Model of Disease Propagation to Predict the Speed of an Outbreak

New Mexico

Supercomputing Challenge

Final Report

April 2013

Team 6

Marcus Dominguez-Kuhne

Albuquerque Academy

Team Member: Marcus Dominguez-Kuhne

Teacher: Mr. Jim Mims

Project Mentor: Ms. Marie Kuhne

Table of Contents

1 Executive Summary

This project is designed to model the speed of the spread of the Influenza Virus. If authorities can predict when and how many people will be infected in a given area then they can prevent the spread of the flu to the best of their abilities. The goal of the project is to simulate the propagation of the H1N1 influenza in 2009. In this project a C++ software application using OpenGL was programmed to model the propagation speed of a virus by creating a rule based algorithm (RBA) model which operates on randomly moving Agents within given four different population density areas. The RBA models the three infectious stages of influenza: (1) infected (2) contagious, and (3) resistant. A hypothetical influenza virus is spreads in four hypothetical areas, all with different population densities. In order to compare our program with truth data, the RBA output data was converted into a graph form to analyze the shape of our graph compared to the shape of a 2009 H1N1 truth data set from www.fludb.org. The RBA model employs a set of virus tuning parameters and Agent initial conditions which can be modified to follow the shape of a given truth data set.

2 Problem Statement

Influenza is an infectious disease characterized by a sore throat, coughing, congestion, fatigue and unfortunately sometimes death. There are numerous strains of the influenza virus that mutate within the course of the flu season. This “flu season” is when the disease is more prevalent than usual which occurs in late fall or winter. The flu spreads through the air of infected saliva or mucus fluids from carriers of the disease. It is very pervasive, taking over large areas very quickly. It can be spread from a distance, so it can be easily and rapidly transported from one person to the next. Being able to predict the speed and scale of an outbreak would help medical authorities be prepared for an outbreak with sufficient materials and an idea of how long it could take for the virus to go out of control if untreated. This task will be accomplished using C++ to create the RBA model with numerous variables to signify how fast the virus will spread through a selected region. The program will utilize an Agent-based modeling system with multiple Agents, each representing a person. These Agents will move in a random pattern over a given area. When two or more Agents, of which one at least is infected, are within a suitable distance of each other, the program will use each Agent's properties to calculate the likelihood of the uninfected Agent catching influenza from the infected Agent. These properties will include immunity and the amount of time an infected Agent will take to get well, among others.

3 RBA Method Used to Solve Problem

A rule based algorithm (RBA) that operates on independent Agents whose movement resides within a population density area was conceived to model four areas (Mexico, California, New York and Texas) where 2009 H1N1 truth data was available. Initially much consideration was given to utilizing a fixed mathematical model, but this was soon set aside due to the capabilities of employing computational random number generators, tuning parameters and algorithmic rules which can be manipulated to represent truth data. The RBA model architecture is displayed in Figure 1.

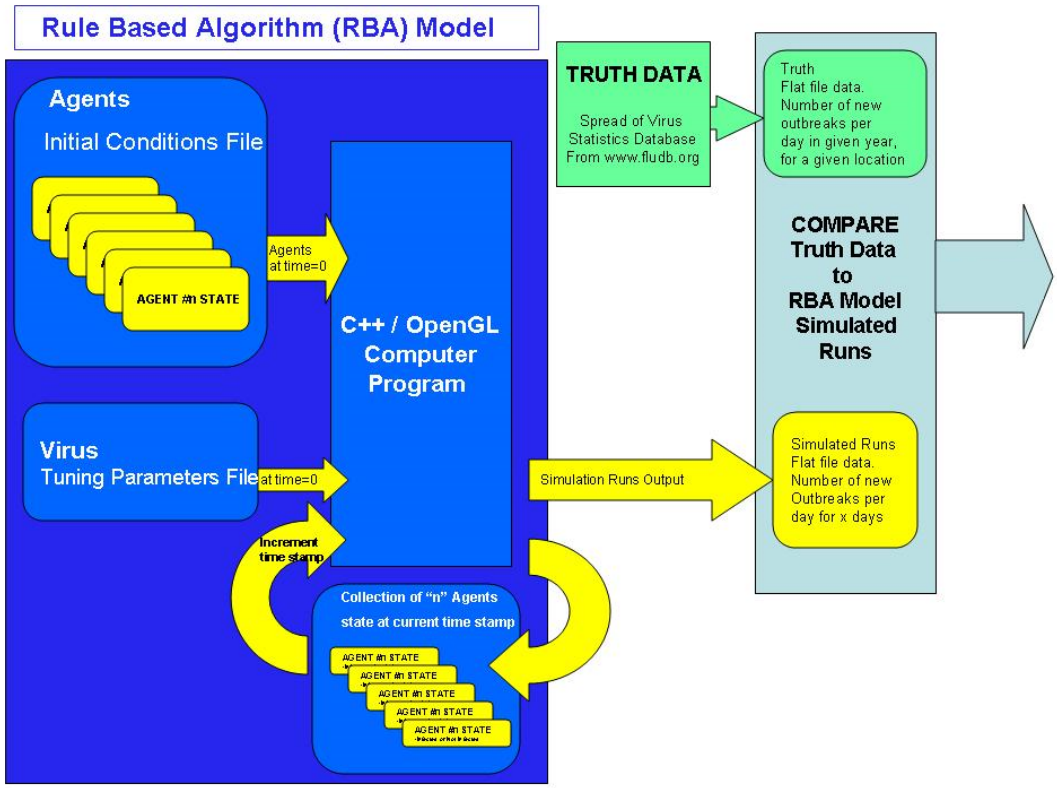

Figure 1 – RBA Model

The research within found adequate truth H1N1 data provided by www.fludb.org *(Influenza Research Database (IRD) [Squires et al. (2012) Influenza research database: an integrated bioinformatics resource for influenza research and surveillance. Influenza and Other Respiratory Viruses DOI: 10.1111/j.1750-2659.2011.00331.x.])* . The Center of Disease Control (CDC) was investigated, however its website and availability of data was no longer obtainable during the mid through latter time period of this project.

The RBA model was conceived after an extensive research of the current available literature found through internet searches. The following is a brief summary of some of the design attributes attained through the following citing found that influenced the design of the RBA model. *Algorithms on Strings*$_A$: Patterns on how to read in text data within a software application was utilized to efficiently read in the Agents data. *Appgen*$_B$: Basic social interaction was studied on Facebook within this report, and the spread of an infection was stated to be based upon how often one interacts with others, if they were sick, or how they felt. Not everyone has the same behavior. Social interaction mobility was implemented by the RBA model's tuning parameter V_TravelActivity. *Spread on Epidemic Disease on Networks*$_C$: This report discussed the usage mathematical modeling to simulate an epidemic. This was used to weigh the pros and cons of mathematical modeling because comparably it is simpler to add random movements to Agents in a rule based model. *Containing Pandemic Influenza with Antiviral Agents*$_D$: This article set up a simulation model based on tuning parameters and estimated fifty-seven day of infection on average produced by one-thousand runs, and three-hundred thirty-four cases of influenza per thousand people. This simulation used twelve seed

Agents with the virus added into the simulation at different times. The project's RBA model also has seeded individuals found in the file myAgentsFile.txt. *Influenza Pandemic Preparedness Planning Tool*$_{E}$: This article discussed using an R-Naught (R0) of 2.5. Herein, the RBA model enables the R-Naught to be tuned to any value with the parameter V_RNaught. The article's planning tool is stated to contain variables with assumptions based on Agent interaction in social networks and can be random. It is noted that RBA Agents movement is random as well. *Biowar Scalable Agent-Based Model of Bio Attacks*$_{F}$: Agents based interaction in social network is random. In this article, the social network size and contacts are tunable for each Agent. The RBA model too is random but also is situated within areas of different densities for dissimilar outcomes and is also tunable.

RBA contains tuning parameters to enable the simulation of "what if" scenarios with multiple run simulations. This gives the end user the ability to input different situations and infection rates and then run multiple runs to see how the different infection rates play out. Virus tuning parameters that can be modified are provided in Table 1. The V_ContagiousDistance is how many units in the distance the infected Agent needs to be in order to infect a non-infected Agent. When an Agent has been infected, after a given number of days, the newly infected Agent will become contagious; this is called V_StartDayToBeContagious. The number of days the Agent can be contagious is V_EndDayToBeContagious. R Naught is the number of Agents an infected Agent can infect, this is the parameter V_RNaught. During a simulation run, the number of minutes between each sample is V_MinutesInEachSample. Percent mortality, labeled as

V_PercentMortality, is available for future use. Last, V_HealthToGetInfect, when the Agent's health is under this level of health, the Agent will get infected. The initial conditions Agent's database is given in Table 2 and this table contains brief explanation for each parameter.

