COMP30260 "Artificial Intelligence for Games & Puzzles"
2nd Programming Assignment

Announced Friday 27 October 2017. Due 7pm Friday 24 November.
This assignment is worth 20% of the marks for the module.

Much-modified Connect-Four with Monte Carlo 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

You are to implement a player for a game, 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 play a modified version of Connect-Four. It should be able to both play against a human opponent, and to play both sides using different parameters controlling thinking effort and hence play quality. You should run experiments to determine the relationship between these parameters and the likelihood of winning.

There are game rules to be programmed and positions to be represented. You will have to implement a move generator, an end-of-game detection component, and a move choice algorithm based on the Monte Carlo method. A simple interface is required, allowing you as the experimenter

1. first, to decide the parameters for the game setup, whether human-vs-computer or computer-vs-computer play is wanted, which player goes first, the effort for each computer player
2. second, to view the state of the game when playing, and to input human-choice moves. The view of the state of the game should be done very simply, using ASCII art rather than any fancy point-and-click graphical interface.

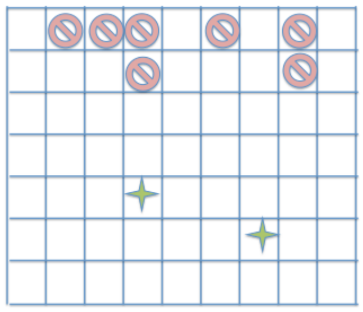
The modified Connect-Four game

Connect-Four in its standard form has a grid containing seven columns and six rows. In turn, players drop a disc (yellow for one player, blue for the other) into one of the columns (provided it is not full): it falls to the lowest row possible in that column. If a player gets four discs in an uninterrupted line, vertically or horizontally or diagonally, they win. If the grid is full and there is no winner, the game is drawn.

The game you are to implement has three modifications.

1. The number of columns in any particular game may be any integer from six to eleven; the experimenter must be able to choose how many columns to play with.
2. The columns are not all of the same height; the experimenter must be able to choose the heights of the columns.
3. Two possible locations (row/column pairs) for discs do not count for making a line of four; the experimenter must be able to choose any two locations.

For example, it should be possible to setup and play on a grid such as the one below left, where pink "no" symbols are nonexistent parts of short columns, and green "star" symbols are locations that do exist but do not count for making a line of four.



This setup might be specified by experimenter as
7 6 6 5 7 6 7 5 7
D3 G2

meaning the columns have heights 7 6 6 5 7 6 7 5 7
(and hence there are 9 columns)
and the don't-count cells are in column 4 row 3 and in column 7 row 2.

You should not attempt to make your program display graphical symbols, Xs and Os will do fine for the yellow and blue discs, # might indicate unplayable cells at the top of short columns, dots might indicate empty don't-count cells, lower-case xs and os might indicate occupied don't-count cells. (Since the cells don't count for making lines of four, all that matters is that empty and full should not be confused).

Move input for human

If the experimenter is playing and it is their turn, they should be prompted for which column (A B C etc) they wish to play in. If the column is full, their move should be rejected and another prompt given.

End of game detection

If a move by either player has resulted in a line of four that does not pass through a don't-count cell, the game is over and - in a human-vs-computer game - the winner should be announced. If a move results in the grid being full, a draw should similarly be announced.

Computer choice of move

When the computer has to choose a move, it should play the move that wins immediately if there is one. Otherwise, it should use a Monte Carlo technique. This means playing many out many random trial sequences of moves for each player, until the end of the game. (Again, it should be assumed that if at any point there is a move that wins immediately, it will be chosen. The trials can therefore often stop just short of the end of the game.) Each such trial results in a win, a loss, or a draw for the computer player. Each possible first move will have a ratio of wins:trials, the move with the best ratio should be chosen.

Experiment and report

You should set up at least four different configurations of columns and don't-care cells. For each one, you should play the program yourself, trying to wins, under at least four effort settings: on each move, the computer tries 80 random sequences, or 150, or 500, or 2000. Also for each one, you should get the computer to play itself with one player using 80 or 150 or 500 or 2000 sequences, and the other player also using 80 or 150 or 500 or 2000, in all of the sixteen possible combinations. For each combination, get computer to play itself at least 10 times (but preferably 100 or more) and determine the proportions of games won by the different computer players. Report what you did, what results you obtained, what you learned from them.

Submission

By 7pm of Friday 24 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

Implement game setup and ASCII-art display	10%
Implement end-of-game detection	25%
Implement human-vs-computer play	25%
Implement computer-vs-computer play	10%
Systematic experimentation	15%
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: substitute eight spaces for tabs if appropriate. 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 28 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!