“Heatbugs” with NetLogo

Introduction

“Heatbugs” is a modeling and simulation exercise which has become firmly established in the canon of agent-based modeling and complexity science. The model serves as a demonstration of self-organization in a population of very simple agents. These agents are able to sense and affect environmental conditions in their immediate neighborhoods only, and interact with other agents only indirectly.

The agents in this model (the heatbugs) were reportedly inspired by actual biological organisms, although their behavior doesn't closely match that of any specific organism. Nonetheless, the behavioral rules employed by the agents in the model serve as a basic introduction to the kinds of rules employed in many agent-based models – including many that deal more thoroughly with some aspect of the behavior of biological organisms.

Behavior

Each heatbug is born with an ideal temperature – but not all heatbugs have the same ideal temperature. Over the course of the simulation, the heatbug seeks out – in a very short-sighted way – territory with a temperature matching its ideal. At the same time, the heatbug radiates some of its own body heat into the environment, warming it; the amount radiated in each unit of time is also an attribute of each individual heatbug.

The further the temperature of the local environment is from a heatbug's ideal temperature, the more unhappy the heatbug is. Thus, the central aim of each heatbug is to be as happy as possible, by repeatedly attempting to move to spaces which are closer to that heatbug's ideal temperature than the space the heatbug currently occupies. Of course, there are constraints on what the heatbug senses, and to how it moves.

In every tick (time step), each heatbug does the following:

1. Sets its happiness level, using the absolute difference between its ideal temperature and the local temperature (the temperature of the patch the heatbug is on).
2. If there is any difference in temperature, the heatbug tries to move in this fashion:
   a) Some given percent of the time (this will be essentially equivalent to flipping a coin that has a stated percentage of turning up heads) …
      - The heatbug selects a neighboring patch at random as its intended destination.

      The rest of the time …

      - If the current patch is warmer than the ideal temperature …
        - The heatbug selects the coldest patch in its immediate neighborhood as its intended destination.

        Otherwise, …

        - The heatbug selects the warmest patch in its immediate neighborhood as its intended destination.

   b) If the intended destination is not an improvement over the current patch …
      - The heatbug decides not to move after all.

   c) If the heatbug intends to move, but its intended destination is currently occupied …
      - If there are other unoccupied patches in the immediate neighborhood …
        - The heatbug selects a new destination at random from the unoccupied patches in its immediate neighborhood.

        Otherwise …

        - The heatbug decides not to move after all.

   d) If the heatbug still intends to move …
      - It moves to the selected destination patch.
3. The heatbug radiates some heat to the patch where it is currently located.

