Modelling Epidemics

Michael McEllin April 2020

As I write this, we are locked-down in the midst of Covid-19 epidemic and like many others I am aware than our Government is having to make difficult decisions about how many resources the NHS will need to cope with those who will fall ill as the epidemic reaches its peak, which restrictions it will need to keep going and when it might be safe to release people from some of the movement restrictions. The correct decisions depend on things that will happen days and weeks from now, so It desperately needs to predict the future with rather more reliability than guesswork.

The Government is currently receiving a lot of advice from experts who specialise in modelling epidemics with mathematics and computers. So, what is a mathematical model? How do we run it on a computer? What can we hope to learn from it? (How much can it really tell us about the future progress of the epidemic?) Can we learn to do some of this modelling for ourselves?

I am assuming at this point that you have some basic knowledge of at least one computer language, and that you are now prepared to develop a moderately challenging program doing a real job of work. Below, I suggest two different types of approach: one starts with mathematical analysis but resorts to numerical solution to solve otherwise difficult equations; the other focusses on producing a computing simulation that in some ways is more closely related to real-world disease transmission mechanisms.

Both ought to be well within the capabilities on anyone studying maths and computing at A level - but do not expect it to be easy. Nor do I intend to give you a complete set of instructions to follow. You must work out most of it for yourself. The problem is open-ended in the sense that having developed the model it would be possible to use it to study epidemic progress under different assumptions, and you could also increase the sophistication of models beyond the basic framework discussed here.

I do, however, provide more information in this brief than I would normally supply for the initial phase a CREST project. Outside lock-down, I would expect to have face-to-face meetings with student teams and would usually be feeding additional material to participants at the regular mentoring sessions.

This is a project that would be suitable for submission for the **British Science Association "Gold" CREST award** - see https://www.crestawards.org/crest-gold . It could be undertaken by an individual student or a small team.

What is a Mathematical Model?

Models are representations of the real world. In some sense they behave and look like a *selected part o*f the real world usually for a *specific purpose*. For example, one model aircraft may be designed to fly - but does not actually look like any real, full-sized aircraft. Other aircraft models, intended for display, may not fly at all, but are intended to *look* very like the real thing - just on a much small scale.

A *mathematical model* is a collection of equations (it may be just one equation, or, in the case complicated applications like weather forecasting, thousands.) The equations are there to constrain the relationships between the *variables* in the equations. Some of the variables will represent things that we are able to feed into the equation, such as - in the case of epidemic modelling - the number of known infections at a certain time (such as *now*), the number of interactions between typical people, the chance they will pass on an infection and so on. Other

variables will be outputs from the solution of the equations, such as the number of infections expected at some specific time in the future.

Epidemic prediction is an example of an important type of model: one of the things we had to feed in was the number of infections we knew about now, so we could predict what we might see next week. In particular, the rate at which the number of infections is increasing depends on the number of current infections. In principle, when we get a situations like this where a rate of increase (or decrease) depends on the current value we know all we need to know and can normally evolve the equations through time - though it may be complicated. In this case we have a type of equation known as *a first order differential equation within initial boundary conditions.*

Sometimes we can solve our mathematical models with a pencil and paper. When things get really complicated and messy, however, this is usually not possible. Computers can let us deal with complicated situations because we can represent the messy details as just more equations, and we let the computer look for the output variables that are solve the equations for the given input variables. You should not get the idea that this is now easy: far from it. It is, however, often possible to make some progress, whereas solving the maths with pencil and paper would be completely impossible.

We are going to try to set up some simple mathematical models and then see whether we can make progress by solving them on a computer.

I hope that we can learn something about the extent to which we can trust computer models to tell us something about the future. They are powerful tools - but powerful tools can injure as well as help.

I am going to talk about two different approaches to epidemic modelling. The first starts from a mathematical equation which we then try to solve with a computer. The second jumps straight in with a more purely computing attempt to represent the real world. They both have their place when trying to predict the uncertain future.

This brief contains more information than I would normally give, because in more normal circumstances I would deliver some in face-to-face meetings, rather than in lock-down.

A Very Basic Mathematical Model

We often start our modelling process by thinking about the simplest model that illuminates some of the mechanisms that might be operating. There is a fair chance that such models are far too simple to be a good representation of the real world, but by thinking about them, we can learn why they are going wrong and therefore what features we need to add.