Virus Tuning parameters from myVirusFile.txt
V_ContagiousDistance
V_StartDayToBeContagious
V_EndDayToBeContagious
V_RNaught
V_MinutesInEachSample
V_PercentMortality
V_HealthToGetInfect

Table 1: Virus Tuning Parameters

Agent Database	
Each Agent initial condition's data read from myAgentFile.txt	
and put into a unique AgentStructy for that agent.	
Initial conditions for all agents is initialized at the start of a simulation run	
A_Name;	// string from data file
A_Health;	// RANGE (0-100), 0=dead
A_InfectedByVirus;	// 0=no, 1=yes
A_StartDayToBeContagious;	// The day number that Agent is contagious
A_V_EndDayToBeContagious;	// computed on the fly, not read in
A_ImmuneTo;	// 0=no, 1=yes
A_TravelActivity;	// Max units the Agent can move per sample
A_Area_X;	// X unit location in area
A_Area_Y;	// Y unit location in area
A_ActivityAreaName;	// Location Name (Human Readable)
A_ActivityArea;	// Location Number (for computer computation)
A_CurrentRnaught_ktr;	// computed on fly, not read in
A_Mortality_Flag;	// ** FUTURE USE, NOT USED

Table 2: Agent Database Initial Conditions

The C++/Open GL RBA model computer program in its entirety is available in Section 7 and the flow chart for this program is given in Figure 2. When the program starts, main (seen in the upper left side) initializes Open GL structure for displaying the Agent's positions and plots for the data provided in myAgentsFile.txt and myVirusFile.txt then calls initSimulationRun. After the simulation starts with the User clicking the mouse inside the plot area, the computeMyAgentsLocation will be in a continuous loop until the last simulation run is reached. Each simulation run is 300 days long. Next, infectMyAgents is called, and when the end of the 300 days is reached, then initSimulationRun function will start again and read in myAgentsFile.txt and myVirusFile.txt data.

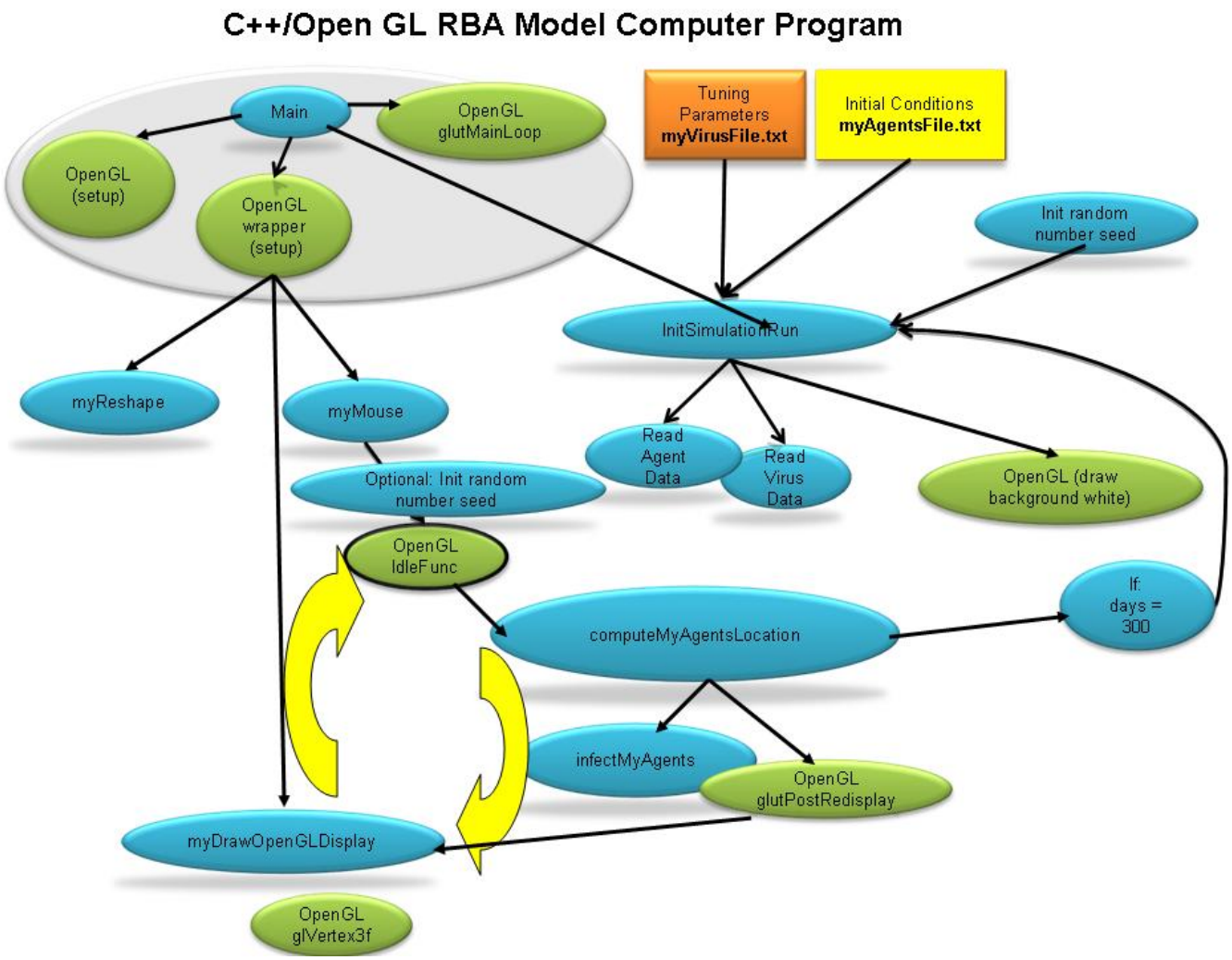

Figure 2: C++ / Open GL RBA Model Flow Chart

4 How Model was Verified and Validated

The RBA model was verified and validated by comparing it with actual fludb.org H1N1 2009 truth data. In order to compare the RBA model with truth data there had to be a media in which both data sets can be compared. This form of media is a graph. Even though the RBA model sample size was much smaller (number of samples) then the truth data, it was compared by comparing the shape of the two graphs. The influenza truth data was obtained with the following steps.

The User opens up a web browser and typed in the following URL: www.fludb.org.

PART 1: Download Data (Instructions performed 2013 Jan)

www.fludb.org -> Workbench Sign In -> Use your account. (create one)
-> Search Data Tab -> Sequences and Strains -> Nucleotide Sequences Search Page -> (Influenza A) **Subtype = H1N1**, Date Range = 2009 to 2010 ->
Geographic Groupings = North America -> Host = Human ->
-> Search -> Select All 20,087 segment -> (Bottom pf web page) download ->
Tab Delimited -> Filename = "H1N1_2009_2010_NorthAmerica_Human.txt ->
wait...
when done, logout.

PART 2: Manually Clean Up Data

Clean up Data text file,
Remove formats not conventional.
Remove data with year only, or year-month only.
Keep Data with Year/Month/Day.
Remove 2010 data.

PART 3: Import text data, space/tab delimited into Excel to a pivot plot.

Microsoft Office 2003
A:
-> Select 2 columns of data (date, location) ->
-> Menu -> Data -> Pivot Table and ... ->
-> Pivot Chart Table = kind -> Finish
B:
-> Drag Date to the lower x axis area ->
-> Drag State_Province to the center of plot area ->
-> Drag State_Province to the center right (total label) box ->
In Excel -> adjust as appropriate.

For completeness, the Center For Disease Control (CDC) was also investigated for data,

but was not selected for the following reasons. Website was valid December 2012, however the Website not available Jan 24 2013. Raw data was no longer found through CDC at this time. The URL was www.cdc.gov/flu.weekly/usservdata.htm (This CDC web page is no longer available)

Fludb.org initial analysis provided the number of strains of the H1N1 virus for regions selected and is seen in Figure 3.

