Two More BLACK Credits for Unit 3

Eckel, TJHSST AI1, Fall 2021

Background & Explanation

These could in theory be four separate documents, but just to make your planning easier, I've put them all here. One BLACK credit in Unit 3 is available in the Othello assignment; this details your options for the other two.

You have two choices to make here.

- Your first choice is between **redoing an earlier assignment in a different programming language and reflecting on the strengths and weaknesses vs Python** or **inventing a new assignment to possibly replace N-Queens in a future year**, as it was the most unpopular assignment last year and so I'm looking for something cooler. Either can be done any time after finishing unit 2.
- Your second credit comes from implementing a different game using minimax & alpha/beta. **You choose the game**. I have a few recommendations (see that section below), but it's up to you.

First credit, option 1: Learn another language & compare w/ Python

Julia is a new programming language that is gaining some popularity as an alternative to Python in many contexts. It's compiled rather than interpreted, and so it can run much faster than Python, while offering an interpreted-like syntax that retains Python's flexibility. Rust is another more recent language that's making waves; it's the most beloved language to program in, according to people who program in that language professionally. I've also heard mentions of Dart, Zig, and Deno as good alternatives; I know **nothing** about these languages and would love to hear more. We're intrigued by the possibility of switching AI to one of these languages at some point in the next several years; probably not yet, but maybe someday. Your advice will help us monitor the situation and determine if this is a good idea.

You can also choose a different language that isn't one of those five. It just can't be Java or Javascript or Kotlin; we know those already. It should be one you're investigating for the first time, not one you're already fluent in.

This is the task:

- Choose an assignment from AI1 where efficiency is relevant. (Ie, pick an assignment with a case that has a runtime of at least a few seconds; choose a long 15-Puzzle path or Sudoku or Othello, etc. Don't choose, like, Peg Solitaire.)
- Learn enough of your chosen language to be able to recreate your algorithm as exactly as possible. Presumably, each language's homepage contains links to may resources; feel free to search for anything beyond that you like. Keep track of any resources you use so you can report them later.
- Recreate your Python code as exactly as possible in the new language. Don't add in additional improvements; try to get as close to line-by-line recreating your Python code as you can. If the language isn't able to do so, make a note of what you had to change.
- Write a *BRIEF* report (this really should not take long) where you write two paragraphs or so about the experience of learning the language, what resources / programs / tutorials / websites you used, and how easy you felt the whole experience was. Report the difference in runtimes between your Python code and your code in the other language. Finally, tell what you think about a future version of this class using that language instead of Python in the future.

THIS ASSIGNMENT MAY BE DONE WITH A PARTNER! The additional requirement is that you must recreate TWO algorithms instead of one, but you can collaborate on writing just one report and, of course, help write / verify each other's code directly.

Submit your Python code, your code in the other language, and your report (three files) to the link on the course website. If working with a partner, this would be five files – two pairs of Python and the other language, and one report.

This assignment is **complete** if:

- The "Name" field on the Dropbox submission form contains your **class period**, then your **last name**, then your **first name**, in that order. (If working with a partner, include both last names and first initials.)
- Your files meet the above requirements. In particular, I should be able to open the files in each language next to each other and see how they correspond; choose similar variable names, etc, to facilitate this.

First credit, option 2: Propose a replacement for N-Queens

There are a lot of constraint satisfaction problems in the world. Here are some that come to mind right away:

- Map coloring
- Cryptarithms
- Any kind of simulated school scheduling problem (rooms, teachers, students, courses)
- Also, any NP-complete problem can be written as a CSP (bin packing, subset sum, vertex cover, …)

I've looked into these a bit, but I haven't found an assignment yet in my head that recreates what the N-Queens assignment has to offer. Specifically:

- Scalability. The N-Queens assignment is easily scalable – all you have to do to generate progressively harder problems is just up the value of N by 1. Cryptarithms are an example of a problem where scalability is a concern; there are tons of resources for quick easy ones, but few resources for increasingly difficult ones.
- Appropriateness to both backtracking and incremental repair. N-Queens works well to learn both of these strategies, and so I'm loathe to give it up for a problem that demonstrates backtracking well but doesn't demonstrate incremental repair well, for instance.
- Not super hard to model. N-Queens takes something like an hour to model. Most of the time is spent learning the constraint satisfaction algorithms. I'm also loathe to replace it with something that takes much more coding time just to get working in the first place.

That said, for two years now N-Queens has been voted the least popular assignment in the course. So: it would be great if I could come up with something!

If you, working with any group of size up to 5 (as long as you produce a thorough and detailed amount of work), feel like you have an idea that will work, that'd be great! So far, I have one entire unit in the course that came completely from a student's idea (Genetic Algorithms) and students have contributed to the Sudoku assignment as well. I'd love for you to join them!

You don't need to formally write up the entire assignment like me, but I would need:

- A description of the coding tasks.
- Solution code demonstrating successful use of both backtracking and incremental repair on problems complex enough to take a few seconds, but simple enough to… well, to take a few seconds. Not too short, not too long.
- A lot of test cases, or a process/script whereby students can easily generate them.
- Why you think this is more fun that N-Queens.

This is a little complicated because I'm likely to be on paternity leave at this point in the course, but if you're interested in taking this one on I'd love to talk to you about it. If I'm out, this is a **good reason** to send me an email and find a time to chat. There's no submission link; we'll figure that out in our conversation as well.

Second credit: Use minimax and A/B pruning to make a game of your choice

This is the task:

- Model your game of choice. Create a way of outputting each game state that is clear to the user.
- Create three AI players:
    - RANDOM should make a random valid move.
    - AGGRESSIVE should always try to capture the most important enemy piece it can, or achieve the biggest score gain it can, etc. If it can't capture anything, it plays as random.
    - BEST should be your best implementation of Minimax, Alpha/Beta, etc. It should beat the previous two players the vast majority of the time.
- Write Python code that takes two command line arguments from the set of "RANDOM", "AGGRESSIVE", "BEST", "USER". It should then play out a game between the two selected players, with the player identified in the first argument going first. If "USER" is selected, instructions for how to input moves myself should be clear. You don't have to deal with bad input; I will only make valid moves.
- Write a *BRIEF* report (this really should not take long) where you give a sketch of how your strategy works. Two to three paragraphs is plenty.

Off the top of my head, I'm happy to allow:

- My favorite game for this assignment is Ultimate Tic-Tac-Toe; if you've never heard of it go to http://bejofo.net/ttt and play a few games; learn how the game Ultimate Tic-Tac-Toe works.
- Checkers
- Chess
- Go (if you can explain to me an analogous option for the AGGRESSIVE AI)
- Hive (if you can figure out a good way to print out a game state and also explain to me an analogous option for the AGGRESSIVE AI)
- Literally anything else you want to propose

Implement the game. You figure out how the board is displayed and how the user interface works, but make it not be terrible, please. When I run your code, I should see a brief explanation of how to play before the game begins.

THIS ASSIGNMENT MAY BE DONE WITH A PARTNER! The additional requirement is that you employ a technique in your AI that we didn't cover in class and explain in your report what it is. Check with me if you're not sure if it counts.

Submit your Python script and your report (two files) to the link on the course website.

This assignment is **complete** if:

- The "Name" field on the Dropbox submission form contains your **class period**, then your **last name**, then your **first name**, in that order. (If working with a partner, include both last names and first initials.)
- Your Python file meets the above specification for input and output.
- Runtime is reasonable (ie, I don't have to wait a ridiculously long time for the AI to make a move). Keep each move under approximately 10 seconds if possible; if not, let me know and we can negotiate.