TESTING THE ACCURACY OF A COMPUTER SIMULATION OF A VIRAL SPREAD

A
RESEARCH PAPER
SUBMITTED TO THE SCIENCE DEPARTMENT
OF
OWEN J ROBERTS HIGH SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE COMPLETION OF BIOTECHNOLOGY

PROGRAM IN SCIENCE: BIOTECHNOLOGY
SCIENCE DEPARTMENT
OWEN J ROBERTS SCHOOL DISTRICT

BY

ZACHARY LINEMAN

POTTSTOWN, PENNSYLVANIA

2022

Signature Page

This research paper was accepted as meeting the research requirement for the completion of the research component of the biotechnology course.

APPROVED:

_____ Date_____
Biotechnology Research Advisor
Mr. Richard Hampson

_____ Date_____
Mrs. Karen Neumier
Department Chairperson

Assurance of Compliance
With Pennsylvania State
Research Requirements
Owen J Roberts School District
Biotechnology Research

Date: May 19, 2022

Zachary Lineman
Student's Name (please print)

I have reviewed the thesis or departmental paper proposal submitted by the above named student and have concluded that:

____ There is no human subject involvement and no human subject research is required.

_____ Human subject research review is required.

_____ There is no animal use involved and no animal use review is required.

_____ Animal use review is required.

_____
Researcher, Z. Lineman

_____
Biotechnology Teacher

_____
Department Chairperson

Arino, J., & Van den Driessche, P. (2006). Disease spread in metapopulations. *Fields Institute Communications*, *48*(1), 1-13.

Abstract

Researchers Arino, and Van Den Driessche have explored the spread of disease across regional groups of populations (Metapopulations). The researchers have created multiple differential equations that model the spread of a disease within a population. Each of these equations covers a different model of viral infection. These will provide a concrete base for the creation of equations for this research paper. The researchers created a structural model of moving individuals into separate containers which represent their stage of infection. The strategies for implementing a viral disease model explored in the researcher's paper will provide a strong baseline for the implementation of this researcher's paper's viral infection model, especially the implementation of quarantining, movement between cities, and birth-death rates.

Key Points

1. "Travel can change disease spread in a complicated way; it may help the disease to per- sist or may aid disease extinction." (Arino, J., & Van den Driessche, P., 2006, p. 1)
2. "Once infected, a susceptible individual harbors an agent of disease and moves to the exposed compartment, then into the infective compartment as the individual becomes able to transmit the disease. On recovering

from the disease, an individual moves to the recovered compartment, and then back to the susceptible compartment as disease immunity fades." (Arino, J., & Van den Driessche, P., 2006, p. 3)

3. "Spatial spread is all the more important for diseases that involve several species, for example, bubonic plague and West Nile virus." (Arino, J., & Van den Driessche, P., 2006, p. 7)
4. "The existence and stability of endemic equilibria if R0 > 1 are open analytical questions. As in many high dimensional epidemic models, these are hard problems. It is sometimes possible to prove that the disease is globally uniformly persistent by appealing to the techniques of persistence theory." (Arino, J., & Van den Driessche, P., 2006, p. 6)
5. "Sattenspiel and Dietz [26] introduced a single species, multi-patch model that describes the travel of individuals, and keeps track of the patch where an individual is born and usually resides as well as the patch where an individual is at a given time. This model has subsequently been studied numerically in various contexts, including the e ects of quarantining." (Arino, J., & Van den Driessche, P., 2006, p. 8)

Beldomenico, P. M., & Begon, M. (2010). Disease spread, susceptibility and infection intensity: vicious circles?. Trends in Ecology & Evolution, 25(1), 21-27.

Abstract

Researchers Pablo M. Beldomenico and Michael Begon have researched the importance of the host condition and immunocompetence in a viral infection situation. In most modern studies and models, the host's condition and immunocompetence are often overlooked. These factors were often thought to not be important to the spread of viral infection and their effect on viral spread was considered disregardable. Through the researcher's research, they found that these factors are indeed important in the accurate modeling of viral infections. This research provides a key detail for the accurate reproduction of a viral infection in this researcher's viral infection model. The charts provided by the researcher's paper will be helpful for the implementation of repetitive infection and for accurate infection rates.

Key Points

1. "A key element, clearly, is host susceptibility: the more susceptible the hosts, the more rapid and widespread will be the spread of disease. Yet epidemiologists and disease ecologists have often neglected variation in host susceptibility." (Beldomenico, P. M., & Begon, M, 2010, 1)
    2. "However, in general, previous infections do not protect against new hetero-specific infections, even when parasites belong to the same group [41]. The resulting scenario for a host in poor condition is thus greater

vulner- ability to the whole parasite community, triggering a vicious circle where host health becomes increasingly impoverished and severe infection generally more prob- able, which might eventually lead to death." (Beldomenico, P. M., & Begon, M, 2010, 5)

3. "Neglect of variability in susceptibility might also result in erroneous interpretation of laboratory data, especially if these interpretations are used to draw conclusions about natural populations." (Beldomenico, P. M., & Begon, M, 2010, 2)
4. "Infections of hosts in poor condition might also be of higher intensity because parasites would encounter less opposi- tion to their survival and proliferation." (Beldomenico, P. M., & Begon, M, 2010, 3)

Ferguson, N. M., Mallett, S., Jackson, H., Roberts, N., & Ward, P. (2003). A population-dynamic model for evaluating the potential spread of drug-resistant influenza virus infections during community-based use of antivirals. Journal of Antimicrobial Chemotherapy, 51(4), 977-990.

Abstract

The researchers have researched a mathematical model of influenza transmission dynamics. Their model covers a large range of transmission scenarios such as population age structure, seasonal transmission, immunity, and inclusion of elderly nursing home residents or non-residents. The model attempts to predict the emergence of drug resistance in viral strains. The model provided by the researchers will help this researcher's development of a viral infection model. The emergence of drug resistance is important in the accurate modeling of a viral infection. In the real world, drug resistance plays a key factor in modeling an accurate spread of the infection, especially when it comes to developing countries that have ready access to medicine.

Key Points

1. "The model (see Figure 1 for overall structure and the Appendix for details and parameter values) divides the population into five compartments as a function of disease status: susceptible, exposed but not infected, asymptomatic infected, symptomatic infected and immune." (Ferguson, N. M., Mallett, S., Jackson, H., Roberts, N., & Ward, P., 2003)

2. “The model adopted here, unlike earlier models,12 explicitly stratifies the population by age, since we wished to explore how treatment in an age-restricted patient group (e.g. elderly persons) might impact on potential development of viral resistance in the population.” (Ferguson, N. M., Mallett, S., Jackson, H., Roberts, N., & Ward, P., 2003)

3. “Recovery from influenza infection results in the acquisition of immunity, resulting in partial protection from re-infection with similar strains of the virus (we assume that NAI treatment of an infected individual has no effect on this process)” (Ferguson, N. M., Mallett, S., Jackson, H., Roberts, N., & Ward, P., 2003)
4. “Vaccination is also modelled, and when effective is assumed to have the same immune protective effect as infection in the model.” (Ferguson, N. M., Mallett, S., Jackson, H., Roberts, N., & Ward, P., 2003)

Innocent, G., Morrison, I., Brownlie, J., & Gettinby, G. (1997). A computer simulation of the transmission dynamics and the effects of duration of immunity and survival of persistently infected animals on the spread of bovine viral diarrhoea virus in dairy cattle. *Epidemiology & Infection*, *119*(1), 91-100.

Abstract

The researchers have explored a computer model that mimics the spread of bovine diarrhea virus throughout a herd. The computer model is able to model the birth of new calves along with determining if those calves are susceptible to the virus or not. The research presented in this article will influence the design of this researcher's computer model for infection. The spread of infection through offspring is essential in the accurate modeling of viruses. Especially because of the possibility for temporary immunization of the offspring.

Key Points

1. **"Fig. 1.** A network representation of the flow of animals between susceptible, immune, temporarily immune and PI groups. Narrow lines represent the movement of animals from one group to another, heavy lines the birth of calves."(Innocent, G., Morrison, I., Brownlie, J., & Gettinby, G., 1997, 2)
2. "Infection of cows during the first trimester, before the foetus has acquired immune competence, results in the birth of calves that are persistently infected (PI) with the virus and immunologically tolerant to it. These animals remain asymptomatic lifelong carriers of the virus, unless mucosal

disease intervenes.”(Innocent, G., Morrison, I., Brownlie, J., & Gettinby, G., 1997, 2)

3. “The mode of transmission assumes that susceptibles become infected from persistently infected animals and that the probability of infection increases as the number of persistently infected animals increases. The infection probability takes the form (1k(1kε)Π) where ε is the transmission rate between animals and Π is the number of persistently infected animals”(Innocent, G., Morrison, I., Brownlie, J., & Gettinby, G., 1997, 2)
4. “The simulation instructions have been implemented using the C++ object-orientated programming language. The model operates in steps of 1 month, each month being 28 days. Numbers of animals in each of the ten classes defined can be estimated over the management period of interest. In particular, the model can be used to plot changes in the numbers of susceptible, immune and persistently infected calves and adults”(Innocent, G., Morrison, I., Brownlie, J., & Gettinby, G., 1997, 2)

Greengard, L., & Rokhlin, V. (1997). A fast algorithm for particle simulations. *Journal of computational physics*, *135*(2), 280-292.

Abstract

The research presented by Leslie Greengard focuses on the creation of fast simulations. Specifically, Greengard creates an algorithm for the evaluation of potential and force fields in a system with a large amount of particles. The algorithm attempts to create a way to efficiently and quickly calculate the physics simulations between a large number of particles. This is not directly related to this researcher's research, however the concepts used in speeding up a complex algorithm are. These concepts will help build a ground level for creating an efficient algorithm for the spread of a virus. If the steps taken in this paper are not applied, the resulting viral simulation may prove too slow for any useful data to be gathered. For that reason, the clustering algorithm presented will be extremely useful at modeling this researcher's viral simulation.

Key Points

1. "The central strategy used is that of clustering particles at various spatial lengths and computing interactions with other clusters which are sufficiently far away by means of multipole expansions. Interac- tions with particles which are nearby are handled directly"(Greengard, L., & Rokhlin, V, 1997, 6)
2. **"FIG. 3.** The computational box (shaded) and its nearest periodic images. The box is centered at the origin ‘‘0’’ and has area one."(Greengard, L., & Rokhlin, V, 1997, 6)

3. **“FIG. 4.** The computational box and three levels of refinement.”(Greengard, L., & Rokhlin, V, 1997, 6)
4. “The following is a formal description of the algorithm…”(Greengard, L., & Rokhlin, V, 1997, 7)

Thornton, K. R. (2014). A C++ template library for efficient forward-time population genetic simulation of large populations. *Genetics*, *198*(1), 157-166.

Abstract

The research done by Keven Thornton led to the creation of the fwdpp c++ library. The purpose of this library is to create help in the creation of an efficient implementation of forward-time population genetic simulations. Fwdpp supports two types of sampling algorithms, the first is a fitness based genetic simulation, the second isa gamete based algorithm. The fitness based genetic simulation uses a method of passing down recombinations of parental genomes. The gamete based algorithm generates the next generations based on a formula for the expected frequency of each gamete in the next generations. However this second method is much less efficient and only present for backwards compatibility. This library will help in the creation of the genetic algorithm used in evolving viruses as they spread throughout the population. This is an important step in simulating the spread of viruses because virus spread is heavily influenced by the mutations that occur as it travels between hosts. Simulating the change in a virus as it moves from one host to another is important in accurately predicting how a virus will travel across the world.

Key Points

1. “The intent of the library is to provide generic routines for mutation, recombination, migration, and sampling of gametes proportionally to their fitnesses in a finite population of *N* diploids.”(Thornton, K. R. 2014)

2. “The library does this in a memory-efficient manner by defining a small number of simple data types. First, there are mutations. The simplest mutation type is represented by a position and an integer representing its count in the population ($0 \leq n \leq 2N$). Second, there are gametes, which are containers of pointers to mutations.”(Thornton, K. R. 2014)

3. "This pointer-based structure is perhaps obvious, but it has several advantages. First, it replaces copying of data with copying of pointers, which is both faster and much more memory efficient”(Thornton, K. R. 2014)
4. “The library is compatible with another widely used C++ library for population genetic analysis [libsequence (Thornton 2003)] and contains functions for generating output compatible with existing programs based on libsequence for calculating summary statistics.”(Thornton, K. R. 2014)
5. “Descendants are generated by sampling parents proportionally to their fitnesses, followed by mutating and recombining the parental gametes”(Thornton, K. R. 2014)

Question

Is it possible for a computer to accurately simulate the spread of a biological virus across the world?

Hypothesis

Predicting the spread of a biological virus can be used to contain and slow the spread of deadly viruses in a population. Predictions can be done either manually by disease experts, or automatically by a computer. Computers have shown to be effective at computing complex processes based in other sectors of the world such as physical simulations, aeronautics, and aviation. Because of this, using a virus to predict the spread of a disease is a perfect task for a computer. It is extremely data driven and can be broken down into complex, yet manageable, equations for a computer to handle. To do this accurately a data-driven model of the earth will be needed. This data-driven model will be based on population data provided by the United Nations in their 2018 Revision of World Urbanization Prospects. There are many complex steps in the transmission of a virus, it first starts with a host and a recipient. The conditions needed for a host to transmit a virus to a recipient are based on both the host and the recipient's health conditions. If the recipient has low health conditions and is unable to fight off disease, they will become infected at a much higher rate than someone with normal health status. If the host is showing symptoms such as coughing or a runny nose, their rate of transfer may be higher. Alongside these conditions the infection rate is also affected by the following recipient and host conditions: population density, diet, living conditions, daily activities, travel, health. The characteristics of the virus also affect the

infection rate of the virus. Taking into account all of the factors needed to correctly determine viral infection rates and how viruses spread across a population, the researcher does not believe it will be possible for a computer to completely simulate the spread of a virus. The researcher believes this because the scale of simulating something the size of the world is going to be much too large for any modern computer to handle, along with the need for so many different variables to be simulated. These issues are much too large for any modern computer to handle, let alone have every interaction correctly modeled.

Procedure

The following procedure will be a surface level overview of the steps taken to create the simulation, and analyze data. If the reader wishes to replicate the procedure, it is recommended they follow the commented code on https://github.com/ActuallyZach/ViralSpread

1. Set up a preferred integrated development environment for c++. The researcher recommends using CLion. This environment must be capable of outputting files to an accessible environment.
2. Create Enums for Disease Stage
    a. Disease Stage
        i. Susceptible
        ii. Latent
        iii. Infectious

iv. Recovered

v. Immune

3. Create structures for a Virus, Person, City Mile / Chunk, City, and World.
4. Include the following parameters in these structures
    a. Virus
        i. Id — UUID or Int
        ii. Name — String
        iii. Infection Methods — Array
        iv. Infection Rate — Double
        v. Infection Period — Int
        vi. Recovery Period — Int
    b. Person
        i. Id — UUID or Int
        ii. X — Int
        iii. Y — Int
        iv. Parent Chunk Index — Int
        v. Trip Starting Chunk Index — Int
        vi. Trip Length — Int
        vii. Round Trip Counter — Int
        viii. Is On Round Trip — Bool
        ix. Trips Today — Int
        x. Trips Counter — Int
        xi. Virus — Optional Virus

xii. Personal Latency Period Int

xiii. Personal Infection Period Int

xiv. Personal Recovery Period Int

xv. Disease Stage Disease Stage Enum

xvi. Stage Counter Int

c. City Mile

i. Id UUID or Int

ii. X Position Int

iii. Y Position Int

iv. Neighbor Indices Integer Array

v. People Person Reference Array

d. City

i. Name String

ii. Population Int

iii. Population Density Double

iv. Square Mileage Double

v. City Chunks City Mile Array

vi. People Person Array

e. World

i. Cities City Array

ii. Viruses Virus Array

5. Add and implement the following functions for the following classes, sample implementations can be found on the GitHub or in the code section of this paper.

a. Person
    i. Advance Infection Void
    ii. CanBeInfected Bool
    iii. Infect Bool
    iv. Is Infected Bool
b. City Mile
    i. Set Possible Indices Void
    ii. Random Neighbor Index Int
c. World
    i. Infect Void
    ii. Simulate Void

6. Implement a system of saving the following data points to a CSV file on every iteration of the simulation in each chunk
    a. Day Number
    b. City Name
    c. City Population
    d. Chunk ID
    e. Chunk Population
    f. Number Susceptible
    g. Number Latent
    h. Number Infectious
    i. Number REcovered
    j. Number Immune

    k. Average Chunk Health

7. Once running a simulation of your desired size, import the exported CSV file into your preferred spreadsheet program and analyze the data for infection curves.

Summary of Trials

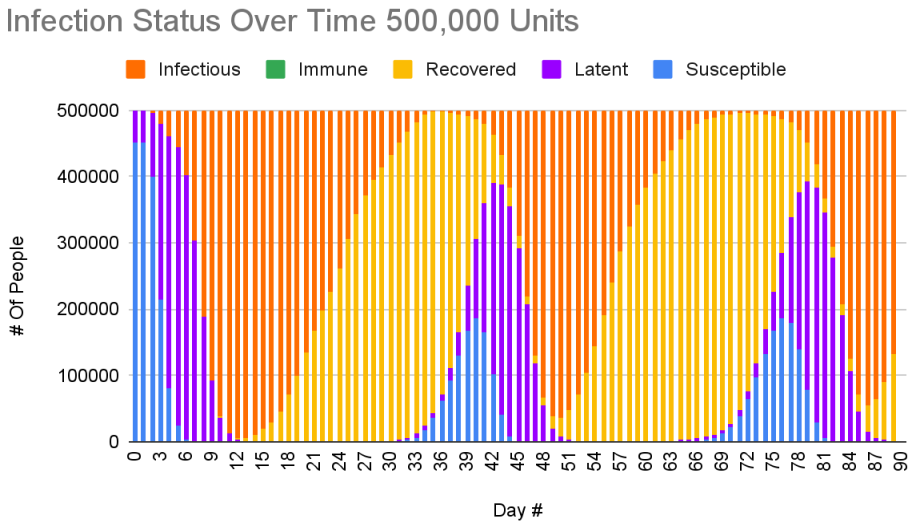
The following results were collected from a series of simulations all with different parameters. The biggest and most prominent simulation was completed over 18 hours with a population size of 500,000 units, 100 chunks, and a starting chunk density of 5,000. Other simulations included; a second 500,000 unit simulation, a 100,000 unit simulation, a 1,000 unit simulation and a 100 unit simulation. Prior to the completion of those simulations, there is a population size 100 simulation with health degradation removed.

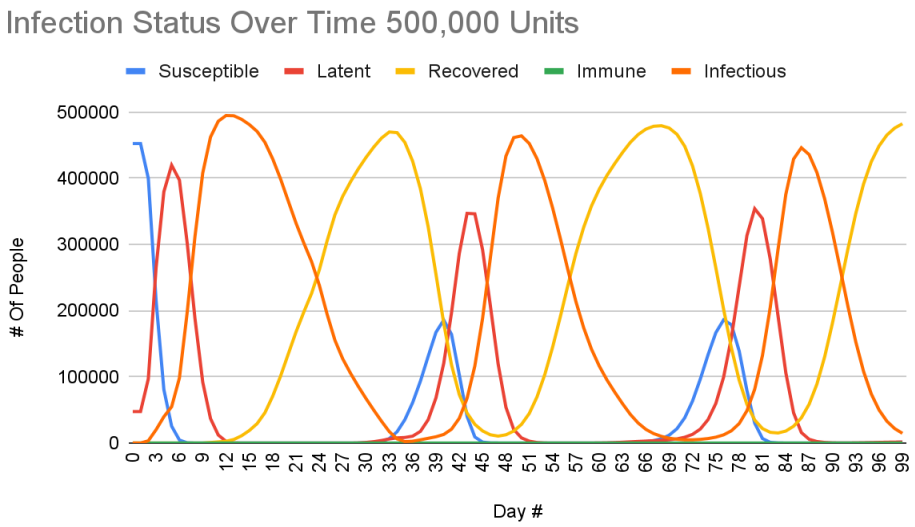
Simulations were performed on two computers. Smaller, quicker simulations (100 - 1,000) were performed on a mac laptop with an Intel i9-9880H processor. These simulations were compiled using the Clang C++ compiler. Larger simulations (10,000 - 500,000) were run on an Intel i7-7700k processor. These simulations were compiled using the Visual C++ compiler. Differences in how the program performed between the two compilers were not noticed. The only difference between the two computers was the speed of the simulation. The i7-7700k was able to perform significantly better, resulting in faster simulations. This is because it was not thermally limited as it was in a desktop computer instead of a laptop.

