ECS 170: Spring 2020
Homework Assignments 3 and 4 (scroll to the end for update)

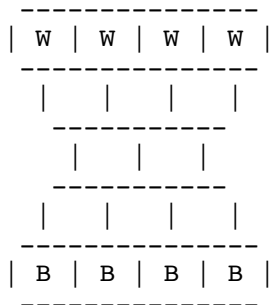**Due Date:**

Assignment 3 is due no later than Sunday, May 17, 2020, 11:59pm PDT.
Assignment 4 is due no later than Sunday, May 24, 2020, 11:59pm PDT.

The Assignment:

Consider the game of Oska, whose rules are as follows:

Oska is played on a board like that shown below. The two players face each other across the board. Each player begins with four pieces, placed one to a square in each of the four squares nearest that player.

```
 ---------------
| W | W | W | W |
 ---------------
  |   |   |   |
   -----------
   |   |   |
   -----------
  |   |   |   |
 ---------------
| B | B | B | B |
 ---------------
```

The player with the white pieces makes the first move, and the players alternate moves after that. A player may choose to move one of his or her pieces in one of these ways:

  A piece may be moved forward on the diagonal, one space at a time,
  to an empty space, as in checkers or draughts.

  A piece may jump forward on the diagonal over an opponent's piece
  to an empty space, thereby capturing the opponent's piece and
  removing it from the board. This capturing move is again like that
  seen in checkers or draughts. Unlike checkers, however, only a
  single capture is permitted on one turn; multiple jumps are not
  allowed. Also, even if a capture is possible, it does not have to
  be made if other moves are possible.

The players alternate moving pieces until one of the players wins. If a player can't make a legal move on a given turn, the player loses that turn and the opponent makes another move. A player wins when:

  All the opponent's pieces have been removed from the board, or

  All the player's REMAINING pieces have been moved to the opponent's

starting (or back) row. Note that this rule makes the strategy for
this game most interesting. A player may want to sacrifice pieces
in order to have fewer pieces to move to the opponent's starting
row. But this approach carries some risk in that every sacrificed
piece brings the player closer to having all of his or her pieces
removed from the board, thereby losing the game.

As envisioned by its inventor, Bryn Jones (with refinements by Michael Woodward, who made
this charming game available commercially), the game of Oska was intended to be played with
only four pieces on each side. Now, however, we are extending the definition of Oska to include
any similar game involving n white pieces, n black pieces, and a correspondingly larger board of
2n - 3 rows (where n is greater than or equal to 4). Thus, an
Oska game with 5 pieces on each side would look like this:

```
         ---  ---  ---  ---  ---
        | W | W | W | W | W |
         -------------------
          |   |   |   |   |
           ---------------
           |   |   |   |
            -----------
            |   |   |
            -----------
           |   |   |   |
           ---------------
          |   |   |   |   |
         -------------------
        | B | B | B | B | B |
         -------------------
```

Over the next three weeks, you are to construct a Python function (and many many supporting
functions, of course) which takes as input a representation of the state of an Oska game (i.e., a
board position), an indication as to which color (black or white) is being played by the function
you have created, and an integer representing the number of moves to look ahead. This function
returns as output the best next move that the designated player can make from that given board
position. The output should be represented as an Oska board position in the same format that
was used for the input board position.

Your function must select the best next move by using MiniMax search.

Assume for now that the name of your top-level function is "oskaplayer". Your function must
then be ready to accept exactly three parameters when called. The sample function call explains
what goes where in the argument list:

```
 oskaplayer(['wwww','---','--','---','bbbb'],'w',2)
                 ^                              ^   ^
                 |                              |   |
   The first argument is a list of 2n - 3      |   |
   strings.   Each of these elements is        |   |
   a row of the board.   The first element     |   |
   is the first or "top" row, and contains     |   |
```

```
n elements.  The second element is the          |  |
next row, and contains n - 1 elements.          |  |
The elements of each of these sub-              |  |
lists are either 'w' to indicate a white        |  |
piece on that square, 'b' to indicate a         |  |
black piece, or '-' to indicate an empty        |  |
square.  The leftmost element in one of         |  |
these nested lists represents the leftmost      |  |
square in the row represented by that           |  |
list, and so on.                                |  |
                                                |  |
The second argument will always be 'w' or --+   |
'b',to indicate whether your function is            |
playing the side of the white pieces or the         |
side of the black pieces.  There will never         |
be any other color used.                            |
                                                    |
The third argument is an integer to indicate --+
how many moves ahead your minimax search is to
look ahead.  Your function had better not look
any further than that.
```

This function should then return the next best move, according to your search function and static board evaluator. So, in this case, the function might return:

```
['www-', '--w', '--', '---', 'bbbb']
```

or some other board in this same format. That result corresponds to the following diagram:

```
    ---------------
   | W | W | W |   |
    ---------------
     |   |   | W |
      -----------
      |   |   |   |
      -----------
     |   |   |   |   |
    ---------------
   | B | B | B | B |
    ---------------
```

The deliverables for this task are due in two parts. First, we'd like to see your move generation function (Homework Assignment 3) by 11:59pm, Sunday, May 17. You can make upgrades to this function after that, but you must turn in a credible working version of the move generator by 11:59pm, Sunday, May 17. The move generator function takes as input only two arguments, a board and an indication as to whose turn it is next ('b' or 'w'), in exactly that order, and returns a list of all new boards that can be generated in one move by the indicated player.

Then, 11:59pm, Sunday, May 24, we'd like to see your entire Oska program (Homework Assignment 4) – move generation, minimax control, and your static board evaluator, all merged into one impressive Oska machine.

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. Don't complain.

2) A static board evaluation function is exactly that -- static. It doesn't search ahead. Ever.

3) You can convert our board representation to anything you want, just as long as when we talk to your functions or they talk to us, those functions are using our representation. Thus the interface to your move generator must be in terms of our board representation, and the interface to your top level function must be in terms of our board representation.

4) We are not asking you to write the human-computer interface so that your program can play against a human. You can if you want to, just for fun, but we don't want to see it.

5) Program early and often. A simple board evaluator is easy. The move generator is harder, but you can adapt some of the peg puzzle program The minimax mechanism is harder still. Get the move generator done in a hurry, then start working on everything else as soon as possible.

6) Name your top-level game-playing function "oskaplayer", just like in the example above. Name your move generator function "movegen". Use those exact names. If the TAs try to run your program by calling "oskaplayer" and your function is named something else, you will lose many many points. Calling your move generator something other than "movegen" will have a similar outcome.

7) You may use any representation your wish within your functions, but for testing and grading purposes, the arguments to your "oskaplayer" and "movegen" functions, as well as what those functions return, must use the representations specified above.

8) You have two weeks to finish the move generator. You already have plenty of experience creating move generators, so it should not take you two weeks to make a move generator. Get that component done early and start working on the rest of program. Do not wait until the last minute. Start working on it now.

9) Before writing any code, play the game a few times.

10) If the final move of the game leaves both players with all their remaining pieces on their opponent's starting (or back) row, the player with the most pieces remaining wins. If both players have the same number of pieces, the game is a draw.

11) Just like with Homework Assignment 2, we want to see lots of well-abstracted, well-named, cohesive, and readable functions in Homework Assignments 3 and 4. Comments are required at the beginning of every function and must provide a brief description of what the function does, what arguments are expected by the function, and what is returned by the function. Missing comments result in missing points. Additional comments that help the TA who grades your program understand how your program works may make the TA happy, and happy graders are generous graders.