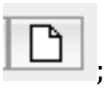
Agent-based SIR model in Nova
Richard M. Salter

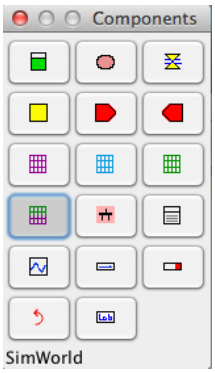
I. Create the framework. This is the contents of SIRTemplate.nva

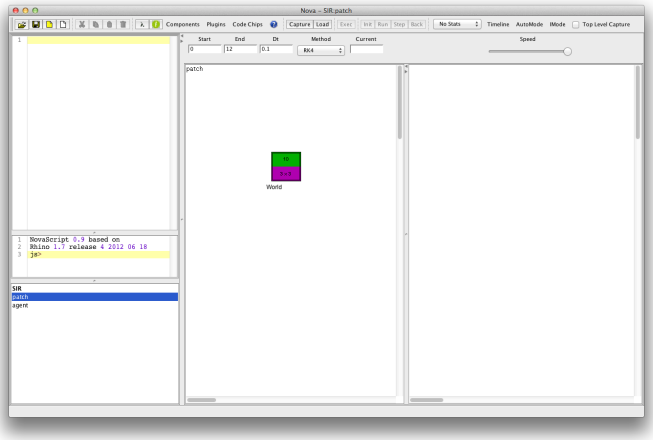
1. Open Nova to a new Untitled model.

   Click on "New Sub Model" on the toolbar  ; call it "agent".
   Return to top level and Click again on "New Sub Model"; call it "patch".

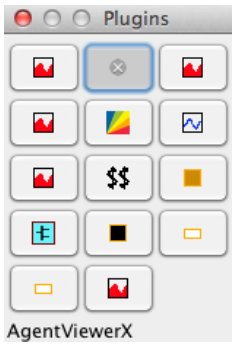
   Click "Save"  and call your model "SIR.nva".

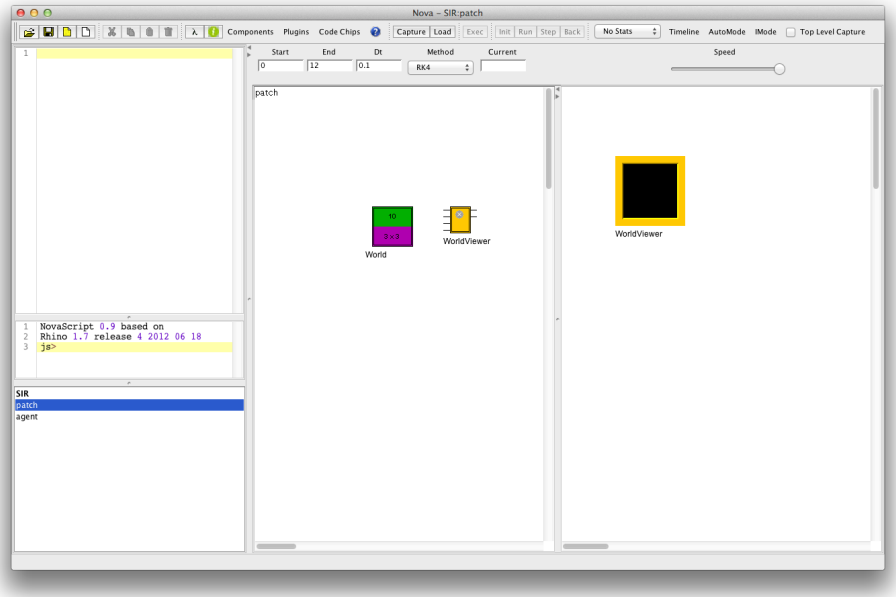
2. All of the following steps are applied to the top-level (SIR) model.
   Open the **Components** pallet and select SimWorld. Click on the design canvas. Call it **World**.





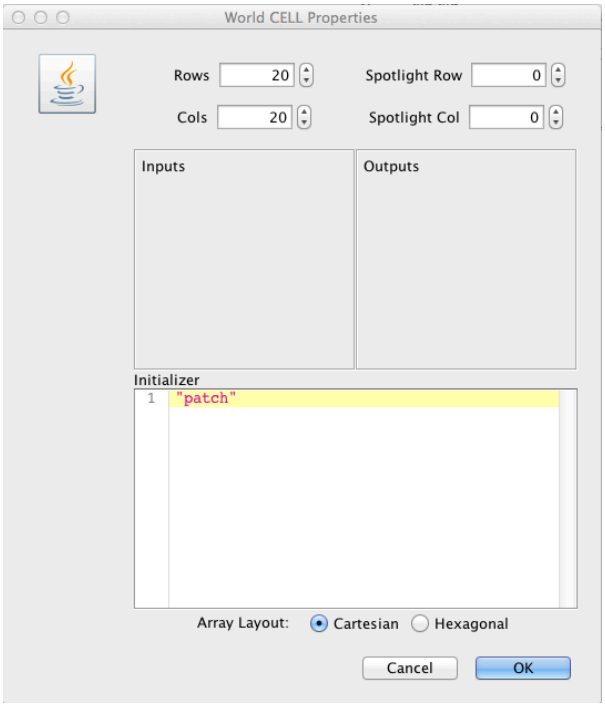
3. Open the **Plugins** pallet and select **AgentViewerX**. Click on the dashboard. Call it **WorldViewer**.
   Save your work.