- Use 300 days of data to be consistent with the RBA model that displays four 300x300 pixel unit areas on the display. 1 January 2013 through 27 October 2013.
- H1N1 Subtype/2009/Mexico/260 Unique Strain Names of the Virus in 2009
- H1N1 Subtype/2009/California/361 Unique Strain Names of the Virus in 2009
- H1N1 Subtype/2009/New York/377 Unique Strain Names of the Virus in 2009
- H1N1 Subtype/2009/Texas/314 Unique Strain Names of the Virus in 2009

After this data was gathered and parsed, the following truth data chart was generated

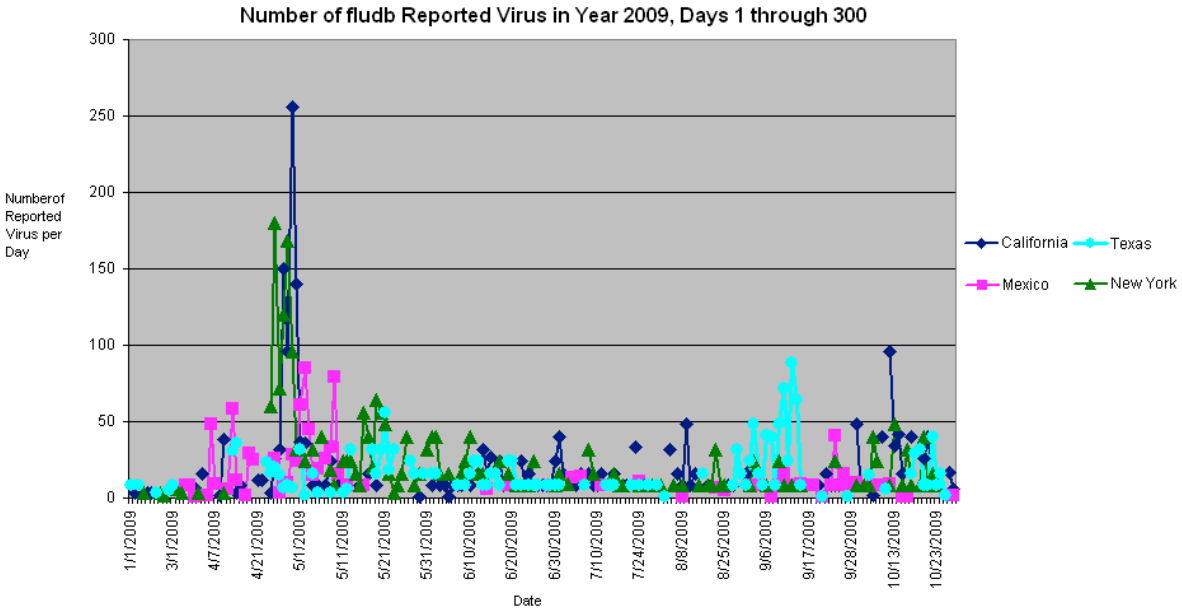

Figure 3: fludb H1N1 2009 Truth Data

Output Color Legend
Green Agent = Not Infected
Blue Agent = Infected, Not Contagious
Red Agent = Infected, Contagious
Black Agent = Immune
Red plot line in each area is the total Agents infected up to that day.
Black plot line in each area is the total Agents infected for a single day.

Table 3: Data Curve and Agent Color Assignment

In Figure 4, the horizontal axis in each ranges from 0 to 300 days. The vertical axis range is also from 0 to 300. The Daily Infected and Sum Infected plots are scaled by ten for ease of viewing. A single simulation screen capture is seen in Figure 4.

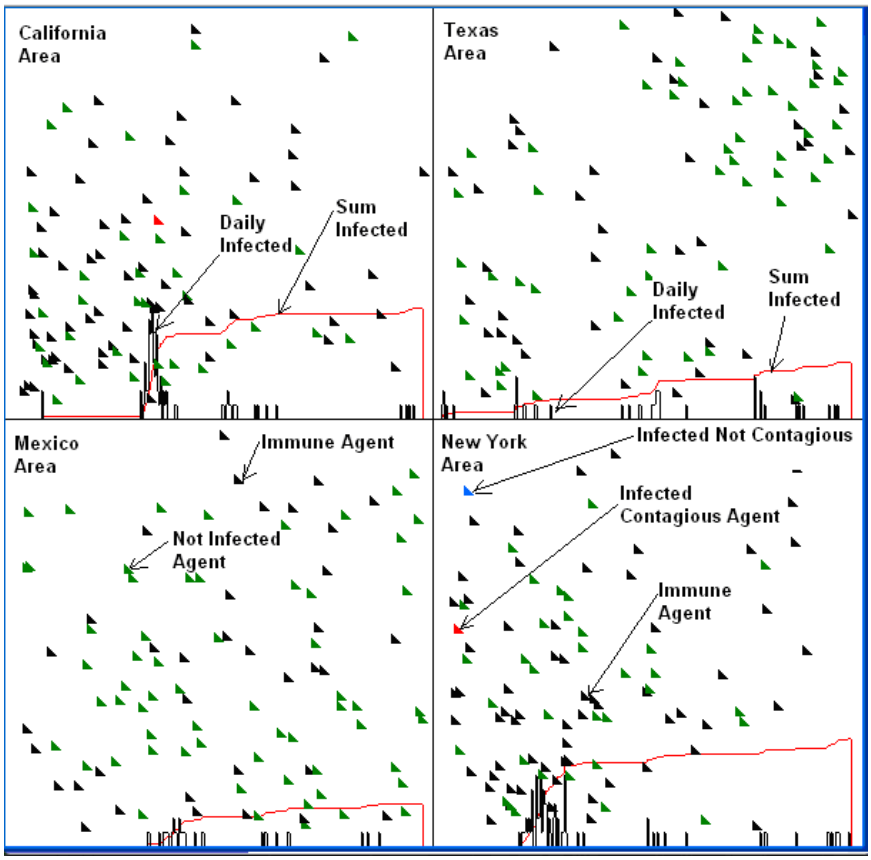

Figure 4: RBA Model running in 4 areas
Each Area's horizontal axis is 300 days

5 Results of Study

The RBA model is robust, meaning it can be tuned to represent many different variations and create more than one outcome. Each run in the RBA is truly a random output. Averages of simulated runs were calculated, however the more runs there are, the stronger the bias towards zero because there are so many zero values that are added to each simulation run. As a result, the more simulated runs that are used to compute the

average, the more zeros it includes and it gets pushed towards zero as well. So averages across many runs were not provided. As seen in Figure 4, each of the four areas consists of 300x300 pixels, and each area contains 100 Agents. The File that contains the initial conditions of the agents within the four areas is called myAgentsFile.txt and the program's initial conditions file is called myUserFile.txt. These may generally be held in c:\work directory. The range of the horizontal axis spans 300 days in each area. The result of a single run of the RBA model is given in Figure 5 which is compared with Figure 3 area by area side by side in Figures 6, 7, 8 and 9.

