Andy Fleischer

Prof. Karla Fant

CS202

31 July 2021

Program 2 Efficiency Writeup

For program 2, we had to make a civilization building game to practice implementing dynamic binding. We needed to implement three different types of civilizations, so I chose a farming civ, a hunting civ, and a mining civ. Each of these civilization types was their own unique class, each of which inherited from an abstract base class Civilization. I pushed up the commons (items, money, civilization name, etc.) and brought down the differences (different plants for the farming civ, ores for the mining civ, etc.). Then, since I would never need to instantiate just a "Civilization", I made it abstract through pure virtual buy and sell functions that were left to be implemented by the different civilization types. This approach was quite effective not just for one civilization, but for storing multiple in the form of, say, an array. However, I also implemented a doubly linked list through a Node class with a Civilization pointer, Node pointer next, and Node pointer previous. Then, I made a DLL class that contained a Node pointer head and Node pointer tail, with public member functions (and associated private recursive functions where needed) to manage the Nodes as a doubly linked list.

Overall, I think this approach was definitely object oriented (dynamic binding is in my opinion one of the strongest applications of object oriented programming), save for maybe the Node class. I did end up needing getters and setters for my Node pointers in the Node class, since the DLL needed to manage them. Also, I needed a getter for my Civilization pointer unless I wanted to make a bunch of wrapper functions in the Node class. Although that would be feasible, it seems clunky to me and I view Node as a way to package data with a pointer to the next (or previous) data, not as a true object oriented class.

Going through the process, I did not ever need to make drastic changes since I did a lot of planning before implementing (measure twice cut once). In program 1, I needed to make big changes to the main method to implement the data structure, but dynamic binding made that so much easier. My main was much cleaner since I did not need three different data structures, each with their own menus with different wrapper functions to call the functions of their respective classes.

In terms of speed efficiency with the data structure, some of the functions were faster than others. For example, the insertion simply added a node to the end of the list. Since I stored a tail pointer, this was always a constant O(1) time function. Remove, however, required searching through the entire list at the worst case, which is O(n) time. Display and remove all were, as usual, always O(n) time to traverse the whole list. Memory efficiency is where a DLL lacks. There are next and previous pointers in every Node. Compared to a linear linked list or circular linked list, which only have one pointer per node, the doubly linked list will take twice the memory. Also, performing dynamic binding required memory to be able to dynamically create civilizations at runtime. Compared to program 1, where I did not need to use pointers for events, this does take more memory. Further, not only does the doubly linked list with dynamic binding approach require more memory, but also much more code to maintain, since you need to hook up both next and previous pointers, and manage a head pointer and tail pointer. The dynamic binding did make the code simpler to maintain, though.