Agent-Based SIR Model in Numerus Model Builder (NMB)

Richard M. Salter

In this exercise we will build a model consisting of agents moving across a landscape.

The agents can be in one of two states: *susceptible* or *infected*. When a susceptible agent encounters an infected one, it may become infected. Infected agents remain infected or may recover and become susceptible again. The probabilities for infection and recovery are set by the user.

Agents move in random fashion over a grid of cells, or *patches*. A patch can be of one of two types: *habitat*, through which an agent can pass, or *barrier*, through which it cannot.

NMB provides a structure called a *capsule* in which you design each part of the model. Our model requires a separate capsule for agent and patch, and also one for the common "world" in which they exist. The component used for the latter, called a "Simulation World", or *SimWorld*, manages the movement of the agents over the grid of patches, the interactions among the agents, and any communication between agents and patches.

Each of the 3 capsules is constructed as a set of interacting graphical components. Each capsule component contains program code defining its behavior. The code required for this model is relatively simple.

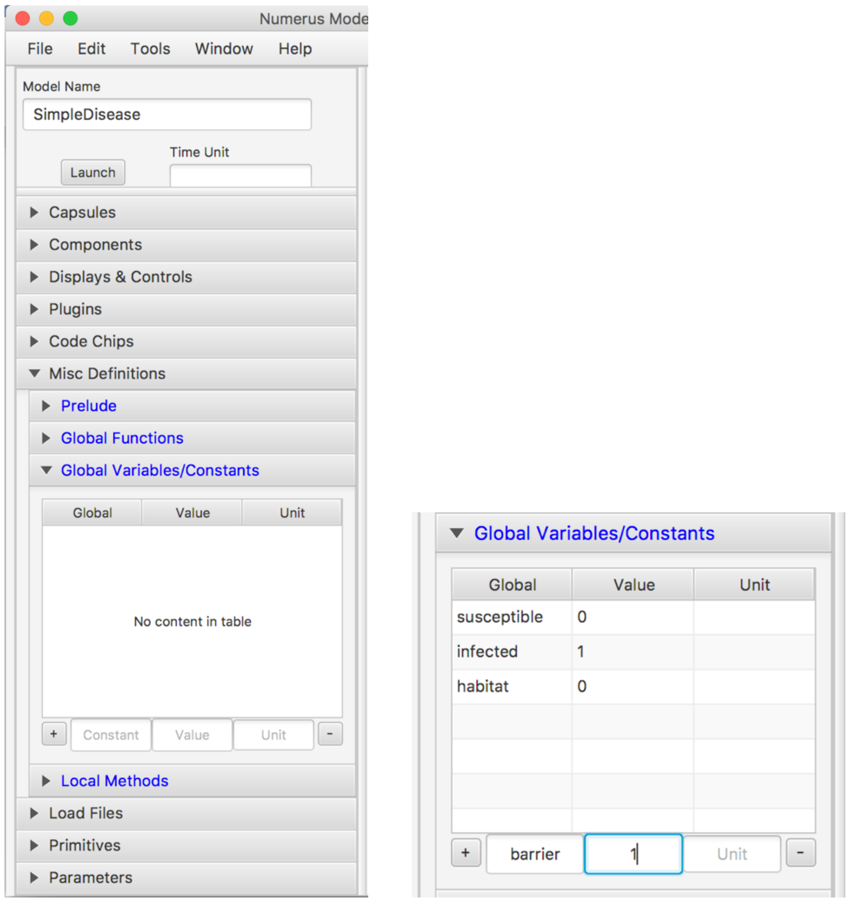
The steps for constructing this model are

I. Preliminaries;
II. Build the patch capsule;
III. Build the agent capsule;
IV. Build the top-level capsule;
V. Configure the visual elements;
VI. Initialize and test the model

I. Preliminaries

1. Open NMB; the Capsule frame will appear. Select "New" from the file menu.
2. Under "Model Name" enter something like "SimpleDisease" or "SIR" (for *susceptible-infected-recovered*).
3. We want to define several *constants* that we can use in the program to represent agent state and patch type. These will make the programs more readable.
    a. In the left panel open the "Misc Definitions" tab, followed by the "Global Variables/Constants" tab. (You will probably have to lift the panel at the top, just above "Capsules", to gain enough vertical space; see Figure 1a.)
    b. Enter the following constant definitions one-at-a-time into the constant text box, followed by a tab, followed by its value, then click the "+" button or press return (see Figure 1b). Whe you are done, close the "Misc Definitions" tab and save your work.

susceptible	0
infected	1
habitat	0
barrier	1



Model Name: SimpleDisease

Time Unit:

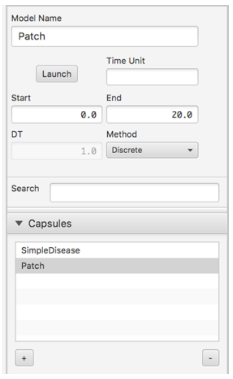
Global	Value	Unit
susceptible	0	
infected	1	
habitat	0	

