Fall 2019 ME459 Final Project Report
University of Wisconsin-Madison

Coronavirus Spread Simulation

Jacob Nellis

December 14, 2019

Abstract

This report documents the creation of a Covid-19 Community Spread simulation. The motive behind making this simulation is to give users the ability to see how different parameters such as social distancing and social network contacts affect the outcome of a pandemic. The final simulation gives users the ability to create rational conclusions about different simulation parameters that accurately reflect choices that individuals and society as a whole must make in the real world.

Link to Git Repository:

Here is my link to my entire class git repository:
https://euler.wacc.wisc.edu/jacob-nellis/me459-jacob-nellis.git

Here is a link to my FinalProject folder in my class git repository:
https://euler.wacc.wisc.edu/jacob-nellis/me459-jacob-nellis/tree/master/FinalProject

Contents

1. General information

1. Mechanical Engineering, UW-Madison
2. Accelerated MS (1 year, class-based program)
3. Group Members:
    - Jacob Nellis
4. Statement of Release of Code:
    - I release the ME459 Final Project code as open source and under a BSD3 license for unfettered use of it by any interested party.

2. Problem statement

Covid-19 has had a huge impact across the entire globe over the past year. Between writing the proposal for this project and today, the U.S. has seen an additional 7 million infections and 71 thousand deaths [1]. The daily life of an average American today would have seemed almost unimaginable a year ago. Throughout this crisis, many Americans have struggled to understand what public health measures are the most effective and what should be done to help slow the spread of this disease. Furthermore, some citizens in both the U.S. and other parts of the world have criticized many of the public health measures that governments have put in place to try to control the spread of Covid-19. Mandatory lockdowns, mask use, closing schools, and enforced social distancing have been especially controversial in the U.S. This computer program seeks to simulate the pandemic given different levels of various public health initiatives such as social distancing, travel limitations, and personal hygiene. Having an easy-to-use program that can demonstrate the effects that these parameters have on the spread of Covid-19 will allow people to better understand what should be done during this pandemic and hopefully buy into some of these measures.

3. Solution description

This project was split into four important milestones in the proposal. The first milestone was to clearly define the parameters for how this simulation will run. Next, a skeleton code was created that has the desired baseline structure of the simulation. Next, a user interface was implemented so that the program was easy to control and understand. Finally, visualization tools were added so that the user could see the spread of Covid-19 in real time. Each milestone is explored further in this section in order to create a comprehensive picture of the project.

Simulation Parameters and Data Collection

While no simulation is perfect, we can gather specific information about a virus to determine what the main drivers of its spread will be and we can then use those parameters to create a model that will hopefully mimic the results that we would observe for this virus in the real world under various scenarios.

The first parameter in the simulation is the **time step**; this is the smallest increment of time that the simulation will use to determine new infections and deaths. Having a timestep that is too small, will create unnecessary calculations for the program and it will run slowly. However, a timestep that is too large will not accurately model real world results. For Covid-19, the average time from infection

to the onset of symptoms is 5.7 days [2]. Additionally, the average time from the initial contagious period to no longer being contagious is 6 days [3]. This makes a time step of 6 days ideal for this simulation. If a person becomes infected in one wave, the next wave he will be contagious, and the wave after that he will have (hopefully) recovered. Currently there are no health measures that can be taken by a group of people to alter this infectious period; as a result, the time step for this simulation can not be altered by the user.

The **lethality** of Covid-19 can be hard to pinpoint as it seems to vary from population to population. On average, it appears that approximately 3% of people who contract Covid-19 have a fatal outcome [1]. Additionally, the fatality rate tends to balloon when the number of people infected begins to overwhelm a population's healthcare system. For this simulation, the fatality rate increases from 3% to 10% if the number of people infected at any one time exceeds 25% of that group's population; this spike in fatality rate is meant to reflect that an infection rate of 25% would completely overwhelm the hospitals in that region.

