Comparison of Disease Propagation Models that Predict the Speed of an Outbreak

New Mexico

Supercomputing Challenge

Final Report

April 2, 2014

Team 5

Albuquerque Academy

Team Members:

Marcus Dominguez-Kuhne, Trevor Kann, Josh Konopka, David Kohler, and Hisham Temmar

Teacher:

Mr. Jim Mims

Project Mentor:

Ms. Marie Kuhne

Table of Contents

Executive Summary

In this project, an Agent based model was programmed in Java to simulate and compare the propagation of disease algorithms that predict the speed of an outbreak. This model creates thousands of Agents that independently move from location to location using a one-year travel schedule based upon Agent roles. Agents get sick when a disease algorithm changes its state-of-health during a simulation. The three H1N1 algorithms created for this project are the Rule Base Algorithm (RBA), Math Based Algorithm (MBA) and Random Number Generator (RNG). During a simulation run, Agents' state-of-health (susceptible, infected not contagious, infected contagious and immune) are graphically displayed by the Java Graphical User Interface (GUI) and saved for analysis and comparison.   A Latin Hypercube sensitivity analysis was performed on each algorithm tuning parameters to determine which produced results similar to the H1N1 truth data.

Problem Statement and Introduction

It is planned to compare disease propagation algorithms that can predict the speed of an outbreak with a computer simulation and sensitivity analysis. The data collected can then be applied to future epidemics to prepare and prevent a potential outbreak in a real life situation. Obtaining prediction results from several algorithms provides comparative analysis with truth data which is more lifelike. An Agent based model is combined with a Stage location running over a 365 day simulation using a population profile. A Stage is defined as a location where Agents reside during a 24 hour period. This project will be programmed using Java to model an Agent based simulation infrastructure and three H1N1 virus propagation algorithms. The algorithms of interest are the Rule Based Algorithm (RBA), Math Based Algorithm (MBA) and the Random Number Generator algorithm (RNG) which will propagate the H1N1 virus through an Agent population.

The results will be compared with the fludb.org truth data of the H1N1 in 2009. This is a continuation of last year's Supercomputing Challenge project, which was implemented in C++ and OpenGL that modeled Agents moving randomly in four areas. This year the problem solution has greatly matured utilizing numerous Agent roles and Stage locations, where each Agent's travel habits are based off of a life-like schedule. Furthermore, the three algorithms mentioned are compared and a sensitivity analysis is performed to better understand them.

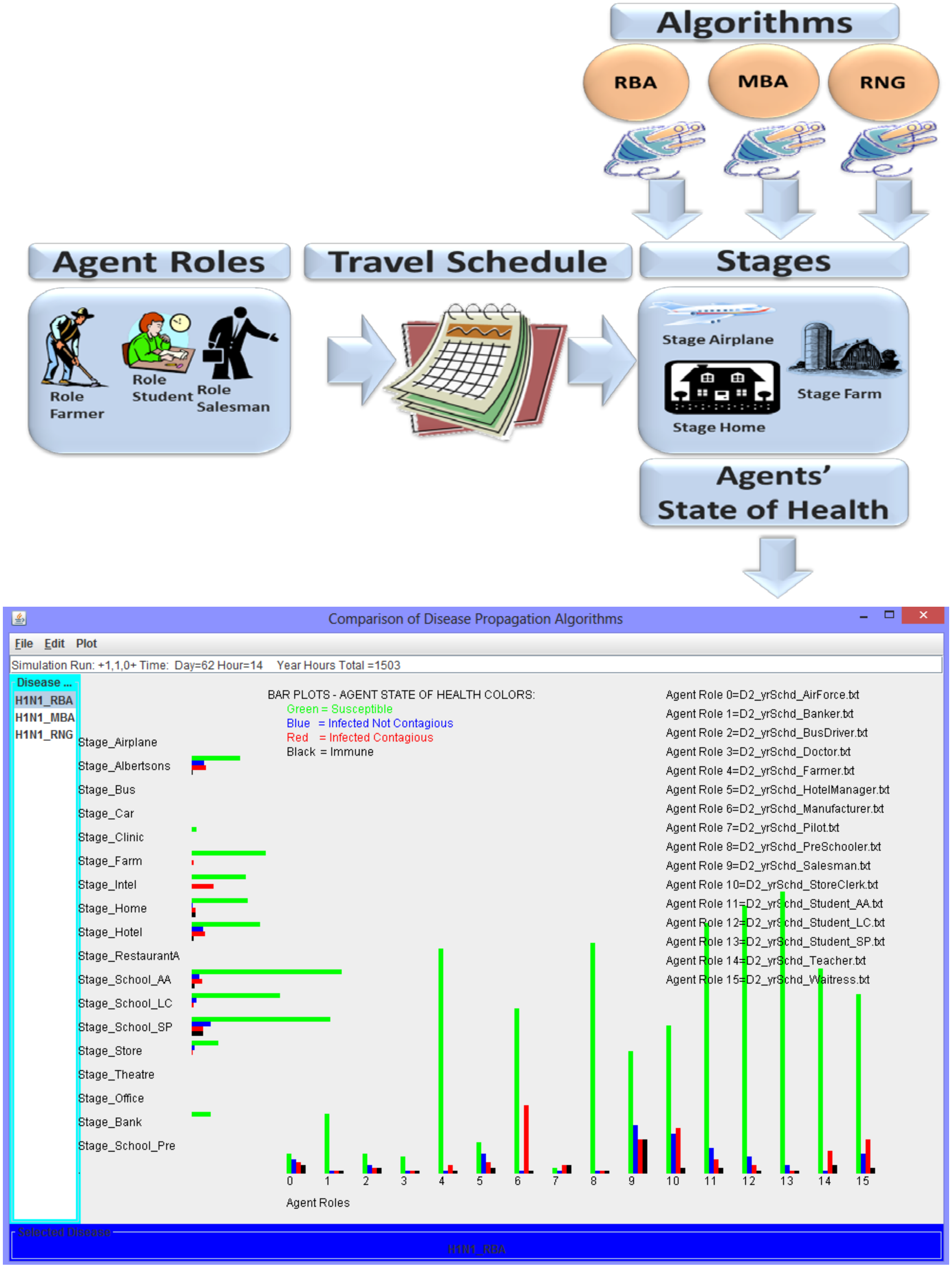
Methods Used to Solve Problem

An Agent-Stage based model is created to move Agents within a 365 day simulation. An Agent-Stage model is the combination of an Agents travel schedule to Stages locations using the

Agent hierarchical schedule. This Agent-Stage model flow diagram is provided in Figure 1 titled which is titled “Overall Agent-Stage Activities.”   Here there are five main activity areas:

- Agent Roles
    - Provides an idea of what roles are
- Schedule
    - A calendar schedule for Agents of a given role
- Stages
    - Locations where Agents can go to
- Plug-in H1N1 propagation algorithms
    - Identifying that these are added to the Agent simulation
- GUI displaying Agents’ state-of-health
    - user can view the simulation run Agent’s state-of-health

To begin, each Agent may be assigned a role which is seen here to be a farmer, student or salesman. That role follows a schedule that determines which stage location they will reside at for a given amount of time such as home, an airplane or a farm. For each stage a virus propagation algorithm acts upon the agent’s state-of-health that resides in each stage. Last the Agents state-of-health is displayed in real-time during a simulation run on the program’s GUI in a bar-chart format.



Simulation Run: +1,1,0+ Time: Day=62 Hour=14 Year Hours Total =1503

BAR PLOTS - AGENT STATE OF HEALTH COLORS:
Green = Susceptible
Blue = Infected Not Contagious
Red = Infected Contagious
Black = Immune

Agent Role 0=D2_yrSchd_AirForce.txt
Agent Role 1=D2_yrSchd_Banker.txt
Agent Role 2=D2_yrSchd_BusDriver.txt
Agent Role 3=D2_yrSchd_Doctor.txt
Agent Role 4=D2_yrSchd_Farmer.txt
Agent Role 5=D2_yrSchd_HotelManager.txt
Agent Role 6=D2_yrSchd_Manufacturer.txt
Agent Role 7=D2_yrSchd_Pilot.txt
Agent Role 8=D2_yrSchd_PreSchooler.txt
Agent Role 9=D2_yrSchd_Salesman.txt
Agent Role 10=D2_yrSchd_StoreClerk.txt
Agent Role 11=D2_yrSchd_Student_AA.txt
Agent Role 12=D2_yrSchd_Student_LC.txt
Agent Role 13=D2_yrSchd_Student_SP.txt
Agent Role 14=D2_yrSchd_Teacher.txt
Agent Role 15=D2_yrSchd_Waitress.txt

Figure 1: Agent-Stage Based Simulation Overview

Coding Framework

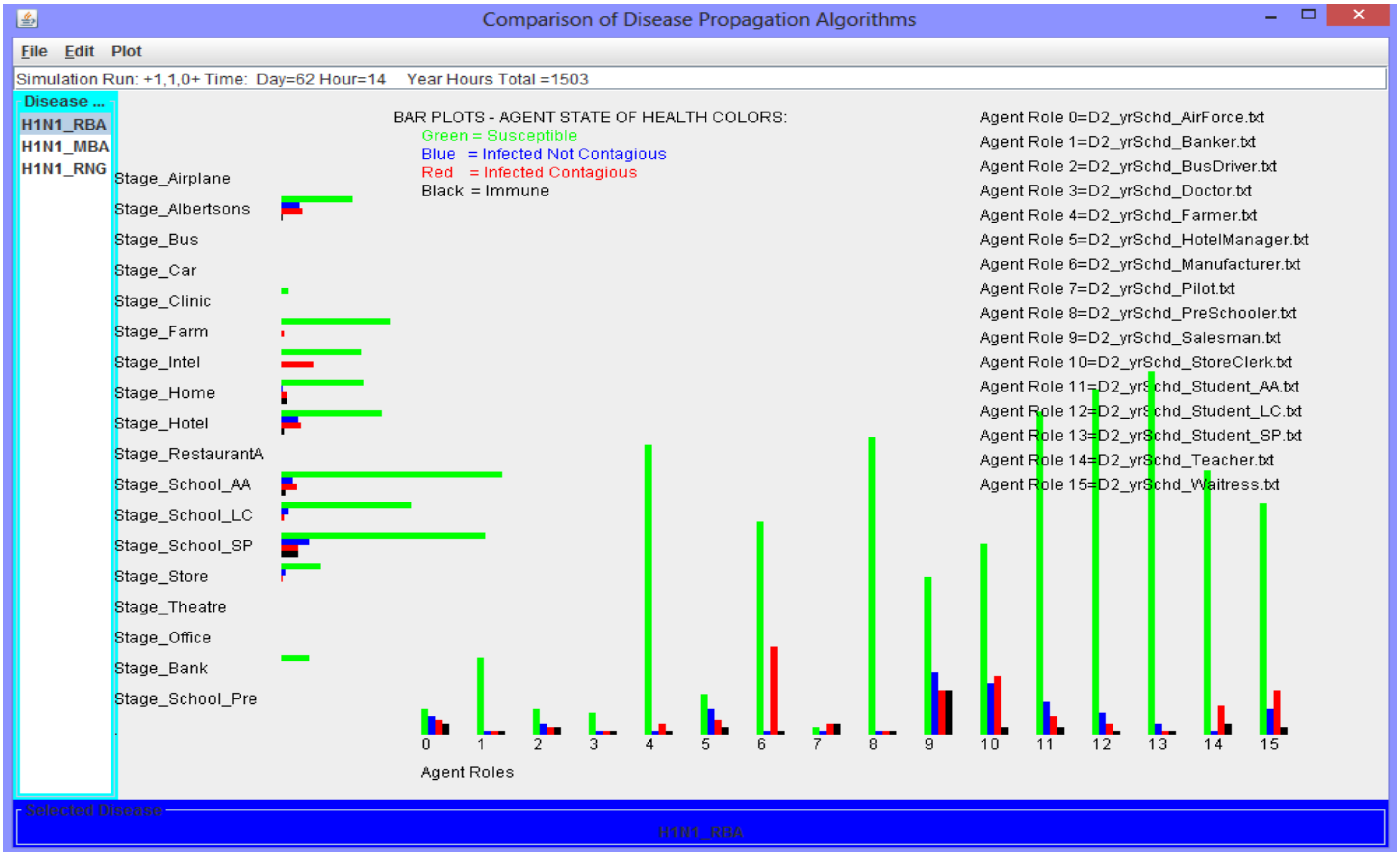
Disease Propagation Class

The disprop class is the main Graphical User Interface (GUI) of the project. The GUI, shown in Figure 2, provides several options, such as selecting which algorithm to propagate, or plotting data from the current run. The plot at the left of the GUI shows bar graphs of the Stages (horizontal state-of-health bars) and the Agent roles or professions at the bottom (vertical state-of-health bars). The Stages locations are listed from top to bottom on the left axis, and the Agent role or profession ID are listed on the bottom axis from left to right. The length of each bar represents the state-of-health for Agents that are in a particular Stage or of a particular profession. The color of each bar represents their state-of-health, as shown below.

- Green bar-Agents that are susceptible
- Blue bar-Agents that are infected, but not yet contagious
- Red bar-Agents that are infected and contagious
- Black bar-Agents immune to the virus.

As a simulation is run, the bars will change their length according to the number of Agents in that state. Additionally, at the top of the GUI is a status bar that dynamically displays the current simulation time values.

- +1,1,0+ Three tuning parameters entered through command line
- Day=62 62nd day of the simulation
- Hr=14 14th hour of the 62nd day of the simulation
- Year Hours Total=1503 Start of next simulation hour



Simulation Run: +1,1,0+ Time: Day=62 Hour=14 Year Hours Total =1503

BAR PLOTS - AGENT STATE OF HEALTH COLORS:
Green = Susceptible
Blue = Infected Not Contagious
Red = Infected Contagious
Black = Immune

Agent Role 0=D2_yrSchd_AirForce.txt
Agent Role 1=D2_yrSchd_Banker.txt
Agent Role 2=D2_yrSchd_BusDriver.txt
Agent Role 3=D2_yrSchd_Doctor.txt
Agent Role 4=D2_yrSchd_Farmer.txt
Agent Role 5=D2_yrSchd_HotelManager.txt
Agent Role 6=D2_yrSchd_Manufacturer.txt
Agent Role 7=D2_yrSchd_Pilot.txt
Agent Role 8=D2_yrSchd_PreSchooler.txt
Agent Role 9=D2_yrSchd_Salesman.txt
Agent Role 10=D2_yrSchd_StoreClerk.txt
Agent Role 11=D2_yrSchd_Student_AA.txt
Agent Role 12=D2_yrSchd_Student_LC.txt
Agent Role 13=D2_yrSchd_Student_SP.txt
Agent Role 14=D2_yrSchd_Teacher.txt
Agent Role 15=D2_yrSchd_Waitress.txt

Figure 2: disprop main GUI

Several plots are provided by the main GUI which enables the user view the data trend of a simulation run. Options to view data are provided in the following list. Figure 3 displays the Plot menu item selections and Figures 4, 5 and 6 displays the three plots created from the plot menu item. The H1N1_RBA algorithm is used in Figures 5, and 6.

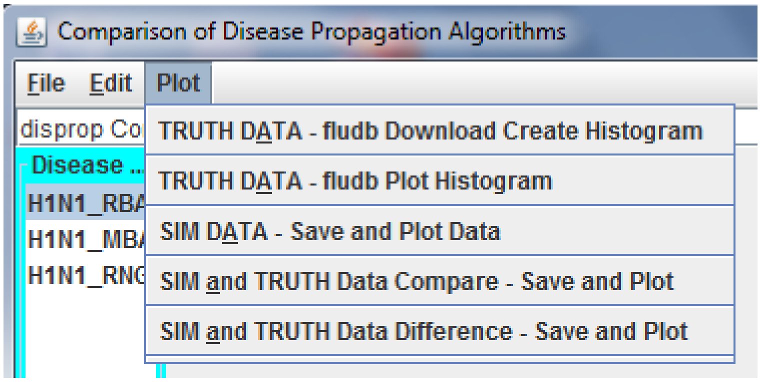

Comparison of Disease Propagation Algorithms
File
Edit
Plot
disprop Co
Disease ...
H1N1_RBA
H1N1_MBA
H1N1_RNG
TRUTH DATA - fludb Download Create Histogram
TRUTH DATA - fludb Plot Histogram
SIM DATA - Save and Plot Data
SIM and TRUTH Data Compare - Save and Plot
SIM and TRUTH Data Difference - Save and Plot

Figure 3: disprop Program Plot Drop Down Menu Figure 3: disprop Figure

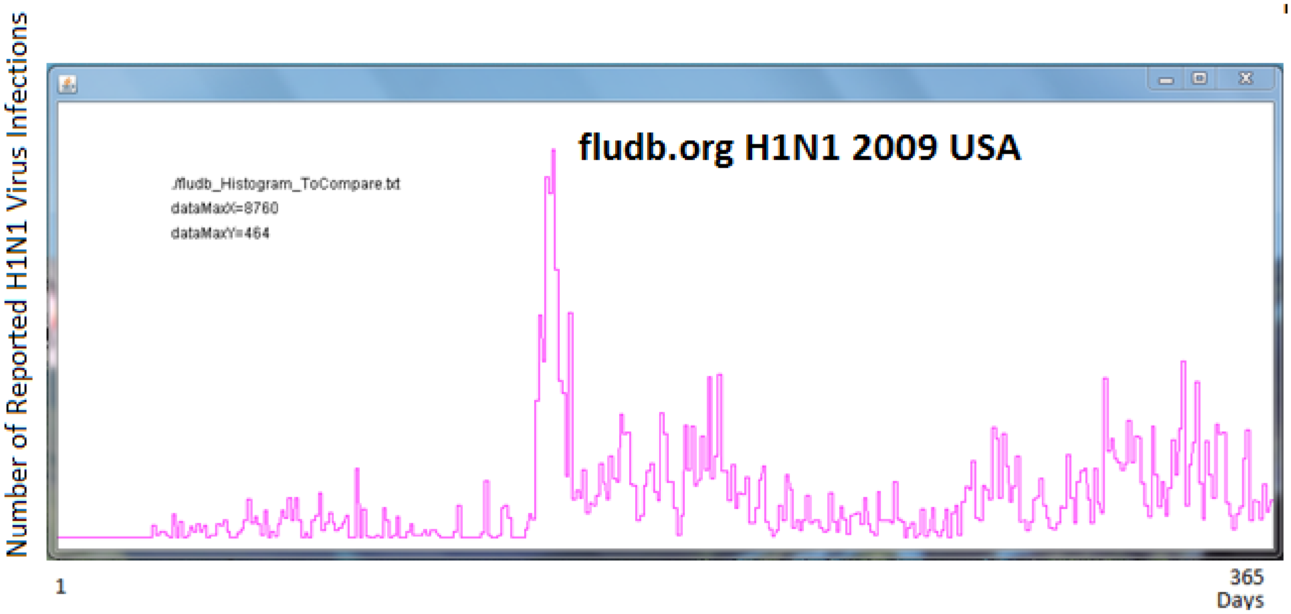


Figure 4: Program TRUTH DATA – fludb Plot Histogram

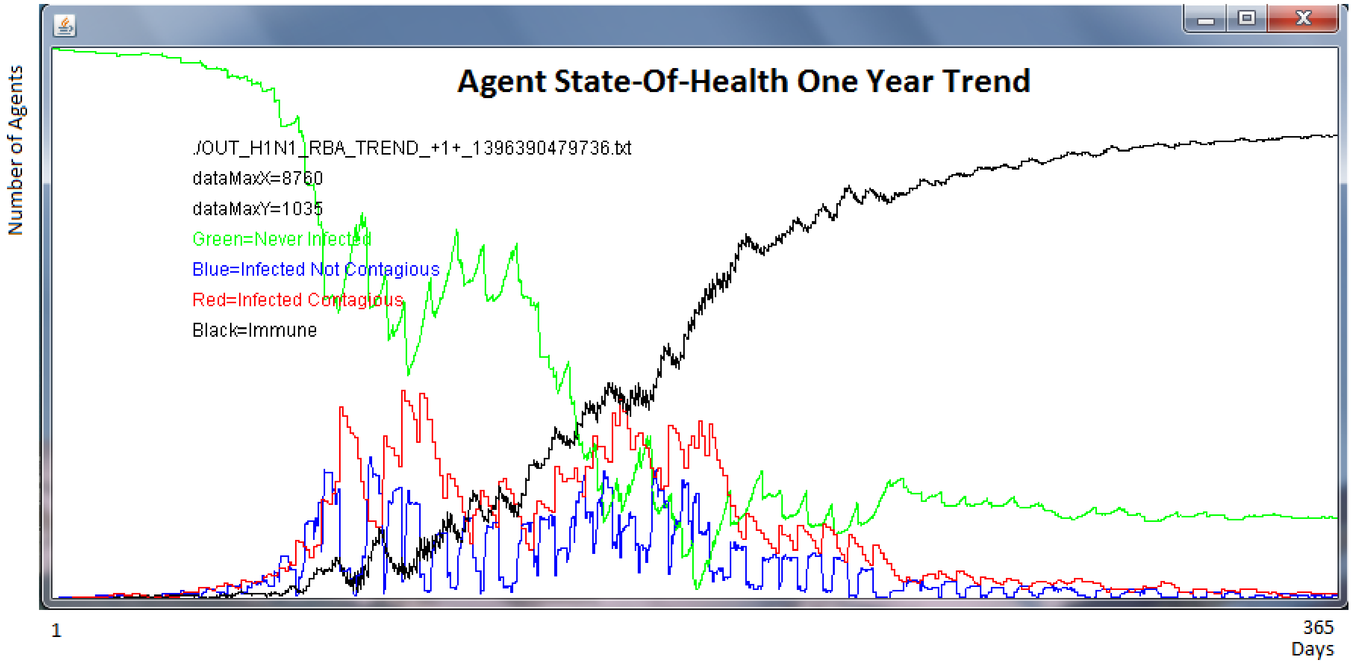


Figure 5: SIM DATA – Save and Plot Data

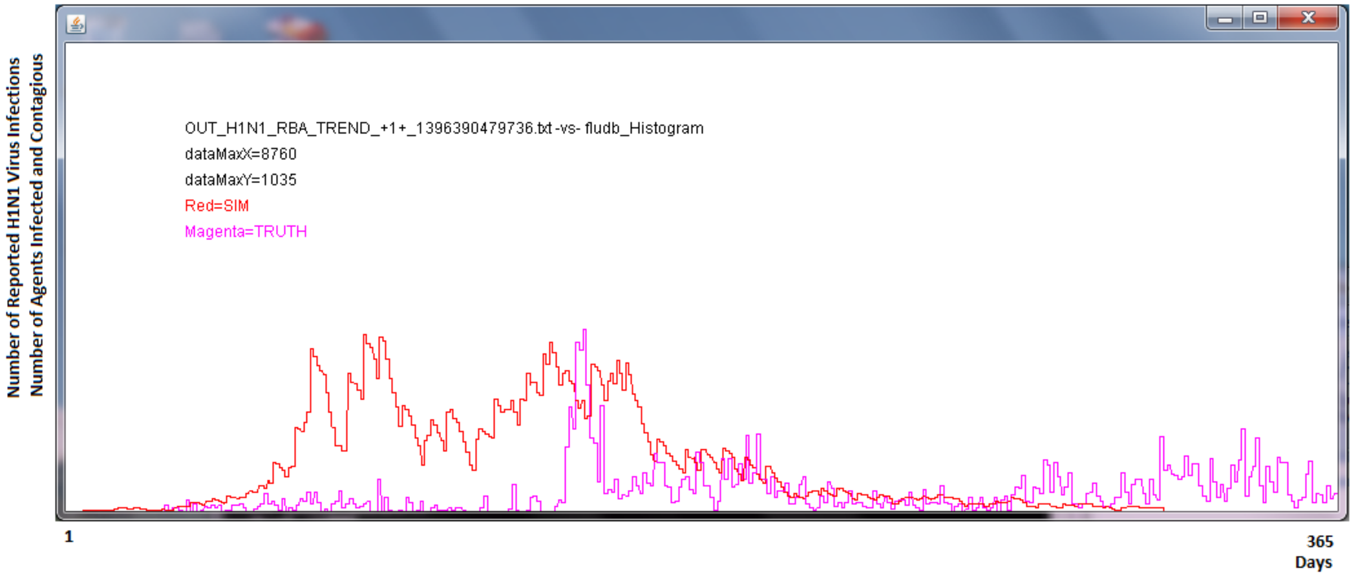


Figure 6: SIM and TRUTH Data Compare – Save and Plot

Factory Class

The Factory class creates the Agent and Stage objects and is the data holder of the simulation software. When the program is started, the disprop main class creates the Factory. Once the Factory is instantiated, it then creates all the Agents and Stages as children. Once all the Agents and Stages have been created, the Factory then serves as a database. The Factory does not push or pull data; instead its data is pushed or pulled from all other classes. In order to make the program fast and simple all Agents and Stages run on separate threads. Figure 7 displays the relationship of the push and pull of data in the Agent-Stage Model.