Constant: barrier | Value: 1

Figure 1: a) Global Variables/Constants Frame; b) Constant value entries

II. Build the patch capsule

1. Open the “Capsules” tab and click the “+” button. A new capsule called “Untitled” will appear. Click on “Untitled” in the capsule list and change its Model Name to “Patch”



Model Name: Patch

Time Unit:

Start: 0.0 | End: 20.0

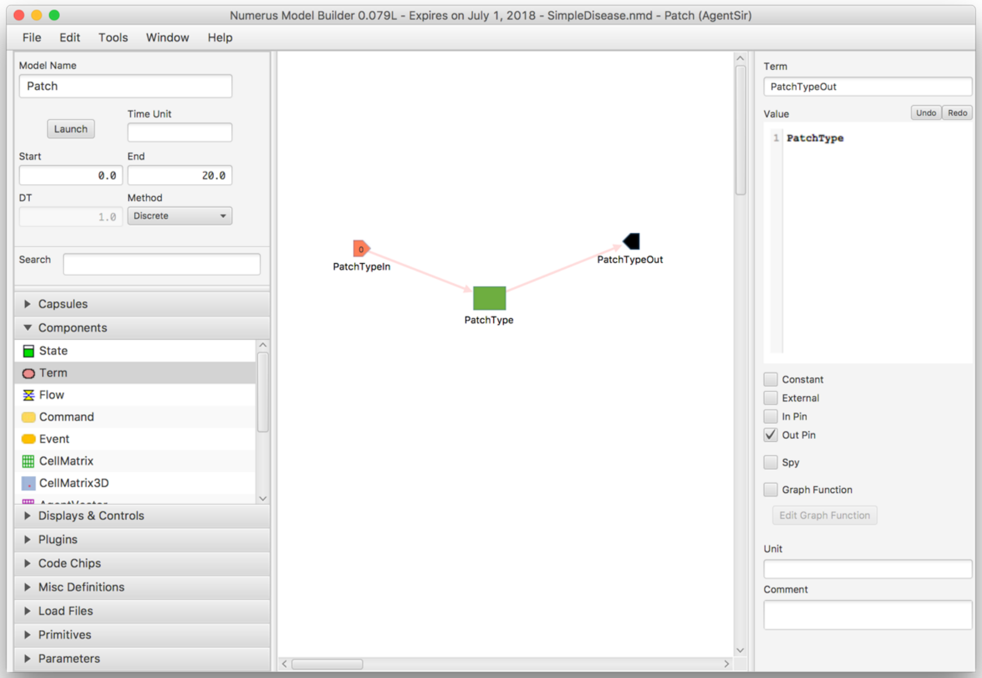
DT: 1.0 | Method: Discrete

SimpleDisease
Patch

Figure 2: Creating the Patch capsule

2. We are now ready to add content to the Patch capsule. *Make sure you have clicked this capsule before proceeding*. Open the components tab and add:

   a. a State called “PatchType”;
   b. a Term called “PatchTypeIn”;
   c. a Term called “PatchTypeOut;
3. In PatchTypeIn’s property pane check “In Pin”; in PatchTypeOut’s property pane check “Out Pin”.
4. Click the PatchType State and into its Init pane replace “0.0” with “PatchTypeIn” (an arrow should appear connecting PatchTypeIn with PatchType).
5. Click the PatchTypeOut Term and replace its value with “PatchType” (a similar arrow should now connect PatchType to PatchTypeOut). See Figure 3.



Model Name: Patch
Start: 0.0
End: 20.0
DT: 1.0
Method: Discrete
Term: PatchTypeOut
Value:
```
PatchType
```
[✓ Out Pin]

