Grader Description:

Submit a script that takes up to two command line arguments, both optional.

The first is an Othello board, represented by a string of 64 characters from {*".xoXO"}. The default, if the board is not provided is '.'*27 + "ox......xo" + '.'*27

The second is the token for which to find the set of moves. The default, if not provided, is the token that would move if there had been no passes in any prior moves.

The output is to be a 2D representation of the board, with possible move positions indicated by an asterisk. Following this (on one line) should be a sequence of all possible moves, as integers, where each integer is in range(64). If no moves are possible for the token in question, print out: No moves possible. Note: it may be the case that no moves are possible for the token in question while the other side does have moves.

In the input, 'X', 'O' are synonymous with 'x', 'o', respectively. Any version should be acceptable to your script. That is: ignore case.

At a minimum, to complete the 100 tests, the submitted script should run all the tests within 60 seconds and achieve at least 80% throughout the testing.

Eric's Notes:

These first few labs will be easy, but if you do well on these labs, you'll have a much smoother ride on the later ones.

The first of this lab is managing the input specifics -- make sure to first change everything to lowercase x's and o's.

For actually coding the possibleMoves() function, the initial thought should be to loop through the board and for each one of your tokens, loop through the possible directions to find possible moves (this is the obvious approach).

When doing this, you should also think about how you're going to play those possible moves and create new boards in the future.

When you're done with all the output specifics (and hopefully have an 100 on the lab), you can try optimizing and speeding things up.

possibleMoves() is a workhorse function; this means that in the final product of the othello labs, possibleMoves() will be called a LOT of times. Therefore, speeding this up will be very important.

Think about what you've learned in AI already. How do you speed things up?

First thing is to avoid recalculation. This is the main issue with slow code. You can work around this with lookup tables or just streamlining your code. Practice makes perfect, and you'll get better as you do more.

Second, you should understand what each loop does in your code. Sometimes, you'll be processing more values than you need to within a loop, and cutting out values you don't need will yield a decent time difference.