CovidSIMVL Technical Report #002.

A HANDBOOK FOR CovidSIMVL 200 #1507

(“CovidSIM Multiverse Viral Load”)

Ernie Chang, MD PhD Victoria BC

**INTRODUCTION**

**This is the 4th of a series of CovidSIM agent based models written in Javascript for execution on browsers. It is open source and free for anyone to use or adapt, under the general terms of GNU Open Source licensing.**

**All CovidSIM models use agents to represent persons, and the key factors are person size and mobility (which we refer to as “mingle” in this document). The third important factor is density (the population in relation to size for a fixed universe).**

**FACTORS AFFECTING THE COURSE OF AN EPIDEMIC**

**Size is also called Hazard Radius – the larger this is, the more likely that two agents, moving within a confined space (an arena) will end up touching or overlapping, in which situation infection may occur, if one is infective and the other is susceptible. Clearly the smaller the Hazard Radius of an agent, the less likely it is to contact another.**

**The Hazard Radius is therefore the resultant of any number of factors in the context of a contagion-based epidemic – for example, age reduces susceptibility so it will tend to decrease the Hazard Radius. Comorbid conditions such as obesity and diabetes will tend to increase the likelihood of being infected, so will be a positive factor for size. The viral load of a person will tend to create more infectivity, equivalent to an increase in size. The mitigation policy of mask wearing can decrease the susceptibility of an agent, so it is a negative size factor.**

**The algorithms that combine the multiplicity of factors are discussed in a following section.**

**Shelter in place, and isolation, can serve both to limit the mobility of an agent, and can be represented either or both by an effect on size, as well as an effect on movement.**

**Mobility or “mingle factor” is used to propose a move for an agent in a time cycle. The CovidSIM algorithm generates a Pareto-distribution for a move, and then modifies that by the various mingle factors. For example, a person who works as a server in a restaurant or bar moves around more than a client at a dinner table. Thus, the mingle factor for the server would be higher. The modelling of a factor like poor air circulation in an enclosed space can be modelled by giving every person in that space an addition to their individual mingle factors.**

**TEMPORAL DYNAMICS AND VIRAL LOAD**

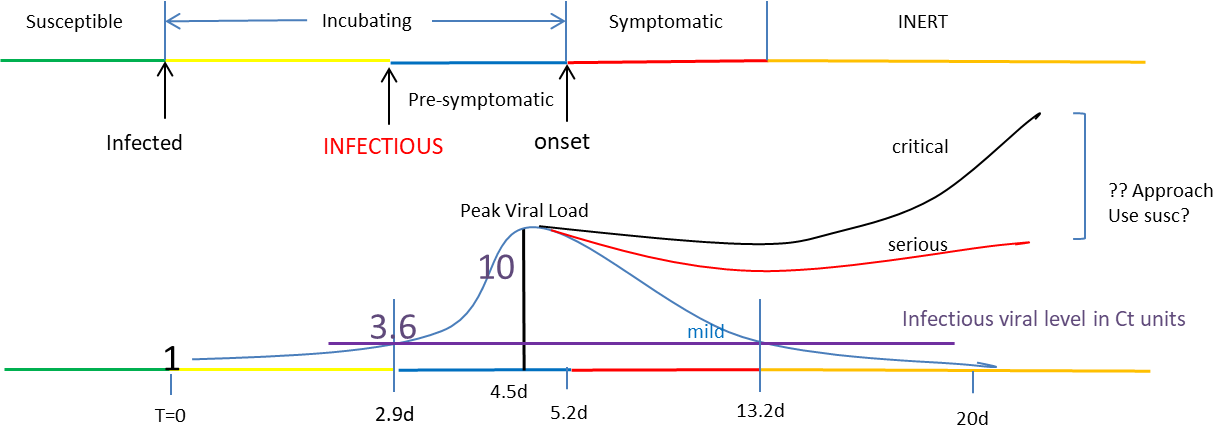
The paper by Xi,He et al. *Nature Medicine* **26,** 672-675(2020) is the basis of a temporal and viral growth model for Covid which is used in CovidSIMVL).

Susceptible

Incubating

Symptomatic

INERT



The time-lines can be changed, and they are applied stochastically for each person, to a +/- randomization of 5%. However, decisions had to be made as to whether the viral load took precedence over the temporal model. The arbitrary decision is to adhere to the time posts, but to also use viral loads as guides to viral growth exponents both from infection to peak, and from peak to undetectable, with 13.2days as loss of infectivity. This approach produced compound growth rate of 1.069 per 0.1days to peak, and a value of 0.865 per day from peak to 20days.

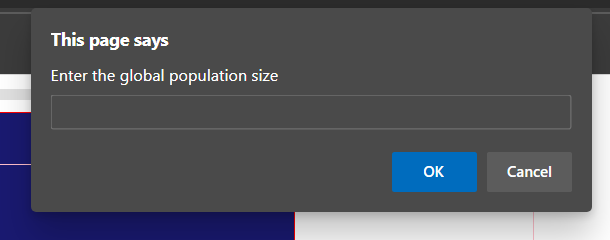
The approach when two agents overlap is to increase the viral load of the agent with a smaller viral load, provided the transfer makes sense in a physiologic way. For example, someone past the infective state (color “orange”) is assumed not to increase their viral load even if they overlap with a heavily infective person.

The viral states of a person progress from “green” – susceptible to “yellow” infected to “blue” – infective to “red” – symptomatic and infective to “orange” – no longer infective and presumed inert (maybe immune or deceased).

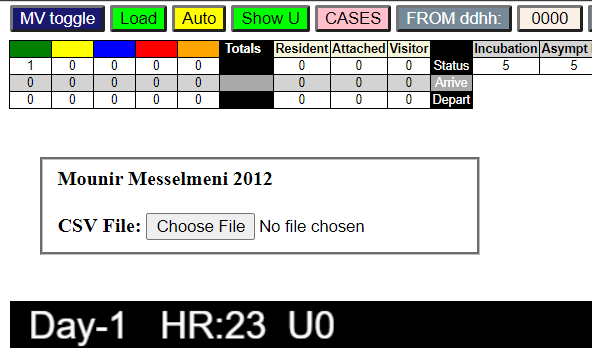
**OPERATING CovidSIMVL**

CovidSIMVL supports nine interacting universes, in the sense that agents can move at specified times from one universe to another, adopting a particular role (one of “resident”, “visitor”, or “attached” – like staff or office worker) with a specific mingle factor.

To start, a total population size covering all the universes is entered, so the system can create universes and skeletal agents.

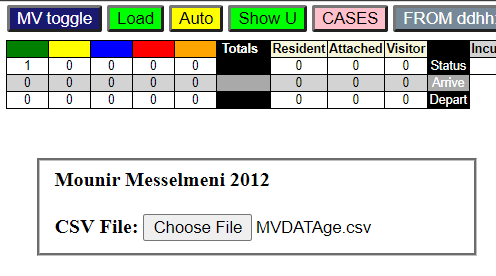


There are two files, at present design…one csv file designates the movement of persons at particular times from one universe to another. Persons are given unique identity numbers (eg starting from 0 to N), and times are in DDHH or just HH format. The Age-Group for that person is also an entry for this first CSV file.

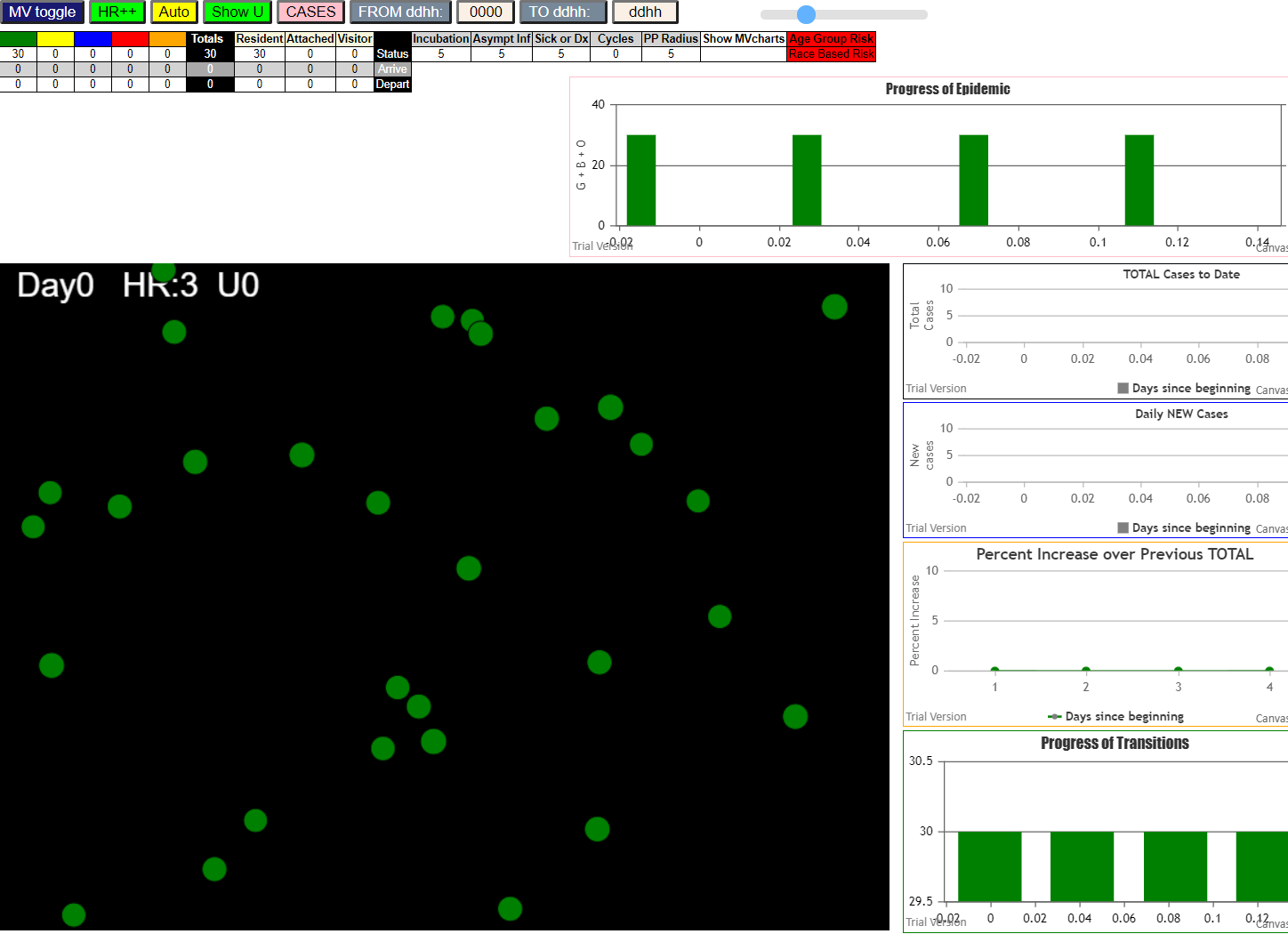


The next csv file is a modifier of selected agents of the population, and assigns a viral status (“blue”), a Time-since-infection to that person, and a viral load from 0 to 10, as well as a single susceptibility factor which is a combination of various personal attributes known to affect the risks of persons with respect to Covidl. These factors include obesity, smoking, chronic diseases, race(?) etc.