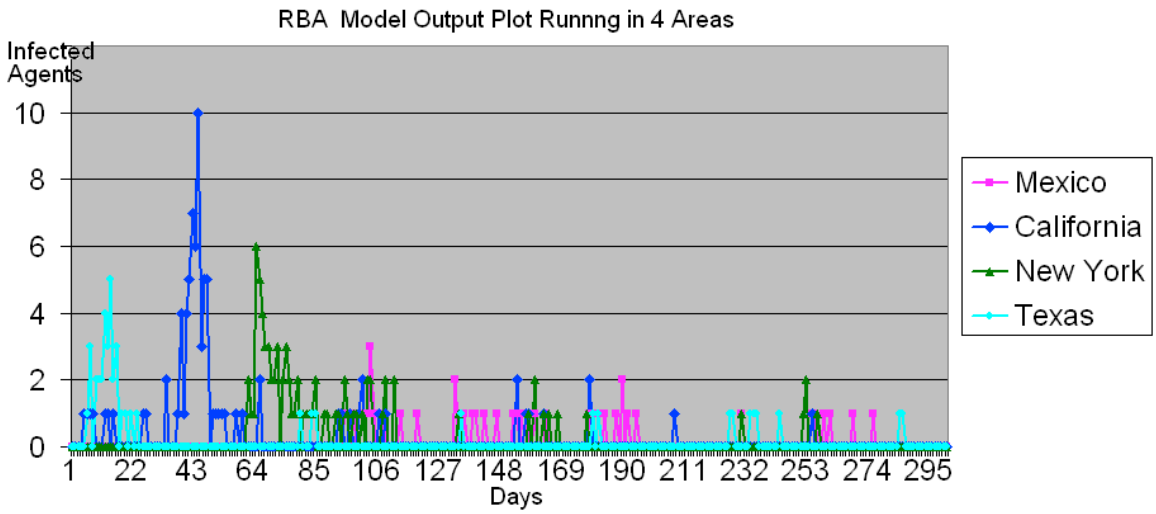

Figure 5: RBA Model Simulated Data

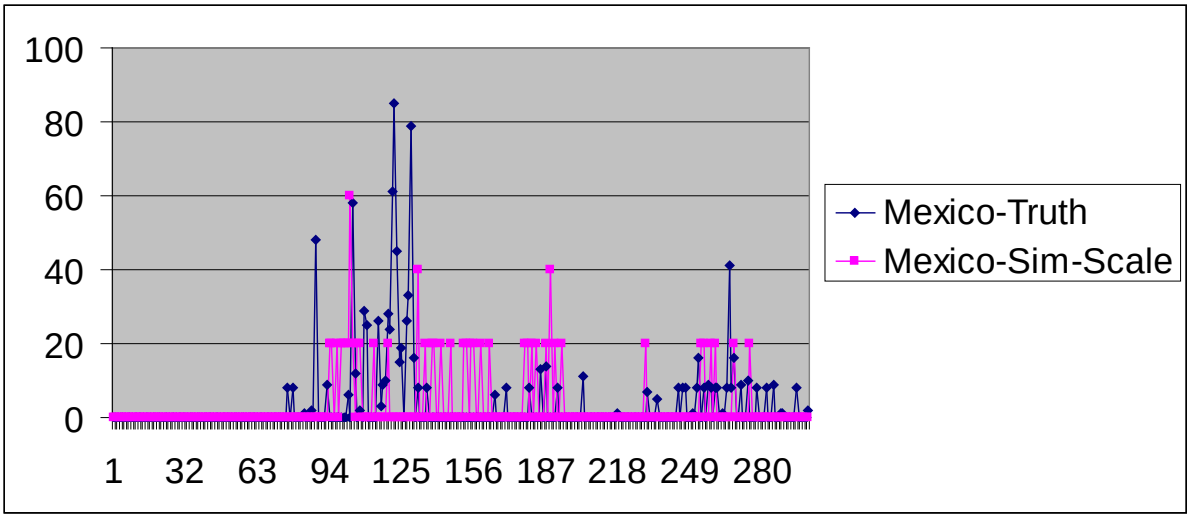

Figure 6: fludb Truth Data –vs- RBA Simulated Data Scaled
Area: Mexico

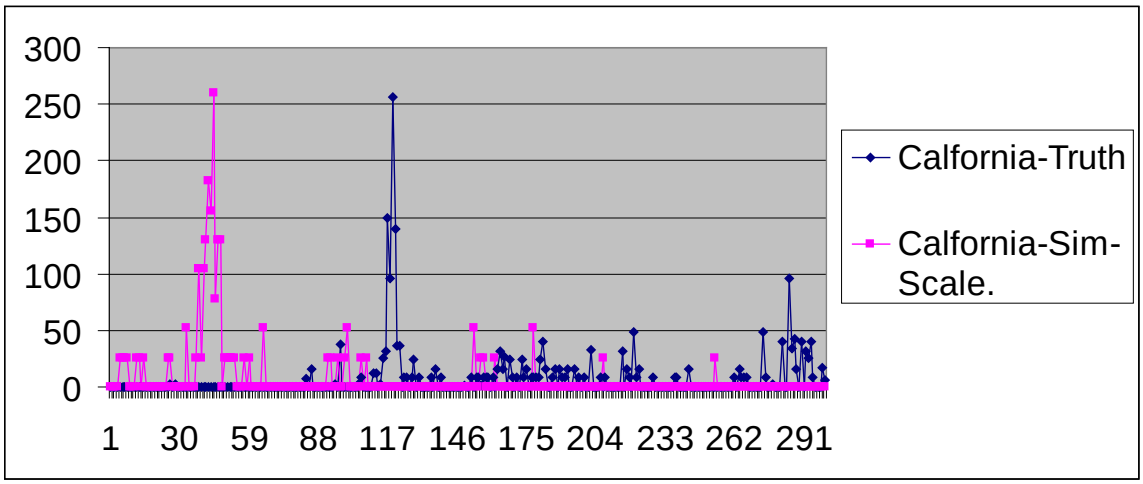

Figure 7: fludb Truth Data –vs- RBA Simulated Data Scaled
Area: California

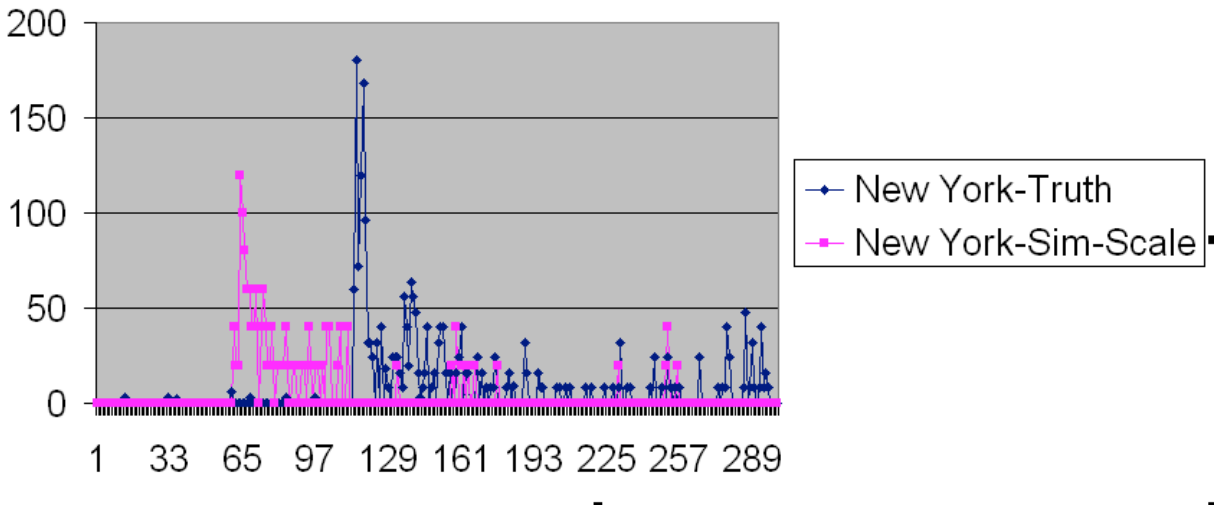

Figure 8: fludb Truth Data –vs- RBA Simulated Data Scaled Area: New York

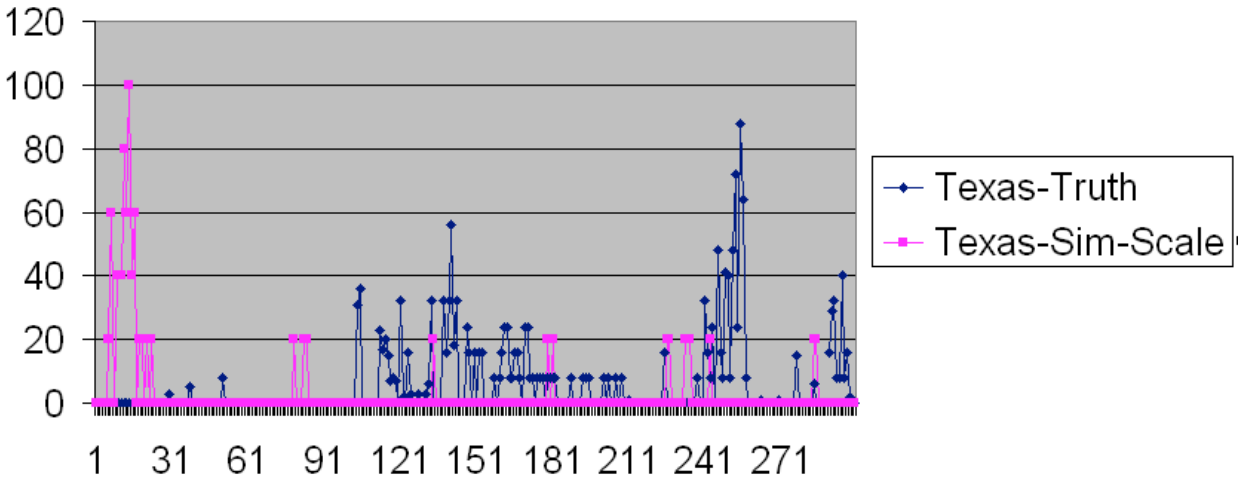

Figure 8: fludb Truth Data –vs- RBA Simulated Data Scaled Area: Texas

6 Conclusions Reached by Analyzing Results

The Rule Based Programming (RBA) model can be used to approximately predict the propagation of a virus through a population. As was discovered in the literature, there is much tuning to the data that needs to be performed, and it can be a hit or miss in

comparison with real data. The RBA model with its tuning parameters does exhibit similar behavior in high Agent population density areas in comparison to actual data. However the “randomness” of real data and the RBA both possess that attribute. In other words, the speed of the spread of the influenza virus has many input variables, but a primary attribute is its “randomness” as was shown in this. This randomness is a contributing factor as to why the medical profession has difficulty pinpointing the design of the flu vaccine. Because each RBA simulation run produces different results, the more simulation runs that are performed the higher probability that one of them will closely match truth data.

