Programming Assignments 6 and 7 - miniChess
ECS 10 - Winter 2017

All solutions are to be written using Python 3. Make sure you provide comments including the file name, your name, and the date at the top of the file you submit. Also make sure to include appropriate docstrings for all functions.

The names of your functions must exactly match the names given in this assignment. The order of the parameters in your parameter list must exactly match the order given in this assignment.

For any given problem below, you will want to write additional functions other than those specified for your solution. That's fine with us.

miniChess is played on a 3 x 3 chessboard. The two players face each other across the board. Each player begins with three either three white pawns or three black pawns, placed one to a square in each of the three squares nearest that player. The player with the white pawns moves first. A player may choose to move one of his or her pawns in one of these ways:

> A pawn may be moved one square forward to an empty square
>
> A pawn may be moved one square diagonally forward to a square occupied by an opponent's pawn. The opponent's pawn is then removed.

The players alternate moving pawns until one of the players wins. A player wins when:

> A player's pawn has advanced to the other end of the board, or
>
> The opponent has no pawns remaining on the board, or
>
> It is the opponent's turn to move a pawn but is unable to do so.

As envisioned by its inventor, miniChess was intended to be played with only six pawns on a 3 x 3 board. Now, however, we are extending the definition of miniChess to include any similar game involving n white pawns, n black pawns, and a n x n board. Over the next two weeks, you are to construct the core functionality necessary for a computer program to play this game.

For this program, the current state of the game (i.e., the board) is represented as a list of lists. In the list of lists, the first sublist is the top row of the board, and the last sublist is the bottom row of the board. A 0 is an empty square, a 1 is a white pawn on the square, and a 2 is a black pawn on the square.

For example, a board that looks like this when displayed on your monitor:

```
- w w
w - -
b b b
```

will be represented in your program like this:

```
[[0,1,1],[1,0,0],[2,2,2]]
```

The core functionality that you will provide will take the form of two functions. The first function is the move generator, called `move_maker`. This function expects two arguments. The first argument is a list of lists representing the current board. The second argument indicates the color of the pawns that your program is moving: 1 indicates that your program controls the white pawns, and 2 says that your program controls the black pawns. Your function returns a list of the possible new boards that can result from the current board in one move. For example, if your program controls the black pawns, and the current board is the board shown above, then your `move_maker` function should behave like this:

```
>>> board = [[0, 1, 1], [1, 0, 0], [2, 2, 2]]
>>> move_maker(board, 2)
[[[0, 1, 1], [2, 0, 0], [2, 0, 2]], [[0, 1, 1], [1, 2, 0],
[2, 0, 2]], [[0, 1, 1], [1, 0, 2], [2, 2, 0]]]
```

As this example illustrates, black has three legal next moves given the current board. Your function should generate all three of these moves, but they won't necessarily appear in the same order as in this example, depending on how your function computes the possible next moves.

The second function is called `move_chooser` and expects two arguments. The first argument is a list of possible next moves returned by your `move_maker` function. The second argument indicates the color of the pawns controlled by your program. Your `move_chooser` function evaluates each of the boards/moves in the second argument and returns the best one. That board is your program's next move and becomes the new current board. Here's an example of how this function should behave:

```
>>> board = [[0, 1, 1], [1, 0, 0], [2, 2, 2]]
>>> mlist = move_maker(board, 2)
>>> mlist
[[[0, 1, 1], [2, 0, 0], [2, 0, 2]], [[0, 1, 1], [1, 2, 0], [2,
0, 2]], [[0, 1, 1], [1, 0, 2], [2, 2, 0]]]
>>> move_chooser(mlist, 2)
[[0, 1, 1], [2, 0, 0], [2, 0, 2]]
```

Part of the grade for your `move_chooser` function will be based on the sophistication of the algorithm you use for evaluating the individual boards. An overly simple board evaluator will get a lower grade than a board evaluator that takes into account a lot of information from the current board. In other words, the "smarter" your board evaluator is, the better your grade will be. You may want to use the board evaluation metric discussed in class as inspiration for your own.

Although we view your functions as two different programming assignments, you must submit both functions in the same file via SmartSite.

Some extra notes:

1) We may need to modify the specifications a bit here or there in case we forgot something. Try to be flexible.

2) Be sure to look at the PowerPoint slides for March 1. There are more miniChess examples there.

3) We are not asking you to write the human-computer interface so that your program can play against a human. We will supply you with the interface program.

4) Program early and often.

5) Your functions don't print, and they don't ask for keyboard input.

6) White always starts at the "top" of the board, and white always makes the first move.

Where to do the assignment

You can do this assignment on your own computer, or in the labs. In either case, use the IDLE development environment -- that's what we'll use when we grade your program. Put all the functions you created in a file called "prog67.py". Do not include your test data. Do not include the program that lets a human play against the computer. Include only the functions that we ask for.

Submitting the Assignment

We will be using SmartSite to turn in assignments. Go to SmartSite, go to ECS 010, and go to Assignments. Submit the single file containing all the functions you have created as an attachment. Do NOT cut-and-paste your functions into a text window. Do NOT hand in a screenshot of your functions' output. We want one file from you: "prog67.py".

Saving your work

If you are working in the lab, you will need to copy your program to your own flash-drive or save the program to your workspace on SmartSite. To save it on flash-drive, plug the flash-drive into the computer (your TAs or the staff in the labs can help you figure out how), open the flash-drive, and copy your work to it by moving the folder with your files from the Desktop onto the flash-drive. To copy the file to your SmartSite workspace, go to Workspace, select Resources, and then use the Add button next to "My Workspace". Your TAs can help you with this if you run into trouble.