:
        return diagonalWinner
    else:
        return  None
```

Winner



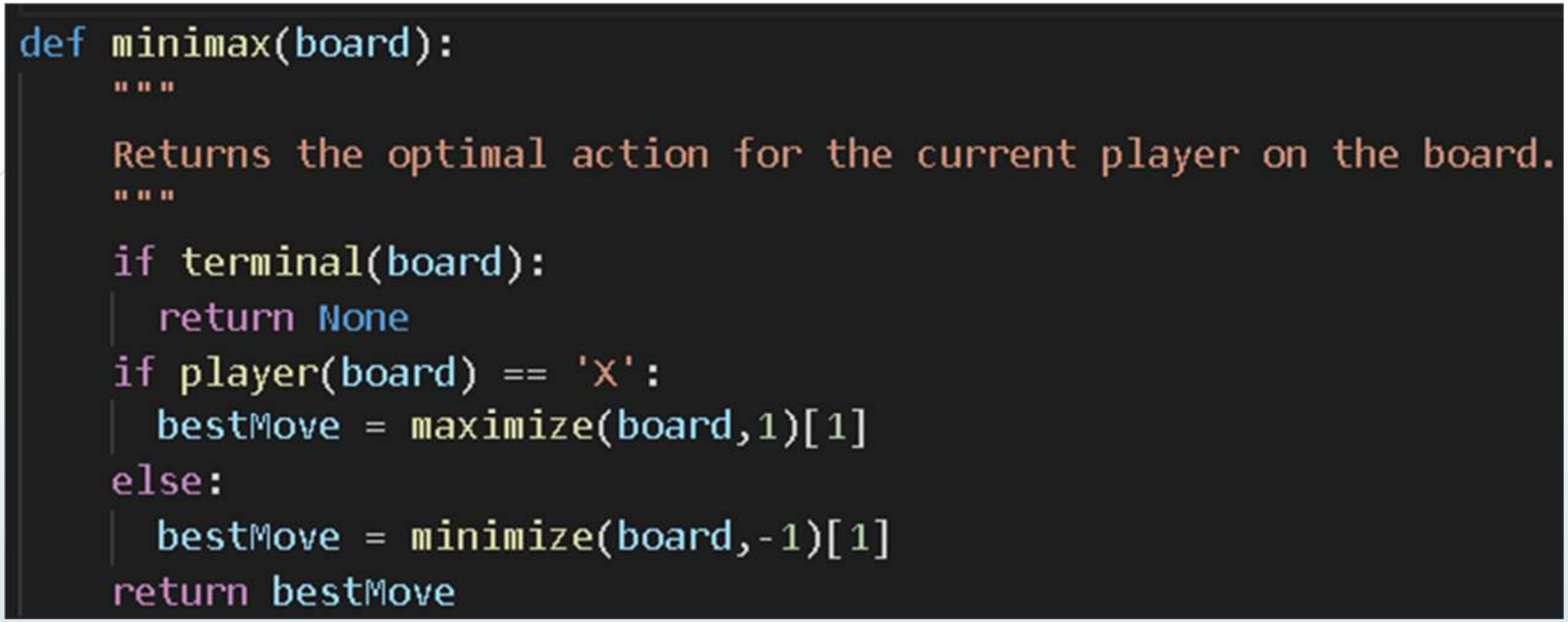
```python
def terminal(board):
    """
    Returns True if game is over, False otherwise.
    """
    if winner(board) or not actions(board):
        return True
    else:
        return False
```

Terminal



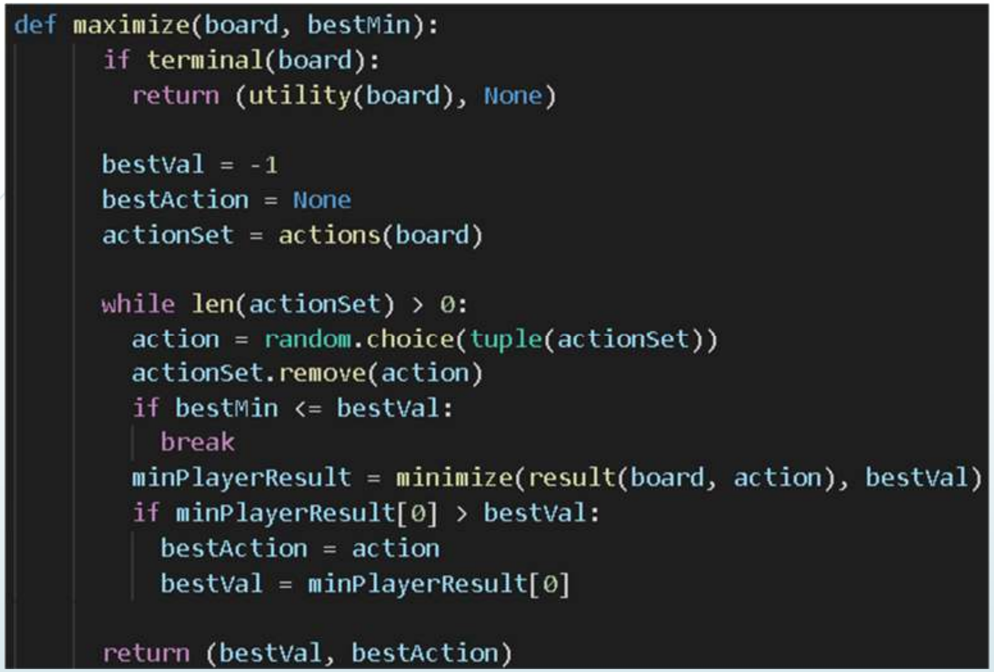
```python
def utility(board):
    """
    Returns 1 if X has won the game, -1 if O has won, 0 otherwise.
    """
    win = winner(board)
    if win == X:
        return 1
    elif win == O:
        return -1
    else:
        return 0
```

Utility



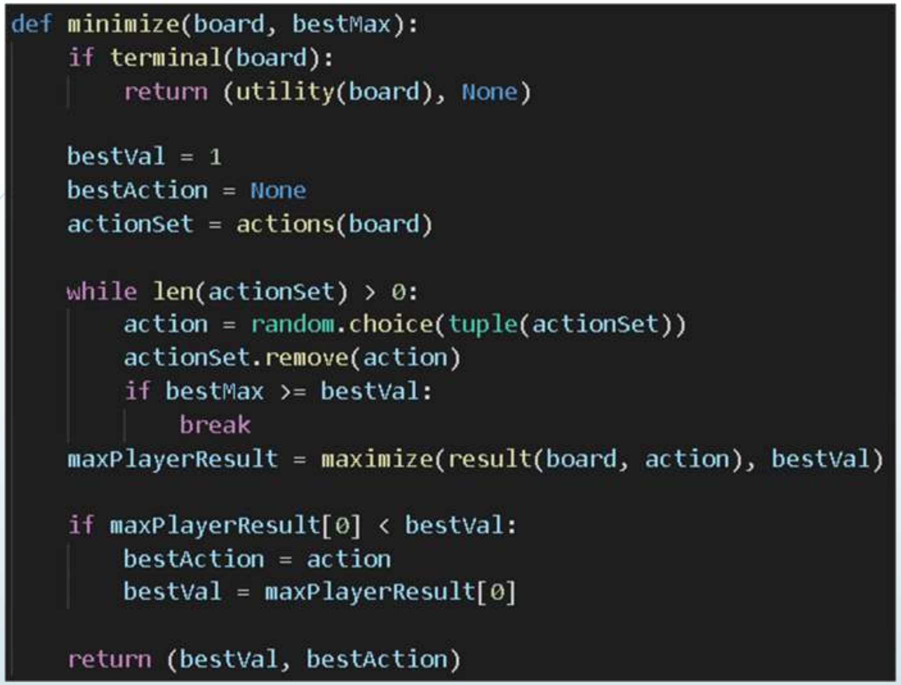
```python
def minimax(board):
    """
    Returns the optimal action for the current player on the board.
    """
    if terminal(board):
        return None
    if player(board) == 'X':
        bestMove = maximize(board,1)[1]
    else:
        bestMove = minimize(board,-1)[1]
    return bestMove
```

Minimax



```python
def maximize(board, bestMin):
      if terminal(board):
        return (utility(board), None)

      bestVal = -1
      bestAction = None
      actionSet = actions(board)

      while len(actionSet) > 0:
        action = random.choice(tuple(actionSet))
        actionSet.remove(action)
        if bestMin <= bestVal:
          break
        minPlayerResult = minimize(result(board, action), bestVal)
        if minPlayerResult[0] > bestVal:
          bestAction = action
          bestVal = minPlayerResult[0]

      return (bestVal, bestAction)
```

Maximize



```
def minimize(board, bestMax):
    if terminal(board):
        return (utility(board), None)

    bestVal = 1
    bestAction = None
    actionSet = actions(board)

    while len(actionSet) > 0:
        action = random.choice(tuple(actionSet))
        actionSet.remove(action)
        if bestMax >= bestVal:
            break
    maxPlayerResult = maximize(result(board, action), bestVal)

    if maxPlayerResult[0] < bestVal:
        bestAction = action
        bestVal = maxPlayerResult[0]

    return (bestVal, bestAction)
```

Minimize