The **transmission rate** of a virus is its ability to spread from person to person. Covid-19, being an airborne respiratory illness, has a high transmission rate. Studies have found that after close contact with an infected person, around 53% of those in contact later tested positive [4]. Mask use, hand washing, and personal hygiene are all factors that may be able to lower this transmission rate. Therefore, transmission rate is a parameter that can be adjusted by the user of the simulation in order to get an idea for how these factors influence the spread of disease. This factor is also one that should be able to be adjusted while the simulation is running in order to reflect changes in policy that happen during the pandemic.

The **number of contacts** a person has is another major factor for the spread of Covid-19. This can be described as the number of "friends" a person interacts with on a weekly basis. A population that has an average number of contacts that is high will have a very high potential to spread the disease throughout the population. This parameter can be controlled by the user and is defined as the average number of friends each person has in a specific group.

At the start of this pandemic, many nations started to rapidly close their borders in an attempt to stop Covid-19 from entering their country. This is an important factor that should be investigated in the simulation. An **isolation parameter** is defined that represents the fraction of people that travel from one group to another during each timestep of the simulation.

The final parameters for the simulation that are unrelated directly to either the characteristics of the virus or the public policies in place are the conditions used to run the simulation. These include the population of each group being simulated, the number of groups being simulated, and the initial number of infected people within each group. A skeleton code was created using these parameters.

Skeleton Code

This project was implemented in Python, there are several reasons for using an interpreted language as opposed to a compiled language like C or C++. First, the main goal of this program was to visualize the spread of Covid-19 in an easy-to-understand way. This is arguably more important than the speed at which the program is run as long as the program can run sufficiently fast for reasonably large

populations in order to keep the user's interest. Python has many built in libraries for visualization and graphical user interface (GUI) implementation that makes it ideal for making this type of program. Additionally, this simulation is centered around creating groups of people in a simulation environment. An object-oriented language is ideal for this structure because we can create many "person" objects and place them into "group" objects, which can then be placed in a final "sim" object. This approach has proven to make writing the code substantially less confusing.

The source code for this project is split into three files; group.py, main.py, and sim.py. Each code is created to handle a specific part of the program. The general setup for a simulation is as follows. First, a simulation object will be created with the parameters set by the user saved in its attributes. A group object is then created for each group. A person object is created for each person in each group. Each person is assigned a list of friends that creates a social network for the group. Figure 1 shows both the structure of the simulation and the social network that is created in each group.