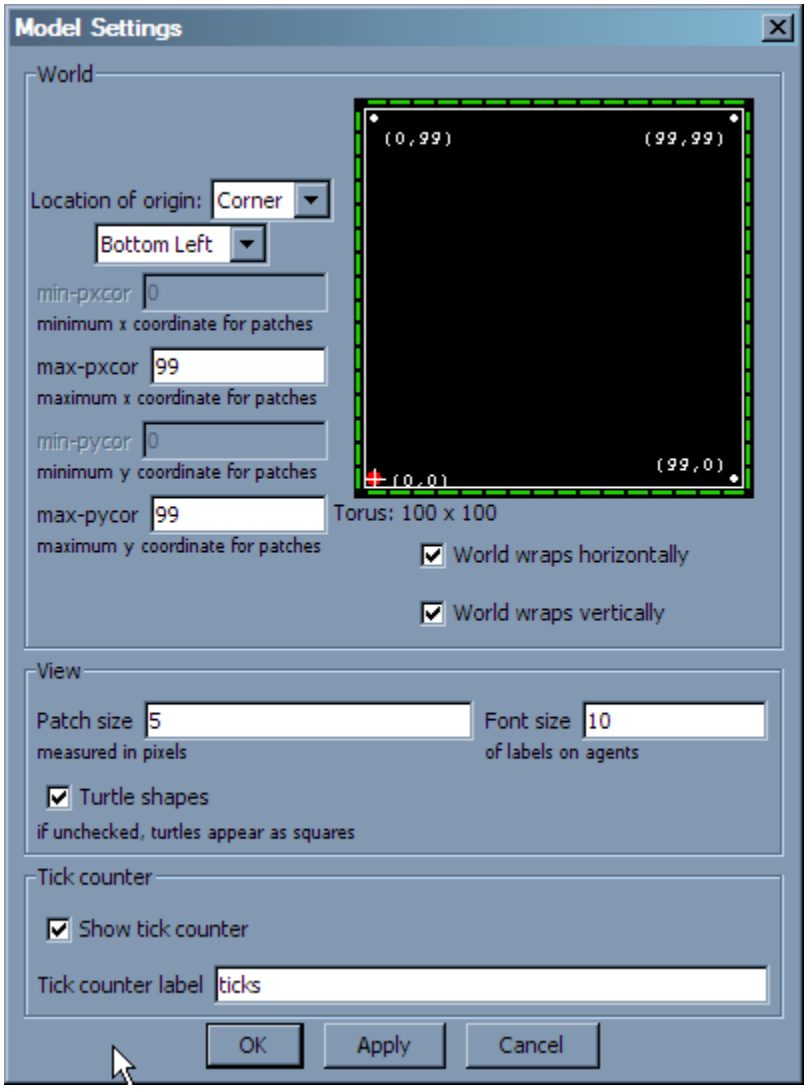
Discussion

1. How is a heatbug's behavior affected by its environment?
2. How does a heatbugs affect its environment?
3. How do the heatbugs interact with each other?
4. Have we described any behavior of the patches? Should we? In fact, patches are stationary agents in NetLogo, and there are two important things that patches will have to do as the simulation runs. What might these be? (Hint: think about heat, and what happens to heat over time.)

Task 1: Getting Started

1. Launch NetLogo v4.0.x.
2. For this model, we'll need a fairly large terrain. On the other hand, our heatbugs don't need to look like real bugs, so we can use pretty small patches. Let's go with a NetLogo world that is a torus of size 100 X 100 patches, with each patch measuring 5 pixels on a side. To do this, click the **Settings…** button, near the top of the NetLogo window, and make these changes:
   a) Change the **Location of origin** to **Corner** (why do we need to do this?), and select **Bottom Left** from the corner selection pull-down.
   b) Set the **`max-pxcor`** value to 99.
   c) Set the **`max-pycor`** value to 99.
   d) Make sure the **World wraps horizontally** checkbox is checked.
   e) Make sure the **World wraps vertically** checkbox is checked.
   f) Set the **Patch size** value to 5.0.
   g) Make sure the **Turtle shapes** checkbox is checked. (With patches this small, it's debatable as to whether it makes sense to show turtle shapes. But we can use a simple circle shape, and it will show up well even at this size.)
   h) Make sure the **Show tick counter** checkbox is checked.

The **Model Settings** window should now look like like this:



3. Click the **OK** button.
4. Adjust the size of the NetLogo window as necessary, so you can see the entire world.
5. Save your model.

Task 2: Setting Up the Model

As usual, we'll need a way to clear one simulation run, and prepare our model for the next. By convention, the procedure that does this is usually called **`setup`**, but it's important to remember that this is simply convention: **`setup`** is not a magic name, and NetLogo has no idea what **`setup`** means – until we explain what it means, by writing a procedure for it.

Before writing it, let's first review what our **`setup`** procedure should do:

1. Clear any existing heatbugs from memory.
2. Set the temperature of every patch to some default level.
3. Create a user-specifiable number of heatbugs, with the following attributes:
   a) An ideal temperature, randomly generated from a user-specifiable range.
   b) A heat radiation rate (i.e. how much heat the heatbug will put into the environment every tick), randomly generated from a user-specifiable range.
4. Place each heatbug on a random patch in the NetLogo world, with the additional condition that no two heatbugs can be on the same patch.

In case you weren't keeping count, there were three uses of the phrase "user-specifiable" in that description. And two of those were for a "user-specifiable" range – i.e. the user will specify a minimum and a maximum value. Clearly, we need a way to let the user specify the values of interest. The easiest way to do this in NetLogo is with a slider.