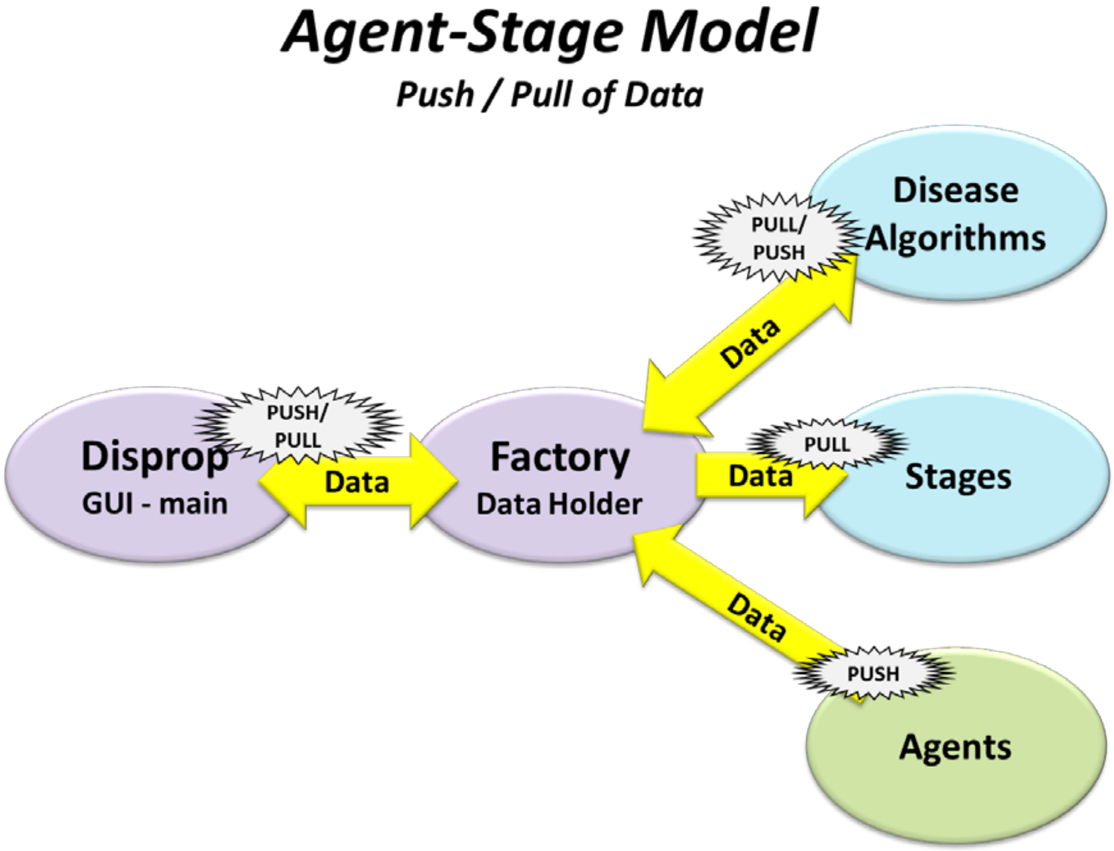


Figure 7: Agent-Stage Model with Push and Pull of Data

Agent Class

Agents represent people in the simulation. Each Agent is an instance of the Agent class that runs on its own separate thread. All Agents can move from Stage to Stage at the top of the simulation hour. The Agent's movement patterns are dictated by its schedule as provided in Figure 8. Each role or profession of an Agent has a unique schedule displayed as a D2_yrSchd_<role>.txt as seen in Figure 8. As an example, a student goes to school every weekday, while traveling salesman has a more robust schedule. These schedules provide a more lifelike simulation compared to randomly moving Agents. At the top of the simulation hour each Agent object looks at its schedule and assigns its current Stages according by updating the Factory database that is holding the Agents' locations

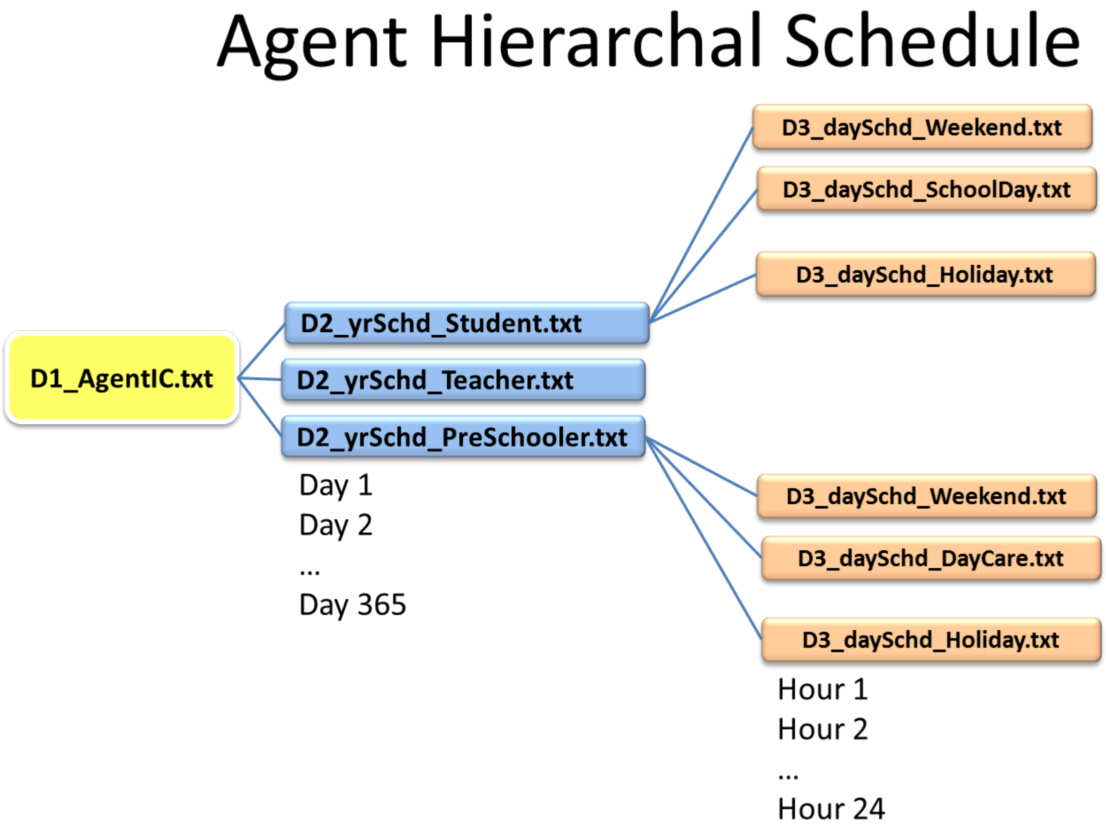
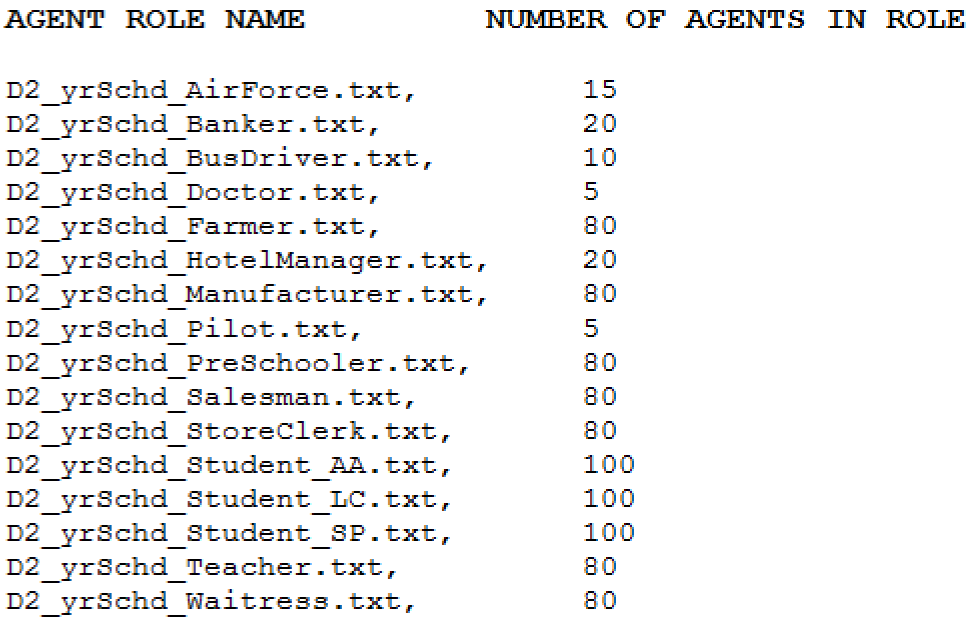


Figure 8: Agent Hierarchal Schedule

The following list is taken from the current D1_AgentIC.txt file in Figure 9 which contains Agent roles names (first column) and the number of Agents in each role (second column) that was used in this project. From this list, one can see that there are ten bus driver roles; i.e. there are ten Agents that are bus drivers. It is noted that this flat file list is read into the program when the simulation starts.



```
AGENT ROLE NAME              NUMBER OF AGENTS IN ROLE

D2_yrSchd_AirForce.txt,           15
D2_yrSchd_Banker.txt,             20
D2_yrSchd_BusDriver.txt,          10
D2_yrSchd_Doctor.txt,             5
D2_yrSchd_Farmer.txt,             80
D2_yrSchd_HotelManager.txt,       20
D2_yrSchd_Manufacturer.txt,       80
D2_yrSchd_Pilot.txt,              5
D2_yrSchd_PreSchooler.txt,        80
D2_yrSchd_Salesman.txt,           80
D2_yrSchd_StoreClerk.txt,         80
D2_yrSchd_Student_AA.txt,         100
D2_yrSchd_Student_LC.txt,         100
D2_yrSchd_Student_SP.txt,         100
D2_yrSchd_Teacher.txt,            80
D2_yrSchd_Waitress.txt,           80
```

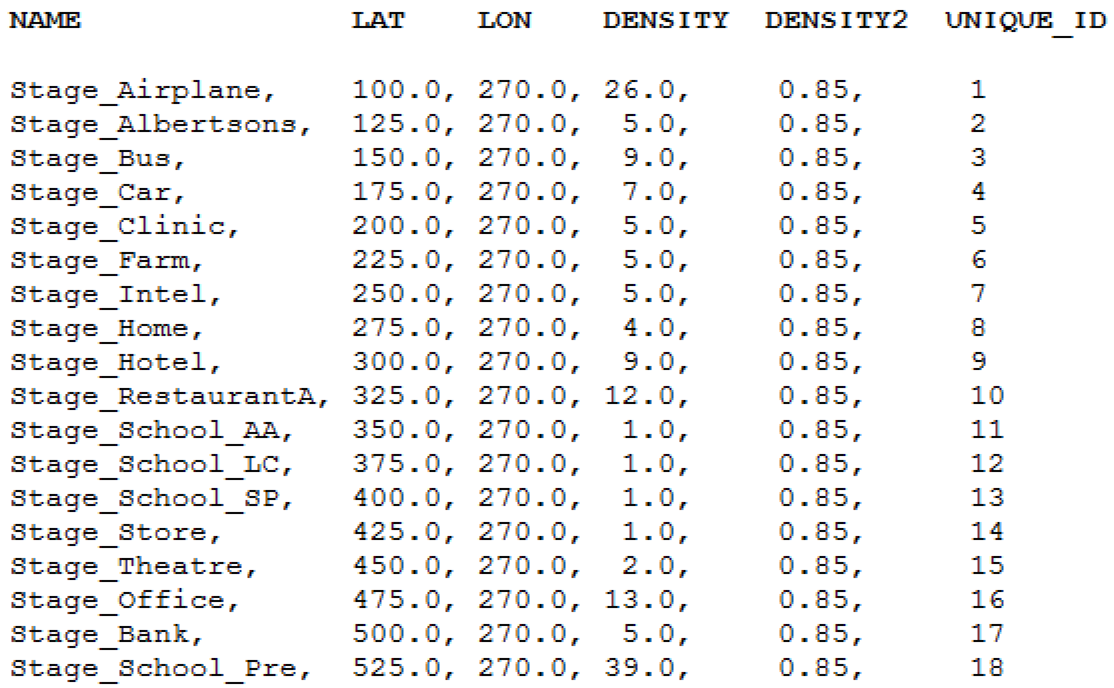
Figure 9: D1_AgentIC.txt

Stage Class

Stages are locations that the Agents travel to per their pre-determined schedule. Each Stage functions as a location or place where Agents conceptually reside in. As an example, a Stage could be a house, an airplane or a school. At startup, each Stage running in its own thread instantiates a child disease propagation algorithm. During a simulation, each Stage's child algorithm propagates a virus to Agents visiting a Stage by modifying the Agents' state-of-health data that resides in the Factory database.   Stages are instantiated by the Factory when the Factory is created and Stages run simultaneously on separate threads.   Upon creation, each Stage instantiates the selected algorithm as a child. Any number of Stages and Agents as can be created in the Factory, but the number created is limited by the amount of memory the machine running the program contains.

Currently eighteen Stages have loosely been modeled after Albuquerque, NM where the following Stages are being used: Airplane, Albertsons (grocery store), Bus, Farm, Clinic, Farm, Intel, Home, Hotel, RetaurantA, School_AA, School_LC, School_SP, Store, Theatre, Office, Bank, and School_Pre. These Stages act as places for the Agents to visit. Each hour of simulation time, each Agent may move to any Stage predetermined by its schedule. The propagation of the disease occurs in these Stages.

Stages additionally contain unique densities found in the D4_StageLatLon.txt flat file as seen in Figure 10. The range of a stage density (column 4) is from one to one-hundred depending on what type of location it is. For example, a higher concentrated place such as an airplane would have a density value such as twenty-six. This is unlike a farm which is quite open, which would have a low density value such as five. The Stage data file used in this project is provided below. Stage name is provided in the first column, the location on the GUI is noted by LAT and LON. Density is the third column, fourth column is an optional density adjustor and the UNIQUE_ID is the last column.



```
NAME                LAT     LON    DENSITY  DENSITY2  UNIQUE_ID

Stage_Airplane,     100.0, 270.0, 26.0,      0.85,       1
Stage_Albertsons,   125.0, 270.0,  5.0,      0.85,       2
Stage_Bus,          150.0, 270.0,  9.0,      0.85,       3
Stage_Car,          175.0, 270.0,  7.0,      0.85,       4
Stage_Clinic,       200.0, 270.0,  5.0,      0.85,       5
Stage_Farm,         225.0, 270.0,  5.0,      0.85,       6
Stage_Intel,        250.0, 270.0,  5.0,      0.85,       7
Stage_Home,         275.0, 270.0,  4.0,      0.85,       8
Stage_Hotel,        300.0, 270.0,  9.0,      0.85,       9
Stage_RestaurantA,  325.0, 270.0, 12.0,      0.85,       10
Stage_School_AA,    350.0, 270.0,  1.0,      0.85,       11
Stage_School_LC,    375.0, 270.0,  1.0,      0.85,       12
Stage_School_SP,    400.0, 270.0,  1.0,      0.85,       13
Stage_Store,        425.0, 270.0,  1.0,      0.85,       14
Stage_Theatre,      450.0, 270.0,  2.0,      0.85,       15
Stage_Office,       475.0, 270.0, 13.0,      0.85,       16
Stage_Bank,         500.0, 270.0,  5.0,      0.85,       17
Stage_School_Pre,   525.0, 270.0, 39.0,      0.85,       18
```

Figure 10: D4_StageLatLon.txt

Time Synchronization

Since the simulation program has hundreds or thousands of objects each on a separate thread, they all must be synchronized to the same 24 hour, 365 day schedule to be organized and avoid data corruption. This is done through the use of a wall clock in simulation time. When the Factory is first instantiated by disprop (disease propagation), it takes the current computer time and makes this time the start time of the simulation. The Stages, Agents, and disprop objects all use this initial start time for their synchronized clocks. This simulation start time variable is shared throughout the program through the Factory data holder. During run time each section of the code has access to the system clock (epoch time) and keeps track of time as necessary. Stages synchronize with the wall clock by using the current computer time at a particular time and then subtract the simulation start time from that. From the amount of milliseconds that have passed it determines the current simulation time. Using the day counter and the milliseconds in one day, the current day start time and the current day end time are calculated. The process is provided as follows.

If the current simulation time is less than the day start time (running behind), then that part of the program will run immediately so that it can catch up. If the current simulation time is greater than the day start time (running too fast), then that selection of code will sleep for one hour of simulation time and tries again. Next the selection of code will then go through the following hour of the day. It will calculate the start and end time of each hour.

If the simulation time is less than the hour start time (running behind), then that part of the program will run immediately so that it can catch up. If the current simulation time is greater than the hour start time (running too fast), then that selection of code will sleep for one hour of simulation time and tries again. When the time is the correct day and hour the Agent, Stage, or disprop that is running will then process its data; when done it will proceed to the next hour. Eventually the day will end and the clock will then calculate a new day start and end, and new hour starts and end, and the process repeats itself for the next iteration.

The simulation program has a granularity of one hour. Thus the program simulation updates on an hourly basis with an acceptable error of one hour offset. The main reason for the clock synchronization is to avoid data corruption of the Factory database’s Agent location status. This is resolved through a simple system. At the top of every hour in simulation time, disprop pulls the data it needs for displaying the GUI, the Agents push their Stage location to the Factory at the top of the hour and the Stages stop the simulation of its algorithm. Then after fifteen minutes of simulation time the Stages pull from the Factory Agent location data and create a new linked list of Agents who are inside them for its child algorithm to use. Since the disease algorithms are a child of the Stage, the disease algorithms also know which Agents are visiting them. This cycle then repeats itself over and over again, till 365 days of simulation time has been run.

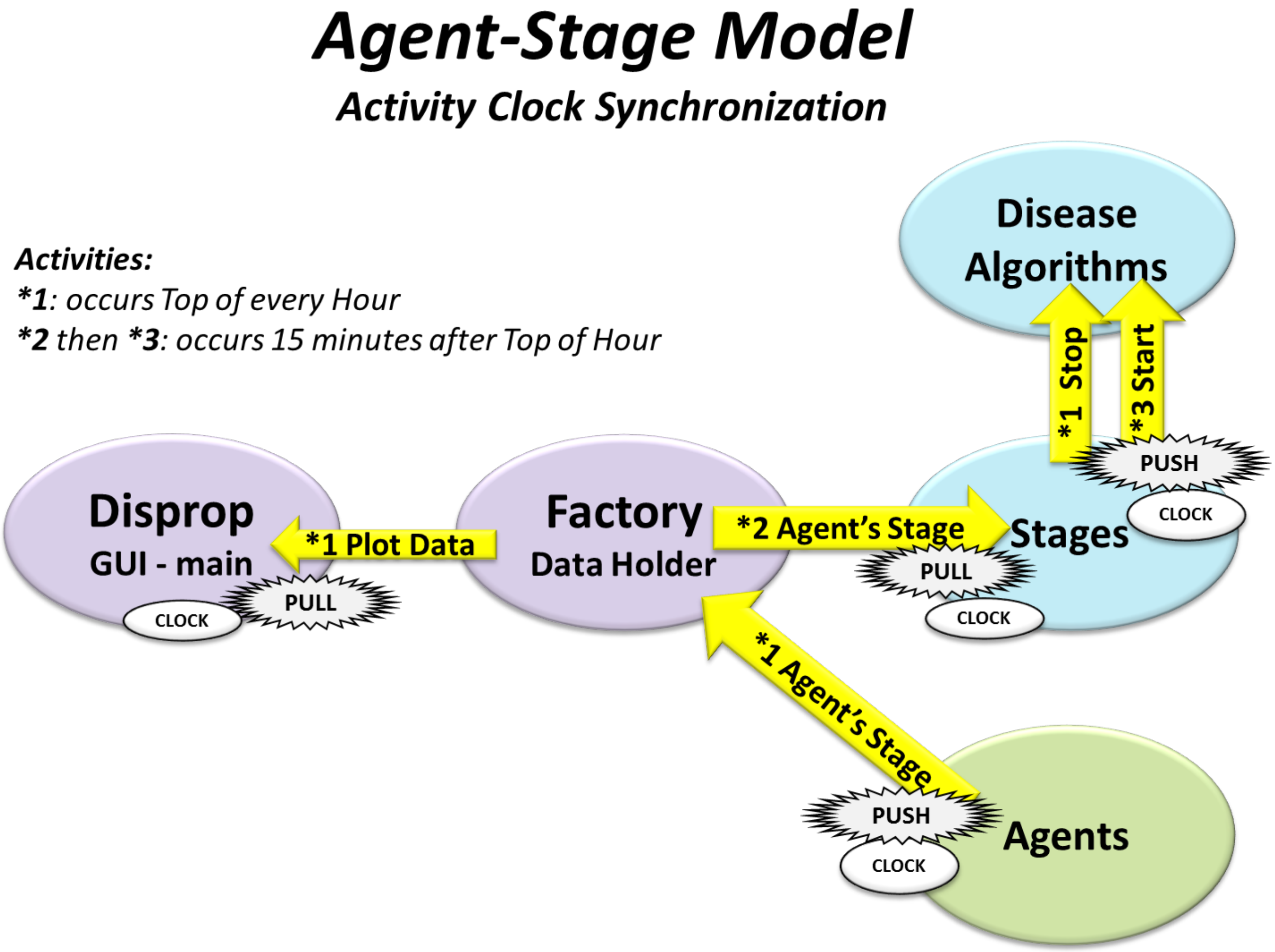


Figure 11: Activity Clock Synchronization

Disease Algorithms

The disease algorithm’s job is to model the propagation of a disease. It does this by propagating a virus to all Agents’ state-of-health that are visiting that algorithm’s Stage. Each Agent has four possible state-of-healths, which are: susceptible, infected but not contagious, infected and contagious, and immune. Each Stage has an identical copy of the selected disease algorithm, thus the disease propagates to the visiting Agents within each Stage. This is possible due to the fact that the selected algorithm is a child of the Stage class. The disease algorithm is updated every hour according to the clock synchronization. Agents can enter, leave, or stay within the Stages. Thus the Agents in the Stage can have their state-of-health modified by the algorithm. In this project three different types of H1N1 algorithms were created: Rule Based Algorithm (RBA), Math Based Algorithm (MBA), and Random Number Generator Algorithm (RNG). The following sections briefly describe each algorithm.

The R-Naught is defined as how many people on average one contagious person infects. The R-Naught of the H1N1 is determined to between 1.4 and 1.6 [www.biomedcentral.com].

Rule Based Algorithm (RBA)

The Rule Based Algorithm (RBA) uses rules in unison with simple math calculations to simulate the propagation of a disease. The rules are applied in the form of logic statements. In this algorithm the first Agent in the link list array of the first Stage is infected with the H1N1 virus, this is the virus seed. After this happens, the algorithm uses “if” statements to determine when Agents should move on to the next state-of-health (susceptible, infected but not contagious, infected and contagious, and immune) based on their current state-of-health and how many days they have been infected. When an Agent first gets infected, they are not contagious. After a set number of days, the Agent will become contagious and then eventually immune. Also this algorithm models mutations. It does this by generating a random number from one to 365 then determining if this number is less than the current day number. If it is, then an immune Agent becomes a susceptible Agent again and its R-Naught is increased to one. This models mutations which become more prevalent throughout the year and which are weaker than the original strain.

This algorithm contains the logic where a fomite can infect an Agent. A fomite is defined as an inanimate object that is contaminated with infectious organisms and serves in their transmission [merrian-wester.com]. This algorithm determines the Agents' state-of-health at the beginning of the hour by modeling disease propagation. When running for each hour, the algorithm first determines if the Stage has any contagious Agents inside it. Once it finds a contagious Agent, it adds one to the fomite counter. Also adds the sum of four hours and the fomite modifier together to determine how long the fomite is infectious for. Then the algorithm searches to find a susceptible Agent. Once this happens a random number is generated in the range from one to one-hundred. This random number is then used in a calculation. This calculation multiplies the random number generated by the reciprocal of the density (how many people on average are in the Stage in respect to the Stage size obtained from the Stages data file) plus one in the denominator.

Once the result of this calculation has been determined, the value calculated is compared to a threshold level. If the value is above the threshold, then the R-Naught (initially set to two) of the infectious Agent is reduced and the susceptible Agent is infected. Next the R-Naught is further reduced again (to attain a goal between 1.4 and 1.6) if another random number between one and 365 (number of days in a simulation year) is less than the current day with a combined probability of a twenty percent probability. After the algorithm infects susceptible Agents from contagious Agents, it tries to infect susceptible Agents with fomites. It does this by finding all of the Agents that are still susceptible. Then it checks to see if the fomite can infect Agents by checking if the fomite counter is above zero (checking for the presence of fomites). Next checking if a random number from one to a hundred is above a threshold value and if the fomite has not expired (if the current hour is less or equal to the last hour that the fomite can be contagious). Once it passes this if statement the program checks to see if it is in the colder months (first 60 and last 60 days of the year) since fomites are prevalent in colder moister weather. If it is winter, then the fomite can infect a susceptible Agent. Then the fomite counter is reduced.

Math Based Algorithm (MBA)

MBA stands for Math Based Algorithm, meaning it uses calculations to determine a chance of infection and limit the amount of people one can infect. In order to do so, it calculates both an adjusted base reproduction number and a percent chance of getting sick. The adjusted R-Naught is calculated by taking the base R-Naught of the disease, cited below, and adjusting it to the amount of immune people in a Stage. The percent chance infection is calculated according to population density in a Stage and the amount of people who are contagious. The two variables that are adjusted in this comparison are the base reproduction number and the density of the Stage.

The first part of the Math Based Algorithm retrieves the Agent state-of-health data from the Factory. The second part of the algorithm deals with the calculation of both the adjusted R-Naught and the percent chance of infection. The original R-Naught of the H1N1 is around 4.89 unadjusted. This came from the equation R-Naught=L * A, where L is the average life expectancy of a population, and A is the average age at which the disease is contracted. In order to adjust the R-Naught, we multiplied it by -1 plus the proportion of immune people in the Stage vs. total people in the Stage. This is an adapted version of how an R-Naught is adjusted to a population according to a threshold determined by the number of people born with immunity. The adapted version used the same principles, but applies them solely to the population of the Stage and the number of immune Agents in the Stage at that hour in simulation time.

Random Number Generated Algorithm (RNG)

The random number generator was intended to be the simplest algorithm possible. Using the equation in Figure 12, it calculates *P*, where *P* is percent chance of infection, and K is an adjustable tuning variable.

$$P = \sqrt{Sick\ Contagious\ Counter} * Density * \frac{K}{100}$$

Figure 12: RNG Equation

The algorithm then creates a random number (between an adjustable range). If the random number is less than *P*, the Agent becomes infected; otherwise the Agent will remain healthy. The Agent will be infected and not contagious for three days. Then the Agent will be contagious for nine days before becoming immune.

Verifying and Validating Computational Results of Study

The output data was validated through the GUI state-of-health bar charts running in real time. This was compared with the Agents calendar schedule by verifying if the schedule was being implemented correctly. Next data outputs were sent to the console window during runtime providing logging information to determine if the propagation algorithm was infecting other Agents according to the intended design. This project provided Agent state-of-health trends from the RBA, MBA and RNG algorithm that will be analyzed in this section.

Sensitivity Analysis Using Latin Hypercube Sampling

A sensitivity analysis was performed on tuning parameters (variables) to determine their sensitivity and thus their importance to each of the three algorithms.   The intent was to determine the tuning parameters which produced a propagation pattern trend that most closely matched the H1N1 truth data trend.

To start, if a Latin Square was used, all tuning parameter mutations would need to be run and compared with the truth data. An Orthogonal Latin Hypercube approach was used to perform sensitivity analysis to make this effort more efficient. An Orthogonal Latin Hypercube is a sampling of a Latin Square. A Latin Square simulation goes through all interval values for in a given range, this is indicated in Figure 12. The value for Variable 1 steadily increases by one just like for the value for Variable 2 steadily increases by two. Also both of the variables being compared have the same number of variations so that the data set makes a “n by n” dimension (where n=3) square. This is also shown in Figure 12 where there are three values of both Variable 1 and 2 being compared. The Latin Square though is varied with a third variable the as shown in Figure 12 where in the grid there are the letters A, B, and C. No two rows or columns have the same permutations of the values A, B, and C. This gives one a sampled analysis of variable importance within the simulation and also what different combinations of variables mean.

An Orthogonal Latin Hypercube sampling of a Latin Square occurs when on sample of each type occurs. No exact permutations exist in any row or column. For example in Figure 13, no duplicate samples taken are in any row or column. This allows a more efficient way to test a Latin Square which the goal is to be randomly distributed. If one compares Figures 13 and 14, one can see that they are quite different; however both are Orthogonal Latin Hypercube. This demonstrates how Orthogonal Latin Hypercube is constructed. In our project we randomly distribute our Orthogonal Latin Cube. This eliminates testing all of the permutations, and tests a set of few which will still give a big picture concerning the sensitivity of variables and the effects of certain permutations.

Latin Square:				
Variab le 1	3	A	B	C
	2	B	C	A
	1	C	A	B
		2	4	6
	Variable 2			
A,B, and C are variations of a third variable				

Figure 12: Latin Square

Latin Hypercube Orthogonal Sampling:				
Variab le 1	3	A		
	2			C
	1		B	
		2	4	6
	Variable 2			
A,B, and C are variations of a third variable				

Figure 13: Latin Hypercube Orthogonal Sampling

Latin Hypercube Orthogonal Sampling:				
Variab le 1	3		C	
	2	A		
	1			B
		2	4	6
	Variable 2			
A,B, and C are variations of a third variable				

Figure 14: Latin Hypercube Orthogonal Sampling, Example 2

Truth Data for Comparison

In this project the algorithms are compared to the H1N1 truth data. The Fludb.org is a website that contains data that is received from hospitals, clinics, doctor's offices, etc. concerning how many confirmed cases there are of a particular virus. We gathered truth data about the outbreak of the H1N1 virus in 2009. The Fludb.org provided us with how many confirmed cases of H1N1 there were throughout the entire United States during 2009. The Fludb.org is funded by the National Institute of Allergy and Infectious Diseases and collaboration between Northrop Grumman Health IT, J. Craig Venter Institute, Vecna Technologies, SAGE Analytica and Los Alamos National Laboratory. This project used the same truth data as the Supercomputing Challenge project titled "Model of Disease Propagation to Predict the Speed of an Outbreak", April 2013.

RBA Sensitivity Analysis and Results

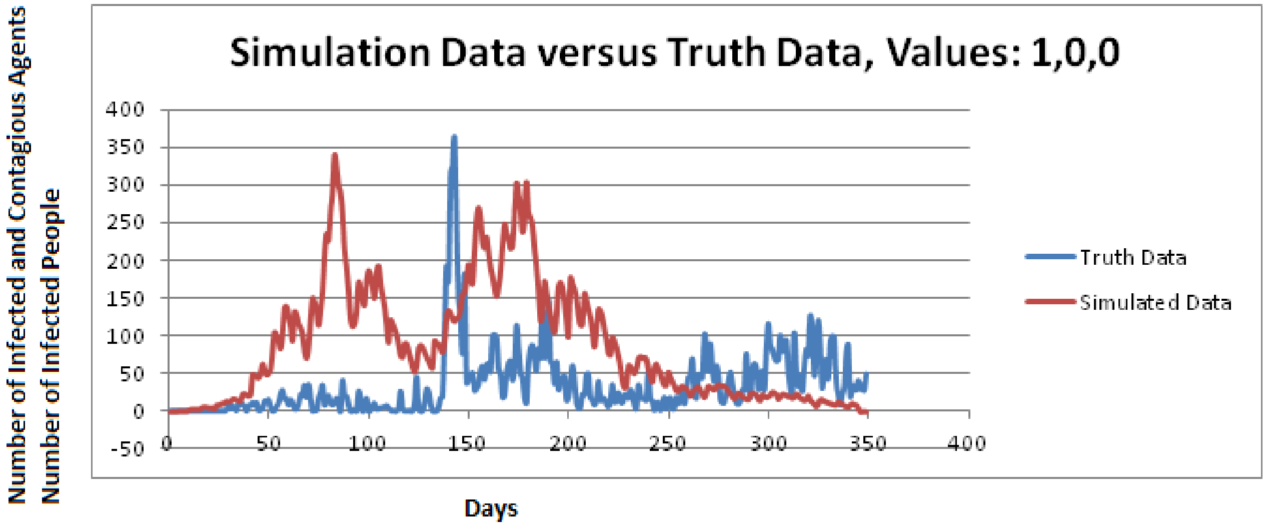
The Rule Based Algorithm's (RBA) variables were analyzed using a Latin Hypercube. The three input variables that were changed are the start day of contagiousness for contagious Agents (how it longs it takes until an Agent becomes contagious or how long an Agent is infected but not contagious), the R-Naught modifier (determines if a contagious Agent can infect one or two Agents, the higher this value, the more probable that two Agents can be infected), and the end hour of contagiousness for a fomite (how long a fomite can be contagious for). It was found that the R-Naught was the most sensitive of the three variables and varied the results of the simulation profoundly. The reason why the R-Naught modifier is most sensitive variable is because if the R-Naught modifier is high then the peaks will appear later in the simulation. Also the R-Naught modifier is the most influential variable because it changes the trend the most. This is shown by Graphs 1 and 7. The effect that the first day contagious has on the output graph is that when its value is high pushes the second hump forward in time best shown by Graphs 1 and 3. When the fomite duration of contagiousness modifier is high it makes slopes of the graphs right before the first peaks more horizontal. This is best shown by the contrast of Graphs 1 and 7.

