DualUse CovidSIMVL

April 1, 2021

Introduction

The motivation for having two variants of CovidSIMVL within a single piece of code is that there is only one body of code to maintain, and changes and customization of the core logic and simulation dynamics do not have to be repeated.

Why two variants and what are they about?

The GUI version

CovidSIMVL is written in Javascript and therefore runs in the Internet environment, out of browsers like Chrome, Safari, Edge, etc. with devtools and console.logs for dumping out information at relevant points in the simulation.

However, the whole idea of browser-based programs is that user interactions (mouse clicks, form filling, etc) are central to their execution. This is the GUI (Graphical User Interface). Parameters, files to open or download, etc are entered by the user.

Indeed, when you call up a javascript progam on your browser, on your computer, that program cannot read your local files from within itself, until and unless you select the file through an input form (select file to open).

The advantage of the GUI version is compliance with security, the availability of the devtools support environment in modern browsers, which permits examination of variables and arrays, and the console.log, which allows the program to produce information relevant to the simulation, at any time from any place in the program.

This version is simple to understand, and also very good for rapid prototyping of changes to simulation processes and outputs.

The no-GUI version

All the user interface elements in the GUI version are dependent on HTML declarations and displays (or suppressed displays), mouse interactions for parameter setting, etc.

HOWEVER, this GUI mode of operation is not conducive to repeated running of the program on one set of parameters, or indeed, on parameters that change from run to run, without having to change the internal parameter variables in the code. The solution is of course to have a parameter file.

Furthermore, the selection of data files to use would be impossible in the usual internet browser environment, because of the security provision that a file from a server running on a local machine should not be able to read a local file without asking. This is called the Cross Boundary problem.

OF COURSE, there is a way around all this. That is to use an environment called node.js.

Node.js when used with files encoded in the JSON format, permits files to be read on local machines from programs served through node.js on the server.

Therefore, the no-GUI node.js version allows a parameter file to be read from the user’s local directory, which can then contain names of data files, and the names of parameters that the specific simulation requires to do its tasks.

DUAL USE Version

Now it seems that there has to be the two versions. However, we have succeeded in bringing the two code streams together in one body, and execution follows one stream or the other depending on the input URL variable “use\_html” that can be appended to the execution path, as seen in many web-pages.

This is done by adding the “?” operator at the end of the URL, such as “URL name.html?use\_html=true”

The Operating Environment in the User Computer and Directory

***To use the HTML GUI version***

To invoke this, one looks, for example, for

DUALuse.html

which is the html segment that refers to DUALuse.js. This program is hard-coded to have the variable

use\_html = true

so you do not have to use the URL parameter ?use\_html=true in invoking the GUI version.

The population and case files are user-selected through the GUI, so do not have to be named.

The parameter file for the no-GUI version can be accepted for both variants, without change (except that the GUI version will accept the .csv format, while the no-GUI version has to have the .JSON format – see below).

***To use the no-GUI nodejs variant***

To use node.js, one has to download the node.js environment in the user computer, from the web-site

[Download | Node.js](https://nodejs.org/en/download/)

Within the user directory, our sample directory has three files

node\_start.bat

node\_requestingServer.js

node\_respondingServer.js

With node.js system installed, double-clicking on node\_start.js will invoke node.js to create processes for requesting and responding servers on the local computer. This will enable the CovidSIMVL to read local files as named in the “param.json” file.

This name is hard-coded in the simulation program. Of course, various parameter files can be created and used, but the relevant one for a particular simulation needs to be “param.json”.

To convert a file from .csv format to .json format, simply use the included file in the example folder called

CSVtoJSON.html

from within a browser, in the usual way…it will be a GUI program which asks for the name of the input .csv file and the name of the output .json file.

NOTE: in the usual web-practice, the newly minted file will be downloaded into the user’s DOWNLOAD directory.

**FORMATS of input Files and a small difference**

Let’s discuss the differences first:

In the no-GUI variant, the parameter file has to come first, to set the population and universe numbers and their arrays. The parameter file also contains the names of the population and case files, as well as the parameter values, some of which are global and others are Universe-specific.

In the HTML GUI variant, the population file can come first, and the parameter file come last, in case no parameters need to be set, or the user wants to set them manually (in the GUI).

Therefore, in the HTML GUI variant, the population file must also contain the number of Universes as well as the population number, so that the arrays can be created appropriately.

In the HTML variant, the number and names of the Universes are in the first line of the population file, which looks like this:



The first line is the new Population file identifier, with the population number of 600, then the number of Universes following the identifier “Universe” (if absent, defaults are used), then their names.

The Case File just has the first line with “Cases” in the first column.

The parameter file is more complex and flexible. Here is a comprehensive example. Any parameter left out will use hard-coded defaults.



The HTML version of parameter parsing will ignore the “UN”, “pop file”, “case file” entries.

The HzR (hazard radius) is a global setting, but “sizeF” is Universe-specific, and is a multiplier on agent entry into a Universe, and a divisor when exiting. It is used as a density-equalizing factor.

The mingle factor mF is also Universe specific.

For these two parameters, each line can specify a different Universe and parameter value.

The RedDays parameter is also global, specifying the number of symptomatic days (symptoms start on day 5.2 following infection).

The STOP parameter permits the user to stop the system at a generation limit unless the self-extinction condition of no further infectives is reached first.

Standard vs Customizing

What is in github in the DUALuse folder is the standard version which should be considered the canonical version, as it also has bug fixes for a number of niggly problems, which will not be retro-fitted.

The STOP condition invites a WRAPUP( ) dump of information for analysis. This is something which can easily be customized by user-programmers, or perhaps found ad hoc from further processing of the console.log files.

MAKING THE PROGRAM ADAPTABLE TO BOTH HTML and not-HTML

To a very large extent, all calls that use GUIs have been funneled to a single function. For example, console.log(“string”) calls are liberally sprinkled in the CovidSIMVL simulator.

They have all been changed to console\_log(“string”) which is a new function that simply takes the argument and outputs it through console.log( ).

Thus, console.log appears in only one place.

Again, the HTML usual commands for display of fields and elements have been converted from:

document.getElementById(“id name”).innerHTML = “string”

document.getElementById(“id name”).style.display = “none” | “block”

have been replaced by the new functions

GUI(id name, string)

GUIstyle(id name, x)

Where this has not been possible, ie reading from an element, effort has been made to ensure that these lines of code are not in the execution stream if use\_html = false.

**CONCISE URL LINE**

**<path>/DUALuse.html defaults to GUI-HTML variant**

**<path>/DUALuse.html?use\_html=true use GUI-HTML**

**<path>/DUALuse.html?use\_html=false no-GUI node.js variant**

**Need node.js, node-start.bat, JSON**