Figure 3: Patch capsule

6. Save your work when done.

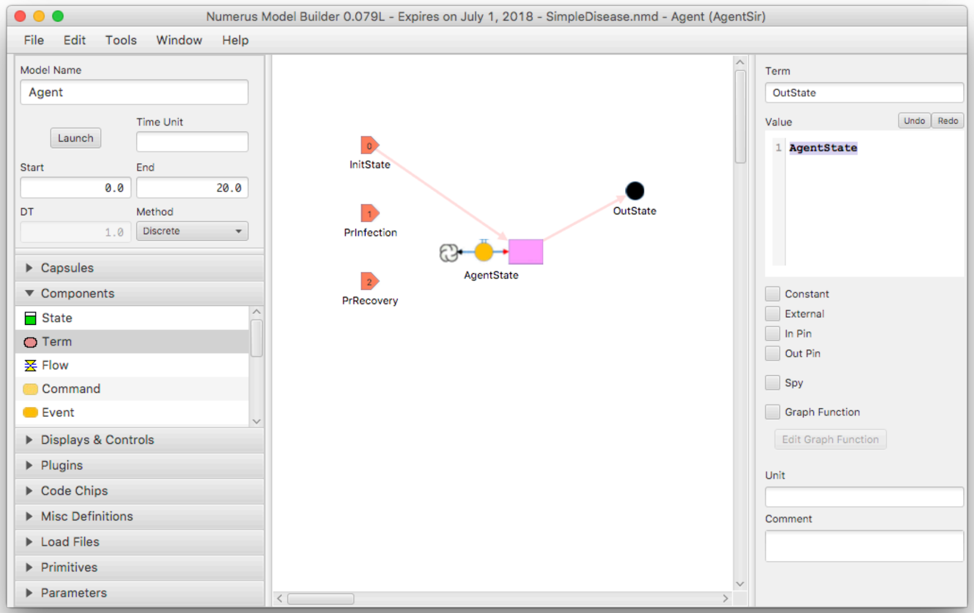
Note that the Patch capsule is quite simple; it merely retains a persistent type value (habitat or barrier) in a stock. PatchTypeIn and PatchTypeOut respectively initialize the type and provide a means of reading it. This type value will be observed by the agent when deciding where to move.

III. Build the agent capsule;

1. Repeat Step II.1 to create a capsule called “Agent”. *Before proceeding make sure you have selected the Agent capsule for input.*
2. The agent requires 2 separate capabilities: it must determine its new state (susceptible or infected) based on its current state and the presence/absence of other nearby agents; and it must move, taking care to avoid moving into barriers. Let’s first consider the agent’s state.

   We will use a special type of state component called a *Sequence* to represent this. From the components tab add a State component and call it “AgentState”. On AgentState’s property pane, select “Sequence” from the list of state types (Stock, Stock+Flow, Sequence, Store). Notice that the Sequence is already attached to a Flow. As with a Stock, the Flow determines the next value of the Sequence; however, whereas the value computed by the Flow to a Stock is *added* to the Stock’s current value, the value of a Flow to a Sequence *replaces* the current value. This is perfect for our application since new states (susceptible or infected) will replace old ones.

3. We need 4 Terms (actually 3 InPins and 1 OutPin) to complete this phase of the program: The InPin “InitState” initializes the agent’s state at the start of the simulation; InPins “PrInfection” and “PrRecovery” respectively are used to control the transition from susceptible to infected and vice versa. The sole OutPin called “StateOut” and it is used to convey the agent’s current state to the outside world. Add these now, and make InitState the value of the Init field of AgentState, and AgentState the value of OutState. See Figure 4.


Numerus Model Builder 0.079L - Expires on July 1, 2018 - SimpleDisease.nmd - Agent (AgentSir)

Model Name: Agent
Time Unit:
Start: 0.0
End: 20.0
DT: 1.0
Method: Discrete

Term: OutState
Value:
```
AgentState
```

Figure 4: Agent capsule (start)

4. We now consider the most complex part of the model, computing an agent’s new state. Fortunately, much of the complexity of this action is handled by special NMB operators called *primitives*.

   First, let’s look at the logic of this process. An agent will act differently depending on whether it is currently sick or well (i.e. infected or susceptible). If the agent is infected, it loses its infection (i.e. becomes susceptible) with a probability given by PrRecovery. This can be carried out by selecting a random number x between 0 and 1, and changing from infected to susceptible if $x <$ PrRecovery (note how in the extremes, if PrRecover = 0 this will never happen, and if PrRecovery = 1 it will always happen.)

   Now consider the case where the agent is susceptible. The agent will only become infected if at least one infected agent is nearby. If this is the case, we use PrInfection in a similar way to determine if in fact the agent is to become infected (note that we are using a particularly simple approach in which the number of nearby infected agents does not affect the probability of the outcome).

Let the variable *newState* represent the next state of the agent. With respect to this variable, the logic of the agent's decision process is

1. To start, assume no change (i.e. newState = current AgentState);
2. if current AgentState is **infected**, then newState = **susceptible** with probability PrRecovery;
3. else if current state is **susceptible**, then
    a. Collect the agent's neighbors;
    b. If any neighbor is infected, then newState = **infected** with probability PrInfection.

In program code, it looks like this:

```
var newState = AgentState;
if (AgentState == infected) {
      if (FLIP(PrRecovery)) {
            newState = susceptible;
      }
} else {
      var neighbors = AGENTBLOCK(1);
      if (SOME_AGENT(neighbors, "AgentState", infected)) {
            if (FLIP(PrInfection)) {
                  newState = infected;
            }
      }
}
```

This code makes ample use of NMB's primitive functions:

FLIP(*x*) returns true with probability *x* (and false with probability 1-*x*). It operates using a random value as described above.

