COMP30260 "Artificial Intelligence for Games & Puzzles"
1st Programming Assignment (of two in total)

Announced Thursday 2 October 2014. **Due Friday 31 October before midnight.**
This assignment is worth 20% of the marks for the module.

SCOUT search with iterative deepening and reordering

There are four programming steps to be carried out, in any programming language you choose, plus a writeup of approximately two to five pages. A tutorial will take place on October 13.

1) Generate a fixed-width fixed-depth game tree (30%)

Write a recursive routine (ie method or procedure or function or whatever, depending on your choice of programming language) to generate a tree of nodes, stored in memory. This function should create a node, and except at leaf nodes, call itself recursively to create subtrees which it then attaches to that node. The top node *is* the tree: there is no need for separate data types for tree and node.

There should be at least five parameters, all integers, all except ***v*** must be strictly positive:
b to control the branching factor,
d to control the depth of tree,
v to control the negamaxed value of an interior node, and the static evaluation value of a leaf,
i to control the inaccuracy of a hypothetical static evaluation function for a non-leaf node,
s to control the spread of values of the children of any parent.

Your main program should prompt the user for values for the top node.

Each node in general has an ordered set of child nodes, and an integer *estimated* score. This is distinct from the integer *actual* score, which is not necessarily stored with the node, but is used in building the tree; later on, the search should rediscover this actual score.

The top node is at height ***d***. Its child nodes, all at height ***d-1***, should have a range of actual scores. The minimum of the actual scores should agree with the negation of the parent's score. The child node with this score should be randomly placed among its parent's children. Other children of the same parent may have actual scores which are greater than this, by a random amount up to ***s***. The estimated score of a leaf node should be the same as its actual score. The estimated value of an interior node should be its actual score plus or minus a random amount up to ***i***.

You should be able to generate trees with branching factor at least 4 and depth at least 8.
Hint: Start off with very small tree; go to b=4 d=8 size (or beyond) when all else is ready.

> Note that this construction and retention of a tree is not typical of what game-playing programs do. Normally there is a game position for which the game rules specify what moves are possible, and these moves are "made" and "unmade" in a depth-first fashion. Here there is no position, and no rules, only random numbers; and it is desirable to compare the performance of several algorithms on the same trees. That is why the trees must be stored in memory, and hence why they must be relatively small (maybe millions but not billions of nodes) for the experiment.

2) Implement negamax variant of alpha-beta (10%)

Implement a simple negamax alpha-beta algorithm, with minor enhancements to

a) keep count of the number of static evaluations performed and
b) use an additional boolean parameter which determines whether to print out information about values of depth and alpha and beta, tell when cutoffs occur, what values are returned
c) return both the score for a node, and for an interior node, the child from which that score came. (Where alpha-beta is called with depth=0, and so does static evaluation, this is null.)

The static evaluation of nodes at the horizon is trivial - retrieve the node's preassigned estimated score. The (parameter-controlled) printing facility should allow informative messages to be printed at a suitable level of indentation whenever a leaf node is evaluated and whenever an interior node's value is being calculated. The messages should identify the node, its level, the values of alpha & beta, the value calculated and the method (static or search) of calculating it.

3) Implement the SCOUT Search algorithm (15%)

This algorithm is described in lecture notes. It involves using alpha-beta with null window repeatedly to perform a binary-chop search of the range of possible game-tree values. (This range will depend on your chosen values for ***v*** ***s*** ***d*** and ***i***: v+i+s*d, v-i-s*d) Try the algorithm out on small trees with printing in the basic alpha-beta algorithm enabled, then try it out on bigger trees only counting static evaluations, with alpha-beta's printing disabled.

4) Implement iterative deepening and principal-variation reordering (20%)

Modify the alpha-beta routine to report the principal variation (in the form of a list of child node index numbers, for example), as well as the negamaxed value of a node; and modify it to accept a principal variation list, and try out the principal variation first when making its recursive calls. Create an iterative-deepening routine which can call either of SCOUT and plain alpha-beta, capturing the principal variation from one search and passing it in to be used in the next-deeper search. Whether it does pass on the principal variation or not should be determined by a boolean parameter.

5) Testing and Writeup (25%)

Compare the results of SCOUT search on big trees with the results of pure alpha-beta on those same trees. Generate ten trees of the same large size. For each tree, apply simple alpha-beta and also SCOUT, always using iterative deepening, and with or without principal variation reordering.

Present the results in tables (and preferably also in graphs), showing counts of evaluations and results returned by searches (which may not always be 'correct' results). Report on what you did, how you did it, what you learned.

Do's and Don'ts

Do Re-read The Do's and Don'ts just before you submit!

***Do not* include files produced by a compiler, such as exe files or java .class files**: only your source files, report and results are required.

***Do not* present your writeup and appendix as anything other than plain text or pdf.** No .doc or .docx or .rtf other word processor specific format should be submitted.

***Do not* submit a .rar, .jar, .7z, or any other form of archive except .zip.**

***Do not* place folders inside folders, even within a .zip**. Make just one flat folder containing only the necessary files. Do this even if your development environment has made a bin and a src folder for you.

***Do not* provide more than one version of your code.**

***Do not* submit your assignment by email, use only the moodle**.

***Do not* present lines of program code or comment having more than 120 characters.**

***Do not* give your submission a long or unhelpful filename. Preferably, use the pattern FirstnameLastname.zip – so mine would be ArthurCater.zip – and make it unzip into a folder with the same name but without the .zip suffix.**

Each of the above *don'ts* violated will attract a penalty of 5%.

***Do* put your name and student number in the first few lines of the report, and also in comments in the first few lines of each source-code file.**

Do remember that it is much better to submit assignment work even if it is incomplete and late (by up to two weeks) than it is to submit nothing at all.

Do not commit or assist plagiarism.