Trial 1 (500,000 Units, 100 square miles)

The biggest and most data rich trial was the 500,000 unit, 100 square mile trial. This trial was the biggest trial run by the researcher. It took over 18 hours to complete the total simulation, producing 110,000 cells of data.

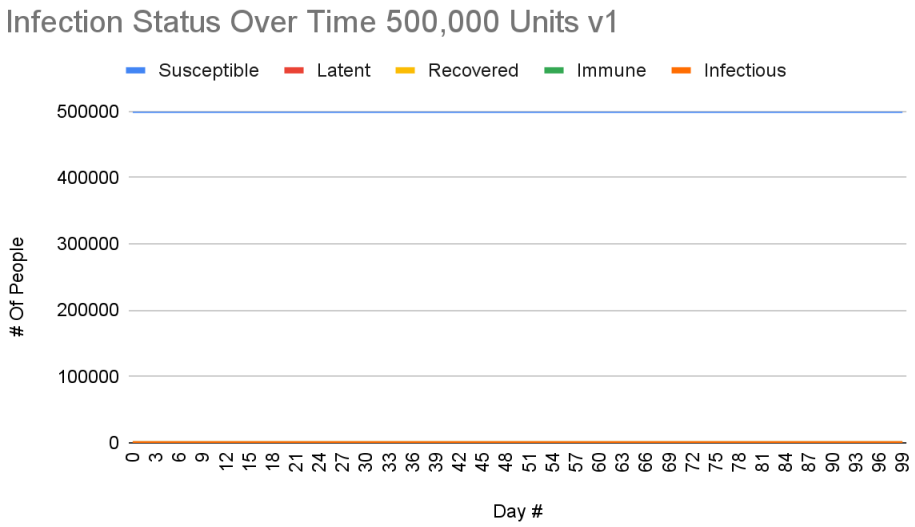
This trial shows the wave of infections that is to be expected from a viral infection. The waves in this data set are extremely pronounced, if not over pronounced. The researcher believes this is a result of the high infection rate and lack of complexity in the spread vectors.

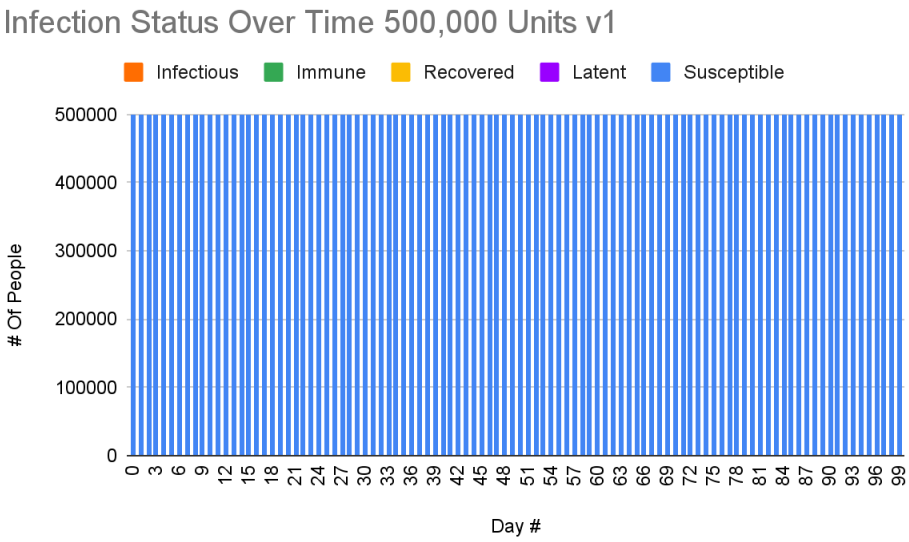



Infection Status Over Time 500,000 Units
Susceptible
Latent
Recovered
Immune
Infectious
Of People
500000
400000
300000
200000
100000
0
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
Day #

Trial 2 (500,000 Units, 100 Square Miles)

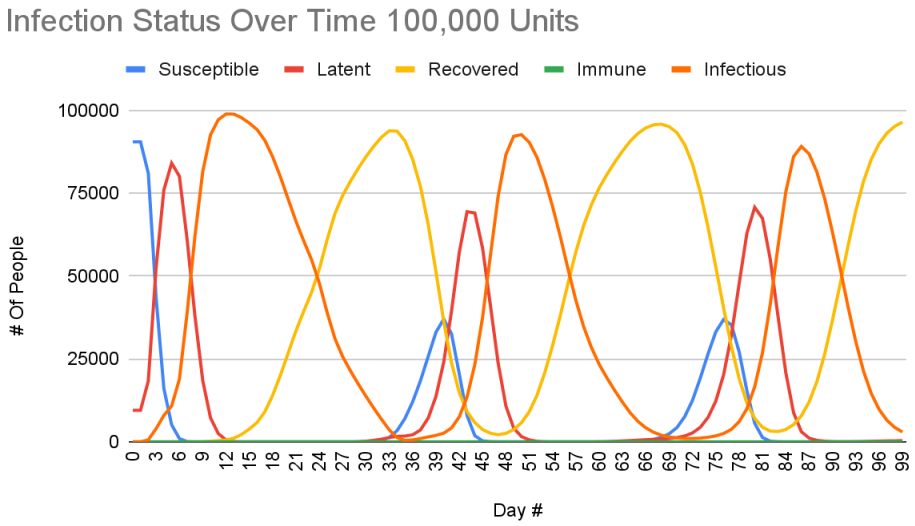
This trial took approximately 10 hours to complete, it had the same parameters as the one above, however it was distinctly different. In this trial, 1 person was infected compared to the 10% of the population normally infected. This singular unit failed to pass on the infection. In a population density of 5,000 people per square mile, the unit failed to interact with any other units and infect them. This is an outlier in all of the other tests performed. Most tests before the final round were tested with only 1 person in a population, these trials always resulted in the infection spreading. However, this one trial lacked any spread in infection.

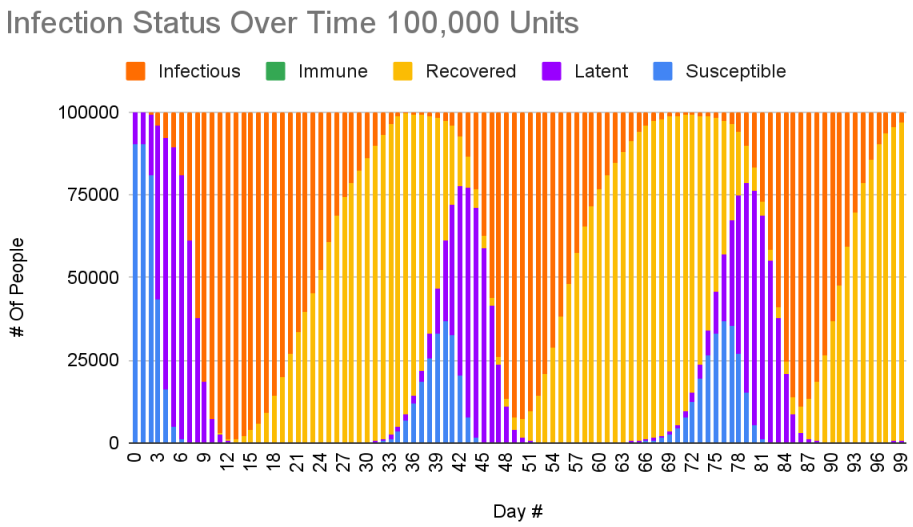



Infection Status Over Time 500,000 Units v1
Infectious
Immune
Recovered
Latent
Susceptible
500000
400000
300000
200000
100000
0
Of People
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99
Day #

Trial 3 (100,000 Units, 100 Square Miles)

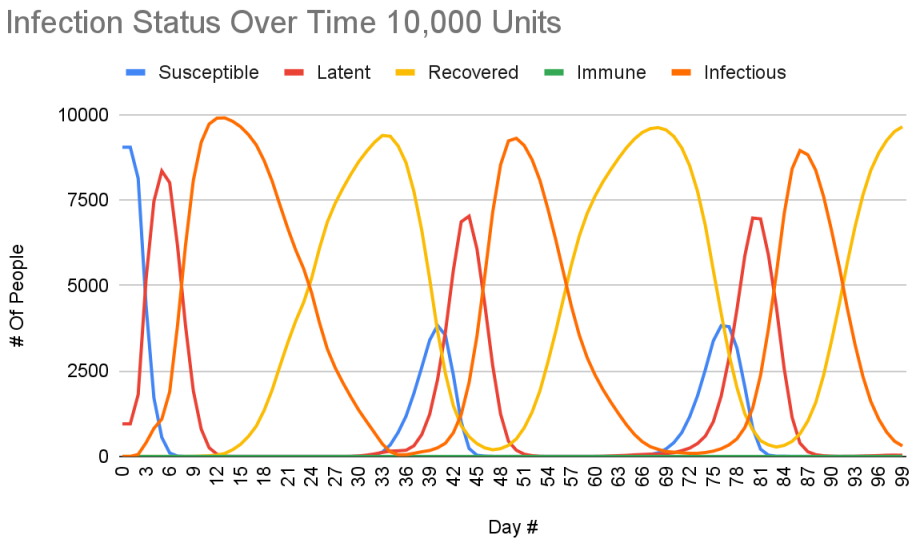
The data received from trial number three is eerily similar to trial one. The infection curves are nearly the same, with only some variation in the peaks. This is interesting because the population density of a single mile was 5 times less than the 500,000 trial, which led the researcher to believe that there would be a significant change in how the infection was spread. The biggest difference between this trial and trials 1 and 2 is that it was significantly quicker. This trial took approximately 2.49 hours to complete.

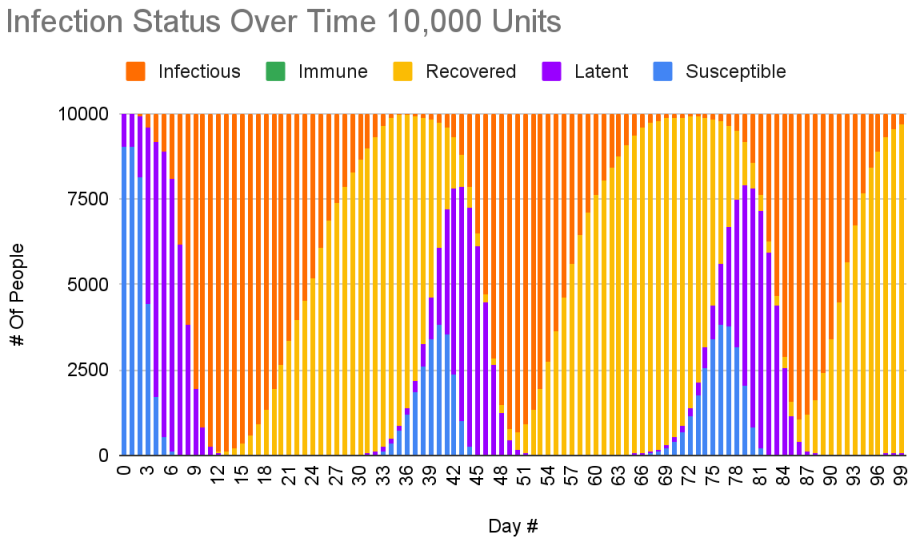



Infection Status Over Time 100,000 Units
Infectious
Immune
Recovered
Latent
Susceptible
100000
75000
50000
25000
0
Of People
0
3
6
9
12
15
18
21
24
27
30
33
36
39
42
45
48
51
54
57
60
63
66
69
72
75
78
81
84
87
90
93
96
99
Day #

Trial 4 (10,000 Units, 100 Square Miles)

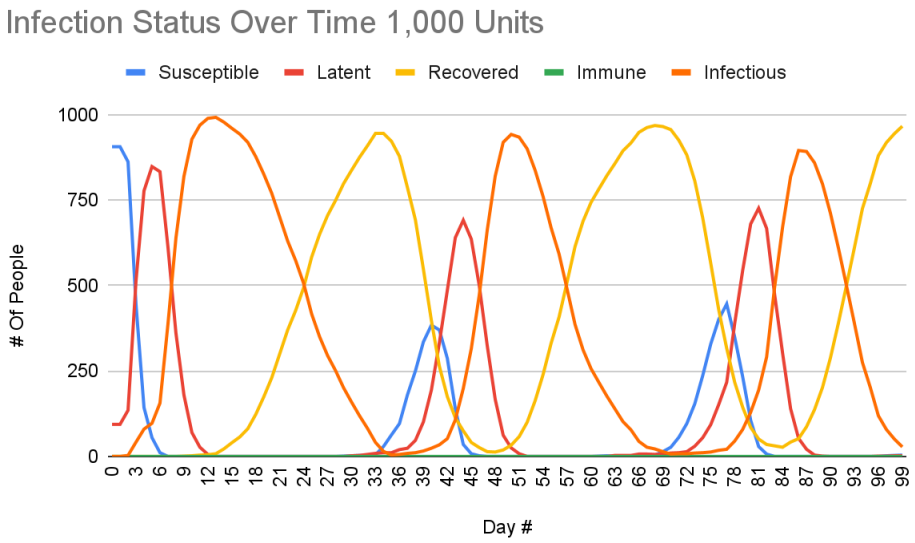
Infection waves are still present in this population size. The waves are consistent with the other population sizes, nearly identical. This simulation took approximately 17 minutes to complete.

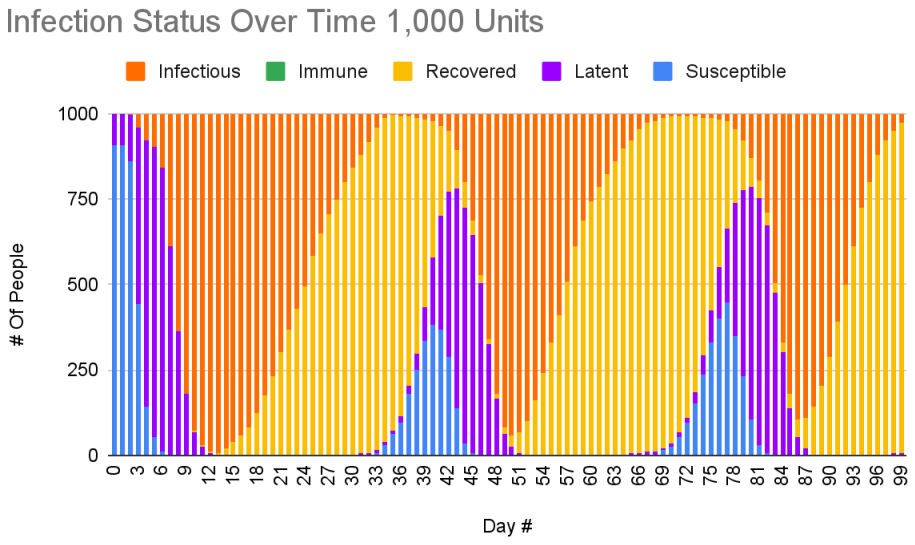




Trial 5 (1,000 Units, 10 Square Miles)

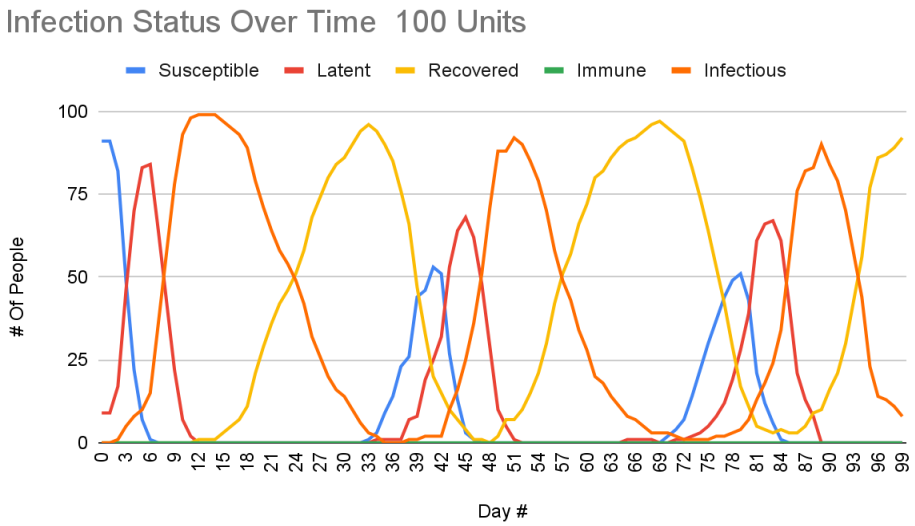
Compared to other trials with a larger population, infection curves are much sharper. The waves of infection are still present, the same as the other, larger populations. This trial took approximately 1 minute to complete.

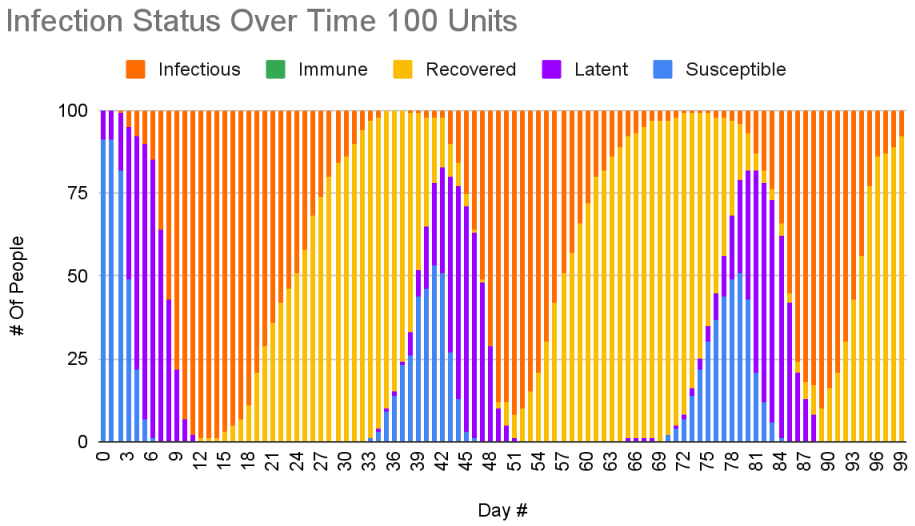




Trip 6 (100 Units, 10 Square Miles)

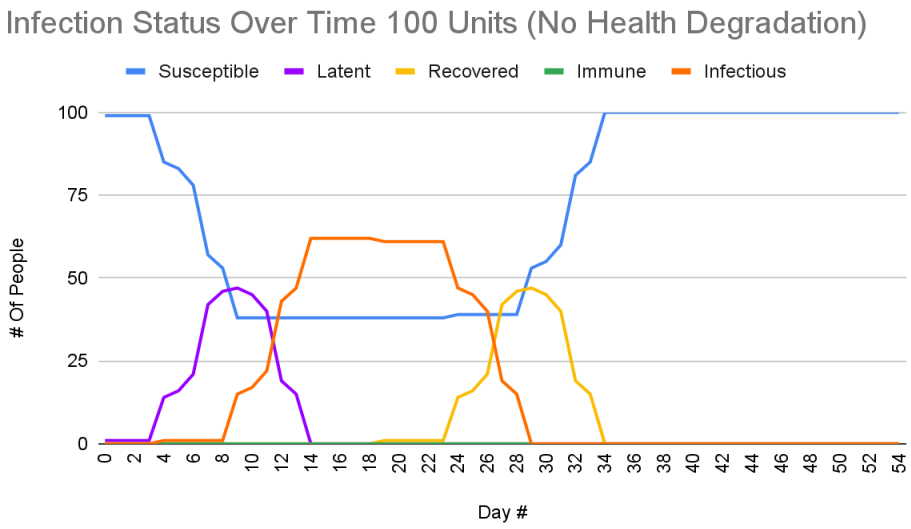
100 units is an extremely small population, despite that, infection curves are still present and very close to the other larger population trials. Like trial 5 the curves are extremely sharp with a lot of sharp increases and decreases. This was the fastest simulation, at an impressive 13 seconds to fully simulate.

