CS310: Advanced Data Structures and Algorithmns

Spring 2014 Programming Assignment 5

Due: Tuesday, April 21, at midnight.

Goals

This assignment aims to help you:

- Learn about backtracking and dynamic programming
- Use more of the Game package: Players, full games
- Add intelligence to the Game decisions through search of the game tree.
- Use timings to explore the complexity of these searches, and how much dynamic programming helps.

Reading

See PA4

Description

In this assignment we continue our study of algorithms and algorithm patterns using Prof. Bolker's game package.

Questions

1. Write a ComputerPlayer that uses a backtracking search of the game tree to choose the best move for the player whose turn it is. The subtlest part is figuring out how to have findbest act consistently in the best interests of the player whose move it is as it proceeds recursively down the game tree, whether it's player 1 or player 2 who's about to move. You will find that implementing this algorithm requires lots of thought and very few lines of code. It's like binary searches or linked list manipulations. Once you understand the problem completely the details are straightforward. But if your programming style is to write some Java that's approximately correct and then tinker with it until it's right you will probably waste a lot of time and may not end up with a working program. Think first.

   If you code something up and it fails, seriously consider throwing it out and starting again rather than trying to modify what you have. Put your code in file cs310.Backtrack.java, replacing the stub code. You can read about backtracking in the class notes and in Weiss, Section 7.7. In fact the code you need is there, ready to be reworked so that it uses the Game package rather than the rules of tic-tac-toe. You are welcome to use it if you wish, but you may in fact be better off thinking the algorithm through for yourself from the start.

2. Testing (write-up in memo.txt). Algorithms that appear to work properly on one game may fail on others. Be sure to test on more than one game. Backtracking may be too slow for some long games, but you can test your algorithm thoroughly on short ones without overly taxing system resources. Make sure you test with the computer going first, and then going second. Play on paper first, predicting what the computer should do, then play the game with the computer and see if it chose the moves you thought it would. Nim, 15, and putnam present deeper game trees and are useful for testing that the

correct decision is made at each level. You may find that your backtracking program chooses moves that surprise you. Carefully analyze those surprises, to determine whether the problem is in the algorithm or in your understanding of how smart it is supposed to be. Use the script command on UNIX, or copy-and-paste on Windows, to capture your program sessions for yourself, but then edit, shorten and annotate them for your memo. Discuss some interesting cases in several of the games, explaining why the move chosen by your algorithm in those cases is what you expected it to be and is (or is not) the choice you would have made yourself. **Hint:** For nim the computer should choose A3 first and win. Moving second it should win if and only if the human player errs.

3. Dynamic programming. Write a ComputerPlayer implementation that incorporates some dynamic programming: use a Map to keep track of the values of positions you've analyzed by backtracking so that you need not reevaluate game tree positions already considered. The pseudocode looks like this:

```
int getValue(Game g)
   try to look up value of g in a map
   if you succeed return the value
   /* else */
   compute the value of g recursively using backtracking
   save the computed value in the map
   return the value
```

   Put your code in cs310.Dynamic.java. Then, for example, "java cs310.PlayGame Easy cs310.Dynamic" will pick up the new Dynamic.class.

4. Performance. Report in your memo.txt on performance of the dynamic programming version compared to the plain backtracking version. Use the timing module in the game package. Present numerical data obtained from the final timing report of PlayGame, using the same moves of the same game and the same computer system. Don't bother with Easy here, just games that actually take a little time. But don't burn up CPU either. If a game move takes more than a minute or so, abort the run and record "more than 2 minutes" or whatever. Report on 3 games.

Deliverables

In your cs310/pa5 directory: These will be tested via "java cs310.PlayGame Nim cs310.Backtrack human", for example, or with Tournament, or both.

- cs310.Backtrack.java
- cs310.Dynamic.java
- memo.txt (indicate whether you used late days).