Group 1
CSC 2180-02
Bryan Smith - Cory Elswick - Scott Arnette
4/8/13

Programming Assignment 5 - Building Simulation

Introduction/Overview

When we started working on this assignment, we divided it into the three major parts and split the load. Bryan with the SimBuilding class, Cory with the Floor class, and Scott with the Elevator abstract class. After this much was completed, we were to implement the move methods that would derive from the Elevator class. As the project went on, our specific roles became less defined and we worked on making all of our efforts work together in a manner that made sense, compiled, and produced the output we desired.

Floor Class

The Floor class wasn't all that bad to write, and the basic things we wanted from it were fairly simple; people arrive here, people wait here, people get on the elevator here, then we do some math to get an average wait time. The Floor class derives from linkedQueueType and takes advantage of some methods that have already been implemented. The most important functions of Floor are responsible for: randomly generating people at the floor into the linked queue, incrementing the wait time of the people in the queue, removing an element from the front of the queue, summing up the wait time of all those who left the queue to board the elevator, and getting the average wait time.

Elevator Class

In regard to the functionality of the program, the elevator class, as well as its subclasses of different elevator types, is created to handle the movement between floors and the loading and unloading of people on different floors. The core process behind ensuring that the elevator works as one would expect is the move function. In the base elevator class, the move function is created as a pure virtual function, because it has no definitions made within the original elevator class. Only in its subclasses of allElevators, evenElevators, and oddElevators do we see the move function in play with specific definitions. The move function works by checking the direction the elevator is moving, by the boolean value “moving up”. If the boolean value is true, the elevator simply moves to the next floor, the next value in the list, and if it is false, the elevator moves down, the previous value in the list. However, as one would expect, there are only so many floors to move up before needing to change direction and move downward. To remedy this, two “if” statements check whether the elevator is about to reach a null value, that being the either above floor 33, or below floor 1 for odd and all elevators and floor 2 for even elevators. If the next item in the queue is a null value, the boolean value is flipped from true to false and vice versa and the elevator is sent in the appropriate direction. The elevator also loads “people” generated in the floor class by taking on as many people as possible until the maxCapacity value of 15 is reached. The unloading process is emulated by generating a random number and reducing the currentLoad value by that number.

SimBuilding Class

The SimBuilding Class is the core of the program. This is where the Floor and Elevator classes are put into play. Each SimBuilding represents one building/simulation which contains a doubly linked list of elevators and floors, also it keeps track of the total wait time that it keeps a running total of that is retrieved from each floors individual total. SimBuilding also manages the overall simulation’s timing. There is a step method that iterates through the floors and elevators using the linkedListIterator and for each step there are a set of actions called: between zero and three people are arrive at the floor, the elevators move up or down, between zero and three people leave the elevator if there is one at that floor, the elevator takes on all it can hold up to a maximum of fifteen from the current floor, and lastly the wait time of the people left on the floors is incremented. This step represents 30 seconds of simulation time. SimBuilding also has a run function that will call the step function for number of iterations the simulation is setup for. Along with controlling the steps, this class also totals up the wait time from all the floors and the gives the average wait time at the end. All of these variables, such as how many floors there are, how many elevators there are, and how many iterations to run the simulation, are all set up using the constructor.

Conclusion

When we first started programming for this project, it was easy to tell that it would take a lot of hard work and compliance to make everything work the way it was supposed to. There were places where we stumbled and had to push to find what to do next, and there were a lot of changes that needed to be made through the course of the project to get everything running smoothly. In the end, the hardest aspect of the assignment was probably the cooperative work; making everybody's part work together was pretty difficult, especially because we all took different approaches and had our own styles.

Running our simulation shows that a building with 33 floors, 5 elevators, and running the simulation for 30 minutes (60 steps) results in an obvious choice between an elevator configuration of 3 even 2 odd or 2 odd 3 even. These two configurations were significantly faster than the others and have negligible difference between the two, as expected.