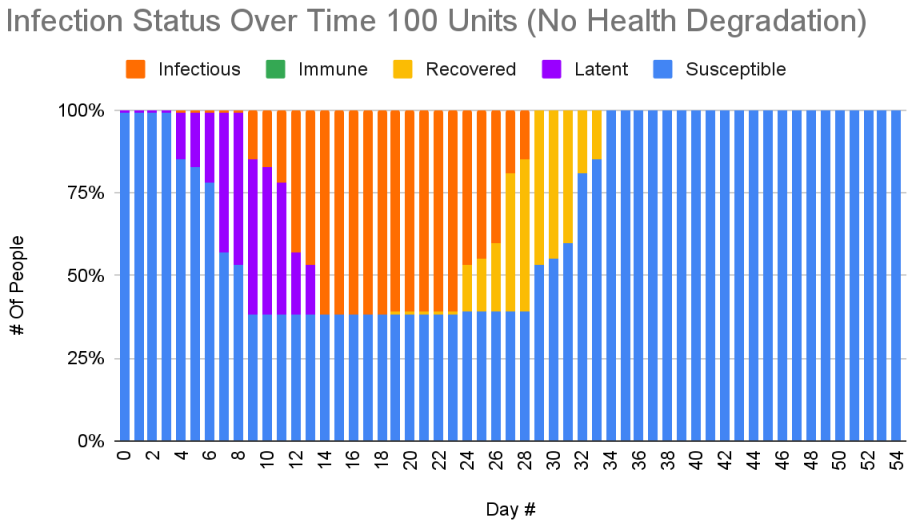




Trial 7 (100 units, 10 Square Miles, No Health Degradation)

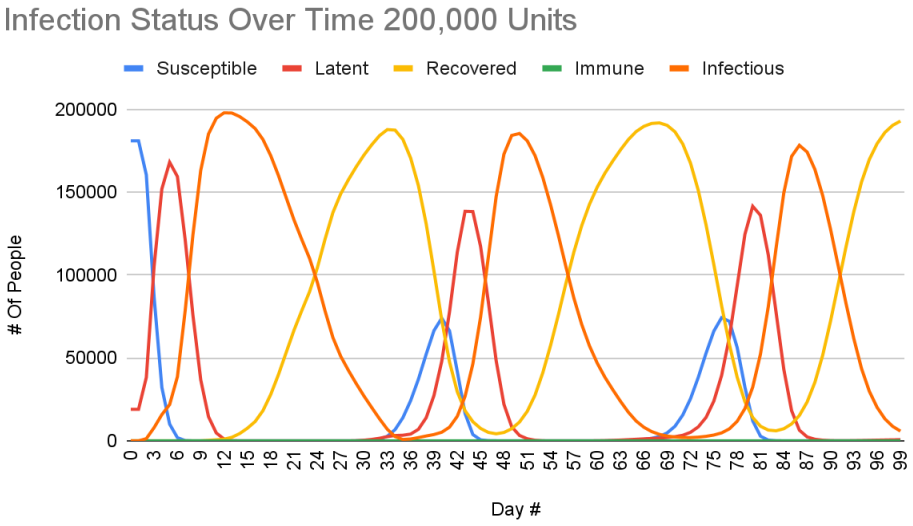
Trials run without the health degradation show an interesting, symmetrical pattern. Health degradation is a part of the simulation where a unit's health will decrease or increase based on their stage of infection. A lower health status makes a unit hold onto a virus longer, show symptoms quicker, and recover slower. A higher health status is the reverse of low health status. Through running simulations without health degradation the researcher was able to see how much the reduction in a unit's health affects how they react to a disease. These are the only trials where waves are not present, this can be attributed to every unit who catches the disease recovering from it at the same time. Small waves are present, represented by the sudden jumps in the graphs, however these are just new groups of people entering the different categories.

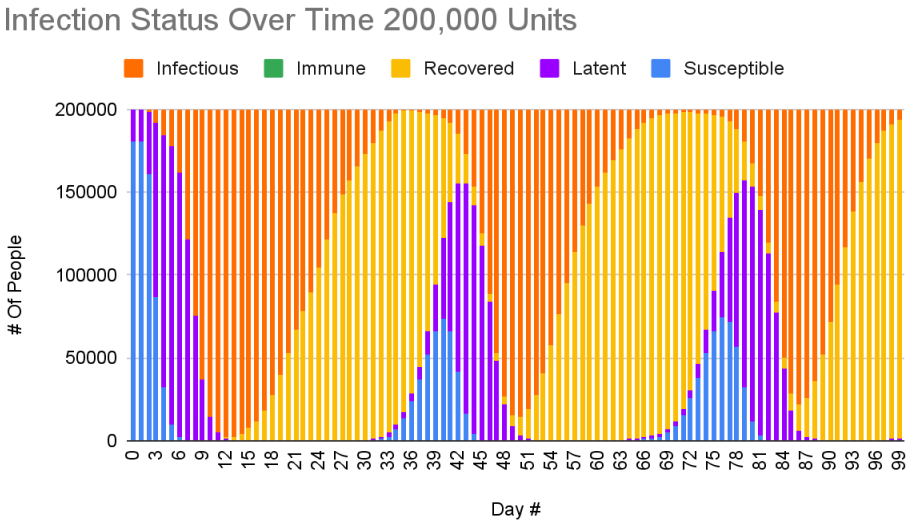



Infection Status Over Time 100 Units (No Health Degradation)
Infectious
Immune
Recovered
Latent
Susceptible
100%
75%
50%
25%
0%
Of People
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
Day #

Trial 8 (200,000 Units, 100 Square Miles)

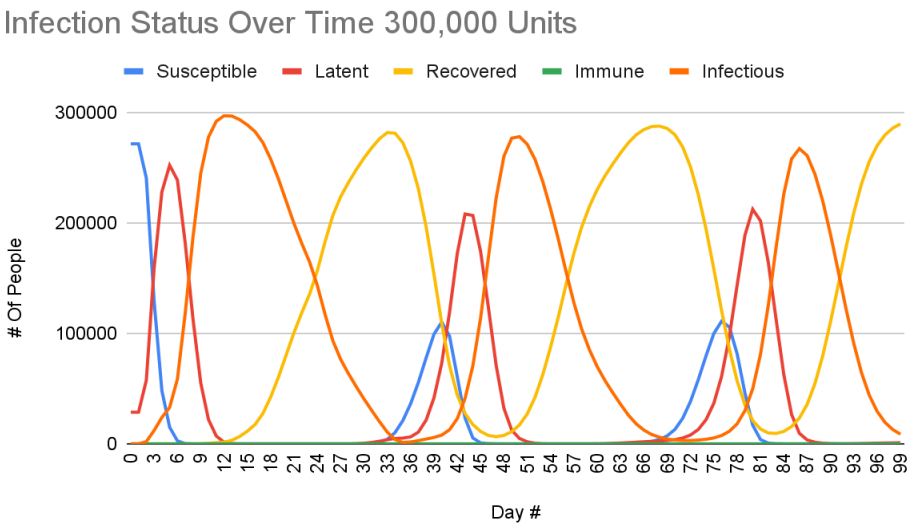
This trial was run to confirm the researchers' hypothesis that an increase in population will lead to a linear increase in simulation time. This trial took approximately 5.65 hours, next to the 100,000 unit trial which took 2.49 hours. The data follows the same trend as the rest of the trials, the data is more refined at the tips, but it is still jagged. This supports the hypothesis that as population increases the accuracy increases.

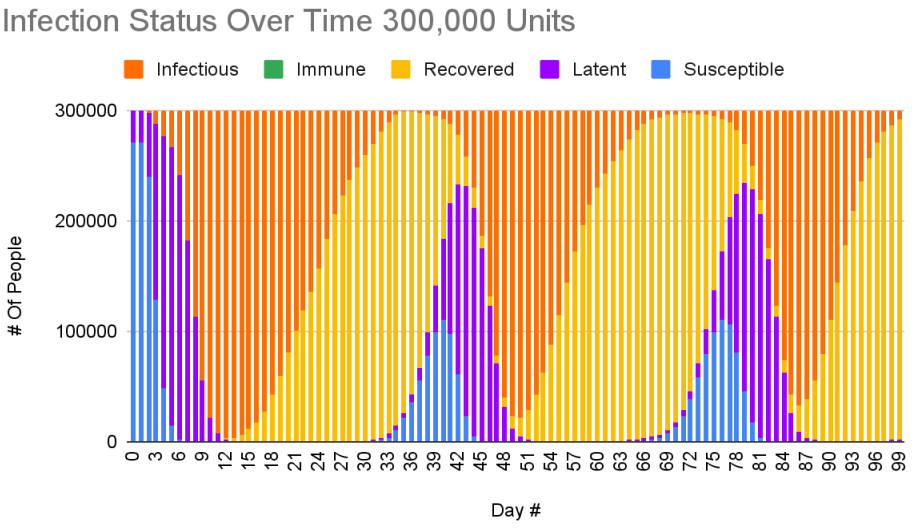




Trial 9 (300,000 Units, 100 Square Miles)

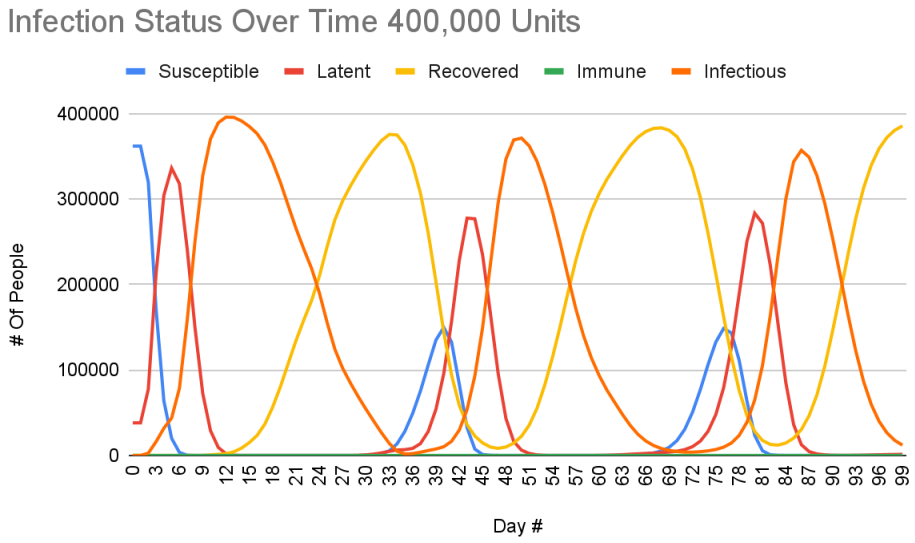
The following trial was run for the same reason as trial 8. This trail took approximately 8.51 hours to complete. The data from this trial is nearly identical to the data from the 200,000 trial. There are only very small differences in the infection curves.

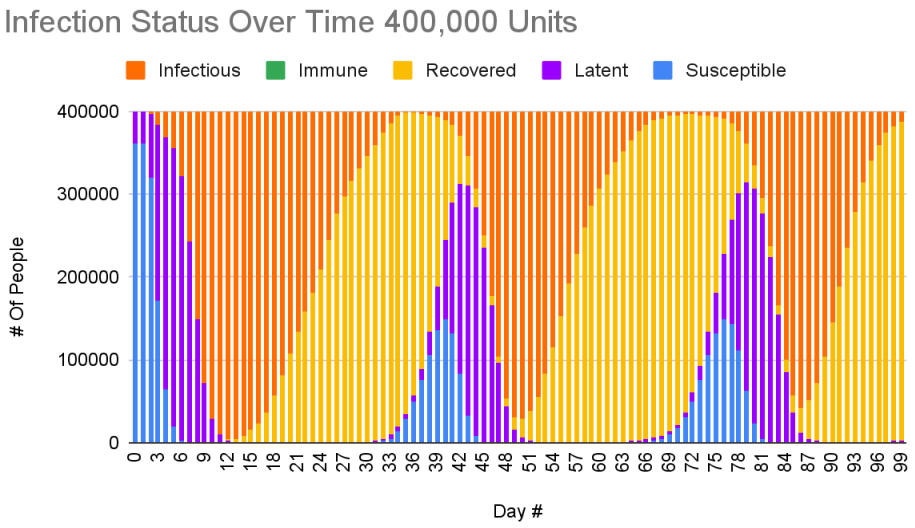




Trial 10 (400,000 Units, 100 Square Miles)

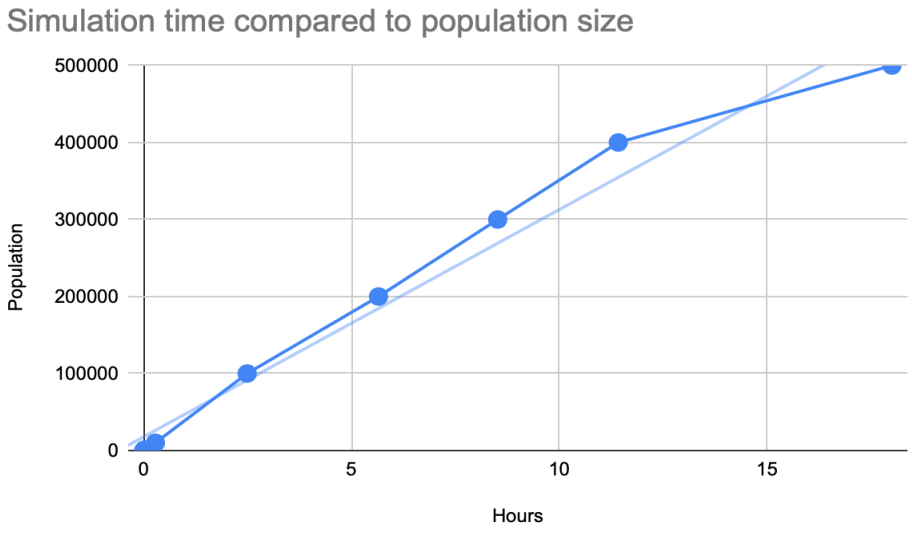
This trial is the final trial run to inspect the linearity in time compared to a population increase. These trails show almost exactly the same data as the other trials, with not much variance in the data. The 400,000 unit simulation took 11.41 hours to complete.





Simulation Runtime

The graph below shows how long in hours a simulation took. The graph is roughly linear with simulation length growing at 3.3 hours for every 100,000 units. The only divergences from almost perfect linearity are when population size grows above 400,000 and below 100,000. There is currently not enough information to find what the cause of this increase is. The researcher believes that these trials are purely outliers and are a result of random number generation that favored harder to calculate numbers. The linearity of the middle 4 trials is very convincing for a linear increase in the simulation time with increasing population. The researcher believes if trials are rerun for the 500,000 unit population, the jump in time will not be present.



Conclusions based on the gathered data

The goal of a research paper is to answer a hypothesis. This paper achieved this goal by answering whether or not it is possible for a computer simulation to accurately simulate a viral infection across the Earth. he researcher believed that it would not be a feasible task, and that the hardware of modern computers is not powerful enough. The data collected shows this perfectly. A rudimentary simulation, capable of spreading a basic virus across a small population of 500,000 took 18 hours to complete. A population of 500,000 units is only 0.0062912588 percent of Earth's current population (7,947,535,100 at the time of writing). Based on data from the trials we are able to assume that the time increase in populations is a linear relationship. With this knowledge, a simulation the size of Earth would take approximately 32.66 years (286,111.26 hours, or 11,921.3 days). This time frame is astronomically higher than any reasonable simulation would need to be. Based on this, the researcher is able to confirm their hypothesis that this kind of viral simulation is not currently a task that a computer is capable of.

Even though the researcher has concluded that a computer simulation of a virus is not possible, that does not mean the work done in this paper can not be used in future simulations. The data collected from this paper shows a clear need for the advancement in simulation technology and the way viral simulations are modeled. The following section will explore the results of the simulation trials, along with what can be done in future models to correct for the issues present. The section will end with a theory for the expansion and continuation of this paper.

The data gathered, however not completely accurate to the spread of any one disease, shows the clear signs of an appropriate spread model. Throughout every trail, a wave of infected units is present. This wave is similar to the waves that are seen in real world viral spread. The major difference between the computer simulation and the real world is the rate at which these waves happen and the severity of them. The computer simulation shows frequent and very large waves. Real world data, such as the Covid-19 virus, show more infrequent waves with a lower number of people being infected in these waves.

Based on the gathered data the researcher is able to conclude that larger populations are not necessarily needed for an accurate prediction of viral spread. All of the trials completed by the researcher that included health degradation (trials 1 - 6), have nearly the same output. The only difference in the data output from the different trials is how smooth the graphs appear. The larger the population, the smooth the resulting data appears. Lower population sizes lead to jagged and rough graphs. One place where the difference between population size and trials was significant and noticeable is in the simulation time. Larger simulations take much longer than smaller simulations. This increase in time means that a sweet spot for simulation time and accuracy is needed. Based on current data, a simulation size of 10,000 is the most optimal. This simulation provides fine grained data alongside the fact of it being much faster than other simulations. A trial of this size should be accurate enough to create meaningful data while not taking up unnecessary time.

Even though there was little to no variability between trials, there is a large visible difference in data output when new complexities are introduced to the spread of a virus. Trial 7 was completed with health degradation removed. Compared to trials 1-6 this trial stands out. It shows a nearly symmetrical spread of the virus, with the different categories being nearly identical in when they increase and then subsequently decrease. Based on these findings, the researcher is able to conclude that the factors that affect the accuracy of a viral simulation, is not the population size, but the complexity of the simulation. Adding more features will greatly increase the effectiveness of the simulation. These features could be: more viral spread vectors, group outings, friends and close family relationships, work, long distance travel, and behavioral changes when sick. Adding any of these features should greatly affect how accurate a viral simulation is.

Future Steps

After all of these considerations and conclusions the researcher has put together a plan for a future simulation that will be able to mitigate all of the issues brought forth in this paper, along with increasing the accuracy and speed of a computer simulation in this field.

The first step in fixing the simulation will be introducing different spread vectors of a disease. This was originally planned to be included in this research paper by the researcher, however time constraints led to the necessary removal of the feature. Different spread vectors would allow the simulation to correctly model all the nuanced ways that a disease can spread within a population. However, adding new spread

vectors is not the only necessary addition. An expansion on the behavioral patterns of each unit are also needed. This behavioral expansion would consist of increasing what a unit is capable of doing, such as long distance travel, friend groups, and family units. Disease often spreads throughout both friend groups and family groups, and missing these connections for every unit, has led to an inaccurate simulation. Another necessary step towards creating a functional viral spread simulation, is in the individual travel of a unit. Travel is important in the spread of a virus. As people travel they bring the virus with them. In this simulation, people are only able to move within their community. Introducing more sub-populations (cities, towns, etc…) into the overall world of the simulation would greatly affect the way we see spread occur.

The researcher believes that if these steps are taken, and a scaled down population size relative to the earth, it would lead to an increase in accurate results and timely results from the simulation.