7 Software, References, Citations, Tables and Plots

The RBA model source code is provided below.

7.1 Software

```
// ----------------------------------------------------
// Super Computing Project
// Albuquerque Academy Supercomputing
// Marcus D-K
// 2012-2013
// ----------------------------------------------------
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
#include <ctime>
#include <windows.h>
#include <GL/glut.h>
#include <cstdlib>    // rand
#include <ctime>      // rand
#include <math.h>

// DISPLAY AREAS
//----+----
//  1 | 3
//----+----
//  0 | 2
//----+----

//-----------------------------------------------------------
// GLOBAL CONSTANT VARIABLES, DO NOT CHANGE
//-----------------------------------------------------------
const int   MAX_AGENTS          = 1000;
const int   WIDTH               =  600;  // FOR DISPLAY
const int   HEIGHT              =  600;  // FOR DISPLAY
const int   BORDER              =   10;  // FOR DISPLAY
const int   WIDTH_AREA          =  300;  // FOR DISPLAY
const int   HEIGHT_AREA         =  300;  // FOR DISPLAY
const int   WIDTH_NUM_AREA      =    2;  // FOR DISPLAY
const int   HEIGHT_NUM_AREA     =    2;  // FOR DISPLAY
const int   DAYS_IN_SIM         =  300;  // DAYS IN A SIMULATION RUN
//-----------------------------------------------------------
// Virus Tuning parameters from myVirusFile.txt
//   Convention V_ = Virus
//   Every Virus should have different unique values
//-----------------------------------------------------------
double      V_ContagiousDistance      =  0.0;
int         V_StartDayToBeContagious  =    0;
int         V_EndDayToBeContagious    =    0;
int         V_RNaught                 =    0; // NUMBER AGENTS WILL CONTAGIOUS AGENT INFECT
int         V_MinutesInEachSample     =    0; // minutes for each agents movement
```

```
int         V_PercentMortality              =   0; // % die // FUTURE
int         V_HealthToGetInfect             =   0;
//---------------------------------------------------------------------------
// Run.  GLOBAL VARIABLES AND ARRAYS, SET TO ZERO in initSimulationRun()
//   Convention R_ = Run, Current Simulation RUN
//---------------------------------------------------------------------------
int         R_NumAgents;
int         R_AgentArea;
int         R_MinuteKtr;
int         R_DayKtr   ;
int         R_DayInfections_DayKtr [WIDTH_NUM_AREA * HEIGHT_NUM_AREA];
int         R_DayInfectionsTotal     [WIDTH_NUM_AREA * HEIGHT_NUM_AREA];
int         R_AreaPlot_SumArray    [WIDTH_NUM_AREA * HEIGHT_NUM_AREA][DAYS_IN_SIM];
int         R_AreaPlot_DayArray    [WIDTH_NUM_AREA * HEIGHT_NUM_AREA][DAYS_IN_SIM];
//---------------------------------------------------------------------------
// GLOBAL ARRAYS, SUM=RUNNING SUM, AVG AND Rate COMPUTED ON THE FLY
//---------------------------------------------------------------------------
int          AreaPlot_SUM    [WIDTH_NUM_AREA * HEIGHT_NUM_AREA][DAYS_IN_SIM];
int          AreaPlot_AVG    [WIDTH_NUM_AREA * HEIGHT_NUM_AREA][DAYS_IN_SIM];

using namespace std;
//---------------------------------------------------------------------------
// AgentStruct()
//   Each Agent initial condition's data read from myAgentFile.txt
//   in put into a unique AgentStructy for that agent.
//   Initial conditions for all agents is initialized at the start of a simulation run
//   Convention A_ = Agent's properties which change throughout each simulation run.
//---------------------------------------------------------------------------
struct AgentStruct
{
 public:
            string      A_Name;                         // string from data file
            int         A_Health;                       // RANGE (0-100), 0=dead,
            int         A_InfectedByVirus;              // 0=no, 1=yes
            int         A_StartDayToBeContagious;       // Day Agent is contagious
            int         A_V_EndDayToBeContagious;       // **COMPUTED ON FLY WITH VIRUS PARAMETER
            int         A_ImmuneTo;                     // 0=no, 1=yes
            int         A_TravelActivity;               // Maximum number of units the Agent can move per
sample
            double      A_Area_X;                       // X unit location in area
            double      A_Area_Y;                       // Y unit location in area
            string      A_ActivityAreaName;             // Location Name (Human Readable)
            int         A_ActivityArea;                 // Location Number (for computer computation)
            int         A_CurrentRnaught_ktr;           // ** Total Counts how many people this Agent has infected
            int         A_Mortality_Flag;               // ** FUTURE USE, NOT USED
  private:
            double      length;  // FUTURE
};
//-------------------------------
// GLOBAL VARIABLES
//-------------------------------
AgentStruct         Agents[MAX_AGENTS];
ofstream            myOutFile;
string              MY_DIRECTORY_NAME;
int                 runCurrent  =   0;              // INITIAL VALUE = 0, RESET in initSimultionRun()
int                 runsTOTAL   =   10;             // Default=10, USE USER INPUT COMMAND LINE
```

```
// ---------------------------------------------------------------------------
// infectMyAgents()
//   called by computeMyAgentsLocation()
//   A=Agent1, B=Agent2
//----------------------------------------------------------------------------
void infectMyAgents()
{
   double dist;
   int   a1, a2, b1, b2, a3, b3;              //  X,Y LOCATION VARIABLES ON DISPLAY

for( int A=0; A<R_NumAgents; A++ ){
   //-------------------------------------------------------------------------
   // (1) CHECK IF AGENT-A IS INFECTED, IF YES THEN CHECK IF AGENT-A RECOVERED.
   //         NOTE:  EndtDay < R_DayKtr
   //-------------------------------------------------------------------------
   if(( Agents[A].A_InfectedByVirus                 == 1          ) &&      // IF: Yes, infected
      ( Agents[A].A_Health                          > 0           ) &&      // IF: Not dead
      ( Agents[A].A_V_EndDayToBeContagious < R_DayKtr )) {        // IF: Contagious Period is Finished

            Agents[A].A_InfectedByVirus  = 0;                              // SET: No Longer Infected.
            Agents[A].A_ImmuneTo         = 1;                                    // SET: Is Now Immune
            R_AgentArea                  = Agents[A].A_ActivityArea;

             cout <<"Day:"<<R_DayKtr<<" Area:"<<R_AgentArea
               <<"  Agent-A "<< Agents[A].A_Name<< "  Recovered" <<endl;
   }
   //-------------------------------------------------------------------------
   // (2) CHECK IF AGENT-A CAN INFECT
   //         NOTE:  StartDay < R_DayKtr < EndDay
   //-------------------------------------------------------------------------
   if(( Agents[A].A_InfectedByVirus              > 0              ) &&   // IF: Infected
      ( Agents[A].A_Health                       > 0              ) &&   // IF: not dead
      ( Agents[A].A_StartDayToBeContagious <= R_DayKtr   ) &&   // IF: In Contagious time
      ( Agents[A].A_V_EndDayToBeContagious >  R_DayKtr ) &&   // IF: In Contagious time
      ( Agents[A].A_CurrentRnaught_ktr     <  V_RNaught)    ){       // IF: R0 ktr not yet reached

for( int B=0; B<R_NumAgents; B++ ){
            //-----------------------------------------------------
            // (3) CHECK IF AGENT-B CAN BE INFECTED
            //-----------------------------------------------------
            if((A != B)                                                          && // IF: Agent-A not Agent-B, can't infect
itself
               (Agents[A].A_ActivityArea       == Agents[B].A_ActivityArea)  && // IF: Agent-A and Agent-B in same area
               (Agents[B].A_InfectedByVirus == 0  )                          && // IF: Agent-B not infected
               (Agents[B].A_ImmuneTo        == 0 )                           && // IF: Agent-B not immune
               (Agents[B].A_Health         <  V_HealthToGetInfect )){             // IF:  Agent-B not good health

                 a1   = Agents[A].A_Area_X;
                 b1   = Agents[A].A_Area_Y;
                 a2   = Agents[B].A_Area_X;
                 b2   = Agents[B].A_Area_Y;
                //-----------------------------------------------
                // COMPUTE THE DISTANCE FROM AGENT-A AND AGENT-B (dist)
```

```
                //-------------------------------------------------------