- We might start by considering a *population* which is, for the purpose of this study, isolated from external contacts (and we will ignore births and deaths or indeed any changes due to causes other than the infection we are studying). Let us say we have **p** individuals.
- At some later time, **t**, when the epidemic is running, we will be able to divide the population into those that are as yet uninfected (i.e. susceptible), s(t) and those who have been infected, which we will call the *cumulative infections*, of which there are a number **c(t)**. Some of these might have an active infection and are able to infect others, less us assume that there are a number **e(t)** of these.
  - We can, of course, immediately write the constraint $p = s(t) + c(t)$, where I am not as yet thinking about any reduction in p because of those who die of the infection. We are including the deceased within the number p.

- Although we are adding members to the *infectious population*, **e(t)**, we must also remember that as people either die or get cured, they leave this population. If we assume that as the disease runs it course individuals are potentially able to infect others for a time T after they are infected[1], then the cumulative number of people where the disease has run its course (either to cure or to death) is the cumulative number of infections a time, T, ago. So those who are still infection must just be $e(t) = c(t) - c(t - T)$.
- How do these numbers change with time. We will assume (this is the crucial modelling assumption - watch carefully!) that the rate of which infections can be passed on will be proportional to the the infected population and also to the susceptible population, and we will need to put in some constant representing the probability that one person can infect others. So, in a time interval dt, we might see an increase in cumulative infections dc:

  $$dc = \alpha \,.\, s(t) \,.\, e(t) dt \,.\, dt$$

  were α must be chosen to correctly represent the likelihood that one infected person will produce new infections. We will probably have to derive a value for this by fitting our solution to graphs from the early stages of the epidemic. In the limit of small dt:

  $$\frac{dc}{dt} = \alpha \,.\, s(t) \,.\, e(t)$$

  but the susceptible population is decreasing as more people get infected because $p = s(t) + c(t)$ so:

  $$\frac{dc}{dt} = \alpha \,.\, (p - c(t)) \,.\, e(t)$$

  or, substituting for e(t):

  $$\frac{dc}{dt} = \alpha \,.\, (p - c(t)) \,.\, (c(t) - c(t - T))$$

  This is a first order differential equation which deals only in the cumulative number of infections, but things have got a bit complicated because of that c(t-T) term - representing the time delay between infection and progressing out of the infectious group. In fact, there is no easy analytical solution that I can find, and though I suspect that some advanced solution techniques might get me to a reasonably good approximate solution in closed form, this is well beyond A-level maths. All we can do now is solve numerically on a computer.

Before we go further it is worth just looking at the equation and trying to understand what the solution might look like before we try to solve it. The normal course of an epidemic is that the infections increase very rapidly at the start, but as more and more become infected and eventually immune, there are fewer and fewer susceptible people to be infected and the infection rate must drop off.

---

[1] This time might be the time that individuals are asymptomatic and before they are isolated, but we may also have to take account, to some extent of those who are diagnosed passing on the virus, even though those around them are taking precautions. The value of T is clearly going to be important - but we probably have rather poor information on what it should actually be.

- The term $(p - c(t))$ is clearly going to shut down the infection rate for very large times, as $c(t)$ starts to become comparable to the total population, p.
- The early exponential growth will be driven by the second bracket $(c(t) - c(t - T))$. This is not so obvious because of the second term in the bracket. But just imagine the first few days where t<T, so $c(t - T)$ is zero and $c(t)$ is much smaller than p. The equation can now be approximated by:

$$\frac{dc}{dt} = \alpha \,.\, p \,.\, c(t)$$

  which is trivial to integrate to $c(t) = c(0)e^{\alpha pt}$, which is our expected exponential growth in the early phase.
- We will need to start off our epidemic by introducing one case at time t=0.