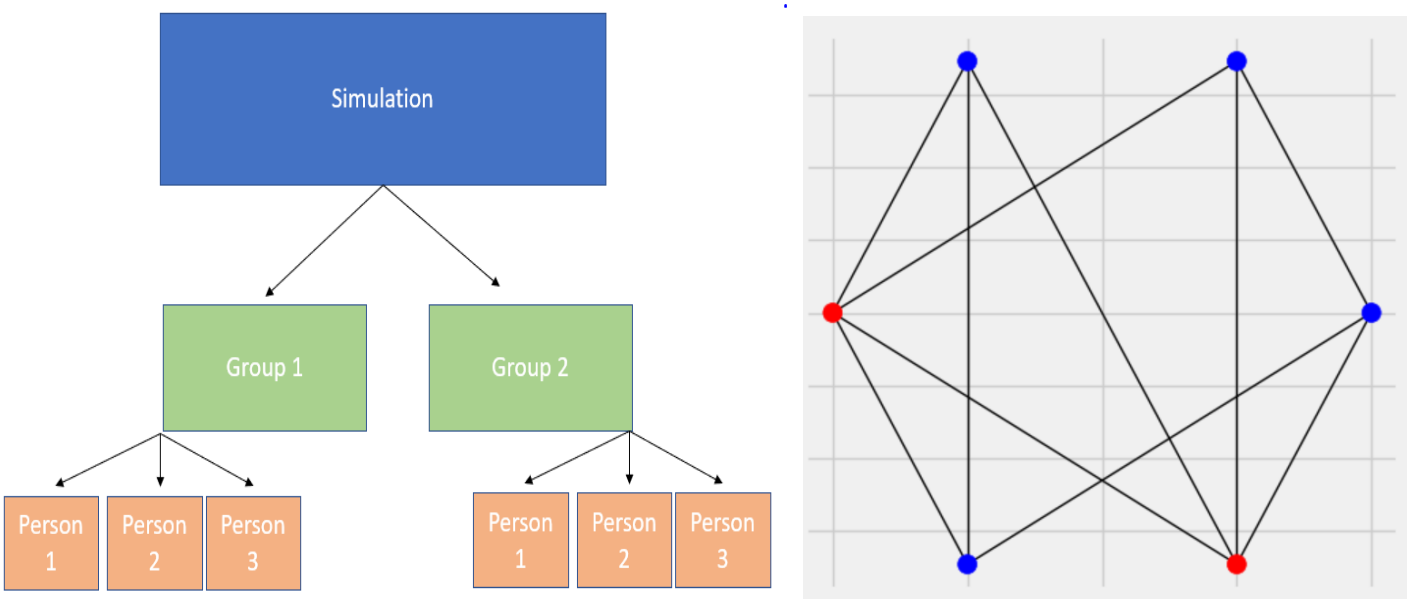


Figure 1. Organization of objects in the simulation (left), and organization of people and friend relationships (right). Dots represent people and lines represent social connections.

When the simulation increments forward one timestep, a function called wave() is called that updates the simulation. First, it is determined which people in the groups will travel to a new group. Two people are randomly selected, and the Boolean variables that define the two peoples infectiousness status are switched. Next, new infections are identified by looking at each infected person's friend list; based on the transmissibility rate, it will be determined which of an infected person's friends will be infected. Additionally, if a person is recovering from infection, it will be determined if that person had a fatal outcome. Finally, the counts of infected, recovered, and dead people are updated in each group using a double nested for loop to iterate through each person to check their status.

User Interface

Once the structure of the code was completed, a working graphical user interface (GUI) was created so that the user could easily set the parameters for the simulation. This is done in the main file and the python library tkinter was used. This library allows for the creation of 'widgets' that can be placed onto a window that pops up when the program starts. Figure 2 shows the parameter settings window that appears when the program starts.

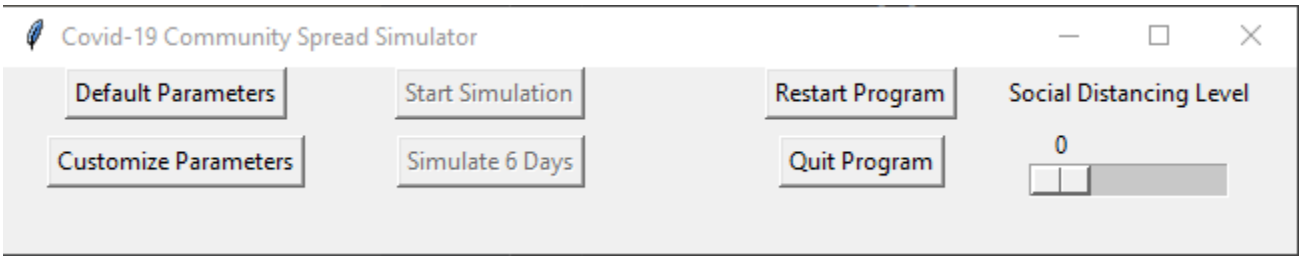


Figure 2. Simulation startup Menu

The user initially has two options, "Default Parameters" and "Customize Parameters". The default parameters button allows the user to quickly start the simulation with parameters that are pre-determined in order to show interesting behavior when run. The customize parameters will bring up a new menu that allows the user to create his/her own groups to be run in the simulation. An example of this is shown in Figure 3

