Final Project

For this project, I decided to create a simulation of a hockey arena lobby. I have based this on my experience going to a Penguins game at PPG Paints Arena in Pittsburgh PA.

It starts off with a customer that gets in line for a metal detector. When going through the metal detector, it has a 20% chance of setting it off. After, the customer goes to the ticket counter if they do not already have a ticket or to the scanners if they do. At the ticket counters and scanners, the customers wait to until they reach the front. When they do, they buy or scan their ticket.

Each stage has at most 10 stations. The metal detectors start will all 10 stations open. For the ticket counters and scanners, they initially start with only 3 stations available. If each line have more than 10 customers, a new station will open and will fill with customers from the back of each line.

The simulation starts with the Main program, which has objects of HockeySim and Statistics. The HockeySim object will create a HockeyInfo object and get its instance from the class, declare and initialize the semaphores needed along with the amount of stations the ticket counters and scanner will initially have. The Main program will then call HockeySim’s run program with the Statistics object as the argument.

Once in the run method, it will create new customer objects which will be added to the metal detector lists that are held in the HockeyInfo instance, and then it will declare and initialize 10 threads of each stage (metal detectors, ticket counters, and scanners). When initializing, the constructor receives the HockeyInfo instance, respective line number, respective semaphore list as parameters. If the thread is a MetalDetector, it also sends the Statistics instance. Finally, it will call each thread’s start method.

Each thread has variables that holds the line number, semaphore list, single HockeyInfo instance, and Statistics instance if it is a MetalDetector. The MetalDetector threads start by acquiring a semaphore permit. Once it has the permit, it will determine if there are any customers waiting in line. If they do, they will update one of the statistics, print out that the metal detector has a new customer and that they are going through the detector. It will then wait a few seconds to represent them going through. Next, the program will generate a random decimal between 0 and 1. If that number is less than 0.2, the program will update one of the statistics, print out that the customer set off the detector and is getting a pat down, and wait some number of seconds to represent the pat down. Finally, the program prints that the customer is leaving the detector. It gets the customer that will be removed, and based on if that customer has a ticket it will use the action method from the Ticket class or the action method from the NoTicket class if that customer doesn’t already have their ticket. In each action method, it will update one of the statistics. If there are no customers waiting, the program will print that the detector is waiting for customers and wait 10 seconds. If there are still no customers, the thread will close. At the end, the program will release the permit. It will then repeat this process until thread is closed.

The TicketCounter and Scanner threads are the same except for their print statements. They start by acquiring a semaphore permit. This is a crucial part since that is what determines what threads are running (which service is open). Once it has the permit, it’ll determine if there are customers waiting in line. If there are, it will print that there’s a new customer at the counter/scanner and they are buying/scanning their ticket. It will wait a few seconds to represent the sale/scan. The TicketCounter thread will move the customer to one of the scanner lines, while the Scanner thread will remove the customer, which represents the customer getting to their seat. If there are no customers waiting, the program will print that the counter/scanner is waiting for customers and wait 10 seconds. If there are still no customers, the thread will close. At the end, the program will release the permit. It will then repeat this process until thread is closed.

Once all threads are close, the HockeySim run method will return to the Main program, where it will then print out the Statistics of the simulation

I chose threads to implement each stage because everything is happening at once. It also allows me to use semaphores. With semaphore, I can decide which counters and scanners are open and which are closed.

The Statistics class keeps information on how many customers have gone through the metal detectors, how many set off the detectors. It also holds how many customers already had their tickets, and how many didn’t have tickets.

I decided to make my HockeyInfo class, which holds almost all information about the simulation, a singleton because it was important to only have one instance. If there was more than 1, then the program would be losing or creating new customers.

UML diagram

Tickets

States

HockeyInfo

Customer

NoTIckets

MetalDetectors

TicketCounter

Scanner

Statistics

HockeySim

Main