COMP30260 "Artificial Intelligence for Games & Puzzles"
1st Programming Assignment

Announced Thursday 5 October 2017. Due 7pm Friday 3 November.
This assignment is worth 20% of the marks for the module.

NegaScout/PVS with Principal Variation Reordering

Each student must complete this assignment independently. While you are positively encouraged to discuss the general issues and ideas that arise, you must not help anyone else (or allow anyone to help you) with coding or any other specific steps in arriving at your solution.

In accordance with UCD policy, plagiarism will be dealt with harshly.
If you have any questions about this policy, please inquire.

Overview

There are five programming steps to be carried out, in any programming language of your choice. You must also write a report around three pages long and produce tables of numerical results.

Using any programming language you choose, you should write a program to simulate negamax-style PVS search with iterative deepening, and compare it with negamax-style alpha-beta ("NABS") search with iterative deepening. In addition, both search methods should be supplemented with an option to use Principal Variation Reordering.

There are no game rules to be programmed or positions to be represented. The program should construct an explicit game tree that will later be searched. It should use a random number generator for two purposes: to simulate results of a static evaluation function, storing a value in each node in the game tree; and to determine the order of daughters of a node.

Your program should build nodes in a game tree in advance, linking them to daughter nodes. Each node abstractly represents a possible position in a hypothetical game, although there is no game, no rules, and no move generator. Since important data within a node are to be determined randomly, nodes should therefore not be destroyed after being visited by search, but instead should be retained in memory for subsequent phases of search.

Any game tree should be built with uniform height and *approximately* uniform branching factor at each interior node. The leaf nodes should be assigned evaluation scores in such a way that negamax search will produce a desired value for the root node. Interior nodes (of which the root node is one) should be assigned evaluation scores which are roughly equal to the values that negamax search of their dependent subtrees will produce.

Four iterative-deepening searches should be performed on any tree: PVS, PVS with PV reordering, AlphaBeta, and AlphaBeta with PV reordering. The iterative deepening should stop when the maximum tree depth has been reached. (The four values returned for the root node should be the same at that extreme depth, if they are not this indicates a serious bug.) The total number of calls to 'Evaluate' in each type of search should be recorded. Averages over numerous trees with the same specifications of height and branching factor should be calculated, reported, thought about, and discussed in the report.

Step One: Building a tree - What and Why.

Three positive integers should be used in building trees: branching factor **b,** height **h,** and approximation **Approx.**

A tree node should store the following information, and only that:

1) its own static evaluation, an integer simulating what value a static evaluation function would produce for the corresponding game position without doing any search;
2) its daughters, in some originally-generated order;
3) its daughters, in an order that might get changed by PV reordering during a search.

In building a tree, distinguish between the estimated *static evaluation* value (***E***) of a node and the true *search-based* value (***T***). **E** should be stored with the node, **T** should be used during tree construction only and should not be stored. When beginning to build a tree, make a root node with **T** randomly chosen between 2500 and -2500 inclusive. For negamax to work, one of its daughters should have its T equal to the negative of this, and all others should have a T greater than or equal to that, up to a maximum of +10001. This pattern applies to any interior node and its daughters: one daughter has the T negated, no daughter has a lesser value.

Leaf nodes should have **E=T**. Interior nodes should have **E=T+*∂*** , where *∂* is a small number between -Approx and +Approx chosen randomly for each one individually.

The "best daughter" B of an interior node N is one[1] whose T is the same as the negation of that interior node's T. The best daughter should be placed randomly among the daughters, it should not systematically be either the first or the last in sequence, its identity should not be stored in its parent or elsewhere.

The daughters of an interior node should dominate subtrees whose height is usually one less than the parent's subtree's specified height. Exceptionally, a node with true value 10000 should have no daughters, representing a won-game position. Interior nodes (but not the root node) should have branching factor **b** with 90% chance, **b+1** with 5% chance, **b-1** with 5% chance.

PV reordering (see Step Three) is to be carried out in an iterative-deepening framework. This means it will sometimes change the order of daughter nodes after a shallow search and before the next-deeper search. This in turn means that after a search is completed, the tree is often not as it was before the search. For this reason, two copies of the set of daughters of a node should be kept. One copy gives the daughters in the order they were originally generated, the other gives the same daughters in an order that has perhaps been changed by PV reordering. Before starting any new search (PVS without reordering, alpha-beta with reordering etc.), the modified-order sets should be reinitialised to match the originally-generated sets. This will enable you to make measurements of search performance which compare like with like.

[1] There may be several daughters that happen to have this same value.