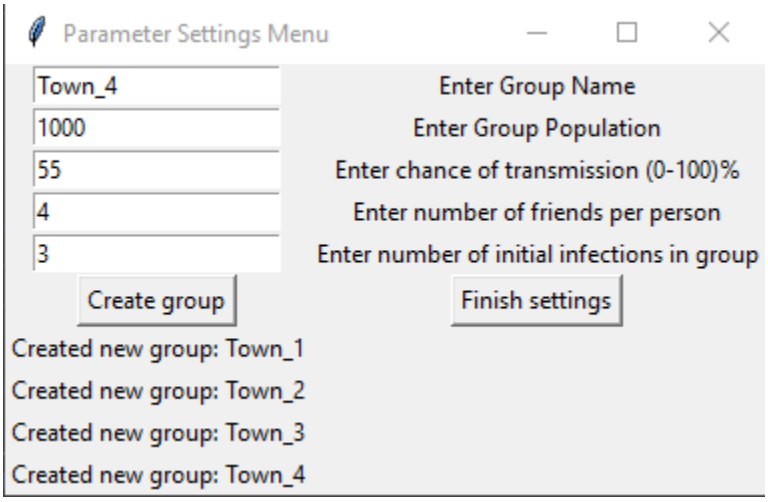


Figure 3. Custom Parameters menu being used to create four groups

After the settings have been configured, the user can press start simulation to begin the simulation. Once started, the user can increment the simulation six days at a time by pressing the "Simulate 6 Days" button. Buttons were also added to reset the simulation parameters at any time during the system and to exit the program.

Visualization

A simulation is not especially useful if the results are not clearly displayed. There are two important displays that allow the user to monitor the status of his population during the simulation. The first is a

plot of currently infected, recovered, and dead people in the entire simulation plotted against time. This plot was created using the python package matplotlib. The simulation object keeps a log of infected, recovered, and dead people that is updated after each time step; therefore, plotting these data was not especially difficult. A canvas widget was created using tkinter so that the plot could be displayed in the same window as the options dialog. It was initially difficult to update this plot as the simulation progressed. However, this issue was solved by creating an animation function "animate(i)" that is called at a specified framerate "i". This function clears the plot and replots the data every 100 ms to make a plot that seems to adjust and change in real time. This approach is not the most efficient way to plot data, as clearing and replotting takes more time then adding a single point; however, I found this to be the most straightforward way of implementing this plot after a substantial amount of troubleshooting alternatives.

The second visualization that is shown to the user is a network of nodes and edges that represent the people in a group and their social contacts. Initially, all the nodes are blue except the initially infected people. Blue indicates that these people are susceptible to infection. When a person is infected, the node will turn red. When they have recovered or died, their node will turn green or black, respectively. This creates a unique way to see how the virus spreads throughout a population based on the social connections that exist within the population. This plot is computationally intensive to generate and becomes very cluttered (providing no real information) if there are more than 100 nodes. For this reason, the node network that is displayed is only for the first group that is created and is only displayed if the population is no more than 100 people. With the code fully functional, the next section discusses some of its results and their implications.

4. Overview of results and demonstration of project

The purpose of this project was to visualize how each parameter in the simulation effects the spread of disease. By running this simulation with different parameters and comparing outcomes, it became clear which factors were the most important in terms of controlling a pandemic.

Herd immunity has been discussed a lot lately. One factor underlying herd immunity is the percent of a population that must become immune to infection in order to stop further spread to susceptible people. Figure 4 shows a simulation that demonstrated this idea well. The virus had completely died out in this simulation and 20% of the population never became infected; they were protected by herd immunity. This suggests that for the specific parameters used in this simulation, about 80% of a population needs to become infected in order to effectively end the pandemic. If we extrapolate this to the United States, currently there are around 16.5 million Covid-19 cases for a population of 328.2 million people. This means that around 5% of the population has become infected over the course of a year. Without a vaccine, this suggests that the pandemic will continue for an additional 15 years before herd immunity is achieved!