Maybe we should also have a quick think about the ways that our model might not be a very good representation of the world - and maybe the ways in which it might well tell us something worth knowing. Here are a few on my thoughts: I am sure you can produce some more:

- It implicitly assumes that everyone in the population has a chance of interacting with everyone else. We are all in one box. In reality this is not the case. People live in different cities, in smaller towns and in country areas. The chance of meeting others who have the virus in these cases may be quite different. Young people also behave differently to older people. Children gather in schools. Older people often gather in churches. 20-somethings gather in clubs, pubs and parties.
- We are also assuming that people continue to behave in the same way even as those around them are falling ill. It is likely that behaviour would change, even if we were not subject to Government regulations limiting movement and contacts. In fact, we KNOW that behaviour has changed: when Covid-19 first started to run infections were rising very rapidly (that is, a high value of α) but governments around the World did the bests they could to reduce rates of transmission, so α is now much smaller than it otherwise would be.
- So, on the other hand, the model is clearly telling us about the ways we might attempt to control the spread of the virus. In fact, we have only three levers: firstly we might try to reduce the general amount of contact in the population (reduce α); secondly we need to identify those who are possibly becoming infectious as soon as possible and isolate them; thirdly, we might hope to increase the general immunity in the population before they become ill with a vaccine - if, *big if*, one is available. These deduction from the model are worth thinking about because they will probably carry over to more sophisticated modelling approaches. Those might produce more accurate predictions of numbers falling ill, but they cannot change the fundamental mechanisms behind the epidemic.

Some of the problems we can fix, perhaps by making our model more complicated - perhaps trying to represent the different ways young and old people behave, for example. There is, however, no point in make the model more complicated if we simply do not have the data that describes the differences between the behaviour of young and old people and how the groups interact with each other. If we put bad data into these more complex models we might even get highly misleading predictions - much worse than those from

simpler models. Furthermore, the more complicated we make the model, the more likely we are to make mistakes when we try to program numerical solutions.

So, my model is probably *much* too simple, but we can use it to learn about models.

Numerical Approaches

Do not get the idea that solving equations on a computer is always easier than with pencil and paper. There are many pitfalls and it is quite possible to get entirely misleading results without realising that something is not right.

There are several ways that things can go wrong. We have already talked about one of them: our mathematical model may just be too simple to really represent the world at all well. Furthermore, as we will see, we are not *actually* solving the equations we have written down above because, as we will see, we cannot represent the completely smooth, continuous behaviour described by these *analytical* equations on a computer. Instead we *actually* solve *discrete* equations that, for example, move time forwards in little jumps (perhaps a day at a time). For some problems this difference does not matter all that much, for others it matters a great deal…and the type of problems where it matters most are the ones where there is likely to be some kind of exponential growth, because, it turns out, they also tend to be quite good at amplifying the small differences between our discrete numerical model and the continuous analytical model. Epidemics usually have a phase of exponential growth so we must expect trouble, and learn to employ the more sophisticated numerical techniques that are able to control the growth of errors.

We therefore always need to take our model through a process of *validation* in order to prove that it works well enough for our particular purpose. Most of those who develop this type of computer modelling professionally would probably agree that about 90% of the total effort in producing a model you actually believe is in the validation, and only 10% in building the software.

In order to validate our model we will need data from the real world and it may be that the only place we can get it is from the early phase of an epidemic. Even if the epidemic has been running in other countries before it reaches us, we may need to be a little circumspect about the data we can get: different cultures behave differently, so the experience of China, or Italy may not be as relevant to us as we might hope.

The other type of problem ought to be less serious: we may introduce errors into the program (“bugs”) because we are not very good at turning the intended algorithm into working code. In reality, of course, bugs are far more common than programmers would sometimes like to believe. Well trained professional programmers doing computer modelling probably leave about 10 mistakes in every 1000 lines of code that they write (and believe that they have fully tested). This is a somewhat sobering thought, but I have found that it is fully born out by my own experience on large scale software development. Even worse, novice programmers can easily be 10 times worse![2]

---

[2] Far more worrying is that some of the least capable programmers have the highest opinions of their own abilities. This explains a lot of very poor software that you come across. Fortunately, when getting things right really matters (such as when designing fly-by-wire control systems for aircraft) there are sophisticated highly technical methods for pretty much getting rid of all errors. These are, however, expensive and require very highly trained software developers, who check each others work all the time.

The process of confirming that we have produced a program that runs the algorithms we intended to implement is called *verification*. Part of this is just looking at the code and seeing if you can actually fully understand what it is supposed to be doing. (Best of all is getting *someone else* to do this.) The rest is *testing*: running our program on cases (probably somewhat simplified cases) where we know the right answer and can compare what we calculate with what we expect. In my very extensive experience (37 years working on very complicated computer models of nuclear reactors which *had* to be right!) novice programmers do not do anything like enough testing, and quite often do not fully understand the code that they have just written. It was my job to check, so I really do know this! So be warned! Do not trust yourself until it can be *proved* that you have not made a mistake.

