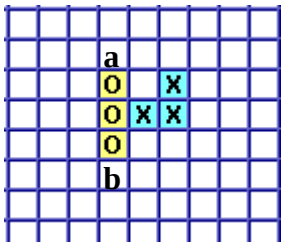
Programming Project #1:
Game-Playing Search and Gomoku
Due Oct 26, 2pm

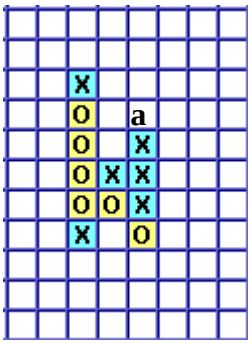
In this programming project assignment, you will develop a game-playing program that is able to beat you in an ancient version of tic-tac-toe. *Gomoku* is an ancient oriental game, traditionally played on a GO board (of size 19x19), whose object is to get 5-in-a-row in any direction. It is considerably more difficult than tic-tac-toe, because of the size of the board, and thus, the size of the search space. To get a feel for this game, I would recommend playing a few matches against the computer: http://www.ics.uci.edu/~gennari/AI-171/gomoku/game.html This implementation shows the moves as colored stones, rather than the Xs and Os of tic-tac-toe. "Gomoku" means "5 stones" in Japanese and a GO board comes with black and white stones.

For this assignment, you must implement a ***mini-max*** search algorithm, employing the ***alpha-beta*** cut-off strategy to reduce the size of the search space. I expect that your algorithm should search to a depth of about 6 moves (3 for each player), but this should be an input parameter. (For debugging, I recommend starting with maxdepth = 4.) It should be clear that any reasonable depth search is impossible if you inspect all 360 positions on the board. Instead, limit your search to all adjacent positions of the placed stones (including diagonals). With only one stone played, this limits the choices to only 8 moves, but this branching factor rapidly increases. For example, the first board position shown below has 17 choices for the next move, and the last position has 27 choices.

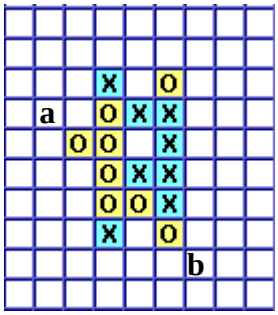
One of your challenges is to construct an effective *evaluation function* to score board positions. The general idea is to look for patterns. Consider the following three successive board positions; in each case, **X** is about to play:



O has an "open three" (uncapped). If **X** does not play at either *a* or *b*, then **O** will win.



O has another "open three", but **X** can make a capped four. Before blocking **O**, **X** should first play at *a*.



O has an "open four". Although **X** has an open three, **X** will lose, since **O** wins at either *a* or *b*.

This suggests creating an evaluation function that scores positions by looking for four types of patterns: open threes, open fours, capped fours, and of course, winning positions. Don't forget that making an open three for yourself should not count as much as blocking an open three of your opponent.

Requirements:

- Your program must play Gomoku successfully—in particular, if given the chance, it must choose to play open threes and open fours. In addition, at least for the first 10 moves or so, your program must make a move in less than 3 minutes.
- Your program must employ mini-max searching and the alpha-beta cutoff algorithm. You must code the search depth as an input parameter. After each turn, you should report (and display) the number of board positions evaluated by your system.
- You must be able to *turn off* alpha-beta pruning, to demonstrate that doing so increases the number of board positions that your system needs to evaluate. The user interface should show whether or not the system is using alpha-beta cutoffs.
- Your project report must describe the exponential growth in the search space as the depth is increased. Your report must also explain how (or if) this is mitigated by the use of alpha-beta pruning. Your report should also describe the details of your evaluation function, and how your program matches patterns against board positions.

Division of labor:

As described on the web pages, each team has two programmers, a team leader/manager, and a documentation leader. For this project, one way to divide up the work between the two programmers is to have a UI programmer and an algorithm programmer. The former will be responsible for all aspects of the user interface: Creating the board, displaying stones, (or Xs and Os), displaying the number of board positions evaluated, and buttons such as "New game?". The latter will be responsible for implementing the evaluation function, the mini-max search algorithm and alpha-beta pruning. This division probably puts too much burden on the algorithm programmer, so probably the UI programmer should be expected to finish early and then help out during the testing and debugging stages of the algorithm and evaluation function programming.

Note that both programmers will need to manipulate a datastructure for the board and stones. This should be designed as a team, and the team leader should make final decisions on such common data structures. In addition, the team should probably design and agree on the evaluation function used by your mini-max algorithm. Remember that the evaluation function is called at the terminal nodes of your depth-limited minimax search: thus, it is quite important that this function be as efficient as possible. Earlier, I suggested an evaluation function that looks for four different patterns – even this can be somewhat expensive to do since you'll have to consider all of the placed stones, and all of the directions for each stone (vertical, horizontal, and 2 diagonals).

Reminder: no late projects! Therefore, you should plan to build your system so that *something* works as soon as possible.