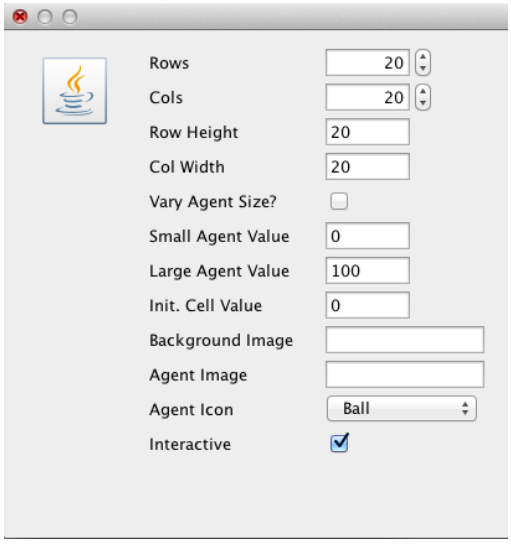


4. Drag the sub model “patch” to the purple part of **World**. Drag the sub model “agent” to the green part of World.

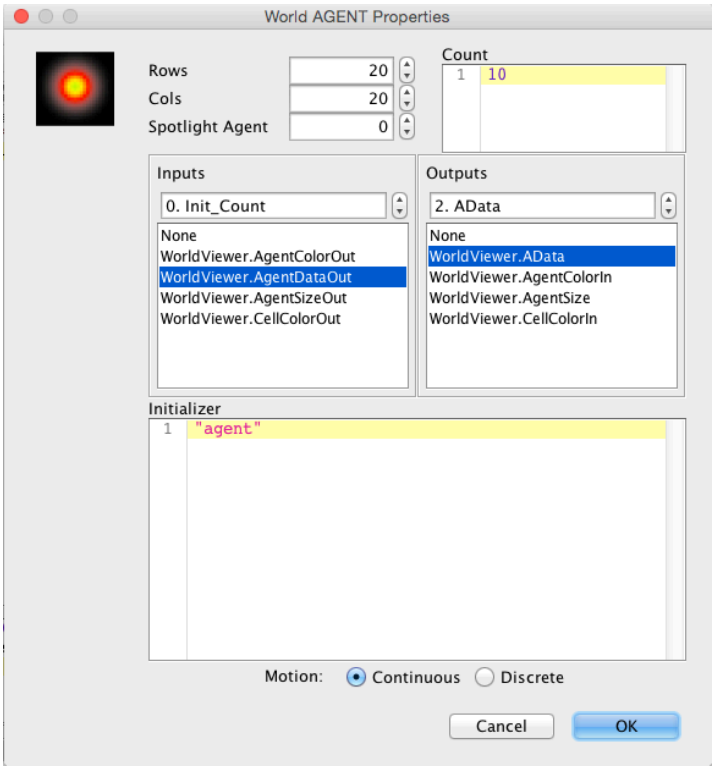
5. Right-click on the purple part of **World**. Set the value of Rows and Cols to both be 20.

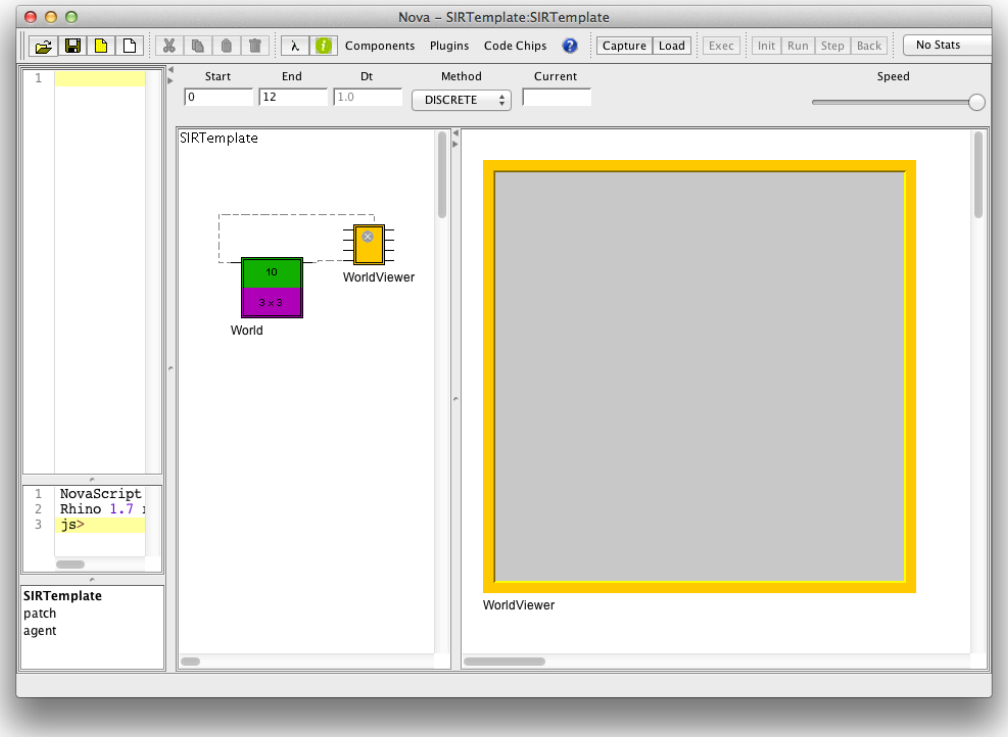


6. Right click on the edge of the **WorldViewer** on the dashboard. Set the value of Rows, Cols, Row Height and Col Width all to be 20 (these controls are all on the left side of the Properties dialog box). Click the Interactive box at the bottom.



7. Right-click on the green part of **World**. Select the connections shown below left. Your model should look like the bottom image. This completes the Framework. Save your work.



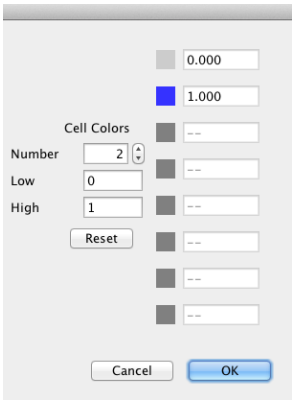


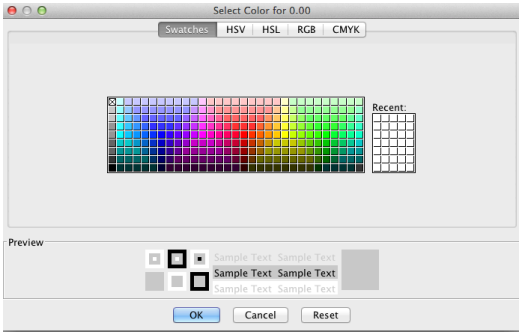
II. Build the patch model.