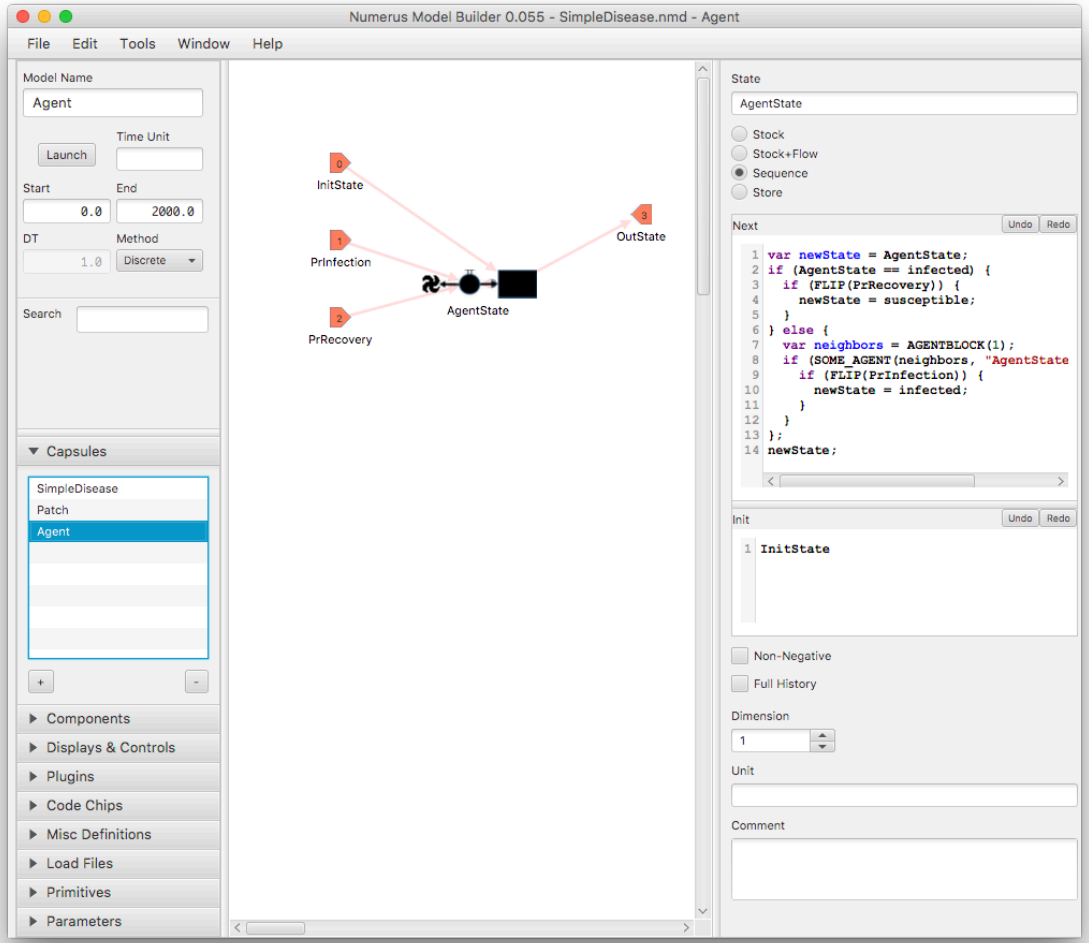
AGENTBLOCK(*x*) returns the set of all agents occupying patches within *x* units from the calling agent. We've chosen a small neighborhood, one unit around the caller.

SOME_AGENT(*set*, *variable, value*) returns true if for any agent in the *set* of agents the value of *variable* is *value*. This reads as "is there some neighbor whose AgentState is **infected**?"

We need to add 1 line at the end of the code shown above, in order to indicate that the value of interest is newState.

```
var newState = AgentState;
if (AgentState == infected) {
      if (FLIP(PrRecovery)) {
            newState = susceptible;
      }
} else {
      var neighbors = AGENTBLOCK(1);
      if (SOME_AGENT(neighbors, "AgentState", infected)) {
            if (FLIP(PrInfection)) {
                  newState = infected;
            }
      }
};
newState;
```

Copy this code into the Next textbox on the AgentState Property panel. Notice that the arrows appear showing that your computation depends on the two probabilities (see Figure 5.)



Model Name: Agent
Time Unit:
Start: 0.0
End: 2000.0
DT: 1.0
Method: Discrete

State: AgentState
Stock
Stock+Flow
[✓ Sequence]
Store

Next

```
var newState = AgentState;
if (AgentState == infected) {
  if (FLIP(PrRecovery)) {
    newState = susceptible;
  }
} else {
  var neighbors = AGENTBLOCK(1);
  if (SOME_AGENT(neighbors, "AgentState
    if (FLIP(PrInfection)) {
      newState = infected;
    }
  }
};
newState;
```

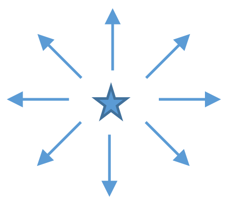
Init

```
InitState
```

Non-Negative
Full History
Dimension: 1
Unit:
Comment:

Figure 5: Agent capsule with logic governing spread of infection

5. The final step in programming the agent is to specify its movement. We want the agent to execute a “random walk” by moving one step in any of 8 different directions at each time step.



