SDS 322 - Introduction to Scientific Programming
Christian R. Garcia - CRG296

Contagion Final Project

Abstract

Contagions have the power to throw society on its knees or to spark a mass hysteria. The power a disease has is based on its transmission however. To study this phenomenal we can simply simulate different situations, giving a disease different transmission properties, having society prepare with vaccinations, or to study how a disease might travel in a high density area versus low density. With people bumping into 10, 20, 100 people in the most dense of cities. We can do this with code to get valuable data on disease and its power to effect a society.

Code Breakdown

The code involved in this simulation is rather easy to model discrete-ly and that's just what I did. The code revolves around a main "population" class with a settable about of people (nPeople), this is reflected in code with a plethora of Person classes within a Population class. The FinalLib.cc holds most of the code needed to run each simulation, it holds the Person and Population class, each with a multitude of methods that allow for simulation.

The Person class is constructed with no arguments and the only thing set is that "healthStatus" is equal to 0, which is equivalent to being susceptible to disease. This healthStatus changes based on the contagion, any number greater than 0 signifies that the person is infected and has those many days until they have recovered. -1 is recovered and is set after a person is infected and has waited the designated amount of time to recovery. Along with that there is -2, which is set by a method at the beginning of the simulation, simulating the part of the population that would be vaccinated. The Person class has 4 methods - infect(), vaccinate(), update(), and statusString(). As the title says, the infect method infects the person if they're still susceptible to disease. Vaccinate() protects the person from getting infected. Then there is update() that when called moves forward a "step," essentially running down the time it takes for an infection to go away, so after 5 update cycles someone that was just infected would go from infected and back to recovered. These methods all get given references as inputs so that the main Population class can keep track of how many people are infected, susceptible, inoculated, and recovered at run time. Then there is the statusString() method that displays through cout the condition of each particular person in one of two modes. The first is verbose, so using words to display what the person's condition is, or the second that simply

displays a symbol for each status someone has.

The Population class is set-up with the characteristics of the simulation such as the probability of the disease to transfer, amount of population vaccinated, people in population, and the amount of random encounters someone has with others. These arguments (sans number of people) all have default arguments that allow for the code to run even without some things declared. The constructor also creates a few variables that will keep track of the states of the population in real time and of course it creates a vector of "Persons." The Population class has 7 methods. The first is randomVaccinations() that when called vaccinates a percentage of the population by using the person.vaccinate() method talked about earlier. Then there is randomInfection() that finds a random person and infects them, allowing the contagion to start. Update() is the next method, it goes through each Person in the populents vector and updates them with the person.update() method. After that is populentsDisplay() which calls on each person to display their non-verbose statusString() allowing for an each to view output of the contagion. Then there is infectPeople() which has two modes, a mode to infect only neighbors to the infected, or to infect through random encounters. Finally there is runSim() which brings everything back together. It vaccinates people based on the values population was set up with, it then infects one random person. The method then, if nPeople < 25, gives a nice display of symbols for each person in the population with populentsDisplay(). If nPeople > 25 then amounts are simply given for each category of person. The method uses update() to go to the next step, or day, in the simulation. Then the method uses the infectPeople() method to infect anyone during that day, finally the method goes to the next day in the contagion.

Main programs 2 through 5 are all rather similar, simple cout's and cin's to get user input for the contagion, then the initialization of srand(time(NULL)) to allow for quasi-randomness, setting up the population with the user inputs, and finally running the simulation with whatever flags might be necessary in infection type. Main program 1 on the other hand is less high level and simply interacts with one person class directly, it simply creates a single person and infects said person if their luck is bad. Through each step person.statusString(true) is used to give a verbose display of what is happening, once the person is recovered the code is done. Rand() in this exercise was not random as I felt like that's what was wanted, it could be easily implemented with srand(time(NULL)) to give unique seeds to the random function.

If you need any help with running the code, the code has a makefile with instructions and also a readme with instructions.

Exercise Questions

EX 39.3 - Run a number of simulations with population sizes and contagion probabilities. Are there cases where people escape getting sick?

- Yes there are, in cases that the probability of transmission of the disease is too low a neighbor to an infected person could just luck out. And as long as one person doesn't get sick then anyone that's neighbors to the lucky person (sans the infected person) are going to get off scot free due to how the disease is spreading (Neighbor to neighbor rather than through random encounters).

EX 39.4 - Describe the effect of vaccinated people on the spread of the disease. Why is this model unrealistic?

- The effect is that there are basically people that act as "walls," due to the fact that an infected person can only transmit the disease to a random person they're unable to spread the disease past people who are vaccinated. However, in real life this circumstance just doesn't happen unless someone is quarantined and only vaccinated people meet them. In real life people are able to communicate with whoever and possibly transmit disease to anyone, it's random rather than set.

EX 39.5 - Record how long the disease runs through the population. With a fixed number of contacts and probability of transmission, how is this number a function of the percentage that is vaccinated? Investigate the matter of 'herd immunity': if enough people are vaccinated, then some people who are not vaccinated will still never get sick. Let's say you want to have this probability over 95 percent. Investigate the percentage of inoculation that is needed for this as a function of the contagiousness of the disease.

- As more and more people are vaccinated the quicker the contagion dies off. Unable to contact as many people due to a lower amount of people in the population that can receive the disease, thus less carriers that can further spread the disease.
- The more contagious the disease the more people that need to be vaccinated in order to have herd immunity affect many people. If more people are not vaccinated then there's more host to spread the disease more easily with a higher transfer probability.