                a3   = (a2-a1)*(a2-a1);
                 b3   = (b2-b1)*(b2-b1);
                 dist = sqrt((double)(a3) + (double)(b3));
                 //-------------------------------------------------------------
                 // (4) IF DISTANCE CLOSE, THEN SECOND AGENT GETS INFECTED. *****
                 //-------------------------------------------------------------
                if (dist < V_ContagiousDistance ) {

                    Agents[A].A_CurrentRnaught_ktr++;       // AGENT-A INFECTED ONE MORE OTHER AGENT

                    // SET: window of time where AGENT-B will be Contagious (start and end day)
                    Agents[B].A_StartDayToBeContagious    = R_DayKtr + V_StartDayToBeContagious;
                    Agents[B].A_V_EndDayToBeContagious = R_DayKtr + V_EndDayToBeContagious  ;
                    Agents[B].A_InfectedByVirus                = 1;
                    R_AgentArea                                 = Agents[B].A_ActivityArea;
                    R_DayInfections_DayKtr[R_AgentArea]++ ;   // Incr count for num of Agents infected this Area

                      cout<<"Day:"<<R_DayKtr<<" Area:"<<R_AgentArea<<"  Agent-A "<<Agents[A].A_Name<<
                              " Infected Agent-B: "<<Agents[B].A_Name<<Agents[B].A_ActivityAreaName<< endl;
                     } // (4)
                     } // (3)
            } // AGENT-B LOOP
          } // (2)
    } // AGENT-A LOOP
} // END: infectMyAgents()


//-------------------------------------------------------------------------------
// AgentFactory()
//   called by initSimulationRun()
//-------------------------------------------------------------------------------
int AgentFactory ( AgentStruct Agents[MAX_AGENTS])
{
    string x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11;
            //-------------------------------------------------------
            // CREATE myBlankAgent STRUCT TO INITIALIZE AGENTS
            //-------------------------------------------------------
            AgentStruct myBlankAgent;
                    myBlankAgent.A_Name                         = "None";
                    myBlankAgent.A_Health                       = 0;
                    myBlankAgent.A_InfectedByVirus              = 0;
                    myBlankAgent.A_StartDayToBeContagious       = 0;
                    myBlankAgent.A_V_EndDayToBeContagious       = 0 ;
                    myBlankAgent.A_ImmuneTo                     = 0;
                    myBlankAgent.A_TravelActivity               = 0;
                    myBlankAgent.A_Area_X                       = 0.0;
                    myBlankAgent.A_Area_Y                       = 0.0;
                    myBlankAgent.A_ActivityAreaName             = "None";
                    myBlankAgent.A_ActivityArea                 = 0;
                    myBlankAgent.A_CurrentRnaught_ktr           = 0;
                    myBlankAgent.A_Mortality_Flag               = 0;
    //-------------------------------------------------------
    // INITIALIZE Agents[] STRUCT ARRAY to myBlankAgent STRUCT
    //-------------------------------------------------------
    for(int i=0; i<MAX_AGENTS; i++ ) Agents[i]=myBlankAgent;
```

```
    //---------------------------------------------------------
    // Read Agents data from File
    //---------------------------------------------------------
    int       agentsKtr=0;
    ifstream  myInFile;
    string    myLine;
    string    AgentsFileName = MY_DIRECTORY_NAME + "myAgentsFile.txt" ;
    myInFile.open (&AgentsFileName[0], ios::in );

    if( !myInFile) {
              cerr << "myProgram:  Can't open myAgentsFile.txt, exiting" <<  endl;
              return (-1);
    }else{
              getline(myInFile,myLine);  // READ IN HEADER, DISCARD, BECAUSE NOT USED.
              while (!myInFile.eof()){   // READ EACH LINE, PUT EACH LINE INTO AGENT ARRAY.

                        //-------------------------------------------------------------------------
                        // From File, put each parsed (space delimited) variable from x columns into x variables
                        //-------------------------------------------------------------------------
      myInFile >> x1 >> x2 >> x3 >> x4 >> x5 >> x6 >> x7 >> x8 >> x9 >> x10;

      cout << agentsKtr<< ":PARSED=" <<x1<<" | "<<x2<<" | "<<x3<<" | "<<x4<<" | "
                          <<x5<<" | "<<x6<<" | "<<x7<<" | "<<x8<<endl;

                        Agents[ agentsKtr ].A_Name                          = x1;
                        Agents[ agentsKtr ].A_Health                        = atoi(x2.c_str());
                        Agents[ agentsKtr ].A_StartDayToBeContagious        = atoi(x3.c_str());
                        Agents[ agentsKtr ].A_InfectedByVirus               = atoi(x4.c_str());
                        Agents[ agentsKtr ].A_ImmuneTo                      = atoi(x5.c_str());
                        Agents[ agentsKtr ].A_TravelActivity                = atoi(x6.c_str());
                        Agents[ agentsKtr ].A_Area_X                        = (double )atof(x7.c_str());
                        Agents[ agentsKtr ].A_Area_Y                        = (double )atof(x8.c_str());
                        Agents[ agentsKtr ].A_ActivityAreaName              = x9;
                        Agents[ agentsKtr ].A_ActivityArea                  = atoi(x10.c_str());
                Agents[ agentsKtr ].A_V_EndDayToBeContagious                =
Agents[agentsKtr].A_StartDayToBeContagious
                                                                              + V_EndDayToBeContagious;
                        agentsKtr++;
            }}
    myInFile.close();

    agentsKtr--;                                // for the last one.
    Agents[agentsKtr] = myBlankAgent; // for the last one.
    R_NumAgents       = agentsKtr;
    cout << "Number of Simulation Agents In File is :" << R_NumAgents << endl;

    return (agentsKtr);
}// END: AgentFactory()
```

```
//---------------------------------------------------------------------------
// readVirusData()
//   called by initSimulationRun()
//   The Virus tunable behavior in this simulation is read from myVirusFile.txt
//---------------------------------------------------------------------------
int readVirusData( )
{
        string      z1, z2, z3, z4, z5, z6, z7;
        string      myLine;
        ifstream    myInFile2;
        string      myUserFileName = MY_DIRECTORY_NAME + "myVirusFile.txt" ;
        myInFile2.open (&myUserFileName[0], ios::in );

        if( !myInFile2) {
                cerr << "myProgram:  Can't open input file 2, exiting" <<  endl;
                return (-1);
        }else{
                getline(myInFile2,myLine);                                  // READ IN HEADER
                 myInFile2 >> z1 >> z2 >> z3 >> z4 >> z5 >> z6 >> z7 ;   // READ IN ONE LINE OF DATA
                 cout << "VirusFile:PARSED= " <<z1<<" | "<<z2<<" | "<<z3<<" | "<<z4<<
                                                 " | "<<z5<<" | "<<z6<<" | "<<z7<<endl;

                V_ContagiousDistance          = (double )atof(z1.c_str());
                V_StartDayToBeContagious      =         atoi(z2.c_str());
                V_EndDayToBeContagious        =         atoi(z3.c_str());
                V_RNaught                     =         atoi(z4.c_str());
                V_MinutesInEachSample         =         atoi(z5.c_str());
                V_PercentMortality            =         atoi(z6.c_str());
                V_HealthToGetInfect           =         atoi(z7.c_str());
   }
   myInFile2.close();

   return(1);
}// End readVirusData()


// ---------------------------------------------------------------------------
// initSimulationRun()
//   called by main()
//   called by computeMyAgentsLocation()
//---------------------------------------------------------------------------
void initSimulationRun()
{
   // INITIALIZE CURRENT SIMULATION VARIABLES TO ZERO
   R_NumAgents  = 0;
   R_MinuteKtr  = 0;
   R_DayKtr     = 0;
   R_AgentArea  = 0;

   // INITIALIZE CURRENT SIMULATION PLOT DATA AND ARRAY TO ZERO
   // ZERO OUT DATA
   for( int area=0; area<(WIDTH_NUM_AREA * HEIGHT_NUM_AREA); area++ ){

              R_DayInfections_DayKtr[area]          = 0;
```

```
            R_DayInfectionsTotal  [area]          = 0;

        for( int d=0; d<DAYS_IN_SIM; d++ ){
                R_AreaPlot_SumArray    [area][d]    = 0;