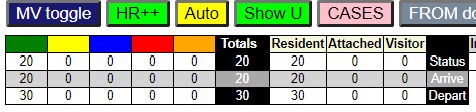
After the first CSV file (“MVDATAge.csv”) has been loaded, the user must select the pink “CASES” button to call up the file open widget that now shows that MVDATAge.csv has been opened, and click on Choose File for the next.



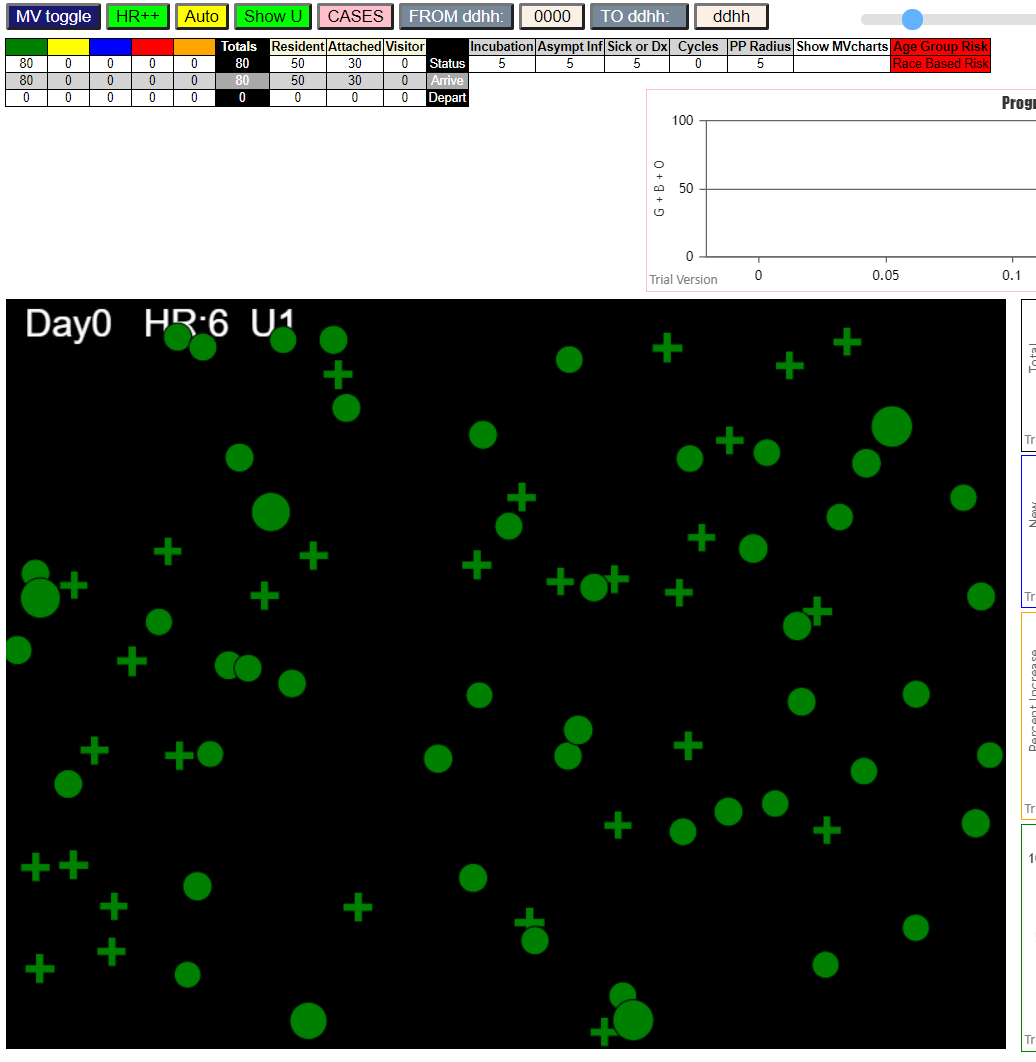
Next, the GREEN “LOAD” button must be clicked, and this will transform to show HR++. One can use this to advance the hours, or use the yellow “AUTO” button to automatically advance the clock. The clock can be stopped by clicking on the AUTO button again. The screen shows time Day and Hour, and the agents in their viral states in the universe shown here as “U0”.



Note here that not only do the agents show up, but the graphs appear, and with each click they will advance accordingly. The smallish table under the menu bar shows the viral states and counts for the universe, as well as the numbers of agents of various classes arriving and departing at the hour, if there is such traffic. For example, we see what happens in hour 6, at which time 30 persons leave for U1 (Long Term Care), where U0 is a universe called “Homes”.

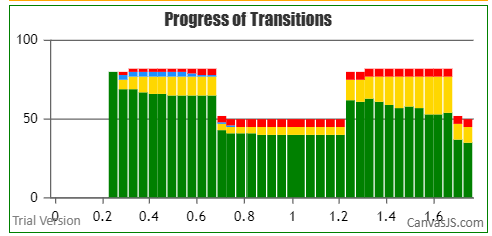


Here we see that Universe U0 has 20 persons, with 30 having left and 20 new arriving. The green “Show U” button will take the viewer to the universe U1 for the same time slot, showing a total population of 80 with 30 new arrivals which are “attached” and 50 “residents” arriving from system initialization.



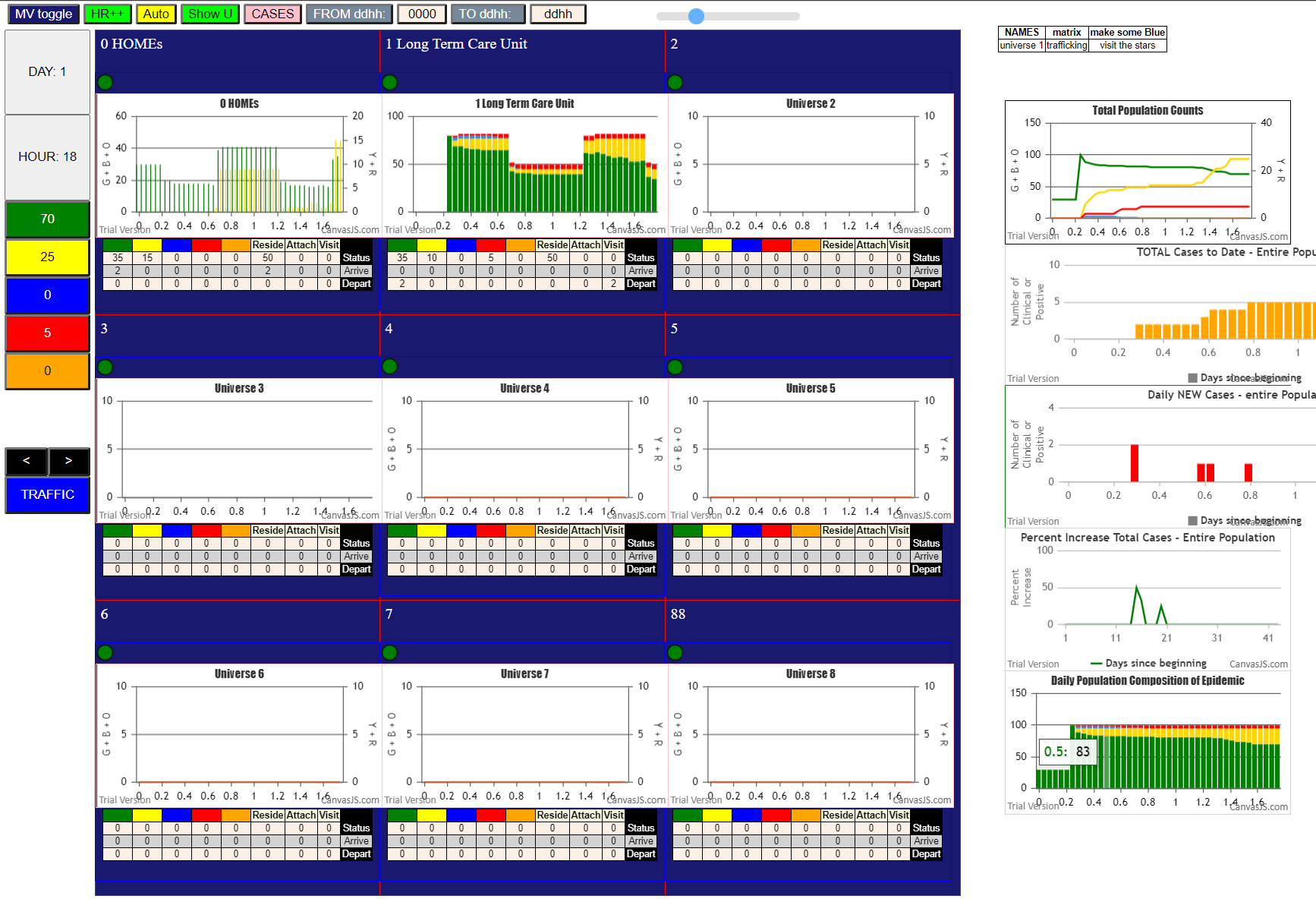
A look at the next hour shows that with viral growth, with the passage of time for the Cases modifications fro the second CSV file, we now have 2 reds (symptomatic), 6 yellows (infected and incubating) three blues (infective), and a number of crosses representing “Attached” roles for staff, while the spheres are permanent residents, and squares will be “visitors”.

By Day 1, Hr 18, we have a situation in the stacked column “Progress of Transitions” which shows



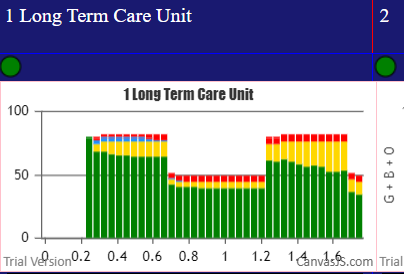
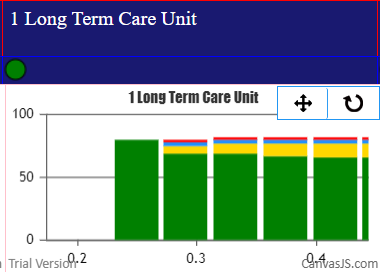
For each hour, the total population in Universe 1 and the numbers of different viral states for the persons in the Long Term Care universe. When the 30 staff leave at 1700, the residents are left on their own (in this case), , and we see in the second day that the new staff of 30 arrivals added significantly to the numbers of incubating persons, which grow in the day through mingling (movement), which takes place as a parameter of five cycles per hour. Then they leave.

The next screen shows the Universes together in one screen, with statistics and a graph for each Universe. Here, we only have the two. The Multiverse iview is invoked by the dark blue “MV toggle” button top left (dblclk).