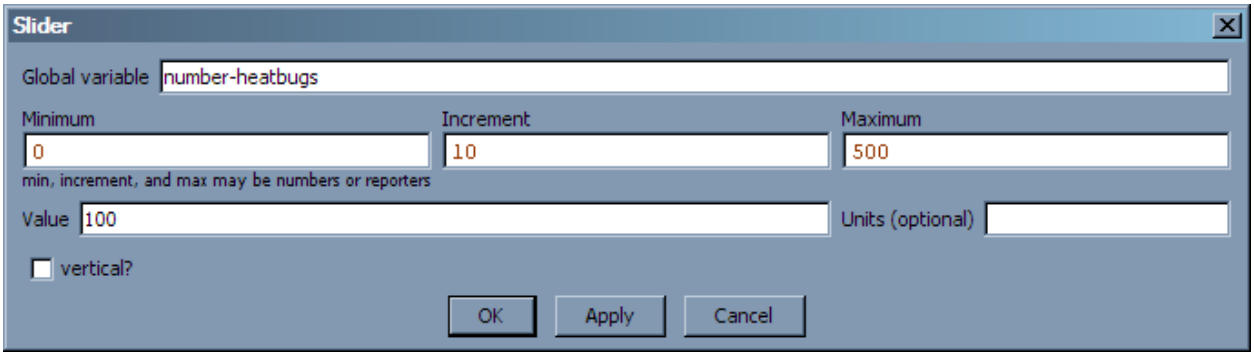
At the top of the NetLogo **Interface** tab, there's a toolbar that includes a button labeled **Add**, and a drop-down menu containing a number of user interface devices. One of these devices is a slider; you can create a new slider in one of these three ways:

- Select **Slider** from the drop-down menu (notice that the **Add** button is automatically pressed), and click somewhere in the whitespace below the toolbar.
- Click the **Add** button, then select **Slider** from the drop-down menu (if it's not already selected), and then click somewhere in the whitespace.
- Right-click in the whitespace below the toolbar, and select **Slider** from the pop-up menu that appears.

For the first of the four sliders we need (for now), do the following:

1. Create a slider, using your method of choice. This slider will be used to specify the desired number of heatbugs.
2. Make the following changes in the **Slider** dialog:
   a) In **Global variable**, type **`number-heatbugs`**.[1]
   b) For **Minimum**, specify 0.
   c) For **Increment**, specify 10.
   d) Set **Maximum** to 500. (According to the restrictions on heatbug placement and movement described previously, and given the dimensions we specified for the NetLogo world, what is the absolute maximum number of heatbugs that can be created?)
   e) **Value** is where the initial value of the slider is specified. For now, specify 100.

   The Slider dialog should now look something like this:



3. Click **OK**. One slider down, four more to go (for now, anyway)!

---

1 Notice that there's a dash between the two words, but no spaces. When creating a slider, you're declaring a global variable that can be referenced in the procedures of the model (and in other user interface elements). NetLogo variable and procedure names can't contain spaces, but they can contain letters, numbers, and some punctuation characters. The fact that many punctuation characters have mathematical or other special meanings is often a source of confusion and programming errors for new NetLogo programmers. Thus, apart from **–** (dash) and **?**, which have well-established usage conventions, you should generally avoid using punctuation characters in this way.

4. Create four more sliders, using the settings given in the table that follows.

Global variable	Minimum	Increment	Maximum	Value
min-ideal-temp	**0**	**1**	**200**	**100**
max-ideal-temp	**min-ideal-temp**	**1**	**200**	**110**
min-radiant-heat	**0**	**1**	**100**	**20**
max-radiant-heat	**min-radiant-heat**	**1**	**100**	**30**

Notice that **min-ideal-temp** appears as the **Global variable** name for one slider, and the **Minimum** value for another; the same is true for **min-radiant-heat**. In each of these cases, one slider is taking its minimum value from the current value of another slider. In fact, **Minimum**, **Increment**, and **Maximum** can be specified as numeric literals, variable names (where those variables are assumed to hold numeric values), or more complicated expressions that return numeric results. If this seems a bit confusing, experiment with moving the **min-ideal-temp** and **min-radiant-heat** sliders (i.e. changing the values of those variables), and see what happens to the **max-ideal-temp** and **max-radiant-heat** sliders when you do so; this should make things a bit more clear.

You should now have five sliders. If you do, and if none of the slider names and values are appearing in red (which would indicate that there is an error in one or more of the slider settings – probably due to inadvertently including a space in the value typed for **Global variable**, or incorrectly spelling one of the variable names referred to in **Minimum**), then you're ready to write the **setup** procedure code.

