Adversarial Search: Minimax with Alpha Beta Pruning
Due Oct. 12th 2016

This assignment will give you the opportunity to gain expertise in adversarial search and to address issues in real-time AI decision making. You will implement minimax with alpha-beta pruning for Othello (aka reversi).

The Game

Using minimax with alpha-beta pruning and a reasonable evaluation function, you will create a game-playing AI for Othello. Your submission will contain a procedure named nextMove which will take 2 arguments:

- *boardState*: This is a nested list which represents the state of the board. Initially the board state is as follows:

```
[
['.', '.', '.', '.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.', '.', '.', '.'],
['.', '.', '.', 'W', 'B', '.', '.', '.'],
['.', '.', '.', 'B', 'W', '.', '.', '.'],
['.', '.', '.', '.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.', '.', '.', '.']
]
```

- *color*: This is either the string "B" or the string "W", representing the color to play.

Your *nextMove* function will return a tuple coordinate in the form *(x, y)* which will indicate the position your program wants to play. In service of this goal, you will need to come up with an evaluation function to score the board state. Below we have defined an example of a simple evaluation function for a board state given that the color being played is white.

```
def value(board):
    value = 0
    for row in board:
          for elem in row:
                if elem == "W":
                      value = value + 1
                elif elem == "B":
                      value = value - 1
    return value
```

It is also recommended that you set a depth that your minimax algorithm will search to before giving up. While there are no time constraints, if your algorithm takes too long to run it will be hard to determine if it has correct performance or if it is in an infinite loop.

Provided Files

The following files are provided to help you get started.

- *gamePlay.py*: Plays two agents against each other. If player1.py and player2.py were two python files that contain a nextMove function and -v was used to flag verbose output, at the command line the function would be invoked with:

  python gameplay.py [-v] player1 player2

- *randomPlay.py*: Sample agent that makes a random legal move.
- *simpleGreedy.py*: Sample agent that uses a brain-dead evaluation function with no search to make a legal move.

In *gamePlay.py* there are a number of useful helper functions defined for you that make it easier to test the outcomes of making a particular move. A particularly useful function is *gamePlay.validMove*, which you will need to use to determine if a potential move is valid or not. It is highly recommended that you look over this file before you start coding to make sure that you don't write a function that is already provided for you.

Grading

- ***100% of grade:*** A correct implementation of minimax with alpha-beta pruning
- ***Bonus 10%:*** A very creative or elaborate evaluation function

How to Submit

A single file named hw3.py containing your implementation of nextMove.