Simulation of the Spread of a Contagious Disease

ENGR 102-503

Carter Thompson
Mason Medrano
Christian Bourgeois
Alex Wachowicz

“An Aggie does not lie, steal, cheat, or tolerate those who do.”

On our honor as Aggies, we swear that we have neither given nor received unauthorized assistance on this assignment.

Mason Medrano

Christian Bourgeois

Carter Thompson

Alex Wachowicz

Plan of Work

Everyone:

- Learn object-oriented programming
- Decide on broad design of program (lists vs. object-oriented)

Mason Medrano:

- Design Person Class
- Responsible for class methods and constructor
- Design object attributes for use in main program

Carter Thompson:

- Design main program by using imported module
- Responsible for designing simulation and reporting any statistics involved

Christian Bourgeois:

- Design any external class functions required by main program
- Focus on debugging module and main program and ensuring correct statistics

Alex Wachowicz:

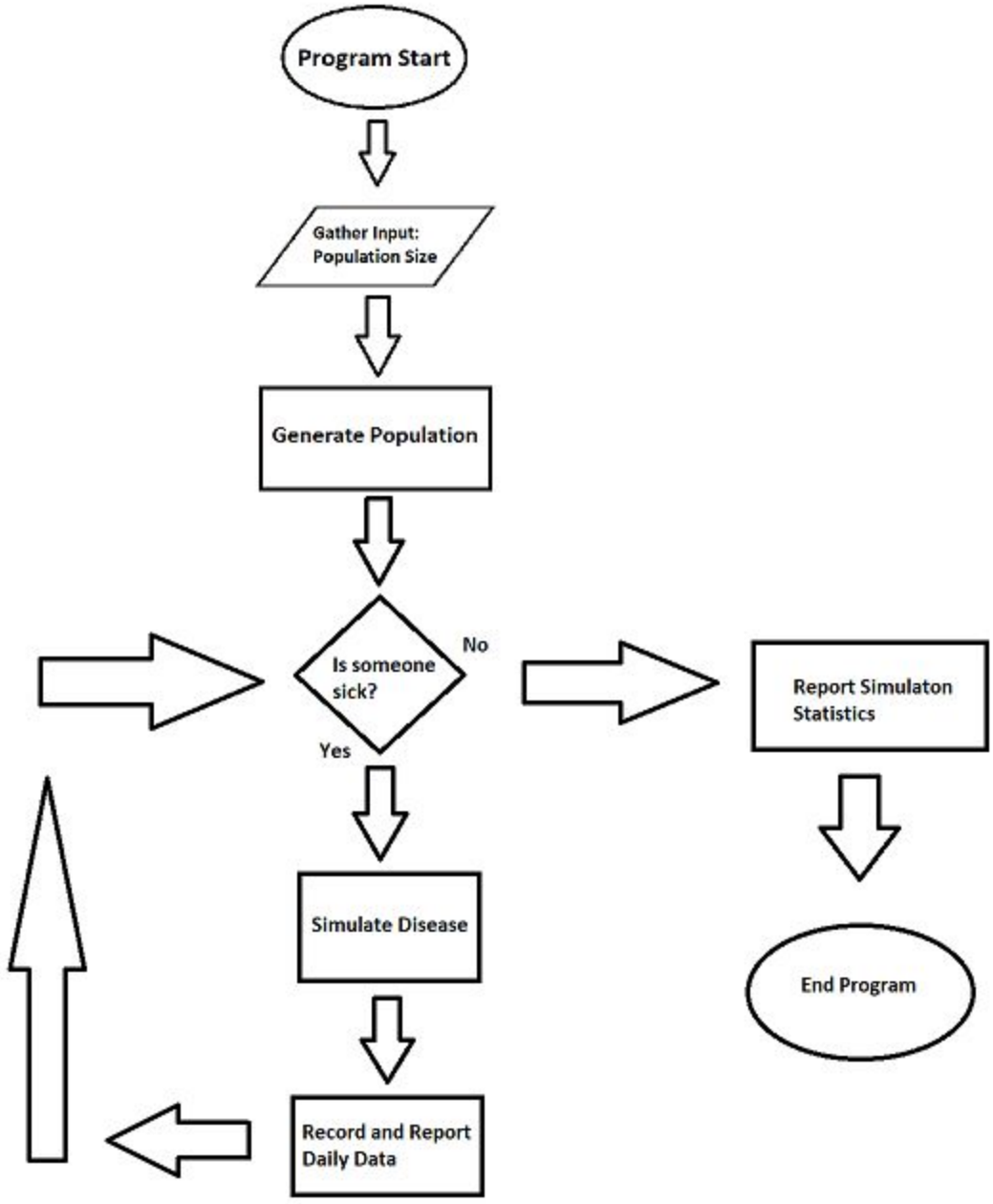
- Analyze test cases
- Report simulation behavior and differences between multiple simulations

Timeline:

- 12/3 - Complete Person Class design (first meeting)
- 12/5 - Complete disease simulation portion of main program (second meeting)
- 12/8 - Complete statistics portion of main program
- 12/9 - Analyze program and complete report
- 12/10 - finalize report and program and turn in (last meeting)

Problem Solving Strategy

The purpose of this program is to generate a scenario in which a single individual spreads a contagious disease throughout a population of variable size. The simulation will terminate when the disease can no longer be found within the population. Below is a flow chart to outline the basic function of the program.