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <utility>
#include <vector>
#include <map>
#include <random>
#include <cmath>
#include <algorithm>
#include <execution>
#include <optional>

/*
 An Implementation of a variation of the SIERS model for infectious disease.
 This simulation attempts to model the spread, infection, and evolution of viral diseases across a population (The World)
 */

using namespace std;
int rollingPersonID { 0 };
int rollingVirusID { 0 };

/*
 * World Constants
 */
// The number of daily social interactions. Based on resarch from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6113687/
const int numberOfInteractionsPerTrip = 3;
// The average number of times somebody travels. Data calculated on findings. https://www.bts.gov/archive/publications/highlights_of_the_2001_national_household_travel_survey/section_02
// & https://www.researchgate.net/publication/279853330_Extended_Range_Electric_Vehicle_Driving_and_Charging_Behavior_Observed_Early_in_the_EV_Project
// Trips are defined as moving from one address to another. This means, they usually come in multiples of 2.
const int numberOfDailyTrips = 4;
const int averageTripDistanceMiles = 8;
const double roundTripChance = 0.6;

// Values representing percentages across the simulation
const double healthLevelOne = 0.1;
const double healthLevelTwo = 0.4;
```

```
37 const double healthLevelThree = 0.5;
38 const double healthLevelFour = 0.8;
39 const double healthLevelFive = 1.0;
40 const double latentHealthDecrease = -0.005;
41 const double infectiousHealthDecrease = -0.05;
42 const double recoveringHealthIncrease = 0.045;
43 const double startingInfectionPercentage = 0.1;
44
45 bool halt = false;
46
47 // https://en.cppreference.com/w/cpp/numeric/random/
   discrete_distribution
48 // These functions will randomly select a random numbers.
49 double randomDouble(double min, double max) {
50     std::random_device rd;
51     std::mt19937 mt(rd());
52     std::uniform_real_distribution<double> dist(min, max);
53     return dist(mt);
54 }
55
56 int randomInt(int min, int max) {
57     std::random_device rd;
58     std::mt19937 mt(rd());
59     std::uniform_int_distribution<> distr(min, max);
60     return distr(mt);
61 }
62
63 int randomWeightedInt(initializer_list<double> weights,
   initializer_list<int> numbers) {
64     random_device rd;
65     mt19937 gen(rd());
66     discrete_distribution<> d(weights);
67     int index = d(gen);
68     return numbers.begin()[index];
69 }
70
71 double randomWeightedDouble(initializer_list<double>
   weights,  initializer_list<double> numbers) {
72     random_device rd;
73     mt19937 gen(rd());
74     discrete_distribution<> d(weights);
75     int index = d(gen);
76     return numbers.begin()[index];
77 }
78 template <typename T>
79 int indexOf(T object, vector<T> vector) {
80     for(int n = 0; n < vector.capacity(); n ++) {
81         if(vector[n] == object) {
```

```
 82                 return n;
 83             }
 84         }
 85         return -1;
 86 }
 87
 88 // https://stackoverflow.com/a/9345144/14886210
 89 template<class BidiIter>
 90 BidiIter random_unique(BidiIter begin, BidiIter end,
    size_t num_random) {
 91     size_t left = distance(begin, end);
 92     while (num_random--) {
 93         BidiIter r = begin;
 94         advance(r, rand()%left);
 95         swap(*begin, *r);
 96         ++begin;
 97         --left;
 98     }
 99     return begin;
100 }
101
102 // Unused in the final simulation. Recommended to
    implement in future versions
103 enum VirusType {
104     Lysogenic, // Long-lasting, can stay dormant for a
    while
105     Lytic // Short, usually takes effect right away
106 };
107
108 // Unused in the final simulation. Recommended to
    implement in future versions
109 enum InfectionMethod {
110     Touch,
111     Saliva,
112     Coughing,
113     Sneezing,
114     SexualContact,
115     Contamination,
116     Insects
117 };
118
119 enum DiseaseStage {
120     Susceptible, // Can become infected
121     Latent, // Infected, but unable to infect others
122     Infectious, // Able to infect others
123     Recovered, // Immune to the disease
124     Immune // A temporary immunity, comes from birth,
    immunity decreases after birth
```

```
125 };
126
127 // https://www.microcovid.org/
128 struct Virus {
129     string name;
130     int id;
131     vector<InfectionMethod> infectionMethods { };
132
133     double infectionRate { 0.14 }; // A percentage out of
    100
134     int latencyPeriod { 3 }; // The period at which it
    takes to cross from infected -> infectious
135     int infectionPeriod { 14 }; // The length of time that
     a person is infected for
136     int recoveryPeriod { 4 }; // The length of the
    recovery period
137
138     Virus(string name, vector<InfectionMethod>
    infectionMethods) {
139         this->name = std::move(name);
140         this->id = rollingVirusID;
141         this->infectionMethods = std::move(
    infectionMethods);
142         rollingVirusID ++;
143     }
144 };
145
146 struct City;
147 struct CityMile;
148
149 struct Person {
150     int id;
151     int x { 0 };
152     int y { 0 };
153     int parentChunkIndex;
154
155     // Dealing with moving chunks
156     int tripStartChunkIndex { 0 };
157     int tripLength { 0 };
158     int roundTripCounter { 0 };
159     bool isOnRoundTrip { false };
160
161     int tripsToday { 0 };
162     int tripsCounter { 0 };
163
164     // Virus Data
165     optional<Virus> infectiousVirus;
166     int personalLatencyPeriod { 0 };
```

```
167     int personalInfectionPeriod { 0 };
168     int personalRecoveryPeriod { 0 };
169 
170     map<int, double> strainResistance { }; // Virus ID | Resistance percent 0-1
171 
172     DiseaseStage stageOfInfection { Susceptible };
173     int stageCounter { 0 };
174 
175     double currentHealthCondition { 1.0 };
176 
177     explicit Person(int id, int parentChunkIndex) {
178         this->id = id;
179         this->parentChunkIndex = parentChunkIndex;
180 
181         currentHealthCondition = randomWeightedDouble({0.1, 0.15, 0.3, 0.3, 0.1}, {healthLevelOne, healthLevelTwo, healthLevelThree, healthLevelFour, healthLevelFive});
182     }
183 
184     inline bool operator == (const Person& rhs) const {
185         return id == rhs.id;
186     }
187 
188     // Advances the infection through the stages fo the SIERS model
189     void advanceInfection() {
190         if (infectiousVirus.has_value()) {
191             if(stageOfInfection == Latent) {
192                 if(personalLatencyPeriod < stageCounter) { // We have crossed the barrier of latency
193                     stageOfInfection = Infectious;
194                     stageCounter = 0;
195                 } else { // While latent, slightly decrease health
196                     currentHealthCondition += latentHealthDecrease;
197                 }
198             } else if (stageOfInfection == Infectious) {
199                 if(personalInfectionPeriod < stageCounter) { // We have crossed the infectious period, begin recovering
200                     stageOfInfection = Recovered;
201                     stageCounter = 0;
202                 } else { // Decrease health as you are infectious.
203                     currentHealthCondition += infectiousHealthDecrease;
```

```
204                 }
205             } else if(stageOfInfection == Recovered) {
206                 if(personalRecoveryPeriod < stageCounter
    ) { // We have finished recovering, start being
    susceptible again
207                     stageOfInfection = Susceptible;
208                     stageCounter = 0;
209                     infectiousVirus.reset();
210                 } else {
211                     currentHealthCondition +=
    recoveringHealthIncrease; // We are healing, increase the
    health condition
212                 }
213             }
214             if(currentHealthCondition > 1) {
215                 currentHealthCondition = 1;
216             } else if(currentHealthCondition < 0) {
217                 currentHealthCondition = 0;
218             }
219             stageCounter ++;
220         }
221     }
222 
223     // Calculates the chance of infection. Based on health
     condition and rate of transfer of the virus. Along with
    the type of interaction.
224     bool canBeInfected(double virusInfectionRate) const {
    // Need to actually implement, 0 - 1.0
225         double infectionRate = virusInfectionRate;
226         if(currentHealthCondition <= healthLevelOne) {
227             // Worst Condition
228             infectionRate = virusInfectionRate *
    randomDouble(2.5, 3.2);
229         } else if (currentHealthCondition <=
    healthLevelTwo) {
230             // Moderate Condition
231             infectionRate = virusInfectionRate *
    randomDouble(1.8, 2.3);
232         } else if (currentHealthCondition <=
    healthLevelThree) {
233             // Normal Condition
234             infectionRate = virusInfectionRate *
    randomDouble(0.9, 1.1);
235         } else if (currentHealthCondition <=
    healthLevelFour) {
236             // Good Condition
237             infectionRate = virusInfectionRate *
    randomDouble(0.75, 0.85);
```

```
238         } else {
239             // Perfect Condition
240             infectionRate = virusInfectionRate *
    randomDouble(0.45, 0.55);
241         }
242 
243         return randomDouble(0, 1) <= infectionRate;
244     }
245 
246     bool infect(const Virus& virus, bool override = false
    ) {
247         bool canInfect = canBeInfected(virus.infectionRate
    );
248         if(stageOfInfection == Recovered ||
    stageOfInfection == Infectious || stageOfInfection ==
    Latent) {
249             canInfect = false;
250         }
251         if (canInfect || override) {
252             infectiousVirus = virus;
253             stageOfInfection = Latent;
254 
255             if(currentHealthCondition <= healthLevelOne) {
256                 // Worst Condition
257                 // Decrease latency period because person'
    s health is low, so symptoms would show quickly
258                 personalLatencyPeriod  = double(virus.
    latencyPeriod) * randomDouble(0.5, 0.9);
259                 // Increase infection period because
    person's health is low, so symptoms would last longer
260                 personalInfectionPeriod = double(virus.
    infectionPeriod) * randomDouble(1.5, 2);
261                 // Decrease the recovery period because
    person's health is low, so it will take longer to recover
    and antibodies will not last as long.
262                 personalRecoveryPeriod = double(virus.
    infectionPeriod) * randomDouble(0.5, 0.9);
263             } else if (currentHealthCondition <=
    healthLevelTwo) {
264                 // Moderate Condition
265                 // Decrease latency period because person'
    s health is low, so symptoms would show quickly
266                 personalLatencyPeriod  = double(virus.
    latencyPeriod) * randomDouble(0.6, 1);
267                 // Increase infection period because
    person's health is low so symptoms would last longer
268                 personalInfectionPeriod = double(virus.
    infectionPeriod) * randomDouble(1.4, 1.9);
```

```
269                 // Decrease the recovery period because
    person's health is low, so it will take longer to recover
    and antibodies will not last as long.
270                 personalRecoveryPeriod = double(virus.
    infectionPeriod) * randomDouble(0.6, 1);
271             } else if (currentHealthCondition <=
    healthLevelThree) {
272                 // Normal Condition
273                 // Only slight variations because we want
    this person to have fairly normal health
274                 personalLatencyPeriod  = double(virus.
    latencyPeriod) * randomDouble(0.9, 1.1);
275                 personalInfectionPeriod = double(virus.
    infectionPeriod) * randomDouble(1.1, 1.3);
276                 personalRecoveryPeriod = double(virus.
    infectionPeriod) * randomDouble(0.9, 1.1);
277             } else if (currentHealthCondition <=
    healthLevelFour) {
278                 // Good Condition
279                 // Increase latency period because health
    is better, and viruses will be fought better
280                 personalLatencyPeriod  = double(virus.
    latencyPeriod) * randomDouble(1.1, 1.4);
281                 // Decrease infection period because we
    have good health
282                 personalInfectionPeriod = double(virus.
    infectionPeriod) * randomDouble(0.6, 1);
283                 // Increase recovery period because we
    have good health and antibodies should stay for a while
284                 personalRecoveryPeriod = double(virus.
    infectionPeriod) * randomDouble(1.1, 1.4);
285             } else {
286                 // Perfect Condition
287                 // Increase latency period because health
    is better, and viruses will be fought better
288                 personalLatencyPeriod  = double(virus.
    latencyPeriod) * randomDouble(1.3, 1.6);
289                 // Decrease infection period because we
    have good health
290                 personalInfectionPeriod = double(virus.
    infectionPeriod) * randomDouble(0.3, 0.7);
291                 // Increase recovery period because we
    have good health and antibodies should stay for a while
292                 personalRecoveryPeriod = double(virus.
    infectionPeriod) * randomDouble(1.3, 1.6);
293             }
294         }
295         return canInfect;
```

```
296     }
297
298     bool isInfected() {
299         return infectiousVirus.has_value();
300     }
301 };
302
303 struct CityMile {
304     int id;
305     int xPosition;
306     int yPosition;
307     int neighborIndices [8] = { -1, -1, -1, -1, -1, -1, -1
    , -1 };
308     vector<Person*> people { };
309     vector<int> possibleIndices;
310
311     CityMile() {
312         id = 0;
313         xPosition = 0;
314         yPosition = 0;
315     }
316
317     CityMile(int id, int xPosition, int yPosition) {
318         this->id = id;
319         this->xPosition = xPosition;
320         this->yPosition = yPosition;
321     }
322
323     void setPossibleIndicies() {
324         for(int x = 0; x < 8; x++) {
325             if (neighborIndices[x] != -1) {
326                 possibleIndices.push_back(neighborIndices[
    x]);
327             }
328         }
329     }
330
331     // Picks a random index in the list of valid neighbors
    .
332     int randomNeighborIndex() {
333         if(!possibleIndices.empty()) {
334             int random = randomInt(0, possibleIndices.size
    () - 1);
335             return possibleIndices[random];
336         } else {
337             return 0;
338         }
339     }
```

```
340 };
341
342 struct City {
343     // Function to check if a number is a perfect square
    or not
344     static bool isPerfect(int number) {
345         return (sqrt(number) - floor(sqrt(number))) == 0;
346     }
347
348     // Adapted from https://www.geeksforgeeks.org/closest-
    perfect-square-and-its-distance/
349     // Function to find the closest perfect square
350     static int getClosestPerfectSquare(int number) {
351         // Check if it is already a perfect number
352         if (isPerfect(number)) {
353             return number;
354         }
355
356         // Variables to store the perfect squares
357         int squareAbove = -1, squareBelow = -1;
358         int counter;
359
360         // Find first perfect square above the given
    number
361         counter = number + 1;
362         while (true) {
363             if (isPerfect(counter)) {
364                 squareAbove = counter;
365                 break;
366             } else {
367                 counter++;
368             }
369         }
370
371         // Find first perfect square below the given
    number
372         counter = number - 1;
373         while (true) {
374             if (isPerfect(counter)) {
375                 squareBelow = counter;
376                 break;
377             } else {
378                 counter--;
379             }
380         }
381
382         // Return the closest square
383         if ((squareAbove - number) > (number - squareBelow
```

```
383 )) {
384             return squareBelow;
385         } else {
386             return squareAbove;
387         }
388     }
389 
390     string name;
391     int population { };
392     double populationDensity { };
393     double squareMileage { };
394 
395     int arraySideLength { 0 };
396     vector<CityMile> cityChunks { };
397     vector<Person> people { };
398 
399     City() {
400         this->name = "";
401         this->population = 0;
402     }
403 
404     City(string name, double populationDensity, double
    squareMileage) {
405         this->name = std::move(name);
406         this->populationDensity = populationDensity;
407         this->squareMileage = squareMileage;
408 
409         int roundedPopulationDensity = ceil(
    populationDensity);
410         int numberOfChunks = ceil(squareMileage);
411         int arraySize = numberOfChunks - 1;
412 
413         arraySideLength = sqrt(getClosestPerfectSquare(
    numberOfChunks));
414         int x { 0 };
415         int y { 0 };
416 
417         // Create a chunk for every needed chunk.
418         #pragma omp parallel for
419         for(int index = 0; index < numberOfChunks; index
    ++) {
420             cout << "Creating Chunk " << index << endl;
421             for(int pIndex = 0; pIndex <
    roundedPopulationDensity; pIndex++) {
422                 Person tempPerson = Person(rollingPersonID
    , index);
423                 people.push_back(tempPerson);
424                 rollingPersonID ++;
```

```
425             }
426             cityChunks.emplace_back(index, x, y);
427             cout << "Finished Creating People Chunk " <<
    index << endl;
428 
429             // Make each chunk aware of its neighbors
430             // Check to see if we are on the edge of the
    square
431             if(x == 0) { // Left Edge
432                 if(index - arraySideLength < 0) { // Top
    Left Edge
433                     setIndex(index, 4, index + 1,
    arraySize);
434                     setIndex(index, 6, index +
    arraySideLength, arraySize);
435                     setIndex(index, 7, (index +
    arraySideLength) + 1, arraySize);
436                 } else if (index + arraySideLength >
    numberOfChunks - 1) { // Bottom Left Edge
437                     setIndex(index, 1, index -
    arraySideLength, arraySize);
438                     setIndex(index, 2, (index -
    arraySideLength) + 1, arraySize);
439                     setIndex(index, 4, index + 1,
    arraySize);
440                 } else { // Left Edge
441                     setIndex(index, 1, index -
    arraySideLength, arraySize);
442                     setIndex(index, 2, (index -
    arraySideLength) + 1, arraySize);
443                     setIndex(index, 4, index + 1,
    arraySize);
444                     setIndex(index, 6, index +
    arraySideLength, arraySize);
445                     setIndex(index, 7, (index +
    arraySideLength) + 1, arraySize);
446                 }
447             } else if (x == arraySideLength - 1) { //
    Right Edge
448                 if(index - arraySideLength < 0) { // Top
    Right Edge
449                     setIndex(index, 3, index - 1,
    arraySize);
450                     setIndex(index, 5, (index +
    arraySideLength) - 1, arraySize);
451                     setIndex(index, 6, index +
    arraySideLength, arraySize);
452                 } else if (index + arraySideLength >
```

```
452 arraySize) { // Bottom Right Edge
453                     setIndex(index, 0, (index -
    arraySideLength) - 1, arraySize);
454                     setIndex(index, 1, index -
    arraySideLength, arraySize);
455                     setIndex(index, 3, index - 1,
    arraySize);
456                 } else { // Right Edge
457                     setIndex(index, 0, (index -
    arraySideLength) - 1, arraySize);
458                     setIndex(index, 1, index -
    arraySideLength, arraySize);
459                     setIndex(index, 3, index - 1,
    arraySize);
460                     setIndex(index, 5, (index +
    arraySideLength) - 1, arraySize);
461                     setIndex(index, 6, index +
    arraySideLength, arraySize);
462                 }
463             } else { // No Edge
464                 if(index - arraySideLength < 0) { // Top
    Edge
465                     setIndex(index, 3, index - 1,
    arraySize);
466                     setIndex(index, 4, index + 1,
    arraySize);
467                     setIndex(index, 5, (index +
    arraySideLength) - 1, arraySize);
468                     setIndex(index, 6, index +
    arraySideLength, arraySize);
469                     setIndex(index, 7, (index +
    arraySideLength) + 1, arraySize);
470                 } else if (index + arraySideLength >
    arraySize) { // Bottom Edge
471                     setIndex(index, 0, (index -
    arraySideLength) - 1, arraySize);
472                     setIndex(index, 1, index -
    arraySideLength, arraySize);
473                     setIndex(index, 2, (index -
    arraySideLength) + 1, arraySize);
474                     setIndex(index, 3, index - 1,
    arraySize);
475                     setIndex(index, 4, index + 1,
    arraySize);
476                 } else { // No Edge
477                     setIndex(index, 0, (index -
    arraySideLength) - 1, arraySize);
478                     setIndex(index, 1, index -
```

```
478 arraySideLength, arraySize);
479                     setIndex(index, 2, (index -
    arraySideLength) + 1, arraySize);
480                     setIndex(index, 3, index - 1,
    arraySize);
481                     setIndex(index, 4, index + 1,
    arraySize);
482                     setIndex(index, 5, (index +
    arraySideLength) - 1, arraySize);
483                     setIndex(index, 6, index +
    arraySideLength, arraySize);
484                     setIndex(index, 7, (index +
    arraySideLength) + 1, arraySize);
485                 }
486             }
487
488             // Register all the possible indices for the
    city
489             cityChunks[index].setPossibleIndicies();
490
491             if (x >= arraySideLength - 1) {
492                 x = 0;
493                 y ++;
494             } else {
495                 x ++;
496             }
497         }
498
499        this->population = people.size();
500     }
501
502     void setIndex(int cityIndex, int arrayIndex, int
    chunkIndex, int arraySize) {
503         if(chunkIndex < arraySize && chunkIndex >= 0) {
504             cityChunks[cityIndex].neighborIndices[
    arrayIndex] = chunkIndex;
505         }
506     }
507 };
508
509 struct World {
510 //    vector<City> cities { };
511     City cities [1];
512     vector<Virus> viruses { };
513
514     static void infect(City& city, Virus& virus) {
515         double infectCount = city.people.size() *
    startingInfectionPercentage;
```

```
516         cout << "Infecting: " << infectCount << " people"
     << endl;
517         for(int x = 0; x < infectCount; x++) {
518             int randIndex = randomInt(0, city.people.size
    ());
519             city.people[randIndex].infect(virus, true);
520         }
521     }
522 
523     void simulate(int dayNumb) {
524         cout << "Begin Simulating Day " << dayNumb << endl
    ;
525         for(City& city: cities) {
526             // Loop over all the possible trip numbers
527             int tripCount = 9;
528 
529             cout << "Start Advance Infections" << endl;
530             #pragma omp parallel for
531             for (auto personIterator = begin (city.people
    ); personIterator != end (city.people); ++personIterator
    ) {
532                 (*personIterator).advanceInfection();
533             }
534             cout << "End Advance Infections" << endl;
535 
536             cout << "Start Trip Simulation" << endl;
537             string csvString = "";
538             for(int trip = 0; trip < tripCount; trip++) {
539                 // Simulate disease spread in parallel
    threads.
540                 cout << "Start Infection Simulation for
    trip " << trip << endl;
541                 #pragma omp parallel for
542                 for(auto chunkIterator = begin (city.
    cityChunks); chunkIterator != end (city.cityChunks); ++
    chunkIterator) {
543                     int numbSusceptible = 0;
544                     int numbLatent = 0;
545                     int numbInfectious = 0;
546                     int numbRecovered = 0;
547                     int numbImmune = 0;
548                     double healthSum = 0;
549 
550                     for(auto personIterator = begin (
    chunkIterator->people); personIterator != end (
    chunkIterator->people); ++personIterator) {
551                         int numberOfInteractions  =
    numberOfInteractionsPerTrip * (*personIterator)->
```

