This exercise rehearses basic steps that can be used to create a model of infection spread. For simplicity, we focus on models where the network is static – i.e. does not change over time.

1. Open the model you built for the exercise “Building a Minimalist Network-Based Model Framework”, or the pre-provided model “MinimalistNetworkABMModel”
2. Save the model to a new model, entitled “MinimalistSIRNetworkABM”. In the “Java package” field, use the lower case version of this name (“minimalistsirnetworkabmmmodel”)
3. Double click on “Person” in the “Project” window. This should open the canvas for the “Person” class, showing the visual specification of a “Person” in this model. Currently, this contains only the visual representation of a person (consisting of an oval and a line).
4. Open the “Model” tab of the “Palette” window
5. Add an “Entry Point” to the “Person” canvas. Note that the procedure for adding this differs by version of AnyLogic. In our version of AnyLogic, this is accomplished by dragging the “Entry Point” from the palette to the canvas

From now on, we’ll assume that you are familiar with the appropriate procedure to add things from the palette.

6. After you add the “Entry Point” to the model, you will see that it has a default name of “statechart”. Type in ‘infectionStatechart’ instead; if you a window appears to ask if you wish to perform refactoring, indicate “no” – this is something only of concern if we’ve already made elements of the model depend on the original name. Please note that this can be changed in the “Palette” window as well as in the Canvas.
7. Add a “State” from the “Palette” window to the model (the canvas for Person). This process involves “dragging” on the canvas. When you add the State, try to position it so that it overlaps with the very end of the “Entry Point”, or at least some of the end segment of the “Entry Point”. Positioning the State in this way will let you automatically connect the State to the Entry Point. As you are dragging the shape across the “Entry Point” in this way, you will know that it will be connected if you see the end of the “Entry Point” turn green (or aqua). See the image below.
8. Click on the entry point, and make sure that the circle at the lower (“pointy”) end of it is coloured green or aqua. If it is not coloured green, drag the circle so that it overlaps with the state, and turns green (aqua). When it is green (aqua), you can release it – AnyLogic may adjust the details of the spacing.

Click on the state that you had added. Note that “Properties” window specifies the “properties” for this state. For “Name”, please enter “Susceptible” (no quotes should be included)

9. Proceeding vertically downward, add two more states from the palette, entitled “Infective” and “Recovered”, respectively. The result of adding these states should look something like the following.

10. From the “Palette” window, add a new “Transition” by first clicking on “Transition” and then clicking and dragging from the Susceptible state to the Infective state; this...
will connect the two. When you add this transition, its properties will appear in the “Properties” menu. Set its properties as follows:

a. In the “Name” field, call this transition “infection”.

b. In the “Triggered by” dropdown, indicate a “Rate”. This indicates that the transition will occur according to a Poisson process, a memoryless process associated with a mean residence time of the reciprocal of the rate.

c. In the rate field, we will specify a constant rate of 0.1 (implying a residence time of \( 1/0.1 = 10 \) time units). Note that while the rate is constant in this case, we can in general have a time-varying rate, although some care is taken to make AnyLogic “know” when the rate has changed.

d. In the “Action” field, for the time being, we’ll have the Person report when they’ve been infected by printing out a message to the “Console” window. For the “Action”, enter

```
traceln(this + " has been infected!")
```

[Note that the double quotes should be included within this fragment of Java]. This reporting message prints the “name” of the agent (“this”) and then follows that by the suffix “has been infected!”.

In drawing this transition, be sure to again check to confirm that the small circles at the end of the arrow are green (aqua) – indicating that both ends of the transition are connected to the appropriate states.

11. From the “Palette” window, add a new “Transition” connecting the Infective and Recovered state. When you add this transition, its properties will appear in the “Properties” menu. Set its properties as follows.

a. In the “Name” field, call this transition “recovery”.

Exercise Rating: Basic
b. In the “Triggered by” dropdown, indicate a “Timeout”. This indicates that the transition will occur exactly a specified residence of time after entering the state.

c. In the rate field, we will specify a timeout of 20.

d. In the “Action” field, for the time being, we’ll have the Person report when they’ve recovered by printing out a message to the Console window. For the “Action”, enter `\trace\ln(this + "has recovered!")’ [Note that the double quotes should be included within this fragment of Java]. This message prints the “name” of the agent (“this”) and then follows that by the suffix “has recovered!”.