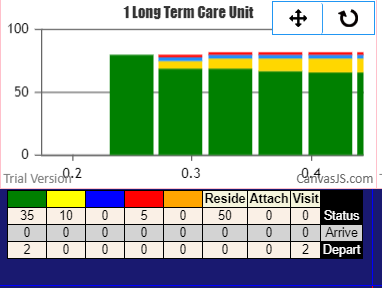


**We see here the side tallies on the left for the entire population, regardless of where each person is, at the present Day-Hour, and the graphs on the right are also with reference to the whole population.**

**For the first two universes (Homes=0 and Long Term Care =1) we have a bar graph for Homes, and a stacked column graph for the Long Term Care. The first graph on the right is a line graph equivalent to SEIR, the next is for Total Cases (symptomatic) in the population, the next for Daily New Cases, the next for % increase day over day for Total Cases, and the last the Stacked Column graph for the entire population.**

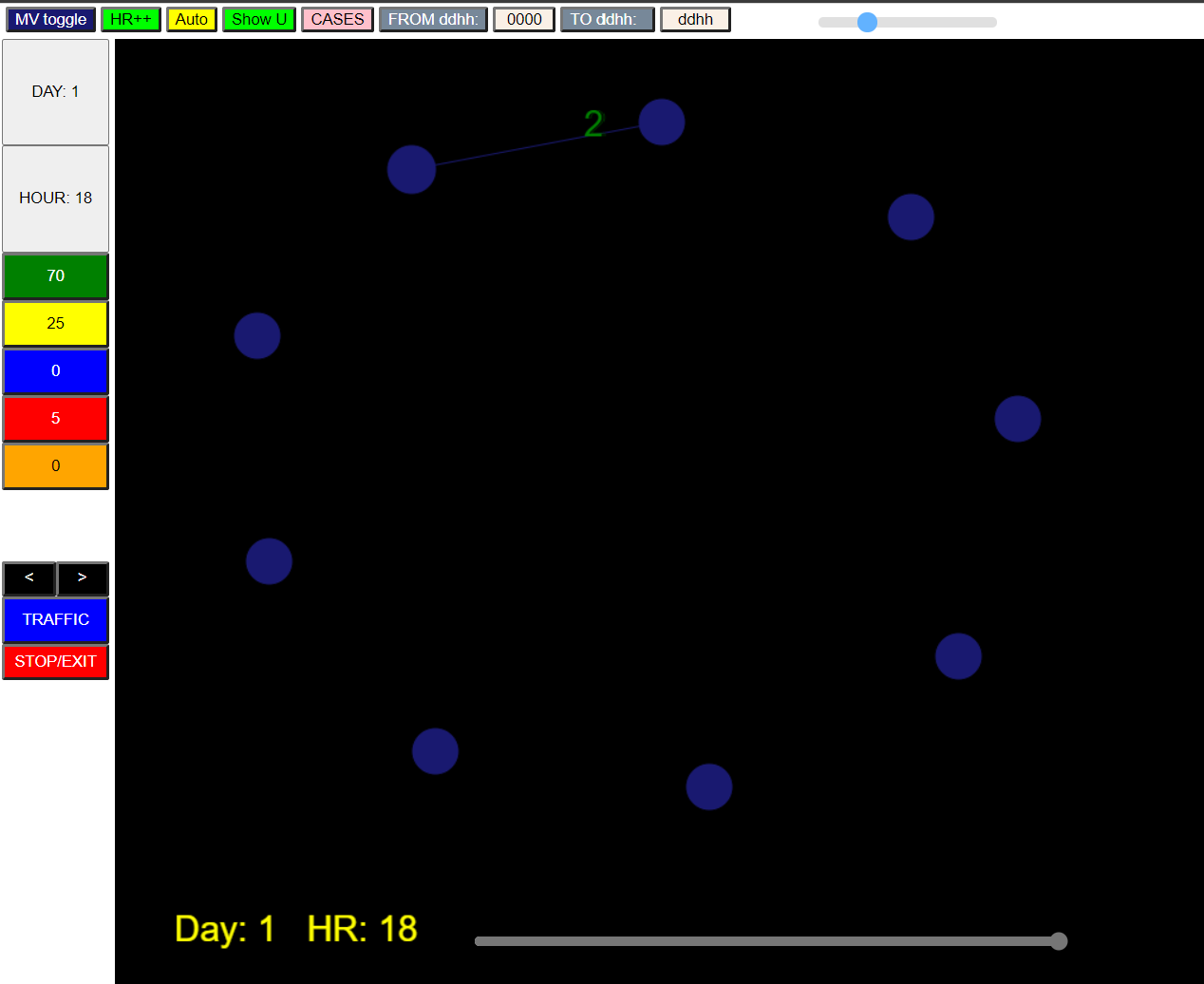
**Each graph can be zoomed. For example, the Long Term Care stacked column can be zoomed to show details of the first few hours on the left, as seen below, by clicking and dragging the mouse over the area.**

**As shown in the single universe view, each Universe has a population and activity tally for the current DayHour.**

****

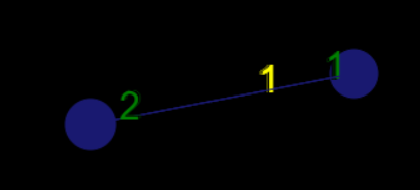
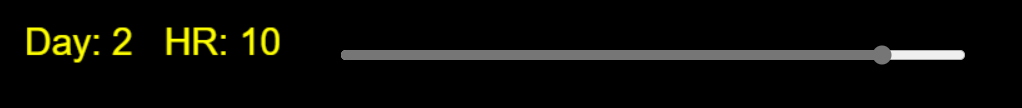
**Thus, Universe 1, at Day1 HR18, has 50 Residents, 35Green, 10Yellow and 5Reds, and has 2Visitors who have departed, and their status was Green.**

**There is another view of the dynamic flow of traffic between the universes, and the intensity and type of the traffic. If one clicks on the “TRAFFIC” blue button, the following screen replaces the previous view:**

****

**The green number “2” is shown here, but in the actual view, the number travels from U1 to U0, along an edge which is instantiated because there is traffic. If there was traffic from U0 to U1, the number would be shown in their appropriate viral state colors moving in the opposite direction.**

**The long slider at the bottom can roll back the clock, and then the black “<” and “>” buttons can change the viewing time by an hour and show the animation.**

**At Day 2 HR10, 2Greens are moving from U1 to U0 and 1Y+1G are just arriving at U1 from U0.**

**Clicking on the red “STOP” button stops the animation. This is useful if the traffic pattern is complex, and playing the clock back and forth allows the viewer to focus on the details of the traffic movements.**

**There will be highlighting of the node which has the highest risk (should this be largest number of infectives arriving, or largest number of infectives present, or the highest ratio of susceptibles to infectives). It would probably be good to show the names of the Universes, and their internal population and composition through right click or mouse-over, in addition to the movement of persons.**

**RELATIONSHIP TO EQUATION BASED COVID MODELS**

Classic epidemiology models of contagions use historical data (from past days) to estimate the parameters of a set of equations describing the dynamics of the epidemic, and then use the parameters to predict the future behaviour of the infection in the population.

The agent based model simulate the forward behaviour of a population using the characteristics of individual persons and the dynamics and parameters of the contagion.

In a sense, validation of each is difficult. In the first, the assumption that the past predicts the future is only really true in hindsight, and so every day brings a new challenge to validation or parameter re-evaluation. In the agent based model, the simulation is generally intended to be illustrative and easily amenable to parameters with physical analogues to demonstrate effects.

However, they should in many senses be equivalent expressions of the same large scale phenomena, of an infection affecting a population through transmission dynamics.

The key to their equivalence may be through the coefficient R0, which is defined as the number of susceptibles infected by an infective person during the time they are infectious.

In the CovidSIMVL model, R0 is easily calculated by tracking the number of susceptibles infected by all orange agents (inert), and dividing the sum by the number of orange agents to get the average R0.

Furthermore, to yield a specific value of R0, the CovidSIMVL can be used as a platform for varying different parameters to yield the same R0, and this should provide a means of assessing the effect of different mitigation strategies in different populations.

The use of multiple universes to represent the interactions of different populations affecting one another through common spaces or visitors can be quantified in CovidSIMVL, and this can therefore be a tool for defining the variations in transmission risk with different densities or duration of stay in a common space, in a rigorous and methodical manner.

**CONCLUDING REMAKRS**

The button on the top right marked “Calc R0” will produce the R0 for the entire population, given its traffic flow. To know which Universes are at most risk from what other Universes, if such correlations can be made, would be of great interest, if it can be demonstrated. For instance, the hypothesis that staff going to bars have more influence for bringing Covid into Long Term Care than visitors who go to bars can be tested.

Schools, and the effect that children have when they mingle, and go back to multigenerational homes, can be modeled in CovidSIMVL, and mitigating effects for the risks involved can be introduced and their effects seen, so that questions about school opening affects long-term care, and the extent of this link, can be assessed.

Currently, work is starting on creating the environment in R to implement goal-directed parameter optimization using CovidSIMVL as the simulation tool.

Many unknown factors and relationships are present in implementing a simulation of the SARS-2 epidemic. For example, what is the quantitative mitigating factor for masks in terms of transmission efficiency? Does it vary linearly with viral load of the infective, or an inverse square relationship?

How do observed incidence of patient characteristics convert into odds ratios, and how are they combined?

What is the extent of transmission for which we use degree of overlap as a proxy for duration of contact, and how do we combine this with the viral gradient between the two?

How does viral load affect infectivity and therefore size in the model?

There are many factors that we are starting to learn about SARS-2 but they all relate in the final analysis to how infective a person is, and how sick they get. Whether there is one or twenty factors, they combine in the human to some single variable result, and this is what CovidSIMVL attempts to simulate – a person, a set of paramaters combining to give an infective result, and operating within the dynamics of population movement and interaction.

**DESIGN OF CSV SPECIFICATION FILES**

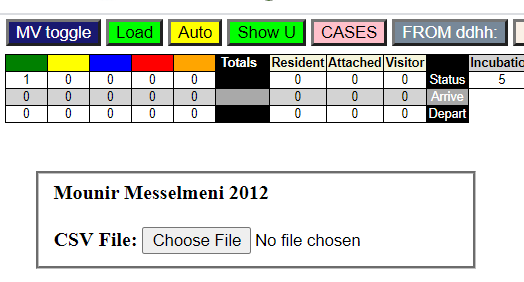
The first CSV file, which is named in the prototype as MVDATAge.csv, is a file that describes the movement of agents between universes, and the roles they have in them.

The second CSV file, which is named VL1.csv, identifies a subset of the agents in the MVDATAge.csv file as Covid cases, with viral loads, time since infection (since we are using the Temporal Dynamics and Viral Load model of Xi, He), and so on.

The program starts by requiring the use of the system to enter the population of agents corresponding to the schedule. If they do not match, there may be errors. The MVDATAge.csv persons are the for whom moves are proposed each cycle, and therefore viral transfer, viral growth, resizing are functions that are called for that population.

If the total population entered in the initial prompt is smaller than that of MVDATAge.csv, we will probably have execution errors from “undefined” array errors. If the total population is larger, the data tallies may be wrong, as the excess do not go to any universe but exist in pre-initialized state but are counted.

The program could check the match between them, but this requires error handling by the user. The program could change the CSV files to have a single multi-level parser, but that is a step for the future. As it stands, the system is bound to the FOUR steps. This is what appears after step 1 has been done.