```
551 tripsToday;
552                         if (numberOfInteractions >
    chunkIterator->people.size()) {
553                             numberOfInteractions =
    chunkIterator->people.size();
554                         }
555 
556                         vector<Person*> interactions =
    chunkIterator->people;
557                         random_unique(interactions.begin
    (), interactions.end(), numberOfInteractions);
558 
559                         for(int n = 0; n <
    numberOfInteractions; n++) {
560                             Person* possibleInfector =
    interactions[n];
561                             // If we are looking at
    ourselves, skip.
562                             if (possibleInfector->id == (*
    personIterator)->id) {
563                                 continue;
564                             }
565                             // Check to see if the other
    person is infectious.
566                             if(possibleInfector->
    stageOfInfection == DiseaseStage::Infectious) {
567                                 if(possibleInfector->
    infectiousVirus.has_value()) {
568                                     (*personIterator)->
    infect(possibleInfector->infectiousVirus.value());
569                                 }
570                             }
571                         }
572 
573                         DiseaseStage stage = (*
    personIterator)->stageOfInfection;
574                         if(stage == Susceptible) {
575                             numbSusceptible ++;
576                         } else if (stage == Latent) {
577                             numbLatent ++;
578                         } else if (stage == Infectious) {
579                             numbInfectious ++;
580                         } else if (stage == Recovered) {
581                             numbRecovered ++;
582                         } else if (stage == Immune) {
583                             numbImmune ++;
584                         }
585                         healthSum += (*personIterator)->
```

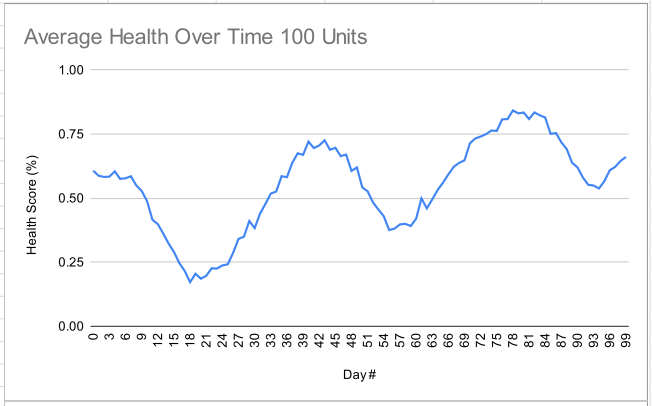
```
585 currentHealthCondition;
586                     }
587 
588                     // Clear people vector to allow for
    movement to happen, we only want to clear if we are about
    to move people
589                     if (trip != tripCount - 1) {
590                         chunkIterator->people.clear();
591                     } else {
592                         // + "," + to_string(trip)
593                         csvString += to_string(dayNumb) +
    "," +city.name + "," + to_string(city.people.size()) + ","
594                                 + to_string(chunkIterator
    ->id) + "," + to_string(chunkIterator->people.size()) +
    ","
595                                 + to_string(
    numbSusceptible) + "," + to_string(numbLatent) + "," +
    to_string(numbInfectious)
596                                 + "," + to_string(
    numbRecovered) + "," + to_string(numbImmune) + "," +
    to_string(healthSum / chunkIterator->people.size()) + "\n"
    ;
597                     }
598                 }
599                 cout << "End Infection Simulation for trip
     " << trip << endl;
600 
601                 cout << "Start Movement Simulation for
    trip " << trip << endl;
602                 // Move person to a new chunk
603                 if (trip != tripCount - 1) {
604                     for(auto personIterator = begin (city.
    people); personIterator != end (city.people); ++
    personIterator) {
605                         if(trip == 0) {
606                             int dailyTrips =
    numberOfDailyTrips + randomWeightedInt({5, 15, 60, 15, 5},
607 
                               {-4, -2, 0, 2, 4});
608                             personIterator->tripsToday =
    dailyTrips;
609                             personIterator->tripsCounter
     = dailyTrips;
610                         }
611 
612                         if (personIterator->tripsCounter
     < trip) {
613                             if (!personIterator->
```

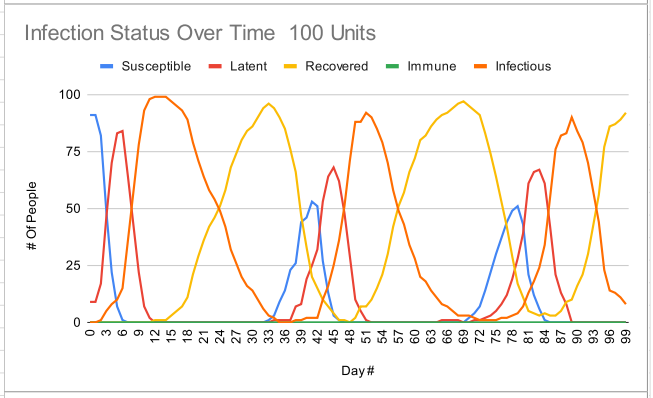
```
613 isOnRoundTrip) { // We are not on a return trip so perform
     a move normally
614                                     if(randomDouble(0, 1) <=
    roundTripChance) {
615                                         personIterator->
    isOnRoundTrip = true;
616                                         personIterator->
    roundTripCounter = 0;
617                                         personIterator->
    tripLength = randomWeightedInt({0.5, 0.3, 0.1},{2, 4, 6
    }); // Pick the number length of the trip based on trip
    cycles.
618                                         personIterator->
    tripStartChunkIndex = personIterator->parentChunkIndex;
619                                     }
620
621                                     int travelDistance =
    averageTripDistanceMiles +
622
    randomWeightedInt({2.5, 10, 20, 30, 20, 10, 5, 2.5},
623
                 {-5, -3, -1, 0, 1, 3, 5, 10});
624                                     for (int x = 0; x <
    travelDistance; x++) {
625                                         int parentChunkIndex
     = personIterator->parentChunkIndex;
626                                         int
    randomNeighborIndex = city.cityChunks[parentChunkIndex].
    randomNeighborIndex();
627                                         personIterator->
    parentChunkIndex = randomNeighborIndex;
628                                         personIterator->
    tripsCounter --;
629                                     }
630                                     personIterator->
    roundTripCounter ++;
631                                 } else { // We are on a round
    trip
632                                     if(personIterator->
    roundTripCounter >= personIterator->tripLength - 1) { //
    We need to be home on our next trip
633                                         personIterator->
    parentChunkIndex = personIterator->tripStartChunkIndex;
634                                         personIterator->
    tripStartChunkIndex = 0;
635                                         personIterator->
    tripLength = 0;
636                                         personIterator->
```

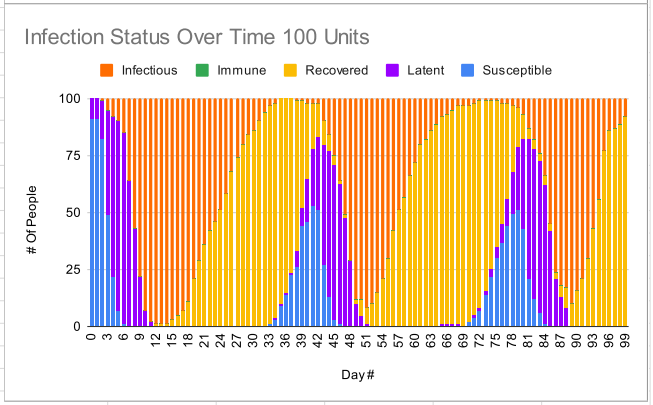
```
636 isOnRoundTrip = false;
637
638                                     personIterator->
    tripsCounter --;
639                                 } else { // We can keep
    moving
640                                     int travelDistance =
    averageTripDistanceMiles +
641
    randomWeightedInt({2.5, 10, 20, 30, 20, 10, 5, 2.5},
642
                      {-5, -3, -1, 0, 1, 3, 5, 10});
643                                     for (int x = 0; x <
    travelDistance; x++) {
644                                         int
    parentChunkIndex = personIterator->parentChunkIndex;
645                                         int
    randomNeighborIndex = city.cityChunks[parentChunkIndex].
    randomNeighborIndex();
646                                         personIterator->
    parentChunkIndex = randomNeighborIndex;
647                                         personIterator->
    tripsCounter --;
648                                     }
649                                     personIterator->
    roundTripCounter ++;
650                                 }
651                             }
652                         }
653
654                         city.cityChunks[personIterator->
    parentChunkIndex].people.push_back(&(*personIterator));
655                     }
656                 }
657                 cout << "End Movement Simulation for trip
     " << trip << endl;
658             }
659             cout << "End Trip Simulation" << endl;
660
661             ofstream dataCSV;
662             dataCSV.open ("./Daily_Infection_Numbers.csv"
    , ios_base::app);
663             dataCSV << csvString;
664             dataCSV.close();
665         }
666         cout << "End Simulating Day " << dayNumb << endl;
667     }
668 };
```