                R_AreaPlot_DayArray[area][d]        = 0;
        }}
         readVirusData( );                          // ##### LOCAL //
        AgentFactory( Agents );                     // ##### LOCAL //
        srand(time(0));                             // GIVE SEED TO RANDOM NUMBER GENERATOR
         glClearColor (1.0, 1.0, 1.0, 0.0);         // WHITE BACKGROUND CANVAS COLOR
        glShadeModel (GL_FLAT);
} // END:  initSimulationRun()

// ---------------------------------------------------------------------------------------
// ComputeMyAgentsLocation()
//   called by myMouse().
//---------------------------------------------------------------------------------------
void computeMyAgentsLocation()
{
        int a, b, d2, y0, y1, y2, y3;
        for( int i=0; i<R_NumAgents; i++ ){
                a = rand() % (3) - 1;
                b = rand() % (3) - 1;
                //-------------------------------------------------------------------------
                // SET: AGENT LOCATION BY SOME RANDOM NUMBER AND TRAVEL ACTIVITY
                //-------------------------------------------------------------------------
                Agents[i].A_Area_X = Agents[i].A_Area_X + (float )a * (float)Agents[i].A_TravelActivity;
                Agents[i].A_Area_Y = Agents[i].A_Area_Y + (float )b * (float)Agents[i].A_TravelActivity;
                //-------------------------------------------------------------------------
                // MAKE SURE AGENTS ARE DRAWN WITHIN BORDER
                //-------------------------------------------------------------------------
                if(Agents[i].A_Area_X <          BORDER) Agents[i].A_Area_X  =
                        BORDER;
                if(Agents[i].A_Area_X > WIDTH_AREA -  BORDER) Agents[i].A_Area_X  =
                        WIDTH_AREA  -  BORDER;
                if(Agents[i].A_Area_Y <          BORDER) Agents[i].A_Area_Y =
                        BORDER;
                if(Agents[i].A_Area_Y > HEIGHT_AREA - BORDER) Agents[i].A_Area_Y =
                        HEIGHT_AREA -  BORDER;
        }
        // ------------------------------------------------------
        // CHECK IF AGENTS WITH NEW LOCATION, INFECT EACH OTHER
        // ------------------------------------------------------
         infectMyAgents();  // ##### LOCAL //
        R_MinuteKtr += V_MinutesInEachSample;

        // ------------------------------------------------------
        // CHECK IF NEXT DAY (1440 minutes), IF YES, READY NEXT DAY
        // ------------------------------------------------------
         if( R_MinuteKtr > 1440 ) {

                for( int area=0; area<(WIDTH_NUM_AREA * HEIGHT_NUM_AREA); area++ ){

                R_DayInfectionsTotal[area]              += R_DayInfections_DayKtr[area];
                        R_AreaPlot_SumArray [area][R_DayKtr] = R_DayInfectionsTotal  [area];
                                // SUM PLOT ALL DAYS
```

```
                    R_AreaPlot_DayArray [area][R_DayKtr] = R_DayInfections_DayKtr[area];
                            // CURRENT PLOT ONE DAY

                    R_DayInfections_DayKtr[area]        = 0; // SET: day ktr to zero for next day
}

        R_DayKtr++;       // increment day counter
        R_MinuteKtr = 0;  // SET: minute ktr to zero
}

//-----------------------------------------------
// IF COMPLETED THIS YEAR's SIMULATION RUN
// >>WRAP UP<<, THEN START NEW SIMULATION RUN
//-----------------------------------------------
if( R_DayKtr > DAYS_IN_SIM){
        //-------------------------------------------------------
        // SAVE: BY WRITING TODAY's DATA TO mySIM_Plots.txt
        //-------------------------------------------------------
        string myOutFileName = MY_DIRECTORY_NAME + "mySIM_Plots.txt";
        myOutFile.open(&myOutFileName[0]   , ios::app);

        for( int d=0; d<DAYS_IN_SIM; d++ ){
                y0 = R_AreaPlot_DayArray[0][d] ;// VERTICAL AXIS
                y1 = R_AreaPlot_DayArray[1][d] ;// VERTICAL AXIS
                y2 = R_AreaPlot_DayArray[2][d] ;// VERTICAL AXIS
                y3 = R_AreaPlot_DayArray[3][d] ;// VERTICAL AXIS
                d2 = d + (runCurrent * DAYS_IN_SIM) + 10; // HORIZONTAL AXIS

                myOutFile << d2 << ", " << y0 << ", " << y1 << ", " << y2 << ", " << y3 << endl;
        }
        myOutFile.close();

        runCurrent++;
        //-----------------------------------------------------------------
        // COMPUTE: AVERAGE NEW FLU PER DAY
        // SAVE:    AVERAGE NEW FLU PER DAY to myAVGERAGE_Runs_Plots.txt
        //-----------------------------------------------------------------
        for( int d=0; d<DAYS_IN_SIM; d++ ){

                AreaPlot_SUM[0][d] += R_AreaPlot_DayArray[0][d]; // add for sum all infections for area
                AreaPlot_SUM[1][d] += R_AreaPlot_DayArray[1][d]; // add for sum all infections for area
                AreaPlot_SUM[2][d] += R_AreaPlot_DayArray[2][d]; // add for sum all infections for area
                AreaPlot_SUM[3][d] += R_AreaPlot_DayArray[3][d]; // add for sum all infections for area

                y0 = AreaPlot_SUM[0][d] / runCurrent; // compute average per day, current run
                y1 = AreaPlot_SUM[1][d] / runCurrent; // compute average per day, current run
                y2 = AreaPlot_SUM[2][d] / runCurrent; // compute average per day, current run
                y3 = AreaPlot_SUM[3][d] / runCurrent; // compute average per day, current run

                AreaPlot_AVG[0][d] = y0;  // This is plotted as average, zeros included in avg
                AreaPlot_AVG[1][d] = y1;  // This is plotted as average, zeros included in avg
                AreaPlot_AVG[2][d] = y2;  // This is plotted as average, zeros included in avg
                AreaPlot_AVG[3][d] = y3;  // This is plotted as average, zeros included in avg

                d2 = d + 10; // + (runCurrent * DAYS_IN_SIM) + 10; // HORIZONTAL AXIS
```

```
                        if ( runCurrent >= runsTOTAL ){ // last run, write to file
                                string myOutFileName2 = MY_DIRECTORY_NAME +
                                "myAVGERAGE_Runs_Plots.txt";
                        myOutFile.open(&myOutFileName2[0]   , ios::app);
                        myOutFile << d2 << ", " << y0 << ", " << y1 << ", " << y2 << ", " << y3 << endl;
                        myOutFile.close();

                        if(d==(DAYS_IN_SIM-1))     exit(0);                    }}
                initSimulationRun();  // ##### LOCAL //
        }
        //---------------------------------------------
        // Some Agents are not immune, for virus mutations after certain days
        //---------------------------------------------
        if( (R_DayKtr ==  90) ||
            (R_DayKtr == 120) ||
            (R_DayKtr == 180) ){
                for(int i=0; i<MAX_AGENTS; i+=4 ) Agents[i].A_ImmuneTo = 0;
        }
        glutPostRedisplay();
} // END:  computeMyAgentsLocation()


//-------------------------------------------------------------------------------------------------
// myDrawOpenGL_Display()
//   Registered with OpenGL, Set in main()
//-------------------------------------------------------------------------------------------------
void myDrawOpenGL_Display()
{
        float T_SIZE=8.0; // SIZE OF TRIANGLE
         float x1,x2,x3;
        float y1,y2,y3;
        int CurActivityArea = 0;
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        //----------------
        // DISPLAY AREAS
        //----+----
        //  1 | 3
        //----+----
        //  0 | 2
        //----+---- x axis ->

        //---------------------------------------------------
        // PLOT: AVERAGE INFECTIONS (PURPLE)
        //---------------------------------------------------
        for( int x=0; x<WIDTH_NUM_AREA ; x++ ){
        for( int y=0; y<HEIGHT_NUM_AREA; y++ ){

        glBegin(GL_LINE_STRIP);
                glColor3f(1.0,0.0,1.0);

                CurActivityArea = (x*WIDTH_NUM_AREA) + y;
                for( int d=0; d<DAYS_IN_SIM; d++ ){
                        x1          = WIDTH_AREA *x + d;
                        y1          = HEIGHT_AREA*y + AreaPlot_AVG[CurActivityArea][d] * 10;
                        glVertex3f(x1,y1,0.0);
                }
```

```
glEnd();
}}
//---------------------------------------------------------
// PLOT: NUMBER OF TOTAL INFECTIONS to DATE (RED)
//---------------------------------------------------------
