Parameterized COVIDSIMVL

March 18, 2021

Introduction

In order to parameterize CovidSIMVL and avoid user interface interactions, we must use three files that have to be read by the program.

The first would be a parameter file which would set the parameters, at present for

Population size

Mingle Factor

Hazard Radius

The name of the Population file

The name of the Case file

They all have to be .json files. The parameter file should always be called:

param.json

and within it, one can specify whatever is desired for the Population and Case Files. Just be sure that the Population Size is correct within the Parameter file.

Operationally

The population and case files are to be defined as .csv files as in the GUI version of CovidSIMVL. They then have to be converted to their .json equivalents, using the program “CSVtoJSON.html”

For reasons discussed below, the .csv files have to be in the SAME directory as the CSVtoJSON.html file.

However, as this is a browser based program, the outputs are considered to be DOWNLOAD files, so they will be found in the download directory:

For user “fredSmith”, usually in a directory like:

C:/users/fredSmith/downloads

These will have to be copied into the directory from which the paramSIMVL is to be run.

Sorry, no way around these. Web browsers cannot write into arbitrary directories, just download.

The easiest way to specify input files is for them to be in the directory from which the program executes.

So, steps are:

1. Convert Population and Case .csv files to Population.json and Case.json files
2. Copy them from the download directory to the program execution directory
3. Create the parameter file using Excel to create a .csv file and convert as above
4. Save your parameter file as some specific name, but the copy it, or rename, to “param.json”
5. Make sure all three files are in the same directory as paramSIMVL.js and its index.html file, which we will call paramIndex.html

Run paramIndex.html in a browser, or through R….no use input will be required for setting parameters.

Technology used

I am not a web or xml maven, so these routines have been cobbled together from examples I found on the web. They work.

The conversion program, CSVtoJSON.html, requires the use to use the Windows system to locate via directory traversal the .csv file that is desired.

This file is read into the program as a blob and what is made available is the string within the blob. This is copied into a variable COLLECTION, and the JSON output routine takes this and

1. Stringify it into the blob format
2. Calls blob to make it into a JSON blob
3. Outputs it using URL.createobject(blob)
4. This gets saved in the download directory through the lines:

json = JSON.stringify(data);

blob = new Blob([json], {type: 'application/json'}),

url = window.URL.createObjectURL(blob);

a.href = url;

a.download = fileName;

a.click();

So the file reference is the url created for the blob JSON file, and the a.download downloads it.

There are probably many different ways to write files out from a browser, but this is what I found that I finally got to work, so there you go.

**The read of a JSON** file into a string that the original .csv reader can process is through this:

We use XMLHTTPrequest and wait for completion, then unwind the blob and the string to get the data into COLLECtiON2 as a variable.

function myLoad() {

iLoad("param01.json");

COLLECTION2 = loadresult.x;

}

var txt = '';

function iLoad(xFile){

var xmlhttp = new XMLHttpRequest();

xmlhttp.onreadystatechange = function(){

if(xmlhttp.status == 200 && xmlhttp.readyState == 4){

txt = xmlhttp.responseText;

loadresult = JSON.parse(txt);

}

};

xmlhttp.open("GET","http://localhost:3001/"+xFile,false);

xmlhttp.send();

NOTE: We download node.js into the system to be able to read and write across domains, which web browsers typically cannot do with local files. Node.js creates a tiny server process to enable file read/writes. There is a start.bat script and two nodeServer and nodeListener .js files that are also needed in doing the conversion and JSON inputs. Furthermore, AJAX is used for these read/write routines, and is included in the html files as <src= >.

**FORMAT of JSON parameter file**

This is the Excel file which should be saved to a .csv format…



The first line must be “Parameters” and I don’t know whether they comparison will accept just lower case or just upper case. Try to use as shown….same with the other parameter names….note they can be in any order…even the STOP parameter…

If a parameter is not entered, don’t leave a blank line…..just use what you need. Don’t put commas in if you are using large numbers.

IMPORTANT: the RedDays is the days from infection to the end of infectivity ie the transition to Orange (inert). Yellow (Incubation) is 2.2 and Blue (pre-symptomatic infective) is 3 days, so symptomatic always starts at 5.2.

If you want clinical symptoms for 1 day, you would enter 6.2 (5.2 plus one).

**For Apple and Linux users**

At the moment, I have not investigated the equivalent to JSON and node.js and AJAX for Apple and Linux. They probably exist. All this code is open source, so feel free to push the boundaries outward.

PROGRAMMING TECHNIQUES USED

The variable

**Var use\_html = true [or false]**

is at the very beginning of the Javascript program, and dictates whether or not calls to HTML and GUI functions will be made in the program as it executes.

Console.log( ) calls have been pooled into one place through the creation of the function:

**Console\_log( x ) which is a function that calls console.log(x) where x is the string**

Similarly, the function ALERT(x) has been collected through the function alertX(x) which calls alert(x).

In the same way, the functions

GUI(x,y) and GUI.style(x,y) are used to collect the HTML calls:

document.getElementById(x).innerHTML = y

document.getElementById(x).style.display = y

So that there is one place that these innate functions can be disabled or re-routed instead of the dozens of such calls sprinkled in the program.

**THE HALTING PROBLEM**

There is no nice way for a javascript program which is not at node.js JS program to self-terminate. The desire to terminate comes in two circumstances (at least):

1. When there are no further chances for infection (no incubating, presymptomatic or clinical cases).
2. When the STOP parameter number of generations condition is met.

There are two program ways to halt in the system and still be able to use devtools to inspect variables, save the console.log etc.

One is to use:

Throw “string” and not to worry about catch( )

The other is to set:

MODE = “manual”;

clearInterval(clockTimer); in this case it waits for input from user….

**FURTHER EXPLORATIONS OF PARAMETER SETTING**

Potentially, this is just the beginning….we could ask the program to provide a different set of viral temporal dynamics; we could periodically dump the status of age-groups, vaccinated persons, we could invoke functions not yet written at various times….we could etc etc…

BUT so far, we have what we have….to do more, we would have to add functions, and invoke them when the time is right, ie something like

CALL gen function name parameters

This is not anticipated as a priority at the moment.