Fortunately, NMB has primitives for this called *RANDOM_MOVE* and *MOVETO*. A random move is achieved by executing the following code:

```
var newXY = RANDOM_MOVE();
MOVETO(newXY)
```

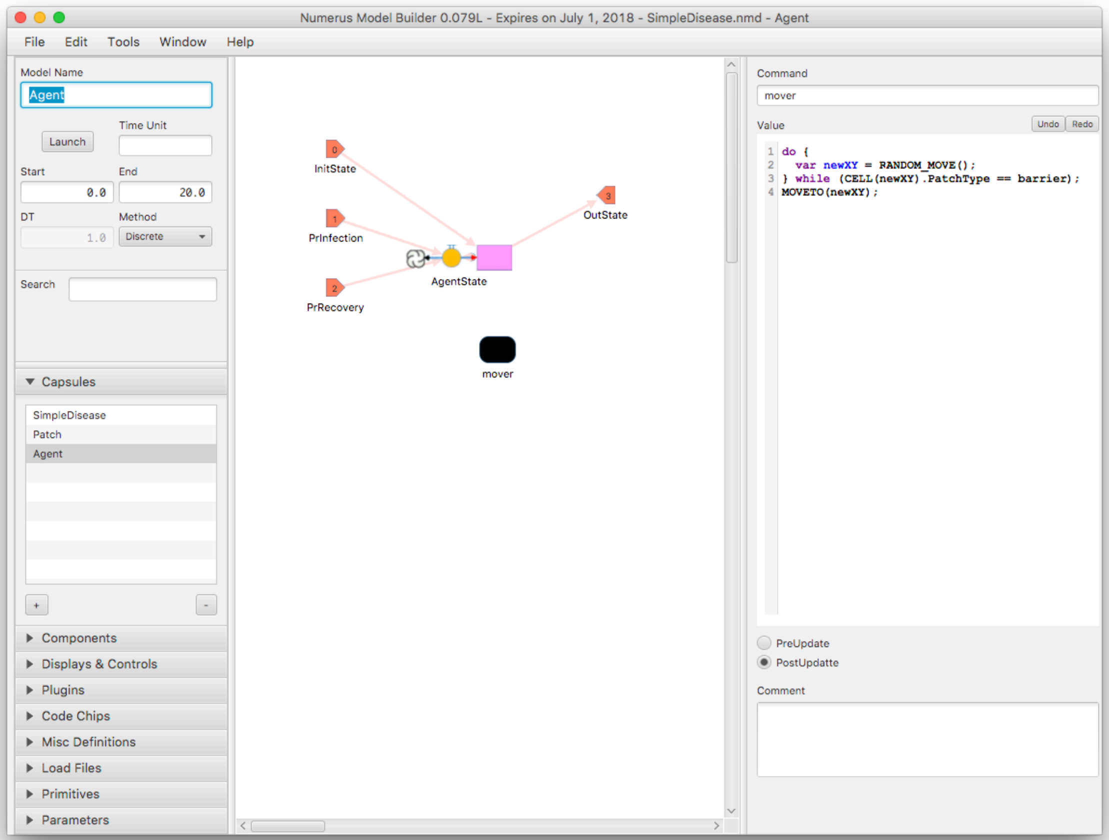
Here `newXY` holds the agent's new cell coordinates in the grid of cells. MOVETO moves the agent to that location.

This is complicated by the existence of barriers possibly blocking an agent's move. If the cell at newXY is of type "barrier" the agent is barred from moving there. The solution is to keep trying a RANDOM_MOVE() until the destination proves to be habitat rather than barrier. Eventually one will be found unless the agent is completely surrounded by barrier, and we'll make sure this doesn't happen. The code make this work is

```
do {
      var newXY = RANDOM_MOVE();
} while (CELL(newXY).PatchType == barrier);
MOVETO(newXY);
```

`CELL(newXY)` grabs the patch specified by the coordinates so that `CELL(newXY).PatchType` can examine the PatchType stock in that patch. The `do` construct repeats the `RANDOM_MOVE` operation until the PatchType of the destination is habitat rather than barrier.

Add a Command component to the Agent capsule and copy this code into its Value textbox (Figure 6). The agent is now complete. Be sure to save your work.



