Problem Set: Simulating an infectious process

Stuart M. Shieber

April 7, 2023

Imagine an infection among a population of people where the infectious agent is transmitted from infected people to susceptible people nearby. The time course of such a process depends on many factors: How infectious is the agent? How much mixing is there of the population? How nearby must people get to be subject to infection? How long does recovery take? Is immunity conferred? For how long?

To get a sense of how such factors affect the overall course of the infection, we can simulate the process, with configurable parameters to control these and other aspects of the simulation.

1 The simulation

In this simulation, a population of people can be in one of several states:

- Susceptible – The person has not been infected or has been infected but is no longer immune.
- Infected – The person is infected and is therefore infectious and can pass the infection on to susceptibles nearby.
- Recovered – The person was infected but recovered and has immunity from further infection for a period of time.
- Deceased – The person was infected but did not recover.

(In the field of epidemiology, this kind of simulation is known as an SIRD model for obvious reasons.)

The simulation proceeds through a series of time steps. At each time step members of the population move on a two-dimensional grid to nearby squares. (How far they move – how many squares in each direction – is a configurable parameter.) Each person's status updates after they've moved. A susceptible person in the vicinity of infecteds may become infected. (This depends on how large a vicinity is considered to be "nearby" and how infectious each of the people in that vicinity are.) An infected person after a certain number of time steps may recover or die. (The relative proportion depends on a mortality parameter.) A recovered person after a certain number of time steps may lose immunity, becoming susceptible again.

To get a sense of how the simulation looks, you can view a video demonstration.

2 *The simulator*

We've provided you the basics of such a simulator, which you will augment to be able to experiment with a wide range of scenarios.

The simulation can be visualized by showing the locations of the people, color-coded as to their status. Figure 1 shows a visualization of a simulation after some simulated time has elapsed. Susceptibles are shown as small blue circles, infecteds as red, recovereds as gray, and deceaseds as light gray "x" shapes. The radius around infecteds where they can infect others is marked with a thin red circle as well.

The visualization also provides a summary chart that shows the proportion of the population in the different statuses over time. Figure 1 shows the full visualization containing the map and summary chart, and Figure 2 shows the visualization at the conclusion of the simulated run. In this particular scenario, by the end of the run, the infection had been eradicated.

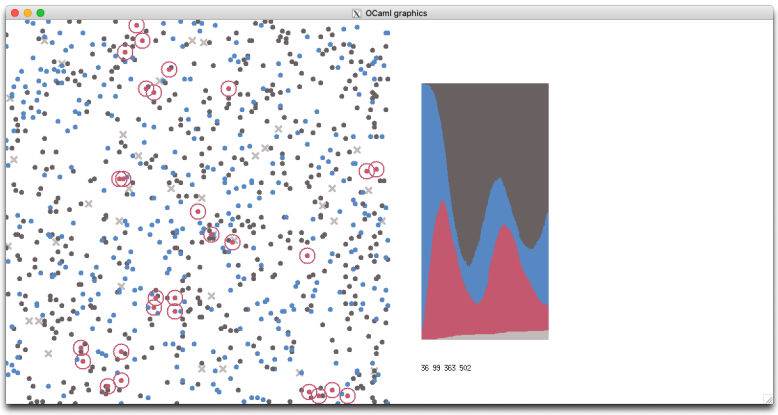

Figure 1: A visualization of an infection spreading through a population. On the left is a map of the population, color-coded by infection status. Susceptibles are shown as small blue circles, infecteds as red, recovereds as gray, and deceaseds as light gray "x" shapes. The radius around infecteds where they can infect others is marked with a thin red circle as well. On the right is a stacked bar chart showing the proportion of the population in different statuses over time. This snapshot was about 40 percent through the simulation.