8. Add the following code to the Programming Window at the top level to create patch types **habitat** and **barrier**:

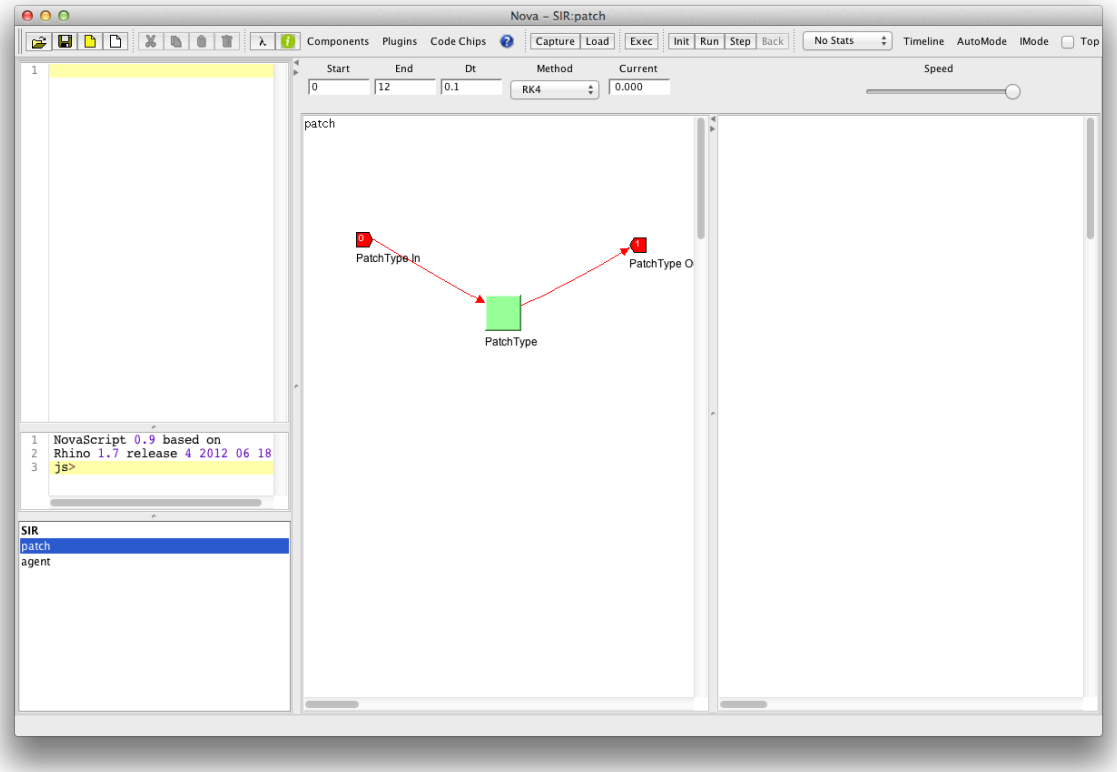
```
const habitat=0, barrier=1;
```

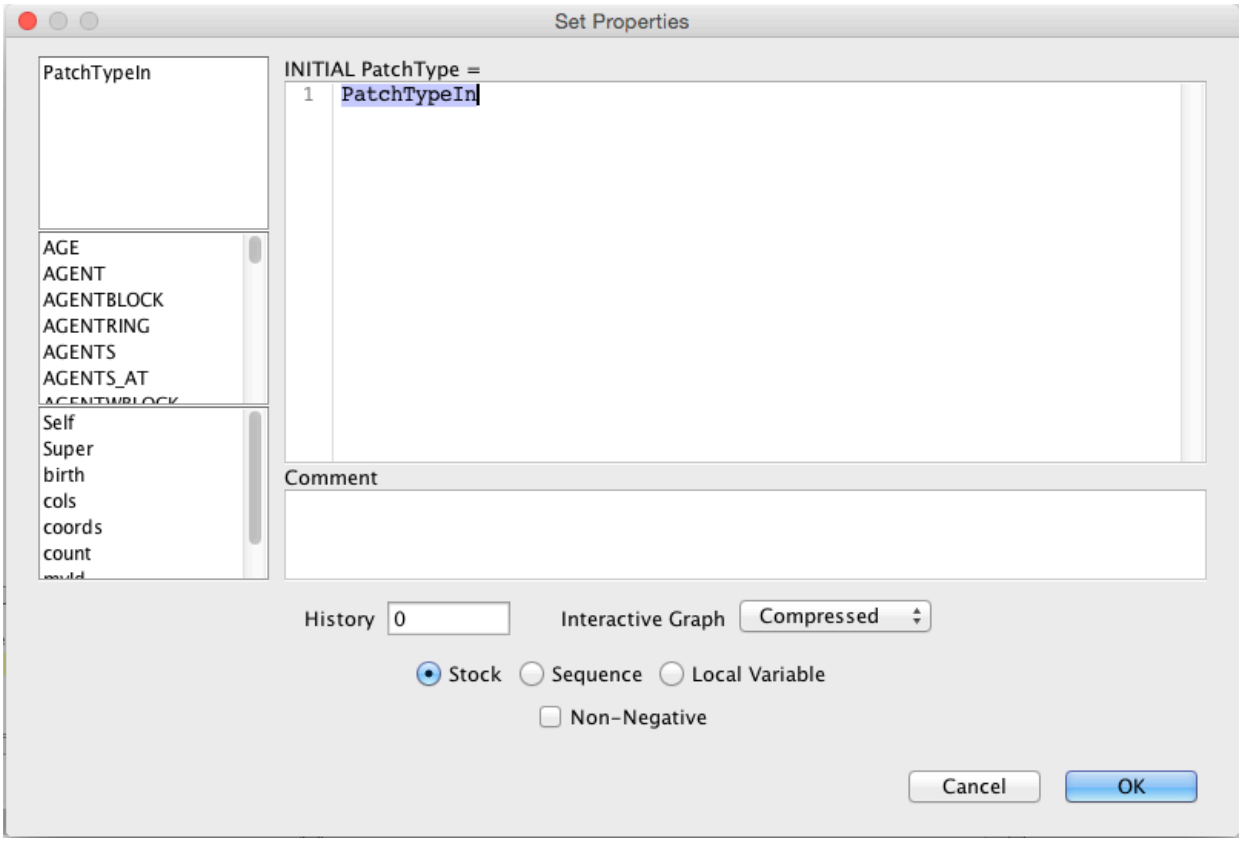
9. We next want to program the background colors for these two types. We'll use gray for habitat and blue for barrier. Right-click the rim of the **WorldViewer** on the dashboard and set the Cell Colors as shown. To change a color, click on the small color square and select the color from the color chooser pallet that appears. Similarly, set the Agent Colors to green and red as shown. Close the **WorldViewer** Properties Panel when you are done.

