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

Announced Thursday 4 October 2018. Due 9am 5 November.
This assignment is worth 20% of the marks for the module.

## Negamax AlphaBeta with Selective Quiescence Search

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 four 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 Selective Quiescence Search within negamax-style AlphaBeta.

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

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 cannot 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 non-uniform height and *only 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.

Two searches should be performed on any tree: AlphaBeta with Selective Quiescence Search, and AlphaBeta with plain Static Evaluation. The total number of calls to 'Evaluate' in each type of search should be recorded, also the disagreement (if any) between the value returned by search and the desired value specified when the tree was built. 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.

Your program must build a tree with non-uniform depth (number of links between leaf node and root node), in order to support selective quiescence search, and non-uniform branching factor, in order to simulate characteristics of real game trees. The tree nodes will have randomly assigned evaluation scores in order to eliminate the need for specifying game rules or evaluation functions. The trees need to be stored in memory, rather than created-and-destroyed on the fly, so that comparison between two algorithms on the same tree are possible: with node characteristics being randomly generated, it would not be possible to generate the same tree a second time.

Five positive integers should be used in building trees: branching factor **b,** horizon **h,** desired value **v**, approximation **Approx**, interestingness **i**. Their use is described below.

**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) whether or not it is considered "interesting" for selective quiescence 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, but **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 +10000. 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+$\partial$** , where $\partial$ is a small number between -Approx and +Approx inclusive, chosen uniformly randomly for each one individually.

The "best daughter" B of any interior node N is one[1] whose T is the the negation of N'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 specified 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+2** with 3% chance, **b+1** with 6% chance, **b** with 80% chance, **b-1** with 7% chance, **b-2** with 4% chance. Naturally, if this results in a negative branching factor, it should be treated as zero.

When a horizon node is generated, at **h** levels below the root, it may or may not be considered *interesting*. If it happens to have the true value 10000 it is never interesting, it will be a leaf node. If a random number 0-99 inclusive is *not below* the interestingness threshold **i**, the node should be a leaf node, otherwise it should be marked as interesting and given a subtree as if it were a normal interior node. In any such subtree the interestingness threshold should be progressively reduced by 2 at each level, thus guaranteeing that trees will not become arbitrarily deep.

When an ordinary interior node is being generated, say **d** links from the root where **d<h**, it should be marked interesting if a random number 0-99 is below **i+(h-d)*30**. This should result in all

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

nodes in the top few levels of a reasonably deep tree being marked interesting, with a few uninteresting ones appearing near the horizon. At the horizon and beyond, with **i=20** for example, uninteresting nodes will be in a clear majority.

## Step Two: Negamax-style alpha-beta algorithm

Implement a simple negamax alpha-beta algorithm, with only these enhancements:
(a) code to count the number of static evaluations performed
(b) a parameter to dictate whether or not a Selective Quiescence Search should be used

-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" mentioned in notes corresponds to picking the next daughter of an interior node, starting with the first in the sequence.
-The operation of "destroy node" mentioned in notes should do nothing[2], you will need to keep those nodes for a second search in order to be able to compare results.

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

## Step Three: Selective Quiescence Search

Implement a selective quiescence search. When AlphaBeta is given the appropriate parameter, and is considering a node at the limiting depth of its search, instead of the plain Evaluate it should use the quiescence search. This does use Evaluate on the node, but also considers whether any daughters of the node are "interesting": if any are interesting, it operates recursively on them, trying to determine if further search gives a better result than static evaluation did. See notes.

## Step Four: Experiment and Report

Step four is to arrange systematic experimentation. Systematically vary parameters for

- tree horizon **h**, from 4 to 7 in steps of 1
- branching factor **b**, from 3 to 11[3] in steps of 4
- Approx, from 100 to 300 in steps of 100
- Interestingness, using 20 and optionally 10 or 30 or both

For each parameter combination, generate 12 trees. Perform searches of those trees with search height limits of **h-2 h-1** and **h**, and compare the number of static evaluations done (in simulation) by alpha-beta, both with and without selective quiescence search. Also measure and compare the average absolute difference between the results of the searches and the desired values specified when the tree was built.

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

---

[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.
[3] Note that a tree with uniform branching factor 11 and horizon 7 and nothing beyond the horizon will have approximately 20 million nodes. If such a size of tree is not feasible for your computer/language combination, do what you can.

## Step Five: Submission

By 9am of Monday 5 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 | 35% |
| Implement alpha-beta | 10% |
| Implement quiescence search | 20% |
| Systematic experimentation | 20% |
| Report | 15% |

# Do's and Don'ts

## *Do re-read these Do's and Don'ts shortly 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 containing only the necessary files. Do this even if your IDE 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. I recommend avoiding tab characters and using spaces instead. Comments count as code.

***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.*

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

***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.

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

***Do** print out and sign the Assessment Submission Form, and scan it or photograph 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!