Model Name: Agent
Time Unit:
Start: 0.0
End: 20.0
DT: 1.0
Method: Discrete

Command: mover

Value:

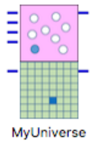
```
do {
  var newXY = RANDOM_MOVE();
} while (CELL(newXY).PatchType == barrier);
MOVETO(newXY);
```

PreUpdate [✓ PostUpdatte]

Figure 6: Agent move command

IV. Build the top-level capsule

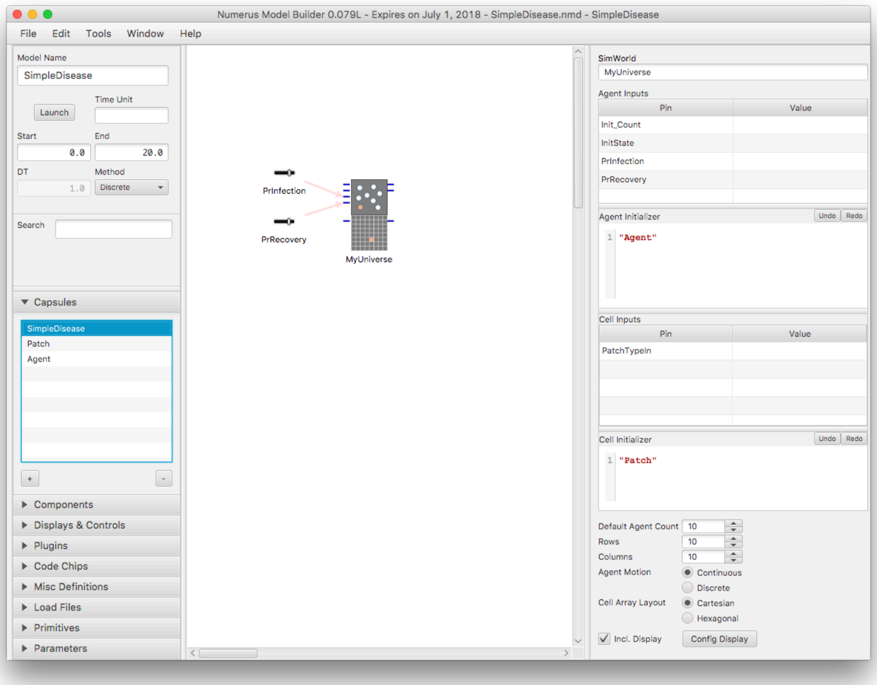
1. In the Capsule tab click the top-level capsule (SimpleDisease or SIR), which is currently empty.
2. From the Components tab add a *SimWorld*. Call it “MyUniverse”. The component is divided in half, with top showing circles representing agents and the bottom showing a grid representing cells.
3. Open the Capsules tab and drag the Agent entry over the circles, then drop it (the background will glow purple when it’s time to drop).
4. Similarly drag the Patch entry over the grid and drop when the background glows green.
5. When you have completed these steps the SimWorld should display input and output pins as shown below. If they don’t, repeat the process.



6. Open the Displays & Controls tab and add two sliders to the capsule. Call them “PrInfection” and “PrRecovery”. These sliders will provide the probabilities used in the agent state logic discussed above. Using their property panes, initialize the sliders as follows:

PrInfection:	Minimum: 0.0, Maximum: 1.0, Step: 0.001, Initial: 1.0
PrRecovery:	Minimum: 0.0, Maximum: 1.0, Step: 0.001, Initial: 0.0

7. Connect the sliders to MyUniverse as follows (see Figure 7):
   a. Open MyUniverse’s property pane;
   b. In the “Agent Inputs” table click on “PrInfection”;
   c. *Right-click* the PrInfection slider;
   d. Similarly click on “PrRecovery” in the table and right-click on the PrRecovery slider



Model Name: SimpleDisease
Time Unit:
Start: 0.0 End: 20.0
DT: 1.0 Method: Discrete

Capsules: SimpleDisease, Patch, Agent

SimWorld: MyUniverse

Agent Inputs

Pin	Value
Init_Count	
InitState	
PrInfection	
PrRecovery	

Agent Initializer

```
"Agent"
```

Cell Inputs

Pin	Value
PatchTypeIn	

Cell Initializer

```
"Patch"
```

Default Agent Count: 10
Rows: 10
Columns: 10
Agent Motion: [✓ Continuous] Discrete
Cell Array Layout: [✓ Cartesian] Hexagonal
[✓ Incl. Display]

Figure 7: Connecting probability sliders

V. Configure the visual elements

When the simulation is run NMB will display MyUniverse as a window of patches containing moving circles representing agents. The patch display must be configured to show a different color for habitat and barrier, while the agents must also use color to distinguish the susceptible from infected. All of this is specified on the display configuration pane.