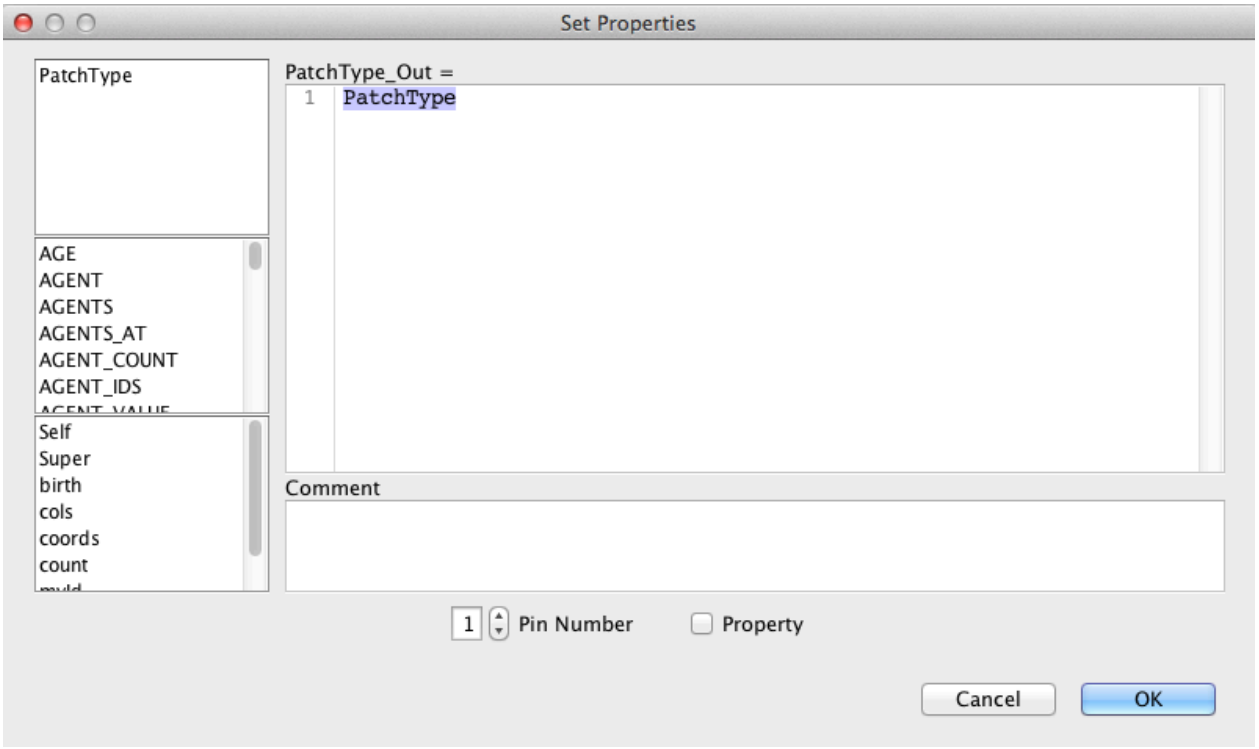




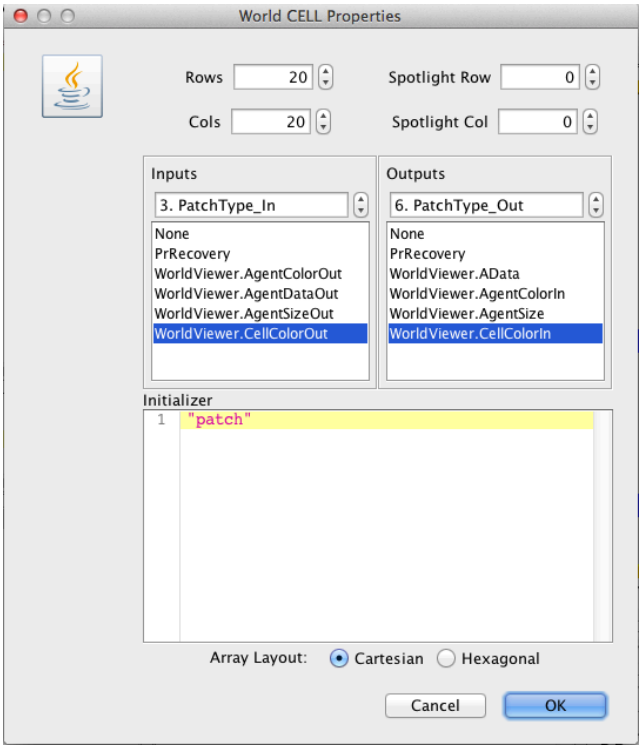
10. Click on the patch sub model, which should be empty. Add a Stock called **PatchType**, a Data Input called **PatchTypeIn** and a Data Output called **PatchTypeOut**. Connect with arrows as shown. Right-click **PatchType** and set its input to **PatchTypeIn**. Right-click **PatchTypeOut** and set its value to **PatchType** (if it's not already set). Save your work.







11. Return to the top level and click on the purple part of the **World**. Make the connections shown.



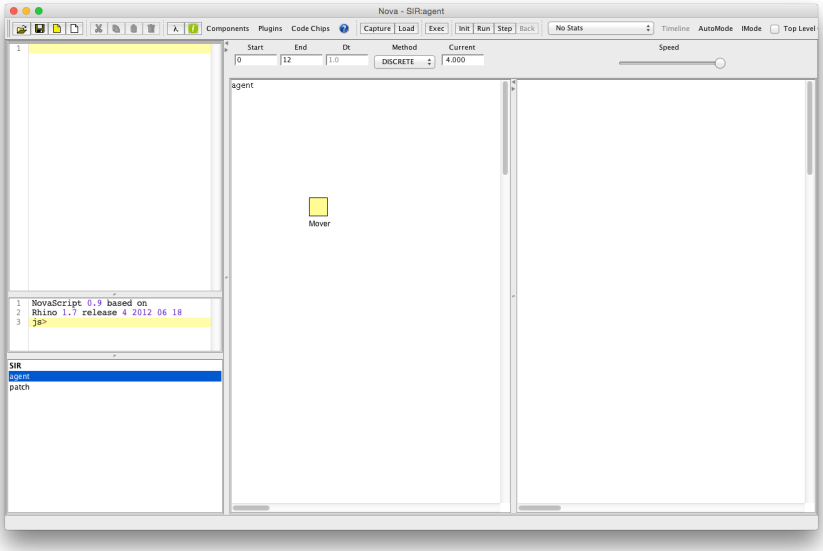
III. Creating the agent 1: Motion

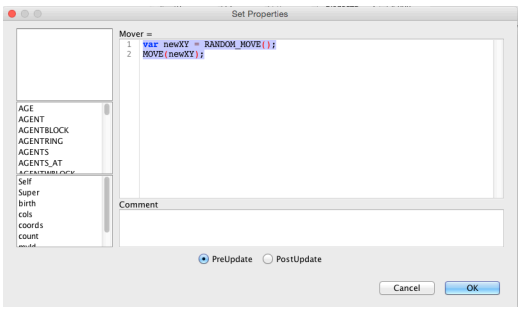
12. Add the following to the top-level Programming Window to define the two agent states: **susceptible** and **infected**.

```
const susceptible=0, infected=1;
```