The RBA algorithm was most similar to the H1N1 truth data when the R-Naught modifier had a value of one, the fomite modifier had a value of zero, and when the time infected but not contagious modifier had a value of zero. This is seen in Graph 1, when the simulated data line (composed of contagious Agents) has a double hump, like the fludb.org truth data. Even though the second hump for the simulated data is much bigger than that of the truth data's, it is the only graph that has this critical curve. Another reason why the values 1, 0, 0 (R-Naught modifiers, Infected but not contagious time modifier and fomite modifier) gave the best result is because right after the first hump, there was some residual activity, which is like the truth data graph. Even though the first humps from both graphs do not line up, this graph is still realistic because the shape of the curve tells the trend of how many people will become infected relative to the rest of the graph. This is why this run has the best graph within the sampled hypercube.

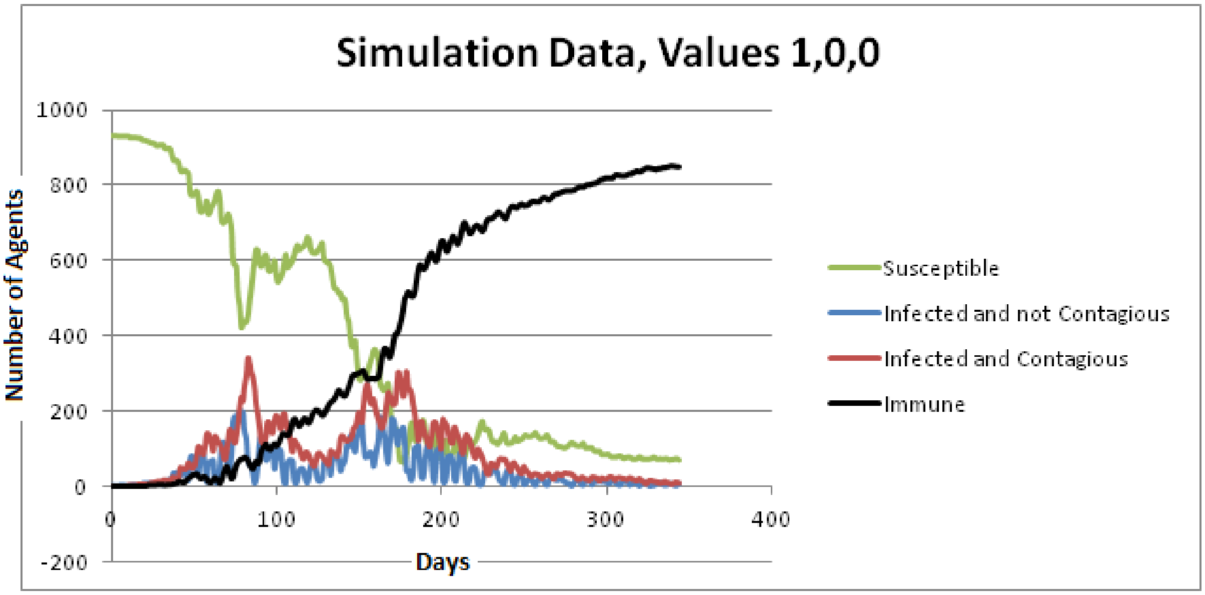
Latin Hypercube Orthogonal Sampling							
Time Infected but not Contagious Modifier	4			C			Hours Fomites are Active Modifier
	3		B				A = 0
	2				D		B = 1
	1					E	C = 2
	0	A					D = 3
		1	2	3	4	5	E = 4
	R-Naught modifier						

Figure 15: Orthogonal Hypercube used for RBA

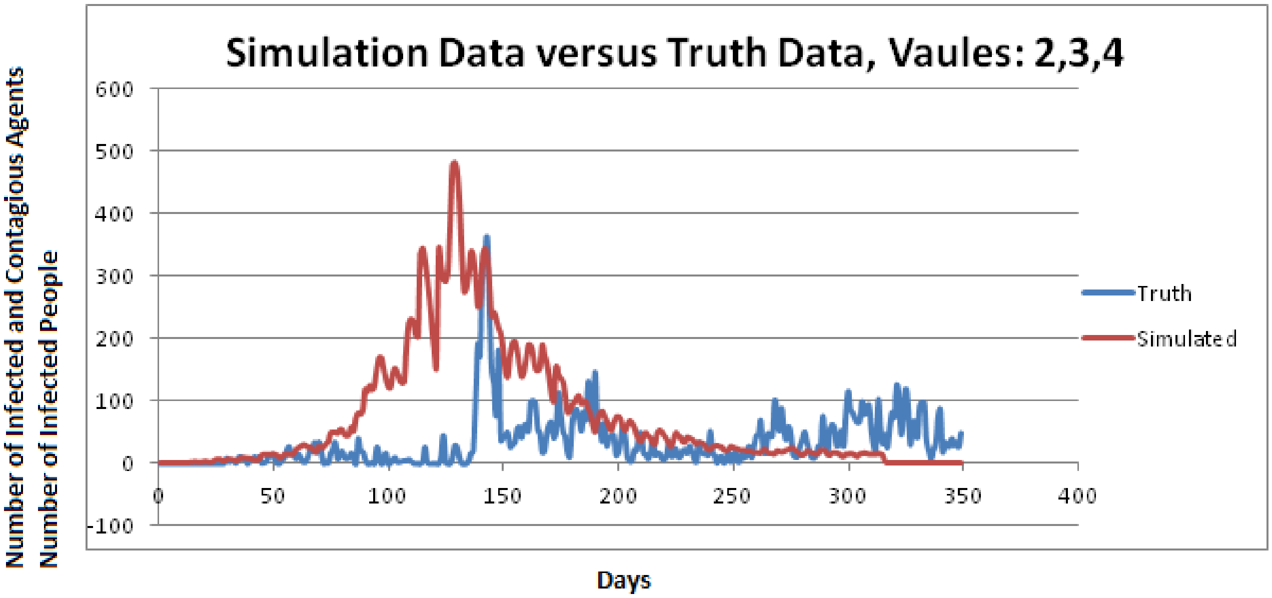
NOTE: First Variable Value is the R-Naught modifier, the second is Time Infected but not Contagious Modifier, and the third variable is Hours Fomites are Active Modifier



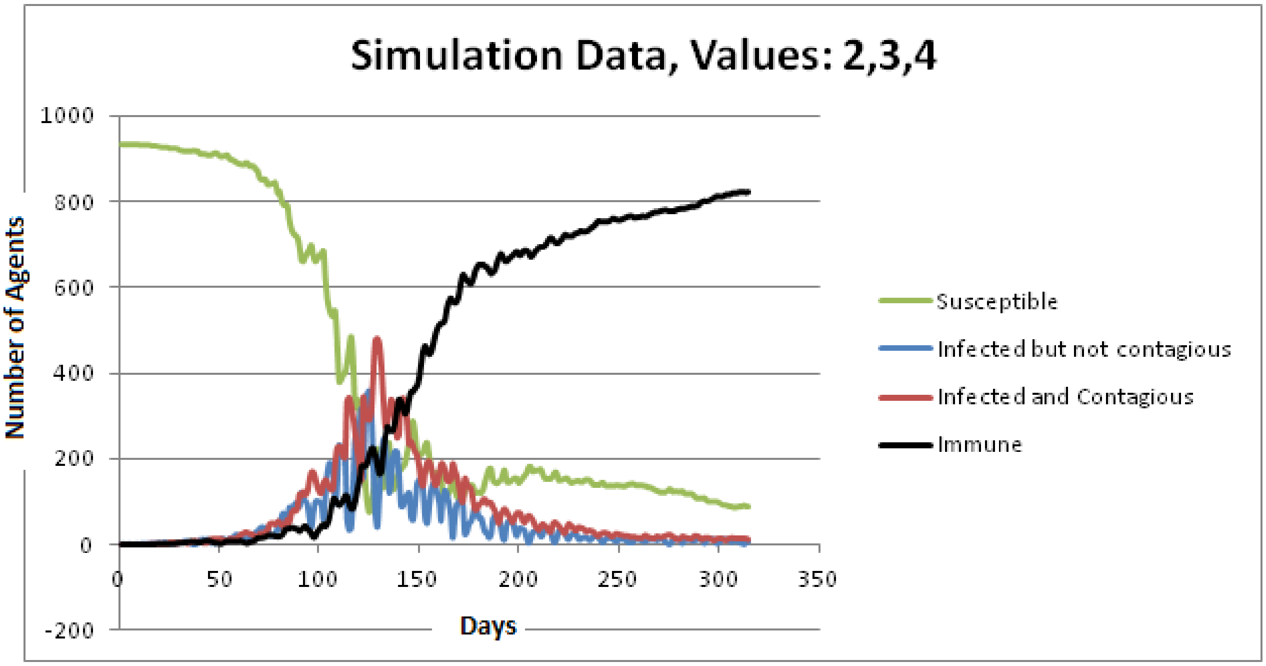
Graph 1: RBA Simulation Data versus Truth Data, Values: 1, 0, 0



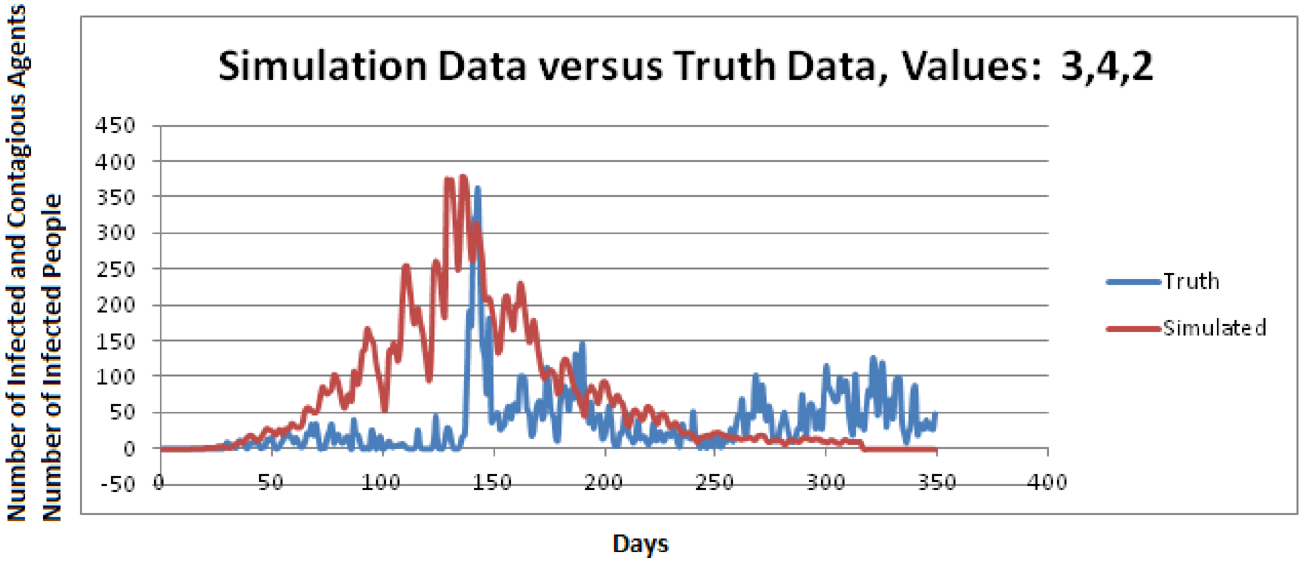
Graph 2: RBA Simulation Data, Values 1, 0, 0



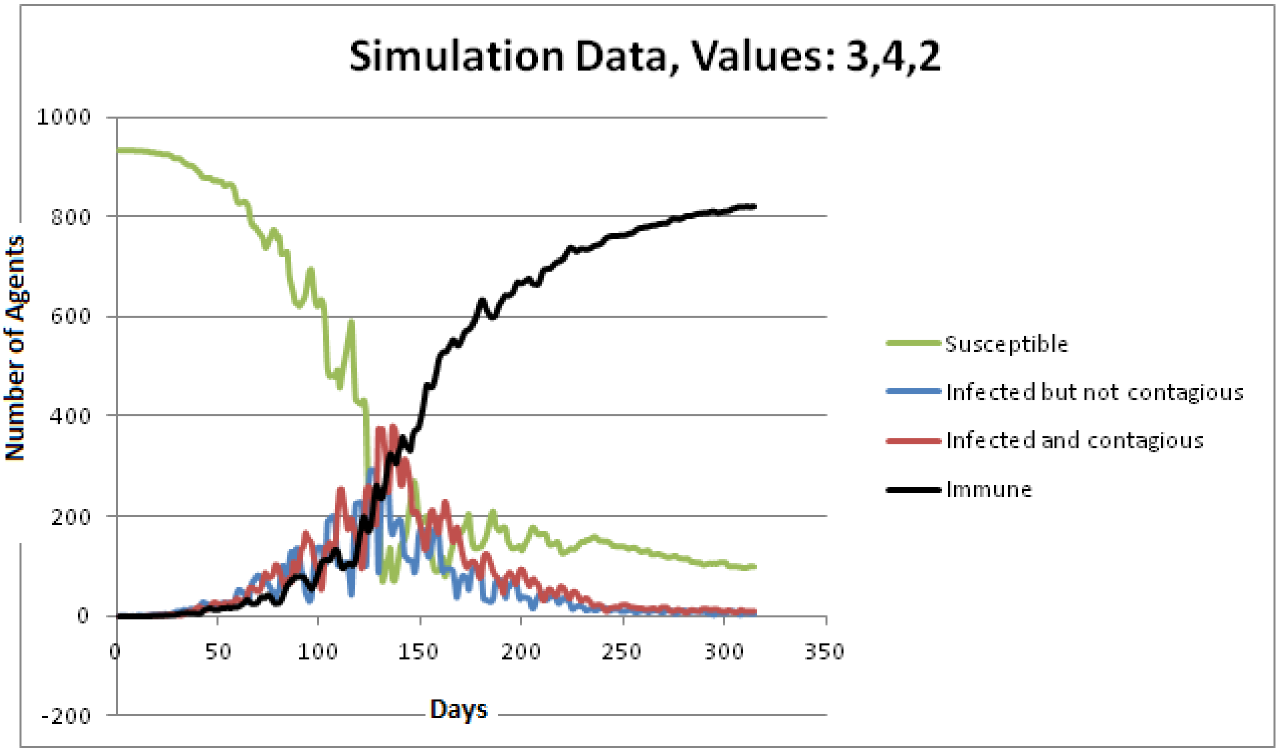
Graph 3: RBA Simulation Data versus Truth Data, Values: 2, 3, 4



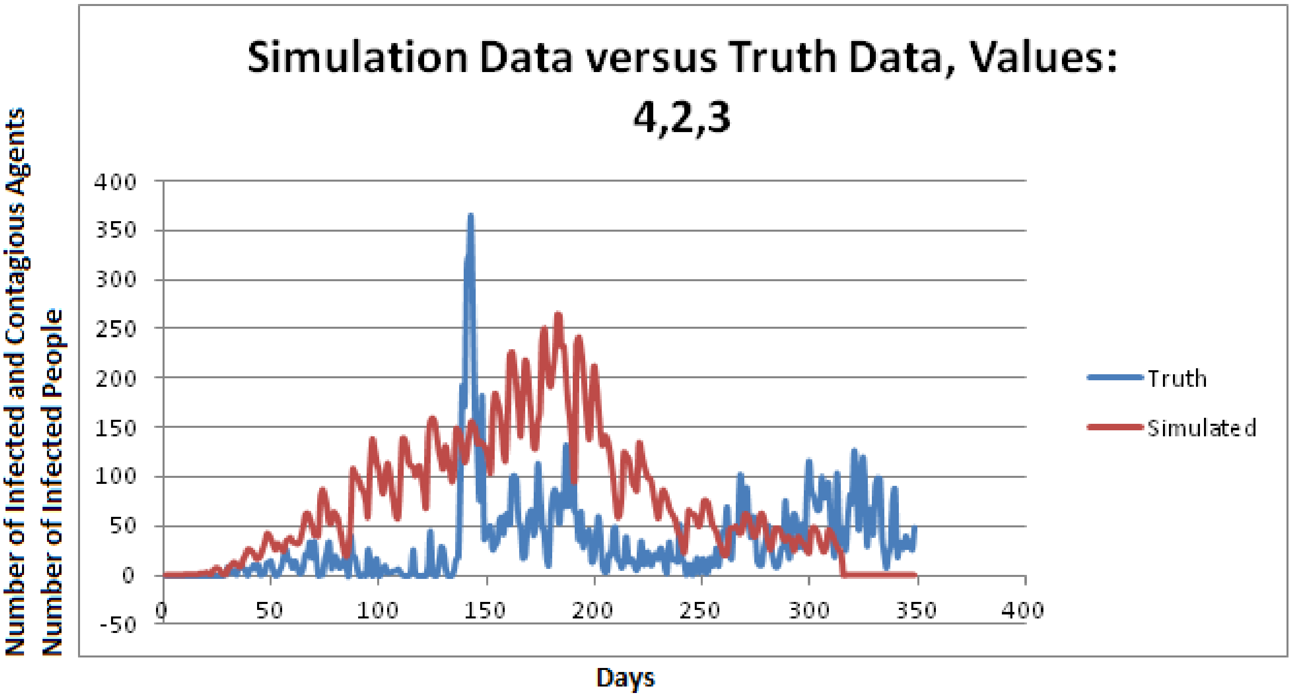
Graph 4: RBA Simulation Data, Values: 2, 3, 4



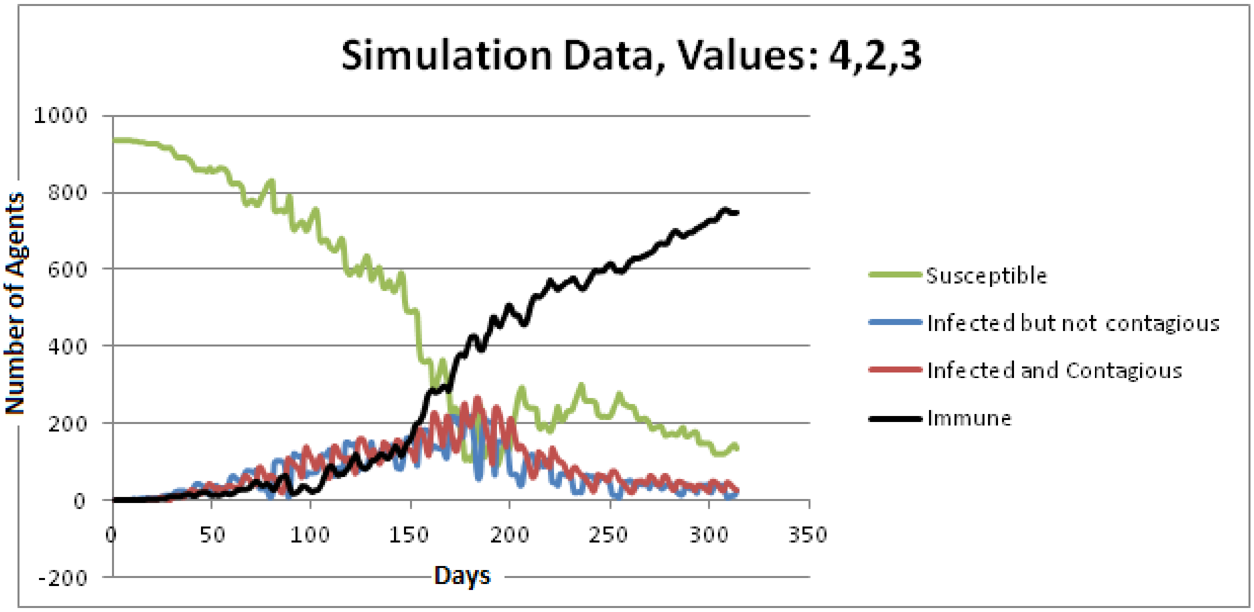
Graph 5: RBA Simulation Data versus Truth Data, Values: 3, 4, 2



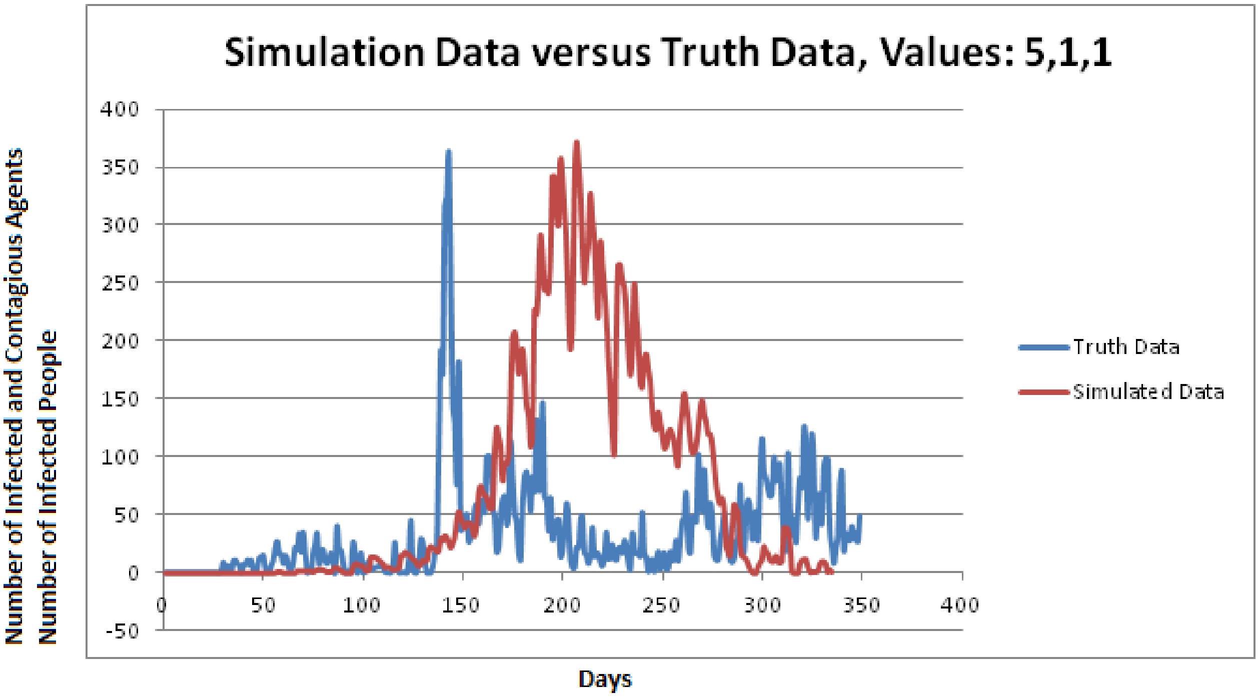
Graph 6: RBA Simulation Data, Values: 3, 4, 2



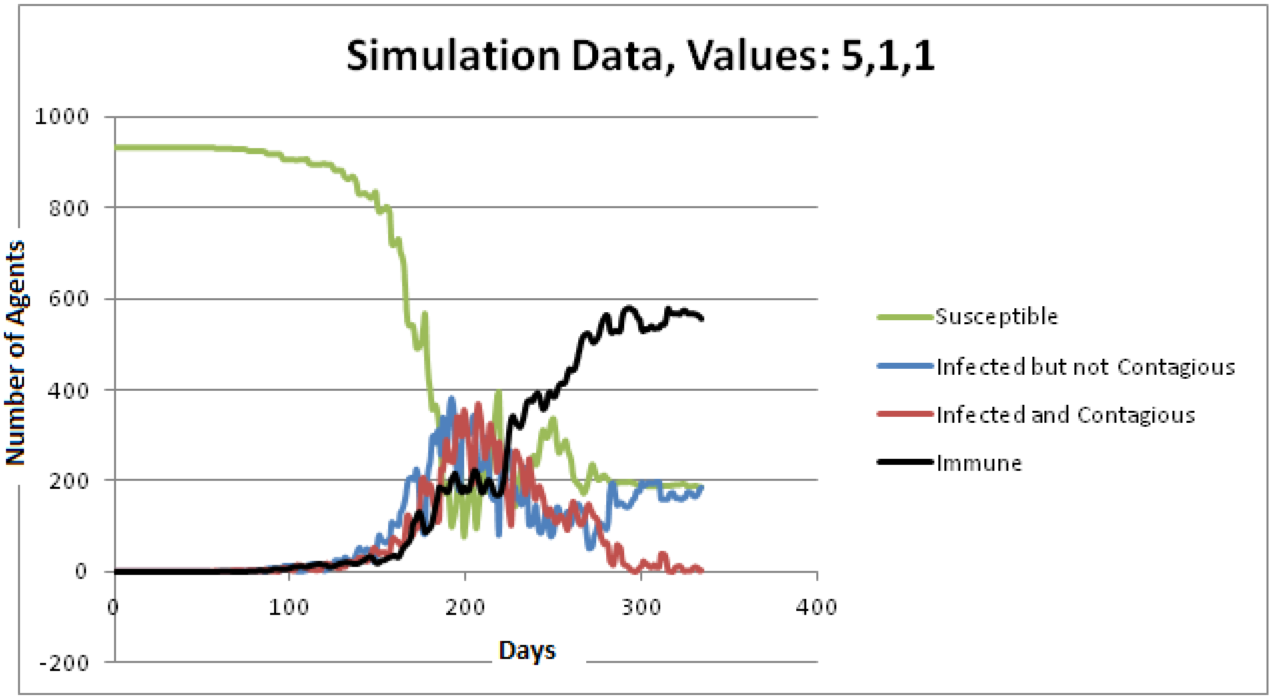
Graph 7: RBA Simulation Data versus Truth Data, Values: 4, 2, 3



Graph 8: RBA Simulation Data, Values: 4, 2, 3



Graph 9: RBA Simulation Data versus Truth Data, Values:



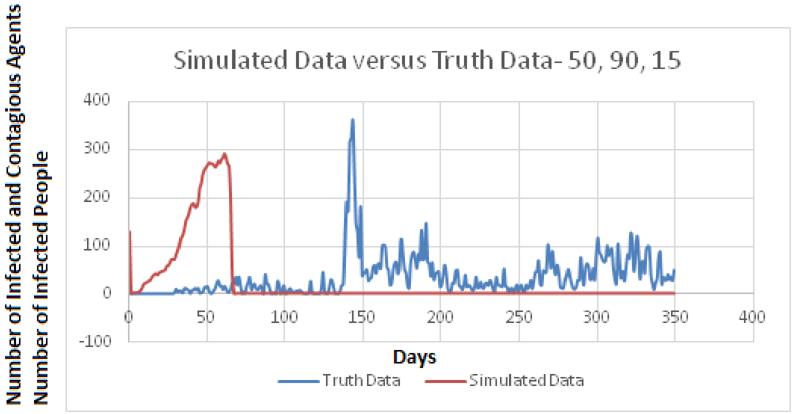
Graph 10: RBA Simulation Data, Values: 5, 1, 1

MBA Sensitivity Analysis and Results

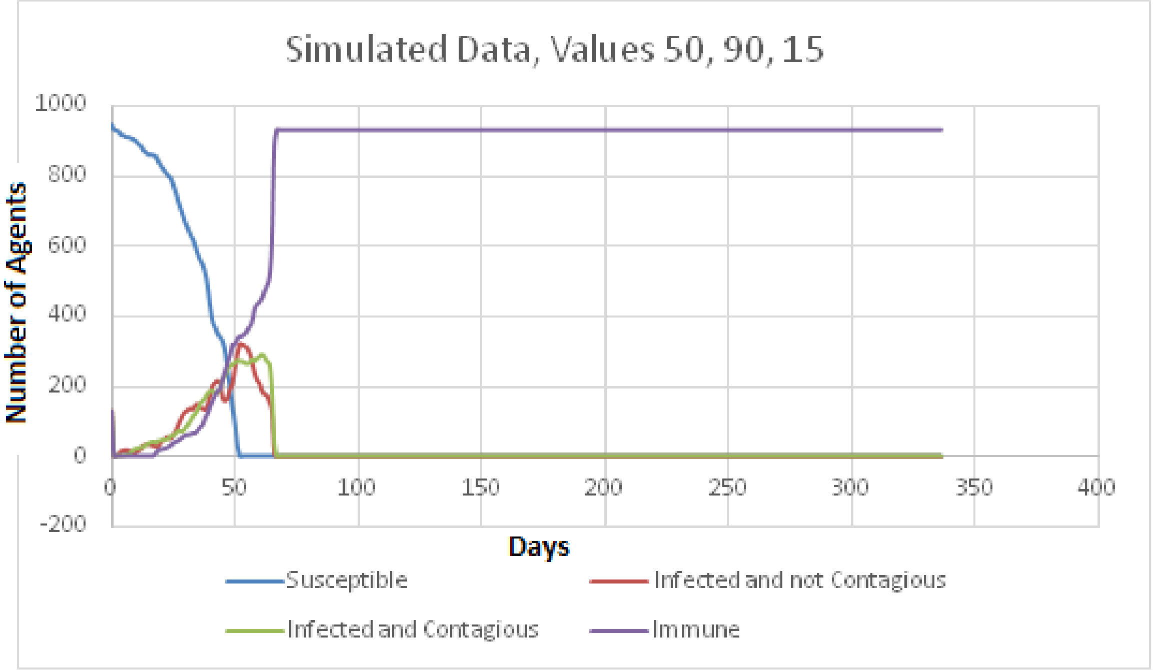
When compared with real data, the MBA algorithm is shown to effectively model the first hump of infection, seen in the plots compared with the H1N1 truth data. This shows that the MBA algorithm does work, and the math based algorithms are a viable way to go when modeling propagation. Using the multiple runs as a sensitivity study it can be seen that the higher the multiplier on both the chance of getting sick and the initial R-Naught, the lower the hump gets. The variable that affects this change in hump the most is the initial R-Naught multiplier. When this multiplier is fairly low, such as the (50, 90, 15) run, the hump begins to become fairly inaccurate, and the same can be said for the (120, 150, 13) run. There were also simulation runs that mostly gave little to no results, such as (200, 40, 4). Seeing this, it is concluded that, if the amount of days contagious is not long enough to assure that the infected will infect someone with its chance of sick multiplier, there will not be a wave of infection. These conclusions have driven the tuning values down to the range keeping in mind that many runs in this range of values will often give results with no hump due to the length of time to insure infection being too short.