5. Switch to the **Procedures** tab and write the following code:

```
breed [ heatbugs heatbug ]

heatbugs-own [
    ideal-temp
    radiant-heat
]

patches-own [
    temperature
]

to setup
    clear-all
    set-default-shape heatbugs "circle"
    ask n-of number-heatbugs patches [
        sprout-heatbugs 1 [
            set color red
            set ideal-temp (min-ideal-temp
                + random-float (max-ideal-temp - min-ideal-temp))
            set radiant-heat (min-radiant-heat
                + random-float (max-radiant-heat - min-radiant-
heat))
        ]
    ]
end
```

6. Let's review the code:

   a) First, we declare that we'll be using an agent breed called **`heatbugs`** – in addition to or instead of the default **`turtles`** breed. For those NetLogo statements that expect a singular form of the breed name, the second name specified – **`heatbug`** – will be used.

   b) The **`heatbugs-own`** statement tells NetLogo that each agent in the **`heatbugs`** breed will have two variables (in addition to those defined automatically by NetLogo): **`ideal-temp`** and **`radiant-heat`**.

   c) **`patches-own`** is similar to **`heatbugs-own`**, in that it declares new agent variables. However, the variables declared in patches-own belong to the patch agents. In this case, we're telling NetLogo that each patch will have a **`temperature`** variable.

d) Next, we have the **setup** procedure itself, which includes the following instructions:

   i. The **clear-all** command removes any turtle or custom breed agents, and clears all variable values – including patch variables, like the **temperature** variable we created. When such variables are cleared, their values are set to zero (0); this is fine for our purposes now, but in many cases we would need to include code to set the initial values of our patch variables.

   ii. **set-default-shape** is used to specify that the default shape for the **heatbugs** breed will be the shape named "circle". (You can see the shapes that are available – and modify those shapes or create new ones – with the **Tools/Turtle Shapes Editor** menu command.)

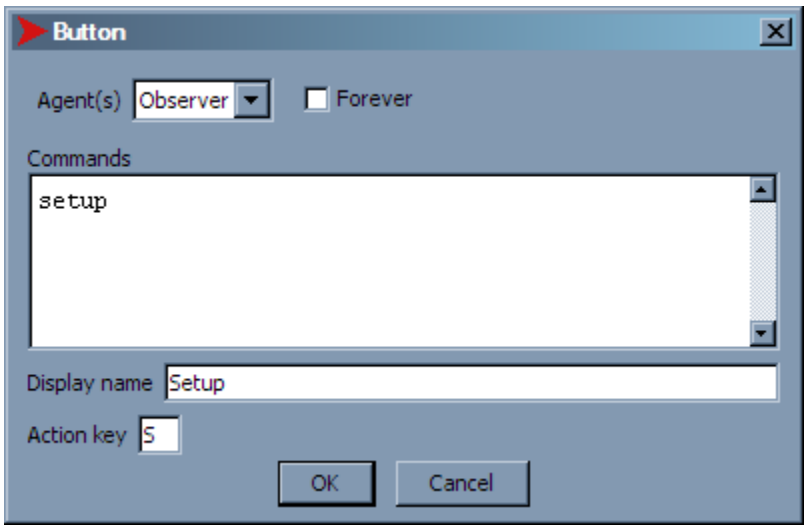
   iii. Rather than create some number of agents, and scatter them randomly – and then have to check to make sure that no two landed on the same patch – this **setup** procedure turns the logic around a bit: it uses **n-of number-heatbugs patches** to select **number-heatbugs** of the **patches** agentset at random. Then, it asks each of these selected patches to **sprout** a single agent of the **heatbugs** breed on the center of the patch. This guarantees that we will have the correct number of distinct patches, each with exactly one **heatbug** agent, with no possibility of two **heatbugs** starting out on the same patch.

   iv. Each of the newly created **heatbugs** sets its color to red, and then uses the current values of the sliders we previously created to generate random values and assign them to **ideal-temp** and **radiant-heat**.

7. Check the syntax of your code by clicking the **Check** button in the toolbar at the top of the **Procedures** tab. If there are errors, an error message will appear just under the toolbar, and the cursor will be placed on the line where the error was found. Try to use the error message, and the knowledge of where the error occurred, to fix your code.