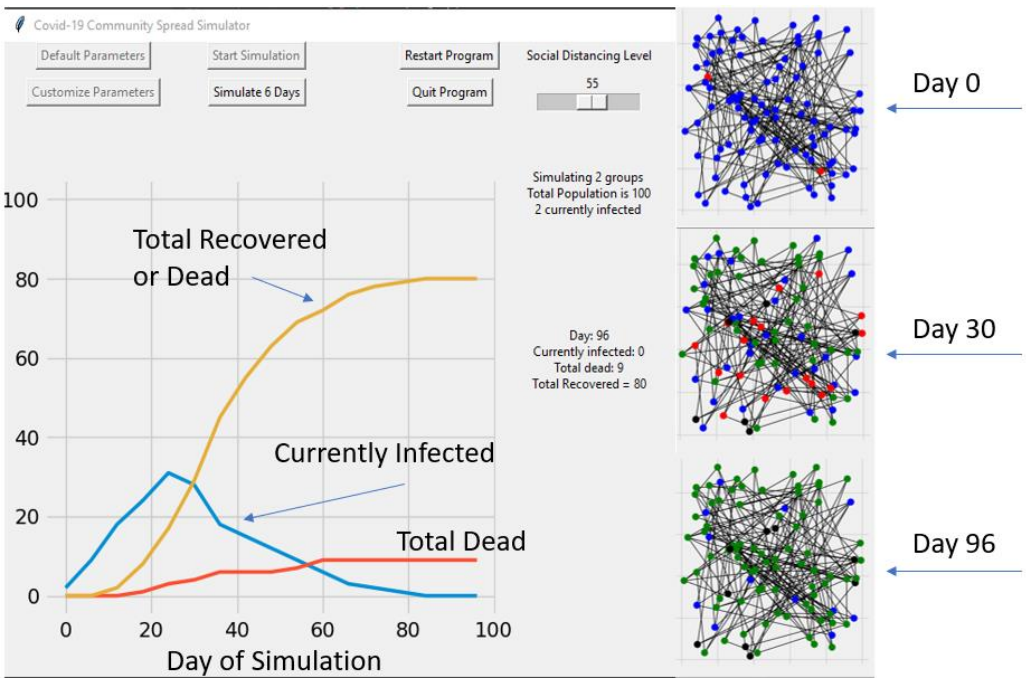


Figure 4. Simulation of a group with 100 people, 2 are initially infected. Each person has an average of 3 friends. Transmissibility rate is set to 55%. The social network is displayed at day 0, 30, and 96 of simulation.

Closing connections between populations (for example, borders of countries) is something else that's been in the news. A related factor that could be explored using this simulation is how limiting the entrance of new people into a group from other groups affects the spread of the disease. In order to show the effect that group isolation has, a simulation was created with two groups. One smaller group (group 1) has 1,000 people and starts with 30 people initially infected. The larger second group (group 2) has a population of 10,000 people, none of whom are infected. When there is no travel between the two groups, the outcome is rather obvious. The virus spreads through group 1 very quickly and dies out. However, if we set the isolation level to only 0.05% (a value that would be extremely hard to enforce in the real world), we can see that once the virus enters group 2 it quickly spreads throughout that group and further isolation has small influence on the spread. These two simulations are shown in Figure 5.