Bottom-Up Design

Required Variables:

- DailySickList - number of sick people each day
- dailyImmuneList - number of immune people each day
- dailyRecoveryLIst - number of people who recovered each day
- dailyTransmissions - number of transmissions each day
- dailyDeadList - number of people dead to the disease each day
- Population List - a list containing objects that each represent an individual
- Population Size - the size of the population inputted by the user
- **Object attributes:**
    - resistance(integer): The likelihood for an individual to resist the disease
    - recovery(integer): The number it takes an individual to recover from infection
    - activity(integer): The number of people an individual interacts with daily
    - transmission(integer): The likelihood for an individual to transmit the disease
    - deathRisk(integer): The likelihood for an individual to die from the disease
    - isImmune(Boolean): The qualifier that tells if an individual is immune
    - isSick(Boolean): The qualifier that tells if an individual is sick
    - isDead - boolean that determines if an individual has died to the disease
    - numTransmissions(integer): The number of time a sick individual transmitted the disease
- **Statistics per day:**
    - day - Day counter
    - dailySick(integer): The number of sick people on that day
    - dailyRecovery(float): The average recovery time for the sick population on that day
    - recoveryTime(integer): The number of times an individual recovered a day from the disease
    - immuneCount(integer): The number of immune individuals on that day
    - dailyTransmission(integer): The number of transmissions on that day
    - dailyDead(integer): The number of dead individuals on that day
- **Statistics for whole simulation:**
    - Recovery Time Average(float) - Total average recovery time
    - Transmission average(float) - Average number of transmissions per infection
    - ratioSick - Fraction of population that became infected
    - maxSick(integer) - Max number of individuals infected at once
    - totalInfected(integer) - total number of people who got sick
    - maxDead(integer) - the number of people killed due to this disease
    - day (but at the end of the sim) - Total number of days in the simulation

Required Class Methods:

- __init__ - constructor for the Person class
- makeSick - changes an object's 'is sick' attribute to true and decreases the individuals activity level
- makeContact - compares 2 ‘members’ of the population, and determines if they infect each other
- recover - reduces recovery value, enabling people to become immune, and presents the chance for an individual to die to the disease

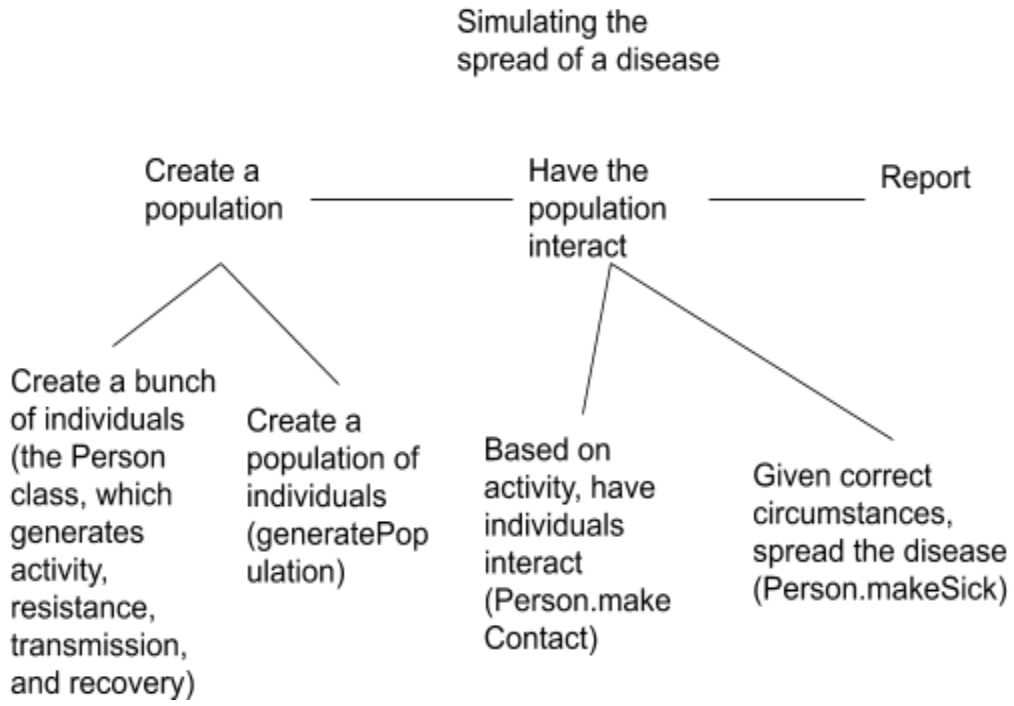
Required Functions:

- checkDisease - determines if at least one member of the population is sick, and returns the corresponding boolean value
- generatePopulation - creates the population to be represented in the simulation
- gatherStats - calculates the statistics of the population everyday throughout the simulation

Top-Down Design

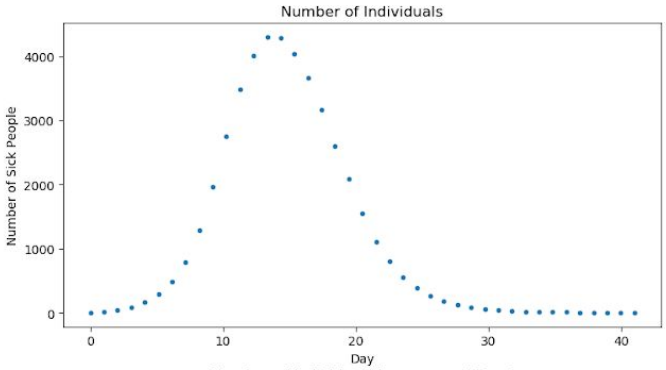
The time step will be 1 day.
Activity will represent how many interactions a person starts per day
Recovery will start decreasing per day once someone is infected, they will be immune afterwards (when recovery = 0)
Distinct values per person: Recovery, Transmission, Activity, Resistance, Immunity, DeathRisk, etc.
Resistance, transmission on a scale 1-10
Complexities to simulate disease:

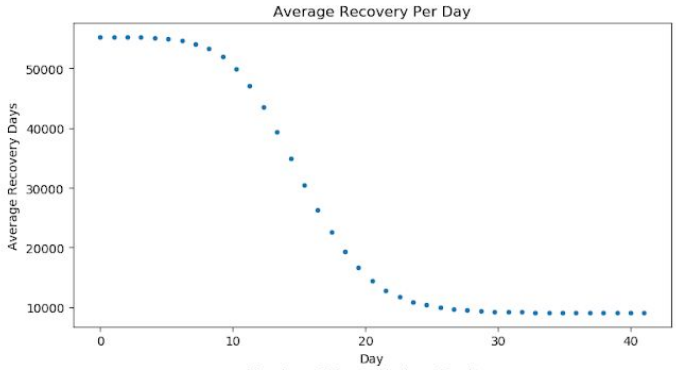
- Individuals have varying chances to die to the disease
- Lethality of disease
- Decreased action when sick
- Attributes of a population(i.e. The social behavior or physical characteristics of people)

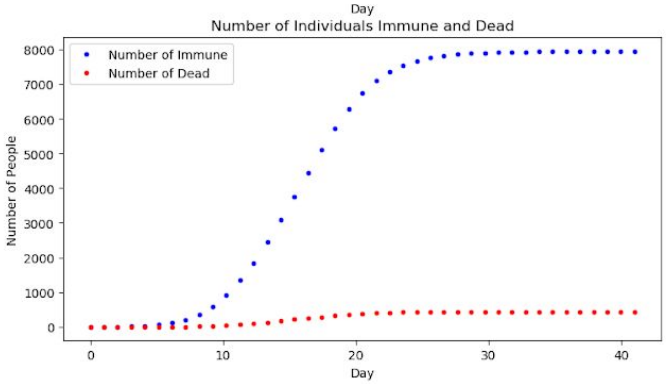


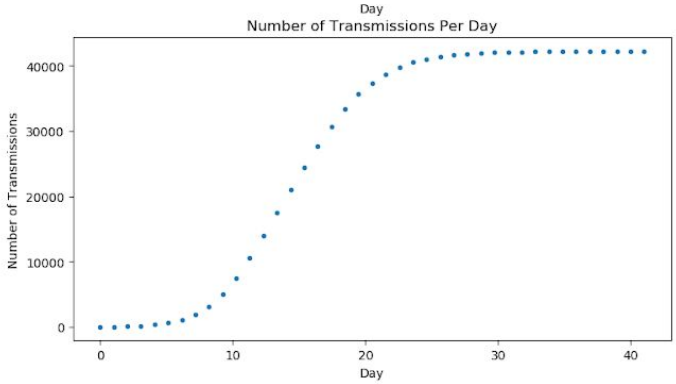
Program and Results Analysis

Sample Results for a Population of 10,000 People









The disease was eradicated by day 42.
4298 was the highest amount of people infected in one day!
441 was the number of people killed due to this disease.
8391 is the total number of people who got sick.
83.91% is the percentage of the population who was infected.
5.64 is the average recovery time in days.
5.31 is the average number of transmissions per infection.