```
669
670 int main() {
671     srand (static_cast <unsigned> (time(nullptr)));
672     World Earth = World();
673
674     // Read in cities from data (Currently Just One City)
675     Earth.cities[0] = City("Philadelphia",  10, 10);
676     Earth.viruses.emplace_back(Virus("Argo 1", {Touch,
    Saliva, Coughing, Sneezing, SexualContact, Contamination,
    Insects}));
677     Earth.infect(Earth.cities[0], Earth.viruses[0]);
678
679     ofstream dataCSV;
680     dataCSV.open ("./Daily_Infection_Numbers.csv");
681     //Trip,
682     dataCSV << "Date,City,City Population,Chunk Number,
    Chunk Population,Susceptible,Latent,Infectious,Recovered,
    Immune,Average Health\n";
683     dataCSV.close();
684
685     int daysToSimulate { 100 };
686     for(int x = 0; x < daysToSimulate; x++) {
687         if (!halt) {
688             Earth.simulate(x);
689             cout << "Finished day: " << x << endl;
690         }
691     }
692     cout << "    Finished Simulation    " << endl;
693
694     return 0;
695 };
```

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
0	Philadelphia	100	0	7	7	0	0	0	0	0.728571
0	Philadelphia	100	1	9	7	2	0	0	0	0.565556
0	Philadelphia	100	2	7	7	0	0	0	0	0.557143
0	Philadelphia	100	3	10	9	1	0	0	0	0.6395
0	Philadelphia	100	4	25	24	1	0	0	0	0.6558
0	Philadelphia	100	5	15	15	0	0	0	0	0.486667
0	Philadelphia	100	6	8	6	2	0	0	0	0.66125
0	Philadelphia	100	7	15	12	3	0	0	0	0.545667
0	Philadelphia	100	8	4	4	0	0	0	0	0.625
0	Philadelphia	100	9	0	0	0	0	0	0	nan
1	Philadelphia	100	0	8	8	0	0	0	0	0.4
1	Philadelphia	100	1	15	15	0	0	0	0	0.613333
1	Philadelphia	100	2	5	4	1	0	0	0	0.698
1	Philadelphia	100	3	11	10	1	0	0	0	0.599091
1	Philadelphia	100	4	21	18	3	0	0	0	0.650952
1	Philadelphia	100	5	12	11	1	0	0	0	0.624167
1	Philadelphia	100	6	5	4	1	0	0	0	0.638
1	Philadelphia	100	7	9	9	0	0	0	0	0.677778
1	Philadelphia	100	8	12	10	2	0	0	0	0.523333
1	Philadelphia	100	9	2	2	0	0	0	0	0.45
2	Philadelphia	100	0	5	5	0	0	0	0	0.4
2	Philadelphia	100	1	16	12	3	1	0	0	0.660937
2	Philadelphia	100	2	13	12	1	0	0	0	0.660385
2	Philadelphia	100	3	8	4	4	0	0	0	0.523125
2	Philadelphia	100	4	23	22	1	0	0	0	0.634783
2	Philadelphia	100	5	10	6	4	0	0	0	0.617
2	Philadelphia	100	6	8	7	1	0	0	0	0.4875
2	Philadelphia	100	7	11	9	2	0	0	0	0.497273
2	Philadelphia	100	8	6	5	1	0	0	0	0.764167
2	Philadelphia	100	9	0	0	0	0	0	0	nan
3	Philadelphia	100	0	6	4	2	0	0	0	0.566667
3	Philadelphia	100	1	12	9	3	0	0	0	0.683333
3	Philadelphia	100	2	10	4	5	1	0	0	0.6075
3	Philadelphia	100	3	12	7	5	0	0	0	0.637917
3	Philadelphia	100	4	19	11	8	0	0	0	0.557368
3	Philadelphia	100	5	10	4	6	0	0	0	0.539
3	Philadelphia	100	6	8	3	3	2	0	0	0.60625
3	Philadelphia	100	7	8	1	6	1	0	0	0.664375
3	Philadelphia	100	8	14	5	8	1	0	0	0.577143
3	Philadelphia	100	9	1	1	0	0	0	0	0.4
4	Philadelphia	100	0	2	0	1	1	0	0	0.5875
4	Philadelphia	100	1	9	4	4	1	0	0	0.568333
4	Philadelphia	100	2	9	3	5	1	0	0	0.625
4	Philadelphia	100	3	20	5	14	1	0	0	0.582
4	Philadelphia	100	4	21	6	15	0	0	0	0.587857
4	Philadelphia	100	5	12	1	10	1	0	0	0.67125
4	Philadelphia	100	6	9	2	7	0	0	0	0.626667
4	Philadelphia	100	7	14	1	12	1	0	0	0.513571
4	Philadelphia	100	8	4	0	2	2	0	0	0.67875
4	Philadelphia	100	9	0	0	0	0	0	0	nan
5	Philadelphia	100	0	7	0	6	1	0	0	0.629286
5	Philadelphia	100	1	11	0	11	0	0	0	0.685455
5	Philadelphia	100	2	10	0	10	0	0	0	0.544
5	Philadelphia	100	3	10	1	8	1	0	0	0.573
5	Philadelphia	100	4	15	3	8	4	0	0	0.595
5	Philadelphia	100	5	21	2	17	2	0	0	0.58
5	Philadelphia	100	6	3	1	2	0	0	0	0.428333
5	Philadelphia	100	7	16	0	14	2	0	0	0.59125







Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
5	Philadelphia	100	8	7	0	7	0	0	0	0.549286
5	Philadelphia	100	9	0	0	0	0	0	0	nan
6	Philadelphia	100	0	4	0	4	0	0	0	0.5625
6	Philadelphia	100	1	10	0	9	1	0	0	0.6905
6	Philadelphia	100	2	8	0	6	2	0	0	0.49125
6	Philadelphia	100	3	16	0	15	1	0	0	0.589062
6	Philadelphia	100	4	20	0	15	5	0	0	0.585
6	Philadelphia	100	5	9	0	7	2	0	0	0.595
6	Philadelphia	100	6	11	1	9	1	0	0	0.535
6	Philadelphia	100	7	12	0	9	3	0	0	0.507917
6	Philadelphia	100	8	10	0	10	0	0	0	0.6425
6	Philadelphia	100	9	0	0	0	0	0	0	nan
7	Philadelphia	100	0	10	0	6	4	0	0	0.4585
7	Philadelphia	100	1	17	0	12	5	0	0	0.579706
7	Philadelphia	100	2	7	0	3	4	0	0	0.592857
7	Philadelphia	100	3	10	0	7	3	0	0	0.7105
7	Philadelphia	100	4	22	0	14	8	0	0	0.467727
7	Philadelphia	100	5	8	0	5	3	0	0	0.4725
7	Philadelphia	100	6	3	0	2	1	0	0	0.668333
7	Philadelphia	100	7	9	0	5	4	0	0	0.656111
7	Philadelphia	100	8	14	0	10	4	0	0	0.663571
7	Philadelphia	100	9	0	0	0	0	0	0	nan
8	Philadelphia	100	0	6	0	0	6	0	0	0.440833
8	Philadelphia	100	1	15	0	9	6	0	0	0.598667
8	Philadelphia	100	2	10	0	3	7	0	0	0.494
8	Philadelphia	100	3	12	0	6	6	0	0	0.54125
8	Philadelphia	100	4	14	0	3	11	0	0	0.575357
8	Philadelphia	100	5	14	0	8	6	0	0	0.535
8	Philadelphia	100	6	5	0	2	3	0	0	0.64
8	Philadelphia	100	7	14	0	5	9	0	0	0.512857
8	Philadelphia	100	8	10	0	7	3	0	0	0.6075
8	Philadelphia	100	9	0	0	0	0	0	0	nan
9	Philadelphia	100	0	7	0	1	6	0	0	0.447143
9	Philadelphia	100	1	13	0	3	10	0	0	0.478077
9	Philadelphia	100	2	11	0	1	10	0	0	0.509091
9	Philadelphia	100	3	6	0	0	6	0	0	0.6225
9	Philadelphia	100	4	20	0	5	15	0	0	0.4725
9	Philadelphia	100	5	12	0	4	8	0	0	0.561667
9	Philadelphia	100	6	9	0	3	6	0	0	0.527222
9	Philadelphia	100	7	10	0	1	9	0	0	0.464
9	Philadelphia	100	8	12	0	4	8	0	0	0.6725
9	Philadelphia	100	9	0	0	0	0	0	0	nan
10	Philadelphia	100	0	7	0	0	7	0	0	0.5
10	Philadelphia	100	1	12	0	3	9	0	0	0.720833
10	Philadelphia	100	2	9	0	0	9	0	0	0.525
10	Philadelphia	100	3	8	0	1	7	0	0	0.425625
10	Philadelphia	100	4	17	0	1	16	0	0	0.400588
10	Philadelphia	100	5	14	0	0	14	0	0	0.471429
10	Philadelphia	100	6	7	0	0	7	0	0	0.387857
10	Philadelphia	100	7	12	0	1	11	0	0	0.545
10	Philadelphia	100	8	14	0	1	13	0	0	0.418571
10	Philadelphia	100	9	0	0	0	0	0	0	nan
11	Philadelphia	100	0	3	0	0	3	0	0	0.09
11	Philadelphia	100	1	15	0	0	15	0	0	0.541
11	Philadelphia	100	2	8	0	0	8	0	0	0.445625
11	Philadelphia	100	3	10	0	0	10	0	0	0.478
11	Philadelphia	100	4	19	0	1	18	0	0	0.430263
11	Philadelphia	100	5	10	0	0	10	0	0	0.4155

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
11	Philadelphia	100	6	7	0	0	7	0	0	0.455714
11	Philadelphia	100	7	12	0	1	11	0	0	0.449167
11	Philadelphia	100	8	16	0	0	16	0	0	0.436563
11	Philadelphia	100	9	0	0	0	0	0	0	nan
12	Philadelphia	100	0	4	0	0	4	0	0	0.49625
12	Philadelphia	100	1	14	0	0	14	0	0	0.454286
12	Philadelphia	100	2	9	0	0	9	0	0	0.307222
12	Philadelphia	100	3	20	0	0	20	0	0	0.34675
12	Philadelphia	100	4	18	0	0	18	0	0	0.482778
12	Philadelphia	100	5	17	0	0	16	1	0	0.370294
12	Philadelphia	100	6	4	0	0	4	0	0	0.345
12	Philadelphia	100	7	9	0	0	9	0	0	0.498333
12	Philadelphia	100	8	5	0	0	5	0	0	0.286
12	Philadelphia	100	9	0	0	0	0	0	0	nan
13	Philadelphia	100	0	2	0	0	2	0	0	0.1175
13	Philadelphia	100	1	19	0	0	19	0	0	0.291316
13	Philadelphia	100	2	7	0	0	7	0	0	0.332857
13	Philadelphia	100	3	15	0	0	15	0	0	0.427667
13	Philadelphia	100	4	29	0	0	29	0	0	0.35069
13	Philadelphia	100	5	9	0	0	9	0	0	0.437778
13	Philadelphia	100	6	5	0	0	5	0	0	0.393
13	Philadelphia	100	7	11	0	0	11	0	0	0.335
13	Philadelphia	100	8	3	0	0	2	1	0	0.575
13	Philadelphia	100	9	0	0	0	0	0	0	nan
14	Philadelphia	100	0	10	0	0	10	0	0	0.3745
14	Philadelphia	100	1	13	0	0	13	0	0	0.247308
14	Philadelphia	100	2	7	0	0	7	0	0	0.288571
14	Philadelphia	100	3	15	0	0	15	0	0	0.319333
14	Philadelphia	100	4	23	0	0	22	1	0	0.301957
14	Philadelphia	100	5	8	0	0	8	0	0	0.320625
14	Philadelphia	100	6	10	0	0	10	0	0	0.367
14	Philadelphia	100	7	8	0	0	8	0	0	0.36875
14	Philadelphia	100	8	6	0	0	6	0	0	0.3175
14	Philadelphia	100	9	0	0	0	0	0	0	nan
15	Philadelphia	100	0	6	0	0	6	0	0	0.359167
15	Philadelphia	100	1	13	0	0	13	0	0	0.197692
15	Philadelphia	100	2	8	0	0	8	0	0	0.30875
15	Philadelphia	100	3	14	0	0	14	0	0	0.354286
15	Philadelphia	100	4	18	0	0	16	2	0	0.244444
15	Philadelphia	100	5	10	0	0	10	0	0	0.3535
15	Philadelphia	100	6	6	0	0	6	0	0	0.308333
15	Philadelphia	100	7	9	0	0	9	0	0	0.244444
15	Philadelphia	100	8	16	0	0	15	1	0	0.239062
15	Philadelphia	100	9	0	0	0	0	0	0	nan
16	Philadelphia	100	0	5	0	0	5	0	0	0.221
16	Philadelphia	100	1	10	0	0	10	0	0	0.1665
16	Philadelphia	100	2	9	0	0	9	0	0	0.258889
16	Philadelphia	100	3	20	0	0	20	0	0	0.151
16	Philadelphia	100	4	18	0	0	17	1	0	0.254444
16	Philadelphia	100	5	13	0	0	12	1	0	0.273462
16	Philadelphia	100	6	5	0	0	5	0	0	0.125
16	Philadelphia	100	7	12	0	0	9	3	0	0.404167
16	Philadelphia	100	8	8	0	0	8	0	0	0.36375
16	Philadelphia	100	9	0	0	0	0	0	0	nan
17	Philadelphia	100	0	4	0	0	4	0	0	0
17	Philadelphia	100	1	15	0	0	15	0	0	0.248333
17	Philadelphia	100	2	8	0	0	7	1	0	0.3
17	Philadelphia	100	3	14	0	0	13	1	0	0.283929

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
17	Philadelphia	100	4	20	0	0	17	3	0	0.186
17	Philadelphia	100	5	13	0	0	12	1	0	0.225385
17	Philadelphia	100	6	5	0	0	4	1	0	0.466
17	Philadelphia	100	7	15	0	0	15	0	0	0.137667
17	Philadelphia	100	8	6	0	0	6	0	0	0.096667
17	Philadelphia	100	9	0	0	0	0	0	0	nan
18	Philadelphia	100	0	8	0	0	8	0	0	0.058125
18	Philadelphia	100	1	14	0	0	14	0	0	0.139643
18	Philadelphia	100	2	4	0	0	4	0	0	0
18	Philadelphia	100	3	10	0	0	9	1	0	0.2085
18	Philadelphia	100	4	22	0	0	19	3	0	0.184318
18	Philadelphia	100	5	12	0	0	10	2	0	0.285417
18	Philadelphia	100	6	5	0	0	4	1	0	0.202
18	Philadelphia	100	7	15	0	0	11	4	0	0.360667
18	Philadelphia	100	8	10	0	0	10	0	0	0.1105
18	Philadelphia	100	9	0	0	0	0	0	0	nan
19	Philadelphia	100	0	3	0	0	2	1	0	0.558333
19	Philadelphia	100	1	19	0	0	12	7	0	0.227632
19	Philadelphia	100	2	11	0	0	9	2	0	0.291818
19	Philadelphia	100	3	11	0	0	7	4	0	0.23
19	Philadelphia	100	4	14	0	0	13	1	0	0.087857
19	Philadelphia	100	5	9	0	0	8	1	0	0.091111
19	Philadelphia	100	6	6	0	0	6	0	0	0.09
19	Philadelphia	100	7	19	0	0	14	5	0	0.173684
19	Philadelphia	100	8	8	0	0	8	0	0	0.094375
19	Philadelphia	100	9	0	0	0	0	0	0	nan
20	Philadelphia	100	0	12	0	0	9	3	0	0.239583
20	Philadelphia	100	1	13	0	0	10	3	0	0.110769
20	Philadelphia	100	2	10	0	0	6	4	0	0.206
20	Philadelphia	100	3	12	0	0	9	3	0	0.102917
20	Philadelphia	100	4	13	0	0	11	2	0	0.119615
20	Philadelphia	100	5	9	0	0	5	4	0	0.332222
20	Philadelphia	100	6	12	0	0	9	3	0	0.161667
20	Philadelphia	100	7	12	0	0	7	5	0	0.287083
20	Philadelphia	100	8	7	0	0	5	2	0	0.110714
20	Philadelphia	100	9	0	0	0	0	0	0	nan
21	Philadelphia	100	0	6	0	0	2	4	0	0.321667
21	Philadelphia	100	1	10	0	0	5	5	0	0.2735
21	Philadelphia	100	2	6	0	0	4	2	0	0.3325
21	Philadelphia	100	3	21	0	0	13	8	0	0.156667
21	Philadelphia	100	4	19	0	0	13	6	0	0.188684
21	Philadelphia	100	5	10	0	0	5	5	0	0.335
21	Philadelphia	100	6	13	0	0	9	4	0	0.112308
21	Philadelphia	100	7	13	0	0	11	2	0	0.053462
21	Philadelphia	100	8	2	0	0	2	0	0	0
21	Philadelphia	100	9	0	0	0	0	0	0	nan
22	Philadelphia	100	0	7	0	0	3	4	0	0.32
22	Philadelphia	100	1	11	0	0	6	5	0	0.178182
22	Philadelphia	100	2	7	0	0	2	5	0	0.471429
22	Philadelphia	100	3	9	0	0	5	4	0	0.216667
22	Philadelphia	100	4	16	0	0	13	3	0	0.096875
22	Philadelphia	100	5	11	0	0	7	4	0	0.096818
22	Philadelphia	100	6	11	0	0	4	7	0	0.360455
22	Philadelphia	100	7	16	0	0	10	6	0	0.176562
22	Philadelphia	100	8	12	0	0	8	4	0	0.119583
22	Philadelphia	100	9	0	0	0	0	0	0	nan
23	Philadelphia	100	0	9	0	0	6	3	0	0.208889
23	Philadelphia	100	1	7	0	0	2	5	0	0.449286

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
23	Philadelphia	100	2	9	0	0	6	3	0	0.121667
23	Philadelphia	100	3	11	0	0	5	6	0	0.257727
23	Philadelphia	100	4	32	0	0	19	13	0	0.177812
23	Philadelphia	100	5	10	0	0	4	6	0	0.3665
23	Philadelphia	100	6	6	0	0	2	4	0	0.255833
23	Philadelphia	100	7	12	0	0	7	5	0	0.156667
23	Philadelphia	100	8	4	0	0	3	1	0	0.03375
23	Philadelphia	100	9	0	0	0	0	0	0	nan
24	Philadelphia	100	0	10	0	0	5	5	0	0.276
24	Philadelphia	100	1	13	0	0	5	8	0	0.258077
24	Philadelphia	100	2	7	0	0	4	3	0	0.116429
24	Philadelphia	100	3	8	0	0	5	3	0	0.15625
24	Philadelphia	100	4	20	0	0	14	6	0	0.10525
24	Philadelphia	100	5	18	0	0	7	11	0	0.366667
24	Philadelphia	100	6	6	0	0	3	3	0	0.2
24	Philadelphia	100	7	12	0	0	4	8	0	0.278333
24	Philadelphia	100	8	6	0	0	2	4	0	0.38
24	Philadelphia	100	9	0	0	0	0	0	0	nan
25	Philadelphia	100	0	6	0	0	1	5	0	0.380833
25	Philadelphia	100	1	7	0	0	2	5	0	0.364286
25	Philadelphia	100	2	5	0	0	4	1	0	0.009
25	Philadelphia	100	3	8	0	0	2	6	0	0.395625
25	Philadelphia	100	4	35	0	0	17	18	0	0.275429
25	Philadelphia	100	5	17	0	0	7	10	0	0.28
25	Philadelphia	100	6	7	0	0	1	6	0	0.128571
25	Philadelphia	100	7	5	0	0	3	2	0	0.211
25	Philadelphia	100	8	10	0	0	5	5	0	0.1335
25	Philadelphia	100	9	0	0	0	0	0	0	nan
26	Philadelphia	100	0	8	0	0	3	5	0	0.340625
26	Philadelphia	100	1	12	0	0	4	8	0	0.234583
26	Philadelphia	100	2	11	0	0	5	6	0	0.216364
26	Philadelphia	100	3	6	0	0	1	5	0	0.305833
26	Philadelphia	100	4	18	0	0	5	13	0	0.347222
26	Philadelphia	100	5	13	0	0	5	8	0	0.343846
26	Philadelphia	100	6	8	0	0	1	7	0	0.465
26	Philadelphia	100	7	13	0	0	5	8	0	0.131923
26	Philadelphia	100	8	11	0	0	3	8	0	0.196818
26	Philadelphia	100	9	0	0	0	0	0	0	nan
27	Philadelphia	100	0	6	0	0	1	5	0	0.218333
27	Philadelphia	100	1	16	0	0	6	10	0	0.204687
27	Philadelphia	100	2	10	0	0	4	6	0	0.272
27	Philadelphia	100	3	12	0	0	4	8	0	0.260417
27	Philadelphia	100	4	20	0	0	4	16	0	0.35375
27	Philadelphia	100	5	13	0	0	3	10	0	0.279615
27	Philadelphia	100	6	5	0	0	1	4	0	0.422
27	Philadelphia	100	7	15	0	0	3	12	0	0.367333
27	Philadelphia	100	8	3	0	0	0	3	0	0.691667
27	Philadelphia	100	9	0	0	0	0	0	0	nan
28	Philadelphia	100	0	7	0	0	0	7	0	0.302857
28	Philadelphia	100	1	7	0	0	1	6	0	0.536429
28	Philadelphia	100	2	11	0	0	2	9	0	0.199091
28	Philadelphia	100	3	12	0	0	2	10	0	0.312917
28	Philadelphia	100	4	26	0	0	3	23	0	0.337885
28	Philadelphia	100	5	14	0	0	6	8	0	0.338929
28	Philadelphia	100	6	13	0	0	3	10	0	0.262308
28	Philadelphia	100	7	10	0	0	3	7	0	0.506
28	Philadelphia	100	8	0	0	0	0	0	0	nan
28	Philadelphia	100	9	0	0	0	0	0	0	nan

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
29	Philadelphia	100	0	10	0	0	2	8	0	0.4265
29	Philadelphia	100	1	12	0	0	2	10	0	0.356667
29	Philadelphia	100	2	6	0	0	2	4	0	0.309167
29	Philadelphia	100	3	14	0	0	2	12	0	0.232143
29	Philadelphia	100	4	19	0	0	3	16	0	0.269737
29	Philadelphia	100	5	13	0	0	1	12	0	0.436154
29	Philadelphia	100	6	3	0	0	0	3	0	0.805
29	Philadelphia	100	7	15	0	0	2	13	0	0.469
29	Philadelphia	100	8	8	0	0	2	6	0	0.3925
29	Philadelphia	100	9	0	0	0	0	0	0	nan
30	Philadelphia	100	0	10	0	0	1	9	0	0.3025
30	Philadelphia	100	1	9	0	0	2	7	0	0.368333
30	Philadelphia	100	2	6	0	0	2	4	0	0.4225
30	Philadelphia	100	3	10	0	0	0	10	0	0.3685
30	Philadelphia	100	4	28	0	0	4	24	0	0.385357
30	Philadelphia	100	5	8	0	0	0	8	0	0.47375
30	Philadelphia	100	6	6	0	0	2	4	0	0.316667
30	Philadelphia	100	7	20	0	0	2	18	0	0.52375
30	Philadelphia	100	8	3	0	0	1	2	0	0.285
30	Philadelphia	100	9	0	0	0	0	0	0	nan
31	Philadelphia	100	0	5	0	0	0	5	0	0.344
31	Philadelphia	100	1	7	0	0	0	7	0	0.454286
31	Philadelphia	100	2	7	0	0	3	4	0	0.334286
31	Philadelphia	100	3	11	0	0	1	10	0	0.365455
31	Philadelphia	100	4	24	0	0	3	21	0	0.404375
31	Philadelphia	100	5	15	0	0	1	14	0	0.436667
31	Philadelphia	100	6	14	0	0	2	12	0	0.506429
31	Philadelphia	100	7	10	0	0	0	10	0	0.4445
31	Philadelphia	100	8	7	0	0	0	7	0	0.667857