People who are very good at mathematical and computer modelling are in high demand and therefore well paid in lots of employment areas, from aerospace to merchant banks.

The first thing we might try is the simplest approach to turning our model from a continuous analytical model into a discrete numerical model. We say we know the cumulative number of infections at some time, t_i, (that is, the ith time step after we start) and then calculate the number at a time $t_i + dt$ (where dt is *not* infinitesimal, but is, perhaps, one day long) as:

$$dc_i = \alpha \,.\, (p - c(t_i)) \,.\, (c(t_i) - c(t_i - n \,.\, dt)) \,.\, dt$$

Note that we have replaced T by n.dt where n is some input parameter chosen such that T=n.dt is the time in which people are infectious.

Then, the cumulative number of infections at time $t_{i+1} = t_i + dt$ is just

$$c(t_{i+1}) = c(t_i) + dc_i$$

and then we make another step forward using $c(t_{i+1})$ instead of $c(t_i)$ and continue doing this for as long as required. (This is called *iteration*.)

Try it. The results may “blow up”. (The technical term is *numerical instability*.) That is the infection numbers will probably increase very rapidly indeed, may go up and down by large amounts, and probably not look anything like what we expect. That is because during the time interval dt we have been assuming that the rate of growth is determined by conditions at the beginning of the time step, when if fact these conditions should actually be changing.

There are two ways of dealing with this type of problem. The first is by making the time step smaller - sometimes much smaller. We may find this controls the growth of errors until the model starts to decrease the rate of infections, and that also suppresses the grow of errors[3]. We can only discover the right size of time step by trying smaller and smaller values until the predictions made by the model (e.g. the total number of infections or the shape of the infection rate curve) no longer change by a significant amount. You should therefore experiment with this and plot a number of graphs.

[3] In fact, I found that I could choose an explicit time step to be small enough for stability.

Another common way of dealing with the problem is the use of so-called *implicit* time stepping which however, makes the algorithm quite a bit more complicated. This is we say:

$$dc_i = \alpha \,.\, (p - c(t_{i+1})) \,.\, (c(t_{i+1}) - c(t_{i+1} - n \,.\, dt)) \,.\, dt$$

That is the rate of growth depends on the number of infections one step into the future. Hang on! Surely we do not know $c(t_{i+1})$ until we have done this calculation step! This is only partly true because wherever we see $c(t_{i+1})$ we replace it with $c(t_i) + dc_i$ and, admittedly, get a more complicated equation that has only $c(t_i)$ and dc_i though, unfortunately not in a simple closed form. We are going to have to solve a quadratic equation to get dc_i in terms of $c(t_i)$. This is an exercise for you.

I am not going to explain why this usually works. You can look it up (and I recommend that you do). It often does, however, make the integration numerically stable (though it is not guaranteed) and in fact is far more commonly used in practice than the *explicit* time stepping that we first used. That still does not mean that it produces values that look like the real solution: that still depends on using time steps that are sufficiently small. Big time steps may be stable but they tend to smooth out real time variations in the solution.

How will we know if they are “sufficiently small”, As with the explicit time stepping, we keep trying the process with smaller and smaller time steps until we see no noticeable change in behaviour.

Even if you find that you can produce a stable solution with explicit time-stepping you should also experiment with the implicit time-stepping in order to understand this very important numerical method and appreciate its advantages.

Sources of Data

There are many sources of data on the Internet. I found several easily.

- https://data.europa.eu/euodp/en/data/dataset/covid-19-coronavirus-data
- https://covid19.healthdata.org/united-kingdom
- https://www.sciencedirect.com/science/article/pii/S1755436518300306

The last citation is particularly interesting because it describes the science behind a BBC citizen science “pandemic” experiment. Highly relevant material!

Producing the Program

I am not going to tell you which programming language to use. It does not really matter. You will be using the same algorithm and it should produce the same answers. You can even use a spreadsheet, such as Excel, though I certainly do not recommend it[4].

---

[4] In my experience, which I remind you is extensive, you are *far* more likely to make programming mistakes when creating spreadsheets with moderately complicated algorithms than when writing in, say, Python. It is also harder to find exactly where you have made a mistake even when you know there is one to find somewhere. I do use spreadsheets sometimes, but only when I need to do the simplest of calculations and then draw a few graphs.