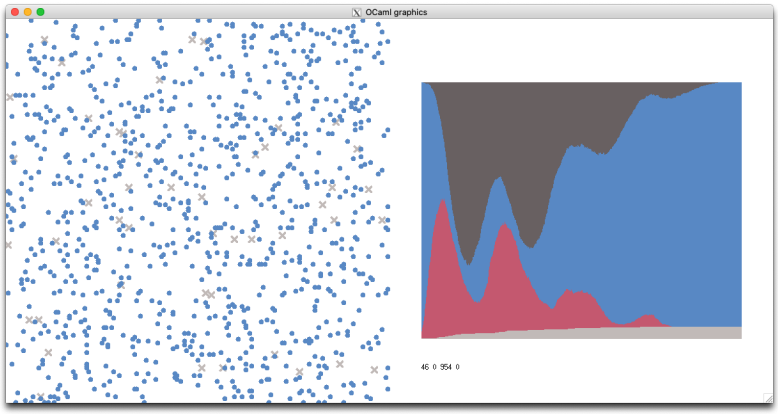

Figure 2: The visualization after the simulation has concluded.

The simulator is made up of several files, most of which you will not need to modify. Those that you will be augmenting are given in italics.

- `utilities.ml` – Some generic utilities, such as functions to sample from distributions, to flip weighted coins, to clip values, and the like.
- *`counter.ml` – Object-oriented counters that keep track of a running total that can be incremented or decremented.*
- `statistics.ml` – A set of counters to keep track of the number of people in the population in various states.
- `registry.ml` – Defines a class type `thing_type` for things in the world that participate in the simulation, and a module `Registry` for storing a population of objects of this type to allow for easy access to the objects.
- *`people.ml` – Defines a `person` object, and provides the beginnings of a set of subclasses for people of different statuses. In particular, it provides a full implementation of `susceptible` people and a partial implementation of `infected` people, but no implementation of `recovered` or `deceased` people. This is the main file that you will be augmenting.*
- `visualization.ml` – Provides all of the code for generating visualizations of the time course of the infection.
- `simulation.ml` – Runs a simulation of an infectious process for a population for a fixed number of time steps, generating the visualization as the simulation proceeds.
- *`config.ml` – Provides configurable parameters that govern all of the details of the simulation and its visualization. You can experiment with different scenarios by adjusting these parameters.*
- `run.ml` – Runs the simulation.

3 Implementing the simulation

You'll want to start by familiarizing yourself with the code by reading through it. Figure 3 depicts a graph of the primary dependencies among the files. The graph shows, for instance, that the `people` file primarily makes use of `registry` and `visualization`. It makes sense to start at the top of this graph and work your way down to get a sense of the overall structure of the code, even though you'll only be modifying a few files. You'll want to look through the file `config.ml` to get a sense

of the configurable parameters that you may want to use in completing the simulation.

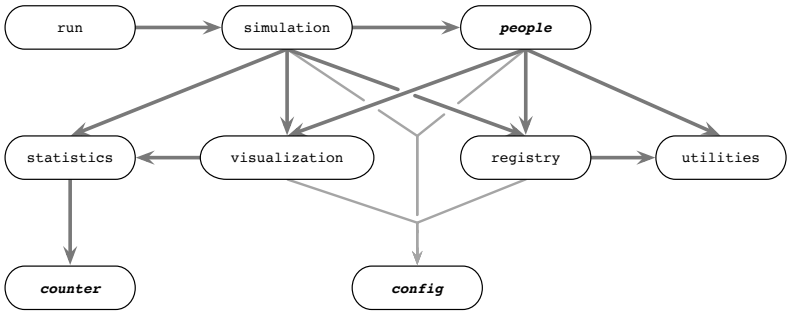

Figure 3: The files implementing the simulation, with arrows indicating which files make substantial use of which others.

Problem 1

By way of a warm-up exercise, you should implement the `counter` class in `counter.ml`. This simple class is used to build some counters for maintaining statistics of how many people are in each of the possible statuses and for tracking the time-steps in the simulation. You can see how these counters are established in `statistics.ml`.