31	Philadelphia	100	9	0	0	0	0	0	0	nan
32	Philadelphia	100	0	10	0	0	0	10	0	0.439
32	Philadelphia	100	1	11	0	0	0	11	0	0.592727
32	Philadelphia	100	2	9	0	0	2	7	0	0.387778
32	Philadelphia	100	3	9	0	0	2	7	0	0.473889
32	Philadelphia	100	4	15	0	0	1	14	0	0.449667
32	Philadelphia	100	5	14	0	0	1	13	0	0.413214
32	Philadelphia	100	6	12	0	0	0	12	0	0.389583
32	Philadelphia	100	7	13	0	0	0	13	0	0.546154
32	Philadelphia	100	8	7	0	0	0	7	0	0.606429
32	Philadelphia	100	9	0	0	0	0	0	0	nan
33	Philadelphia	100	0	16	0	0	2	14	0	0.505
33	Philadelphia	100	1	11	0	0	0	11	0	0.581818
33	Philadelphia	100	2	4	0	0	0	4	0	0.5075
33	Philadelphia	100	3	12	0	0	0	12	0	0.482917
33	Philadelphia	100	4	21	0	0	1	20	0	0.439762
33	Philadelphia	100	5	9	0	0	0	9	0	0.615
33	Philadelphia	100	6	9	0	0	0	9	0	0.670556
33	Philadelphia	100	7	8	1	0	0	7	0	0.41875
33	Philadelphia	100	8	10	0	0	0	10	0	0.4395
33	Philadelphia	100	9	0	0	0	0	0	0	nan
34	Philadelphia	100	0	5	0	0	0	5	0	0.407
34	Philadelphia	100	1	9	1	0	0	8	0	0.416667
34	Philadelphia	100	2	12	0	1	1	10	0	0.42625
34	Philadelphia	100	3	12	1	0	0	11	0	0.671667
34	Philadelphia	100	4	29	0	0	1	28	0	0.587931
34	Philadelphia	100	5	13	1	0	0	12	0	0.566923
34	Philadelphia	100	6	7	0	0	0	7	0	0.632143
34	Philadelphia	100	7	6	0	0	0	6	0	0.528333

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
34	Philadelphia	100	8	7	0	0	0	7	0	0.495
34	Philadelphia	100	9	0	0	0	0	0	0	nan
35	Philadelphia	100	0	4	0	0	0	4	0	0.75
35	Philadelphia	100	1	7	1	0	0	6	0	0.529286
35	Philadelphia	100	2	9	1	0	0	8	0	0.510556
35	Philadelphia	100	3	11	1	0	0	10	0	0.520455
35	Philadelphia	100	4	23	3	1	0	19	0	0.568696
35	Philadelphia	100	5	13	1	0	0	12	0	0.566154
35	Philadelphia	100	6	7	0	0	0	7	0	0.53
35	Philadelphia	100	7	16	1	0	0	15	0	0.604375
35	Philadelphia	100	8	10	1	0	0	9	0	0.692
35	Philadelphia	100	9	0	0	0	0	0	0	nan
36	Philadelphia	100	0	5	0	0	0	5	0	0.506
36	Philadelphia	100	1	13	5	0	0	8	0	0.671154
36	Philadelphia	100	2	3	1	0	0	2	0	0.468333
36	Philadelphia	100	3	9	0	0	0	9	0	0.644444
36	Philadelphia	100	4	21	3	0	0	18	0	0.544524
36	Philadelphia	100	5	20	2	0	0	18	0	0.7095
36	Philadelphia	100	6	7	1	1	0	5	0	0.547857
36	Philadelphia	100	7	17	2	0	0	15	0	0.616176
36	Philadelphia	100	8	5	0	0	0	5	0	0.53
36	Philadelphia	100	9	0	0	0	0	0	0	nan
37	Philadelphia	100	0	12	2	0	0	10	0	0.617083
37	Philadelphia	100	1	9	1	0	0	8	0	0.718889
37	Philadelphia	100	2	9	2	0	0	7	0	0.633333
37	Philadelphia	100	3	11	3	1	0	7	0	0.754091
37	Philadelphia	100	4	18	5	0	0	13	0	0.629444
37	Philadelphia	100	5	15	5	0	0	10	0	0.566667
37	Philadelphia	100	6	5	1	0	0	4	0	0.551
37	Philadelphia	100	7	12	2	0	0	10	0	0.655
37	Philadelphia	100	8	9	2	0	0	7	0	0.622778
37	Philadelphia	100	9	0	0	0	0	0	0	nan
38	Philadelphia	100	0	8	1	1	0	6	0	0.69
38	Philadelphia	100	1	6	1	0	0	5	0	0.625833
38	Philadelphia	100	2	9	0	0	0	9	0	0.747778
38	Philadelphia	100	3	10	4	1	0	5	0	0.6935
38	Philadelphia	100	4	24	5	2	1	16	0	0.626875
38	Philadelphia	100	5	14	4	3	0	7	0	0.693571
38	Philadelphia	100	6	4	3	0	0	1	0	0.73625
38	Philadelphia	100	7	12	4	0	0	8	0	0.677083
38	Philadelphia	100	8	13	4	0	0	9	0	0.583462
38	Philadelphia	100	9	0	0	0	0	0	0	nan
39	Philadelphia	100	0	12	5	1	0	6	0	0.734167
39	Philadelphia	100	1	9	4	0	0	5	0	0.727222
39	Philadelphia	100	2	7	3	2	0	2	0	0.803571
39	Philadelphia	100	3	16	5	3	0	8	0	0.609062
39	Philadelphia	100	4	25	13	1	0	11	0	0.7232
39	Philadelphia	100	5	8	6	0	1	1	0	0.67625
39	Philadelphia	100	6	5	2	0	0	3	0	0.558
39	Philadelphia	100	7	11	3	1	0	7	0	0.693636
39	Philadelphia	100	8	7	3	0	0	4	0	0.496429
39	Philadelphia	100	9	0	0	0	0	0	0	nan
40	Philadelphia	100	0	8	5	1	0	2	0	0.7675
40	Philadelphia	100	1	11	5	2	1	3	0	0.720455
40	Philadelphia	100	2	10	3	0	0	7	0	0.57
40	Philadelphia	100	3	19	6	5	1	7	0	0.638947
40	Philadelphia	100	4	15	8	5	0	2	0	0.713667
40	Philadelphia	100	5	15	8	2	0	5	0	0.657333

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
40	Philadelphia	100	6	4	3	0	0	1	0	0.95125
40	Philadelphia	100	7	8	4	3	0	1	0	0.805
40	Philadelphia	100	8	10	4	1	0	5	0	0.663
40	Philadelphia	100	9	0	0	0	0	0	0	nan
41	Philadelphia	100	0	10	4	5	0	1	0	0.63
41	Philadelphia	100	1	8	4	1	0	3	0	0.700625
41	Philadelphia	100	2	10	3	3	1	3	0	0.622
41	Philadelphia	100	3	13	8	2	0	3	0	0.687308
41	Philadelphia	100	4	20	11	6	0	3	0	0.716
41	Philadelphia	100	5	11	7	2	0	2	0	0.618636
41	Philadelphia	100	6	7	4	1	0	2	0	0.717857
41	Philadelphia	100	7	15	7	4	1	3	0	0.821667
41	Philadelphia	100	8	6	5	1	0	0	0	0.745
41	Philadelphia	100	9	0	0	0	0	0	0	nan
42	Philadelphia	100	0	5	1	2	1	1	0	0.586
42	Philadelphia	100	1	12	7	3	0	2	0	0.67875
42	Philadelphia	100	2	11	4	6	0	1	0	0.619545
42	Philadelphia	100	3	21	13	1	0	7	0	0.664048
42	Philadelphia	100	4	12	4	6	0	2	0	0.684167
42	Philadelphia	100	5	11	5	5	1	0	0	0.796818
42	Philadelphia	100	6	9	6	3	0	0	0	0.847222
42	Philadelphia	100	7	9	6	3	0	0	0	0.73
42	Philadelphia	100	8	10	5	3	0	2	0	0.745
42	Philadelphia	100	9	0	0	0	0	0	0	nan
43	Philadelphia	100	0	11	0	9	0	2	0	0.615455
43	Philadelphia	100	1	13	4	5	1	3	0	0.602308
43	Philadelphia	100	2	7	1	5	1	0	0	0.735
43	Philadelphia	100	3	10	5	4	1	0	0	0.784
43	Philadelphia	100	4	24	7	12	3	2	0	0.671458
43	Philadelphia	100	5	12	3	7	1	1	0	0.717083
43	Philadelphia	100	6	9	4	3	1	1	0	0.84
43	Philadelphia	100	7	9	1	5	2	1	0	0.746667
43	Philadelphia	100	8	5	2	3	0	0	0	0.82
43	Philadelphia	100	9	0	0	0	0	0	0	nan
44	Philadelphia	100	0	3	0	3	0	0	0	0.52
44	Philadelphia	100	1	16	2	10	1	3	0	0.68375
44	Philadelphia	100	2	6	0	3	3	0	0	0.735833
44	Philadelphia	100	3	15	1	12	2	0	0	0.700667
44	Philadelphia	100	4	22	3	14	4	1	0	0.729091
44	Philadelphia	100	5	13	2	8	2	1	0	0.675385
44	Philadelphia	100	6	6	3	0	2	1	0	0.680833
44	Philadelphia	100	7	8	1	7	0	0	0	0.759375
44	Philadelphia	100	8	11	1	7	2	1	0	0.714545
44	Philadelphia	100	9	0	0	0	0	0	0	nan
45	Philadelphia	100	0	10	1	4	3	2	0	0.735
45	Philadelphia	100	1	8	1	7	0	0	0	0.868125
45	Philadelphia	100	2	6	0	5	1	0	0	0.674167
45	Philadelphia	100	3	12	0	10	1	1	0	0.737917
45	Philadelphia	100	4	26	0	18	8	0	0	0.690577
45	Philadelphia	100	5	13	0	7	5	1	0	0.639615
45	Philadelphia	100	6	8	0	6	2	0	0	0.645
45	Philadelphia	100	7	10	1	5	4	0	0	0.598
45	Philadelphia	100	8	7	0	6	1	0	0	0.680714
45	Philadelphia	100	9	0	0	0	0	0	0	nan
46	Philadelphia	100	0	6	0	4	2	0	0	0.6825
46	Philadelphia	100	1	13	0	6	7	0	0	0.71
46	Philadelphia	100	2	6	0	4	2	0	0	0.535833
46	Philadelphia	100	3	14	0	9	5	0	0	0.695357

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
46	Philadelphia	100	4	18	0	11	7	0	0	0.709722
46	Philadelphia	100	5	16	1	11	4	0	0	0.739062
46	Philadelphia	100	6	4	0	3	1	0	0	0.66125
46	Philadelphia	100	7	12	0	10	2	0	0	0.695833
46	Philadelphia	100	8	11	0	4	6	1	0	0.546364
46	Philadelphia	100	9	0	0	0	0	0	0	nan
47	Philadelphia	100	0	6	0	5	1	0	0	0.76
47	Philadelphia	100	1	13	0	5	8	0	0	0.687308
47	Philadelphia	100	2	3	0	1	2	0	0	0.496667
47	Philadelphia	100	3	15	0	4	11	0	0	0.597
47	Philadelphia	100	4	19	0	8	11	0	0	0.568158
47	Philadelphia	100	5	17	0	10	7	0	0	0.661765
47	Philadelphia	100	6	11	0	6	5	0	0	0.818636
47	Philadelphia	100	7	12	0	7	4	1	0	0.647083
47	Philadelphia	100	8	4	0	2	2	0	0	0.79375
47	Philadelphia	100	9	0	0	0	0	0	0	nan
48	Philadelphia	100	0	8	0	1	7	0	0	0.595625
48	Philadelphia	100	1	13	0	3	10	0	0	0.486538
48	Philadelphia	100	2	7	0	2	5	0	0	0.592857
48	Philadelphia	100	3	11	0	4	7	0	0	0.688182
48	Philadelphia	100	4	18	0	5	13	0	0	0.586111
48	Philadelphia	100	5	12	0	6	6	0	0	0.814167
48	Philadelphia	100	6	10	0	3	7	0	0	0.589
48	Philadelphia	100	7	19	0	4	15	0	0	0.709474
48	Philadelphia	100	8	2	0	1	1	0	0	0.395
48	Philadelphia	100	9	0	0	0	0	0	0	nan
49	Philadelphia	100	0	4	0	1	3	0	0	0.55625
49	Philadelphia	100	1	11	0	0	11	0	0	0.670455
49	Philadelphia	100	2	9	0	1	8	0	0	0.615
49	Philadelphia	100	3	16	0	1	15	0	0	0.574062
49	Philadelphia	100	4	20	0	0	19	1	0	0.48125
49	Philadelphia	100	5	10	0	1	8	1	0	0.4445
49	Philadelphia	100	6	8	0	2	6	0	0	0.844375
49	Philadelphia	100	7	14	0	1	13	0	0	0.591071
49	Philadelphia	100	8	8	0	3	5	0	0	0.8025
49	Philadelphia	100	9	0	0	0	0	0	0	nan
50	Philadelphia	100	0	14	0	1	12	1	0	0.638214
50	Philadelphia	100	1	11	0	1	9	1	0	0.468182
50	Philadelphia	100	2	5	0	0	5	0	0	0.685
50	Philadelphia	100	3	15	0	0	12	3	0	0.603333
50	Philadelphia	100	4	22	0	3	17	2	0	0.6425
50	Philadelphia	100	5	7	0	0	7	0	0	0.415714
50	Philadelphia	100	6	10	0	0	10	0	0	0.528
50	Philadelphia	100	7	10	0	0	10	0	0	0.3935
50	Philadelphia	100	8	6	0	0	6	0	0	0.505
50	Philadelphia	100	9	0	0	0	0	0	0	nan
51	Philadelphia	100	0	7	0	0	7	0	0	0.362857
51	Philadelphia	100	1	6	0	1	5	0	0	0.573333
51	Philadelphia	100	2	7	0	0	5	2	0	0.557857
51	Philadelphia	100	3	16	0	0	14	2	0	0.629687
51	Philadelphia	100	4	21	0	0	20	1	0	0.399524
51	Philadelphia	100	5	15	0	0	15	0	0	0.544
51	Philadelphia	100	6	11	0	0	11	0	0	0.514091
51	Philadelphia	100	7	9	0	0	7	2	0	0.477778
51	Philadelphia	100	8	8	0	0	8	0	0	0.680625
51	Philadelphia	100	9	0	0	0	0	0	0	nan
52	Philadelphia	100	0	12	0	0	11	1	0	0.565
52	Philadelphia	100	1	12	0	0	12	0	0	0.34

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
52	Philadelphia	100	2	9	0	0	8	1	0	0.408889
52	Philadelphia	100	3	10	0	0	8	2	0	0.481
52	Philadelphia	100	4	14	0	0	13	1	0	0.395714
52	Philadelphia	100	5	13	0	0	12	1	0	0.417692
52	Philadelphia	100	6	7	0	0	6	1	0	0.531429
52	Philadelphia	100	7	14	0	0	12	2	0	0.604643
52	Philadelphia	100	8	9	0	0	8	1	0	0.602222
52	Philadelphia	100	9	0	0	0	0	0	0	nan
53	Philadelphia	100	0	7	0	0	7	0	0	0.580714
53	Philadelphia	100	1	18	0	0	15	3	0	0.339722
53	Philadelphia	100	2	14	0	0	12	2	0	0.493214
53	Philadelphia	100	3	15	0	0	12	3	0	0.401
53	Philadelphia	100	4	11	0	0	10	1	0	0.525909
53	Philadelphia	100	5	7	0	0	5	2	0	0.347857
53	Philadelphia	100	6	10	0	0	9	1	0	0.421
53	Philadelphia	100	7	10	0	0	9	1	0	0.453
53	Philadelphia	100	8	8	0	0	6	2	0	0.538125
53	Philadelphia	100	9	0	0	0	0	0	0	nan
54	Philadelphia	100	0	13	0	0	9	4	0	0.433077
54	Philadelphia	100	1	17	0	0	15	2	0	0.292353
54	Philadelphia	100	2	8	0	0	7	1	0	0.400625
54	Philadelphia	100	3	14	0	0	10	4	0	0.398214
54	Philadelphia	100	4	15	0	0	12	3	0	0.512333
54	Philadelphia	100	5	12	0	0	8	4	0	0.405417
54	Philadelphia	100	6	5	0	0	4	1	0	0.362
54	Philadelphia	100	7	12	0	0	11	1	0	0.430417
54	Philadelphia	100	8	4	0	0	3	1	0	0.62875
54	Philadelphia	100	9	0	0	0	0	0	0	nan
55	Philadelphia	100	0	7	0	0	6	1	0	0.383571
55	Philadelphia	100	1	9	0	0	6	3	0	0.345556
55	Philadelphia	100	2	11	0	0	7	4	0	0.437273
55	Philadelphia	100	3	15	0	0	10	5	0	0.365
55	Philadelphia	100	4	22	0	0	16	6	0	0.462727
55	Philadelphia	100	5	10	0	0	8	2	0	0.229
55	Philadelphia	100	6	8	0	0	6	2	0	0.34625
55	Philadelphia	100	7	13	0	0	7	6	0	0.506923
55	Philadelphia	100	8	5	0	0	4	1	0	0.303
55	Philadelphia	100	9	0	0	0	0	0	0	nan
56	Philadelphia	100	0	5	0	0	2	3	0	0.61
56	Philadelphia	100	1	16	0	0	6	10	0	0.500312
56	Philadelphia	100	2	7	0	0	4	3	0	0.32
56	Philadelphia	100	3	12	0	0	6	6	0	0.399167
56	Philadelphia	100	4	15	0	0	8	7	0	0.405667
56	Philadelphia	100	5	9	0	0	7	2	0	0.249444
56	Philadelphia	100	6	8	0	0	5	3	0	0.301875
56	Philadelphia	100	7	19	0	0	12	7	0	0.379211
56	Philadelphia	100	8	9	0	0	8	1	0	0.265
56	Philadelphia	100	9	0	0	0	0	0	0	nan
57	Philadelphia	100	0	6	0	0	3	3	0	0.515833
57	Philadelphia	100	1	12	0	0	9	3	0	0.194583
57	Philadelphia	100	2	3	0	0	0	3	0	0.505
57	Philadelphia	100	3	13	0	0	7	6	0	0.435769
57	Philadelphia	100	4	18	0	0	7	11	0	0.499722
57	Philadelphia	100	5	15	0	0	8	7	0	0.333333
57	Philadelphia	100	6	9	0	0	3	6	0	0.423889
57	Philadelphia	100	7	14	0	0	8	6	0	0.333929
57	Philadelphia	100	8	10	0	0	4	6	0	0.337
57	Philadelphia	100	9	0	0	0	0	0	0	nan

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
58	Philadelphia	100	0	8	0	0	4	4	0	0.40375
58	Philadelphia	100	1	17	0	0	8	9	0	0.352353
58	Philadelphia	100	2	8	0	0	2	6	0	0.344375
58	Philadelphia	100	3	8	0	0	4	4	0	0.41875
58	Philadelphia	100	4	22	0	0	9	13	0	0.407727
58	Philadelphia	100	5	9	0	0	4	5	0	0.319444
58	Philadelphia	100	6	6	0	0	2	4	0	0.516667
58	Philadelphia	100	7	8	0	0	4	4	0	0.45
58	Philadelphia	100	8	14	0	0	6	8	0	0.385357
58	Philadelphia	100	9	0	0	0	0	0	0	nan
59	Philadelphia	100	0	7	0	0	3	4	0	0.458571
59	Philadelphia	100	1	11	0	0	3	8	0	0.443636
59	Philadelphia	100	2	8	0	0	2	6	0	0.44625
59	Philadelphia	100	3	17	0	0	6	11	0	0.458529
59	Philadelphia	100	4	20	0	0	6	14	0	0.44
59	Philadelphia	100	5	9	0	0	6	3	0	0.255556
59	Philadelphia	100	6	7	0	0	3	4	0	0.345714
59	Philadelphia	100	7	15	0	0	3	12	0	0.428667
59	Philadelphia	100	8	6	0	0	2	4	0	0.244167
59	Philadelphia	100	9	0	0	0	0	0	0	nan
60	Philadelphia	100	0	7	0	0	1	6	0	0.317143
60	Philadelphia	100	1	10	0	0	4	6	0	0.381
60	Philadelphia	100	2	7	0	0	2	5	0	0.42
60	Philadelphia	100	3	14	0	0	4	10	0	0.414643
60	Philadelphia	100	4	19	0	0	3	16	0	0.460526
60	Philadelphia	100	5	6	0	0	2	4	0	0.320833
60	Philadelphia	100	6	11	0	0	2	9	0	0.566364
60	Philadelphia	100	7	14	0	0	6	8	0	0.351071
60	Philadelphia	100	8	12	0	0	4	8	0	0.54125
60	Philadelphia	100	9	0	0	0	0	0	0	nan
61	Philadelphia	100	0	9	0	0	3	6	0	0.298889
61	Philadelphia	100	1	16	0	0	4	12	0	0.426562
61	Philadelphia	100	2	11	0	0	2	9	0	0.461818
61	Philadelphia	100	3	16	0	0	1	15	0	0.445
61	Philadelphia	100	4	14	0	0	4	10	0	0.4475
61	Philadelphia	100	5	13	0	0	3	10	0	0.247692
61	Philadelphia	100	6	5	0	0	0	5	0	0.99
61	Philadelphia	100	7	8	0	0	2	6	0	0.57875
61	Philadelphia	100	8	8	0	0	1	7	0	0.59625
61	Philadelphia	100	9	0	0	0	0	0	0	nan
62	Philadelphia	100	0	7	0	0	2	5	0	0.228571
62	Philadelphia	100	1	16	0	0	4	12	0	0.448437
62	Philadelphia	100	2	7	0	0	2	5	0	0.292143
62	Philadelphia	100	3	13	0	0	1	12	0	0.501538
62	Philadelphia	100	4	22	0	0	4	18	0	0.562955
62	Philadelphia	100	5	9	0	0	1	8	0	0.436667
62	Philadelphia	100	6	9	0	0	2	7	0	0.561667
62	Philadelphia	100	7	8	0	0	2	6	0	0.339375
62	Philadelphia	100	8	9	0	0	0	9	0	0.768889
62	Philadelphia	100	9	0	0	0	0	0	0	nan
63	Philadelphia	100	0	6	0	0	1	5	0	0.6775
63	Philadelphia	100	1	13	0	0	2	11	0	0.387308
63	Philadelphia	100	2	7	0	0	0	7	0	0.390714
63	Philadelphia	100	3	16	0	0	2	14	0	0.520312
63	Philadelphia	100	4	23	0	0	6	17	0	0.54087
63	Philadelphia	100	5	15	0	0	0	15	0	0.668
63	Philadelphia	100	6	4	0	0	0	4	0	0.22875
63	Philadelphia	100	7	12	0	0	3	9	0	0.413333

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
63	Philadelphia	100	8	4	0	0	0	4	0	0.62875
63	Philadelphia	100	9	0	0	0	0	0	0	nan
64	Philadelphia	100	0	8	0	0	1	7	0	0.50875
64	Philadelphia	100	1	9	0	0	0	9	0	0.712778
64	Philadelphia	100	2	7	0	0	1	6	0	0.390714
64	Philadelphia	100	3	10	0	0	1	9	0	0.5305
64	Philadelphia	100	4	19	0	0	0	19	0	0.493947
64	Philadelphia	100	5	12	0	0	1	11	0	0.594583
64	Philadelphia	100	6	8	0	0	2	6	0	0.30625
64	Philadelphia	100	7	20	0	0	4	16	0	0.57725
64	Philadelphia	100	8	7	0	0	1	6	0	0.668571
64	Philadelphia	100	9	0	0	0	0	0	0	nan
65	Philadelphia	100	0	6	0	0	0	6	0	0.6975
65	Philadelphia	100	1	13	0	1	0	12	0	0.356538
65	Philadelphia	100	2	8	0	0	1	7	0	0.600625
65	Philadelphia	100	3	11	0	0	1	10	0	0.579091
65	Philadelphia	100	4	18	0	0	3	15	0	0.541667
65	Philadelphia	100	5	19	0	0	2	17	0	0.673947
65	Philadelphia	100	6	5	0	0	0	5	0	0.631
65	Philadelphia	100	7	14	0	0	1	13	0	0.600714
65	Philadelphia	100	8	6	0	0	0	6	0	0.361667
65	Philadelphia	100	9	0	0	0	0	0	0	nan