If you are already familiar with a computer language, use that one. For those who want to learn a new language, I find that Python is a very good for this type of relatively straightforward numerical experiment. It has the advantage that one can pass the results straight through to a graph-plotting library, such at *matplotlib*, that can be called straight from your simulation.

Your project is first of all to create a working and verified program. (That is you have carefully check the code and created a number of test cases that prove that it works.) This will not be easy because you do not know what the answer should look like, so how do you know your results are correct.

You have probably been taught the right approach at anytime since you started the GCSE course. *You check your solution works by putting it back into the equation*. This will involve a little more programming, but it is worth it.

What we should have got out of our simulation is a list of values of the cumulative number of infections at equal time intervals, so we have $c(t_i), \quad for \quad i = 1....N$ where N is some large number of days. We can therefore *approximate* dc/dt at any of these points by writing:

$$\frac{dc}{dt} \approx \frac{c(t_{i+1}) - c(t_{i-1})}{2.dt}$$

Here, were are just drawing a notional gradient line between points either side of the point where we want the gradient. Now put this back into our original equation, along with the values of $c(t_i)$ and $c(t_i - T)$ and our other terms and see if everything more-or-less cancels. If it does not, either we are using a tilmestep that it too large, or we have got something badly wrong.

I would also look carefully to see if the early behaviour was the type of exponential increase predicted by the considerations I have already outlined, and that the large-time behaviour was shutting down as the total infections approached the population size.

I rarely leave my program at the point where I get it working. For my first implementations I choose the simplest algorithm that I can invent, which is usually the one that where I am least likely to have made mistakes. After that point, I think about ways to improve the efficiency of the calculation and ways that I can make the operation of the algorithm even more transparent to the code reader. I always remember an excellent bit of advice from a great computer scientist C.A.R. Hoare, a professor at Oxford:

> *There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.*

You should aim to make your programs obviously free from errors.

Further Investigations

Now explore what happens when you change α and T, so you can start to understand the way your model behaves.

What we really need to do now is to obtain some real-World data about the Covid-19 epidemic. There is a lot of it around, including data from the World Health Organisation and Public Health England, but you will have to search for it.

Since we do not know the best values for our controlling parameters, α and T, we have to use some early-stage data from the epidemic to see if we can fit the graphs and derive the best-fit values of α and T. Now we let the model run until later times and see if it predicts the later evolution of infections.

Plot lots of graphs! If you are working in Python, you might choose to use the plotting libraries that can be called directly from your program, such as *matplotlib*. (https://matplotlib.org). You could also print tables of numbers and transfer to Excel for plotting, if you are familiar with Excel graphing. My favourite tool is *gnuplot* (http://www.gnuplot.info) which runs on almost all computer systems, is extremely widely used by scientists and engineers and can produce high quality graphs that of acceptable quality for the editors of professional journals - it is well worth learning if you intend to follow a maths, science or engineering study programme.

Do not be surprised if your simple model is not very good! Try to work out why it might be going wrong.

Look at the sensitivity of your model to the controlling parameters (in our case α and T). You may well find that you can fit the early stage data from the real epidemic fairly well with a range of values of these parameters. The professional modellers look very carefully at this, and then run their predictions with a range of configuration parameters consistent with the fit so they can tell decision makers of the range of outcomes that might occur (and what they can confirm is not likely to occur). Our predictions may well be very sensitive to the assumed value of α. Hence, this is often where we spend quite a lot of time.

The professional modellers would now start adding more sophistication to deal with the areas where they think their simple model is failing to represent the real world, checking at each stage that they are getting better and better at following reality.

In particular, they would look at modelling different groups within the population and how infection moves within groups and between groups. The mathematics behind this, however, is a little more than you would expect to handle at A-level.

A Confession!

In reality few professional computer modellers write their own algorithms to solve differential equations. The need to solve differential equations is so common that experts in numerical analysis have produced extremely sophisticated software libraries that will solve most types of problem very efficiently and reliably. All we do is marshal the data for the solvers in the right format and then try to interpret the outputs correctly.

Python, for example, has the *scipy* and *numpy* libraries (https://www.scipy.org) which would normally be my first port of call if I were working in Python (and they are a very good reason for doing a lot of scientific and engineering programming in Python).