13. Move to the agent submodel. Add a Command called **Mover**. Include the code as shown

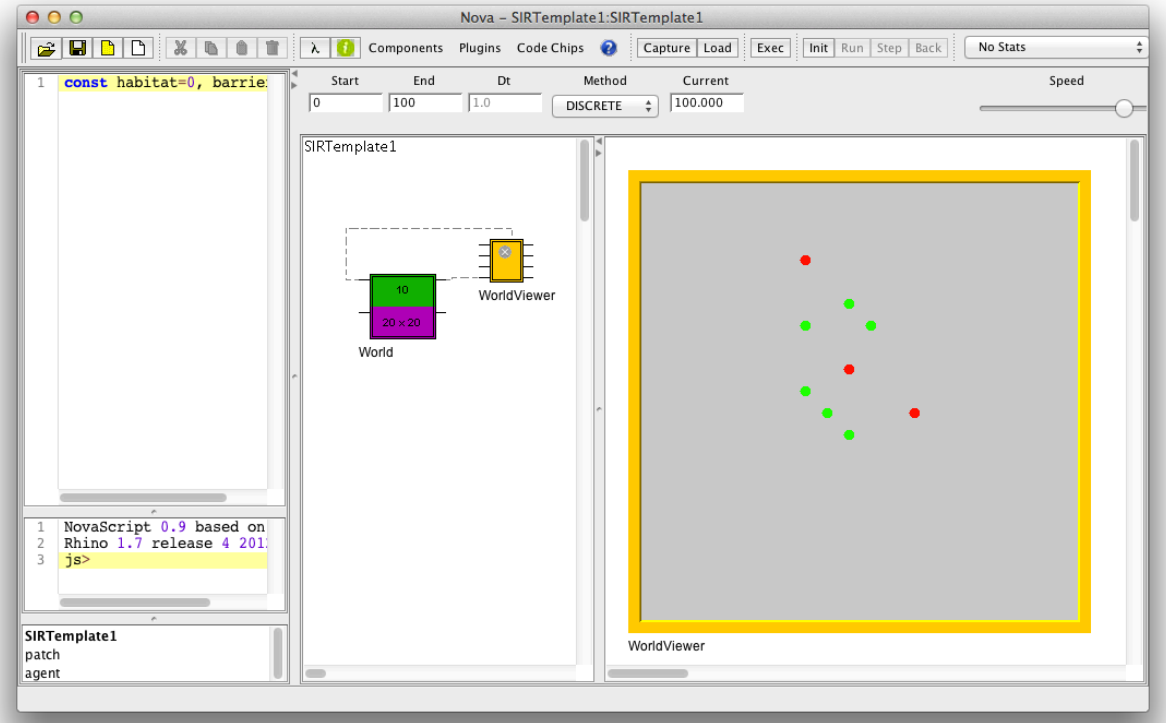
```
var newXY = RANDOM_MOVE();
MOVE(newXY);
```





IV. Testing the agents

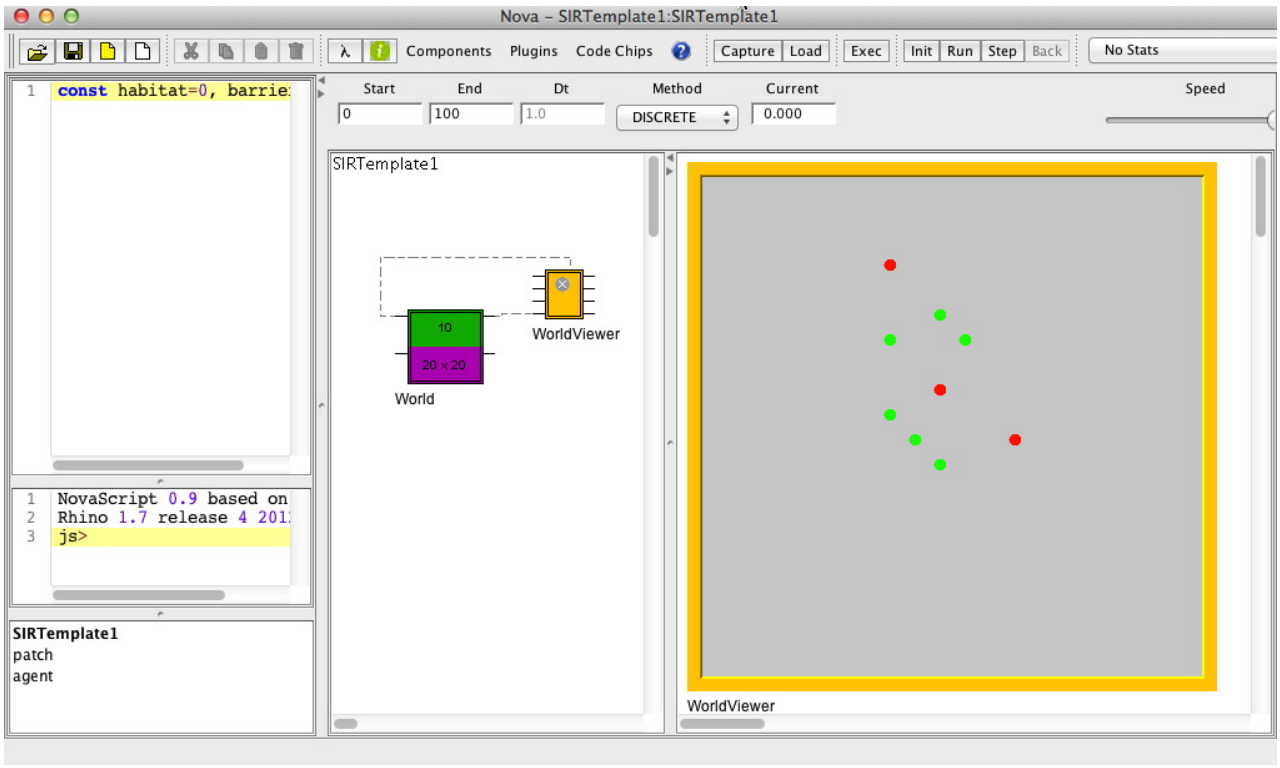
14. Return to the top level and use the interactive feature of **AgentViewerX** to add agents.
    To add agents: Shift-click the mouse on a cell in the **WorldViewer**. Continuing to shift-click will rotate the agent color from green to red, and then delete the agent. Add several green and red agents throughout the space.