Latin Hypercube Random Sampling							
Density Modifier	150			C			Duration of being Infected but not Contagious Modifier
	130				B		A = 4
	111		D				B = 10
	90	E					C = 13
	40					A	D = 14
		50	100	120	145	200	E = 15
	Constant/R-Naught Modifier						

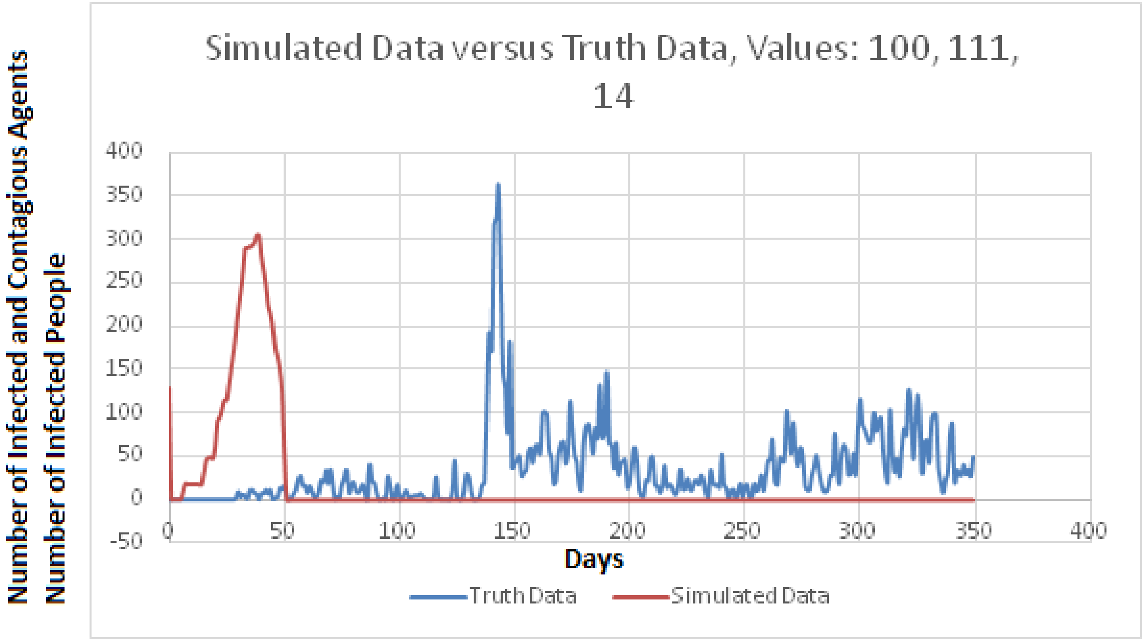
Figure 16: Latin Hypercube Random Sampling used for MBA



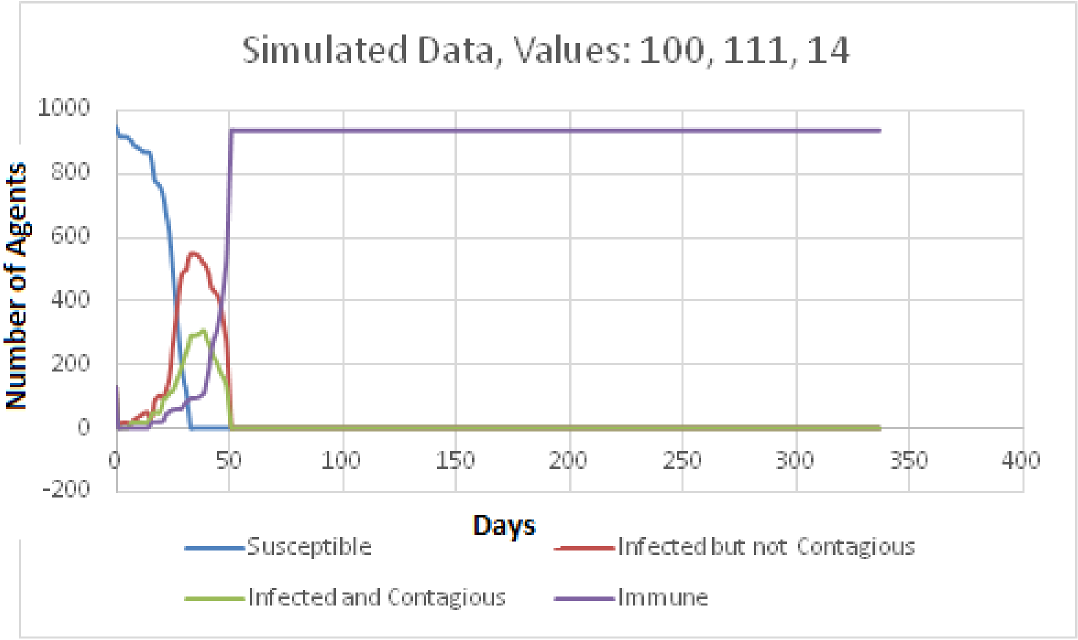
Graph 11: MBA Simulated Data vs. Truth Data, Values: 50, 90, 15



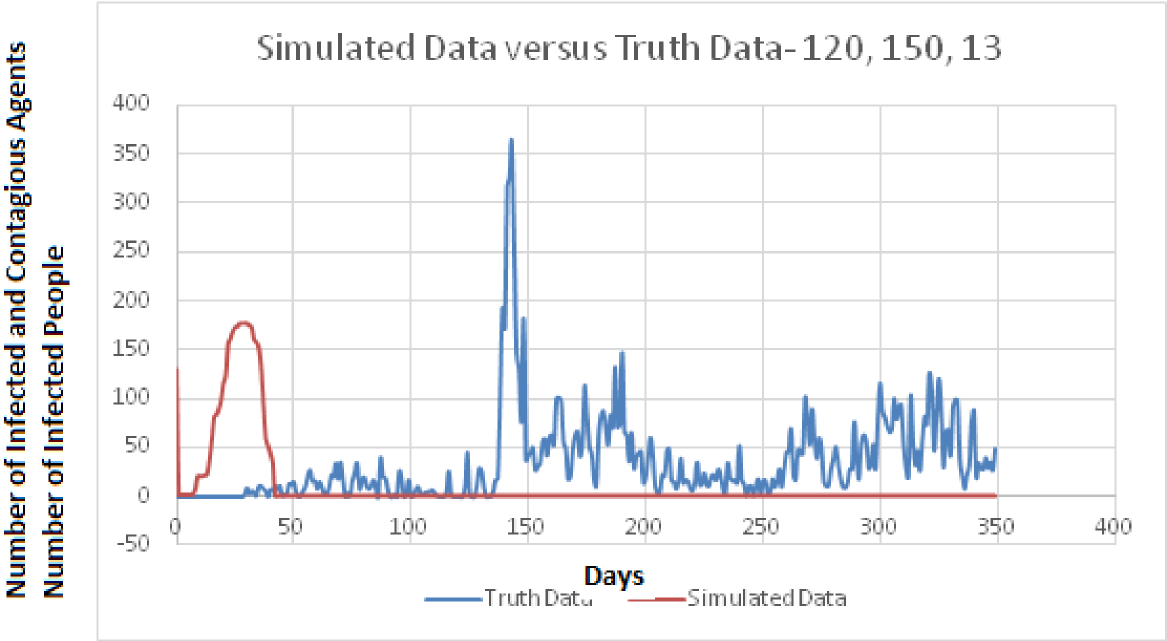
Graph 12: MBA Simulated Data, Values: 50, 90, 15



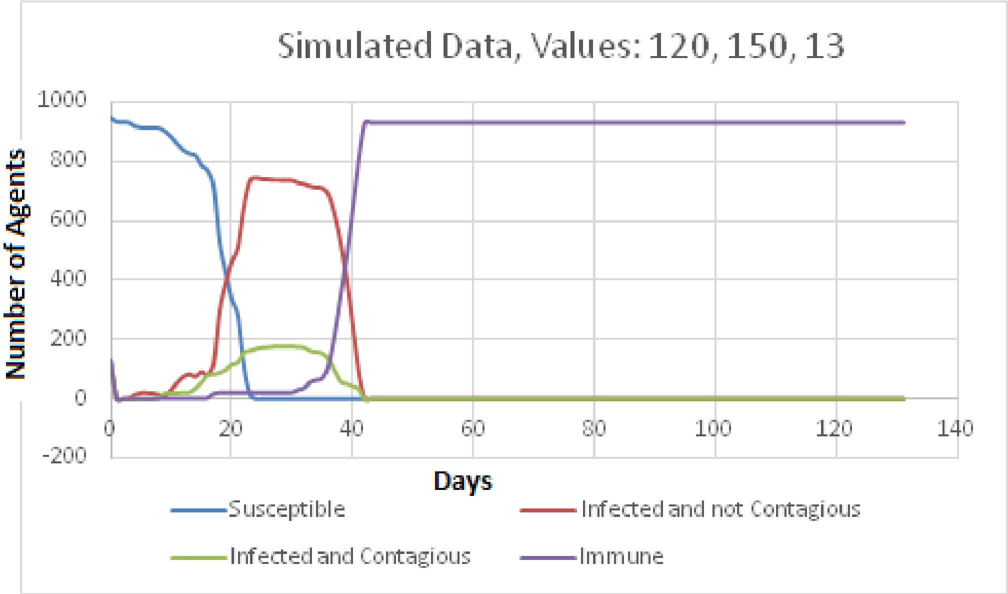
Graph 13: MBA Simulated Data versus Truth Data, Values: 100, 111, 14



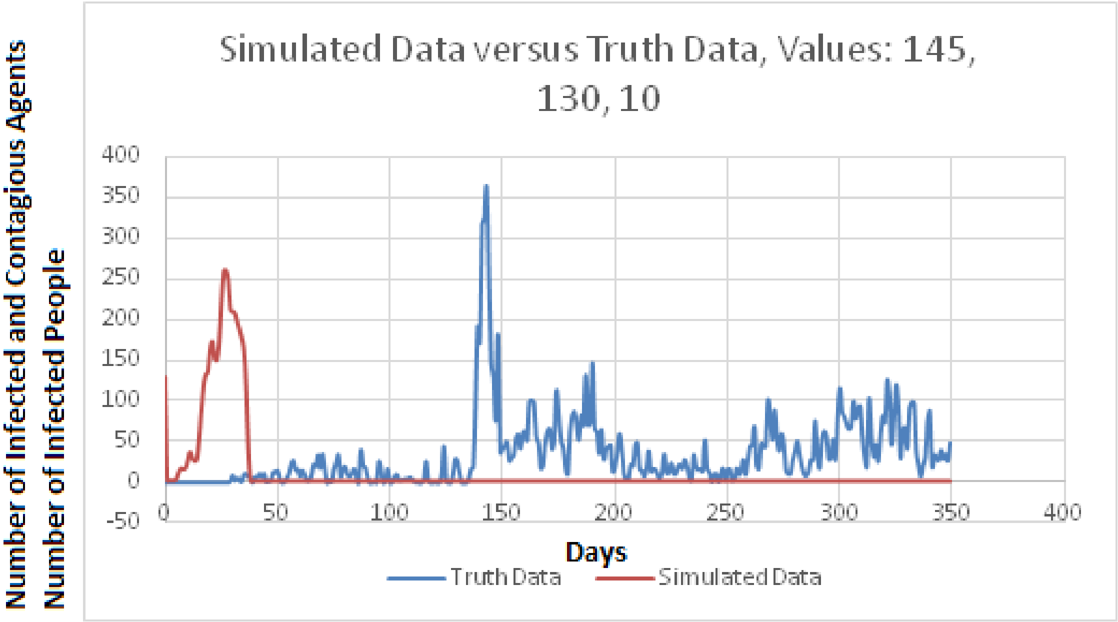
Graph 14: MBA Simulated Data, Values: 100, 111, 14



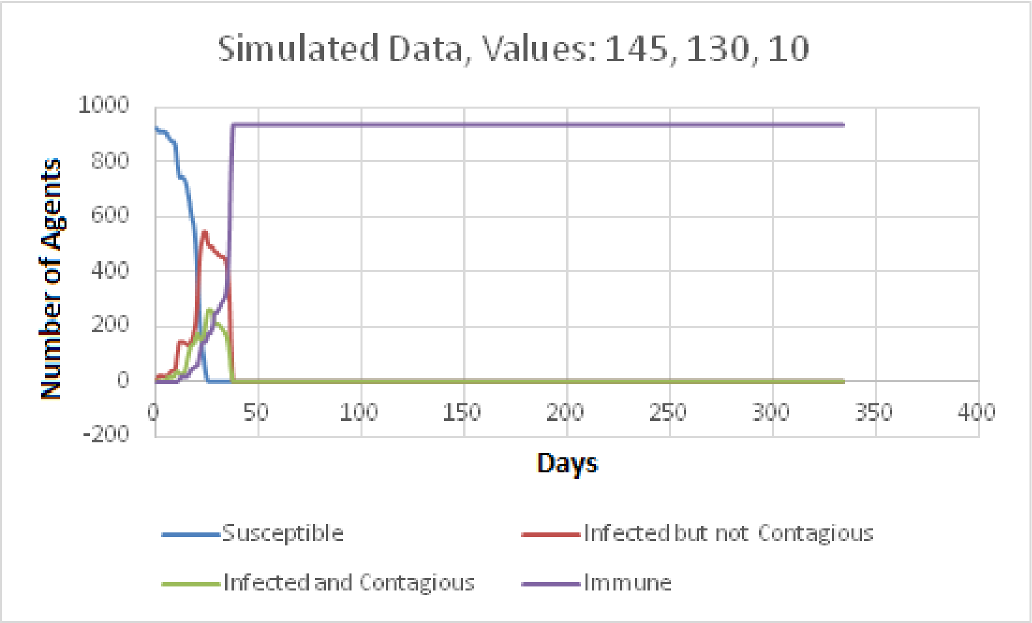
Graph 15: MBA Simulated Data versus Truth Data 120, 150, 13



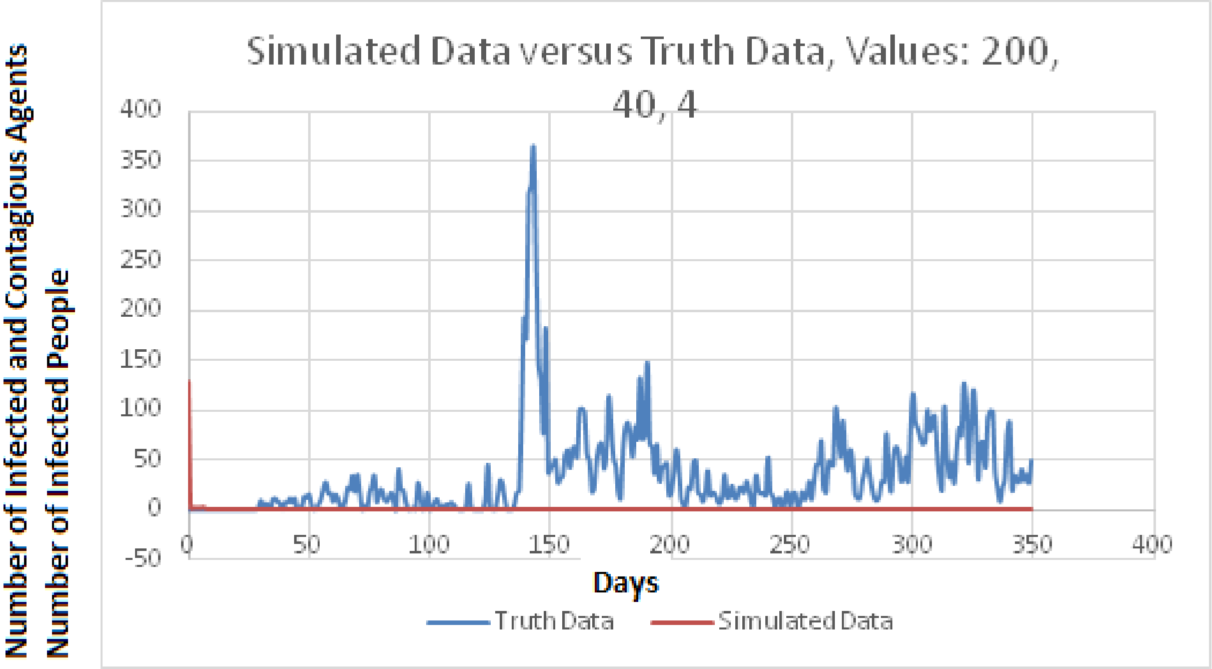
Graph 16: MBA Simulated Data, Values: 120, 150, 13



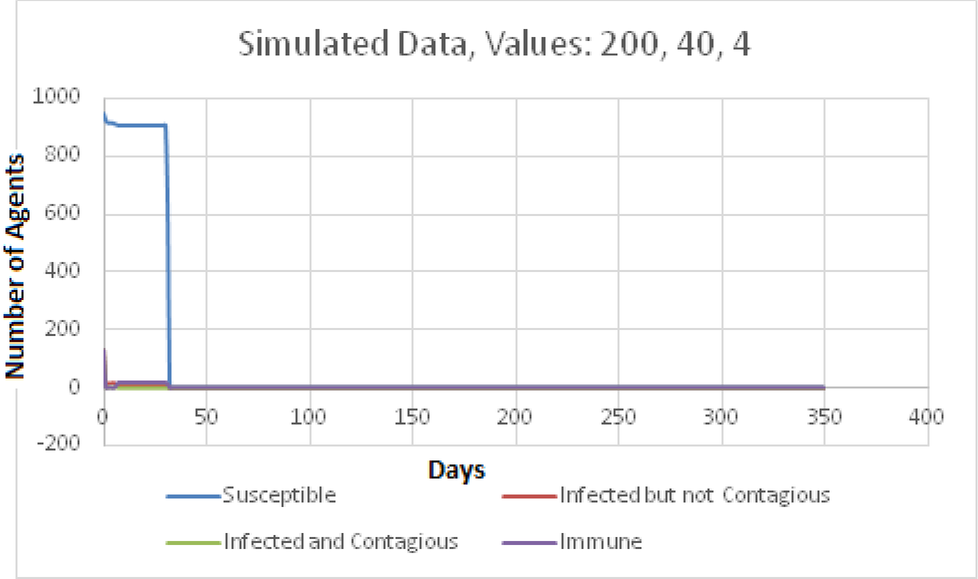
Graph 17: MBA Simulated Data versus Truth Data, Values: 145, 130, 10



Graph 18: MBA Simulated Data, Values: 145, 130, 10



Graph 19: MBA Simulated Data Values: 200, 40, 4



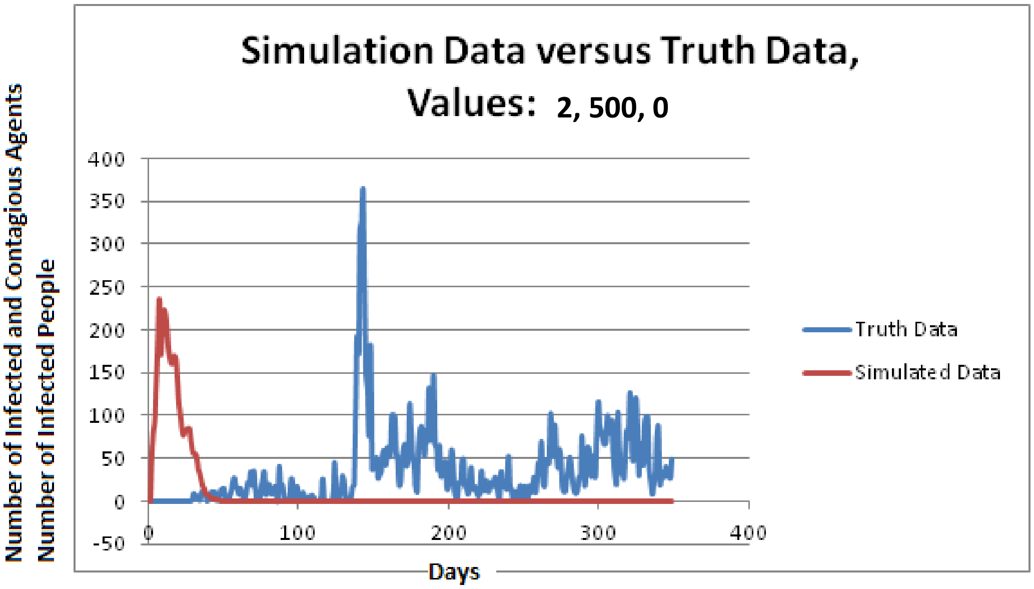
Graph 20: MBA Simulated Data, Values: 200, 40, 4

RNG Sensitivity Analysis and Results

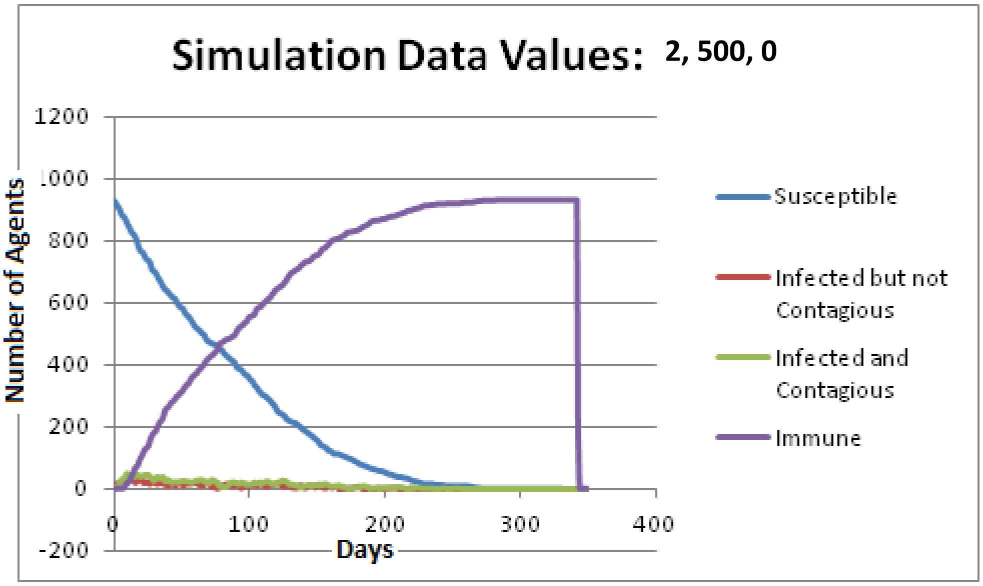
When conducting the sensitivity analysis, it was found that changing the Adjust value and the Range variable affected the rate or the slope of the incline of the graph. When the Adjust variable was raised, and the Range was lowered, the hump(s) became larger and shorter. It also created less of a small, sustained propagation, and instead a large hump. The Duration variable is unique because it makes more humps in the simulation. It also lengthens the humps when increased.

Latin Hypercube Orthogonal Sampling							
Adjust Modifier	40	E					Incubation Days Modifier
	30		D				A = 2
	20			C			B = 4
	10				B		C = 6
	0					A	D = 8
		100	200	300	400	500	E = 10
	Range Modifier						

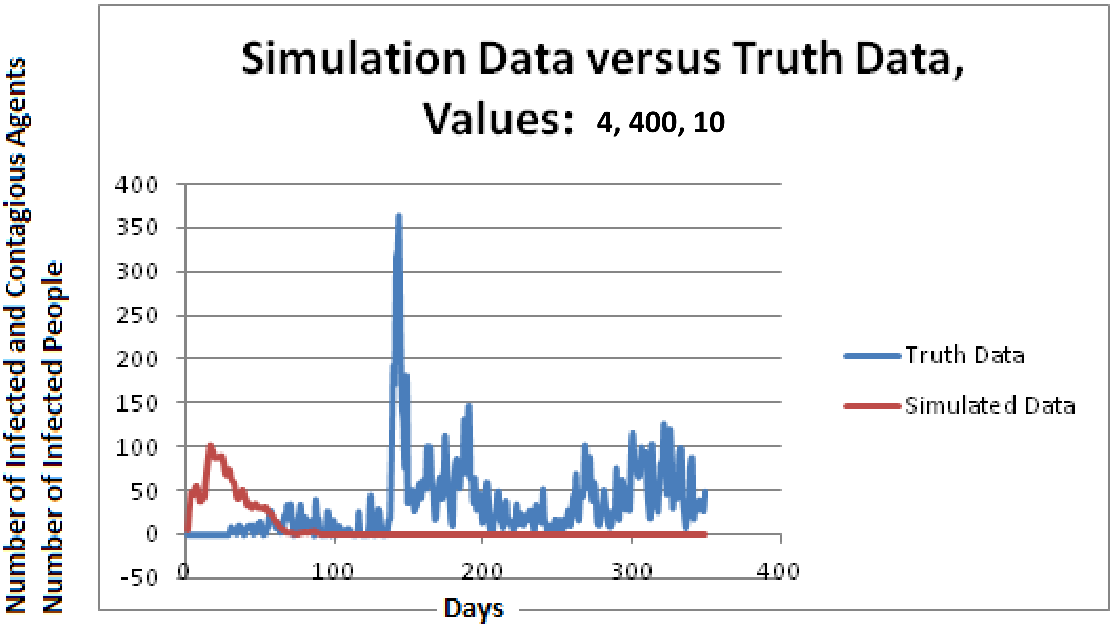
Figure 17: Latin Hypercube Orthogonal Sampling used for RNG



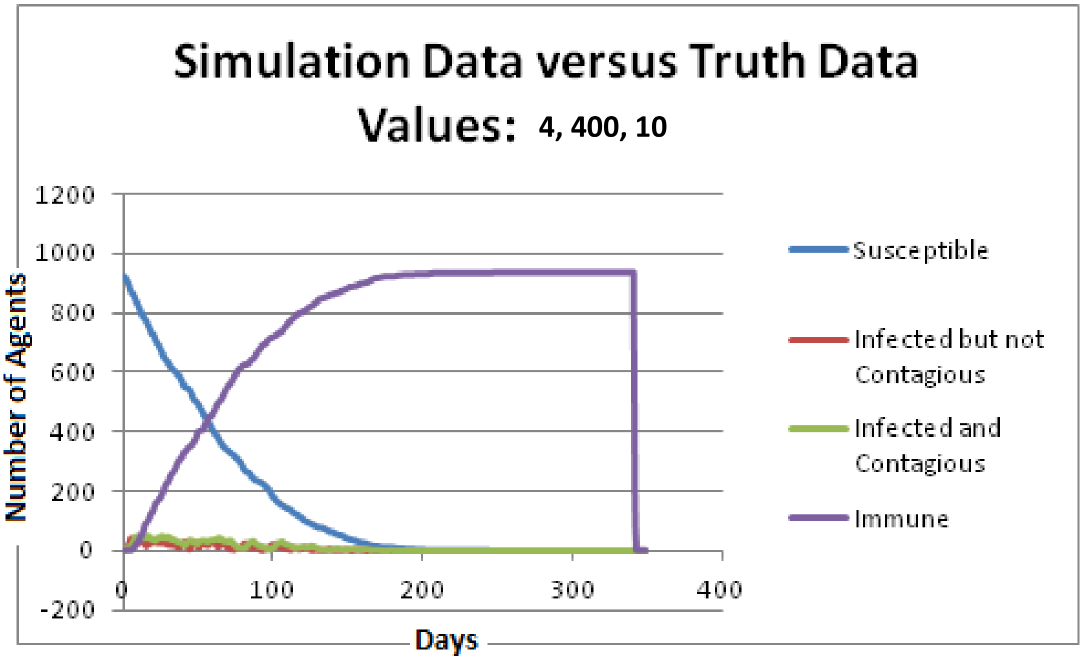
Graph 21: RNG Simulation Data versus Truth Data, Values: 2, 500, 0



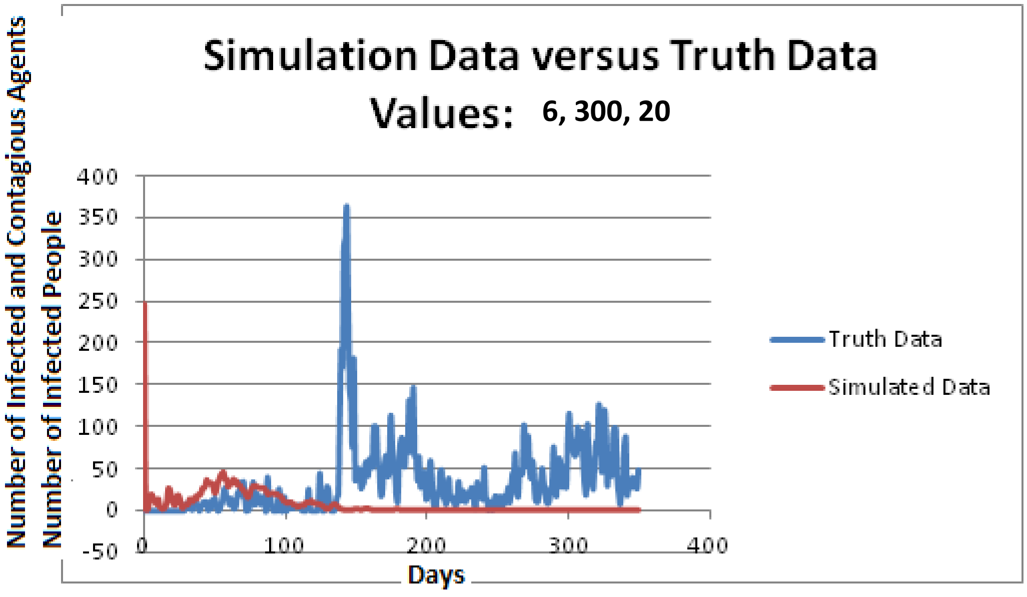
Graph 22: RNG Simulation Data Values: 2, 500, and 0