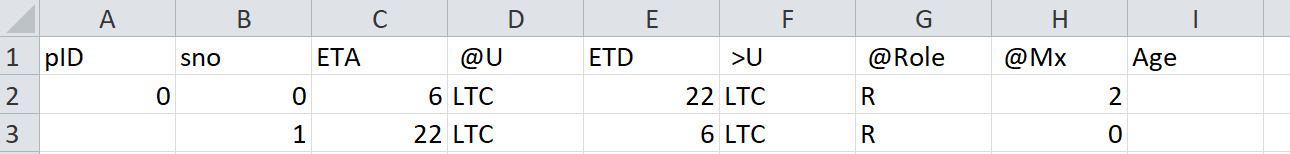


1. Input the correct population number at the startup prompt
2. Input the MVDATAge.csv file (or any other named file with data in the correct correspondence.
3. No positive correct loading is given. When control comes back to the user, click the “CASES” button and enter the VL1.csv file containing the initial cases.
4. PRESS THE LOAD BUTTON (in green). Sorry, but this is a crucial initialization step.

If the VL1.csv file is not entered, there are no infective agents, and the system will not simulate any epidemic.

MVDATAge.csv

Of course, the cvs file is created by an Excel table, saved in .csv format. The table entry looks like this:



The pID field is the unique identifier for an agent (person, patient, class of persons, etc). The simplest approach is to number them from 0 to N.

This file describes the arrival and departure times for agents with respect to different universes (the maximum number is 9, for visualization purposes). The setting, M.UCt is a program variable, not an input parameter.

Think of the model of a master schedule for a train system. It knows about TIME (Days, Hours), Passengers, Arrivals, Departures. For each Passenger, the train reservation and scheduling system has a Journey profile with a number of stops, each with an ETA (expected to arrive) and ETD.

Thus, “sno” is the Stop Number for this passenger (0). Here we show two Stops (which may be thought of as ticket stubs). The ETA for pID=0 is at 0600hrs or 6am, and “@U” indicates the place that the person is at, or will be at, at 0600.

For initialization, this first set of tickets have no corresponding departure tickets which show their arrival on line 2 above. They land from zen-space at the ETA time, into the @U universe. Here we see that it is called “LTC” (for Long-Term-Care). The “ETD” field indicates the time that this person will be departing from @U, and the field “>U” shows the Universe to which that person will be going (which in this case, is also “LTC”, which means that person=0 will be staying there.

The “role” of this person for this duration is “R” which is “Resident” which means they will stay there permanently (long term care, hospitals, submarines, cruise ships, prisons; at a different level, these could be persons who do not travel by car, train, foot, air, or ship from a city, municipality, region, county, state).

The role “A” means “Attached” and these persons would be connected on a partial basis to the universe, such as staff who come for 8-hour shifts and leave. While there, their activity level (see below) is higher than for “residents”.

The role “V” stands for “Visitors”. Persons who visit patients in hospitals, who are outpatients in clinics, who are in transit in airports, students in schools and universities, are visitors. In general, they have less mobility and mingling or exposure to others than “Attached”.

“@Mx” is the mingle factor from 1 to 10, the higher the more active. This notion of activity is expressed the following way: at each cycle, for each person, a move from current location to another location is proposed, first stochastically according to random selection from a table of values which represents a Pareto distribution (more smaller numbers than larger – I suppose in theory the number should be randomly distributed in the table rather than clustered, as they are at present). Then the mingle factor is applied, in such a way that with the maximum of 10, the distance moved is still going to be within the bounds of the universe (if the universe had no bounds then everyone would disperse further and further away).

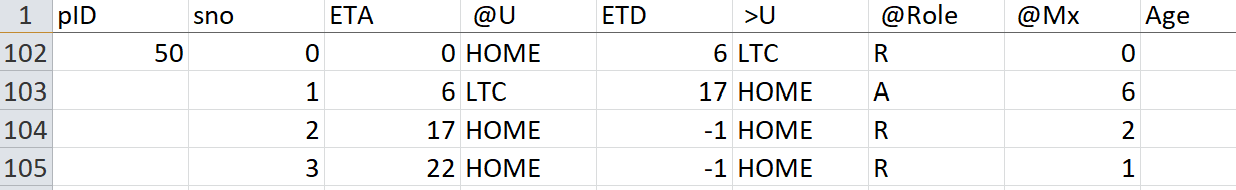
The calculation of the proposed move may also be subject to a mingle factor for the universe itself, but at present this is not a parameter but would have to be introduced through a modification of the calculation, though the field M.minglf is part of the Universe data definition already.

Essentially, the proposed move generates a pseudo-random walk (biased) for an agent, and the mingle factor increases the area covered by the random walk. Clearly a mingle factor of ZERO should cause the agent to remain in place, in which case the random walk covers an area of zero, and infection by contact cannot occur.

CovidSIMVL has the capability of defining a number of epicenters within a Universe, to which agents are stochastically drawn (but not magnetically)….thus, this feature could be used to represent persons in restaurants, coffee shops, bars, and receptions, in which most guests stay around a space, but the attendant staff mingle much more – same for students and Teaching Assistants.

The last field “Age” is for an age-group of the person. A function ageRisk(G,g) exists within the code for looking up an age risk table based on 10-year age groups, the population age distribution of BC for 2019, and the incidence distribution of age-group among Covid cases in Spain and Italy, with the risks ranging from 0.33 for age group 0 to 2.33 for age group 90+. Using the numbers is not clearly defined, so at present this calculation has been left out in the Temporal Dynamics – Viral Load model used in CovidSIMVL.

Here is another example:



Person=50 has a role of “R” in @U=HOME from ETA=00 (midnight) to 0600 with a mingle factor of zero (sleeping), and leaves at that time for LTC. At LTC (line 103), arrival is at 0600 (instantaneous transport) and there, assumes a role of “A” (could stand for Attendant), with a mingle factor of 6.

At 1700 hours, P(50) arrives at HOME and there is not departure (-1) in this ticket, but now assumes a role of “R” – resident with a mingle factor of 2. The next ticket (sno=3) at 2200 again without departing, changes the mingle factor to 1 (getting ready for bed), and then the ticket recycles to midnight (line 102) with departure at 0600 for LTC.

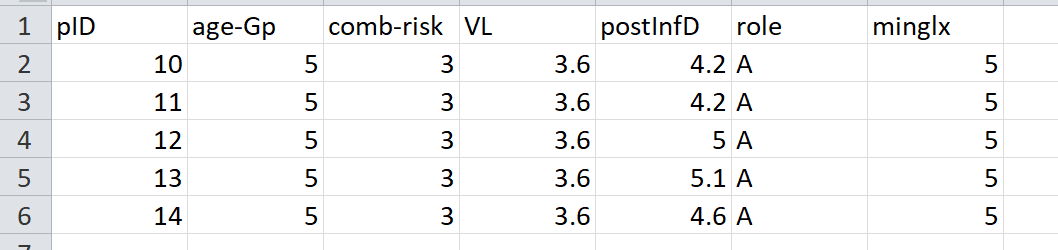
HANDLING TIME

ETA and ETD can be in the form of “DDHH” – what has been shown is just the HH. The calculation is to take this field, and use floor(ETA/100) as DD with remainder (ETA/100) as HH.

The routine called “conductor()” matches the next clock time with the times on the ticket stubs (sno) for the current DD, so that a specific schedule with Thursday being different can be constructed. Note that the default is for the schedule for the next day to use the current day if there is no explicit entry for the next day. This is how the one-day schedule becomes the default recurrent schedule if no explicit tickets for a day is found. I am not sure at this point how robust the testing is for this schedule by exception approach.

**VL1.csv File**

This file is a “Case” file and supersedes corresponding entries in the MVDATAge.csv data.



This example modifies the data for 5 persons, from P=10 to P=14. The first field, “pID”, is the unique identifier. The second is a duplicate field for age-group (and can be removed in time). The third is a combined-risk factor, to cover innate risk factors such as obesity, diabetes, asthma, vaping, hypertension, serious heart disease, chronic kidney disease, perhaps race. Rather than entering these in the simulation program, and requiring it to combine these risk factors, I have made the decision that this should be done better externally by the model creator, and combined into a single factor which affects the size (or viral growth? Or mobility?) of the person.

The “VL” field stands for Viral Load according to the Xi,He paper, using the CT times as score. The postInfD is important – this is the number of days after infection. For example, a newly infected person in the simulation could be given a number here of 1 but this would require the simulation to run for 1.9 days before the person becomes infective.

A creative example of this specification is to give the agent a mingle of 0, and a postInfD of 12.2d with a high viral load, say of 6. This then places the agent with 1 day left for infectivity, and immobility – exactly what a door knob would be.

These CSV files should be considered evolutionary, and their evolvement is inevitable.

**STRUCTURE OF CovidSIMVL #1508 2020.08.14**

The program is written in Javascript in one continuous module, with a few uses of external routines. The most prominent of these are:

Canvasjs invoked line 1329 note: we did not use chart.js but canvas.js

<script src="https://canvasjs.com/assets/script/canvasjs.min.js"></script>

File read and processing routines code adapted from

<https://blog.mounirmesselmeni.de/2012/11/20/reading-csv-file-with-javascript-and-html5-file-api/>

The canvasjs routines complement the html5 canvas intrinsics, and all the graphs are created with canvasjs, which are dynamic, real-time, and provide zoom and pan.

The program has three major functions which are somewhat interwoven, rather than discrete. This is due to my approach to agent-based simulation, in which interactions of agents and the immediate results at various stages can be visualized and modified by the user. Thus, instead of the structure of:

* Parameterize
* Action
* Output and display

The structure of CovidSIMVL has the components:

* Initialization and parameter setting
* Cycle simulation activities somewhat interspersed
  + Create actions
  + Create outputs and counts
  + Display outputs

The main structures are as follows:

Line 1 - 662 html statements for position and properties of DOM elements

L 670 – 1347 CSS statements for various menus, displays including the data tables for each universe in MV

Note that there are three distinct screens:

* + - Single Universe at a time screen plus output graphs
    - Multiverse showing up to 9 active universes and their data as well as overall data
    - Traffic screen showing movement of agents between universes at DDHH times

These displays and their fields are all specified in the CSS section

L1349 – 5129 The Javascript section, which includes:

* + - Global variables that are scattered across the code near the functions using them
    - Global data structures (objects) for:
      * The Multiverse (the entirety of the simulation space) (“M.”)
      * Each Universe (“U.”)
      * Persons (“P.”)
      * Tickets (per person - “T.”)
      * Day Schedule (“D.”)
      * Right-click on the web page of the program will give INSPECT in devtools
        + “WATCH” in devtools will show contents of these data structures
    - Routines for simulation based on”
      * Initialization
      * Advance Time
      * Viral Growth from last cycle, check if viral state changed for all agents
      * Apply new viral load to size of agent
      * For each universe perform arrivals and departures
      * Within each universe propose move for each agent
      * Make move for N cycles per hour
      * After all moves, check overlap for any pair of agents
      * If overlap, check qualifications for viral transfer (infection)

We will provide two sets of descriptions for this code: by line sections, serially, and by functional flow.

