Reversi Lab

Purpose
Build a game-playing program.

Task
Build a program to play Reversi (see https://en.wikipedia.org/wiki/Reversi). In your program, you must implement minimax search, alpha-beta pruning, and you must use an evaluation function to limit depth. You'll want to limit the amount of time your algorithm has to move so you can evaluate it, play against it, etc.

Partners
You can do the lab either by yourself or with a partner (encouraged). If you do it with a partner, make sure both of you contribute and learn.

Reversi Rules
You can find the rules for Reversi on the Wikipedia page linked above. There are some different versions of the rules, however, so here are some clarifications of what we will use in the class:

1. The first four moves (two for each player) must be in the four center squares on the game board. Opposite colors do not need to be diagonal to each other. No captures are made in the first four moves.
2. Each move after that must flip one or more discs of the opposing color. (That means you cannot put your discs in the corners if you cannot flip any of your opponent's discs by doing so.) If you cannot flip any of their discs, play goes to your opponent.
3. The game is over when the entire board fills up or neither player can make a legal move.

The results are implemented in the (Server) code provided. The code only allows legal moves and automatically calculates the score.

Creative part
After implementing minimax search, alpha-beta pruning, and a heuristic evaluation function, try other things to try to improve your algorithm. For example, you could experiment with a variety of heuristic evaluation functions, or you could select a game playing improvement described in Section 5.4 in the textbook, such as lookup tables or forward pruning.

Deliverables
Submit the following:

1. Your agent (client) code. We'll run a preliminary tournament so you can get a better feel for where you stand.
2. A written report (in PDF format) that includes:
    a. A description of your pruning and evaluation function.
    b. A declaration of time spent by each lab partner.
    c. A report evaluating your algorithm. This likely will include some kind of analysis of different variations on your algorithm. For example, did your creative work improve or hurt the performance of your algorithm? Did giving your algorithm more time to move improve its abilities? How deep in the tree could you search within X seconds? Does your computer code beat you? You can use videos (short), writing (a couple of pages), etc. to report your evaluations. Make it interesting – focus on things you would like to learn.

Grading

Here's a guideline for the grading of the lab.

- 50 points for demonstrating that you successfully implemented minimax search, alpha-beta pruning, and a heuristic function. How do we know that your implementation is correct?
- 25 points for implementing something else (creative portion).
- 25 points for effectively evaluating and improving your ideas.

In general, the overall quality of your work will supersede the technical specifications provided. In other words, do a good job of finding and evaluating (in your write-up) a good game-playing algorithm, and you should be in good shape.

Code

Later in the semester, we'll be conducting a class Reversi tournament for those who choose to enter. You'll each create an agent, and we'll see who builds the best agent (winning team gets an A in the class assuming they pass the final exam). The code you submit for this lab will not be your final code submission. You can continue to modify and improve your code throughout the semester.

In order to facilitate a class tournament, code is provided that allows you to connect two programs to a server and play the game. Code is provided in Java and Python – you can decide which language you would like to use. If you would like to use another language, you'll have to create the socket code, etc. for a client program that allows you to connect to the server and play the game. Your code must connect to and run with the Server provided. A human client is also provided so that you can play against your algorithm or against another person.

The supplied code contains the following four folders/programs:

1. ReversiServer – Java code that acts as the Server for two client programs. Client programs (e.g., your algorithm) connect to the server so that they can play games. You shouldn't have to change this code. But you need to compile it (command-line: javac *.java) on your computer, meaning you'll need to have Java installed.
2. ReversiHuman – Java code for a human client. It displays a GUI that a user can click on to make moves. You'll need to compile this code on your computer (javac *.java) before running it.
3. ReversiRandom_Java – Java code that is a random client. If you want to code your program in Java, then add to this code. Currently, it connects to the server and makes random moves when it is its turn. You'll need to compile this code on your computer (javac *.java) before running it.
4. ReversiRandom_Python – Same functionality as ReversiRandom_Java, but written in Python.

Suppose that I wanted to have a human player play against a random player, all running on the same computer. Then I would do the following:

1. Start the server: `java Reversi 10`
   Note that the parameter 10 specifies the number of minutes that each player has of move throughout the game.

2. Start player 1 (the Human player in this case): `java Human localhost 1`
   See the files for descriptions of the parameters

3. Start player 2 (the Random player): `java RandomGuy localhost 2`
   See the files for descriptions of the parameters

   To run the Python random player instead: `python RandomGuy.py localhost 2`