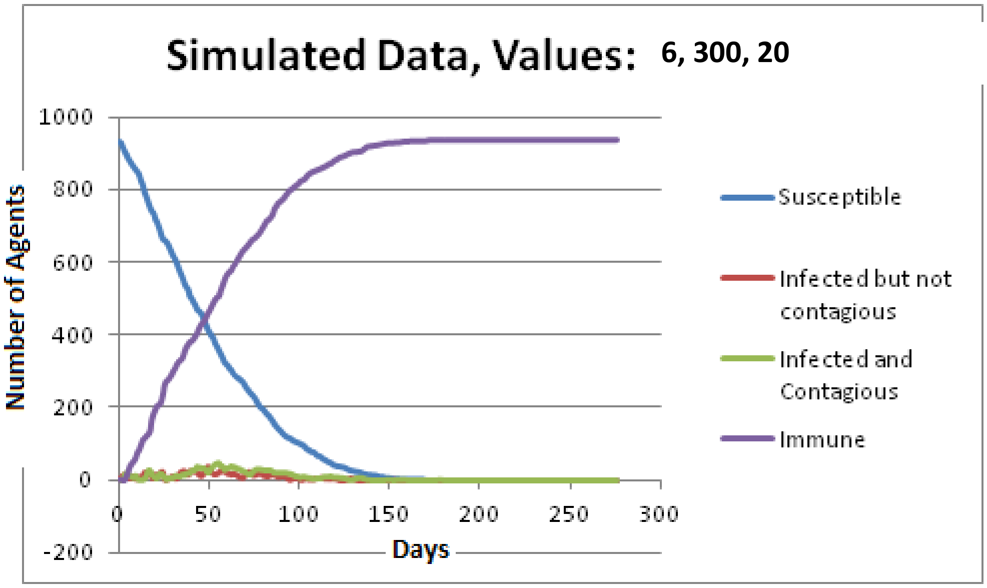
Graph 23: RNG Simulation Data versus Truth Data, Values: 4, 400, 10



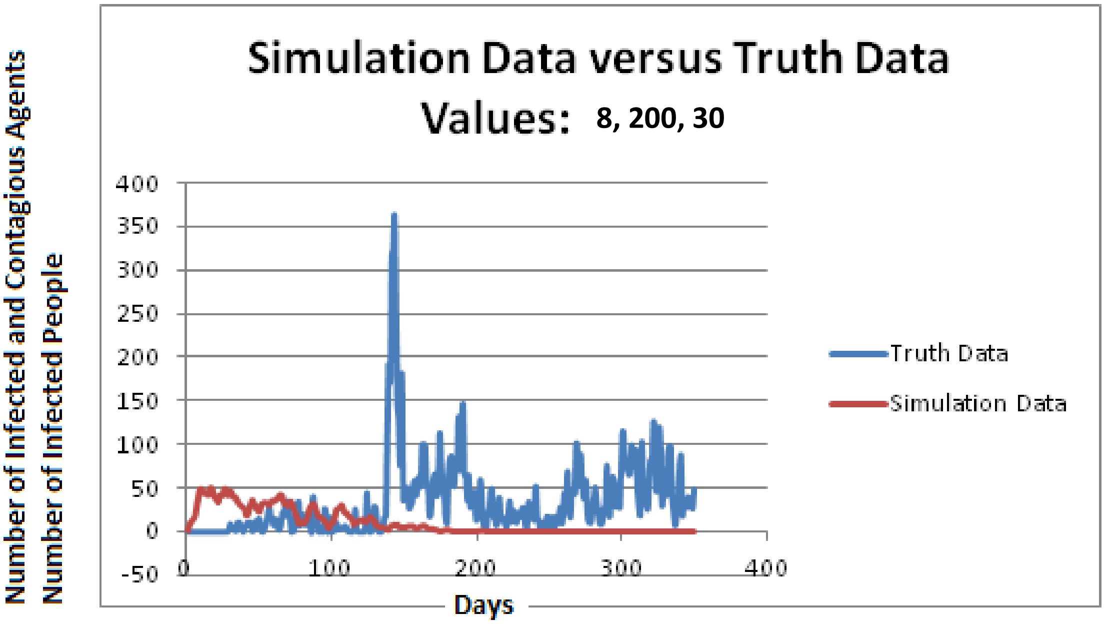
Graph 24: RNG Simulation Data versus Truth Data, Values: 4,400, 10



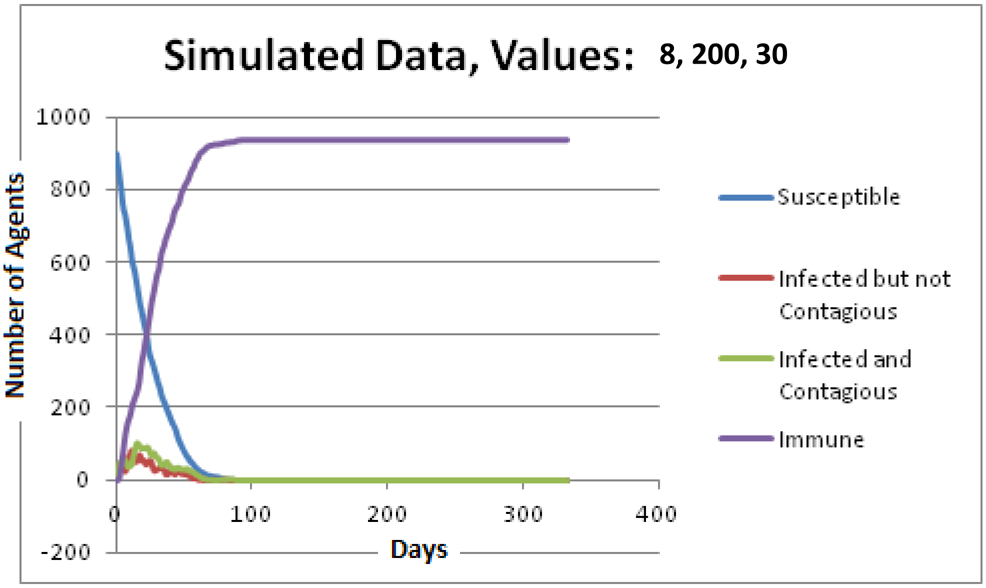
Graph 25: RNG Simulation Data versus Truth Data Values: 6, 300, 20



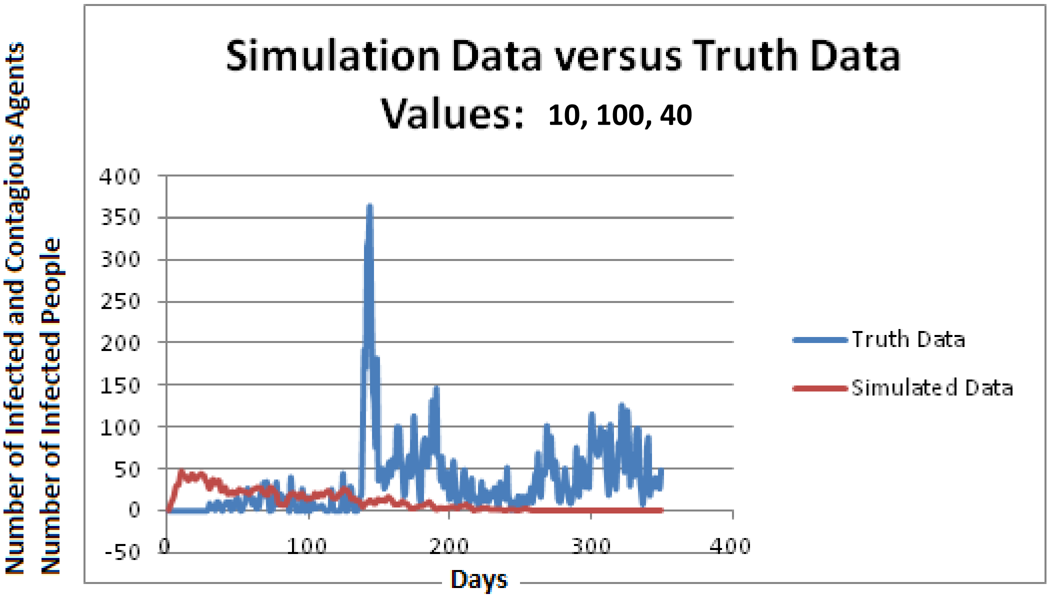
Graph 26: RNG Simulated Data, Values: 6, 300, 20



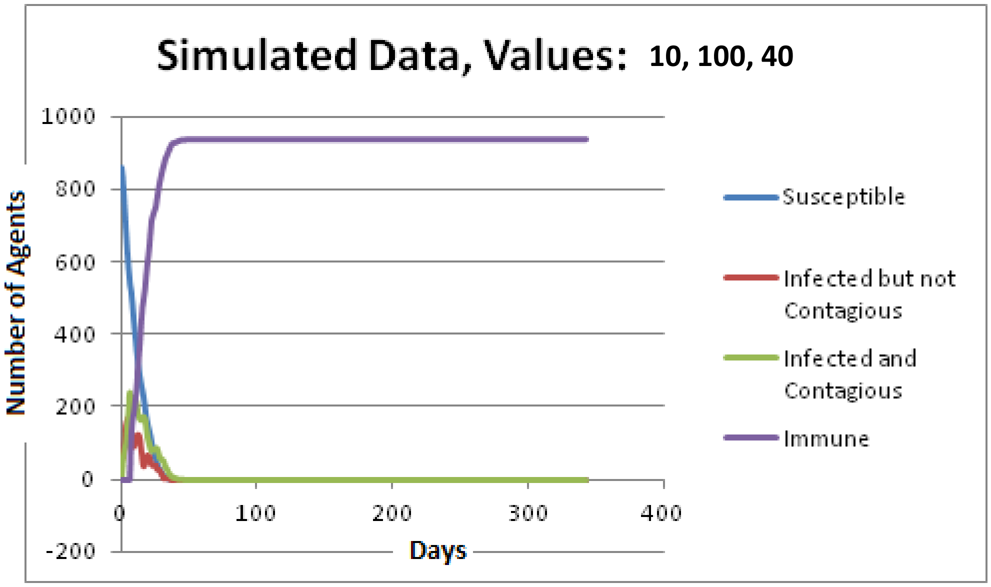
Graph 27: RNG Simulation Data versus Truth Data, Values: 8, 200, 30



Graph 28: RNG Simulated Data, Values: 8, 200, 30



Graph 29: RNG Simulation Data versus Truth Data, Values: 10, 100, 40



Graph 30: RNG Simulated Data, Values: 10, 100, 40

Overall Conclusion

The propagation of the H1N1 virus in 2009 can be modeled using an Agent-Stage Based Model. The use of individual Agents to model a population can be accomplished with the use of Stages where the Agents live life like schedules. The stages also have life like densities to model real life locations. Also there are various possible algorithm approaches for modeling disease propagation. The approaches used were a Rule Based Algorithm, a Math Based Algorithm, and a Random Number Generator Algorithm. These three algorithms all modeled disease propagation, although the Rule Based Algorithm produced the results that most closely matched the H1N1 truth data. This is shown in Graph 1 because of the double hump. The truth data has this double hump although the second hump is much smaller than the first humpThe other two algorithms did not produce this critical double hump. Another conclusion that can be drawn from this study is that R-Naught is a critical factor in disease propagation as evidenced by the sensitivity analysis of the Rule Based Algorithm and the Math Based Algorithm. It is shown that if the R-Naught (the amount of people one person can infect) is high, then more people can become infected. Also density is a fairly important factor in disease propagation because if an area is denser, then the disease is more likely to propagate. This is evidenced by the use of the density variable in all three of the algorithms, and that when the density value is higher (when a Stage area is dense) then the probability of becoming infected is higher. Another conclusion is that although fomites are important in the spread of disease, they are not nearly as important as direct human to human transmission of the disease. Fomites' importance is amplified when there are not many contagious agents. This makes fomites contagious because it can allow the disease to have a head start, or to infect more people during the early stages of the disease. This is evidenced by the Rule Based Algorithm's incorporation of how long a fomite is infectious for modifier. As the fomite was infectious for longer periods of time, then there were more people infected before the first big hump. This is evidenced by the gradual slope before the first big hump as the time for contagious fomites grew. When using a sensitivity study, one can identify significant variables which affect disease propagation, as shown above. Being able to know which variables are the most important can be applied to real world scenarios, and can initiate the incorporation of preventatives to avoid an epidemic early.

This program is also flexible and can accommodate many scenarios through the editing of a simple flat file, as pointed out earlier in this report. The number and types of Stages along with its properties (density) can be easily modified through a flat file, which is read into the database. The total number of Agents as well as their schedules can be modified in this fashion also. This program can easily accommodate any type of algorithm written to model disease propagation, and the already existing algorithms can have their variables adjusted to model other types of diseases.

Most Significant Achievement

The ability to successfully write an Agent based simulation program that allows Agents to travel from Stage to Stage based upon a life style travel schedule is a significant achievement. Also having disease propagation algorithms plug into this software infrastructure and to be able to propagate a virus among the Agents while the Agents are visiting different stages is a major achievement. Another important achievement was the incorporation of fomites into this software and being able to test the importance of different variables within the algorithms, through sensitivity analysis. Being able to know which variables are the most important can be applied to real world scenarios, and can initiate the incorporation of preventatives to avoid an epidemic early. The last achievement was the ability to be able to compare the simulated results to truth data, in this case the fludb.org H1N1 2009 data from the USA to validate and verify this program.

Acknowledgements

Dr. Michael Griesmeyer, Sandia National Laboratories, Providing advice on Sensitivity Analysis and Latin Hypercube

Ms. Marie Kuhne, Sandia National Laboratories, Software development, real time simulation advice and mentoring

Mr. Jim Mims, Albuquerque Academy, Sponsoring Computer Science Teacher who gave advice and direction

Citations

Anderson, Roy M., and Robert M. May. *Infectious Diseases of Humans: Dynamics and Control*. Oxford: Oxford UP, 1991. Print.

Fraser, C., and NC Grassly. "Mathematical Models of Infectious Disease Transmission." *Nat. Rev. Microbiol.* (2008): 477-87. June 2008. Web.

Riley, S. "Large-scale Spatial-transmission Models of Infectious Disease." *Science Mag* (2007): 316. *Science Mag*. June 2007. Web.

Vynnycky, Emilia, and Richard G. White. *An Introduction to Infectious Disease Modeling*. Oxford: Oxford UP, 2010. Print.

Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on strings. Cambridge University Press, Cambridge (2007)

Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: A review. ACM Computing Survey 31, 264–323 (2000) .

National Center for Biotechnology Information (NCBI): Influenza virus resources (2008).

Russell, C.A., Jones, T.C., Barr, I.G., Cox, N.J., Garten, R.J., et al.: The global circulation of seasonal influenza A (H3N2) viruses. Science 320, 340–346 (2008).

Zhang T, Soh SH, Fu X, Lee KK, Wong L, Ma S, Xiao G, Kwoh CK. HPCgen a fast generator of contact networks of
Large urban cities for epidemiological studies. International Conference on Computational Intelligence, Modeling and Simulation. 2009. pp. 198–203.

Martín G, Marinescu MC, E Singh D, Carretero J. EpiGraph: a scalable simulation tool for epidemiological studies. The 2011 International Conference on Bioinformatics and Computational Biology. 2011. pp. 529–536.

Keeling MJ, Eames KT. Networks and Epidemic models. J R Soc Interface. 2005;2(4):295–307. doi: 10.1098/rsif.2005.0051.

Newman M. Spread of epidemic disease on networks. Phys Rev E Stat Nonlin Soft Matter Phys. 2002;66:016128.

Read J, Eames K, Edmunds W. Dynamic social networks and the implications for the spread of infectious disease. J R Soc Interface. 2008;5(26):1001. doi: 10.1098/rsif.2008.0013.

Coelho FC, Cruz OG, Codeco CT. Epigrass: a tool to study disease spread in complex networks. Source Code Biol Med. 2008;3:3. doi: 10.1186/1751-0473-3-3.

Chung F, Lu L. Connected components in random graphs with given expected degree sequences. Annals of Combinatorics. 2002;6:125–145. doi: 10.1007/PL00012580.

Volz E, Meyers L. Susceptible-infected-recovered epidemics in dynamic contact networks. Proc Biol Sci. 2007;274(1628):2925. doi: 10.1098/rspb.2007.1159.

Mossong J, Hens N, Jit M, Beutels P, Auranen K, Mikolajczyk R, Massari M, Salmaso S, Tomba G, Wallinga J. et al. Social contacts and mixing patterns relevant to the spread of infectious diseases. PloS Med. 2008;5(3):e74. doi: 10.1371/journal.pmed.0050074.

Bian L. A conceptual framework for an individual-based spatially explicit epidemiological model. Environment and Planning B. 2004;31(3):381–396. doi: 10.1068/b2833.

Mao L, Bian L. Spatial-temporal transmission of influenza and its health risks in an urbanized area. Computers, Environment and Urban Systems. 2010;34(3):204–215. doi: 10.1016/j.compenvurbsys.2010.03.004.

Carley K, Fridsma D, Casman E, Yahja A, Altman N, Chen LC, Kaminsky B, Nave D. BioWar: scalable agent-based model of bioattacks. IEEE Transactions on Systems, Man and Cybernetics. 2006;36(2):252–265.

Eichner M, Schwehm M, Duerr HP, Brockmann S. The influenza pandemic preparedness planning tool InfluSim. BMC Infect Dis. 2007;7:17. doi: 10.1186/1471-2334-7-17.

Miritello G, Moro E, Lara R. Dynamical strength of social ties in information spreading. Phys Rev E Stat Nonlin Soft Matter Phys. 2011;83(4):045102.

Longini IM, Halloran EM, Nizam A, Yang Y. Containing pandemic influenza with antiviral agents. Am J Epidemiol. 2004;159(7):623–633. doi: 10.1093/aje/kwh092.

Java Software

Agent.java

```
//----------------------------------------------------------------------------------------------------------
// Albuquerque Academy
// Comparison of Disease Propagation Algorithms that Predict the Speed of an Outbreak
// 2013-2014
// Supercomputing Challenge Team 5
//----------------------------------------------------------------------------------------------------------
// Agent.java
//----------------------------------------------------------------------------------------------------------

import java.io.*;
import java.util.Vector;
import java.util.Random;
import java.util.StringTokenizer;

// ===========
// Class Agent
// ===========
public class Agent{

  // Agent Unique Data
  private int    A_ID    = 0;
  private String A_Name  = "none";
  private String A_Stage = "none";
  // Time Data
  private long   T_WallclockStart;           //*
  private int    T_MSinOneHr;                //*
  private int    T_MSinOneDay;                           //*
  private long   T_SimNow;                   //*
  private long   T_SimDayStart;                          //*
  private long   T_SimDayEnd;                            //*
  private long   T_SimHrStart;               //*
  private long   T_SimHrEnd;                 //*
  private int    hrKtr  = 0;                 //*
  private int    dayKtr = 0;                 //*
  // External Data Files
  private String D2_yrSchd_Name [] = new String[365];
  private String D3_daySchd_Name[] = new String[24];
  private int    D3_daySchd_Num [] = new int   [24];
  private String thisClassName = "Agent() ";

  // Constructor-----------------------------------------------
  public Agent(int inA_ID, long inT_WallclockStart, int inT_MSinOneHr ) {
                A_ID            = inA_ID;
        T_WallclockStart = inT_WallclockStart;
        T_SimNow         = System.currentTimeMillis() - T_WallclockStart;
        T_MSinOneHr      = inT_MSinOneHr;
        T_MSinOneDay     = T_MSinOneHr * 24;
```

```
}

//--------------------
// Getters and Setters
//--------------------
public int    getID    () {return ( A_ID    );}
public String getName  () {return ( A_Name  );}
public String getStage () {return ( A_Stage );}
public void   setName  (String inA_Name ){A_Name = inA_Name;}

//-----------------------------
// Factory() calls startMyLife()
//-----------------------------
public int startMyLife(String  inD2_yrSchd_FileName){
  final String D2_yrSchd_FileName = inD2_yrSchd_FileName;

  final Thread myThread = new Thread( new Runnable() {
  public void run() {
    // LIVE FOREVER
         //while( true ){ // 2014-03-10 // TODO: OK ONLY WANT SIM FOR 365 DAYS

                   iTravel( D2_yrSchd_FileName );
         //}
 }});myThread.start(); // goes back and starts myThread
 return 1;
}
//-----------------------------------------------------
private int iTravel(String inD2_yrSchd_FileName) {

   //-------------------------------------------------
   // Read D2 File into D2_yrSchd_Name[] (yr schedule)
   //-------------------------------------------------
   // System.out.println(A_ID+ " START iTravel");
   int numDays = D2_readFile(inD2_yrSchd_FileName);
   if( numDays == 0 ) {
     System.out.println(thisClassName+"iTravel() Start:  D2_readFile() numDays=0  "+A_ID+ " File="+inD2_yrSchd_FileName);
          return(0);
   }
         //
   if( Factory.F_numDays < numDays ) {
                   System.out.println(thisClassName+" CHECK MAX DAYS:  numDays in "+inD2_yrSchd_FileName+" : "+Factory.F_numDays+"  "+ numDays);
                   System.out.println(thisClassName+" CHECK MAX DAYS:  numDays override Factory.F_numDays");
                   Factory.F_numDays = numDays; // NEW 2014-02-05
         }

   //-------------------------------------------------
   // TIME-A: Day Loop, 1->365 (if 365 days scheduled)
   //-------------------------------------------------
    dayKtr =0;
         while( dayKtr < numDays ){
                   //--------------------------------------
         // TIME-B: Get Time Now, DayStart, DayEnd
         //     ... Wait until current day starts
         //-------------------------------------
         T_SimNow      = System.currentTimeMillis() - T_WallclockStart;
```

```
            T_SimDayStart = T_MSinOneDay *  dayKtr   ;
            T_SimDayEnd   = T_MSinOneDay * (dayKtr+1);

            while( T_SimNow < T_SimDayStart ){
                  try{Thread.sleep(T_MSinOneHr);}catch(InterruptedException e2){}
                  T_SimNow = System.currentTimeMillis() - T_WallclockStart;
            }
                        //------------------------------------------------------------
                        // DATA: Read D3 File into D3_daySchd_Name[], D3_daySchd_Num[]
                        //-----------------------------------------------------------
                        D3_readFile ( D2_yrSchd_Name[dayKtr] );

                        //------------------------
                        // TIME-C: Hour Loop, 1->24
                        //------------------------
            hrKtr = 0;
            while( hrKtr < 24 ){
              //--------------------------------------
              // TIME-D: Get Time Now, HrStart, HrEnd
              //     ... Wait until current hour starts
              //--------------------------------------
              T_SimNow     = System.currentTimeMillis() - T_WallclockStart;
              T_SimHrStart = (T_MSinOneHr *  hrKtr   )  + T_SimDayStart;
              T_SimHrEnd   = (T_MSinOneHr * (hrKtr+1))  + T_SimDayStart;

              while( T_SimNow < T_SimHrStart ){
                  try{Thread.sleep(T_MSinOneHr);}catch(InterruptedException e2){}
                  T_SimNow = System.currentTimeMillis() - T_WallclockStart;
              }
              //---------------------------------------------------------
              // DATA: Push Current Hour "Stage" Num and Name to Factory()
              //---------------------------------------------------------
                          Factory.F_AgentStageName[A_ID] = D3_daySchd_Name[hrKtr];
                          Factory.F_AgentStageNum [A_ID] = D3_daySchd_Num [hrKtr];
                          hrKtr++;
                        } // hrKtr

                        dayKtr++;
            } // dayKtr
            return 1;
}
//----------------------------------------------------------------
public int D2_readFile(String inFileName)
{
    int    inTmp      = 0;
    int    readDayKtr = 0;
    try{
        FileInputStream fstream = new FileInputStream(inFileName);
        DataInputStream in      = new DataInputStream(fstream);
        BufferedReader br       = new BufferedReader(new InputStreamReader(in));
        String strLine;
        while ((strLine = br.readLine()) != null){
            StringTokenizer st            = new StringTokenizer(strLine,",");
            inTmp                         = Integer.parseInt(st.nextToken().trim());
                        D2_yrSchd_Name[ readDayKtr++ ] = st.nextToken().trim();
            st  = null;
```

```
      }
      in.close();
          fstream = null;
          in      = null;
          br      = null;
      strLine = null;
    }catch (Exception e){
      System.out.println(thisClassName+"D2_readFile() Exeception: >"+inFileName+"<");
    }
    return readDayKtr;
  } // Read D2 year File

  //----------------------------------------------------------------
  public int D3_readFile(String inFileName)
  {
    int    inTmp      = 0;
    int    readHrKtr = 0;
    try{
      FileInputStream fstream = new FileInputStream(inFileName);
      DataInputStream in      = new DataInputStream(fstream);
      BufferedReader br       = new BufferedReader(new InputStreamReader(in));
      String strLine;
      while ((strLine = br.readLine()) != null){
        StringTokenizer st         = new StringTokenizer(strLine,",");
        inTmp                      = Integer.parseInt(st.nextToken().trim());
                D3_daySchd_Name[readHrKtr] = st.nextToken().trim();
        D3_daySchd_Num [readHrKtr] = Integer.parseInt(st.nextToken().trim());
                readHrKtr++;
        st  = null;
      }
      in.close();
          fstream = null;
          in      = null;
          br      = null;
      strLine = null;
    }catch (Exception e){
      System.out.println(thisClassName+"D3_readFile() Exception: AGENT "+A_ID+" >"+inFileName+"<  dayKtr="+dayKtr);
    }
    return readHrKtr;
  } // Read D3 year File
}
// EOF
//------------------------------------------------------------------------------------------------------
// Agent.java
//------------------------------------------------------------------------------------------------------
```