LINE SECTION DESCRIPTION OF JAVASCRIPT CODE

L1351-1356 Routine to declare slider (top menu) In Fixed and Multiverse screens and its relationship to the

ClockTimer which determines the speed of the AUTO progression from hour to hour of the clock

L1359-1361 some global variables – VIEW for whether Fixed Universe or Multiverse is current

MODE – for whether the simulation progresses via the HR++ button, or by automation

graphFlag – whether the Traffic display is current

L1368=1375 function drawc() – the base routine to draw a circle of a size and color at coordinates x,y

L1378-1385 function CreateCanvases() – prototype function to create instances of objects as specified

L1387-1392 more global variables

L1393-1407 calculations for creating the nine universe windows in the array cn[ ]

L1412-1583 the declarations for the nine Universe panes (“canpi”) positions and sizes

L1589-1702 routines for mouse click, move, right click, etc over the Universe panes – not used at present

L1709-1789 creating the canvases for the Universes in which graphs from graphsjs will be drawn

L1798-1840 creating the display names of the Universes (input prompt

L1814-1827 function parseMVnames – reads and parses prompt for all names of the universes

Here the universes are numbered 1 to 9, but elsewhere they may be 0 to 8.

THERE MAY BE INCONSISTENCIES AND ERRORS if this is a significant conflict

L1830-1840 function showMVname( ) – displays names of the panes read from prompt onto DOM elements

L1842-1859 function rBlues( ) – old routine for adding BLUE agents (infective) – replaced by better VL1.csv

The button for this data entry has been removed from the CSS statements

L1865-1898 Old routine for placing Universe names into a matrix for traffic…not in current use

Traffic graph and individual Universe arrivals and departures take its place

If this matrix is to be used in the future consider changing index to 0 to 8

L1904-1904 auto-function to hide the MV on initialization – through calling function hideMV( )

L1908-1975 functions showMV( ) and hideMV( ) – which shows the Multiverse Screen or the Fixed U screen

L1986-2013 IMPORTATN GLOBAL VARIABLES for the simulation – objects, time, working universe, etc

L2013 the line that construct the Multiverse characteristics (L2035)

L2019 array tViral – table of values for days since infection from Xi,He paper

L2023-2032 viral temporal and growth parameters inferred from Xi,He paper

L2035 the definition of the object M characterized by this function ConstructMVC( )

L2076-2112 the initial values are assigned here to the object M – some of these may not yet be used

L2119-2211 the definition, initialization and creation of the 9 universes from 0 to 8 (NBNBNB \*\*\*)

L2214-2222 this prototype is used for declaring arrays where counts of viral infected person types are tallied

L2226-2366 function initNet(Q,gen) creates datastructures in the Universe Q, for the generation (total hours

since start of simulation, for total departures each agent viral class, as well as for the arrival sand

departures relating Universe Q to another universe

L2241-2294 function CreatePerson( ) is the template for creating population of agents – the final three

touchCt, susCt, and failedCt are to record, when an agent is infective, the overlaps (contacts)

with other agents, the susceptible agents (green) that was infected by this agent, and the

susceptibles that were not high enough in overlap and viral gradient to cause infection

L2353-2357 function stochast(b, factor) – randomizes **b** by a **factor** to return (b-factor) +/- random(factor)\*b

L2369-2411 creates the data structures for Date (DDHH), ticket, stop, transfer, relating to persons pID and cID

Transfer is used just to hold data from MVDATAge.csv as each line is read in from the blob object

Transfer is created in L2404

Ticket[ ] is create in L2369-2376 (auto-call), one entry per agent

L2420-2438 function parseL(lineStr) is used to parse lines from the MVDATAge.csv file

It is called from function processLines( ) line L2505

The function eval( ) converts strings to integers

L2440-2502 function setupTicket( ) creates Tx according to the template CreateStop L2383, and

creates a stop for the tickets of the person T[pID]..S[sno] L2463

It also creates an entry in the master DATE schedule D[d].DD and the hourly entries

H from template CreateH( ) L2411 and L2475 the actual creation and assignments

Thus D[d}.H[h] for the Day and Hour has an entry which describes the:

cID person

cDir “A” for arrival, “D” for depart

cS stop number

u FROM or TO universe

HH hour

Note that we use PUSH to put new entries into the H[ ] array for efficiency

L2505-2526 function processLine( ) invoked from the readfile routines L4722-4761

L252902531 function caseload( ) – set flag when Cases button is pressed to read VL1.csv files

L2534-2557 function parseC(lineStr) – this processes the lines from blob from VL1.csv file

Updates the specific pIDs in the file with viral information

L2560-2564 function resizeRisk(ID) – the combined risk is expressed as a susceptibility number, and

this function takes the cube root (since a sphere is volume based) and changes the base size

of an agent by this factor, and then uses it for current size by +/- 5% of the new basesize

L2569-2634 function changeState(ID) and newState( ) changes the viral state of an agent according to

time-based breakpoints from the Xi,He model. changeState( ) checks what the new state should be (expressed by coior). newState( ) then reduces the count of current states, increases count of new states for the Universe in which the agent ID is currently present

L2640-2783 this code block relates to variables used for movement of agents and their imaging

Movements are essential for simulating the progress of the contagion

Imaging them are only realized in the drawX( ) routines but they are here together. Logically

L2662 this is the table which expresses the pseudo-Pareto distribution of proposed moves

L2668-2710 this code specifies five epicenters and their perimeters to which agents are drawn if they

land nearby in their moves – this is not being used currently in CovidSIMVL

L2713-2717 routine to write text on a canvas

L2752-2771 function drawAgent(x,y,g,clrFlag) draws the agent g (=pID) at x,y with clrFlag=clr as

Sphere for g.role = “resident”

Cross for g.role = “attached”

Square for g.role = “visitor”

L2773-2783 function drawU( ) draws all agents in the “vU” (viewing Universe), using the data from

U[vU].person. This array of person IDs needs to be copied or pop( ) will remove them

L2793-2848 These are routines which prompt the user after a button is pressed to enter some of the

parameters used in CovidSIMVL. Not all of these are in use in this version #1508.

L2830-2835 function showCycle( ) allows the modeleer to specify a number of moves of agents per hour

Moving once an hour is unrealistic, but moving every 10 minutes may be too much

L2838-2848 function showPradius( ) would change all the sizes of agents in M.PCt to the hazard radius

entered, except that it assigns the new value to P[pCt].size, and this field does not exist.

We have P.baseSize and P.currSize.

Javascript does not halt at errors like this, just blithely goes on!

L2851-2857 This allows automation of CovidSIMVL to start at date DDHH=0000 and end at DD’HH’

L2860-2861 function sizeP(G, g) is a function that returns the current size of person ID=g with structure G

L2865-2885 data structure for age-based risk in 10-year age groups, and the function ageRisk(G, g)

The risk numbers have been derived from a Fixed Universe CovidSIM, which combines the

2019 demographics for BC and the incidence of age-groups in Covid-19 patients in Italy and Spain

Function ageRisk(G,g) takes an age in years and returns the corresponding ageRisk

L2891-3156 This is a set of functions concerned with the movement of agents, their representation and their

consequences – contact and overlap after they all move, and the calculation of viral transfers for

overlapping agents where one is infectious and the other is susceptible to viral transfers

L2895-2928 function moveItmoveIt() repeats moves for X cycles per hour, for each person in the wU – the

Working Universe (as compared to the Visualized Universe vU). A move is proposed by

function proposeMove(G, g) L2933-2955. Here is the place for the mingle factors to apply to the

proposed moves, and for the epicenters to draw agents towards them, if they are used.

L2933-2955 function proposedMove(G, g) called by function moveItmoveIt( ), and not only do mingle factors

apply here but also the introduction of stochastic randomness, and increments delX and delY for animation display of the movements. The newX and newY coordinates thus generated also need to be tested for out-of-bounds of the simulation arena

L2960-2992 function testWall(G, g) modifies the proposed coordinates newX, newY and corresponding

Increments by out-of-counds testing by the crude method of reversing the delta values and

Setting the out-of-bound newX and newY to arbitrary positions of 10 pixels from the boundaries.

L2998-3022 function testOverlap tests the persons in the Universe Q by comparing the radius and centers for

persons (I, j) while ensuring that the comparison is triangular in the square matrix so we do not

also compare (j, i) which would be redundant and double the effect. If there is overlap, the

function VLtransfer L3088 is called

L3026-3058 function nearestEpicenter( ) for an agent at (x, y) finds the nearest defined epicenter through a

Tournament and applies it in function adjminglf L3062-3080, called from L2914, currently OFF.

This function is called in L2912, and returns the winner to the variable epic, currently unused.

L3062-3080 function adjminglf draws the newX and newY for an agent toward the nearest epicenter, also

adding some random element to the new position for X and Y as influenced by the epicenter.

L3088-3156 function VLtransfer(I, j) is at the core of the CovidSIMVL simulation. Its calculations will be

described in the next section.

L3165-3304 This section of code controls the main flow of the simulation, with the advancement of the

Master Clock, the examination of the HOUR schedules for the Universes, the control of the

transfers of agents between universes, and invoking both these transfers as well as the viral

actions due to agent movements within each universe.

L3165-3183 global variables for these routines

L3187-3195 function auto( ) invoked by button “AUTO” press to move between auto and manual (HR++)

mode, by means of MODE flag. Note that in manual mode, we clear the Interval Timer

“clockTimer”

L3197-3215 function load( ) invoked by the LOAD button at startup. The button then displays “HR++” for

manual mode, and hereafter every click calls the function “TUmesUp( )”. Note that the

defaults for universes, time are set here, AS WELL AS the functions startNet( ), which creates

the universe as nodes for the Traffic screen, and their positions, and the function initNet(U[i],0)

which initializes the data structures for the Traffic screen that supports the count of persons

in different virus classes (stages) at each Universe for each destination and each hour (“gen”).

Also crucially important, gen is initialized to -1, because it is in the main flow, incremented at the

beginning of each such sequence.

L3217-3236 function TimesUp( ) is invoked when the clock advances. Note that we have to call initNet every

clock hour because the data structures are newly created every new generation. Here we test

whether we have reached the time limit for the automation simulation set by the prompt in the

menu bar. If not, we advance time, and ask the “conductor( )” routine to work through all the

tickets for the new HH, then tabulate the results. Note that if we are in the Traffic screen

(“graphFlag” == YES, we put the traffic slider to the end, and call on function graphB( ) to

display the traffic for the nodes (and replay or stop them mid-flow).

L3240-3265 function advanceTime( ) will advance the 24hr clock to the next day at midnight, up to 99 days.

If there are schedules with defined specific DD days in the MVDATAges.csv file, they will be

identified here. If we are in manual mode or not yet finished automation, we advance the time in

the appropriate global variables cH, cD, cT (DDHH time) and show it on MV or the Fixed Universe

L3267-3273 screen. Note we invoke function drawLocal( ) L3264 which has a safety in checking if the MODE is

indeed LOCAL and the VIEW is “local” and not “MV”.

L3276-3294 function conductor( ) is the master animateur of CovidSIMVL, as it goes through all the tickets

for the new day and hour it has been given, and identifies for each Stop, the Working Universe

and the agent ID, and the action (ARRIVE or DEPART) from the indicated Universe. Then it

calls the functions “inject( )” or “expel( )” to execute the traffic action for that agent. Of course,

after that, it in turn asks each Universe to move the agents within them with moveItmoveIt( ).

