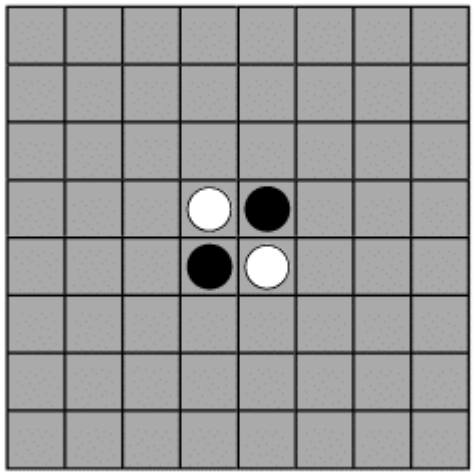
CSC447/547 Artificial Intelligence - Spring 2016
Programming Assignment #3: Othello

Introduction
Othello (also known as Reversi) is played on an 8x8 grid with 64 stones that are black on one side and white on the other. The start position is:



The players (Black and White) alternate placing stones on the board. Stones must be placed to surround enemy pieces. When a row, column, or diagonal of contiguous white pieces are surrounded on each end by black pieces, the white pieces are "flipped" and become black pieces (or vice-versa). A player who cannot place a stone to capture enemy pieces must forfeit that turn.

For example, Black might begin by placing a stone in row 3, column 4. The white stone on row 4, column 4 is flipped as a result of this move, leaving 4 black stones and 1 white stone.

Play continues until the board is filled, or neither player can make a legal move. At this point the stones are counted and the player with the most stones wins.

Problem
Write a Lisp program to play the game Othello. Your program should allow both interactive play against a human opponent, and non-interactive play against a computer opponent.

Implementation
This assignment is designed to investigate AI game playing techniques. You must use *minimax with α-β pruning* in this program.

Your Lisp program (*othello.lsp*) may be run interactively from the command line:

```
clisp othello.lsp player (Black or White)
```

or from inside CLISP:

```
(load 'othello)
(othello [player])
```

Ask the user if they wish to move first (unless they have supplied this info in the parameter list), and then start the game. Print a picture of the board after every move (see example session below). When the game is over, congratulate the winner and print the score (how many tiles for each side). Then offer an opportunity for revenge!

In order to compete against other Othello programs in a computer tournament, your program should provide a *make-move* function: `(make-move position player ply)`

This function takes three arguments: the current board position, which player (B/W) has the next move, and the depth of search (number of moves, or ply, to look ahead in the game tree). It returns a (row col) list that specifies the selected move. For example, the game might begin with the following call:

```
(make-move '( - - - - - - - -
              - - - - - - - -
              - - - - - - - -
              - - - W B - - -
              - - - B W - - -
              - - - - - - - -
              - - - - - - - -
              - - - - - - - - ) 'B 4)
```

Following the example session below, the returned move might be `'(3 4)`.

You must also provide a function named *othello-init*, which takes no arguments. This function will be called once, prior to the start of tournament play. You may place initialization code in this routine. If you have no initialization tasks, then provide an empty *othello-init* function.

This will allow two programs to compete against one another in a computer Othello tournament. Prizes will be awarded for the tournament winners!

This program is designed to investigate AI game playing techniques. You must use *minimax with α-β pruning* in this program. More efficient implementations with more sophisticated static evaluation functions will be awarded greater credit.

Googling Othello or Reversi brings up links with rules, strategies, and online games. (Please do not download source code!) Here are some Web sites to get you started:

https://en.wikipedia.org/wiki/Reversi
http://www.wikihow.com/Play-Othello

Notes

- When you are finished writing, testing, and debugging your program, submit your source code in a *zip* or *tar* archive using the *Submit It!* link on the MCS Department Website. The submit program is accessed via the MCS Web page (http://www.mcs.sdsmt.edu), by selecting the list item on the left entitled "Submit it!". Usage is self-explanatory: enter your name, choose the instructor and click "Select Instructor", choose the appropriate course, browse to the filename you wish to submit, and click "Upload".
- Submit your program by the due date (Thursday April 21) in order to receive credit for this assignment. Late programs will not be accepted for partial credit unless prior arrangements have been made with the instructor. If you have any problems with the submit program, report them to your instructor and submit your program by email instead.
- To receive full credit, your code must be readable, modular, nicely formatted, and adequately documented, as well as complete and correct. It must build and run successfully under the current CLISP interpreter (v.2.49). If your program does not run correctly, indicate why. This will make it easier to give you partial credit.
- You must work in teams of two students on this assignment. Teams should make one joint submission, not individual submissions for each team member. Each team member should complete and submit a teamwork evaluation form as well, assessing distribution of workload and team interactions. Teams for this assignment are:

    1. Johnathan Ackerman + Jacob St.Amand
    2. Hannah Aker + Alex Herman
    3. Jason Anderson + Mackenzie Smith
    4. Stephanie Athow + Daniel Nix
    5. Marcus Berger + Benjamin Kaiser
    6. Allison Bodvig + Scott Carda
    7. Julian Brackins + Marcus Haberling
    8. Samuel Carroll + Leif Torgersen
    9. Matthew Dyke + Christian Sieh
    10. Dylan Geyer + Luke Meyer
    11. Steven Huerta + Matthew Rames
    12. John Mangold + Cassidy Vollmer
    13. Alex Nienhueser + Savoy Schuler
    14. Derek Lane + instructor

Sample LISP session (user responses in **boldface**):

```
> (othello)

Would you like to move first [y/n]? y

OK! You will be playing Black. When asked for your move, please enter the row
and column in which you would like to place a Black stone. Remember, you must
outflank at least one White stone, or forfeit your move.

     1 2 3 4 5 6 7 8
   1 - - - - - - - -
   2 - - - - - - - -
   3 - - - - - - - -
   4 - - - W B - - -
   5 - - - B W - - -
   6 - - - - - - - -
   7 - - - - - - - -
   8 - - - - - - - -

What is your move [row col]? 3 4

     1 2 3 4 5 6 7 8
   1 - - - - - - - -
   2 - - - - - - - -
   3 - - - B - - - -
   4 - - - B B - - -
   5 - - - B W - - -
   6 - - - - - - - -
   7 - - - - - - - -
   8 - - - - - - - -

Here is my move: 5 3

     1 2 3 4 5 6 7 8
   1 - - - - - - - -
   2 - - - - - - - -
   3 - - - B - - - -
   4 - - - B B - - -
   5 - - W W W - - -
   6 - - - - - - - -
   7 - - - - - - - -
   8 - - - - - - - -

What is your move [row col]?

      etc.
```