Assignment #8: The Othello Board Game

Due Date: 12/9/19, by the start of class
Points: 15
Assignment Type: Same team of 2 to 3 people as for Doggie

Introduction & Preparation:

In this assignment, you will be building an AI for the classic board game Othello, also known as Reversi. The Othello Framework is downloadable nearby. Unfortunately it does not have the networking capability I was hoping for, so we'll have to get by without it.

Please make every effort to have your game ready and playable in time for our class on Monday, December 9th, which is our last day of class. You can take a couple more days to get the final version onto LMS, but your first version needs to be there and also ready for class. Please have the game running on all your laptops so we can all be playing the game during class, which will be our version of Show and Tell for this course. We can also try out playing your versions of Doggie if we don't get a chance to do that prior to then.

The Framework has everything you need minus, of course, your AI. Note that one thing it does is to place the first four pieces in the center of the board automatically at the start of the game, and also that Black always goes first.

For this assignment, please build two AI's using related but different search tree algorithms. One could be Negamax, the other a more ambitious algorithm like AB Negamax or even Negascout. You should make available a public variable that stores MaxDepth, which of course your search depth must stop at. You will want this so that you can play with this setting.

Of course, get Negamax working first, then expand and extend it. You will likely find that, while getting it working is perhaps a bit tricky at first, it should be reasonably stable thereafter.

This project also requires you to build a good static evaluation function (SEF) to go with the tree search mechanism. Here you want to create the most powerful player that you can. Again, for this assignment, construct two such SEFs. That way, you could conceivable match either of your search trees with either of your SEFs.

For the SEF, there are many approaches you can take, and you can use some of these in combination:

Compute the greatest number of pieces of your color on the board as a measure of "score"

Maybe vary this by seeking a lower piece count when in mid-game, which is known to work better.

Note that corner squares are ultra-desirable.

Each of the squares could also be given its own desirability score, not just the corners.

You could introduce some randomness by choosing between identically scored best moves.

Do some research and see what else you can come up with.

Although transposition and “memory” mechanisms are also possible, they are not required for the assignment. But don’t let that stop you from giving them a try if you like.

There is a fair amount of commentary that you should check out in the Framework, particularly in BoardScript.cs.