15. We are now ready to run the model for the first time. Set the clock parameters to 0, 100 and select DISCRETE as the integration method. Click Capture-Load-Exec and watch the agents move randomly across the space. (You might want to lower the speed so that they don't move too fast).

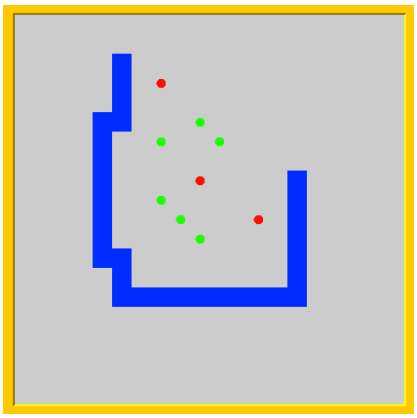
    When complete, you can click **Init** and set up a different group of agents using the mouse as before. (Right-clicking on the WorldViewer gives you the option of first resetting the space to be empty.)

    Be sure to click **Init** after any changes, before you rerun the model.



V. Adding the barrier

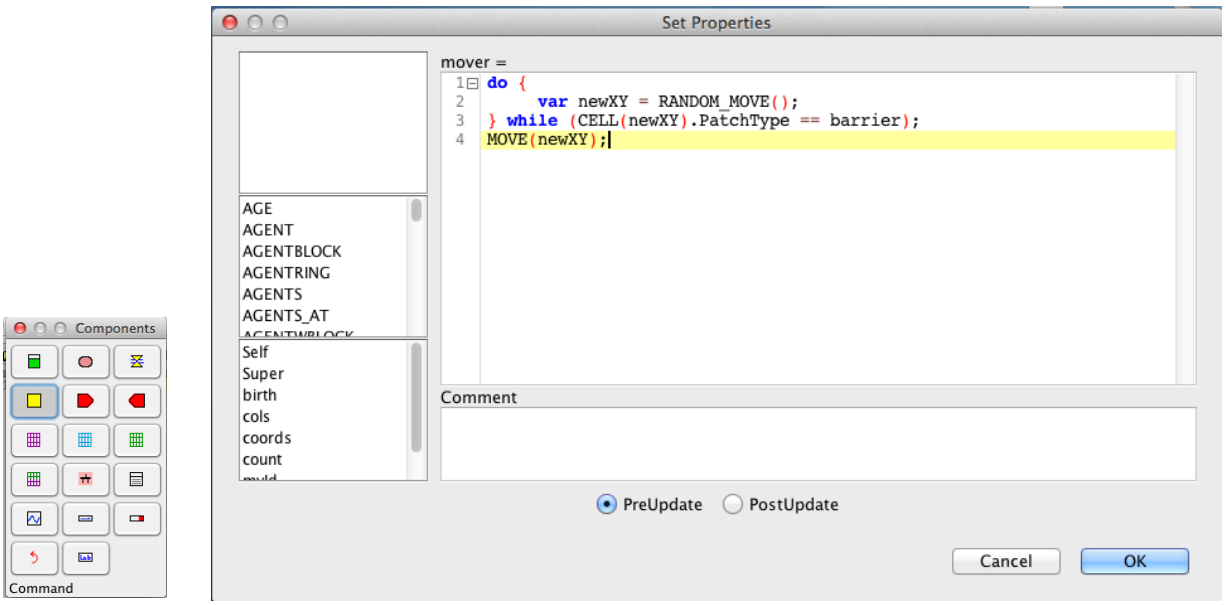
16. To add barrier cells simply click on a cell to turn it blue and drag the mouse over neighboring cells.



Note, however, that if you run the model the agents do not notice the barrier. We need to modify the move command to prevent the agents from moving into a barrier cell.

17. Revisit the agent and change the code in the mover to be the following:

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



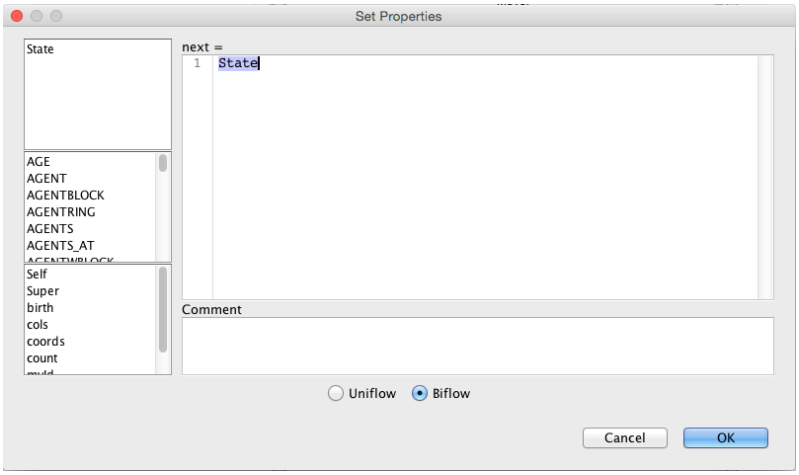
This code prevents the agent from moving into a cell that represents a barrier.