L3296-3304 function finishHour( ) is called by conductor( ) to get the universes to move the agents according

to their properties and the Xi,He model of temporal dynamics

L3308-3342 function growVL( ) is called in function advanceTime( ) after the clock has advanced but before

conductor moves agents to and from universes. The notion is that in the previous hour, the

finishHour( ) routine has called on moveItmoveIt( ) to get the agents to viral transfer. It has an

hour to initiate growth which is accounted for in advanceTime( ) at the beginning of what is the

next hour. So the viral transfer and viral loads had one hour to grow, and this growth is tallied

at the start of the hour, before any traffic or internal movement occurs.

L3344-3367 function reSizeALL( )

At the end of growVL, the function changeState( ) L2566 is called. As the transitions in this

simulation engine uses days since infection, changeState( ) is called after time change and

after viral growth, so that the changeState( ) function can use both, even if it uses only time

at present.

The function reSizeALL( ) is called from growVL( ) after all the growth has been calculated,

and the new viral loads are used to change the P.currSize of the agents. NOTE that in order to

prevent sizes from ballooning, we have kept the newSize to a maximum of 3xbaseSize for each

agent.

L3369-3443 function injectXY(G, g, stop) is responsible for looking at the ticket stops for this DDHH and

taking the Arrival agents and inserting them into the destination Universe on the stop, for the

agent “g”. This requires an adjustment for the counts in the Universe for role numbers, and

for the viral class numbers, based on the viral class (or state) of the arriving agent.

L3445-3510 function expel(G, g, slot) is the companion function to inject( ), and takes care, from the

agent “g” given to it, the task of finding the information from the array “T” and then to

remove that person from the appropriate Universe, and adjust the counts, not just of the

role and viral class membership for the Universe, but also the Traffic flow counts for departures

from the Universe, which are in the Universe data definitions as U.dep and U.depT, where depT

is a total of departures in that DDHH and if zero, the edges do not have to be examined, while

U.dep points to traffic going along each edge, if any, to another Universe.

L3512-3520 function getToTime( ) is the routine that responds to the button press for “To DDHH” in the

long menu for automation. Note that if this is not set, automation will work anyway but has

no stopping point.

L3523-4715 The code in this section relates to tabulations and displays. While some designs separate the

gathering of data from their presentation, in this case we are using a WYSIWYG approach for

showing data as they are generated from simulation. Nevertheless, it would not be too hard

to see what is tabulation and what is display in this program.

L3523-3531 function showUstat(Q, q) displays the current number of viral classes (green, yellow, etc) in

the specified Universe q for the DOM IDs “xxCt” which is for the Single Universe screen.

L3533-3558 function sumUcount( ) takes care of the tallies of the total distribution of viral states across

all Universes (the Multiverse). It sets the counts to zero each time because this is the most

economical way to get the sums without having to update them with every state change,

although indeed that may be happening in the function changeState. Nevertheless, this is

an approach to derive the current state of the Multiverse by adding their components from

each Universe. The data structures M.logGreen[gen] will for each generation (clock hour)

track the counts for the entire population. These data are used in the Charts but of course

can be used for other purposes since they reflect the changing state of the population.

L3563-3578 function showU() is used by the button “SHOW U” to advance the Single Universe screen to

the next Universe, modulo the number of active Universes. Note that the function

changes the globals wU (working Universe) and vU (viewing Universe) to the next U, then calls

on functions drawU( ) L2770, and showUstat(U[vU, vU) L3520 to display the next (now current)

Universe. The functions cleanCharts( ) L3577 and upDateGraph(Q, vU) L3704 take care of the

dynamic Chart6 that shows the dynamic SEIR graph of the Single Universe as the simulation

proceeds.

L3580-3611 function cleanCharts( ) resets the data for the Single Universe Charts, of which Chart6 is the

visible dynamic SEIR graph that changes with each clock hour. The other Charts for the Single

Universe view are charts 1 to 4, and what this function does is to reset their data sets to those

of the new Universe invoked through showU( ) L3563.

L3613-3654 function tabulate( ) is in two parts, the second being a call to function Mtabulate( ) L3657.

tabulate( ) works on each Universe, and updates the log of viral states (agents) by generation,

and creates the Q.endGreen data structures, which are data pairs of the log and Day for each

viral state in each Universe. These data, again, are used for chart displays but valuable for

analytics should they be required. In addition to daily counts kept in the logX arrays, the data

is also calculated for new cases (today less yesterday) and for velocity (new cases/yesterday’s

cases).

Following these tabulations for every Universe, the functions sumUcount( ) L3530, Mtabulate( )

L3657, and upDateGraph( ) L3704 are called.

L3657-3703 function Mtabulate( ) is the corresponding set of data structures for tabulation of the population

of the entire Multiverse, whereas the tabulate( ) function does it for each Universe.

L3706-3782 function UpdateGraph( ) is a large routine mainly because it handles the views for both the

Single Universe screen as well as the Multiverse screen, through the device of turning the DOM

elements’ style.display to BLOCK or NONE (visible or invisible). In addition, the chartjs routines

are called for each screen as appropriate. The detailed tallies for the viral states as well as the

roles for each Universe, and the count of arrivals and departures for these states, are also

displayed from their respective data structures.

L3784-4071 these functions display the numerical data for each Universe, for the current hour HH, of the

counts of viral states present, arriving and departing as well as the roles compositions for each

universe. L3772-3780 is an efficient and terse way of executing as many Universe tabular displays

as there are active universes, provided that they are active in ascending sequence numerically.

L4075-4716 These functions are the charting functions for 20 charts – six for the single Universe screen, the

rest for the nine Universes and the Multiverse as a whole. They should be self-explanatory but

if not guides to canvas.js are all over the Internet.

L4722-4763 these are the readfile routines, adapted or copied wholesale from Messelman previously

cited above. My part comes in L4756 where the call to processLines( ) calls my routine L2503

L4771-4779 global variables for the animation of the TRAFFIC screen

L4784-4789 function CreateNode( ) – the prototype object for data structure of a node (Universe), with

coordinates x,y

arrays to[ ] and from[ ]

L4791-4802 function CreateEdge( ) – the object prototype for an edge for nodes (I, j).

Note that edge (i -> j) while occupying the same space is not the same as edge (j -> i) because

of agents that move from (I -> j) are different from those that move from (j -> i)

The dx, dy fields are for animation, the count ct will be expressed as numbers that move

L4806-4827 function graphB( ) is the main routine for the TRAFFIC screen, and is invoked by the TRAFFIC

button, or by the “>” and “<” buttons when activated through the TRAFFIC slider

This routine initializes the TRAFFIC screen through the boolean netFlag which is TRUE if in

initialization is required, and set off otherwise until exit from TRAFFIC screen through DOUBLE

CLICK of the “STOP/EXIT” button. Thereafter, following initialization, the TRAFFIC screen and

network traffic is calculated and shown every time the “TRAFFIC” button is selected, for the

DDHH that the TRAFFIC slider and its modifiers “<” and “>” selects.

L4829-4830 function graphNetwork(gen) is the main routine for showing the traffic for the generation “gen”.

It calls functions to draw the TRAFFIC screen with the active universes, then calculating the

traffic data, and animating them.

L4837-4843 function graphDDHH(gen) just writes the DD and HH for the numerical generation “gen” onto

the TRAFFIC screen. It is called from graphNetwork L4829

L4845-4848 function graphStop( ) is invoked when the “STOP/EXIT” button is clicked ONCE. It stops the

TRAFFIC animation by clearing the interval timer “Ntimer”, and because the TRAFFIC routine

can be called while in AUTOMATION mode from the Multiverse, also stops automation by

clearing the interval timer “clockTimer”

L4850-4858 graphExit( ) is called when the “STOP/EXIT” button is DOUBLE-CLICKED. Some house-keeping is

done – making the TRAFFIC screen invisible, clearing the interval timer “Ntimer”, resetting flags.

L4864-4874 function graphIN( ) is called when the TRAFFIC slider is touched, and sets the slider to the

extreme right, with value of 1920 (set as maximum for the slider), and calculates, for the

slider value set by the mouse, the generation equivalent for the slider position, given the

current generation, and resets the global variable “gen” to it, so that the appropriate data

structures can be accessed. Note that the current “gen” has been saved to “saveGen” L4824.

L4876-4883 function backGraph() is called by the “<” button to take the slider generation to one previous

L4885-4892 function foreGraph( ) is called by the “>” button to take the slider generation to one following

L4894-4898 function writeSlider( ) is used to reset the slider position when the value is changed by < and >

L4902-4915 function StartNet( ) is used to initialize the positions for the nine nodes representing the

Universes, and to create them as TRAFFIC nodes according to the prototype in L4784

L4920-4926 function drawNArena( ) simply draws the TRAFFIC screen in black and the Universe nodes

L4929-4939 function loadNet(gen) is a function that initiates the creation of the data structures for each

Universe for departing agents at the hour corresponding to generation “gen”. These structures

are multi-layered – for any Universe, it is not just the viral state counts (green yellow blue etc)

but it is for each edge to another Universe, the count of these viral states along that edge. So

first the function sumDep(I, gen) checks if there are ANY agents of whatever state along an

edge, and if there is, then the function markEdges(I, j, gen) creates the structures for the viral

states along that edge

L4941-4950 function sumDep(I, gen) checks for the Universe[i] at the current time represented by “gen”

whether there are ANY departures of any viral state (color), as depT[ ] is an array for each

generation that has a structure defined by CreateType( ) L2212 which has green, yellow, etc

counts. So without testing each, if their sums are > 0, there are one or more departures.

L4953-4960 function markEdges(I, j, gen) checks the more detailed data structure for Universe[i] relating

to Universe[j] in the U[i].dep[gen][j] which is an object with green, yellow, etc counts for

agents from [i] to [j]. If the for the color from [i] to [j] is greater than zero then the function

loadEdge(I, j, gen, clolor) is called

L4962-4986 function loadEdge creates a new data structure of type “Edge” using the CreateEdge( ) function

L4791. The approach here is that in the NETWORK (traffic) array N where N[i] is the ith Universe,

to N[i].to = [ ] array is a set of instances of Edges, each object Edge describing the destination

Universe from [i], the color of that edge (viral state), the numbers of agents going from U[i] to

the destination, and each Edge of N[i].to is a <destination, color, count> structure which is

part of the Edge object

L4948-4993 function calcdydx( ) initiates the graphics calculations for each Universe

L4995-5002 function focusN(n) is called by calcdydx for Universe[n], and for all its Edges as created in the

function loadEdge L4962, calls the function calcE(n, e) to do the animation

L5004-5027 function calcE(n, e) for universe[n] and its departure Edge “e” finds the positions of the two

Universes involved in the TRAFFIC screen as defined in function startNet( ) L4902, generates

a dx value using the tangent ratio of the delta y/delta x and the GRANULARITY constant

L5029-5132 the remaining functions in this section enable the animation of the agent classes as numbers

moving simultaneously from one node to its destination nodes along edges. The code is

straightforward, using set Interval animation, and the various parameters can change the

speed of the numbers moving along edges as well as their separation or clustering. As this code for visualization is of little interest to the R-integration project aimed at the core

simulation before visualization, we will dwell no further on the details of these functions.