In drawing this transition, be sure to again check to confirm that the small circles at the end of the arrow are green (aqua) – indicating that both ends of the transition are connected to the appropriate states. Specifically, beware connections that look like this. Instead, they should look as

```
Try running the model. When you get to the initial model open screen, just press the large button to start the model.

Click on the model exploration menu, that currently includes the text “root:Main”. Pull this down. Your screen should look something like the following:

![Model Exploration Screen]

Click on the item highlighted above (which reads “population[0..99]”).

Leaving the view of the model as a whole, you have now “drilled down” to the level of individuals, see a screen that shows the characteristics for a particular person – including their location within the network (shown by a their node and connections), the values associated with their properties (here, the two parameters you have created above). By clicking on the up-and-down arrows to adjust the numbers shown in the upper left corner (just to the upper right the yellow “tool tip” shown below), you can change the view to show different members of the population. The number itself shows the count of that person in the population. Note that as you proceed to view multiple agents, different

Exercise Rating: Basic
agents will be in different states. This reflects the fact that agents get infected only stochastically – only according to a rate that yields different time in the Susceptible state prior to infection.

Note also that the “Console” window below contains the infection and recovery status reports of the agents.

12. The above model offers virtues to recommend it, but in this case, the textual status reports from agents are somewhat tedious to read. We can make the visual representation encode the state of individuals in many ways. Here, we explore the use of color to encode the state. One way to pursue this task is through a variable that encodes the appropriate color. This approach is pursued in slides from lectures. Here, we will pursue a different approach to help the reader understand some of the many options that afford themselves for implementing this task. Yet another approach would be to use a data structure, such as an “array” or a “Hashmap.” It is arguable as to which of these approaches is simpler, more extensible or easier to understand.

a. From the “Palette” window, add a “Function” to the model. We can define a function whose job it is to tell us the appropriate color for whatever state is active.

b. Set the properties of the function as follows in the “Properties” window:
   i. The name of the function should be “colorForInfectionState”.
   ii. Scroll down to row that starts with the text “Return Type.” Click on the radio button in this row that says “Other” and in the text field that follows it type “Color” (without quotation marks) This indicates that the function can return only identities of Colors.

Exercise Rating: Basic
iii. Click on the “Code” tab of the “Properties” window. The code for the function translates from the identity of the active state in the state chart and turns it into a Color. One way of implementing this code is as follows:

```java
switch (infectionStatechart.getActiveSimpleState()) {
    case Susceptible:
        return GREEN;
    case Infective:
        return RED;
    default: // Recovered
        return GRAY;
}
```

The “Switch” statement “looks” at the value given to it (in parentheses), and then tries to match it to one of a set of values (the values given by the “case”) statements. If the value given to switch matches one of those cases, the appropriate code below the case statement is executed – here, this just returns the appropriate values. If no “case” statement is matched, the statement executes the code in any “default” statement.

Note: In typing the above, you may wish to use a “shortcut” to get the Java code editor to “fill in” suggestions for you. You can do this by pressing “Control-Space” on a PC (Command-Space on a Mac) after typing the first few characters of a word. The Java editor will then try to propose ways to complete the words being typed. (Note that that it is possible that you will encounter a bug where the editor keeps on suggesting “package” or “public”, we would suggest closing and

Exercise Rating: Basic
13. Having created a way of mapping the current infection state to a color, we can now put the final component in place – creation of simple code to make sure that the appropriate color is displayed on the visualization. To accomplish this, click on the oval in the “Person” canvas. The “Properties” window will then display the properties for this oval. Click on the “Dynamic” tab in the “Properties” window. Scroll down (as necessary) to the entry for “Fill Color.” For this entry, we just need to request that the function we just created give us the appropriate color, based on the Person’s current infection state – in other words, we just need to “call” the function. Please insert the following code (without quotes) to call the function to accomplish this: “colorForInfectionState()”

14. Run the model. You should see something similar to the following. As we had hoped, the colors characterize the infection state of individuals in the network.

15. To better understand the difference between transition types, we now re-examine the type of the “infection” transition. Click to select the infection transition within the “Person” canvas. As is shown in the “Properties” window, the transition is currently triggered according to a rate of 0.1 – yielding a mean time infected of 10 time units, but variability in the particular lengths of time spent in the Susceptible state prior to infection for different individuals. We will now examine the effect of changing this transition so that it is triggered not by a rate, but by a timeout.

   a. For the “Triggered by” dropdown, please choose “Timeout”.

Exercise Rating: Basic
b. For the “Timeout” field, enter a value of “10” (without the quotes). Note that by choosing this value, we are preserve the mean length of time spent in the Susceptible state. However, the degree of variability associated with the two triggering mechanisms are very different. While the Rate is associated with significant variability around the mean (the standard deviation is in fact equal to the mean), the timeout exhibits no variability at all – in every case, the duration of time in the Susceptible state will be identical.

What do you expect to see if we are to run this model?

16. Run the model. You should see a progression among states through the model. Given the understanding expressed above, does this make sense to you?

17. The model we have created here has some interesting features, but it’s not yet a model in which infection spreads between people. Right now, each individual in the model may be connected to others via the network, but there is no interaction between individuals. To simulation infection spread, we need to incorporate such interaction. While there are a number of ways that agents (e.g. Persons) in AnyLogic can interact, our way of realizing this interaction is through exchanging messages. Incorporating messages into the model requires several steps. These are explored below.

18. Click to select the “infection” transition within the “Person” canvas. As is shown in the “Properties” window, the transition is currently a Timeout – used for the most recent exploration. We will now change it again – so that infection depends on receiving a message. We will establish the logic such that this message is only be sent by an infectious individual. Specifically, in the “Triggered by” dropdown in the “Properties” window for the “infection” transition, select “Message”. All other settings can retain their current or default values. We note that for simplicity, we have chosen here to represent infection messages as if they inevitably lead to infection.

19. There is one final step required to handle this message properly. The need for this step reflects the fact that while a Person here has just a single statechart (“infectionStatechart”) a given agent (here, Person) may in general have more than one statechart. For example, we could imagine elaborating our model of a “Person” here with a statechart representing the progression of some other infection or strain of the same infection, of a chronic condition, whether the individual is Pregnant or not, or to represent a behavioural risk factor (e.g. smoking). Just as progression within this statechart is contingent upon reception of a message, we can imagine that progression along the other statecharts could also depend on reception of a different message. There is currently no mechanism for AnyLogic to “automagically” know by what statechart – or what statecharts – a given message needs to be handled. In order to complete the handling of an incoming message, we therefore need to “route” it to the appropriate statechart for the person. This routing needs to be accomplish not at the level of the particular statechart, but for the person as a whole. AnyLogic provides a place for this handling associated with each Agent class. Please click on the “Person” class within the Project window, so that the properties for the “Person” class are shown in the “Properties” window. Click on the “Agent” tab of the “Properties” window. This tab shows a set of properties specific to Agents (as opposed to other Active Objects, for example, of which an Agent is just one type). Scroll down as necessary within this tab to find the field entitled “On Message
Received”. This is where you will place the code to “route” message to the appropriate statechart.

What code needs to be placed here? In this particular case, when we receive a given message, we need to simply tell the only statechart present (“infectionStatechart”) to itself receive and handle that message. Fortunately, AnyLogic makes this very easy to do. Firstly, notice a “lightbulb” next to the field. “Hovering” the mouse over this lightbulb will reveal a “tool tip” (see below), indicating that the message received by the person at this point (when this “handler” is triggered) is called “msg”. We just have to pass this message called “msg” to the “infectionStatechart”. We do so with the following code (note the trailing semicolon):