Factory.java

```
//--------------------------------------------------------------------------------------------------------
// Albuquerque Academy
// Comparison of Disease Propagation Algorithms that Predict the Speed of an Outbreak
// 2013-2014
// Supercomputing Challenge Team 5
//--------------------------------------------------------------------------------------------------------
// Factory.java
//--------------------------------------------------------------------------------------------------------
//
import java.io.*;
import java.util.StringTokenizer;
import java.util.Vector;
//==============
// Class Factory
//==============
public class Factory {

  private Vector<Object> agentVector = new Vector<Object>();
  private Vector<Object> stageVector = new Vector<Object>();

  // CLASS VARIABLES - DATA ARRAYS F_*
  // CONSTANTS
  static double F_AGENT_IMMUNE                        = 100.0; // SOH
  static double F_AGENT_SUSCEPTIBLE                   =  75.0; // SOH
  static double F_AGENT_INFECTED_NOT_CONTAGIOUS_YET =  50.0; // SOH
  static double F_AGENT_INFECTED_CONTAGIOUS         =  25.0; // SOH
  static int    F_AGENT_RNAUGHT                     =     2; // SOH
  static String thisClassName = "Factory() ";

  static String F_Agent_D1_Roles      [];  // set in Factory
  static String F_Agent_D2_yrSchd_Role[];  // set in Factory

  static int    F_AgentFirstDayContagious[];  // set by Algorithm
  static int    F_AgentLastDayContagious [];  // set by Algorithm
  static int    F_AgentRNaughtKtr        [];  // set by Algorithm in constructor
  static int    F_AgentRNaughtInfect     [];  // set by Algorithm in constructor
  static double F_AgentSOH               [];  // Set By Algorithm
  static String F_AgentStageName         [];  // Set By Agent
  static int    F_AgentStageNum          [];  // Set By Agent
  static double F_AgentStageConstant     []; // SOH

  static String F_VirusAlgorithm = "none"; // Set by disprop
  static int    F_SimulationRun  = 0;      // Set by disprop 2014-03-10
  static int              F_SimulationRunB = 0;
  static int              F_SimulationRunC = 0;
  static int              F_SimulationRunD = 0;
  static int              F_SimulationRunE = 0;
  static int              F_SimulationRunF = 0;
  static int              F_SimulationRun_LatinDimension = 0;
  static int    F_RunStage       = 1;            // set by Stage
  static int    F_numAgentRoles  = 0;      // set by Factory
  static int    F_numAgents      = 0;            // Set by D1_AgentIC.txt
  static int    F_numDays        = 0;            // Set by Agent, max days
  static int    F_numStages      = 0;            // set by disprop
  static long   F_WallclockStart = 12345;  // Set by Factory
  static int T_MSinOneHour;       // Set by IN_TimeIC.txt
  static int    DEBUG           = 0;
  //-----------
  // Constructor
  //-----------
  public void Factory() {
          System.out.println("Factory() constructor method Evoked!");
  }
  //------------------------------
  // Mark Simulation Start Time Here
  //------------------------------
  public void initWallclockStartTime(){
          F_WallclockStart = System.currentTimeMillis();
          System.out.println("Factory() initWallclockStartTime() F_WallclockStart="+ F_WallclockStart);
  }
  //-------------------------------------------------------
  // disprop 1st call createAgents(), 2nd call createStages()
  //-------------------------------------------------------
  public Vector createStages(int inT_MSinOneHour, int inStageNum[], String inStageName[], double inStageDensity[], double inStageAdjust[],  int inDEBUG){

           System.out.println("Factory() createStages() Attempt to create "+F_numStages+" Stages");
           T_MSinOneHour    = inT_MSinOneHour;

           for( int i=0; i<F_numStages; i++){
                   //----------------------------------------------------------------
                   // Instantiate Stage Objects, start lightsCameraAction, add to Vector
                   //----------------------------------------------------------------
                   Stage oneStage = new Stage( inStageNum[i], inStageName[i], inStageDensity[i], inStageAdjust[i], F_WallclockStart, T_MSinOneHour );
                   oneStage.lightsCameraAction();
                   stageVector.add((Object) oneStage );
           }
          return (stageVector);
  }
  //-------------------------------------------------------
  // disprop 1st call createAgents(), 2nd call createStages()
  //-------------------------------------------------------
  public Vector createAgents(int inT_MSinOneHour, int inDEBUG){
          //----------------------------------------
          // (1) Get F_numAgents from "D1_AgentIC.txt"
          //----------------------------------------
          F_numAgents = D1_readFile("D1_AgentIC.txt", "");
          System.out.println("Factory() createAgents() Attempt to create "+F_numAgents+" Agents");
          //----------------------
          // (2) Create Data Arrays
          //----------------------
      F_AgentFirstDayContagious = new int    [F_numAgents];
      F_AgentLastDayContagious  = new int    [F_numAgents];
      F_AgentRNaughtKtr         = new int    [F_numAgents];
          F_AgentRNaughtInfect      = new int    [F_numAgents];   // set by Algorithm in constructor
          F_AgentSOH                = new double [F_numAgents];
      F_AgentStageName          = new String [F_numAgents];
      F_AgentStageNum           = new int    [F_numAgents];
```

```
        //----------------------------------------
        // (3) Initialize F_Agent_D2_yrSchd_Role[]
        //----------------------------------------
        F_Agent_D2_yrSchd_Role   = new String [F_numAgents];
        F_numAgents = D1_readFile("D1_AgentIC.txt", "getdata" );
        System.out.println("Factory() Read in "+F_numAgents+" Agents Roles into F_Agent_D2_yrSchd_Role[]");

        //--------------------------------
        // (4) Initialize Data Default, IC
        //--------------------------------
    // FACTORY CONSTANTS ========================
    // BLACK = Factory.F_AGENT_IMMUNE;
    // GREEN = Factory.F_AGENT_SUSCEPTIBLE;
    // BLUE  = Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET;
    // RED   = Factory.F_AGENT_INFECTED_CONTAGIOUS;

        for( int i=0; i<F_numAgents; i++ ){
        F_AgentFirstDayContagious[i] = 365;
        F_AgentLastDayContagious [i] = 0;
                F_AgentRNaughtKtr        [i] = Factory.F_AGENT_RNAUGHT;
            F_AgentRNaughtInfect     [i] = 0;
                F_AgentSOH               [i] = Factory.F_AGENT_SUSCEPTIBLE;
        F_AgentStageName         [i] = "none";
        F_AgentStageNum          [i] = -1;
        }
        //-----------------------------------------------------------
        // (5) Instantiate Agent Objects, startMyLife(), add to Vector
        //-----------------------------------------------------------
        for( int i=0; i<F_numAgents; i++){
                Agent oneAgent = new Agent(i, F_WallclockStart, T_MSinOneHour);
                oneAgent.startMyLife( F_Agent_D2_yrSchd_Role[i] );
                agentVector.add((Object) oneAgent);
    } // LOOP F_numAgents

  System.out.println("Factory() createAgents() DONE!");
  return (agentVector);
}//createAgents()

//--------------------------------------------
// getters and setters
//--------------------------------------------
public static double [] getAgentSOH       (){return (F_AgentSOH      );}
public static String [] getAgentStageName (){return (F_AgentStageName);}
public static int    [] getAgentStageNum  (){return (F_AgentStageNum );}
public Agent getAgent(int i){return (Agent )agentVector.get(i);}
public Stage getStage(int i){return (Stage )stageVector.get(i);}

//---------------------------------------------------------------
public int D1_readFile(String inFileName, String inAction  )
{
   String a = "Factory() Attempt File Read >"+inFileName+"<";
   System.out.println(a);
   String inAgent_D2_RoleName = "";
   int    inAgent_D2_RoleCount = 0;
   int    AgentKtr = 0;
   int    RoleKtr = 0;
   int    outKtr   = 0;
   try{
      FileInputStream fstream = new FileInputStream(inFileName);
      DataInputStream in      = new DataInputStream(fstream);
      BufferedReader br       = new BufferedReader(new InputStreamReader(in));
      String strLine;
      while ((strLine = br.readLine()) != null){
                    RoleKtr++;
      }
      in.close();
   }catch (Exception e){
      System.out.println(thisClassName+ "D1_readFile() EOF >"+inFileName+"<");
   }
   if( inAction == "getdata" ){
          System.out.println(thisClassName+"D1_readFile() RoleKtr="+RoleKtr);
     F_Agent_D1_Roles = new String[RoleKtr];
     F_numAgentRoles  = RoleKtr;
   }
   RoleKtr = 0;

   try{
      FileInputStream fstream = new FileInputStream(inFileName);
      DataInputStream in      = new DataInputStream(fstream);
      BufferedReader br       = new BufferedReader(new InputStreamReader(in));
      String strLine;
      while ((strLine = br.readLine()) != null)   {
                    StringTokenizer st = new StringTokenizer(strLine,","); // comma delimiter
                    //StringTokenizer st = new StringTokenizer(strLine); // std delimiters
                    inAgent_D2_RoleName  = st.nextToken().trim();
                    inAgent_D2_RoleCount = Integer.parseInt(st.nextToken().trim());

                    if( inAction == "getdata" ){
                          F_Agent_D1_Roles[RoleKtr++] = inAgent_D2_RoleName;
                          for( int i=0; i<inAgent_D2_RoleCount; i++ ){
                                   F_Agent_D2_yrSchd_Role[outKtr++] = inAgent_D2_RoleName; // DR-or-Lawyer-or-Engineer
                                   //System.out.println("FACTORY CHECK "+F_Agent_D2_yrSchd_Role[outKtr-1]+" "+outKtr);
                                }
                    }
                    AgentKtr += inAgent_D2_RoleCount;
                    st = null;
      }
      in.close();
    }catch (Exception e){
      System.out.println(thisClassName+"D1_readFile() EOF >"+inFileName+"<");
    }
    System.out.println(thisClassName+"D1_readFile() "+inFileName+" "+" obtained "+AgentKtr+" "+" Agents_D2_Roles");
    return AgentKtr;
 }
//----------------------------------------------------------------
public int aReadTimeIC(String inFileName)
{
   String a = "Factory() Attempt File Read >"+inFileName+"<";
   System.out.println(a);
   String tmpStr;

   try{
      FileInputStream fstream = new FileInputStream(inFileName);
      DataInputStream in      = new DataInputStream(fstream);
      BufferedReader br       = new BufferedReader(new InputStreamReader(in));
      String strLine;
      while ((strLine = br.readLine()) != null)   {
                //StringTokenizer st = new StringTokenizer(strLine);//tab space delimiter
                StringTokenizer st = new StringTokenizer(strLine,",");//comma delimiter
                tmpStr             = st.nextToken().trim(); // READ IN DESCRIPTOR, DON'T USE
                T_MSinOneHour      = Integer.parseInt(st.nextToken().trim());
                System.out.println("Factory() T_MSinOneHour="+T_MSinOneHour);
                st  = null;
      }
      in.close();
    }catch (Exception e){
      System.out.println("Factory() EOF >"+inFileName+"<");
    }
    System.out.println("Factory() aReadTimeIC: inFileName T_MSinOneHour="+T_MSinOneHour);
    return T_MSinOneHour;
  }
}// Factory()
// EOF
//-----------------------------------------------------------------------------------------------------------
// Factory.java
//-----------------------------------------------------------------------------------------------------------
```

Stage.java

```
//------------------------------------------------------------------------------------------------------
// Albuquerque Academy
// Comparison of Disease Propagation Algorithms that Predict the Speed of an Outbreak
// 2013-2014
// Supercomputing Challenge – Team 5
//------------------------------------------------------------------------------------------------------
// Stage.java
//------------------------------------------------------------------------------------------------------

import java.io.*;
import java.util.Vector;
import java.lang.reflect.*;

//============
// Class Stage
//============
public class Stage{

  // VALUES FILLED IN FROM Algorithm INITIAL CONDITIONS
  private int    S_ID      = -1;
  private String S_Name    = "none";
  private double S_Size    = 50.0;
  private double S_Density = 100.0;
  private double S_Adjust  = 0.85;

  // Linked List:q!
  private int    LL_numAgents = 0;
  private int    LL[];

  private int             T_MSinHalfHr    = 0;
  private int             T_MSinThirdHr   = 0;
  private int             T_MSinQuarterHr = 0;
  private long   T_WallclockStart;      //*
  private int    T_MSinOneHr;           //*
  private int    T_MSinOneDay;          //*
  private long   T_SimNow;              //*
  private long   T_SimDayStart;         //*
  private long   T_SimDayEnd;           //*
  private long   T_SimHrStart;          //*
  private long   T_SimHrEnd;            //*
  private int    T_dayKtr = 0;          //*
  private int    hrKtr  = 0;            //*

  private Class    myVirusAlgoClass;
  private Object   myVirusAlgoObj;
  private Method   Algo_init   = null;
  private Method   Algo_start  = null;
  private Method   Algo_stop   = null;
  private Method   Algo_updateFactory = null;
  private String   thisClassName = "Stage() ";

  // Constructor ------------------------------------------------------------
  public Stage(int inStage_ID, String inStageName, double inStageDensity, double inStageAdjust, long inT_WallclockStart, int inT_MSinOneHr ) {
                S_ID            = inStage_ID;
        S_Density        = inStageDensity;
        S_Adjust         = inStageAdjust;
        S_Name           = inStageName;
                T_WallclockStart = inT_WallclockStart;
        T_SimNow         = System.currentTimeMillis() - T_WallclockStart;
        T_MSinOneHr      = inT_MSinOneHr;
        T_MSinOneDay     = T_MSinOneHr * 24;
                T_MSinHalfHr     = (int )((double )T_MSinOneHr / 2.0    );
                T_MSinThirdHr    = (int )((double )T_MSinOneHr / 3.33333);
                T_MSinQuarterHr  = (int )((double )T_MSinOneHr / 4.0    );
                System.out.println(thisClassName+"                constructor: T_WallclockStart "+ T_WallclockStart+
                                                " S_ID = "+S_ID+
                                                " S_Name = "+S_Name+
                                                " S_Adjust = "+S_Adjust+
                                                " S_Density = "+S_Density);

        // Initialize the Linked List
        LL = new int [Factory.F_numAgents];

                try{
                  // Factory has Algorithm Name, used here.
                  myVirusAlgoClass    = Class.forName(Factory.F_VirusAlgorithm);
                  myVirusAlgoObj      = myVirusAlgoClass.newInstance();
                  Method[] allMethods = myVirusAlgoClass.getDeclaredMethods();

         Factory.F_RunStage = 1;

                 for(Method m : allMethods){
                                String mName = m.getName();
                                if( mName == "init" ){
                                    Algo_init = m;
                                    //m.setAccessible(true);
                                }
                                if( mName == "start" ){
                                    Algo_start = m;
                                }
                                if( mName == "stop" ){
                                    Algo_stop = m;
                                }
                                if( mName == "updateFactory" ){
                                    Algo_updateFactory= m;
                                }
                  }
                }catch(Exception eee) {
                                System.out.println("Stage()  Exception:1  Cannot Find Algorithm: "+eee);
                                Factory.F_RunStage = -1;
                }
                System.out.println(thisClassName+"S_ID="+S_ID+" S_Name="+S_Name);
  } // end constructor

  //------------------------------------------------------------
  public int lightsCameraAction(){
        if ( Factory.F_RunStage == -1 ) return 0;
```

```

    try{  // Call method init()
                    Algo_init.invoke(myVirusAlgoObj,S_ID);
    }catch(Exception err1){ System.out.println("Stage() Exception:2 lightsCameraAction()"+err1);}

    final Thread myThread = new Thread( new Runnable() {
    public void run() {
      // Run Thread forever
      //while( Factory.F_RunStage > 0 ){ // Not Used

//----------------------------------
// TIME-A: Day Loop, 1->365 (example)
//----------------------------------
T_dayKtr =0;
while( T_dayKtr < Factory.F_numDays ){
    //--------------------------------------
    // TIME-B: Get Time Now, DayStart, DayEnd
    //     ... Wait until current day starts
    //-------------------------------------
    T_SimNow      = System.currentTimeMillis() - T_WallclockStart;
    T_SimDayStart = T_MSinOneDay * T_dayKtr   ;
    T_SimDayEnd   = T_MSinOneDay * (T_dayKtr+1);

    while( T_SimNow < T_SimDayStart ){
            try{Thread.sleep(T_MSinOneHr);}catch(InterruptedException e2){}
            T_SimNow = System.currentTimeMillis() - T_WallclockStart;
    }
    //-------------------------
    // TIME-C: Hour Loop, 1->24
    //-------------------------
    hrKtr = 0;
    while( hrKtr < 24 ){
       //--------------------------------------
       // TIME-D: Get Time Now, HrStart, HrEnd
       //     ... Wait until current hour starts
       //--------------------------------------
       T_SimNow     = System.currentTimeMillis() - T_WallclockStart;
       T_SimHrStart = (T_MSinOneHr *  hrKtr   )  + T_SimDayStart;
       T_SimHrEnd   = (T_MSinOneHr * (hrKtr+1))  + T_SimDayStart;

       while( T_SimNow < T_SimHrStart ){
            try{Thread.sleep(T_MSinOneHr);}catch(InterruptedException e2){}
            T_SimNow = System.currentTimeMillis() - T_WallclockStart;
       }

                       //-------------------------------------------------------------------------
                       // DATA - Stop Child Algorithm
               try{
                            Algo_stop.invoke(myVirusAlgoObj,S_ID);
                       }catch(Exception err2){System.out.println("Stage() Exception:3 Algo_stop() "+err2);}

                       // DATA - Wait Until All Agents Synchronize
       try{Thread.sleep(T_MSinQuarterHr);}catch(InterruptedException e2){}

                       // DATA - Update Linked List containing Agent Visitors to this Stage
                       updateLL();
```

```

                  // DATA - Start Algorithm, give today's day
              try{
                if( LL_numAgents != 0 ) Algo_start.invoke(myVirusAlgoObj, S_ID, S_Name, S_Density, S_Adjust, LL_numAgents, LL, T_dayKtr, hrKtr);

                  }catch(Exception err3){System.out.println("Stage() Exception:4 Algo_start() Stage_ID="+S_ID+" Stage_Name="+S_Name+"  "+err3);}
                  //---------------------------------------------------------------------------

          hrKtr++;
        } // hrKtr
        T_dayKtr++;
      } // T_dayKtr

        // } // while() // TODO 364 days ??
   }});myThread.start();

   return 1;
 } // lightsCameraAction()---------------

 //--------------------------------------------------
 // Linked List containing only Agents in this Stage
 //--------------------------------------------------
 public void updateLL()
 {
        for( int a=0; a<Factory.F_numAgents; a++ ) {
                LL[a]        = -1;
                LL_numAgents =  0;
        }
        for( int a=0; a<Factory.F_numAgents; a++ ){
        if( Factory.F_AgentStageNum[a] == S_ID){
                        LL[LL_numAgents++] = a;
                }
        }

        //if(LL_numAgents > 0)System.out.println(thisClassName+"updateLL() S_ID="+S_ID+" S_Name="+S_Name+" Agents="+LL_numAgents);
 }

 //-----------------------------------------------------------
 // Setters and Getters
 //-----------------------------------------------------------
 public void setDensity(double  inDensity){S_Density = inDensity;}
 public void setName   (String  inName )  {S_Name    = inName   ;}
 public void setSize   (double  inSize )  {S_Size    = inSize   ;}

 public double getDensity(){return ( S_Density );}
 public int    getID  ()   {return ( S_ID      );}
 public String getName()   {return ( S_Name    );}
 public double getSize()   {return ( S_Size    );}

}
// EOF
//-----------------------------------------------------------------------------------------------------
// Stage.java
```

```
//-------------------------------------------------------------------------------------------------------------------------------------------
```

H1N1_MBA.java

```
//--------------------------------------------------------------------------------------------------------
// Albuquerque Academy
// Comparison of Disease Propagation Algorithms that Predict the Speed of an Outbreak
// 2013-2014
// Supercomputing Challenge – Team 5
//--------------------------------------------------------------------------------------------------------
// H1N1_MBA
//--------------------------------------------------------------------------------------------------------
//
import java.io.*;
import java.util.Random;

// VIRUS ALGORITHM = VA
public class H1N1_MBA{
        private int stageNum = 0;
        private String thisClassName = "H1N1_MBA";
    private boolean INITIAL_CONDITIONS = true;
        private boolean VA_run;
        private double  VA_SOH;
        private double  VA_rndSOH_1;
        private double  VA_rndSOH;
    private double  StageName;
    private int        Algo_RnaughtKtr = 2;
        private Random  VA_rndGenSOH;
    private int     ID_i;
    private int     ID_a;
    private int     ID_b;
        private int               InfectedKtr;
        private int               LL_AgentWell [];
        private int     LL_AgentContag [];;
        private double  ChanceOfSick=0.0;
        private int     StageInfKtr=0;
        // FACTORY CONSTANTS ========================
        // BLACK = Factory.F_AGENT_IMMUNE;
        // GREEN = Factory.F_AGENT_SUSCEPTIBLE;
        // BLUE  = Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET;
        // RED   = Factory.F_AGENT_INFECTED_CONTAGIOUS;

 // COMMON START GROUP ------------------------------------------------------------
 public H1N1_MBA(){
        ;
 }
 public int init( int inStageID){
         System.out.println(thisClassName + " init() stageID="+inStageID);
         VA_run       = true;
         VA_rndGenSOH = new Random();
         return 1;
 }
 public int stop( int inStageID){
         VA_run = false;
         return 1;
 }
 public int updateFactory ( int inStageID){
```

```
                return 1;
    }
    // COMMON END GROUP ---------------------------------------------------------------


    // DISEASE_ALGORITHM ---------------------------------------------------------------
    public int start( int inStageID, String inStageName, double inStageDensity, double inStageAdjust, int inLL_numAgents, int inLL[]
   , int inDayKtr, int inHrKtr){
      VA_run = true;
      ID_i   = 0;
      ID_a   = 0;
      ID_b   = 0;

       InfectedKtr = 0;
      // THE FIRST INFECTED AGENT ASSIGNED HERE -- CUSTOM
      if( INITIAL_CONDITIONS && (inStageID == 1)){
            INITIAL_CONDITIONS = false;
            System.out.println(thisClassName + " start "+inStageID+" Initialized.  Density="+inStageDensity);

            ID_i = inLL[1];
        Factory.F_AgentSOH              [ID_i] = Factory.F_AGENT_INFECTED_CONTAGIOUS; // BLUE
        Factory.F_AgentFirstDayContagious[ID_i] =  inDayKtr+3;
        Factory.F_AgentLastDayContagious [ID_i] =  inDayKtr+7;
            InfectedKtr = 1;
      }

     //while (VA_run){
     // ALGORITHM TIMER
     try{Thread.sleep(20);}catch(InterruptedException e2){}

     // RUN ONE ITERATION NOW FOR AGENTS IN THIS STAGE **************************
     try{
            int ImmuneKtr = 2;
            double AdjustedRnought = 0;
            int myRandom;
            double stageR = 1.0;
            int HealthyKtr = 0;
            int ContagKtr = 0;

            // (1) Find num of Healthy
            for( int a=0; a<inLL_numAgents; a++ ){
                    ID_a = inLL[a];
                    if(Factory.F_AgentSOH [ID_a] == Factory.F_AGENT_SUSCEPTIBLE){
                            HealthyKtr ++;
                    }
            }
            if( HealthyKtr == 0 ) return 0;

            // (2) Create Array of size HealthyKtr
            LL_AgentWell = new int [HealthyKtr];
            HealthyKtr = 0;

            // (3) Put Healthy Agents in Healthy LL Array
            for( int a=0; a<inLL_numAgents; a++ ){
                    ID_a = inLL[a];
                    if(Factory.F_AgentSOH [ID_a] == Factory.F_AGENT_SUSCEPTIBLE){
                            LL_AgentWell[HealthyKtr++] = ID_a;
                    }
            }


            for( int a=0; a<inLL_numAgents; a++ ){
                    ID_a = inLL[a];
                    if(Factory.F_AgentSOH [ID_a] == Factory.F_AGENT_INFECTED_CONTAGIOUS){
                            ContagKtr ++;
                    }
            }
            if( ContagKtr == 0 ) return 0;

            // (2) Create Array of size HealthyKtr
            LL_AgentContag = new int [ContagKtr];
            ContagKtr = 0;

            // (3) Put Healthy Agents in Healthy LL Array
            for( int a=0; a<inLL_numAgents; a++ ){
                    ID_a = inLL[a];
                    if(Factory.F_AgentSOH [ID_a] == Factory.F_AGENT_INFECTED_CONTAGIOUS){
                            LL_AgentContag[ContagKtr++] = ID_a;
                    }
            }

            // (4) Find number of Agents immune and sick
            for( int a=0; a<inLL_numAgents; a++ ){
                    ID_a = inLL[a];

                    if( Factory.F_AgentSOH[ID_a] == Factory.F_AGENT_IMMUNE ){
                            ImmuneKtr++;
                    }

                    if( (Factory.F_AgentSOH [ID_a] == Factory.F_AGENT_INFECTED_CONTAGIOUS)      || ( Factory.F_AgentSOH [ID_a] == Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET)) {
                            InfectedKtr++;
                    }
            }

            if( InfectedKtr == 0 ) return -1;

            double Run = (double) Factory.F_SimulationRun * 0.01;
        double Run2 = (double) Factory.F_SimulationRunB * 0.01;
            int Run3 = Factory.F_SimulationRunC;
            int Run4 = Factory.F_SimulationRunD;
            double InfectDiv = (double) InfectedKtr/inLL_numAgents;
            double ImmuneDiv = (double) (ImmuneKtr/inLL_numAgents)*5;
            //-------------------------------------------------
            // BLACK = Factory.F_AGENT_IMMUNE;
            // GREEN = Factory.F_AGENT_SUSCEPTIBLE;
            // BLUE  = Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET;
            // RED   = Factory.F_AGENT_INFECTED_CONTAGIOUS;
        //-------------------------------------------------

            // (5) calculate adjusted RNAUGHT
            AdjustedRnought= (4.89*Run2)*(1-(ImmuneDiv))-1.5;
            System.out.println("Adjusted Rnought " + AdjustedRnought);
            // (6) calculate percent chance of getting sick
            // LATIN SQUARE VARIABLE
            ChanceOfSick = (inStageDensity*Run*InfectDiv)*10;
            if(ChanceOfSick>100) ChanceOfSick = ChanceOfSick-50;
            System.out.println("IMMUNEDIV =" + ImmuneDiv);
            System.out.println("CHANCEOFSICK=" +ChanceOfSick);
            System.out.println(thisClassName +" 7 DAY="+inDayKtr);
            System.out.println("LLAGENTS "  + inLL_numAgents);
            System.out.println("Infect Ktr "+ InfectedKtr);
            System.out.println("INFECTDIV: " + InfectDiv);
            // (7)
            for( int a=0; a<inLL_numAgents; a++ ){

              ID_a = inLL[a];
                    if(( inDayKtr > Factory.F_AgentLastDayContagious[ID_a]) &&( Factory.F_AgentSOH[ID_a]==Factory.F_AGENT_INFECTED_CONTAGIOUS)) {
                            Factory.F_AgentSOH[ID_a] = Factory.F_AGENT_IMMUNE;
                    }
                    if( inDayKtr > Factory.F_AgentFirstDayContagious[ID_a] && Factory.F_AgentSOH[ID_a]==Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET) {
                            Factory.F_AgentSOH[ID_a] = Factory.F_AGENT_INFECTED_CONTAGIOUS ;
                            }
              if(Factory.F_AgentSOH[ID_a]== Factory.F_AGENT_INFECTED_CONTAGIOUS)  {

            //(8) If Rnought of stage - infectcount of agent 0<x<1, multiply chance of getting sick by this double

                    if( AdjustedRnought < Factory.F_AgentRNaughtInfect[ID_a] ){
                    System.out.println("NOPE");
                     return 0;
                    }

                    // (9) generate random number 1-100
                    myRandom = VA_rndGenSOH.nextInt(100);

                    System.out.println("ChanceOfSick="+ChanceOfSick+"   myRandom="+myRandom + " inStageID="+inStageID);

                    if( (int )ChanceOfSick >= myRandom ) {
                            myRandom = 1;
                            if( HealthyKtr > 2 ){
                                myRandom = VA_rndGenSOH.nextInt(HealthyKtr-1);
                                }                                        int pointer;
                                pointer = LL_AgentWell[myRandom];
                                Factory.F_AgentSOH[pointer] = Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET;
                            Factory.F_AgentFirstDayContagious[pointer] =  inDayKtr+Run3;
                                StageInfKtr++;
                                Factory.F_AgentLastDayContagious [pointer] =  inDayKtr+Run4;

                            //Factory.F_AgentSOH[ID_a] = Factory.F_AGENT_INFECTED_CONTAGIOUS
                            myRandom = 1;
                            if(ContagKtr >2){
                                myRandom = VA_rndGenSOH.nextInt(ContagKtr-1);
                            }
                                int contag;
```

```
                                        contag = LL_AgentContag[myRandom];
                                   Factory.F_AgentRNaughtInfect[contag]++;
                    //  A SICK AGENT KTR IS INCREASED
                              // TODO:  Update above to have the Infector be random
                    }
            }

          } // loop
          // ALGORITHM END ******************************************
  }catch(Exception err){
                    System.out.println(thisClassName + " Exception:  "+err);
  }
 //}// VA_run
 return 1;
 }

}//EOF CLASS
```

