

CIS*3700 (Winter 2007) Assignment Two

Instructor: F. Song

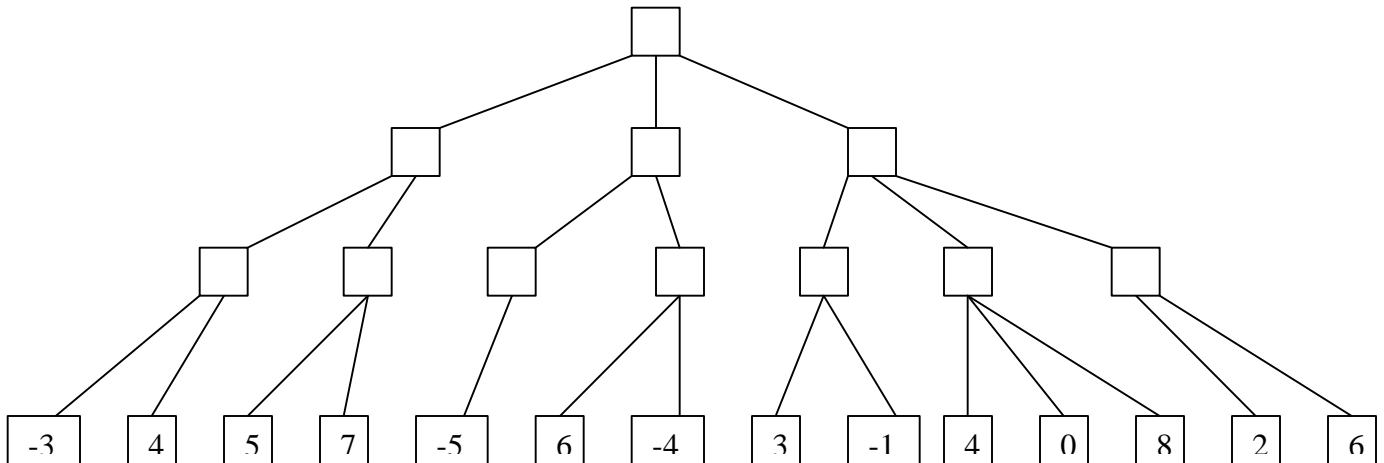
Due Time: *Paper-and-pencil questions due on Feb. 15, right after the class and the implementation questions due on Mar. 4, by midnight.*

Part I: Paper-and-Pencil Questions (30 marks)

- One utility function for the Tic-Tac-Toe game is defined as follows. Let X_n be the number of rows, columns, and diagonals with exactly n X's and no O's. Similarly, let O_n be the number of rows, columns, and diagonals with just n O's. The utility function assigns (+10) to any configuration with $X_3 = 1$, (-10) to any configuration with $O_3 = 1$, and 0 to all other terminal configurations. For any intermediate configuration, we can use the following linear evaluation function:

$$\text{Eval} = 3X_2 + X_1 - (3O_2 + O_1)$$

- Approximately how many possible games are there for Tic-Tac-Toe?
 - Show the whole game tree starting from the empty board down to depth 2 (i.e., one X and one O on the board), taking symmetry into account. You should have three configurations at level one and 12 configurations at level two.
 - Mark on your tree the evaluations of all the configurations at level two.
 - Mark on your tree the backed-up values for the positions at levels one and zero, using the MiniMax algorithm, and then decide the best move from the starting position.
- Circle those nodes that would not be evaluated by the alpha-beta pruning procedure, along with those backup values necessary for the computation.



Part II: Implementation Questions (70 marks)

3. Game trees provide a useful representation for two-player perfect-information games, where the two players alternate in making moves and there are no chance elements such as a dice. For this exercise, you are asked to implement the MiniMax search strategy along with alpha-beta pruning. Then, you should test your implementation on the Tic-Tac-Toe game. The user (human opponent) should have an option of playing against the program at different levels of competence, which can simply be implemented by restricting the number of ply in the look-ahead search process. A nice interface is not necessarily required; far more important is to document your program properly and make sure the game board is clearly represented and easily understood.

The pseudocode for alpha-beta pruning is described in the lecture notes. For a general description of MiniMax search with alpha-beta pruning, see the textbook or any other general AI books for more information.

In addition to the basic MiniMax search with alpha-beta pruning, you are also asked to implement other possible improvements described in the lecture notes. In particular, rotations of the board are helpful for the first couple of ply in the game tree representation, and you can also explore different utility functions for the Tic-Tac-Toe game.

Submission requirements for the implementation question(s):

Your implementation can be done in either C++/STL or Java so that you can take advantages of the support for object-oriented programming and data structures. In addition, your programs should run in a Linux machine such as those in Reynolds 004/008 labs. Your program will be marked for both correctness and style. In particular, a complete submission should include: (i) a README file, describing what has been done, what are the limitations, what improvements could be made if you were to do it again, how a user can use it, and how it is tested for correctness; (ii) a Makefile that allows for “make clean” and “make” so that we can create a new build for marking; and (iii) all the source code and testing files. Tar and gzip all of your files into one compressed file (name it in the form of <userid>_a<#>.tar.gz, e.g., fsong_a2.tar.gz) and email it to ta3700@cis.uoguelph.ca by the due time.

Important Note: You should carefully test your program; incomplete code and code that doesn't compile will face a big penalty.