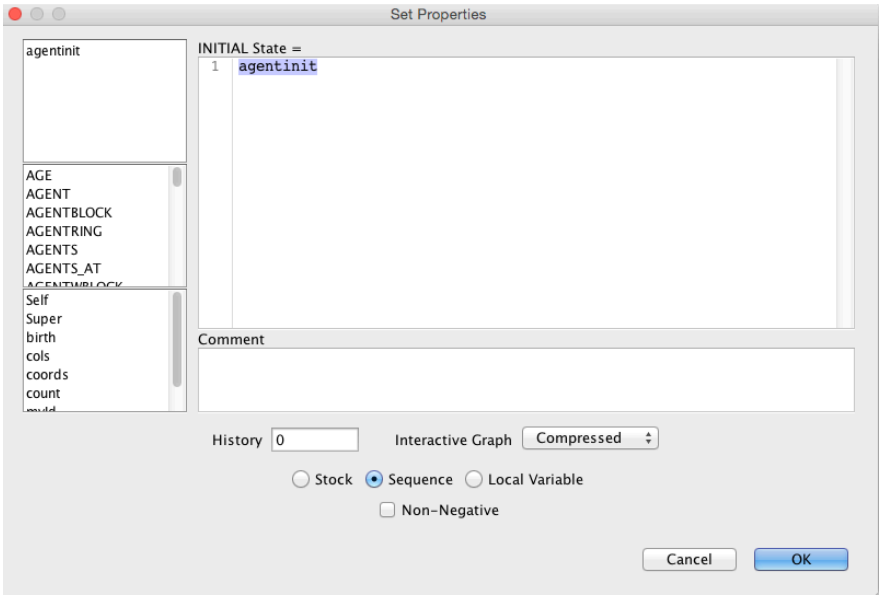
It takes the current XY location of the agent (which it determines using the LOCATION function) and randomly adds either -1, 0, or 1 to each coordinate. The result may be illegal either because it is outside the boundary or because it is a barrier patch. For that reason we have the process repeated until a new legal pair of coordinates is found. Once found, the MOVE command actually moves the agent.

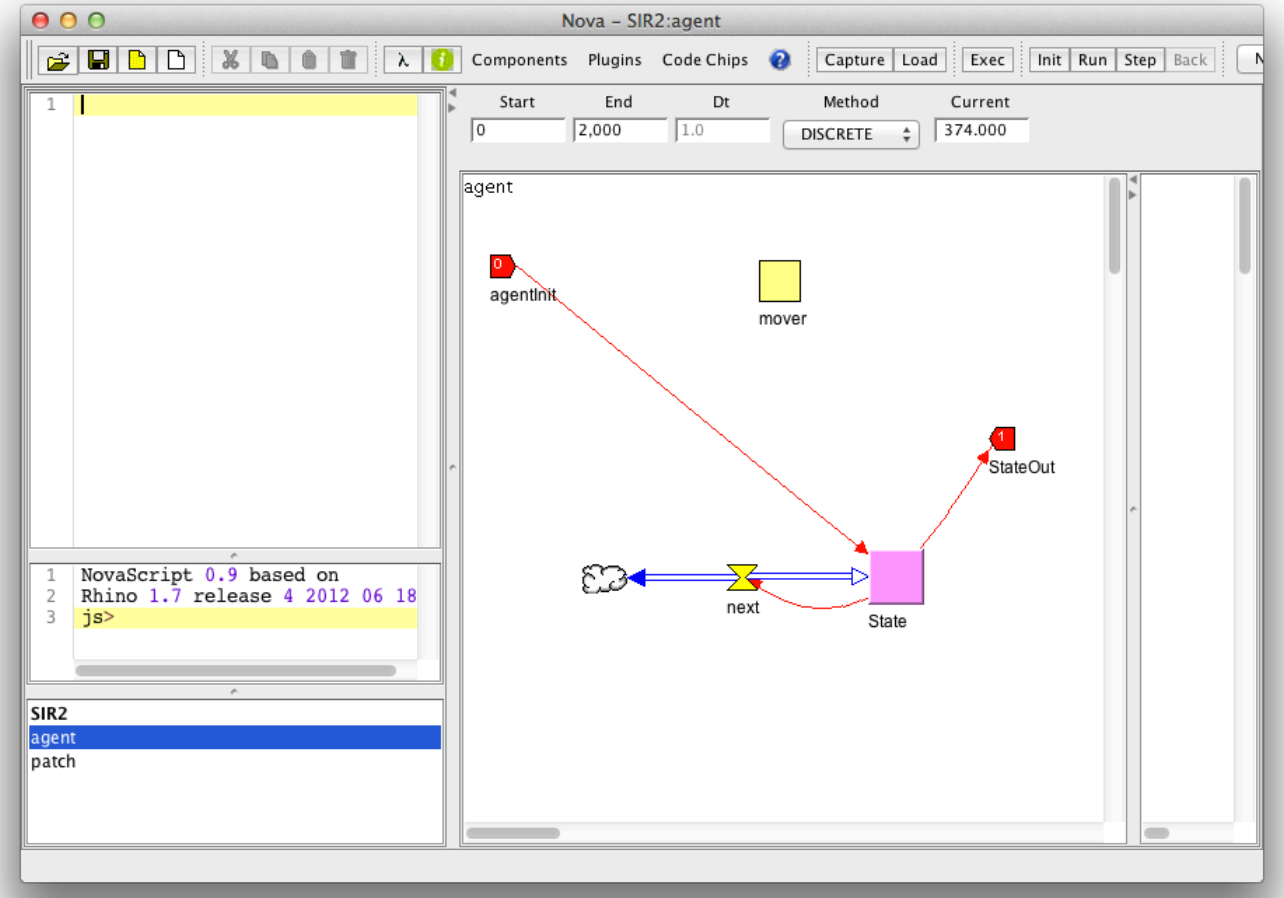
Now return to the top level and rerun the model.

