Adversarial Search: Othello

Due Oct 19, 2020 by 11:59pm **Points** 40 **Submitting** a file upload

Goals

The goals of this assignment are:

- Understanding adversarial search
- Implementing minimax and alpha-beta pruning

Background

Othello, also known by its non-trademark name reversi, is a strategy game for two players. You can find the **rules for the game on wikipedia.** **(http://en.wikipedia.org/wiki/Reversi)** There are also numerous sites to play online, and many free versions that you can download and play on your own computer.

I have written a **working version in python3** **(https://canvas.umn.edu/courses/193654/files/15527486/download?download_frd=1)** . If you want to use a different language, that's fine, feel free to adapt my classes and methods. The game rules themselves are a little tricky to implement, and I don't want you to spend too much time debugging those. (Note on the implementation: I haven't had time to test it thoroughly, I'm not certain that a game that ends with two players skipping will end properly. I'll try to test this Tuesday and then update if necessary.)

You can run the game as-is by running **python3 othellogame.py** on your command line. It will begin a game with two human players, who must take their moves by typing into the terminal.

Tasks

RandomPlayer

To help you understand the structure of the code, start by building a RandomPlayer class that will choose a random move from among all legal moves on its turn. Take a look at my OthelloPlayerTemplate class for guidance. This player will also come in handy when you design your "smarter" agents. If you can't beat random.... then you know something is not right.

Utility Function

Step one in building a **good** agent to play a game is to create a utility function. I have left that out of the code, and it is up to you to decide what a good utility is for this game. A very basic utility function might count the number of pieces you have and subtract the number of pieces the opponent has. This way, the more pieces you have on the board, the higher the utility. Additionally, you would want the utility function to report a terminal state with a win as a very high number utility and a terminal state with a loss as a very low number utility.

You will almost certainly want to redesign your utility function once you understand more about the game, so I recommend starting with something simple then coming back to it later.

Minimax Agent

Now that you have those pieces, create a **MinimaxPlayer** class that implements the minimax search algorithm to decide its moves. Have the depth limit of your MinimaxPlayer class be set as a parameter when the object is created. If you follow the python template, we should be able to create a player with the code

```
p1 = MinimaxPlayer(WHITE, 4)
```

that plays as white and has a search depth limit of 4 plies.

Alpha-Beta Agent

Finally, create an **AlphabetaPlayer** class that implements minimax search with alpha-beta pruning to decide its moves. Again, have the depth limit be set as a parameter when an object is created. It is possible that the only difference between results of Alpha-beta and Minimax will be the difference in time to arrive at a solution, but that means that it should be reasonable to run alpha-beta with a larger depth limit.

Grading

Submit your code via canvas. Grades will be given approximately as follows:

RandomPlayer - 5 points

Utility Function - 5 points

Minimax - 20 points

Alpha-beta - 10 points

Tournament

If you want to see how your agent stacks up against other agents in the class, then put your agent(s) that you want to compete in a separate module named **tournament_username.py**, where username is your U of M username (i.e. for me it would be *tournament_exle0002.py*). Your agents in the tournament must adhere to the following guidelines:

- The class must be named TourneyPlayer
- The class must have methods get_color and make_move
- The constructor for the class must take a color. You should use default values to set its ply limit to what you think is best
- Your agent cannot take longer than one minute on any given move. (As this may depend on the computer that it is running on, I'll tell you that I have a 2.8 GHz Quad-core with 16 GB of RAM.)
- If your agent takes too long or attempts to make an illegal move, it will receive an immediate forfeit of that game
- The agent will play as black and as white against each other agent. A win is worth 1 point, a loss is worth 0, and a draw is worth 1/2.

I will collect tournament submissions and try to run a round-robin tournament over the weekend after they are due.

hw3

Criteria	Ratings						Pts
RandomPlayer - 5 points	**5 pts Full Marks**	**4 pts minor mistakes**	**3 pts mistakes but correct idea**	**2 pts incorrect answer**	**0 pts No Marks**		5 pts
Utility Function - 5 points	**5 pts Full Marks**	**4 pts minor mistakes**	**3 pts mistakes but correct idea**	**2 pts incorrect answer**	**0 pts No Marks**		5 pts
Minimax - 20 points	**20 pts Full Marks**	**17 pts minor mistakes**	**14 pts mistakes but correct idea**	**8 pts major mistakes**	**6 pts incorrect answer**	**0 pts No Marks**	20 pts
Alpha-beta - 10 points	**10 pts Full Marks**	**8 pts minor mistakes**	**6 pts mistakes but correct idea**	**3 pts incorrect answer**	**0 pts No Marks**		10 pts
							Total Points: 40