Adversarial Search: RandOthello

New Attempt

- Due Oct 24 by 11:59pm
- Points 60
- Submitting a file upload

Goals

The goals of this assignment are:

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

You may work with a partner on this assignment

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.

We will be playing a variation of the game. At the game's start, we will choose a random square in the top row and a random square in the bottom row and block those out as unplayable locations.
I have **written a working version in python3 (https://canvas.umn.edu/courses/518032/files/53586547?wrap=1)** **(https://canvas.umn.edu/courses/518032/files/53586547/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.

You can run the game as-is by running **python3 randothellogame.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.

Eval Function

Step one in building a **good** agent to play a game is to create an evaluation function. I have left that out of the code, and it is up to you to decide what a good evaluation is for this game. A very basic 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 evaluation. Additionally, you would want the evaluation 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 evaluation function once you understand more about the game, so I recommend starting with something simple then coming back to it later.

Minimax Cutoff 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 Cutoff Agent

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. I **strongly recommend** that you closely follow the book's algorithm or code when implementing alpha-beta pruning. It is a tricky algorithm.

Advanced Agent

Create and AdvancedPlayer class that implements minimax with alpha-beta pruning and uses one of the other discussed techniques to increase the speed of its search: Move reordering, probabilistic pruning, monte-carlo tree search for evaluation, or some combination of all these.

Report

Write a report (submitted as a file named `report.pdf`) that details what your advanced agent is doing and why it is better then your alpha-beta cutoff agent.

Output and Submission

Set up your code so that by default it plays your AlphabetaPlayer (with a reasonable depth limit that doesn't take too long) against RandomPlayer, then swaps colors and does it again.

Submit your changed `randothello.py` module

Grading

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

RandomPlayer - 5 points

Evaluation Function for cutoff agents - 5 points

Minimax - 10 points

Alpha-beta - 10 points

Advanced agent - 20 points

Report - 10 points

Playing Each Other

The code is designed with some modularity in mind. If you write your agents as self-contained classes that inherit from the template class, then you should be able to test your agents against each other. For this assignment, this level of collaboration is allowed.

Hw3

Criteria	Ratings						Pts
RandomPlayer	**5 pts** **Full Marks**		**4 pts** **Minor Mistakes**		**2 pts** **Major mistakes**	**0 pts** **No Marks**	5 pts
Utility Function	**5 pts** **Full Marks**		**4 pts** **Minor Mistakes**		**2 pts** **Major mistakes**	**0 pts** **No Marks**	5 pts
Minimax agent	**10 pts** **Full Marks**	**9 pts** **Minor Mistakes**	**8 pts** **some non critical mistakes**	**6 pts** **Major mistakes, incorrect answer but somewhat good understanding of the algorithm**	**3 pts** **Major mistakes, incorrect attempt and understanding**	**0 pts** **No Marks**	10 pts
Alpha-beta	**10 pts** **Full Marks**	**9 pts** **Minor Mistakes**	**8 pts** **some non critical mistakes**	**6 pts** **Major mistakes, incorrect answer but somewhat good understanding of the algorithm**	**3 pts** **Major mistakes, incorrect attempt and understanding**	**0 pts** **No Marks**	10 pts
Advanced Agent	**20 pts** **Full Marks**	**18 pts** **Minor mistakes**	**16 pts** **Some non critical mistakes**	**8 pts** **Not much change from alpha-beta**	**0 pts** **No Marks**		20 pts
Report	**10 pts** **Good full description of advanced agent**		**6 pts** **Description is unclear**		**0 pts** **No report submitted**		10 pts
							Total Points: 60