Ryan Filgas
CS202
Karla Fant
5/7/2021

Efficiency Write-up for Program 2 – See GDB write-up on page 3

The design for this project would have been better with more focus on small implementation details in the design phases. My Design was a data structure that made perfect sense, however when it came down to how it was implemented, and how data was passed between the classes, it fell embarrassingly short.

As far as OOD goes, the structure itself of the base and derived classes performed how it was supposed to. Deriving a menu from the list class was implemented later, and also helped simplify some of the work. I see really good potential for these data structures. The complications came in actually coding the dynamics of gameplay for the user. My use of random was at first incorrect. Instead of seeding the time once and calling random, the random function was seeded every time a node was created for example. This meant that since everything was allocated at the same time, the seed was relatively identical, and so what was supposed to be random wasn't so random. I ran into complications using const with copy constructors, as well as trying to pass information in and out of the string class. I didn't have a full understanding of these before I started coding them.

The conceptual ideas behind my programming approach were valid, however the execution lacked. The structure consists of a player class to hold data, and trip victory flags, a menu and list class to manage the data structure, and a hierarchy of classes to manage the obstacles. The interaction between these classes needed to be made simpler. I was initially attached to the idea of having these structures being fully modular, and I think the project would have been more successful had I done this. When it got to the time to start making menus, I looked at what others were doing, and it seemed fairly common to keep the menus separate from the data structure. Trying to pass information from a player class, to a list class, through a node, and down to a hierarchy and back is in itself an incredibly problematic thing to do if it can be avoided. Programming this in a modular way would have allowed better incremental testing, a much simpler programming experience, and easier to track errors. My project ended up having menus scattered to different areas with no real reason for them being there. It ended up having no unified structure overall.

To improve on the next project, I plan to design the project from bottom to top before coding, or to work in a more modular way that can be more easily incrementally tested. Where I got caught up this time was programming for the incremental turn-ins, but never taking the time to really dial in my design before starting. I think had I slowed down to make a thoughtful approach, I could have had a better program finished in half the time. The data structure algorithms and traversals are very easy to understand, but game dynamics and interaction between classes really need to be planned out.

Ryan Filgas
CS202
Karla Fant
5/7/2021

As far as major changes go, I made a few. I originally was going to have each step up in complexity have its own user interface. I realized the flaw in this would be that I wasn't using dynamic binding to call my functions from the node level, so I tried to move my menus upwards in hopes of realizing the goals of the assignment rather than sidestepping them. The other change I made was adding a menu class, and deriving it from the list class. This was perhaps the one good choice I made, but I didn't effectively differentiate what its job was in comparison to the list class, so their purposes were muddled.

Data structure wise, I actually really like OOD for this assignment. The array of DLLs allowed for random entry into new lists without excessive traversals, and the marking of checkpoints when a list was fully traversed. The ability to move forwards and backwards was good as well. As far as gaming goes, it's hard to say with this many nodes to traverse if it would be efficient enough for high performance gaming, but it does provide a lot of flexibility.

A more object oriented design would have clearly separated the duties of each class in a cleaner way. Part of the idea is to reduce the amount of code needed for the client, and see relationships in such a way that every class has what it needs to function and the passing of information between them is kept to a minimum. These structures accomplished that goal in part. It was in fact easy to call a function from a node to any of three derived classes, and it was really awesome to support multiple types of objects in a single node type. It's a really clever construct.

Ryan Filgas
CS202
Karla Fant
5/7/2021

GDB Writeup – Program 2

In this project I used gdb extensively to step through code. Part of this was displaying addresses to nodes to make sure information was accurately being passed between classes. Part of this was looking for seg faults and the source of memory leaks. Where I would have liked to use gdb more was testing the parts of the program I couldn't test from the user side quite yet. This would have allowed me to detect problems sooner. I found gdb helpful in making sure my destructors were properly built, and the ability to backtrace problems helped me see where the problematic code was called. This code was particularly hard to debug because of everything being randomly allocated. It wasn't possible to look at returns for the number of nodes and know that the destructors missed something, so it was very important to be able to step through. I modified values for the first time inside the debugger, and in the future I think that's going to be incredibly helpful. Rather than making changes, finding where I was last, and recompiling, I can fork the process to save the instance, change the variable to see if it works, and then go back. I'd like to explore using the watch function to stop when a variable changes, as that would be incredibly helpful as well.