4100/5100 Assignment 2: Othello Values
Due Friday May 22 9PM

Provided files: othello.py (stub), clearBestMove.txt, clearBestCounterMove.txt, arbitraryBoard5.txt, arbitraryBoard.txt, board1.txt, endgame.txt

In this assignment, you'll program a minimax module for the board game Othello, also known as Reversi. The main module that you'll program and test in HackerRank will calculate the value for a given board position. But, code is also provided for you that will let you play against your AI if you like.

The rules of Othello are as follows:
a) The two player colors are white and black. The white player goes first.
b) You capture an opponent's pieces when they lie in a straight line between a piece you already had on the board and a piece you just played. (A straight line is left-right, up-down, or a 45 degree diagonal.)
c) You can only play a piece that would capture at least one piece. If you have no legal moves, the turn is passed.
d) The game is over when neither player has any legal moves left. Whoever controls the most pieces on the board at that point wins.

Something that is slightly unusual about Othello for minimax is the fact that **a turn might be skipped** if a player has no legal plays. You'll have to take that into account in your minimax calculations. (Don't have skipped turns count against the search depth.)

The AI is always presumed to be white for this assignment; if you try the demo mode, you as the human will be playing black.

The input will be the search depth on a single line, followed by an ASCII representation of the board (W for white, B for black, - for an empty space).

A) Download the provided `othello.py` code.
B) Implement basic depth-limited minimax for the `minimax_value` function, ignoring the alpha and beta arguments for now. Your evaluation function, when you bottom out, should just be the difference in piece count between white and black if it's not the end of the game, or WIN_VAL, -WIN_VAL, or 0 if it is the end of the game. You should be able to effectively use a depth of 5 or so without waiting too long.
Tip: While debugging, you'll find it useful for minimax to print the board state it is evaluating and the value that it assigns to that board. Also, debug small depths before attempting larger ones.
C) Try running on the following boards that have a search depth of 5 or less:
`clearBestMove.txt`: Value should be 2.
`clearBestCounterMove.txt`: Value should be -3.
`arbitraryBoard5.txt`: Value should be 1.

Recall that you can feed a text file to a program with input redirection:

```
python3 othello.py < clearBestMove.txt
```

1) In your PDF, draw the game tree for `clearBestCounterMove.txt,` but only draw the nodes that would be evaluated under the following conditions:

*Alpha-beta pruning is being used.

*The first node that white evaluates as MAX is the move directly to its left horizontally (not the diagonal capture).

*In the two out of three cases for MIN's play where there is an obvious play that captures a lot of pieces, evaluate that move first. In the other branch, you can evaluate in an arbitrary order.

(You don't need to implement the order-of-evaluation rules in your code.)

Each node should show at least its minimax value under alpha-beta pruning, assuming a piece count evaluation function at depth 2, as well as a sketch of where the pieces are in relation to each other.

D) Implement alpha-beta pruning.

E) Check that you still get the same values on the files from part C.

2) Your code should now run in a reasonable amount of time on `arbitraryBoard.txt`, `board1.txt`, and `endgame.txt`. Submit the value of each board in your PDF. Note that endgame.txt will check that your code behaves reasonably when one player has no moves, and at least one of these tests will check that you're returning WIN_VAL or -WIN_VAL instead of a piece count difference when appropriate.

Optional: You can play against your AI if you just run `python3 othello.py` and type `play` at the prompt. Set `DEMO_SEARCH_DEPTH` to whatever degree of lookahead you have the patience for.

Othello is actually a somewhat tricky game to craft an evaluation function for; here we adopt the philosophy of not overthinking it.

<u>Submission Checklist:</u>

othello.py (with minimax_value fully coded)

Answers PDF