**CONTROL FLOW FOR CovidSIMVL**

To better understand the logic and modular structure of CovidSIMVL, the following describes the flow of control of

CovidSIMVL briefly.

Initialization

The design of CovidSIMVL is a simulation tool that collects information about agents as the simulation proceeds, while displaying this information at each step, if desired. Therefore, the simulation can proceed manually, an hour at a time, or in automation mode, running to and end time specified as DDHH from the start.

In the spirit of visualization, CovidSIMVL uses three main screens for the same stage. The first is the Single Universe screen, and the one that is invoked in the startup. The second is the Multiverse screen, called from the first, and the third is the Traffic screen, called from the MV screen.

Initialization proceeds in three steps, all requiring user intervention at this point. The first is a prompt for the user to enter the total population of agents for the run, and this allows the system to create the agent population in the array P, and their number in the data structure for the Multiverse M, as M.PCt.

The second and third steps are for opening CSV files describing population characteristics. Instead of a fixed name for them, and opening them automatically, CovidSIMVL as currently construed opens a window with a file-open button, and this directs the user to point to the file in whatever directory the user has placed it, for the file that describes the traffic pattern of agents to and between Universes, and their ROLES in those Universes, as well as the mingle factors associated with them in each such Universe. This file for demonstration purposes has been MVDATAage.csv, and as other scenarios are modelled, others with different names will be created.

This file does not describe the viral state of the agents, and since CovidSIMVL is using the temporal dynamics – viral growth model from Xi,He as referenced elsewhere in this document, there is a need to start the simulation with a number of infected agents.

These are called CASES, and following the reading of the MVDATAge.csv file (for which there is no specific completion code), the user needs to press the CASES button along the top row, in order to bring up the file reader window that allows the second CSV file to be designated. This file has the name of VL1.CSV, and is described earlier in the document. The final step is the click on the “LOAD” button – this must be done to start the Master Clock.

At this point, initialization is complete. The routines for initialization are straightforward, with the exception that the data structures for the Traffic presentation have also been generated in large part, though the actual execution of the Traffic presentation creates a set of temporary data structures for each Universe at any DDHH of the simulation (using a slider bar and “<” and “>” buttons), describing the numbers of agents in their viral states (green, yellow, blue, red, orange for susceptible, infected, infective asymptomatic, symptomatic, inert) that leave the Universe along each edge joining it to their destination Universes.

These data structures are derived in part from the traffic definitions in MV, and in part from their current viral status at the current time. As the viral states for each step (“generation”) are saved in data structures called logxxx

for each Universe as well as the totality of the Multiverse, these temporary structures are created just to make coding and animation easier.

Initialization, data structure definitions, creation and loading, and utility routines such as prompts for parameter changes, draw routines, canvas management, charting functions for visual displays, screen management, take up the lines of code to line 2895. From here to L3510 are functions that are concerned with the simulation of agent behaviour (the core Simulation Engine). From L3510 on, the code is again about tabulating counts and displaying them appropriately on the screens, including the routines for managing the data structures and animation for the traffic screen.

Therefore, the main SIMULATION ENGINE goes from LINE 2895 to LINE 3510, only about 600 lines of code.

CovidSIMVL Simulation Engine and Control Flow

The concept in this simulation engine is that there is a Master Clock, which advances TIME by the hour, and provides the current time in DDHH format (although some of the charts may show time as DD.dd where .dd is decimal hours, to correspond to the Xi, He model).

The simulation has to deal with agents who are in a Universe, and move around there, with contact and viral contagion as well as transitions in viral states according to the time after infection for each infected agent. This important and subtle point requires each agent data structure to carry the time of its infection, and for the change in viral state to be relative to that time, for each agent.

The simulation has to deal with travel of agents between Universes at their designated DDHH, as specified in the first csv file.

The simulation has to deal with the growth of the viral load for each agent, which in the model is increasing from time after infection T=0 to the peak viral load at T=5, and then declines from T=5 to T=13 approximately. Growth has to happen after contact and viral load transfer.

The simulation has to deal with the Hazard Sixe of each agent, which is a proxy for both susceptibility as well as infectivity. If the size is zero, no one contacts any other, If the sizes are large compared to the fixed arena in which the simulation takes place (800 x 600 pixels) the likelihood of contact is not just between pairs but tuples.

In summary, the control modules deal with time management, traffic management, contact and viral transfer, viral growth and viral state change among agents, and the sequence in which they take place.

THE CONTROL SEQUENCE

This is probably the most important part of the simulation. The concept is that things happen at the top of the hour. In other words, if workers leave at 5pm (100hrs), the actions happen when the time advances to 1700, and not at the end of the hour. Thus, AdvanceTime( ) L3240 has to happen first.

Following this reasoning, the arrivals and departures should happen right after, or very soon after. For reasons that will be clear, before the traffic moves persons from Universe to Universe, we actually invoke growVL( ) L3308 after,

because the viral transfers from infection subsequent to contacts within a Universe grows in the interval between one clock time and the next.Following growth, resizing takes place in reSizeAll( ) L3344

The actual movement of agents takes place after growth, through functions injectXY( ) L3369 and expel( ) L3445.

Conceptually, the flow is:

advanceTime( ) 🡪 growVL( ) -> reSizeAll( ) 🡪 injectXY( ) and expel( )

From there, we would invoke the routines for movement and contagion within each Universe, calculate all parwise contacts, estimate their overlap and viral gradient, and produce a viral transfer from high to low for applicable agents (makes no sense for inert to infect inert, or infective to viral transfer to inert).

THE ACTUAL CONTROL SEQUENCE

The actual control sequence is slightly more complex, because the logic is driven by the AdvanceTime( ) function being called from either the manual “HR++” button, which advances the clock by one HH, or by the same function in automation mode.

So, clicking on “HR++” button will take us to the load( ) L3197 function, which has, after initialization, had its appearance change from “LOAD” to “HR++”. This routine, behaving as “HR++” invokes TimesUp( ) L3217.

This is the routine that will lead to advanceTime( ). Be patient.

The function TimesUp( ) L3217 does initialization of the data structure for Traffic for this generation (from start), and if in AUTO and a terminal DDHH time has been reached, shuts off the interval timer, and reverts to “MANUAL” mode. In all other cases, increments the master clock through advanceTime( ), then calls conductor( ) L3276. The master menu



is visible above all three screens, and therefore if the HR++ is clicked in TRAFFIC screen mode, the TimesUp() function, having advanced the clock, called conductor( ) and tabulate( ) functions, then puts redraws the

TRAFFIC screen with the new DDHH, and shows any traffic movement for that time, through graphB( ) L4803.

The function conductor( ) L3276may also be thought of as the main controller because it directs the movement of agents from one Universe to another at the new time.

So, advanceTime( ) comes first, increments the DDHH clock, invokes growVL( ) L3308. Then conductor( ) gets first dibs at actions following the new time. It is conductor( ) that examines all the tickets in the data array T{ }, and for the DDHH that is now current, finds what universes are involved, the agents that are involved, the actions for that time slot for that agent, and calls injectXY L3369 or expel( ) L3445.

The revised and actual sequence of control then, to this point of description, is:

HR++ -> TimesUp( ) -> advanceTime( )+growVL( ) -> conductor( ) -> injectXY( ) / expel( )

The reality is slightly richer than the above, because growVL( ) is also responsible for changeState( ) L2566, and for any size changes through reSizeAll( ) L3346.

A few words about these two: since the Xi, He model is temporal-dynamics driven, advanceTime( ) is the opportunity to check the time-based segments of the model, and thus change the state from yellow (incubating) to blue (infective pre-symptomatic) and so on. Furthermore, we have chosen here to equate infectivity with viral load – the larger the viral load, the greater the gradient for transfer, and the greater the infectivity, which is expressed through size as the increased likelihood of contact during movement within a Universe.

So, the final control flow in this part is:

HR++ -> TimesUp() 🡪>advanceTime( )+growVL( ) 🡪 conductor( ) 🡪injectXY( ) / expel( )

|

|

changeState( ) 🡪reSizeAll( )

Movement with a Universe

This happens from conductor( )L3276 after the inject( ) and expel( ) actions have been taken. Now all the agents are where they are supposed to be after the new time has come about. Note that the TRAFFIC screen is a view of what happens after the within-Universe moves (which has a number of moves per HOUR as specified by the parameter “cycleMax”, The code exists for this parameter to be changed by the user but the button and function is at present inactive.

A WORD about when the generation count is incremented. After some trial and error, the least inconsistency between action reported and time-specific action is for gen++ to happen immediately in advanceTime( ) function, in keeping with the clock advancing there, before any action is taken.

IN ORDER FOR the cycle to start at 00, consistent with midnight at hr 24 == Dnext + HH=00, the initial value for gen before everything starts is set to -1, and the clock max is 23, so that if the clock is 23, and it advances, it becomes 00 and not 24. These details were nitpicky but very important for correlating the simulation with expectation.

MOVEMENT routines: moveItmoveIt( ) L2895. This function is supported by, or extended into, the functions:

moveItmoveIt( ) L2895

proposeMove( ) L2933

testWall( ) L2960

testOverlap( ) L2998

nearestEpicenter( ) L3026

adjminglf( ) L3062

VLtransfer( ) L3088

The flow of control in this is pretty clear.

The mingle factor is a variable for every agent, and is applied in proposeMove( ), L2943 and L2944 when delX and delY (the displacement from current X, Y in proposing a move) are generated. THE UNIVERSE-specific base mingle factor is a concept that has not been implemented, although the data structure for a Universe is present as U.minglf. Its semantic is that a Universe may have an underlying situation such as poor ventilation, or inherent mingling such as a dance hall, which would add to the mingle parameter for an individual agent. Its application would be in the proposeMove( ) function.

The epicenter concept would be used to model places where a cluster of persons would linger around a certain location within the arena, such as in restaurants, or after-conference receptions where there are stand-up tables, or functional departments within open-area offices. The concept has been implemented in demonstration mode, and the code exists here in nearestEpicenter( ) but not activated in this CovidSIMVL, as the epicenter specifications for each Universe does not have data structures supporting them as yet.

The calculations of viral growth and viral load transfer form the next and last part of this Handbook.

**CALCULATIONS IN CovidSIMVL**

There are two sets of calculations that will be described below:

1. Calculating movement within a universe, the overlap between pairs of agents, and the viral transfer that may occur between qualified pairs.
2. Calculating viral growth within an agent, and the resulting change in size as a consequence

MOVEMENT RELATED CALCULATIONS

There are three parts to the movement calculations: the first is proposing and finalizing a new position for an agent in a Universe, the second is to test whether or not overlaps occur for pairs of agents, and if so, whether they satisfy the conditions for viral transfer, and the third is to calculate the amount of viral transfer between such pairs.