1. Select MyUniverse and in the property pane set the number of rows and columns to 60.
2. Now click the "Config Display" Button at the bottom (make sure "Incl Display" is checked). The initial contents of this panel is shown in Figure 8a.
3. Make the following selections in the first column:
    a. Agent Color Pin: StateOut
    b. Cell Color Pin: PatchTypeOut
    c. Interactive: Checked
    d. Initialized Agent Pin: InitState
    e. Initialized Cell Pin: PatchTypeIn
4. In the Agent Colors section:
    a. Agent Colors: 2
    b. Random Unchecked
    c. AutoFill Checked
    d. Low 0
    e. High 1
5. Now change the agent colors to be green (0 - susceptible) and red (1 - infected). (Red is probably already selected for 1). Do this by clicking on the Black tab and using the color selector find your favorite shade of green.
6. Finally, you must similarly select 2 cell colors for 0 and 1 (habitat and barrier, respectively). Use whatever colors you like best – you can always change them later. I prefer light gray and blue.
7. When you are done the display configuration panel should look like Figure 8b.

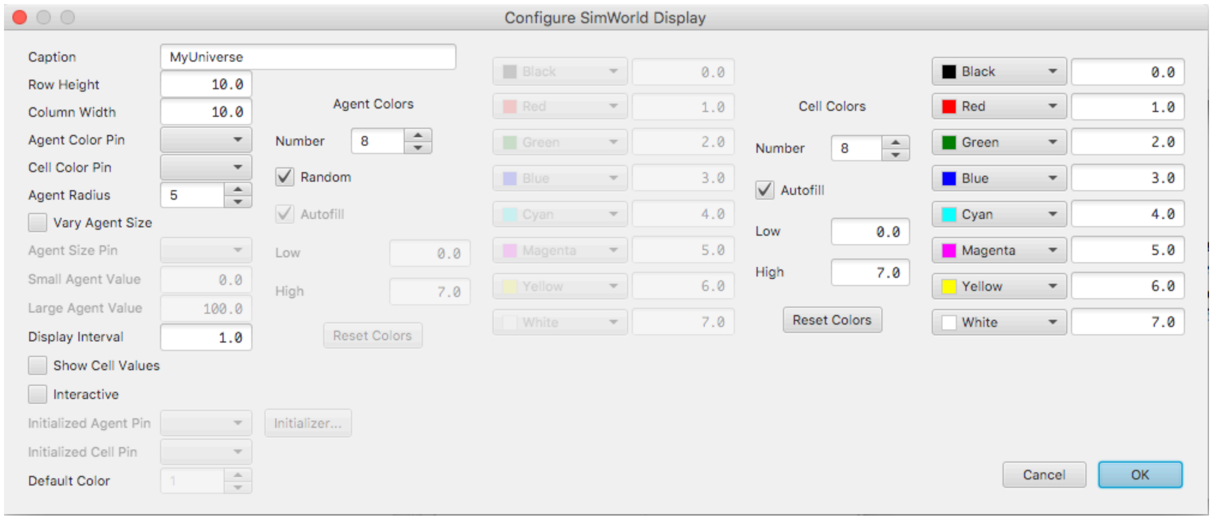
VI. Initialize and test the model

We're now ready to try the model. In the upper left panel change the end time to 2000 and click the launch button. In a few seconds a new frame looking like Figure 9a (the Runtime frame) will appear. This frame shows a window "world" consisting entirely of habitat with a population of 0 agents. We'll use the mouse to populate this world.

1. To select barrier or habitat:
    a. Click on any cell to toggle from habitat to barrier; click again to toggle back to habitat;
    b. Press on any cell and drag across the window to change patch types to that cell's type;
    c. Click the "Clear Cells" button to remove all barrier patches.
2. To add, remove or change agent state
    a. Shift and click on any cell to produce an (initially) susceptible agent;
    b. Click on that agent to change its state to infected;
    c. Click a third time to delete that agent;
    d. Click the "Clear Agents" button to delete all agents.

When you have finished, click "Run" and watch what happens. You can run the simulation numerous times by clicking Reset followed by Run. You can also change the configuration between runs.

To save a configuration be sure to save the program as in earlier steps. This configuration becomes your initial configuration on subsequent runs of the model. You can restore an earlier saved configuration by reloading the saved file from the Capsule frame.


a.