for( int x=0; x<WIDTH_NUM_AREA ; x++ ){
for( int y=0; y<HEIGHT_NUM_AREA; y++ ){

glBegin(GL_LINE_STRIP);
        glColor3f(1.0,0.0,0.0);

        CurActivityArea = (x*WIDTH_NUM_AREA) + y;
        for( int d=0; d<DAYS_IN_SIM; d++ ){
                x1      = WIDTH_AREA *x + d;
                y1      = HEIGHT_AREA*y + R_AreaPlot_SumArray[CurActivityArea][d];
                glVertex3f(x1,y1,0.0);
        }
glEnd();
 }}

//---------------------------------------------------------
// PLOT: NUMBER OF NEW INFECTIONS PER DAY (BLACK)
//---------------------------------------------------------
for( int x=0; x<WIDTH_NUM_AREA ; x++ ){
for( int y=0; y<HEIGHT_NUM_AREA; y++ ){

glBegin(GL_LINE_STRIP);
        glColor3f(0.0,0.0,0.0);
        CurActivityArea = (x*WIDTH_NUM_AREA) + y;
        for( int d=0; d<DAYS_IN_SIM; d++ ){
                x1 = WIDTH_AREA *x + d;
                y1 = HEIGHT_AREA*y + (R_AreaPlot_DayArray[CurActivityArea][d] * 10);
                glVertex3f(x1,y1,0.0);
        }
glEnd();
 }}
//---------------------------------------------------------
// DRAW: AGENTS COLOR AND LOCATION IN EACH AREA
//---------------------------------------------------------
glBegin(GL_TRIANGLES);
 for( int i=0; i<R_NumAgents; i++ ){

        //---------------------------------------------------------
        // SET COLOR OF CURRENT AGENT FOR OPEN_GL
        //---------------------------------------------------------
        glColor3f(0.0,0.5,0.0);                 // GREEN NOT INFECTED AGENT

        if(Agents[i].A_InfectedByVirus > 0) {
                glColor3f(0.0,0.0,1.0);         // BLUE INFECTED, NOT CONTAGIOUS AGENT

         if(( Agents[i].A_StartDayToBeContagious     <= R_DayKtr ) &&
            ( Agents[i].A_V_EndDayToBeContagious  >  R_DayKtr  )){
                        glColor3f(1.0,0.0,0.0);             // RED CONTAGIOUS AGENT
        }}
        if(Agents[i].A_ImmuneTo == 1) glColor3f(0.0,0.0,0.0);    // BLACK IMMUNE AGENT
```

```
                    // DETERMINE CURRENT AREA LOCATION
                    for( int x=0; x<WIDTH_NUM_AREA ; x++ ){
                    for( int y=0; y<HEIGHT_NUM_AREA; y++ ){

                            CurActivityArea = (x*WIDTH_NUM_AREA) + y;

                            if( Agents[i].A_ActivityArea == CurActivityArea){

                                    x1= WIDTH_AREA *x ;        y1= HEIGHT_AREA *y ;
                                    x2= x1                ;        y2= y1 + T_SIZE;
                                    x3= x1 + T_SIZE   ;        y3= y1;

                                     glVertex3f(Agents[i].A_Area_X + x1, Agents[i].A_Area_Y + y1, 0.0);
                                     glVertex3f(Agents[i].A_Area_X + x2, Agents[i].A_Area_Y + y2, 0.0);
                                     glVertex3f(Agents[i].A_Area_X + x3, Agents[i].A_Area_Y + y3, 0.0);
                    }}}
            }
            glEnd();


//-----------------------------------------------
// DRAW:  LINES SEPARATING FOUR ACTIVITY AREAS
//-----------------------------------------------
            glBegin(GL_LINES);
             glColor3f (0.0,0.0,0.0);
             glVertex2f(0,  300);  glVertex2f(600, 300);
             glVertex2f(300,  0);  glVertex2f(300, 600);
            glEnd();
            glFlush();
            glutSwapBuffers();
} // END:  myDrawOpenGL_Display()


//-------------------------------------------------------------------------------
// myReshape()
//   Registered with OpenGL, set in main()
//-------------------------------------------------------------------------------
void myReshape (int w, int h)
{
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h);
} // END:  myReshape()


// -------------------------------------------------------------------------------
// myMouse()
//   Registered with OpenGL, set in main()
//-------------------------------------------------------------------------------
void myMouse(int button, int state, int x, int y)
{
  switch (button) {

    // START THE SIMULATION
    case GLUT_LEFT_BUTTON:
```

```
        if (state == GLUT_DOWN)
          glutIdleFunc( computeMyAgentsLocation ); // OPEN-GL RUN FOREVER **************
        break;

      // RESET RANDOM NUMBER GENERATOR
      case GLUT_RIGHT_BUTTON:
        if (state == GLUT_DOWN)
          srand(time(0));
          glutIdleFunc(NULL);
        break;
      default:
        break;
    }
} // END:  myMouse()




// ---------------------------------------------------------------------------------
// main()
//----------------------------------------------------------------------------------
int main(int argc, char** argv)
{
int dummy;
    cout << "-------------------------------------------------------------"                  << endl;
    cout << "Supercomputing Challenge Virus Propagation Computer Program"                   << endl;
    cout << "-------------------------------------------------------------"                  << endl;
    cout << "Please Enter directory of:  myAgentFile.txt and myVirusFile.txt"               << endl;
    cout << "   please try c:work with back slashes for Windows"                            << endl;
    cin  >> MY_DIRECTORY_NAME;
    cout << "Directory to find files is: " << MY_DIRECTORY_NAME << "   "                   << endl;
    cout << "Please enter total number of Simulation Runs"                                  << endl;
    cin  >> runsTOTAL;
    cout << "Number of Simulations Runs is: " << runsTOTAL                                  << endl;
    cout << "Select any key, then click Return key to continue"                            << endl;
    cin  >> dummy ;

    // SETUP OpenGL ----------------------------------------
    glutInit(&argc, argv);                                  //(1/5) initializes gl util toolkit
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);            //(2/5)
    glutInitWindowPosition (100,100);                       //(3/5) set window position
    glutInitWindowSize     (WIDTH, HEIGHT);                 //(4/5) set window size in pixels
    glutCreateWindow       (argv[0]);                       //(5/5) create above window, disp w/glutMainLoop()
      initSimulationRun();                                  // CALL MY INIT FUNCTION
    glutDisplayFunc(myDrawOpenGL_Display);                   // REGISTER MY FUNCTION
    glutReshapeFunc(myReshape);                             // REGISTER MY FUNCTION
    glutMouseFunc  (myMouse  );                             // REGISTER MY FUNCTION
    glutMainLoop();                                         // must be last line of code - starts simulation.

    return 0;
}// END:  main
// ---------------------------------------------------------------------------------------------------------------------------------
```

7.2 References and Citations

- [A] Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on strings. Cambridge University Press, Cambridge (2007)
- [B] 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.
- [C] Newman M. Spread of epidemic disease on networks. Phys Rev E Stat Nonlin Soft Matter Phys. 2002;66:016128.
- [D] 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.
- [E] 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.
- [F] 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.
- The Following were not cited within this report.
    - 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)
    - Marta-n 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.

- 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.
- 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.
- 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.

8 Most Significant Achievement

The most significant achievement in this project was the ability to view the Agents' movements within different population densities while showing the resulting daily infections and summary infections plots in real time. The curves are simultaneously viewed in each simulation run while the movements of the Agents are displayed. The example screen capture is seen in Figure 4.

9 Acknowledgments

Mr. Jim Mims is acknowledged for teaching C++ my freshman year at the Albuquerque Academy which many software concepts he taught were incorporated in the RBA model. Ms. Kuhne is acknowledged for advice in algorithm development and OpenGL programming. The Supercomputing Judges are acknowledged for comments provided during the Project Evaluation period which were incorporated within this report.