```
infectionStatechart.receiveMessage(msg);
```

The resulting properties looks as follows (note also below the “tool tip” giving the hint about the availability of a variable name for the message and for the sender of the message). Incorporating this code completes what is necessary to handle the message.

Please note that in our version of AnyLogic, a “shortcut” can be used by simply clicking the checkbox next to “infectionStatechart” in the section entitled “Forward message to”. However, we are doing that work manually here. While the automatic mechanism can be relied upon for simple models, more complex models (involving multiple statecharts, each associated with different subsets of messages) will in general require the sort of code you have written here (but with logic to determine “which message goes where”).

20. Having set the logic handling a person’s infection due to exposure to “infection” another party (realized in the model by reception of an “infection” message), we now turn to realize the other side of the process – the process in which an infectious person exposes others to their pathogen. Specifically, we realize this exposure process on the part of an infective by having them send a message. This message is only sent when a person is in fact infective – that is, when they are present in the infective state.

We can implement the sending of this message in a number of ways, but perhaps the simplest is to do so via a transition. Recall that we can associate such a transition with an action to be performed when the transition occurs – for example, in the steps above we associated reporting action (printing a message to the Console window) with a transition firing.

However, it is worth reflecting that the process by which an infective exposes another individual to infection does not by itself lead the infective to immediately recover
from infection. Thus, if we are to implement this message sending process via a transition, it will need to be a special sort of transition – a transition that fires, but maintains the person in the same state. Fortunately, AnyLogic does support this sort of “self-transition”. We now add and configure such a self-transition.

a. Please add a self-transition to the “Infective” state by adding a Transition from the “Model” tab of the Palette window, and connecting both ends of that transition (i.e. both the origin and destination of the transition) to the Infective state. You can choose your own aesthetic style for such self-transitions, but the figure below shows a common visual representation.

b. The properties for this transition must now be set in the “Properties” window.
   i. Set the name of the transition to “pathogenExposure”. (Note that a better name might reflect the fact that this exposure occurs at a level that it is an infecting dose if it is received by a Susceptible individual)
   ii. As with any other transition, we may wish to have the exposure events within the “Infective” state occurring according to a timeout (i.e. at regular intervals), as poisson arrivals (i.e. a certain average rate, but where the occurrence in the next small amount of time is independent of the last time exposure occurred), or other variants. In this case, please
      1. Set the “Triggered by” to be a Rate
      2. Set the “Rate” itself to 0.5 – i.e. 2 time units on average between exposure events.
   iii. Recall that the motivation for this transition lies in the action to be performed (after all, the transition does not go between two different states, but instead leads a person to remain in the same state). Here, the action will involve exposing others people to the pathogen. But which other people?

Here, we will assume that the individuals at risk of exposure are the “next-door neighbours” to the infective person within the network – in other words, the individuals who are just one connection away from the infective person.

The patterns of exposures of these neighbours will differ by pathogen

Exercise Rating: Basic
and context. For example, an STI or bloodborne infection would generally only be transmitted to one connected individual at a time – that is, each potentially infecting event (e.g. a sexual contact or case of needle sharing) would only infect at most one other individual. By contrast, an airborne infection such as TB or Influenza could in principal be transmitted to multiple people within a single infecting event.

Here, we choose to assume that each potentially infecting event only affects a single person at a time. Given this, we need to find a way that this infective could send a message to exactly one neighbour (say, selecting among the possible neighbors with uniform probability) at a time. Fortunately, AnyLogic provides ready support both for exposing a single randomly selected connection at a time and all connections at a time. To send to just one neighbour we use the code fragment:

```java
this.send("Infection", RANDOM_CONNECTED);
```

Please insert this code (including the trailing semicolon!) in the “Action” field of the “Properties” window.

Note that when “send” is called here it tells AnyLogic to deliver a message. The first “argument” given to “send” (“Infection”) is just a message; here, we do not pay attention to the contents of that message, so the choice of the message to send is optional. In other models – and in future evolutions of this model in other exercises! – we may wish to have many different messages, so it’s nice to stick to a consistent naming. The second argument tells AnyLogic to what particular Agent (Person) or set of persons the message should be sent. Here, “RANDOM_CONNECTED” indicates to AnyLogic that it should select a neighbour selected with equal probability. (Note that sending to all neighbours would be accomplished in a similar fashion, but would replace “RANDOM_CONNECTED” by “ALL_CONNECTED”. Other options are also available)

The model should then look as follows:
21. There is just one step remaining before our infection model is ready to simulate outbreaks of infection. Right now, infection can spread from one person to others. However, that infection won’t spread unless it is already carried by somebody. To start an epidemic, we need some way of “seeding” the infection – of introducing the very first infectious person into a model.

There are many ways we might think to seed an infection in the model. For example, models with an open population might have some immigrants arriving who are already ill. In some cases (e.g. prion based illnesses or zoonoses), a person might pick up an infection from the broader environmental reservoirs that happen to be excluded from the model. Here, we will simply infect one person – randomly selected from the population with equal likelihood – at the inception of the simulation. Once again, AnyLogic makes this process quite easy. We accomplish by telling the environment to deliver a message – here, the same “Infection” message that we sent to spread the infection – to a randomly selected person in the population. Because the population of people is “known about” by the environment, the environment can take care of this task.

To accomplish this, click on the “Main” class in the “Project” window. In the (default) “General tab”, there is a field entitled “Startup Code”. Please enter `environment.deliverToRandom("Infection");`
22. The above steps have having handled the mechanics of message reception, message sending, and the initial seeding of infection in the population. All we have to do now is to run the model!

Over time, the infection will spread within the “connected components” of the network (see below).

Exercise Rating: Basic
Exercise Rating: Basic