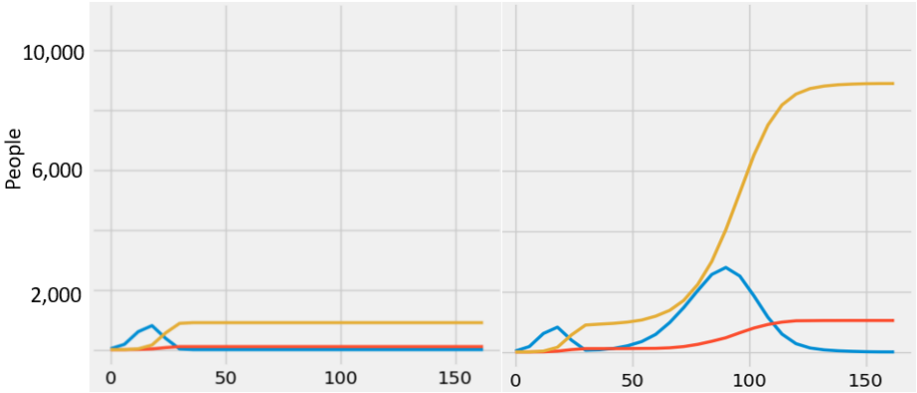


Figure 5. Two simulations comparing isolation levels; 0% travel is on the left and 0.05% travel is on the right.

This result leads me to believe that the only benefit that isolating a group has in a pandemic is additional time to prepare for the virus before it inevitably enters the population. Isolation is not enough to completely avoid the virus, however, as it potentially only takes one infectious person to enter that will cause the virus to spread.

The number of contacts a person has also had a substantial effect for how the pandemic develops. This is shown by comparing two simulations of 10,000 people. The first simulation has 3 friends per person, and the other has 8.

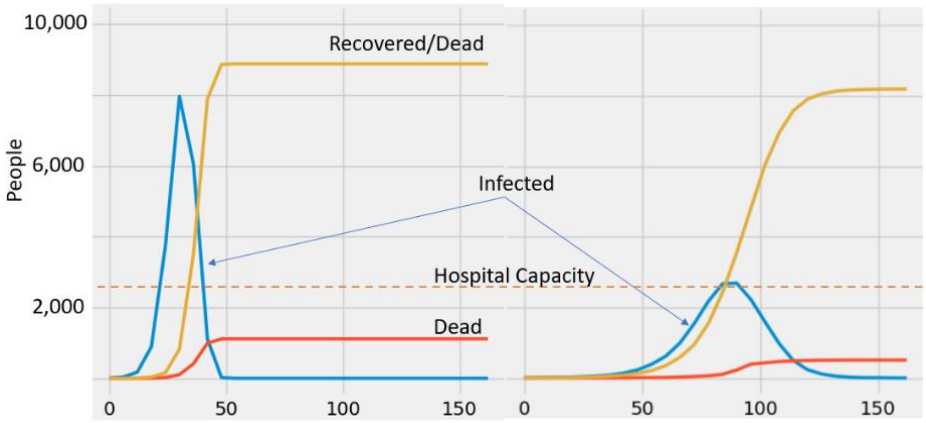


Figure 6. Simulation with 8 friends per person (left) and simulation with 3 friends per person (right)

There are a few important things to note in Figure 6. First, having more friends resulted in a far faster infection rate. This is particularly bad because it means that everyone is getting sick at essentially the same time, putting enormous strain on healthcare facilities. The total people that ended up becoming infected is also higher, but not by a large margin. The main difference is the rate of spread. Because the spread was slower for the more socially distanced group, they did not exceed the hospital capacity for most of the outbreak, which resulted in a lower number of deaths.

The final parameter that was looked at was transmissibility. Recall that this parameter captures the impact of a group's mask use, hand washing, and personal hygiene. This was also the one parameter that can be adjusted during a simulation in order to mimic social policies that might be put in place because of high community spread. Figure 7 compares a group that has a constant transmissibility rate to a group that turns their transmissibility up or down based on the current number of infections.

