omework Assignment 2A**:**
eleased 10/12/01
ue 10/19/01

. Write a program that plays tic-tac-toe using game tree search.

Tic-tac-toe is such a small game that it would not be hard to simply write an ordinary, deterministic program that plays perfectly; but the point of this exercise is to be a warm-up for chess. You should design your data structures and code so it can be easily reused for chess.

In particular, your program should include:

- **A move generator routine**, that can be called to incrementally generate the children of a node. In tic-tac-toe this will enumerate all of the possible places to put an X or an O in some defined order. In chess this will enumerate all of the possible ways to move the chess pieces. This will determine the branching factor of your game tree.
- **A static evaluation function**. Although it won't be very useful in tic-tac-toe, it will be critical to your chess program performance. The static evaluation function can be any of the following as specified on the command line:
  (1) endgame == return 1 for a win, -1 for a loss, 0 otherwise
  (2) center == assign a value to each square, take difference
  of sum of your squares - sum of opponents
  squares, where
  center = 3 points
  corner = 2 points
  middle = 1 points
  win = +100
  loss = -100
  (3) squeezeplay == +2 for a win, +1 if you have *two*
  different ways of winning on the *following* turn,
  and vice-versa for opponent.
- **Alpha-beta pruning.**
- **A text file output,** that will print information after each move and at the end of the game on the number of nodes searched and the execution time.
- **An option to trace the execution of the program**, so you can reconstruct the game tree searched.

Other details:

* You should work in teams of 2 or 3 people.
* Programming language doesn't matter as long as the program can be run from the command line on a grad Linux box. The executable should be called "ttt". All debug/trace output should be put in a file called "ttt_output.txt" The executable should take the following command line options:
  + "-ply N" the maximum number of plies to search on your turn. i.e. how deep in your tree you may search. If N = 0 there is no limit to how deep in the tree you may search. e.g. N=1 means you may evaluate the board position after you move once. N=2 means you may look ahead to the board position after you move once and the opponent moves once.
  + "-node N" the maximum number of nodes that you may explore. if N=0 there is no

limit. if N=1 you may consider one option for your move and no others. if N=2 you may consider either two options for your move, or one option for your move and one response from the opponent. (Having a small N doesn't really make a lot of sense.) Both -ply N and -node N may be given to your program at the same time.

+ "-traceoff" This will tell your program not to output a game trace. By default trace should be on.

+ "-sef [endgame,center,squeezeplay]" Set the static evaluation function to be used in the game to one of the four listed.

* X always goes first.
* Communication should take place via stdin and stdout pipes, from and to a server program that will manage the communication between games. Your program should receive information about the move that the opponent took, as well as some simple commands via stdin. You should communicate your moves through stdout. Don will provide a Perl script that will allow two tic-tac-toe programs to play eachother. All communications should end with a newline character. Make sure that you aren't buffering your output to stdout. **You should not have to do any networking programming for this assignment.**

Messages that you might receive are:

* **NX** to indicate a new game in which you will be X. The response will be your move in the format below.
* **NOX[A,B,C][1,2,3]** to indicate a new game in which you will be O and the first move of X is implied by the remainder of the input. Your response will be the second move of the game. For example you might receive the command NOXB2 indicating that this is a new game in which you are O and the X player moves to position B2.
* **[X,O][A,B,C][1,2,3]** to indicate your opponent's move: as in "XA1", the response should be your move in the same format.
* **L** to indicate the previous game is over and you lost. Reponse is "A" for acknowledge.
* **W** to indicate the previous game is over and you won. Response is "A" for acknowledge.
* **Q** to indicate that your program should quit. No response required just quit.

Messages that you might send are:

* **E** to indicate an error or that you resign. This will be treated as a forfeit. The program trace can then be evaluated to see what happened.
* **[X,O][A,B,C][1,2,3]** to indicate your move: as in "XA1".
* **A** to indicate that you acknowledge a win or a loss.