Step Two: Negamax-style alpha-beta algorithm with iterative deepening

Implement a simple negamax alpha-beta algorithm, with only these enhancements:
(a) code to count the number of static evaluations performed
(b) code to return both a value and a principal variation
(c) code to unpick the returned values appropriately
(d) a parameter indicating whether or not to use the modified daughter

-The function 'Evaluate' is trivial: given a node, it retrieves its preassigned E value.
-The test "no moves available" succeeds when a node has no daughters.
-The operation of "make new node" corresponds to picking the next daughter of an interior node, starting with the first in the original or the modifiable sequence, as determined by parameter. The parameter for a call to alpha-beta may be set to true when searching at the root if reordering is required, and passed on to the first daughter, granddaughter etc..
-The operation of "destroy node" should do nothing[2], you will need to keep those nodes for a second search in order to get comparability of results.

The principal variation from a leaf node is an empty sequence; the principal variation from an interior node is a sequence whose first element is the best daughter (which caused a cutoff or provided the return value), and the remainder is the best daughter's own principal variation.

Arrange for alpha-beta to be called repeatedly with increasing depth limits, alpha= -10000, beta=+10000.

Step Three: Principal Variation Reordering

Implement move reordering. After a search, the root node's modifiable-daughters sequence should be altered so that the best-seen daughter is in first position, and is followed by other daughters in the order they first occurred, without repetition. The same is then to be done with its best-seen daughter, and so on for all elements of the root node's principal variation.

Step Four: PVS with iterative deepening

Implement the PVS algorithm, which consists in calling the alpha-beta algorithm with varying values for its alpha and beta, and occasionally searching a subtree a second time with different alpha and beta values. See the lecture notes for week 5. Embed the algorithm in a iterative-deepening framework, as also shown in notes, in which the "resources available" test corresponds to not having reached the limiting depth of the tree.

Step Five: Experiment and Report

Step five is to arrange systematic experimentation. Arrange for the modifiable-daughters set of every node to be reinitialized between searches. Arrange for the principal variation reordering step to be carried out (or not carried out, depending on a parameter) between different-depth searches for both alpha-beta and PVS searches. Systematically vary parameters for

- tree height, from 4 to 6 in steps of 1
- branching factor, from 3 to 21 in steps of 3
- Approx, from 0 to 300 in steps of 50

---

[2] Note that this is not typical of how game playing programs work. It is a feature of this assignment that allows identical trees to be used in multiple searches, without needing game rules or evaluation function.

For each parameter combination, generate 25 trees, and compare the number of static evaluations done (in simulation) by alpha-beta, both with and without principal variation reordering, and by PVS, both with and without principal variation reordering.

Note that a tree with branching factor 21 and depth 6 will have approximately 90 million nodes. If such a size of tree is not feasible for your computer/language combination, do what you can.

Report what you did, what results you obtained, what you learned from them.

Step Six: Submission

By 7pm of Friday 3 November, submit a zip file containing your source code, a report, and if necessary a small appendix containing numerical results, via the course moodle.

Remember, you may use any programming language.

Late submissions will be handled according to UCD norms: up to one week late, 10% is deducted; up to two weeks late, 20% is deducted; beyond two weeks late, all marks are lost. These deductions are calculated on the basis of what is available, not what you have earned. So if you submit something ordinarily worth 66%, but two weeks late, you will receive 46%.

There may well be other assignments due around the same time for other modules: be aware of them and plan your time accordingly, do not act all surprised about it and ask for extensions.

Marking scheme

Build tree nodes	25%
Implement alpha-beta with progressive deepening	20%
Implement PVS with progressive deepening	20%
Implement move reordering	10%
Systematic experimentation	10%
Report	15%

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 source files, report and outputs of your program are required. You may also include a scanned signed Assessment Submission Form.

***Do not* present your writeup and appendix as anything other than pdf.** No .doc .docx .rtf or other word processor specific formats should be submitted. All decent word processing programs (such as Microsoft Word) allow saving a pdf, often as a Print option.

***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 folder of containing only the necessary files. Do this even if your environment has created eg a bin and src folder for you.

***Do not* provide multiple versions of your code.** Non-working "improved" versions are useless.

***Do not* send your assignment using email**. Use moodle.

***Do not* present code in lines with more than 100 characters.** Tab characters count as eight ordinary characters. Comments count as code.

***Do not* give your submission a long filename.** Use your own name for the folder created by unzipping what you submit: if you are Tom Thumb, make the folder be called Tom_Thumb or Thumb-Tom. Preferably, also make the zip file be Tom_Thumb.zip. Don't put things like Assignment1 or your student number or comp30260 into the filename.

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

Do not commit plagiarism of any kind: using the work of another without proper attribution, or allowing another to use your work, are serious offences attracting serious penalties.

Do put your name and student number at the top of page 1 of your report. Do put your name and student number in a comment at the start of each source code file.

***Do not* ask for an extension on the grounds that you have other assignments too.**

Do print out and sign the Assessment Submission Form, and either hand it in to the School Office by midday Tuesday 7 November (end of lecture), or scan it and submit it in your zip file.

Do feel free to ask me (Arthur Cater) for advice if you need it.

Finally, take heart: it is always far better to submit an assignment a little late and/or incomplete than to submit nothing at all. You should of course aim to produce a complete, correct, thoughtful and timely submission, and if you do you will get top marks!