CS-GY 6613 **Spring 2016**
Project: 5 x 5 Lines of Actions **E. K. Wong**

Total points = 100. Extra credits: up to 40 points.

Project Description: Design and implement a 5 *x 5 Lines of Actions* interactive game for a human to play against a computer. The rules of the game are as described below. You can refer to https://en.wikipedia.org/wiki/Lines_of_Action for a description of the 8 x 8 version of the game.

Rules of Game: The game board consists of 5 x 5 squares with a checker-board pattern. There are 6 black pieces and 6 white pieces, initially arranged along the four sides of the game board as shown in Figure 1.

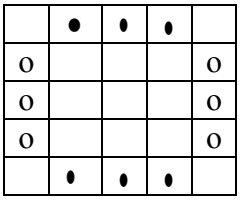

Figure 1. Initial Game Board Configuration

- The first player to bring all of his/her checkers together into an 8-connected contiguous body wins. If a player has only one checker left, he/she also wins.
- Black player moves first and the 2 players alternate moves.
- Checkers can move horizontally, vertically, or diagonally.
- A checker moves exactly as many spaces as there are checkers (both friendly and enemy) on the line in which it is moving.
- A checker may jump over friendly checkers, but not over enemy checkers.
- A checker may capture an enemy checker by landing on the current position of the enemy checker.
- If after a capturing move, both players have their pieces in a contiguous body, then the player who makes the move wins.

Please see https://en.wikipedia.org/wiki/Lines_of_Action for graphical illustrations of the various moves.

Implementation*:* Your program should have the following features:

- The human player can choose to be the black player or the white player. The black player moves first when the game starts.
- The program declares the winner by displaying a message on the screen.

In designing your program, you should use the *Alpha-Beta-Search Algorithm* in Figure 2 on page 3 of this project description. If the computer cannot search all the way to the bottom (and reach a terminal state) in 10 seconds or less for a move, you should cutoff the search and use an evaluation function to estimate a measure of desirability for the board configuration. Design an evaluation function for the game if it is necessary to use cutoff. (Suggestion: define 3 terminal states: black player wins with a utility value of +100, white player wins with a utility value of -100, and draw with a utility value of 0.)

Every time your program calls the Alpha-Beta-Search function to return an action, compute and display the following statistics on the computer screen: (1) maximum depth of game tree reached (let the root node be level 0), (2) total number of nodes generated (including root node) in the tree, (3) number of times the evaluation function was called within the MAX-VALUE function, (4) number of times the evaluation function was called within the MIN-VALUE function, (5) number of times pruning occurred within the MAX-VALUE function, and (6) number of times pruning occurred within the MIN-VALUE function.

You can use any high-level language such as Python, C/C++, C#, or Java to do the project. If you plan to use a language other than those listed, send me an e-mail first. I just want to make sure that the language you choose is appropriate for the project.

Extra Credits: The following parts are *optional*. You can implement one or more of the following to get up to 40 points in extra credits. If you implement multiple parts of the following and the total number of points you get exceeds 40, you will still get 40 points (as a maximum) for extra credits.

1. [15 points] Design and implement a graphical user interface for displaying the game board and allow the human player to make moves by clicking with the mouse.
2. [10 points] In the design above, the computer plays the best strategy. Design your program in such a way that the computer does not always play the best strategy so that it is easier for the human to win. The human player can choose different levels of difficulty to play (say from 1 to 3).
3. [15 points] If you use an evaluation function, you may get up to 15 points in extra credits if the design of your evaluation function is exceptionally good. Generally, you do not get extra credits for this part if your evaluation function is just an average and a reasonable one but not exceptional. In case there is no need to use an evaluation function, then you do not get extra credits for this part, of course.
4. [15 points] Extend your game to a 6 x 6 game board with 8 pieces of checkers for each player. The rules are the same as the 5 x 5 version described above. If you do the 6 x 6 version for extra credits, you are still required to do the 5 x 5 version above.

What to hand in:

- Source code for your program. Documentation and in-line comments are required for your source code (Points will be taken off if you do not have this.)
- A report (in MS Words or PDF format) that contains:
    a. The programming language and compiler you used. Instructions on how to compile and run your program.
    b. A high level description of your design and program. If you use cutoff and evaluation function, describe the heuristics that you use in your evaluation function and how they work. If you implement different levels of difficulty for extra credits, describe the method you use to implement it. No need to describe the basic *alpha-beta* algorithm. However, if you modify the alpha-beta algorithm, you need to describe your modifications and explain why you modified it.
- Upload your files to NYU Classes by the due date.

Demo: After you submit your source code and documentation, a 15-minute demo of your program will be required to show that your program works. Bring a hardcopy of your report and source code during your demo. Details about the demo (when and where) will be announced later.

****

```
function ALPHA-BETA-SEARCH(state) returns an action
        v ← MAX-VALUE(state, minimum_utility, maximum_utility,0)
        return the action in ACTIONS(state) with value v

    function MAX-VALUE(state, α, β,depth) returns a utility value
        if TERMINAL-TEST(state) then return UTILITY(state)
        if CUTOFF-TEST(state, depth) then return EVAL(state)
        v ← -∞
        for the next a in ACTIONS(state) do
            v ← MAX(v, MIN-VALUE(RESULT(state, a), α, β, depth+1))
            if v ≥ β then return v
            α ← MAX(α, v)
        return v

    function MIN-VALUE(state, α, β,depth) returns a utility value
        if TERMINAL-TEST(state) then return UTILITY(state)
        v ← +∞
        for the next a in ACTIONS(state) do
            v ← MIN(v, MAX-VALUE(RESULT(state, a), α, β,depth+1))
            if v ≤ α then return v
            β ← MIN(β, v)
        return v
```

Figure 2. Alpha-Beta Algorithm