H1N1_RBA.java

```java
//----------------------------------------------------------------------------------------------------
// Albuquerque Academy
// Comparison of Disease Propagation Algorithms that Predict the Speed of an Outbreak
// 2013-2014
// Supercomputing Challenge – Team 5
//----------------------------------------------------------------------------------------------------
// H1N1_RBA.java
//----------------------------------------------------------------------------------------------------
// For performing a Lating Hypercube sampling the following variables are used:
//          Factory.F_SimulationRun
//          Factory.F_SimulationRunB
//          Factory.F_SimulationRunC
//----------------------------------------------------------------------------------------------------

import java.io.*;
import java.util.Random;

// VIRUS ALGORITHM = VA
public class H1N1_RBA{

 private int stageNum = 0;
 private String thisClassName = "H1N1_RBA";
 private boolean INITIAL_CONDITIONS = true;
 private boolean VA_run;
 private double  VA_SOH;
 private double  VA_rndSOH1;
 private double  VA_rndSOH;
 private double  StageName;
 private Random  VA_rndGenSOH;
 private int     ID_i;
 private int     ID_a;
 private int     ID_b;
 private int     ID_c;
 private int     sickKtr = 0;
 // FOMITE START
 private int     stageFomiteKtr = 0;
 private int     stageFomiteThreshold = 80;
 private int     stageFomite_LastHrContagious = 0;
 // FOMITE END

          // FACTORY CONSTANTS ========================
          // GREEN = Factory.F_AGENT_SUSCEPTIBLE;
          // BLUE  = Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET;
          // RED   = Factory.F_AGENT_INFECTED_CONTAGIOUS;
          // BLACK = Factory.F_AGENT_IMMUNE;

  // COMMON START GROUP -----------------------------------------------------------------
  public H1N1_RBA(){
          ;
  }
  public int init( int inStageID){
            System.out.println(thisClassName + " init() stageID="+inStageID);
```

```
                VA_run       = true;
                VA_rndGenSOH = new Random();
                return 1;
  }
  public int stop( int inStageID){
                VA_run = false;
                return 1;
  }
  public int updateFactory ( int inStageID){
                return 1;
  }
  // COMMON END GROUP ---------------------------------------------------------------


  // DISEASE_ALGORITHM ---------------------------------------------------------------
  public int start( int inStageID, String inStageName, double inStageDensity, double inStageAdjust, int inLL_numAgents, int inLL[]
, int inDayKtr, int inHrKtr){

    if( inLL_numAgents == 0 ){
              // IF NO ONE IS IN THIS STAGE, RETURN
              //System.out.println(thisClassName+"S_ID="+inStageID+" S_Name="+inStageName+" Empty Stage");
              return 0;
    }
    VA_run = true;
    ID_i   = 0;
    ID_a   = 0;
    ID_b   = 0;

    //---------------------------------------------------
    // THE FIRST INFECTED AGENT ASSIGNED HERE -- THE SEED
    //---------------------------------------------------
    if( INITIAL_CONDITIONS && (inStageID == 1))
    {
               INITIAL_CONDITIONS = false;
               ID_i = inLL[0];
               System.out.println(thisClassName + " start() S_ID="+inStageID+" S_Name="+inStageName+
                                                                      " INITIAL_CONDITIONS:  Agent ID Infected="+ID_i+"
NumAgentsThisStage="+inLL_numAgents+"\n");

        Factory.F_AgentSOH              [ID_i] = Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET; // BLUE
        Factory.F_AgentFirstDayContagious[ID_i] =   3 + inDayKtr;
        Factory.F_AgentLastDayContagious [ID_i] =  14 + inDayKtr;
    }

   //while (VA_run){ // FUTURE USE
   // ALGORITHM TIMER
   try{Thread.sleep(20);}catch(InterruptedException e2){}


   //------------------------------------------------------------
   // RUN ONE ITERATION FOR THIS HOUR, ON AGENTS IN THIS STAGE
   // ALGORITHM START ********************************************
   //------------------------------------------------------------
   try{
     for( int a=0; a<inLL_numAgents; a++ ){
              ID_a = inLL[a];

              // SANITY CHECK
              //if( ID_a == 1 ) System.out.println("AGENT 1 at stage "+inStageID + " day="+inDayKtr+"  hr="+inHrKtr);

              //--------------------------------------------
              // SOME IMMUNE MAY AGENTS GET SUSCEPTIBLE AGAIN
              //--------------------------------------------
              if((Factory.F_AgentSOH[ID_a] == Factory.F_AGENT_IMMUNE) &&   // BLACK
                (VA_rndGenSOH.nextInt(365) < inDayKtr ) &&
                (sickKtr > 100))
              {
                      Factory.F_AgentSOH        [ID_a] = Factory.F_AGENT_SUSCEPTIBLE; // GREEN
                      Factory.F_AgentRNaughtKtr [ID_a] = 1;
                      sickKtr=0;
                      //Factory.F_AgentFirstDayContagious [ID_a] = 365;
              }

              //-------------------------------------------
              // INFECTED NOT CONTAGIOUS BECOME CONTAGIOUS
              //-------------------------------------------
              if((Factory.F_AgentSOH[ID_a] == Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET) &&  // BLUE
                (inDayKtr >Factory.F_AgentFirstDayContagious[ID_a]                      ))
              {
                      Factory.F_AgentSOH                 [ID_a] = Factory.F_AGENT_INFECTED_CONTAGIOUS; // RED
                      //Factory.F_AgentFirstDayContagious [ID_a] = 365;
              }

              //-------------------------------------------
              // CONTAGIOUS BECOME IMMUNE
              //-------------------------------------------
          if((Factory.F_AgentSOH[ID_a] == Factory.F_AGENT_INFECTED_CONTAGIOUS) && // RED
            (inDayKtr > Factory.F_AgentLastDayContagious[ID_a] ))
              {
                      Factory.F_AgentLastDayContagious[ID_a] = 0;
                      Factory.F_AgentSOH             [ID_a] = Factory.F_AGENT_IMMUNE;  // BLACK
              }

              //-------------------------------------------
              // FOUND CONTAGIOUS AGENT, READY TO INFECT...
              //-------------------------------------------
          if((Factory.F_AgentSOH       [ID_a] == Factory.F_AGENT_INFECTED_CONTAGIOUS)  &&
            (Factory.F_AgentRNaughtKtr[ID_a] > 0                                ))
              {
                      sickKtr++;
                      // FOMITE START ------------------------------------
              stageFomiteKtr++;
              stageFomite_LastHrContagious  = inHrKtr+4;
              // FOMITE END --------------------------------------

                      for( int b=0; b<inLL_numAgents; b++ )
                      {
                              ID_b = inLL[b];
                              //----------------------------------
                              // FOUND AGENT THAT CAN BE INFECTED
                              //----------------------------------
                      if (Factory.F_AgentSOH[ID_b] == Factory.F_AGENT_SUSCEPTIBLE)
```

```
                    {
                    VA_rndSOH1 = (double)(VA_rndGenSOH.nextInt(100));
                            VA_rndSOH  = VA_rndSOH1 * (1.0 - 1.0/(inStageDensity+1.0));
                    if( VA_rndSOH > 90.0 ) // THRESHOLD
                            {
                            if( Factory.F_AgentRNaughtKtr[ID_a] > 0) // default rnaught=2
                                    {
                                    Factory.F_AgentRNaughtKtr[ID_a]--;
                                    // ALSO... Reduce RNaught 1.5, .
                            if(VA_rndGenSOH.nextInt(365) < inDayKtr )  // as years goes on mutation increases
                                            {
                                            if (VA_rndGenSOH.nextInt(100) > 80+ Factory.F_SimulationRunB )
                                                    {
                                                    Factory.F_AgentRNaughtKtr[ID_a]--;
                                                    }
                                            }

                                    System.out.println(thisClassName+
                                            " ["+Factory.F_Agent_D2_yrSchd_Role[ID_a]+
                                            "] ID="+ID_a+
                                            " <INFECTED> ["+ Factory.F_Agent_D2_yrSchd_Role[ID_b]+
                                            "] ID="+ID_b+
                                            "_____"+inLL_numAgents+
                                            " Agents in "+inStageName+" "+inStageID+
                                            " Density="+inStageDensity+
                                            "    DAY="+inDayKtr+
                                            "   rndSOH="+VA_rndSOH+
                                            "   rndSOH1="+VA_rndSOH1);

                                    try{Thread.sleep(5);}catch(InterruptedException e2){}

                                            //------------------------------------------------
                                            // GETS INFECTED, SET FIRST AND LAST DAY CONTAGIOUS
                                            //------------------------------------------------
                                    Factory.F_AgentSOH[ID_b]
                                                    = Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET;
                                    Factory.F_AgentFirstDayContagious[ID_b]
                                                    = inDayKtr + 3 + Factory.F_SimulationRun;
                                    Factory.F_AgentLastDayContagious [ID_b]
                                                    = inDayKtr + 7 + VA_rndGenSOH.nextInt(10) + Factory.F_SimulationRun ;
                                    }
                            }
                    }
            } // One that gets infected inLL_numAgents
    } // Contagious ready to infect

// FOMITE START
for( int c=0; c<inLL_numAgents; c++ )
    {
    ID_c = inLL[c];

    if((Factory.F_AgentSOH[ID_c] == Factory.F_AGENT_SUSCEPTIBLE) &&
        (stageFomiteKtr > 0)                                &&
        (VA_rndGenSOH.nextInt(100)> stageFomiteThreshold)      &&
            (inDayKtr <= stageFomite_LastHrContagious))
        {
            if((inDayKtr < 60) || (inDayKtr > 300))
            {
                System.out.println("***** Fomite Infected Agent "+ID_c);
                stageFomiteKtr--;
                Factory.F_AgentSOH[ID_a] = Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET;
                Factory.F_AgentFirstDayContagious[ID_a] = inDayKtr + 3;
                Factory.F_AgentLastDayContagious [ID_a] = inDayKtr + 7;
                stageFomite_LastHrContagious--;
            }
        }
    } // FOMITE END


   }// AgentLL
  }catch(Exception err){
                System.out.println(thisClassName + " Exception:1 start()"+err);
  }
  // ALGORITHM END *********************************************


 //}// VA_run // FUTURE USE
 return 1;
 }

}//EOF CLASS
//-----------------------------------------------------------------------------------------------------------
// H1N1_RBA.java
//-----------------------------------------------------------------------------------------------------------
```

H1N1_RNG.java

```
//-----------------------------------------------------------------------------------------------------
// Albuquerque Academy
// Comparison of Disease Propagation Algorithms that Predict the Speed of an Outbreak
// 2013-2014
// Supercomputing Challenge – Team 5
//-----------------------------------------------------------------------------------------------------
// H1N1_RNG.java
//-----------------------------------------------------------------------------------------------------

import java.io.*;
import java.util.Random;
import java.util.*;
import java.lang.Object.*;
import java.lang.Math.*;
import java.lang.*;

// VIRUS ALGORITHM = VA
public class H1N1_RNG{
        private int     stageNum          = 0;
        private String  thisClassName      = "H1N1_RNG  ";
    private boolean INITIAL_CONDITIONS = true;
        private boolean VA_run;
        private double  VA_SOH;
        private double  VA_rndSOH_1;
        private double  VA_rndSOH;
    private double  StageDensityMultiplier;
    private double  StageName;
    private int        Algo_RnaughtKtr = 2;
        private Random  VA_rndGenSOH;
    private int     ID_a;
    private int     ID_b;
        private int numWell;
        private double pct;
        private double a1;
        private double a2;
        private double a3;
        private double a4;
        private int    Range;
        // FACTORY CONSTANTS ========================
        // BLACK = Factory.F_AGENT_IMMUNE;
        // GREEN = Factory.F_AGENT_SUSCEPTIBLE;
        // BLUE  = Factory.F_AGENT_INFECTED_NOT_INFECTED_YET;
        // RED   = Factory.F_AGENT_INFECTED_CONTAGIOUS;

 // COMMON START GROUP -------------------------------------------------------------
 public H1N1_RNG(){
        ;
 }
 public int init( int inStageID){
         System.out.println(thisClassName + " init() stageID="+inStageID);
         VA_run       = true;
         VA_rndGenSOH = new Random();
         return 1;
 }
 public int stop( int inStageID){
                VA_run = false;
                return 1;
 }
 public int updateFactory ( int inStageID){
                return 1;
 }
 // COMMON END GROUP ---------------------------------------------------------------


 // DISEASE_ALGORITHM ---------------------------------------------------------------
 public int start( int inStageID, String inStageName, double inStageDensity, double inStageAdjust, int inLL_numAgents, int inLL[] , int inDayKtr, int inHrKtr){
    VA_run = true;
    ID_a   = 0;
    ID_b   = 0;
    //StageDensityMultiplier = (1.0 / inStageDensity);

    // THE FIRST INFECTED AGENT ASSIGNED HERE -- CUSTOM
    if( INITIAL_CONDITIONS && (inStageID == 8)){
               INITIAL_CONDITIONS = false;
               System.out.println(thisClassName + " start STAGE "+inStageID+" Initialized.  Density="+inStageDensity+" inDayKtr"+inDayKtr);

        if( inLL_numAgents >= 1 ){
                               ID_a = inLL[0];
               Factory.F_AgentSOH              [ID_a] = Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET; // BLUE
               Factory.F_AgentFirstDayContagious[ID_a] = inDayKtr +  2;
               Factory.F_AgentLastDayContagious [ID_a] = inDayKtr + 14;
                               System.out.println(thisClassName + " FIRST INFECTED_NOT_CONTAGIOUS_YET AGENT: "+ID_a);
               }
    }

   //while (VA_run){
   // ALGORITHM TIMER
   try{Thread.sleep(20);}catch(InterruptedException e2){}

   // CONVERT NOT_CONTAGIOUS_YET TO CONTAGIOUS
   for( int a=0; a<inLL_numAgents; a++ ){
               ID_a = inLL[a];
                               // NOT_CONTAGIOUS_YET -> CONTAGIOUS
                               if(( inDayKtr                >  Factory.F_AgentFirstDayContagious [ID_a]) &&
                                 ( Factory.F_AgentSOH[ID_a] == Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET))      // BLUE
                               {
                                         Factory.F_AgentSOH[ID_a]=Factory.F_AGENT_INFECTED_CONTAGIOUS;    // RED
                               }

                               // CONTAGIOUS -> IMMUNE
                               if(( inDayKtr                >  Factory.F_AgentLastDayContagious [ID_a]) &&
                            ( Factory.F_AgentSOH[ID_a] == Factory.F_AGENT_INFECTED_CONTAGIOUS))               // RED
                               {
                                         Factory.F_AgentSOH[ID_a]=Factory.F_AGENT_IMMUNE;                              // BLACK
                               }

  }
```

```

// RUN ONE ITERATION NOW FOR AGENTS IN THIS STAGE *************************
try{
    int SickContagiousKtr = 0;
  for( int a=0; a<inLL_numAgents; a++ ){
        ID_a = inLL[a];
        if (Factory.F_AgentSOH[ID_a] == Factory.F_AGENT_INFECTED_CONTAGIOUS){ // RED
            SickContagiousKtr++;
        }
    }
    //if (SickContagiousKtr == 0) return 1;

    //------------------------------------------------------------------------
    numWell = inLL_numAgents - SickContagiousKtr;
    for( int a=0; a<inLL_numAgents; a++ ){
        ID_a = inLL[a];



        // NOTE:  There is no list for well agents. ********
        // NOTE:  Agents at end of list will never get sick.
        //for(int i=0; i<numWell;i++)
        //{
            if (Factory.F_AgentSOH[ID_a] == Factory.F_AGENT_SUSCEPTIBLE){

                //-------------------------------
                // HYPERCUBE SAMPLING SIMULATION
                // "Just an idea"
                //-------------------------------
                //inStageAdjust = inStageAdjust * (double )Factory.F_SimulationRun;

                a1  = Math.sqrt((double)SickContagiousKtr);
                a2  = a1 * inStageDensity;
                a3  = a2 * inStageAdjust;
                pct = a3/(float)99.0;

                //-------------------------------
                // HYPERCUBE SAMPLING SIMULATION
                //-------------------------------
                //Range = 900 * Factory.F_SimulationRun;
                Range = 900;

                double Percent = (double)(VA_rndGenSOH.nextInt(Range));
                try{Thread.sleep(1);}catch(InterruptedException e2){}

                if (Percent<= pct){
                    Factory.F_AgentSOH[ID_a]=Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET;
                Factory.F_AgentFirstDayContagious[ID_a] =  inDayKtr + 3;
                Factory.F_AgentLastDayContagious [ID_a] =  inDayKtr + 9;
                }
            }// susceptible
        //}//well agents
    }// all agents
}catch(Exception err){
        System.out.println(thisClassName + " Exception:  "+err);
    }
  //}// VA_run
   return 1;
   }

}//EOF CLASS
//----------------------------------------------------------------------------------------------------------
// H1N1_RNG.java
//----------------------------------------------------------------------------------------------------------
```

disprop.java

```
//----------------------------------------------------------------------------------------------------
// Albuquerque Academy
// Comparison of Disease Propagation Algorithms that Predict the Speed of an Outbreak
// 2013-2014
// Supercomputing Challenge Team 5
//----------------------------------------------------------------------------------------------------
// disprop.java
//----------------------------------------------------------------------------------------------------
//
// Main
// disprop.java
// To auto start...
//  java disprop H1N1_RBA
//  java disprop H1N1_MBA
//  java disprop H1N1_RNG
//
//
//  java disprop H1N1_RBA 1 2 3 (the three arguments for Latin Hypercube sampling of a Latin Square)
//  java disprop H1N1_RBA 3 1 2 (the three arguments for Latin Hypercube sampling of a Latin Square)
//  java disprop H1N1_RBA 2 3 1 (the three arguments for Latin Hypercube sampling of a Latin Square)
//
//  java disprop H1N1_MBA 1 2 3 (the three arguments for Latin Hypercube sampling of a Latin Square)
//  java disprop H1N1_MBA 3 1 2 (the three arguments for Latin Hypercube sampling of a Latin Square)
//  java disprop H1N1_MBA 2 3 1 (the three arguments for Latin Hypercube sampling of a Latin Square)
//
//  java disprop H1N1_RNG 1 2 3 (the three arguments for Latin Hypercube sampling of a Latin Square)
//  java disprop H1N1_RNG 3 1 2 (the three arguments for Latin Hypercube sampling of a Latin Square)
//  java disprop H1N1_RNG 2 3 1 (the three arguments for Latin Hypercube sampling of a Latin Square)
//----------------------------------------------------------------------------------------------------
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

import javax.swing.border.Border;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.DefaultListModel;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.ImageIcon;
import javax.swing.ListSelectionModel;

import javax.imageio.ImageIO;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.geom.RoundRectangle2D;
import java.awt.geom.Rectangle2D;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.Shape;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

import java.lang.Object;
import java.lang.Runtime;
import java.util.Random;
import java.util.Vector;
import java.util.StringTokenizer;

// -------------------------------------------------------------------------
// The class "disprop" is a subclass of it's parent class "JFrame".
// -------------------------------------------------------------------------
public class disprop extends JFrame implements ActionListener {

  static  int           DEBUG            = 0; // DEBUG SWITCH
  private String     IN_ALGO_FILENAME     = "IN_Algorithm.txt";
  private String     IN_AUTO_SIM_FILENAME = "IN_autoSimulation.txt";
  static  String     myVirusAlgorithm     = "";
  static  int           mySimulationRun      =  1; // Default use 1
  static  int                   mySimulationRunB             = -1; // 2nd Dimension Latin Square
  static  int                   mySimulationRunC             = -1; // 3rd Dimension Latin Square
  static  int                   mySimulationRunD             = -1; // 4th Dimension Latin
  static  int                   mySimulationRunE             = -1;
  static  int                   mySimulationRunF             = -1;
  static  int        mySimulationRun_LatinDimension = 1;
  static  boolean    autoStart           = false;

  private int        MAXDISEASENAMES      = 100;
  private String[]   myDiseaseNames       = new String[MAXDISEASENAMES];
  private String[]   myDiseaseNamesUpdate = new String[MAXDISEASENAMES];
  private String[]   myMenuList           = new String[50];
  private JTextField  myStatus            = new JTextField(100);
  private String     Rtn                  = null;
  private JList                   myA_List;
  private DefaultListModel        myA_Model;
  JButton                         myJB2;
  //JLabel                background;
  private Vector    myAgentVector;
  private Vector    myStageVector;
  private int    [] StageNum_Immune;                  // SOH = 75.0
  private int    [] StageNum_NotInfected;    // SOH = 100.0
  private int    [] StageNum_Infected_NotContagious;// SOH = 50.0
  private int    [] StageNum_Infected_Contagious;   // SOH = 25.0
  private int       tmpI;
  // STAGE DATA FOR BAR CHARTS
  private double [] D4_StageDensity;
  private double [] D4_StageAdjust;
  private String [] D4_StageName;
  private double [] D4_StageLat ;
  private double [] D4_StageLon ;
  private int    [] D4_StageNum;
  private int    [] D1_Role_numSick;
  private int    [] D1_Role_numImmune;
  private int    [] D1_Role_numSusceptible;
  private int    [] D1_Role_numSickNotContagious;
  private String TrendFileName;
  static String TrendFileName_LatinDimension;
  static  double    aStageLat;
  static  double    aStageLon;
  // FOR DRAWING MAP ON SCREEN
  static Image                    myImage;
  static ImageIcon   myImageIcon;
  static int                      myImageWidth= 0;  // set during runtime
  static int                      myImageHeight= 0; // set during runtime
  static int             mapOriginX= 200;
  static int             mapOriginY= 100;
  // TIME DATA
  private int                     T_maxSamples = 8760; // NUM HOURS IN YEAR // TODO:  keep at one year

  // Time Data
  private long   T_WallclockStart;      //*
  private int    T_MSinOneHr;           //*// VALUE FROM IN_TimeIC.txt
  private int    T_MSinOneDay;          //*
  private long   T_SimNow;              //*
  private long   T_SimDayStart;         //*
  private long   T_SimDayEnd;           //*
  private long   T_SimHrStart;          //*
  private long   T_SimHrEnd;            //*
  private int    hrKtr  = 0;            //*
  private int    dayKtr = 0;            //*

  // PLOT ARRAYS
  static int    [] plotRun_Immune;
  static int    [] plotRun_NotInfected;
  static int    [] plotRun_Infected_NotContagious;
  static int    [] plotRun_Infected_Contagious;
  static int    [] plotRun_Infected_Total;
  static int       plotKtr = 0;