1. **Proposing and finalizing a new position for an agent in a Universe**
2. **Test for contact and overlap between pairs (I, j) agents in a Universe after their moves**
3. **For those pairs satisfying viral transfer conditions, calculate the addition of viral load to the recipient**

**Preparing and Finalizing a new position for an agent at co-ordinate (X,Y)**

**For each of “cycleMax” repetitions in a generation (clock HH), moves are generated and finalized, so that the opportunity for infection happens with an intensity of not once per HH, but cycleMax times per HH.**

**The functions involved are:**

**moveItmoveIt() 🡪 proposeMove( ) 🡪 testWall( )**

function moveItmoveIt( ) L2895

sets up the repetition for cycleMax, and uses the global “wU” for the “working University” to identify its occupants at the current time given by Q.person. For each i-th person k[i], the parameters for the Person structure and the Person ID is passed to proposeMove( ) L2933. Then testOverlap( ) is called, which calls VLtransfer( ) as needed.

PROPOSING AND FINALIZING A MOVE with function proposeMove(G, g)

where g is the person ID and G is the data structure for that agent

The calculation first finds a random selection from 0 to 43 through Math.random( ) \* 43. This is used as an index into the array “travel” which contains numbers 1 to 9 in a decreasing number of repetitions, so that smaller numbers are more likely to be selected by the index than the larger ones.

This delX is multiplied by the size (G.currSize) of the agent / 2 [minimum displacement half the radius]

This method is used independently to create a proposed “delX” and “delY” which are displacements from the current position (X, Y).

Now the mingle factor for G is applied:

G.delX = G.delx \* G.minglf / 10

We divide by 10 in order to control displacements. If the size is 12 pixels, then if on average the stochastic array return a value of 5, and the mingle factor is 5, we would have a delX of 12 x 5 / 2 x 5 = 150 which on average moves everyone 25% of the height in delY and 35% of the width.

Then we use a random direction finder (if math.random( ) \* 2 > 1) to change the sign of delX and delY to go in the opposite direction.

In summary, the calculation is:

delX = +/- (Paretp[1 to 9]) x agent radius/2 x agent mingle factor / 10

It is easy to change the calculation, either by changing the constants, or any other way, to generate stochastic movements of the agents. ZERO movement agents could represent items like door knobs, table surfaces.

The proposed move causes G.newX to be G.X + delX and similarly for G.newY. But we have to see if the new positions are out of bounds. This is done in testwall( )

.

function testwall(G, g) L2960

If the newX position is < 0, it is out of the screen on the left, and if it is > canvas width, it is out of the screen on the right. Similarly for newY.

These conditions are modified by testing for the new position within half the size of the agent, and that the direction of the proposed move is indeed pushing it out of bounds.

If so, then the direction (sign) of the delX or delY is reversed, and the new position is the current plus the reverse direction of the delta displacement.

G.newX = G.X + delX where delX has been given a change in sign

G.newY = G.Y + delY if movement of Y is also out of bounds

If this still causes problems (eg the delX is just too large), then the new position for X and Y is just 10 pixels within the offending border. Now to test for overlap

TESTING IF TWO AGENTS OVERLAP with function testOverlap(G, g, epic)

The code for moveItmoveIt( ) contains a call to nearestEpicenter( ) L2912 assigning that epicenter number of the variable “epic”. However, this is presently not used in CovidSimVL for reasons described above.

testingOverlap(G, g, indx) ensures that a pair of agents (a, b) is tested once for (a, b) and not again for (b, a). This is done in the code lines:

k = Q.person;

len = k.length;

for (i=0; i<len; i++){

j = k[i];

..

..

testOverlap(P[j], j, i);

}

For the Universe Q, the array Q.person contains the pIDs of the persons in that Universe at the current time. The assignment

j = k[i]

gives j the i-th entry in the array k[ ] which has a copy of Q.person. Therefore P[j], j refer to person with pID=j and data structure P[j], while I is the index in the array k for this person.

In testOverlap(G, g, indx), the person g ie P[g] is tested against everyone in the array k[ ], in a for loop with control:

for (j = indx+1; j<k.length; j++) {

if (j != indx) {

….[proceed to test P[g] against P[k[j]] where j is the index into k[ ] so P[k[j] is the person

…

and the comparison is into the triangular matrix and no element is compared against itself

The test for overlap between two agents basically looks at the separation of their centers (X, Y) compared to the sum of their radii, which would be their distance If they were just touching.

The separation of the centers is simply the square root of the hypotenuse defined by the distances between the X’s and thd Y’s.

The function call VLTransfer(g, k[j]) applies only If both agents are not susceptible at this time. Otherwise one of them has a viral load and can potentially transfer it to the other.

CONSIDERING VIRAL TRANSFER with function VLtransfer(I, j)

First, if either of the agents “i” or “j” are “ORANGE” – ie inert then the routine exits…as inert agents cannot affect or be affected. So our consideration space is only Yellow, Blue, Red (infected, infective asymptomatic, and symptomatic infective).

Two considerations need to be dealt with: one is the viral load gradient between the two agents, and the other is the extent to which the two agents make contact (overlap). The idea here, conceptually, without present evidence, is that a higher gradient (difference) means more viral transfer to the lower, and furthermore that if the overlap is very small, only a bit of the potential transfer will occur, while if the overlap is total, then the entire viral gradient applies.

This approach is pragmatically reasonable, and without it, we would need to re-cast what happens with contact. Does every contact no matter how much overlap create the same degree of infectivity? If so, we would be abandoning the idea of viral load making a difference in infectivity, which appears at the moment to be adopted as a working lemma; and furthermore, we would be using size as a binary threshold for infection without consideration of whether overlap matters. Overlap indeed is a proxy for proximity of two agents, which is another lemma that is in current adoption for Covid-19 epidemiology. Perhaps it could be a proxy for both nearness as well as duration of contact. Thus, abandoning viral load and overlap is difficult to justify, whereas keeping them and adjusting their influence can give rise to variations of simulations.

In this routine, we satisfy more constraints after the “Orange-inert” test.. From the Xi, He model we have constants L2022:

var VLlower = 3.6;

var VLincD = 2.9;

var VLpeak0 = 4.5;

var VLpeakVL = 10;

var VLonsetT = 5.2;

var VLpeakFnd = 6.2;

var VLinfEnd = 13.2;

and we check that both of the agents have viral loads that exceed the stated average load for infectivity ( VLlower = 3.6), and that for both agents, the maximum elapsed time between their initial infection time and current time is greater than the time when infectivity starts (VLincD = 2.9). If either of these are not met, we exit from the function. We have eliminated the case of two YELLOWs

These hurdles having been overcome, we now identify which agent has the larger viral load, and calculate the gradient. In testOverlap( ) we had calculated a global variable “raMax” which is the sum of the sizes of the two agents, and “yLength” which is the actual distance between their centers.

Now we calculate a factor “Vdist” as the (cube of the [ratio of the actual distance/raMax) where the cube nakes the length proportional to the volume of a sphere, so that this ratio is related to the intersection of the volumes defined by the two agents with their different radii.

This factor, Vdist is used to calculate the amount of Viral Transfer, “VTrans”, from the Viral Gradient “Vgrad” in;

VTrans = Vdist \* Vgrad

If this number VTrans, arbitrarily is very small (0.05) we consider that the infection does not take, and is a failed contact, and we add one to the infecting agents count of failed contacts:

P[vbig].failedCt++

At this point we don’t know or care whether the infecting agent is Blue or Red, but from the qualifying tests, we know it cannot be yellow, or have a viral load less than 3.6.

Now we calculate in L3128-3129 the post-infective days for both agents, given the current time DDHH expressed in decimal form as “CT” global variable, and the P[ ].tInfect fields for time of first infection for the agents.

The lines3131-3134 are another example of cascading if statements that are more efficient than if else or case. First, if agent P[i] has passed its pre-infective period VLincD (incubating days) then its P[i].touchCt++ is incremented. Then if this is the case, and the other agent is “green” or susceptible, we add one in P[i].susCT++. If the other agent is also post-incubation we add one to its touchCt, and finally in L3134, we cannot have both be “Green” so only one of P[i].susCt++ or P[j].susCt++ would be executed.

The next set of code form L3136 to L3152 performs the following:

1. Ensures that the smaller agent has a new viral load at least as large as the model Xi,He by looking up its post-infective days against the table “tViral” L2018.
2. Ensures that the smaller agent has a new viral load at least as large as 1 (this will be for new infected)
3. If the smaller agent has a time of first infection = 0, it must be a susceptible, and we set that time to CT.

This ends the rather complex VL transfer function, which permits the addition of viral load to all but inert agents from an agent with a higher viral load, using viral gradient and overlap distance as factors controlling the extent of additional viral load.

**NBNBNB**

THIS SIMULATION MODEL DOES NOT COVER SERIOUS AND CRITICAL CASES THAT BECOME HOSPITALIZED, ENTER ICU OR LEAD TO DEATH AS A COMPLICATION OF COVID-19. THE CENTRAL THEME OF COVIDSIMVL DEALS WITH THE SPREAD OF THE VIRUS AMONG SUSCEPTIBLE POPULATIONS THROUGH CONTAGION, CONTACT AND MOBILITY.

**ADDENDA**

Aug 14, 2020 - #1508

calculate R0 in Mtabulate()

display R0 as run proceeds in Single Universe View and in Multiverse views

modified data structures to accommodate R0

Aug 22/2020 #1508a

L2176 replace blank line by “ U.allTouch = 0”

L2133 replace blank by “ this.allTouch = [ ];

remove blank L2159

Aug 23/2020 #1508b

replace or remove references to “P.size”

already have P.baseSize and P.currSize

Aug 26/2020 #1508c

fixed tViral[ ] table lookup error when total time in trial > 24 days

L3140 and L3152:

if total days < 21 use the table otherwise use [ viral load = 1 ]

Aug 31/2020 #1508d

change Mingle=0 in .csv file to mingle = 0.01;

if mingle==0 then the action should be NO MOTION ie parking

two places – read in CSV file

blank out L2939 to L2941

SO FROM NOW ON, CSC files should have minglf==0 for no motion, and minglf=0.01 for v. small moves

Aug 31/2020 #1509

introduce FAMILY KEY as a new field in CSV file, so that within the universe called HOME, only family

members will infect one another…..this is a way to partition persons within a universe…..in theory, this

could be extended to other kinds of sub-groupings, but enough for now

This requires a change in Person definition P.this and a change in the readfile function, as well as a

change in the testOverlap( ) function that goes into VLtransfer( ) if an infection is taking place.

L2248 replace blank line by “this.famKey;

L2313 replace blank line by “ P.famKey = -1; “ -1 is an independent – no family members for -1 L2437 new “P[ID].famKey = eval(lineS[9]);

L3017 MAKE HOME UNIVERSE defined as 8 until we get symbolic names set up

L3017 – L3019 if overlap and not HOME go to VLtransfer( ), and if overlap and Home do VLtransfer( ) if

famKeys are the same and not -1

Universe names are stored as UN[i].name L1826 but they are being co-opted for R0 at present, though we

are finding that R0 is not useful metric in the multiverse. However, we need to display them for each of

the Universes not just the whole Multiverse, so we need to find another place for the R0s.

Ideally, they will go where the green circles are!

ctx1, ctx2, etc L1418

can use canvTxt L2710

L2704 new routine drawR0Txt(canv,xy,y,R0val)

L3776 shows “pane1”.innerHTML = U[0].allTouch

we don’t need this because the count of allTouch in this universe no longer very interesting

but we can put it into the drawR0 – remove

If we want allTouch we have to go to WATCH in devtools and inspect