CS1210 Computer Science I: Foundations
Homework 2: Discrete Event Simulation, Revised
Due Friday, November 17 at 11:59PM

Introduction

In this assignment, we're going to be making a slate of changes to the solution to our discrete even simulator from HW1. Like HW1, you'll be given a template of sorts, but unlike HW1 you'll have broad leeway to implement your code as you see fit. In fact, the template I am providing you is simply a slight modification of my original solution to HW1, with some added tweaks and hints and there.

The changes to the simulator are in two categories. First, there are changes to the code in the simulator designed to make the simulation more useful, more efficient, or just easier to use. Second, there are changes to the pandemic model that improve the simulation's ability to model a particular disease. You should work through each of these changes (hint: probably best to do so in order), saving your code as you go.

Part 1: Changes to the Simulator

We're going to make three changes to the simulator. The first will change the way parameters are passed to the sim() function. A quick look at the HW1 solution shows that sim() has a pretty large number of parameters. As we make it more complicated, this number is only likely to increase, which makes for a terribly messy function signature, even with liberal use of default values. The second change is the ability to improve the repeatability of the simulation. Because the simulator is nondeterministic, each time you call it the results will vary, if only a little bit (which is, after all, exactly the desired behavior). Unfortunately, this feature can make debugging very difficult. The third change is relatively small but can provide a performance boost for larger N: not enough to affect the runtime analysis, but real time also matters in the real world.

The first change is to redfine sim() to have the following signature:

```
def sim(cfile='hw2.cfg'):
```

giving the file name from which your revised version of sim() will need to fetch its parameters. I suggest you write a new function, readConfig(cfile), that opens the specified file, reads and parses the parameters in the file, and returns a dictionary of parameter names and values. Note that I have imported some functions and parameters from the os module to facilitate accessing files on disk; see the Python documentation for more information.

For example, if a simple configuration file looks like:

```
# Simulator configuration
max: 100
verbose: False

# Simulation parameters
N: 100
I: 1

# Agent parameters
m: 4
vp: 0.9
tpe: 0.01
tpi: 0.02
```

```
rp: 0.5

# Disease parameters
de: 3
di: 5
```

after parsing which, your readConfig() function should then return:

```
{'N': 100, 'I': 1, 'm': 4, 'max': 100, 'verbose': False, 'de': 3, 'di': 5,
 'tpe': 0.01, 'tpi': 0.02, 'vp': 0.9, 'rp': 0.5}
```

which should also incidentally be the default value returned if the specified configuration file is not found. Your configuration file parser should skip blank lines or lines starting with '#', and should similarly flush any portion of a line following a '#'. It should also properly handle attribute values that are int, float, or Boolean. Here, you will note that we have split HW1's 'tp' parameter (originally a tuple of two probabilities) into two separate parameters, 'tpe' and 'tpi', representing the transmission probability in the exposed and infected states, respectively. And while this dictionary should be the current default value, note that the default should gently change as you incorporate additional changes later in this homework and the next.

You will need to make a number of changes to accomodate the configuration value dictionary returned by readConfig(). First, you should probably adjust the signatures of your existing functions to receive the configuration dictionary as opposed to any parameters that were originally passed to sim() directly (see, for example, newPop() and update()). Second, we will adopt the convention that the values in the dictionary should not change during the simulation. By enforcing the idea that the configuration read from the configuration file remains immutable throughout the simulation (even though the data structure that holds it is mutable — not the same thing), we avoid the risk that is often associated with "promiscuous" global variables. After all, if we pass the configuration dictionary around enough, its very much like having global variables! So while my original HW1 solution used, for example, max and I, you'll note the modified HW2 template version does not.

The second change, then, is to add a seed parameter to the configuration file, by which we can specify (and then set) an integer seed for the random number generator immediately after reading the configuration file. A seed is a number used to initialize a pseudorandom number generator (which is what Python and other programming languages use to generate random numbers). Because of the nature of pseudorandom number generating algorithms, providing a seed ensures subsequent "random" coin flips will follow exactly the same predictable sequence, meaning successive runs of the simulator will be identical. Consult the Python documentation.