Compare Multiple Simulations

Min/Max Data for Recovery Range in 1-10 Days over 5 Trials:

- The disease was eradicated within range from day 37 to day 50.
- The maximum sick in one day ranged from 4277-4423.
- 445-460 was the number of people killed due to this disease.
- 8359-8491 is the total number of people who got sick.
- 5.6-5.64 is the average recovery time in days.
- 5.30-5.33 is the average number of transmissions per infection.

Min/Max Data for Recovery Range in 1-4 Days over 5 Trials:

- The disease was eradicated anywhere from day 34 to day 56.
- The maximum sick in one day ranged from 701-914.
- 94-137 was the number of people killed due to this disease.
- 4099-4486 is the total number of people who got sick.
- 2.52-2.54 is the average recovery time in days, which makes sense considering the program is generating a random value between 1 and 4.
- 2.75-2.8 is the average number of transmissions per infection.

Analysis:

In general, altering the recovery time had little effect on the timing of the spread of the disease. The shortest simulations hovered around the mid-thirties, and the longest the mid-fifties; as expected from that trend, most ‘strands’ of the disease survived for about forty-five days. Though the survival time was barely affected, the maximum number of people with the disease at one time varied greatly with altered recovery times. In a population of 10000 people with recovery time anywhere from 1-10 days, about 4300 people usually became sick all at once. However, with recovery time shortened to 1-4 days, no more than one thousand were ever sick at once. Total number of infected individuals was also slashed significantly, with an average of about 8400 being cut down to about 4200. Finally, the average number of transmissions per infection was cut down a fair amount due to there being fewer opportunities to infect people with the shorter recovery times.

Change the Simulation Behavior

In order to make our simulation more realistic, our group modified various traits of our disease and population. The first major change was to cut the activity level of each individual based on their sickness. Within our Person class, we structured our makeSick method to floor divide an individual's activity value by two when they became infected. This added a new layer of complexity, cutting down the opportunities for healthy individuals to become sick pretty significantly. The second change was to add lethality to the disease. This factor wasn't super fine-tuned for realism, and resulted in about a 4% mortality rate with default values of recovery, transmission, resistance, and activity. If we were to explore this idea further, we would consider having individuals with longer recovery times become more likely to die from the disease, among other options. In addition, it is easy to change the program to simulate different aspects of the disease and population. This concept is very malleable and can be applied to many different areas. For instance, in a population filled with solely secluded introverts, the transmission and activity levels of each person can be changed to reflect the behavior of the group.

External Modules

- random.random() - generates a random number between 0 and 1; used to generate various random values for the program
- math.ceil(), math.floor() - used to round randomized values to integers, making them much easier to use
- matplotlib.pyplot.plot(), other plotting functions - used to plot statistics gathered from the simulation
- numpy.linspace() - used to generate x values for plots

Summary of Team's Work

The original plan consisted of designing the basic program for the simulation and then adding complexities to reflect a real situation more accurately. We accomplished most of the original plan by completing the basic program and adding a few complexities, such as introducing lethality, changing activity, and modifying recovery. However, some aspects were not implemented, such as having healthy individuals interact with sick individuals.

The largest problem the team faced was implementing functions to track statistical data. There were many components involved and measured, so the statistical analysis became messy. Another problem was the overarching design of the program. We were not sure whether or not to learn object-oriented programming or stick to what we know and use lists and dictionaries. The group decided to use object-oriented programming, which simplified a lot of the program. The group did not make any assumptions that affected our final product.

Team Members' Contributions:

Mason Medrano - Mason was the most familiar with the concepts involved in object-oriented programming. He designed and implemented the Person class, including all of its methods that handled most of the brunt work of the program.

Carter Thompson - Carter was responsible for designing the main portion of the program involving the simulation of the disease. He used the class methods and functions to not only simulate the disease but also analyze and report various statistics from the trial.

Christian Bourgeois - Christian designed the functions used outside of the Person class. This includes generatePopulation() and checkDisease(). Christian also played a large role in debugging the program and ensuring the statistics functions reported correct data.

Alex Wachowicz - Alex mainly focused on formulating and running test cases and analyzing data from cases with modified recovery times. He reported the differences between multiple simulations where recovery times varied from a few days to almost two weeks, giving us a deeper understanding of our program.

In retrospect, analyzing and reporting the statistics involved with the simulation would have been easier if we had started with functions. Initially, we programmed the calculations for the statistics into the main body, which made the code overly complex and difficult to read. A lot of time was spent converting this portion from the main body into functions. Despite these issues, we were still able to design a program with clearly defined and easily manipulable code, particularly in the class methods and constructor.