Configure SimWorld Display

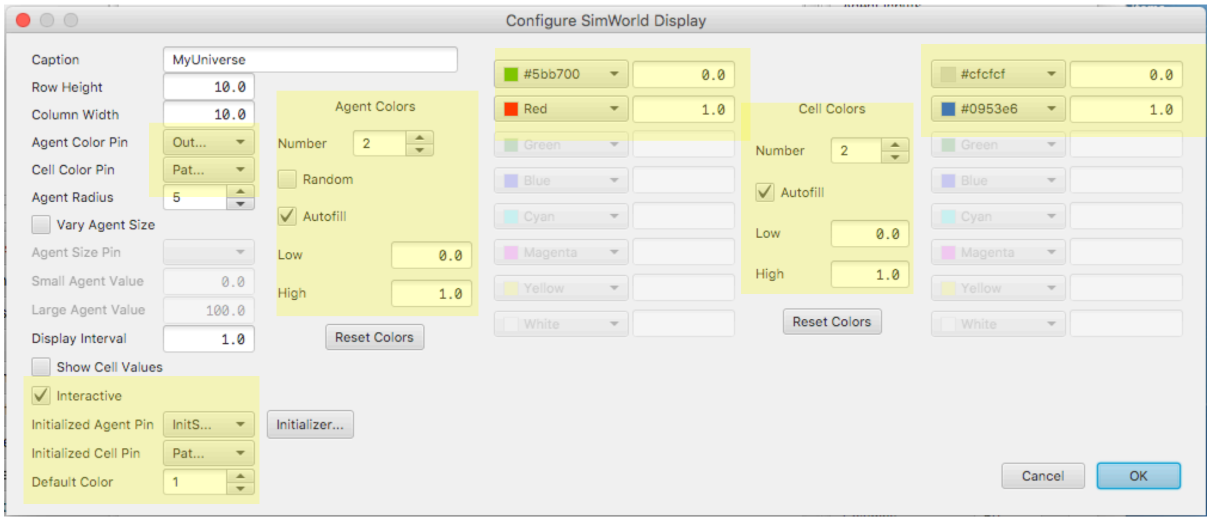
Caption	MyUniverse
Row Height	10.0
Column Width	10.0
Agent Color Pin	
Cell Color Pin	
Agent Radius	5
Vary Agent Size	☐
Agent Size Pin	
Small Agent Value	0.0
Large Agent Value	100.0
Display Interval	1.0
Show Cell Values	☐
Interactive	☐
Initialized Agent Pin	
Initialized Cell Pin	
Default Color	1

Agent Colors: Number 8; Random ✓; Autofill ✓; Low 0.0; High 7.0; Reset Colors

Black	0.0
Red	1.0
Green	2.0
Blue	3.0
Cyan	4.0
Magenta	5.0
Yellow	6.0
White	7.0

Cell Colors: Number 8; Autofill ✓; Low 0.0; High 7.0; Reset Colors

Black	0.0
Red	1.0
Green	2.0
Blue	3.0
Cyan	4.0
Magenta	5.0
Yellow	6.0
White	7.0


b.

Configure SimWorld Display

Caption	MyUniverse
Row Height	10.0
Column Width	10.0
Agent Color Pin	Out...
Cell Color Pin	Pat...
Agent Radius	5
Vary Agent Size	☐
Agent Size Pin	
Small Agent Value	0.0
Large Agent Value	100.0
Display Interval	1.0
Show Cell Values	☐
Interactive	✓
Initialized Agent Pin	InitS...
Initialized Cell Pin	Pat...
Default Color	1

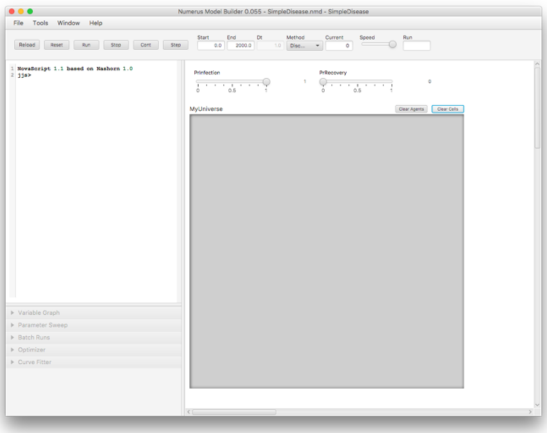
Agent Colors: Number 2; Random ☐; Autofill ✓; Low 0.0; High 1.0; Reset Colors

#5bb700	0.0
Red	1.0
Green	
Blue	
Cyan	
Magenta	
Yellow	
White	

Cell Colors: Number 2; Autofill ✓; Low 0.0; High 1.0; Reset Colors

#cfcfcf	0.0
#0953e6	1.0
Green	
Blue	
Cyan	
Magenta	
Yellow	
White	

Figure 8: Display configuration a) before changes; b) after changes (highlighted)


a.

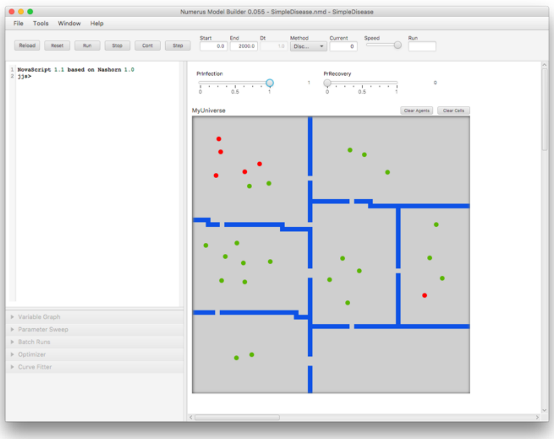

b.

Figure 9: a) Empty runtime frame; b) After adding barrier and agents