VI. Add agent states

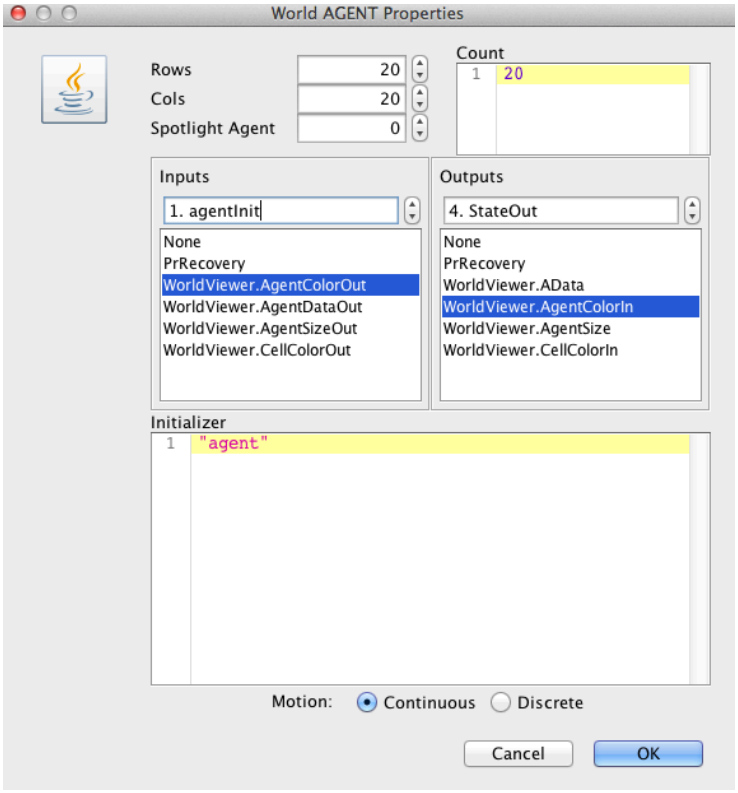
18. Return to the agent level and add components **State**, **next, agentinit** and **StateOut** as shown. Be sure that the initial value of **State** is `agentInit`, and that **StateOut** and **next** each have `State` for its component expression (this is only temporary in **next**; we'll change it in the last step). Note that the *Sequence* box of **State** has been selected. This causes the value of the Flow **next** to replace the current value in **State** rather than be added to it, which is appropriate behavior for our model




Set Properties
agentinit
INITIAL State =
1 agentinit
AGE
AGENT
AGENTBLOCK
AGENTRING
AGENTS
AGENTS_AT
Self
Super
birth
cols
coords
count
Comment
History 0
Interactive Graph
Compressed
Stock
Sequence
Local Variable
Non-Negative
Cancel
OK


Nova – SIR2:agent
Components
Plugins
Code Chips
Capture
Load
Exec
Init
Run
Step
Back
Start
End
Dt
Method
Current
0
2,000
1.0
DISCRETE
374.000
agent
agentInit
mover
StateOut
next
State
NovaScript 0.9 based on
Rhino 1.7 release 4 2012 06 18
js>
SIR2
agent
patch

Return to the top level, click on the green section of World, and make the following connections to **agentInit** and **StateOut**.



Now when you run the model, the green and red agents correspond to appropriate **State** values.

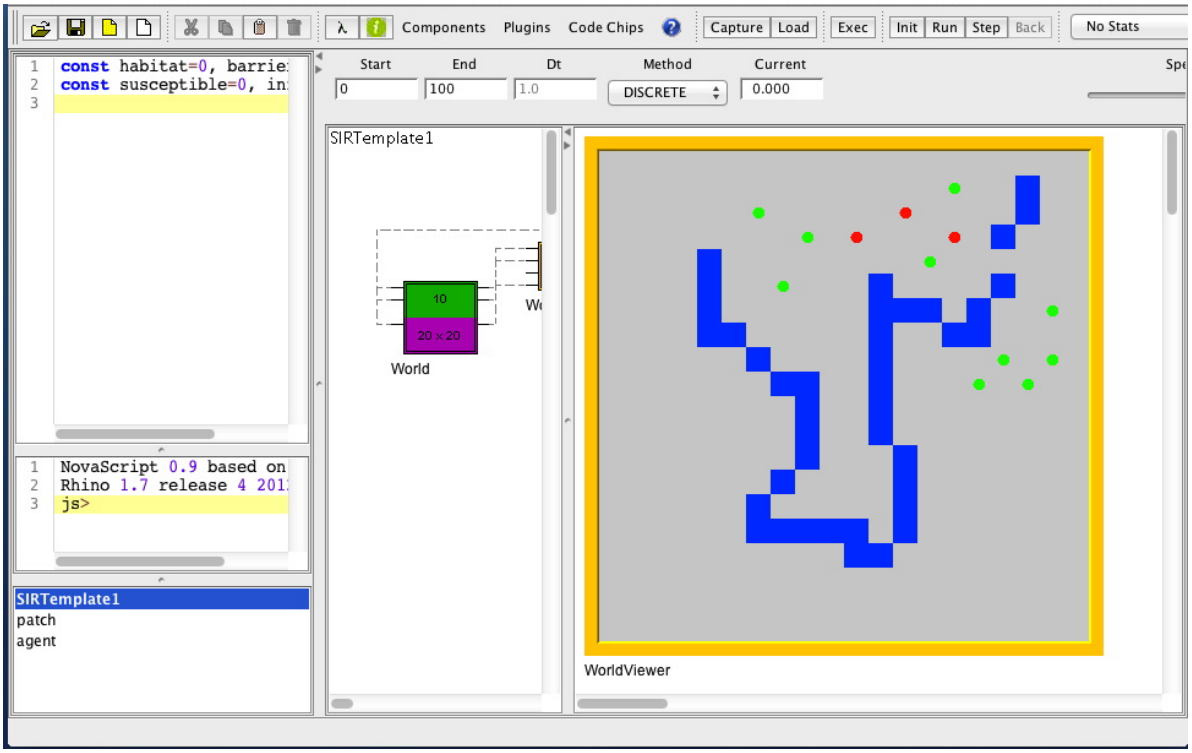
VII. Final step: adding the state change rule

19. We must determine whether or not any neighbor is infected. Return to the agent level and replace the code in **next** with the following:

```
var newState = State;
var neighbors = AGENTBLOCK(1);
if (State == susceptible) {
     if (SOME_AGENT(neighbors, "State", infected))
          newState = infected;
}
newState;
```

`AGENTBLOCK(1)` returns all agents within 1 cell of the caller. The agent checks to see if any neighbor is infected. If so, the agent becomes infected as well. If it is already infected it remains so.

- Return to the top level, click Capture-Load-Exec and see your effort pay off.



Some things to try:

- Add randomness to the infection process; i.e., infection may or may not occur when a susceptible agent encounters an infected one (hint: use RANDOM()).
- Allow an infected agent to randomly spontaneously recover.
- Put all infected on one side of the barrier and see how this affects how long it takes for all agents to become infected.
- Add more agents, keeping the same number of infected agents, and see if that makes infection occur quicker.