Once the `counter` class is working, the system should already be able to be compiled and run. You can

```
% ocamlbuild run.byte
```

and then

```
% ./run.byte
```

to see the simulation in action. However, since only the two statuses of susceptible and infected are implemented, and the latter only partially, the simulation will not be complete and its visualization won't conform to the desired one.

Problem 2

Complete the implementation of the `draw` method for the `infected` class so that infecteds show up as red circles with a radius marker as in Figure 1.

Next, you'll complete the implementation of the `infected` class by allowing for infecteds to recover after a certain number of time steps. The number of time steps required before recovery should be determined by sampling from a Gaussian distribution with mean and standard deviation given by the parameter `cRECOVERY_PERIOD` from the file `config.ml`. (This sounds more difficult than it is. We've provided a Gaussian sampler in the `utilities.ml` file. You can call it to get the sample, convert it to an integer, and store it somewhere appropriate. Then at each update, you'll decrement it until it reaches zero, at which time the infected has recovered.)

Problem 3

Implement the `update` method for the `infected` class. It should check to see if the infected has recovered, and if so, it should replace the `infected` object in the registry with a `recovered` object. (The process is similar to how the `susceptible` object is replaced with an `infected` object in the `susceptible` class we've provided.)

In order to complete this change, you'll need a `recovered` class.

Problem 4

Add a `recovered` class, in addition to the `susceptible` and `infected` classes. Objects in this class should have an immunity period, again sampled from an appropriate Gaussian distribution (using the parameter `cIMMUNITY_PERIOD`). After the immunity period is over, recovereds become susceptible again.

You now have a full simulation of the infection process, with people cycling through from susceptible to infected to recovered to susceptible again. Use the opportunity to start experimenting with the various configuration parameters in `config.ml`. What happens if you decrease the neighbor radius (`cNEIGHBOR_RADIUS`), which is akin to "social distancing"? What happens if you increase the step size (`cSTEP_SIZE_SUSCEPTIBLE`), which might be thought of as modeling increased traveling. What happens if you greatly increase the immunity period? Try out different scenarios and see what happens.

Finally, and perhaps most dramatically, the infection might have an additional outcome, by virtue of its mortality.

Problem 5

Add a `deceased` class. In the `infected` class `update` method, after the infection period is over, a proportion of people (governed by `cMORTALITY`) will become deceased rather than recovered.

4 Exploration

Once you've got this all working, try out different scenarios. See how parameters affect the results.

Feel free to augment the implementation. Here are some possibilities, but you can certainly come up with your own.

- You might add a notion of "central quarantining". After a certain number of time steps have elapsed, at each time step thereafter a small proportion of infecteds might change their properties (if not their object class) to become quarantined. They move to a central location (say, the middle of the grid) and their movement is restricted by resetting their step size to zero and their infectiousness to zero. When they recover, they move back to their location before the quarantine.
- You might add a small number of locations on the map – "grocery stores" we'll call them – where people tend to congregate. Every time step, a small proportion of people are relocated to the grocery stores for a few time steps before returning back to where they came from.
- You might establish a maximum capacity for treating infecteds such that when there are more infecteds than the capacity, mortality

increases. It would then become more important to "flatten the curve". Can you adjust parameters to do so? If so, which parameters work best?

Problem 6

To present your experiments, we ask that you make a short video of perhaps three to five minutes presenting some scenarios that you've looked at. If they involve extensions of the sort above, all the better.

To make the recording, you can use whatever video or screen-recording system you prefer, but a simple one-person Zoom session using Zoom's built-in local recording feature should be sufficient. To submit the recording, redefine the variable `recording_url` in the `counter.ml` file to return the URL where your video recording can be accessed (at Youtube or Vimeo for instance), for example,

```
let recording_url () : string = "https://url.cs51.io/video-ps8" ;;
```

Version information: Commit 3799084 from 2023-04-07 by Stuart Shieber. CI build of I_TAG ().