```

```
  private int xTmp = 0;
  private int yTmp = 0;

  static Factory myFactory;
 // --------------------------------------------------------------
 // main
 // --------------------------------------------------------------
 public static void main(String[] args) {
        try{
                  myVirusAlgorithm = args[0];
                  System.out.println("main() myAlgorithm="+myVirusAlgorithm);
                  autoStart = true;
                }catch(Exception e0){} // keep going.

        try{
                                  mySimulationRun  = Integer.valueOf(args[1]);
                                  mySimulationRunB = Integer.valueOf(args[2]);
                                  mySimulationRunC = Integer.valueOf(args[3]);
                                  mySimulationRunD = Integer.valueOf(args[4]);
                                  mySimulationRunE = Integer.valueOf(args[5]);
                                  mySimulationRunF = Integer.valueOf(args[6]);
                }catch(Exception e00){} // keep going.

                  if( mySimulationRunB >= 0 ) mySimulationRun_LatinDimension++;
                  if( mySimulationRunC >= 0 ) mySimulationRun_LatinDimension++;
                  if( mySimulationRunD >= 0 ) mySimulationRun_LatinDimension++;
                  if( mySimulationRunE >= 0 ) mySimulationRun_LatinDimension++;
                  if( mySimulationRunF >= 0 ) mySimulationRun_LatinDimension++;

                  System.out.println("main() mySimulationRun="+mySimulationRun);
                  System.out.println("main() mySimulationRunB="+mySimulationRunB);
                  System.out.println("main() mySimulationRunC="+mySimulationRunC);
                  System.out.println("main() mySimulationRunD="+mySimulationRunD);
                  System.out.println("main() mySimulationRunE="+mySimulationRunE);
                  System.out.println("main() mySimulationRunF="+mySimulationRunF);
                  System.out.println("main() mySimulationRun_LatinDimension ="+mySimulationRun_LatinDimension);


      switch (mySimulationRun_LatinDimension){
      case 1:
                   TrendFileName_LatinDimension = "+"+ Integer.toString(mySimulationRun)+"+";
                                   break;
      case 2:
                  TrendFileName_LatinDimension = "+"+ Integer.toString(mySimulationRun) + "," +
Integer.toString(mySimulationRunB) +"+";
                                          break;
      case 3:
                  TrendFileName_LatinDimension = "+"+ Integer.toString(mySimulationRun) + "," +
Integer.toString(mySimulationRunB) + ","+
                                           Integer.toString(mySimulationRunC) +"+";
                                          break;
      case 4:
                  TrendFileName_LatinDimension = "+"+ Integer.toString(mySimulationRun) + "," +
Integer.toString(mySimulationRunB) + ","+
                                           Integer.toString(mySimulationRunC) + "," + Integer.toString(mySimulationRunD)+"+";
                                          break;
```

```
    case 5:
              TrendFileName_LatinDimension = "+"+ Integer.toString(mySimulationRun) + "," +
Integer.toString(mySimulationRunB) + ","+
                                  Integer.toString(mySimulationRunC) + "," + Integer.toString(mySimulationRunD) + "," +
                                  Integer.toString(mySimulationRunE) +"+";
                                 break;
    case 6:
              TrendFileName_LatinDimension = "+"+ Integer.toString(mySimulationRun) + "," +
Integer.toString(mySimulationRunB) + ","+
                                  Integer.toString(mySimulationRunC) + "," + Integer.toString(mySimulationRunD) + "," +
                                  Integer.toString(mySimulationRunE) + "," + Integer.toString(mySimulationRunF) +"+";
                                 break;
    default:
                                 System.out.println(" ERROR2-  mySimulationRun_LatinDimension = "
                                                                 +mySimulationRun_LatinDimension)     ;
                                 break;
    }



      disprop myDisprop = new disprop();  // make an object and run constructor
      myDisprop.setSize(1000,700);                              // set GUI size
      myDisprop.setJMenuBar(myDisprop.createMenuBar());         // create menu bar
      myDisprop.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // close normally
      myDisprop.setVisible(true);                               // display it
 }
  // ---------------------------------------------------------------
  // CONSTRUCTOR
  // ---------------------------------------------------------------
  public disprop() {

      setTitle("Comparison of Disease Propagation Algorithms");
      myA_Model = new DefaultListModel();
      myA_Model.ensureCapacity(MAXDISEASENAMES);
      // ---------------------------------------
      // (1) ALGORITHM LIST - open and read file
      //----------------------------------------
      Rtn  = read_IN_Algorithm(IN_ALGO_FILENAME);
      System.out.println(Rtn);
      myStatus.setText  (Rtn);

      // -----------------------------------------------------
      // (2) DISEASE LIST -
      // Create the myA_List and put it in a Jlist scroll pane.
      // -----------------------------------------------------
      myA_List = new JList(myA_Model);
      myA_List.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
      myA_List.setSelectedIndex(0);
      myA_List.setVisibleRowCount(30);
      JScrollPane A_listScrollPane = new JScrollPane(myA_List);
      A_listScrollPane.setBorder(BorderFactory.createTitledBorder("Disease List"));
      A_listScrollPane.setBackground(Color.cyan);

      // ------------------------------
      // (3) DISEASE LIST - Add Listener
      // ------------------------------
 myA_List.setSelectedIndex(0);
 try{
   myA_List.addListSelectionListener (new ListSelectionListener() {
          public void valueChanged   (     ListSelectionEvent e) {
        myVirusAlgorithm =
            myA_Model.getElementAt(myA_List.getSelectedIndex()).toString();
        myJB2.setText (myVirusAlgorithm);
     }
   });
      }catch (Exception abc){
                System.out.println("disprop() Exception=ListSelectionListener"+abc);
      }
 // -------------------------------------------------------
 // (4) EXECUTE DISEASE ANALYSIS BUTTON
 // SET INITIAL NAME, WHICH CHANGES AS LIST IS SELECTED
 // -------------------------------------------------------
 myJB2 = new JButton    (myDiseaseNames[0]);
 myJB2.setBorder(BorderFactory.createTitledBorder("Selected Disease"));
 myJB2.setBackground    (Color.blue);
 myJB2.addActionListener(this);
      System.out.println("1  myVirusAlgorithm="+myVirusAlgorithm);
      if( myVirusAlgorithm == "" )  myVirusAlgorithm = myA_Model.getElementAt(0).toString();

      System.out.println("2  myVirusAlgorithm="+myVirusAlgorithm);
 myJB2.setText(myVirusAlgorithm);

 //background=new JLabel(new ImageIcon("IN_map.jpg"));
 // ------------------------------------------------------
 // (5) CREATE JAVA CONTAINER AND ADD COMPONENTS
 // ------------------------------------------------------
 Container cp = getContentPane();
 cp.add(        myStatus, BorderLayout.NORTH );
 cp.add(          myJB2, BorderLayout.SOUTH );
 cp.add(A_listScrollPane, BorderLayout.WEST  );
 //cp.add(background, BorderLayout.EAST,  background);

 // ----------------------------------------------
      // (6) PREPARATION FOR MAP TO BE DRAWED ON DISPLAY
 // ----------------------------------------------
      myImage       = Toolkit.getDefaultToolkit().getImage("IN_map.jpg");
      myImageIcon   = new ImageIcon(myImage);
      myImageWidth  = myImageIcon.getIconWidth();  // X
      myImageHeight = myImageIcon.getIconHeight(); // Y
      System.out.println("disprop() IN_map.jpg  W,H = "+myImageWidth+","+myImageHeight
                                      +"  origin X,Y="+mapOriginX+","+mapOriginY);
// ---------------------------------------
// (7) CREATE AGENTS THEN STAGES IN FACTORY
// ---------------------------------------
      myFactory                 = new Factory(); // Do Nothing Constructor
      myFactory.F_numStages      = D4_readFile("D4_StageLatLon.txt");
      myFactory.F_VirusAlgorithm = myVirusAlgorithm;
      myFactory.F_SimulationRun  = mySimulationRun;  // 2014-03-10 // For Latin Hyper Cube
      myFactory.F_SimulationRunB = mySimulationRunB;  // 2014-03-10 // For Latin Hyper Cube
      myFactory.F_SimulationRunC = mySimulationRunC;  // 2014-03-10 // For Latin Hyper Cube
      myFactory.F_SimulationRunD = mySimulationRunD;  // 2014-03-10 // For Latin Hyper Cube
      myFactory.F_SimulationRunE = mySimulationRunE;  // 2014-03-10 // For Latin Hyper Cube
```

```
        myFactory.F_SimulationRunF = mySimulationRunF;  // 2014-03-10 // For Latin Hyper Cube
        myFactory.F_SimulationRun_LatinDimension = mySimulationRun_LatinDimension;  // 2014-03-10 // For Latin Hyper Cube

        myFactory.initWallclockStartTime();
        T_MSinOneHr = myFactory.aReadTimeIC ("IN_TimeIC.txt"); //
        System.out.println("disprop() call Factory.createAgents...................");
        myAgentVector = myFactory.createAgents(T_MSinOneHr,DEBUG);
    // createStages -- old location
        System.out.println("disprop() T_MSinOneHr="+T_MSinOneHr);


        D1_Role_numImmune           = new int [Factory.F_numAgentRoles];
        D1_Role_numSusceptible       = new int [Factory.F_numAgentRoles];
        D1_Role_numSickNotContagious = new int [Factory.F_numAgentRoles];
        D1_Role_numSick             = new int [Factory.F_numAgentRoles];
        //--------------------------
        // (8) INITIALIZE PLOT ARRAYS
        //--------------------------
        plotRun_Immune               = new int [T_maxSamples];
        plotRun_NotInfected           = new int [T_maxSamples];
        plotRun_Infected_NotContagious = new int [T_maxSamples];
        plotRun_Infected_Contagious    = new int [T_maxSamples];
        plotRun_Infected_Total         = new int [T_maxSamples];

        for(int i=0; i<T_maxSamples; i++ ){
                plotRun_Immune              [i]=0;
                plotRun_NotInfected          [i]=0;
                plotRun_Infected_NotContagious[i]=0;
                plotRun_Infected_Contagious   [i]=0;
                plotRun_Infected_Total        [i]=0;
        }
        plotKtr = 0;

    System.out.println("disprop() Constructor Complete.===================================");
    myStatus.setText  ("disprop Constructor Complete.");
    myJB2.setText (myVirusAlgorithm);

    if( autoStart )  {
                System.out.println("constructor done, calling startYearSim() myAlgorithm="+myVirusAlgorithm);
                startYearSim();
        }
} // END Constructor

//-----------------------------------------------------------------
public String read_IN_Algorithm( String inFileName )
{
    String a = "disprop() read_IN_Algorithm() Attempt File Read >"+inFileName+"<";
    System.out.println(a);
    try{
          FileInputStream fstream = new FileInputStream(inFileName);
          DataInputStream in      = new DataInputStream(fstream);
          BufferedReader br       = new BufferedReader(new InputStreamReader(in));
          String strLine;
      int i=0;
          while ((strLine = br.readLine()) != null)   {
                  System.out.println  ("disprop() read_IN_Algorithm(): "+ strLine);
         myA_Model.addElement(strLine);
           }
           in.close();
           fstream = null;
           in      = null;
      br      = null;
      strLine = null;
    }catch (Exception e){
      System.out.println("disprop() read_IN_Algorithm() Exception: File read>"+inFileName+"<");
    }
    return a;
}

//--------------------------------------------------------------------
// LISTEN FOR USER ACTION PERFORMED
//--------------------------------------------------------------------
public void actionPerformed(ActionEvent e)
{
      String myActionButton = e.getActionCommand();
      System.out.println("disprop() Analyzing Disease: "+myActionButton);
      int userClickMenuItemFlag = 0;
      // -----------------------------------
      // CHECK IF MENU ACTION SELECTED
      // -----------------------------------
      for( int k=0; k<myMenuList.length; k++ ){

        if( myActionButton == myMenuList[k] ){
                    // MENU BUTTON CLICKED --------------------------
          if( myActionButton == "Edit Disease List" ){
                        myFileEditor myFE = new myFileEditor();
              myFE.startMe();
          }

                    // MENU BUTTON CLICKED --------------------------
          if( myActionButton == "TRUTH DATA - fludb Download Create Histogram" ){
                            fludb_Download_Create_Histogram();
                    }

                    // MENU BUTTON CLICKED --------------------------
          if( myActionButton == "TRUTH DATA - fludb Plot Histogram"){
                            fludb_Plot_Histogram();
                    }

                    // MENU BUTTON CLICKED --------------------------
          if( myActionButton == "SIM DATA - Save and Plot Data" ){
                    saveAndPlotTrend();
          }

                    // MENU BUTTON CLICKED --------------------------
          if( myActionButton == "SIM and TRUTH Data Compare - Save and Plot" ){
                    saveAndPlotTrend_fludbCompare();
                    }

      // MENU BUTTON CLICKED --------------------------
          if( myActionButton == "SIM and TRUTH Data Difference - Save and Plot" ){
                    saveAndPlotTrend_fludbDiff();
                    }


                    // MENU BUTTON CLICKED --------------------------
          if( myActionButton == "Refresh Disease List" ){
                            try{
                            int listSize = myA_Model.getSize ();
                    myA_List.setSelectedIndex(listSize-1);
                            for( int i=0; i<listSize-1; i++ ) {
                                    myA_Model.removeElementAt(0);
                            }
                                    // RE-POPULATE LIST
                            Rtn = read_IN_Algorithm( IN_ALGO_FILENAME );
                            myStatus.setText  (Rtn);
                    myA_List.setSelectedIndex(0);
                            myA_Model.removeElementAt(0);

                            }catch(Exception aaaaa){
                                    System.out.println("disprop() read_IN_Algorithm() Exception: Updating "+IN_ALGO_FILENAME);
                            }
              myA_List.setSelectedIndex(0);
              myVirusAlgorithm = myA_Model.getElementAt(0).toString();
                    myFactory.F_VirusAlgorithm = myVirusAlgorithm;
              myJB2.setText (myVirusAlgorithm);
          }
                    // MENU BUTTON CLICKED --------------------------
          if( myActionButton == "Exit" ){
              System.out.println("disprop() Program Exiting Normally.");
              System.exit(0);
          }
          userClickMenuItemFlag = 1;
          break;
        }
      }
      //--------------------------------------
          // IF MENU ITEM NOT CLICKED, THEN
          // BY DEFAULT, "SOUTH BUTTON CLICKED"
          // >>>>>>>> RUN YEAR SIMULATION <<<<<<<<
      //--------------------------------------
      if ( userClickMenuItemFlag == 0 ){
                startYearSim();

      } // SOUTH BUTTON CLICKED
return;
} // END actionPerformed()


//---------------------------------------------------------------------------------------
public void startYearSim(){

        // createStages -- moved here
        System.out.println("disprop() Call Factory.createStages()");
        myFactory.F_VirusAlgorithm = myVirusAlgorithm;
```

```
    myStageVector = myFactory.createStages(T_MSinOneHr, D4_StageNum, D4_StageName, D4_StageDensity, D4_StageAdjust, DEBUG);

    //----------------------------------------------------
    // CREATE NEW THREAD HERE, RUN ONE YEAR, T_maxSampels
    // Does not use wall clock, not necessary.
    //----------------------------------------------------
final Thread myThread = new Thread( new Runnable() {
public void run() {

    plotKtr  = 0;
    int numDays = 360; //
    int hrNow   =   0;
T_WallclockStart = Factory.F_WallclockStart;
T_SimNow         = System.currentTimeMillis() - T_WallclockStart;
T_MSinOneDay     = T_MSinOneHr * 24;

while( dayKtr < numDays ){
    //--------------------------------------
    // TIME-B: Get Time Now, DayStart, DayEnd
    //     ... Wait until current day starts
    //--------------------------------------
    T_SimNow      = System.currentTimeMillis() - T_WallclockStart;
    T_SimDayStart = T_MSinOneDay *  dayKtr   ;
    T_SimDayEnd   = T_MSinOneDay * (dayKtr+1);

    while( T_SimNow < T_SimDayStart ){
        try{Thread.sleep(T_MSinOneHr);}catch(InterruptedException e2){}
        T_SimNow = System.currentTimeMillis() - T_WallclockStart;
    }
    //-------------------------
    // TIME-C: Hour Loop, 1->24
    //-------------------------
    hrKtr = 0;
    while( hrKtr < 24 ){
              hrNow++;
      //--------------------------------------
      // TIME-D: Get Time Now, HrStart, HrEnd
      //     ... Wait until current hour starts
      //--------------------------------------
      T_SimNow     = System.currentTimeMillis() - T_WallclockStart;
      T_SimHrStart = (T_MSinOneHr *  hrKtr   )  + T_SimDayStart;
      T_SimHrEnd   = (T_MSinOneHr * (hrKtr+1))  + T_SimDayStart;

      while( T_SimNow < T_SimHrStart ){
        try{Thread.sleep(T_MSinOneHr);}catch(InterruptedException e2){}
        T_SimNow = System.currentTimeMillis() - T_WallclockStart;
      }

      switch (mySimulationRun_LatinDimension){
      case 1:
              myStatus.setText("Simulation Run: +"+
                              mySimulationRun+
                              "+ Time:  Day="+dayKtr+" Hour="+hrKtr+"     Year Hours Total ="+hrNow);
                      break;
      case 2:
```

```
                    myStatus.setText("Simulation Run: +"+
                              mySimulationRun+","+ mySimulationRunB +
                              "+ Time:  Day="+dayKtr+" Hour="+hrKtr+"     Year Hours Total ="+hrNow);
                             break;
               case 3:
                    myStatus.setText("Simulation Run: +"+
                              mySimulationRun+","+ mySimulationRunB + "," + mySimulationRunC +
                              "+ Time:  Day="+dayKtr+" Hour="+hrKtr+"     Year Hours Total ="+hrNow);
                             break;
               case 4:
                    myStatus.setText("Simulation Run: +"+
                              mySimulationRun+","+ mySimulationRunB + "," + mySimulationRunC + "," +mySimulationRunD +
                              "+ Time:  Day="+dayKtr+" Hour="+hrKtr+"     Year Hours Total ="+hrNow);
                             break;
               case 5:
                    myStatus.setText("Simulation Run: +"+
                              mySimulationRun+","+ mySimulationRunB + "," + mySimulationRunC + "," +mySimulationRunD +
                              "," + mySimulationRunE +
                              "+ Time:  Day="+dayKtr+" Hour="+hrKtr+"     Year Hours Total ="+hrNow);
                             break;
               case 6:
                    myStatus.setText("Simulation Run: +"+
                              mySimulationRun+","+ mySimulationRunB + "," + mySimulationRunC + "," +mySimulationRunD +
                              "," + mySimulationRunE + "," + mySimulationRunF +
                              "+ Time:  Day="+dayKtr+" Hour="+hrKtr+"     Year Hours Total ="+hrNow);
                             break;
               default:
                             System.out.println(" ERROR-  mySimulationRun_LatinDimension = "
                                            +mySimulationRun_LatinDimension)     ;
                             break;
               }


                        //------------------------------------
                        // (1) CLEAR PLOT DATA
                        // (2) PAINT, THEN SLEEP 1Hr STOPWATCH
                        //------------------------------------
                        for(int i=0; i<myFactory.F_numStages; i++ ){
                             StageNum_Immune               [i] = 0;
                             StageNum_NotInfected          [i] = 0;
                             StageNum_Infected_NotContagious[i] = 0;
                             StageNum_Infected_Contagious   [i] = 0;
                        }
                        repaint();

               hrKtr++;
              } // hrKtr
              dayKtr++;
        } // dayKtr
             saveAndPlotTrend(); // For Sensitivity Study

             System.out.println("----------One Year Simulation Thread Completed Successfully----------");
             System.out.println("----------One Year Simulation Thread Completed Successfully----------");
```

```
                System.out.println("----------One Year Simulation Thread Completed Successfully----------");
        System.out.println("disprop() Program Exiting Normally.");
        System.exit(0);

                 } // run()
        }  // runnable()
        );  // Thread
        myThread.start(); // goes back and starts myThread

                System.out.println("-------------------One Year Simulation Thread Started----------------------- ");
                System.out.println("-------------------One Year Simulation Thread Started----------------------- ");
                System.out.println("-------------------One Year Simulation Thread Started----------------------- ");
  } // startYearSim()


//------------------------------------------------------------------------------------------
  public void paint(Graphics g) {
    //--------------------------------------------
    // PERFORM SUPER to REPAINT ALL GUI COMPONENTS
    //--------------------------------------------
    super.paint(g);
    Graphics2D g2 = (Graphics2D)g;

    //--------------------
    // DRAW BACKGROUND MAP
    //--------------------
    g2.drawImage(myImage, (int )mapOriginX, (int )mapOriginY, null);

    int aSOH     = 0;
    int aStage   = 0;
    int XX, YY;

    for( int i=0; i<Factory.F_numAgents; i++ ){
     try{
                aSOH   = (int )Factory.F_AgentSOH[i];
                aStage = Factory.F_AgentStageNum [i];
                if( aStage >= Factory.F_numStages )
                                System.out.println("disprop() WARNING: Stage Number "+aStage);

                switch(aSOH){
                 case 100:
                                StageNum_Immune                [aStage]++;//SOH=100.0
                                plotRun_Immune                 [plotKtr]++;
                                break;
                 case 75:
                                StageNum_NotInfected           [aStage]++;//SOH= 75.0
                                plotRun_NotInfected            [plotKtr]++;
                                break;
                 case 50:
                                StageNum_Infected_NotContagious[aStage]++;//SOH= 50.0
                                plotRun_Infected_NotContagious [plotKtr]++ ;
                        plotRun_Infected_Total         [plotKtr]++;
                                break;
                 case 25:
                                StageNum_Infected_Contagious   [aStage]++;//SOH= 25.0
                                plotRun_Infected_Contagious    [plotKtr]++;
                break;
         default:
                System.out.println("disprop paint() UNKNOWN AgentSOH=>"+aSOH+"<");
                break;
        }
 }catch(Exception e7){
        System.out.println("disprop() Exception: retrieving AgentSOH i="+i+"  aStage="+aStage+"  aSOH="+aSOH );
        System.out.println("disprop() ... check D3*.txt file, ensure stage numbers appropriate");
 }
}

plotKtr++;
// --------------------
// BAR CHART EACH STAGE
// --------------------
//g2.setStroke(new BasicStroke(3)); // to draw text
/* ORIGINAL VERTICAL BAR CHART
for( int stg=0; stg<Factory.F_numStages; stg++ ){
        XX  = mapOriginX + (int )D4_StageLat[stg];
        YY  = mapOriginY + (int )D4_StageLon[stg];
        // DRAW STRING
        g2.setPaint(Color.black);
        g2.drawString(Integer.toString(stg+1),XX-10,YY+15);
   tmpI = StageNum_NotInfected[stg];
        g2.setPaint(Color.green);
   g2.fill(new Rectangle2D.Double( XX    ,YY-tmpI,10, tmpI));
   tmpI = StageNum_Infected_NotContagious[stg];
        g2.setPaint(Color.blue);
   g2.fill(new Rectangle2D.Double( XX+10,YY-tmpI,10, tmpI));
   tmpI = StageNum_Infected_Contagious[stg];
        g2.setPaint(Color.red);
   g2.fill(new Rectangle2D.Double( XX+20,YY-tmpI,10, tmpI));
   tmpI = StageNum_Immune[stg];
        g2.setPaint(Color.black);
   g2.fill(new Rectangle2D.Double( XX+30,YY-tmpI,10, tmpI));
} // Stage
*/

// HORIZONTAL BARS
for( int stg=0; stg<Factory.F_numStages; stg++ ){

        XX = mapOriginX - 120;
        YY = mapOriginY + (stg*25);

        // DRAW STRING
        g2.setPaint(Color.black);
        g2.drawString(D4_StageName[stg],XX,YY+50);

        XX+=120;

   tmpI = StageNum_NotInfected[stg];
        g2.setPaint(Color.green);
   g2.fill(new Rectangle2D.Double( XX,YY+10, tmpI, 5));

   tmpI = StageNum_Infected_NotContagious[stg];
        g2.setPaint(Color.blue);
        g2.fill(new Rectangle2D.Double( XX,YY+15, tmpI, 5));

        tmpI = StageNum_Infected_Contagious[stg];
              g2.setPaint(Color.red);
        g2.fill(new Rectangle2D.Double( XX,YY+20, tmpI, 5));

        tmpI = StageNum_Immune[stg];
              g2.setPaint(Color.black);
        g2.fill(new Rectangle2D.Double( XX,YY+25, tmpI, 5));
    } // Stage

    // VERTICAL BARS - PLOT EACH ROLE SOH
    int q2=0;
    for( int q1=0; q1<Factory.F_numAgents; q1++){
              if( Factory.F_Agent_D1_Roles[q2] != Factory.F_Agent_D2_yrSchd_Role[q1] ){
                          q2++;
              }
              if( Factory.F_AgentSOH[q1] == Factory.F_AGENT_SUSCEPTIBLE)
                          D1_Role_numSusceptible[q2]++;
              if( Factory.F_AgentSOH[q1] == Factory.F_AGENT_INFECTED_NOT_CONTAGIOUS_YET )
                          D1_Role_numSickNotContagious   [q2]++;
              if( Factory.F_AgentSOH[q1] == Factory.F_AGENT_INFECTED_CONTAGIOUS )
                          D1_Role_numSick   [q2]++;
              if( Factory.F_AgentSOH[q1] == Factory.F_AGENT_IMMUNE)
                          D1_Role_numImmune [q2]++;
   } // GET HISTOGRAM

   XX  = mapOriginX + 100;
   YY  = mapOriginY + 500;

  // g2.setPaint(Color.black);
  // g2.drawString("Actor Roles",XX, YY+40);

xTmp = 0;
yTmp = 0;

    // VERTICAL BARS - PLOT EACH ROLE SOH
   for( q2=0; q2<Factory.F_numAgentRoles; q2++ ){

        tmpI = D1_Role_numSusceptible[q2] + 1;
        tmpI *=3;
              g2.setPaint(Color.green);
        g2.fill(new Rectangle2D.Double( XX+(q2*40)    ,YY-tmpI,5, tmpI));

              g2.setPaint(Color.black);
              g2.drawString(Integer.toString(q2),XX+(q2*40), YY+12);

              g2.setPaint(Color.black);
              g2.drawString(Integer.toString(q2),XX+(q2*40), YY+12);

// DRAW STRING
xTmp = mapOriginX + 500;
yTmp = mapOriginY + (q2*20) - 50;
g2.drawString("Agent Role "+Integer.toString(q2)+"="+Factory.F_Agent_D1_Roles[q2],xTmp,yTmp+50);
xTmp+=120;

```

```
        tmpI = D1_Role_numSickNotContagious[q2] + 1;
        tmpI *=3;
                g2.setPaint(Color.blue);
        g2.fill(new Rectangle2D.Double( XX+(q2*40)+5 ,YY-tmpI,5, tmpI));

        tmpI = D1_Role_numSick[q2] + 1;
        tmpI *=3;
                g2.setPaint(Color.red);
        g2.fill(new Rectangle2D.Double( XX+(q2*40)+10 ,YY-tmpI,5, tmpI));

        tmpI = D1_Role_numImmune[q2] + 1;
        tmpI *=3;
                g2.setPaint(Color.black);
        g2.fill(new Rectangle2D.Double( XX+(q2*40)+15 ,YY-tmpI,5, tmpI));

        g2.setPaint(Color.black);
        g2.drawString("Agent Roles" ,XX, YY+35);

        D1_Role_numImmune           [q2] = 0;
                D1_Role_numSusceptible      [q2] = 0;
                D1_Role_numSickNotContagious[q2] = 0;
        D1_Role_numSick             [q2] = 0;
   }// VERTICAL BARS EACH ROLE

xTmp = mapOriginX + 100;
yTmp = mapOriginY  ;

   g2.setPaint(Color.black);
   g2.drawString("BAR PLOTS - AGENT STATE OF HEALTH COLORS:",xTmp-20, yTmp); yTmp+=15;
   g2.setPaint(Color.green);
   g2.drawString("Green = Susceptible"             ,xTmp,    yTmp); yTmp+=15;
   g2.setPaint(Color.blue);
   g2.drawString("Blue   = Infected Not Contagious" ,xTmp,    yTmp); yTmp+=15;
   g2.setPaint(Color.red);
   g2.drawString("Red    = Infected Contagious"     ,xTmp,    yTmp); yTmp+=15;
   g2.setPaint(Color.black);
   g2.drawString("Black  = Immune"                 ,xTmp,    yTmp);

   g2 = null;
  } // paint()

  //----------------------------------------------------------------------
  // COOKBOOK - CREATE MENU
  //----------------------------------------------------------------------
  public JMenuBar createMenuBar()
  {
         JMenuBar              menuBar;
         JMenu                 menu;
         JMenuItem             menuItem;
        // ----------------
        // CREATE MENU BAR
        // ----------------
        menuBar = new JMenuBar();
        int   ii=0;

        // FILE MENU
```

```
menu = new JMenu("File");
menu.setMnemonic(KeyEvent.VK_F);
menu.getAccessibleContext().setAccessibleDescription("my File");
menuBar.add(menu);

   // REFRESH ACTION FILE LIST ITEM
   myMenuList[ii] = "Refresh Disease List";
   menuItem = new JMenuItem(myMenuList[ii++],KeyEvent.VK_R);
   menuItem.getAccessibleContext().setAccessibleDescription("my Refresh Action List");
   menuItem.addActionListener(this);
   menuItem.setBackground(Color.cyan);
   menu.add(menuItem);
   menuBar.add(menu);

   // EXIT ITEM
   myMenuList[ii] = "Exit";
   menuItem = new JMenuItem(myMenuList[ii++],KeyEvent.VK_X);
   menuItem.getAccessibleContext().setAccessibleDescription("my Exit");
   menuItem.addActionListener(this);
   menu.add(menuItem);

// EDIT MENU
menu = new JMenu("Edit");
menu.setMnemonic(KeyEvent.VK_E);
menu.getAccessibleContext().setAccessibleDescription("my Edit");
menuBar.add(menu);

   // ACTION FILE LIST ITEM
   myMenuList[ii] = "Edit Disease List";
   menuItem = new JMenuItem(myMenuList[ii++],KeyEvent.VK_A);
   menuItem.getAccessibleContext().setAccessibleDescription("my Action List");
   menuItem.addActionListener(this);
   menuItem.setBackground(Color.cyan);
   menu.add(menuItem);
   menu.addSeparator();

// PLOT MENU -----------
menu = new JMenu("Plot");
menu.setMnemonic(KeyEvent.VK_E);
menu.getAccessibleContext().setAccessibleDescription("my Plot");
menuBar.add(menu);

   // ACTION FILE LIST ITEM
   myMenuList[ii] = "TRUTH DATA - fludb Download Create Histogram";
   menuItem = new JMenuItem(myMenuList[ii++],KeyEvent.VK_A);
   menuItem.getAccessibleContext().setAccessibleDescription("my Action List 1");
   menuItem.addActionListener(this);
   menu.add(menuItem);
   menu.addSeparator();

   // ACTION FILE LIST ITEM
   myMenuList[ii] = "TRUTH DATA - fludb Plot Histogram";
   menuItem = new JMenuItem(myMenuList[ii++],KeyEvent.VK_A);
   menuItem.getAccessibleContext().setAccessibleDescription("my Action List 2");
   menuItem.addActionListener(this);
   menu.add(menuItem);
```

```
        menu.addSeparator();

        // ACTION FILE LIST ITEM
        myMenuList[ii] = "SIM DATA - Save and Plot Data";
        menuItem = new JMenuItem(myMenuList[ii++],KeyEvent.VK_A);
        menuItem.getAccessibleContext().setAccessibleDescription("my Action List 3");
        menuItem.addActionListener(this);
        menu.add(menuItem);
        menu.addSeparator();

        // ACTION FILE LIST ITEM
        myMenuList[ii] = "SIM and TRUTH Data Compare - Save and Plot";
        menuItem = new JMenuItem(myMenuList[ii++],KeyEvent.VK_A);
        //menuItem.setBackground(Color.white);
        menuItem.getAccessibleContext().setAccessibleDescription("my Action List 4");
        menuItem.addActionListener(this);
        menu.add(menuItem);
        menu.addSeparator();


        // ACTION FILE LIST ITEM
        myMenuList[ii] = "SIM and TRUTH Data Difference - Save and Plot";
        menuItem = new JMenuItem(myMenuList[ii++],KeyEvent.VK_A);
        menuItem.getAccessibleContext().setAccessibleDescription("my Action List 4");
        menuItem.addActionListener(this);
        menu.add(menuItem);
        menu.addSeparator();

    return menuBar;
} // END JMenuBar()

//---------------------------------------------------------------
public int D4_readFile(String inFileName)
{
   System.out.println("disprop() D4_readFile() Attempt File Read >"+inFileName+"<");
   System.out.println("disprop() D4_readFile() 1/3");
   int    inStageKtr  = 0;
   try{
      FileInputStream fstream = new FileInputStream(inFileName);
      DataInputStream in      = new DataInputStream(fstream);
      BufferedReader br       = new BufferedReader(new InputStreamReader(in));
      String strLine;
      while ((strLine = br.readLine()) != null)   {
               StringTokenizer st  = new StringTokenizer(strLine,",");
                         inStageKtr++;
               st  = null;
      }
      in.close();
    }catch (Exception e){
      System.out.println("disprop() D4_readFile() Exception:  EOF_1 >"+inFileName+"<");
    }
    System.out.println("disprop() D4_readFile() 2/3");

    D4_StageName                   = new String[inStageKtr];
    D4_StageLat                    = new double[inStageKtr];
    D4_StageLon                    = new double[inStageKtr];
```

```
     D4_StageDensity              = new double[inStageKtr];
     D4_StageAdjust               = new double[inStageKtr];
     D4_StageNum                  = new int   [inStageKtr];
     // BAR CHART
     StageNum_Immune              = new int   [inStageKtr];
     StageNum_NotInfected         = new int   [inStageKtr];
     StageNum_Infected_NotContagious= new int   [inStageKtr];
     StageNum_Infected_Contagious   = new int   [inStageKtr];

    inStageKtr = 0;
    try{
       FileInputStream fstream = new FileInputStream(inFileName);
       DataInputStream in      = new DataInputStream(fstream);
       BufferedReader br       = new BufferedReader(new InputStreamReader(in));
       String strLine;
       while ((strLine = br.readLine()) != null)   {
                StringTokenizer st  = new StringTokenizer(strLine,",");
                D4_StageName   [inStageKtr] = st.nextToken().trim();
                D4_StageLat    [inStageKtr] = Double.parseDouble(st.nextToken().trim());
                D4_StageLon    [inStageKtr] = Double.parseDouble(st.nextToken().trim());
                D4_StageDensity[inStageKtr] = Double.parseDouble(st.nextToken().trim());
                D4_StageAdjust [inStageKtr] = Double.parseDouble(st.nextToken().trim());
                D4_StageNum    [inStageKtr] = Integer.parseInt  (st.nextToken().trim());

                            System.out.println("disprop()    Name,lat,lon="+D4_StageName[inStageKtr]+","
                              +D4_StageLat[inStageKtr]+","+D4_StageLon[inStageKtr]);
                            inStageKtr++;
                st  = null;
       }
       in.close();
       System.out.println("disprop() D4_readFile() 3/3");
     }catch (Exception e){
       System.out.println("disprop() D4_readFile() Exception: EOF_2 >"+inFileName+"<");
     }
     System.out.println("disprop() D4_readFile() Num Stages="+inStageKtr);
     return inStageKtr;
  } //


 public int fludb_Download_Create_Histogram(){
           try{
           String trendCommand = "java trend ./fludb_Download.txt 0";
                       System.out.println("Executing ******************************** "+trendCommand+"...wait few seconds to write to file");
                       Runtime.getRuntime().exec("java trend ./fludb_Download.txt 0");
           }catch( Exception ErunA ){ System.out.println("Exception: Executing fludb_Download_Create_Histogram()"+ErunA);}
           return 0;
 }

 public int fludb_Plot_Histogram(){
           try{
           String trendCommand = "java trend ./fludb_Histogram_ToCompare.txt 1";
                       System.out.println("Executing ******************************** "+trendCommand);
                       Runtime.getRuntime().exec("java trend ./fludb_Histogram_ToCompare.txt 1");
           }catch( Exception ErunB ){ System.out.println("Exception: Executing fludb_Plot_Histogram()"+ErunB);}
           return 0;
```

```
}

public int saveAndPlotTrend_fludbCompare(){
   try{
       saveTrend();
   String trendCommand = "java trend "+TrendFileName+" 2";
       System.out.println("Executing ******************************** "+trendCommand);
           Runtime.getRuntime().exec(trendCommand);
       }catch( Exception ErunC ){ System.out.println("Exception: Executing saveAndPlotTrend_fludbCompare "+TrendFileName + ErunC);}
       return 0;
    }

    public int saveAndPlotTrend_fludbDiff(){
           try{
               saveTrend();
           String trendCommand = "java trend "+TrendFileName+" 3";
               System.out.println("Executing ******************************** "+trendCommand);
               Runtime.getRuntime().exec(trendCommand);
           }catch( Exception ErunD ){ System.out.println("Exception: Executing saveAndPlotTrend_fludbDiff "+TrendFileName + ErunD);}
           return 0;
        }


    // PLOT CURRENT TREND
    public int saveAndPlotTrend() {
       try{
           saveTrend();
       String trendCommand = "java trend "+TrendFileName+" 4";
           System.out.println("Executing ******************************** "+trendCommand);
           Runtime.getRuntime().exec(trendCommand);
       }catch( Exception ErunE ){ System.out.println("Exception: Executing saveAdPlotTrend"+TrendFileName +ErunE);}
       return 0;
    }

    // SAVE CURRENT TREND
    public int saveTrend() {

       TrendFileName =
     "./OUT_"+Factory.F_VirusAlgorithm+"_TREND_" + TrendFileName_LatinDimension + "_" +
           (long )Factory.F_WallclockStart+".txt";
       try{
           System.out.println("disprop() Attempt FULL Create "+TrendFileName);

           File myOutFile = new File(TrendFileName);
       if( !myOutFile.exists()){myOutFile.createNewFile(); }

       FileWriter     myFW = new FileWriter(myOutFile.getAbsoluteFile());

       BufferedWriter myBW = new BufferedWriter(myFW);
       for( int i=0; i<T_maxSamples; i++ ){
               myBW.write(i+" "+
               plotRun_NotInfected          [i] +" "+
               plotRun_Infected_NotContagious[i] +" "+
                              plotRun_Infected_Contagious   [i] +" "+
                              plotRun_Immune                [i] +" "+
                              plotRun_Infected_Total        [i] +"\n");
                    }
                    myBW.close();
      }catch(Exception myWriteErr){
                    System.out.println("disprop() Exception: opening/writing file"+myWriteErr);
          }
          return 0;
  }
}
// EOF
//-------------------------------------------------------------------------------------------------------
// disprop.java
//-------------------------------------------------------------------------------------------------------
```