You might like to see if you can find an appropriate subroutine in scipy and check your own simulation. It is, however, still important that you investigate some of the numerical solution methods for yourself. You will always need to understand the strengths and weaknesses of each method and the description of the subroutines in the numerical

libraries may not make much sense if you have not learned something about the underlying technique for yourself.

Now for Something Completely Different

The so-called *analytical* approach we have used up till now is from one point of view a little odd. The actual mechanism of disease transmission in the real world is discrete, moving from person-to-person at a specific time. We have, in contrast, represented the average rate of transmission as a function of time in a differential equation. We then had to go back to a discrete numerical solution to solve that smoothly varying mathematical model. Why cannot we just set up a computer model in which we represent individual people and the actual individual events by which the disease is transmitted?

Clearly, we will not be able to follow the 60+ million people of the UK individually, but perhaps we could follow a sample, representing a typical mix of young, old, those in cities, those in the country and so on? Disease transmission event would occur by some random toss of a notional dice. This approach is entirely respectable and widely used. It is often called a “Montecarlo” modelling because we treat the system as a game of chance.

Remembering the earlier advice about choosing the right way to store information, we will need to have some way of holding information about individuals, where we will need to associated a state, which is one of susceptible, infectious, cured (and immune) and dead. In addition we need to think about representing the proximity of one person with another, because infection tends to go between people who have encounters and these are more likely between those who are more likely to come into contact.

I therefore propose that we should consider an area perhaps representing part of the country where we randomly distribute people over a horizontal rectangular area. They will therefore be at different distances from each other. All but one of the individuals will be susceptible and the odd one will be the infectious “seed” chosen at random.

We should regard the distances between individuals are really representing social distance - though people who are physically widely separated are often widely socially separated. (This connection between physical separation and social separation is something that can be investigated with models of this type. Different social classes living in the same geographical area may hardly mix at all - except that perhaps their children go to some of the same schools.)

We now have a cycle in which we iterate over all the susceptible individuals and ask how close they are to someone who is infectious and we toss a dice to see whether infection can occur, with a probability that depends inversely on distance. If infection occurs we add that individual to the infectious list, together with the time at which they were infected.

On each cycle we also look at each member of the infectious list and check how long they have been on the list. After a certain time they can be moved to cured-and-immune - or with another toss of the dice to deceased.

As before, over time we expect the susceptible individuals to decrease and the immunes to increase, to the rate of infections will at first increase rapidly then die away.

Models of this type are in many ways simple to understand and often not very difficult to program. They are therefore sometimes used to validate more complicated methods, thought they may not be used for routine modelling because of their computational

expense. They often take *much* longer to run than the analytical models because have to look at a *lot* of person-to-person interactions to get representative behaviour. Then, for example, if we double the size of the population we are modelling we immediately quadruple the potential number of interactions we must consider. As a result, if we want quick results we might have to make do with a smaller sample population than we would really wish to deal with, and this means that because we are feeding in random events every time we run the program we will probably get slightly different answers, so we have to regard our results as having a degree of uncertainty.

The natural technique for this type of problem is *object-oriented programming,* which can be used in most modern computer languages such as Python, Java or Javascript. We define the people in the population as members of a *Class* (where the class membership defines the properties each member has - such as the information held about each member). In our case each “person” needs at least the following attributes:

- The state which is one of susceptible/infectious/immune/dead.
- The time of infection, if infected.
- The position on our geographical field.

We are allowed to make arrays of members of a class, so we can have a list of the susceptible, a list of infected and a list of immune or deceased. We can move people from one list to another as their state changes.

We start with a population where one individual is defined to be in the infected state and all the rest are on the susceptible list. Then we proceed in iteration cycles as explained previously.

As with the previous approach, we would need to adjust the controlling parameters to fit some actual epidemic data in its early stages, and then check whether it continues to follow the actual progress in later stages.

There are lots of ways in which we could develop this model in a more sophisticated direction, and in many ways it would be easier to extend this type of model than our previous more mathematical approach.

- We might start with a non-uniform distribution of people, some more concentrated in “cities” others widely spread out. We could then study how disease spreads more rapidly in crowded areas, but sometimes jumps long distances.
- We can divide members of the “people” class into sub-types, such as “young”, “old” etc.

It also would be interesting to program this model with a graphical tool, such as Processing (see http://processing.org) which uses the Java language. Each person could be represented as a circle on a place, with colour showing their state. This would allow an informative visual representation of the way infection spreads.