The third change enhances the efficiency of the sim() function by making a small but useful change to update(), newPop() and sim(). The insight is that the main loop of sim() steps through all N agents looking for those who are infectious — often a number much smaller than N — to process possible transmission events. An alternative approach is to have sim() dynamically maintain a set of infectious agent ids (*i.e.*, an index into the pop list), which would make transmission modeling step more efficient by directly stepping thorough only the set of infectious agents. Thinking this through entirely, we see that we would have to modify newPop() to return both pop and an initial set of infectious agent id's. The set of infecteds should subsequently be passed to update() for modification, and also modified in the main loop of sim() accordingly. Note that you may struggle a bit with updating sets inside loops that iterate over elements of that set: you will have to devise simple/clean workarounds.

Part 2: Changes to the Agent Model

Here, we want to make three changes that make the simulation more realistic. First, we want to change the meaning of the vaccination probability, vp. Second, we want to model self-isolation, when an agent stays home because they're sick. And, third, we want to model the use of face masks, which affect both the agent wearing the mask as well as the agents they may come into contact with.

In the HW1 solution, vp governs whether the agent in question receives a vaccine prior to the start of the simulation. The vaccine is assumed to be perfectly effective, meaning once vaccinated, the agent is 100% immune to the disease. Here, if the agent is vaccinated (again with probability vp), we will assign a random vaccine effectiveness value to each agent at creation time. This means our newPop() function should now return something that looks like:

```
({'state': -1, 'vaccine': 0.0},
 {'state': 9, 'vaccine': 0.12990310244006897},
 {'state': -1, 'vaccine': 0.6693354025692533},
 {'state': -1, 'vaccine': 0.29464171141226736},
 {'state': -1, 'vaccine': 0.0},
 {'state': -1, 'vaccine': 0.04224554060542318},
 {'state': -1, 'vaccine': 0.42596710495642565},
 {'state': -1, 'vaccine': 0.0})
```

if, for example, agents 0, 4 and 7 opted out of vaccination. It also means that susceptible(), the helper function which figures in the main infection loop, should now depend on flipping a random coin based on the effectiveness of a particular agent's past vaccination.

The second change here is to model an infected agent's tendancy to socially isolate once symptoms emerge. You may recall that people become symptomatic — that is, display symptoms and become aware of their illness — when they transition from exposed to infected. But some agents remain asymptomatic; that is, they never develop symptoms and remain unaware of the fact that they are infecting others.

To model this behavior, we will introduce two more parameters for our configuration file (see Part 1 above):

```
ip: 0.0     # isolation probability
ap: 0.0     # asymptomatic probability
```

which work as follows. When an agent transitions from exposed to infected, we add a new value, 'sociso', a random number between 0 and 1 that reflects that agent's social isolation choices. We set the sociso value as follows. First, we flip a weighted coin with probability ap; if the value is true, then the sociso value is set to 1, meaning no change in agent behavior. If the value is false, we flip another weighted coin this time with probability ip: the result is set as the social isolation for this particular agent, and reflects the extent to which that agent engages in their usual mixing behavior while infected. A sociso value of 0 means the individual agent interacts with 0% of the m agents they normally interact with while infected (total self isolation), while an isolation factor of 1 means the individual agent doesn't change their behavior at all. The main infection loop should be modified to treat the agent's sociso factor much like the tpe or tpi, that is, as an additional condition that must be satisfied in order for disease transmission to occur.

Finally, the third change is to add masking. Masking is a little like vaccination, in that we can model agents having a certain probability of adhering to masking just like vaccination. We'll use the masking probability, mp, in the configuration file in a manner directly analogous to how we used the vaccination probability, vp, to indicate whether the agent in question wears a mask. Depending on the outcome of a coin flip at agent creation time, we then establish that specific agent's probability (stored away as the value of the key ’mask’ in the agent description) that they are actually wearing a mask at any given transmission event. Finally, you will need to modify the main infection loop to account for the mask status of both the infected and susceptible agents.[1]

[1] Be aware that all of these new features will likely reduce the overall probability of transmission, so you may need to adjust tpe and tpi in order to kick the simulator into activity.