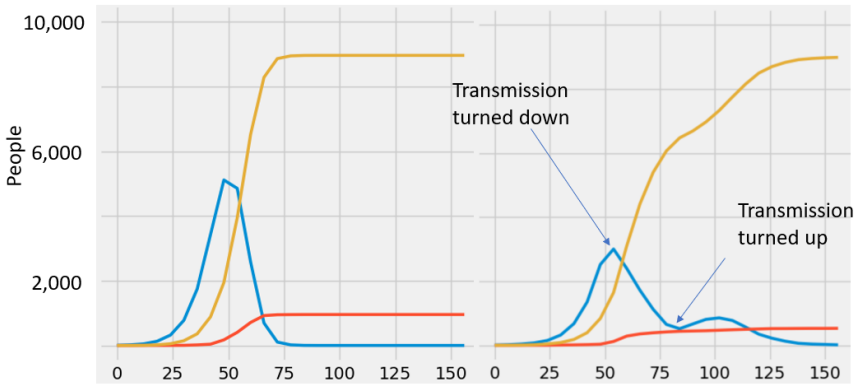


Figure 7. Simulation with constant transmission (left) vs. simulation where transmission is turned down when infected people reaches 25% of population (right).

The results that were obtained by the simulation make me feel confident that the code works as desired. It is a good visualization tool that helps people explore how different parameters affect the spread of a virus during a pandemic. Throughout this process, I learned a lot about pandemics and their behaviors (in addition to computer coding). Hopefully soon we will have a vaccine that will bring the susceptible population down to levels that will eradicate this disease.

5. Deliverables:

Below is a list of the following deliverables for my final project.

- This report
- A doxygen html (Doxygen_HTML.zip) describing my code, submitted to Canvas
- Final Code in Git Repository
    - In the Git repo there are my 3 source code files (main.py, group.py, and sim.py). To run this code you should run the main file (python3 main.py).
    - An environments file is also added that describes the dependencies required to run this program (matplotlib and networkx).
        - Note: I had trouble getting this to run through ubuntu. I fixed this issue with: (sudo apt install python3-tk). I am not sure what the problem was but I guess tkinter may not be downloaded into python by default when using ubuntu.
- Environment.yml file for running code

6. Use of ME459 knowledge

- Github repository maintenance (committing, pushing, cloning, etc.)
- Interpreted Languages (Python)
- Object Oriented Programming
- Doxygen commenting and HTML/Latex creation
- Debugging
- Virtual Environments
- Complexity analysis to increase execution speed (rearranging loops to improve spatial locality)
- Variable Types and Casting
- Common Programming algorithms

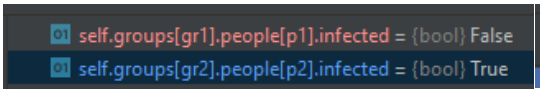
- Local and global variable handling

Debugger Example

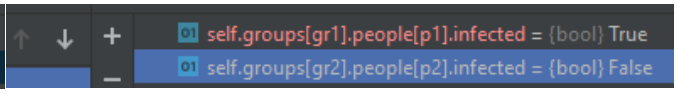
When testing my code, I initially found that when simulating two groups, one of which has no infections, no matter how much travel occurred between the groups the group with no infections would never become infected. My initial thought was that my travel function was not correctly switching the infection status between the two traveling people. I used the Pycharm debugger to stop my travel function at the point where the people are supposed to switch groups, as shown below.

```
182
183     end1 = self.groups[gr1].people[p1].infEnd
184     end2 = self.groups[gr2].people[p2].infEnd
185     inf1 = self.groups[gr1].people[p1].infected
186     inf2 = self.groups[gr2].people[p2].infected
187     self.groups[gr1].people[p1].infEnd = end2
188     self.groups[gr2].people[p2].infEnd = end1
189     self.groups[gr1].people[p1].infected = inf2
190     self.groups[gr2].people[p2].infected = inf1
```

Before:

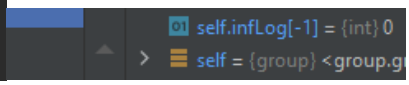


After:



It turned out that the infected status was correctly switching between two people in different groups so this was not the problem. After stepping through the code further, I found that the problem occurred in the updateCounts() function.

```
128 def updateCounts(self):  self: <group.group object at 0x000002136C5B7DC0>
129     """!
130     Will look through the people in the group and will count
131     the number of people infected, dead, and removed in the population.
132     @param self: The object pointer
133     """
134     if self.infLog[-1] != 0:
135         numInf = 0
136         numDead = 0
137         numRem = 0
138         self.infNodes = []
139         self.deadNodes = []
140         self.remNodes = []
```



In this function I was trying to speed up execution time by not running the for loops to recount everyone's status if nobody in the group was infected in the last wave. This results in a logic error when a person who is infected travels into the group. The counts have not yet been updated so the infLog list still thinks that nobody is infected and therefore it does not bother to check. This was solved by setting the infLog of a group to 1 if they had no infection but an infectious person traveled into the group.

7. Conclusions and Future Work

There are a few places where this code could improve. The biggest one is the speed of the simulation. When the number of people exceeds approximately 20,000 people then the initialization slows dramatically. Additionally, when the number of infected people increases the speed of the simulation also slows down. This is because the checkInfections() for loop iterations continues to exponentially increase. This issue is shown in Figure 8 which shows the computational time per step as a function

of simulated time. The peaks in Figure 8 correspond directly to when the number of infections is very high.

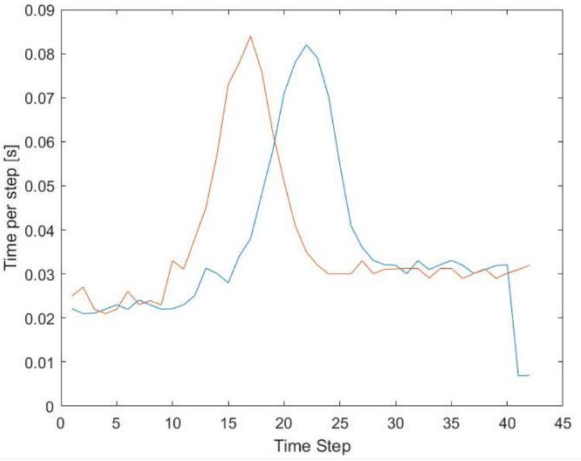


Figure 8. Time to increment simulation by one timestep vs timestep of simulation

Another area of improvement would be the ability to adjust the number of friends that each person has while the simulation is running. A slider that the user could adjust that would immediately alter the social network diagram to show either more or less connections would be very interesting and would allow investigation into government mandated lockdowns.

Lastly, there are minor things in the display that should be improved upon. The plot that is displayed to the user has no axis labels and numbers can occasionally be slightly cutoff. This was a consistent struggle to fix as I could not find a reliable way to add labels that would fit onto the plot without having them overlap.

Overall, I would say that this code does what I set out to accomplish.

References

[1] "United States Coronavirus: 15,232,774 Cases and 289,430 Deaths - Worldometer." https://www.worldometers.info/coronavirus/country/us/ (accessed Dec. 07, 2020).

[2] "Estimated Incubation Period of COVID-19," *American College of Cardiology*. http%3a%2f%2fwww.acc.org%2flatest-in-cardiology%2fjournal-scans%2f2020%2f05%2f11%2f15%2f18%2fthe-incubation-period-of-coronavirus-disease (accessed Dec. 13, 2020).

[3] CDC, "Coronavirus Disease 2019 (COVID-19)," *Centers for Disease Control and Prevention*, Feb. 11, 2020. https://www.cdc.gov/coronavirus/2019-ncov/hcp/duration-isolation.html (accessed Dec. 13, 2020).

[4] C. G. Grijalva, "Transmission of SARS-COV-2 Infections in Households — Tennessee and Wisconsin, April–September 2020," *MMWR Morb. Mortal. Wkly. Rep.*, vol. 69, 2020, doi: 10.15585/mmwr.mm6944e1.