Tom Dale

CS-273

FinalSummary.doc

7/24/2016

Traffic Simulation Final Summary

After finishing my CS 273 final project I can reflect back on how much I’ve learned about data structures, general programming, and traffic intersection. The development process was the longest I have yet gone through, but upon completion I am very satisfied with my work. After CS 273 I feel confident in the manipulation of data in C++ and perhaps more importantly how to tackle a problem that is initially overwhelming.

My findings from my traffic simulator agree with my hypothesis stated in my property description. As the graph shows, both roundabouts and traffic lights increase the average wait as traffic inflow becomes greater. While a traffic light is more efficient when few cars use it, a roundabout is most likely better when the intersection is in a populated area. An interesting find was the extreme inefficiency of the roundabout to handle traffic from just one direction. This is logical because every car has to drive in a continuous line around the circle until they exit. The overall efficiency of both intersections is increased when a more equal amount of cars are entering from all directions rather than concentrated on one street. This simulation could be very valuable to city planners who are debating which type of intersection to build because it appears that neither intersection type is always better.

Y axis = average car time (seconds) X axis = average cars per hour Line % = distribution of

The development process of this simulation took by far the most amount of time when compared to other programs I have written in the past. Starting with pen and paper I drew out diagrams of intersection use, create a basic UML, and wrote out use cases of each car moving though both intersections. After I felt confident that I knew the basic requirement, I moved to the computer and wrote the first draft my FinalSpec.doc. After I started programming I had to scrap most of the use cases and pseudo code that I wrote as I found better solutions to move cars around in the intersections. The requirement to use a set at first seemed easy to fulfill, but as I designed I realized that this project didn’t necisarily need a set or map. To fulfill the requirement I eventually decided to create a set of cars that would never be deleted so that each resident of CS273ville could keep his or her own car. This was an added feature to my original design but it adds realism to the simulation. This also allowed me to practice reading in from a .txt file to get name which I had many challenges with. The part of this program that caused the most bugs was the many queues, sets, and traffic spaces of pointers to Cars. Initially I tried to write the whole program referencing the actual cars but I quickly realized this would be impossible to keep all the cars in their proper scopes. After switching to pointers the program started compiling, but I had many errors and had to create a crude ASCII represented intersection to see what the cars were really doing.

I have learned a lot from programming this simulation. One valuable lesson is to take a project one step at a time. As I was slowly designing the project I was really worried about the time I needed to finish the program. Just knowing where to begin was a hard part of this assignment. The real key to solid programming that I found was just to relax. When I was tense about a due date I was making a lot of errors and designing code that made no sense. Now that I have successfully completed this simulation I feel much more confident to take on larger scale programming projects. I just need to break down the problem into many small pieces that I can isolate and process, eventually forming an entire solution.