66	Philadelphia	100	0	10	0	0	2	8	0	0.4475
66	Philadelphia	100	1	6	0	0	0	6	0	0.608333
66	Philadelphia	100	2	6	0	1	1	4	0	0.691667
66	Philadelphia	100	3	7	0	0	0	7	0	0.485
66	Philadelphia	100	4	13	0	0	0	13	0	0.679615
66	Philadelphia	100	5	18	0	0	2	16	0	0.571111
66	Philadelphia	100	6	15	0	0	1	14	0	0.657667
66	Philadelphia	100	7	16	0	0	1	15	0	0.4925
66	Philadelphia	100	8	9	0	0	0	9	0	0.701667
66	Philadelphia	100	9	0	0	0	0	0	0	nan
67	Philadelphia	100	0	7	0	0	0	7	0	0.510714
67	Philadelphia	100	1	18	0	0	0	18	0	0.621944
67	Philadelphia	100	2	6	0	0	0	6	0	0.8875
67	Philadelphia	100	3	7	0	0	0	7	0	0.686429
67	Philadelphia	100	4	19	0	0	1	18	0	0.677105
67	Philadelphia	100	5	8	0	0	0	8	0	0.764375
67	Philadelphia	100	6	6	0	0	1	5	0	0.431667
67	Philadelphia	100	7	20	0	1	3	16	0	0.51825
67	Philadelphia	100	8	9	0	0	0	9	0	0.501667
67	Philadelphia	100	9	0	0	0	0	0	0	nan
68	Philadelphia	100	0	7	0	0	0	7	0	0.615
68	Philadelphia	100	1	11	0	0	0	11	0	0.753182
68	Philadelphia	100	2	11	0	0	1	10	0	0.513636
68	Philadelphia	100	3	10	0	1	0	9	0	0.634
68	Philadelphia	100	4	21	0	0	1	20	0	0.641429
68	Philadelphia	100	5	12	0	0	1	11	0	0.6925
68	Philadelphia	100	6	6	0	0	0	6	0	0.661667
68	Philadelphia	100	7	14	0	0	0	14	0	0.614643
68	Philadelphia	100	8	8	0	0	0	8	0	0.615625
68	Philadelphia	100	9	0	0	0	0	0	0	nan
69	Philadelphia	100	0	2	0	0	0	2	0	0.6875
69	Philadelphia	100	1	12	0	0	1	11	0	0.6775
69	Philadelphia	100	2	5	0	0	0	5	0	0.687
69	Philadelphia	100	3	12	0	0	0	12	0	0.549583
69	Philadelphia	100	4	25	0	0	0	25	0	0.7732
69	Philadelphia	100	5	11	0	0	1	10	0	0.675

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
69	Philadelphia	100	6	8	0	0	1	7	0	0.439375
69	Philadelphia	100	7	12	0	0	0	12	0	0.785
69	Philadelphia	100	8	13	0	0	0	13	0	0.552308
69	Philadelphia	100	9	0	0	0	0	0	0	nan
70	Philadelphia	100	0	11	1	0	1	9	0	0.64
70	Philadelphia	100	1	11	0	0	1	10	0	0.657273
70	Philadelphia	100	2	7	0	0	0	7	0	0.730714
70	Philadelphia	100	3	17	1	0	0	16	0	0.619706
70	Philadelphia	100	4	21	0	0	0	21	0	0.645
70	Philadelphia	100	5	8	0	0	0	8	0	0.80625
70	Philadelphia	100	6	8	0	0	0	8	0	0.596875
70	Philadelphia	100	7	12	0	0	1	11	0	0.789167
70	Philadelphia	100	8	5	0	0	0	5	0	0.941
70	Philadelphia	100	9	0	0	0	0	0	0	nan
71	Philadelphia	100	0	5	0	0	0	5	0	0.95
71	Philadelphia	100	1	8	1	0	0	7	0	0.776875
71	Philadelphia	100	2	6	0	0	0	6	0	0.771667
71	Philadelphia	100	3	13	0	0	0	13	0	0.692308
71	Philadelphia	100	4	23	0	0	1	22	0	0.706957
71	Philadelphia	100	5	10	0	0	0	10	0	0.6025
71	Philadelphia	100	6	11	2	0	0	9	0	0.735455
71	Philadelphia	100	7	14	1	0	1	12	0	0.653929
71	Philadelphia	100	8	10	0	1	0	9	0	0.7075
71	Philadelphia	100	9	0	0	0	0	0	0	nan
72	Philadelphia	100	0	6	1	0	0	5	0	0.830833
72	Philadelphia	100	1	11	0	0	0	11	0	0.730909
72	Philadelphia	100	2	8	1	0	0	7	0	0.696875
72	Philadelphia	100	3	14	0	0	0	14	0	0.759286
72	Philadelphia	100	4	24	0	0	0	24	0	0.691458
72	Philadelphia	100	5	14	3	1	0	10	0	0.738214
72	Philadelphia	100	6	5	1	0	0	4	0	0.76
72	Philadelphia	100	7	11	1	0	1	9	0	0.842727
72	Philadelphia	100	8	7	0	0	0	7	0	0.615
72	Philadelphia	100	9	0	0	0	0	0	0	nan
73	Philadelphia	100	0	5	1	0	0	4	0	0.67
73	Philadelphia	100	1	12	0	0	0	12	0	0.709167
73	Philadelphia	100	2	9	1	0	0	8	0	0.78
73	Philadelphia	100	3	16	1	0	1	14	0	0.770938
73	Philadelphia	100	4	21	4	1	0	16	0	0.793571
73	Philadelphia	100	5	10	3	0	0	7	0	0.7675
73	Philadelphia	100	6	8	0	0	0	8	0	0.85375
73	Philadelphia	100	7	10	2	0	0	8	0	0.723
73	Philadelphia	100	8	9	2	1	0	6	0	0.679444
73	Philadelphia	100	9	0	0	0	0	0	0	nan
74	Philadelphia	100	0	8	1	0	0	7	0	0.8225
74	Philadelphia	100	1	17	2	1	0	14	0	0.78
74	Philadelphia	100	2	3	0	0	0	3	0	0.42
74	Philadelphia	100	3	14	4	0	1	9	0	0.7575
74	Philadelphia	100	4	25	7	1	0	17	0	0.7784
74	Philadelphia	100	5	5	2	1	0	2	0	0.927
74	Philadelphia	100	6	6	1	0	0	5	0	0.798333
74	Philadelphia	100	7	14	4	0	0	10	0	0.738214
74	Philadelphia	100	8	8	1	0	0	7	0	0.854375
74	Philadelphia	100	9	0	0	0	0	0	0	nan
75	Philadelphia	100	0	14	4	1	0	9	0	0.769643
75	Philadelphia	100	1	10	3	0	0	7	0	0.9075
75	Philadelphia	100	2	5	0	1	0	4	0	0.572
75	Philadelphia	100	3	15	6	1	0	8	0	0.905667

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
75	Philadelphia	100	4	18	8	0	1	9	0	0.870833
75	Philadelphia	100	5	13	5	0	0	8	0	0.775
75	Philadelphia	100	6	8	1	0	0	7	0	0.593125
75	Philadelphia	100	7	12	2	1	0	9	0	0.779583
75	Philadelphia	100	8	5	1	1	0	3	0	0.688
75	Philadelphia	100	9	0	0	0	0	0	0	nan
76	Philadelphia	100	0	9	4	0	0	5	0	0.707778
76	Philadelphia	100	1	5	3	0	0	2	0	0.753
76	Philadelphia	100	2	7	3	2	0	2	0	0.887143
76	Philadelphia	100	3	13	5	1	0	7	0	0.908846
76	Philadelphia	100	4	25	9	3	0	13	0	0.8514
76	Philadelphia	100	5	12	3	1	1	7	0	0.797917
76	Philadelphia	100	6	11	3	0	1	7	0	0.628636
76	Philadelphia	100	7	11	4	1	0	6	0	0.771364
76	Philadelphia	100	8	7	3	0	0	4	0	0.962143
76	Philadelphia	100	9	0	0	0	0	0	0	nan
77	Philadelphia	100	0	9	1	4	0	4	0	0.747222
77	Philadelphia	100	1	9	5	0	0	4	0	0.891667
77	Philadelphia	100	2	8	4	0	0	4	0	0.814375
77	Philadelphia	100	3	13	6	1	0	6	0	0.849231
77	Philadelphia	100	4	24	11	3	2	8	0	0.834792
77	Philadelphia	100	5	10	5	1	0	4	0	0.798
77	Philadelphia	100	6	5	2	0	0	3	0	0.695
77	Philadelphia	100	7	14	10	1	0	3	0	0.902143
77	Philadelphia	100	8	8	0	2	0	6	0	0.74625
77	Philadelphia	100	9	0	0	0	0	0	0	nan
78	Philadelphia	100	0	4	2	1	0	1	0	0.86
78	Philadelphia	100	1	14	10	2	0	2	0	0.944643
78	Philadelphia	100	2	7	4	0	0	3	0	0.946429
78	Philadelphia	100	3	16	8	3	0	5	0	0.845313
78	Philadelphia	100	4	14	4	5	0	5	0	0.812857
78	Philadelphia	100	5	15	9	1	1	4	0	0.748667
78	Philadelphia	100	6	7	4	2	0	1	0	0.911429
78	Philadelphia	100	7	16	3	5	2	6	0	0.769375
78	Philadelphia	100	8	7	5	0	0	2	0	0.743571
78	Philadelphia	100	9	0	0	0	0	0	0	nan
79	Philadelphia	100	0	7	3	2	1	1	0	0.824286
79	Philadelphia	100	1	19	11	6	0	2	0	0.905526
79	Philadelphia	100	2	10	5	1	1	3	0	0.812
79	Philadelphia	100	3	11	7	2	0	2	0	0.835
79	Philadelphia	100	4	13	6	5	1	1	0	0.798846
79	Philadelphia	100	5	10	5	2	1	2	0	0.6885
79	Philadelphia	100	6	9	7	0	0	2	0	0.916667
79	Philadelphia	100	7	14	4	7	0	3	0	0.866786
79	Philadelphia	100	8	7	3	3	0	1	0	0.831429
79	Philadelphia	100	9	0	0	0	0	0	0	nan
80	Philadelphia	100	0	10	3	4	1	2	0	0.796
80	Philadelphia	100	1	10	5	3	1	1	0	0.8585
80	Philadelphia	100	2	8	6	2	0	0	0	0.87625
80	Philadelphia	100	3	14	9	3	0	2	0	0.895
80	Philadelphia	100	4	21	8	8	3	2	0	0.836667
80	Philadelphia	100	5	14	3	10	1	0	0	0.858214
80	Philadelphia	100	6	5	2	3	0	0	0	0.778
80	Philadelphia	100	7	12	4	5	0	3	0	0.777083
80	Philadelphia	100	8	6	3	1	1	1	0	0.8325
80	Philadelphia	100	9	0	0	0	0	0	0	nan
81	Philadelphia	100	0	5	0	4	0	1	0	0.764
81	Philadelphia	100	1	13	4	8	1	0	0	0.931923

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
81	Philadelphia	100	2	10	0	8	1	1	0	0.6565
81	Philadelphia	100	3	12	1	9	2	0	0	0.83
81	Philadelphia	100	4	23	5	13	4	1	0	0.858043
81	Philadelphia	100	5	14	5	5	3	1	0	0.816071
81	Philadelphia	100	6	7	1	5	1	0	0	0.849286
81	Philadelphia	100	7	14	4	9	0	1	0	0.906429
81	Philadelphia	100	8	2	1	0	1	0	0	0.6675
81	Philadelphia	100	9	0	0	0	0	0	0	nan
82	Philadelphia	100	0	12	0	8	4	0	0	0.8325
82	Philadelphia	100	1	8	3	4	1	0	0	0.81875
82	Philadelphia	100	2	9	1	7	1	0	0	0.868333
82	Philadelphia	100	3	12	3	8	1	0	0	0.81375
82	Philadelphia	100	4	11	1	10	0	0	0	0.869091
82	Philadelphia	100	5	15	0	10	4	1	0	0.782333
82	Philadelphia	100	6	7	1	3	3	0	0	0.84
82	Philadelphia	100	7	17	2	9	3	3	0	0.795294
82	Philadelphia	100	8	9	1	7	1	0	0	0.891667
82	Philadelphia	100	9	0	0	0	0	0	0	nan
83	Philadelphia	100	0	5	0	5	0	0	0	0.912
83	Philadelphia	100	1	15	0	13	2	0	0	0.854
83	Philadelphia	100	2	9	0	8	1	0	0	0.765556
83	Philadelphia	100	3	9	0	6	3	0	0	0.782778
83	Philadelphia	100	4	18	1	10	6	1	0	0.772778
83	Philadelphia	100	5	16	0	8	6	2	0	0.806563
83	Philadelphia	100	6	9	1	4	4	0	0	0.817222
83	Philadelphia	100	7	14	3	9	2	0	0	0.867857
83	Philadelphia	100	8	5	1	4	0	0	0	0.832
83	Philadelphia	100	9	0	0	0	0	0	0	nan
84	Philadelphia	100	0	5	0	3	2	0	0	0.955
84	Philadelphia	100	1	12	0	8	4	0	0	0.881667
84	Philadelphia	100	2	5	0	1	4	0	0	0.596
84	Philadelphia	100	3	12	0	7	3	2	0	0.717917
84	Philadelphia	100	4	24	0	13	9	2	0	0.755833
84	Philadelphia	100	5	12	0	9	3	0	0	0.875417
84	Philadelphia	100	6	7	0	6	1	0	0	0.862857
84	Philadelphia	100	7	16	1	10	5	0	0	0.782813
84	Philadelphia	100	8	7	0	4	3	0	0	0.905714
84	Philadelphia	100	9	0	0	0	0	0	0	nan
85	Philadelphia	100	0	3	0	1	1	1	0	0.59
85	Philadelphia	100	1	18	0	8	9	1	0	0.830278
85	Philadelphia	100	2	7	0	4	3	0	0	0.859286
85	Philadelphia	100	3	14	0	4	10	0	0	0.7925
85	Philadelphia	100	4	19	0	9	9	1	0	0.727632
85	Philadelphia	100	5	10	0	3	7	0	0	0.8965
85	Philadelphia	100	6	5	0	2	3	0	0	0.677
85	Philadelphia	100	7	20	0	10	10	0	0	0.8245
85	Philadelphia	100	8	4	0	1	3	0	0	0.56125
85	Philadelphia	100	9	0	0	0	0	0	0	nan
86	Philadelphia	100	0	5	0	1	4	0	0	0.8
86	Philadelphia	100	1	9	0	3	6	0	0	0.754444
86	Philadelphia	100	2	6	0	2	2	2	0	0.614167
86	Philadelphia	100	3	17	0	2	14	1	0	0.717647
86	Philadelphia	100	4	22	0	3	19	0	0	0.833182
86	Philadelphia	100	5	6	0	3	3	0	0	0.8975
86	Philadelphia	100	6	11	0	1	10	0	0	0.631364
86	Philadelphia	100	7	16	0	4	12	0	0	0.796563
86	Philadelphia	100	8	8	0	2	6	0	0	0.741875
86	Philadelphia	100	9	0	0	0	0	0	0	nan

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
87	Philadelphia	100	0	7	0	1	6	0	0	0.687857
87	Philadelphia	100	1	15	0	1	13	1	0	0.701
87	Philadelphia	100	2	4	0	0	3	1	0	0.81875
87	Philadelphia	100	3	19	0	2	14	3	0	0.701579
87	Philadelphia	100	4	16	0	1	15	0	0	0.7025
87	Philadelphia	100	5	15	0	3	12	0	0	0.818
87	Philadelphia	100	6	5	0	1	4	0	0	0.674
87	Philadelphia	100	7	13	0	3	10	0	0	0.773077
87	Philadelphia	100	8	6	0	1	5	0	0	0.586667
87	Philadelphia	100	9	0	0	0	0	0	0	nan
88	Philadelphia	100	0	8	0	0	6	2	0	0.600625
88	Philadelphia	100	1	15	0	3	11	1	0	0.743667
88	Philadelphia	100	2	10	0	1	8	1	0	0.688
88	Philadelphia	100	3	12	0	1	9	2	0	0.687083
88	Philadelphia	100	4	15	0	1	12	2	0	0.630333
88	Philadelphia	100	5	11	0	0	11	0	0	0.724545
88	Philadelphia	100	6	8	0	1	7	0	0	0.588125
88	Philadelphia	100	7	17	0	1	16	0	0	0.707941
88	Philadelphia	100	8	4	0	0	3	1	0	0.855
88	Philadelphia	100	9	0	0	0	0	0	0	nan
89	Philadelphia	100	0	6	0	0	5	1	0	0.603333
89	Philadelphia	100	1	10	0	0	9	1	0	0.716
89	Philadelphia	100	2	6	0	0	6	0	0	0.49
89	Philadelphia	100	3	11	0	0	11	0	0	0.535
89	Philadelphia	100	4	21	0	0	18	3	0	0.645238
89	Philadelphia	100	5	15	0	0	13	2	0	0.651333
89	Philadelphia	100	6	5	0	0	5	0	0	0.691
89	Philadelphia	100	7	20	0	0	18	2	0	0.72375
89	Philadelphia	100	8	6	0	0	5	1	0	0.686667
89	Philadelphia	100	9	0	0	0	0	0	0	nan
90	Philadelphia	100	0	7	0	0	5	2	0	0.666429
90	Philadelphia	100	1	8	0	0	7	1	0	0.6025
90	Philadelphia	100	2	7	0	0	6	1	0	0.456429
90	Philadelphia	100	3	10	0	0	9	1	0	0.6095
90	Philadelphia	100	4	26	0	0	22	4	0	0.628462
90	Philadelphia	100	5	13	0	0	11	2	0	0.673077
90	Philadelphia	100	6	12	0	0	12	0	0	0.485417
90	Philadelphia	100	7	13	0	0	11	2	0	0.632692
90	Philadelphia	100	8	4	0	0	1	3	0	0.8275
90	Philadelphia	100	9	0	0	0	0	0	0	nan
91	Philadelphia	100	0	8	0	0	7	1	0	0.695
91	Philadelphia	100	1	11	0	0	7	4	0	0.615
91	Philadelphia	100	2	2	0	0	1	1	0	0.5875
91	Philadelphia	100	3	15	0	0	14	1	0	0.556333
91	Philadelphia	100	4	21	0	0	15	6	0	0.627381
91	Philadelphia	100	5	14	0	0	9	5	0	0.516071
91	Philadelphia	100	6	9	0	0	8	1	0	0.565556
91	Philadelphia	100	7	14	0	0	13	1	0	0.536786
91	Philadelphia	100	8	6	0	0	5	1	0	0.526667
91	Philadelphia	100	9	0	0	0	0	0	0	nan
92	Philadelphia	100	0	7	0	0	5	2	0	0.570714
92	Philadelphia	100	1	12	0	0	12	0	0	0.532917
92	Philadelphia	100	2	8	0	0	6	2	0	0.534375
92	Philadelphia	100	3	13	0	0	10	3	0	0.513846
92	Philadelphia	100	4	15	0	0	8	7	0	0.573333
92	Philadelphia	100	5	12	0	0	8	4	0	0.63625
92	Philadelphia	100	6	5	0	0	5	0	0	0.566
92	Philadelphia	100	7	18	0	0	8	10	0	0.585278

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
92	Philadelphia	100	8	10	0	0	8	2	0	0.456
92	Philadelphia	100	9	0	0	0	0	0	0	nan
93	Philadelphia	100	0	11	0	0	6	5	0	0.534091
93	Philadelphia	100	1	6	0	0	3	3	0	0.543333
93	Philadelphia	100	2	7	0	0	4	3	0	0.585714
93	Philadelphia	100	3	11	0	0	7	4	0	0.588182
93	Philadelphia	100	4	22	0	0	14	8	0	0.516364
93	Philadelphia	100	5	12	0	0	9	3	0	0.500833
93	Philadelphia	100	6	7	0	0	3	4	0	0.611429
93	Philadelphia	100	7	14	0	0	7	7	0	0.5075
93	Philadelphia	100	8	10	0	0	4	6	0	0.557
93	Philadelphia	100	9	0	0	0	0	0	0	nan
94	Philadelphia	100	0	8	0	0	3	5	0	0.638125
94	Philadelphia	100	1	12	0	0	5	7	0	0.567917
94	Philadelphia	100	2	11	0	0	3	8	0	0.476364
94	Philadelphia	100	3	10	0	0	5	5	0	0.546
94	Philadelphia	100	4	21	0	0	9	12	0	0.518095
94	Philadelphia	100	5	11	0	0	7	4	0	0.565
94	Philadelphia	100	6	7	0	0	3	4	0	0.415
94	Philadelphia	100	7	10	0	0	5	5	0	0.56
94	Philadelphia	100	8	10	0	0	4	6	0	0.553
94	Philadelphia	100	9	0	0	0	0	0	0	nan
95	Philadelphia	100	0	5	0	0	0	5	0	0.569
95	Philadelphia	100	1	8	0	0	1	7	0	0.6575
95	Philadelphia	100	2	5	0	0	0	5	0	0.556
95	Philadelphia	100	3	8	0	0	1	7	0	0.574375
95	Philadelphia	100	4	23	0	0	9	14	0	0.510652
95	Philadelphia	100	5	14	0	0	5	9	0	0.510714
95	Philadelphia	100	6	16	0	0	3	13	0	0.611562
95	Philadelphia	100	7	14	0	0	3	11	0	0.452143
95	Philadelphia	100	8	7	0	0	1	6	0	0.657143
95	Philadelphia	100	9	0	0	0	0	0	0	nan
96	Philadelphia	100	0	6	0	0	0	6	0	0.7
96	Philadelphia	100	1	18	0	0	5	13	0	0.485556
96	Philadelphia	100	2	4	0	0	0	4	0	0.86625
96	Philadelphia	100	3	13	0	0	4	9	0	0.511154
96	Philadelphia	100	4	20	0	0	2	18	0	0.56825
96	Philadelphia	100	5	17	0	0	1	16	0	0.605882
96	Philadelphia	100	6	4	0	0	0	4	0	0.49375
96	Philadelphia	100	7	11	0	0	1	10	0	0.567727
96	Philadelphia	100	8	7	0	0	1	6	0	0.681429
96	Philadelphia	100	9	0	0	0	0	0	0	nan
97	Philadelphia	100	0	2	0	0	0	2	0	0.71
97	Philadelphia	100	1	20	0	0	4	16	0	0.5715
97	Philadelphia	100	2	7	0	0	0	7	0	0.77
97	Philadelphia	100	3	10	0	0	1	9	0	0.62
97	Philadelphia	100	4	18	0	0	3	15	0	0.609444
97	Philadelphia	100	5	14	0	0	2	12	0	0.662143
97	Philadelphia	100	6	11	0	0	0	11	0	0.509091
97	Philadelphia	100	7	12	0	0	2	10	0	0.579167
97	Philadelphia	100	8	6	0	0	1	5	0	0.565833
97	Philadelphia	100	9	0	0	0	0	0	0	nan
98	Philadelphia	100	0	5	0	0	1	4	0	0.683
98	Philadelphia	100	1	12	0	0	1	11	0	0.625833
98	Philadelphia	100	2	9	0	0	0	9	0	0.789444
98	Philadelphia	100	3	12	0	0	3	9	0	0.577917
98	Philadelphia	100	4	19	0	0	3	16	0	0.608421
98	Philadelphia	100	5	13	0	0	2	11	0	0.564231

Date	City	City Population	Chunk Number	Chunk Populatio	Susceptible	Latent	Infectious	Recovered	Immune	Average Health
98	Philadelphia	100	6	10	0	0	1	9	0	0.5345
98	Philadelphia	100	7	13	0	0	0	13	0	0.717692
98	Philadelphia	100	8	7	0	0	0	7	0	0.704286
98	Philadelphia	100	9	0	0	0	0	0	0	nan
99	Philadelphia	100	0	6	0	0	0	6	0	0.808333
99	Philadelphia	100	1	10	0	0	2	8	0	0.672
99	Philadelphia	100	2	4	0	0	1	3	0	0.4825
99	Philadelphia	100	3	13	0	0	1	12	0	0.595
99	Philadelphia	100	4	24	0	0	1	23	0	0.696042
99	Philadelphia	100	5	12	0	0	0	12	0	0.596667
99	Philadelphia	100	6	4	0	0	1	3	0	0.66625
99	Philadelphia	100	7	16	0	0	2	14	0	0.573438
99	Philadelphia	100	8	11	0	0	0	11	0	0.856818
99	Philadelphia	100	9	0	0	0	0	0	0	nan