8. When your code is free of syntax errors, return to the **Interface** tab.

9. Create a button to run your setup procedure. The **Button** properties dialog should look something like this (**Display name** and **Action key** are optional; the other values and selections are required):



   Make sure that you have **Observer** selected in the **Agent(s)** drop-down, and that the **Forever** checkbox is *not* checked.

10. Click **OK**.

11. Once you've corrected any error conditions, save your model.

12. Click the **Setup** button. You should see a bunch of red circles, distributed randomly around on the NetLogo world.

Task 3: Making the Heatbugs Move

Now we'll implement the behavior of the heatbugs. There are many ways to write this behavior in NetLogo code – we only have time to explore one approach, but you're encouraged to experiment with others.

First, remember that we need to allow for the heatbug moving randomly a user specifiable fraction of the time. This sounds like a job for another slider.

1. Create a slider in the **Interface** tab, with these attribute values:

Global variable	Minimum	Increment	Maximum	Value
random-move-prob	**0**	**0.01**	**1.00**	**0.20**

2. Now we need to create a **move** procedure. Instead of copying the code that follows, try to create at least some portion of it yourself. Remember to create a new procedure, preceding the name with the **to** keyword, and finishing with the **end** keyword. And remember: procedures can't be nested in NetLogo, so make sure you don't start your new procedure inside the **setup** procedure you already wrote.

   If you get stuck, or have written as much of the procedure as you feel comfortable writing without help, check your work against the following, and fill in the gaps as necessary:

```
to move
    if (temperature != ideal-temp) [
        let destination nobody
        ifelse ((random-float 1) < random-move-prob) [
            set destination one-of neighbors
        ] [
            ifelse (temperature < ideal-temp) [
                set destination max-one-of neighbors [temperature]
                if (([temperature] of destination) < temperature) [
                    set destination nobody
                ]
            ] [
                set destination min-one-of neighbors [temperature]
                if (([temperature] of destination) > temperature) [
                    set destination nobody
                ]
            ]
        ]
        if ((destination != nobody)
                and (any? heatbugs-on destination)) [
            set destination
                (one-of neighbors with [not any? heatbugs-here])
        ]
        if (destination != nobody) [
            move-to destination
        ]
    ]
    set temperature (temperature + radiant-heat)
end
```

3. Rather than review the code line by line, let's review some key NetLogo language statements and structures seen in the code:

   a) **`if`** is used to execute a block of statements only if a stated condition is true. The **`if`** keyword is always followed immediately by a conditional expression, which must evaluate to **`true`** or **`false`** (parentheses are optional around the conditional expression). After the conditional expression comes a block of statements, enclosed in a pair of square brackets: **`[…]`**; these statements will be executed only if the conditional expression is **`true`**.

   b) **`ifelse`** is similar to **`if`**, with one key difference: instead of one statement block after the conditional expression, there are two. The statements in the first block are executed if the conditional expression is **`true`**; the statements in the second are executed if the condition expression is **`false`**.

   c) **`neighbors`** is a built-in reporter (i.e. a statement that computes and returns a value) which returns the agentset consisting of the eight patches adjacent to (or diagonal to) the patch where the current agent is located.

   d) **`max-one-of`** is a reporter that computes the value of an expression (given in brackets) for each agent in a specified agentset, and returns the agent in the agentset that has the largest value for the computed expression.

   e) **`min-one-of`** returns the agent in the agentset that has the smallest value for the computed expression.

   f) **`one-of`** simply returns a single agent, selected at random from a specified agentset. (In the **`setup`** procedure, we used **`n-of`** to select a given number of agents at random from a specified agentset; we can think of **`one-of`** as a special case of **`n-of`**, where $n = 1$.)

   g) **`nobody`** is a special value which is returned by some reporters to indicate that no agent satisfied the specified criteria. We can also assign the value **`nobody`** to our own variables in our code, to indicate that such a variables does not refer to any agent.

   h) **`any?`** is a reporter which returns a value of **`true`** if there are any agents in the specified agentset, and a value of **`false`** if there are none.

   i) **`with`** is an operator that filters a specified agentset by some criteria (in the brackets that follow **`with`**), and returns a new agentset, containing all of the agents in the first agentset that satisfy the criteria.

j) **-here** and **-on** are suffixes that filter a breed agentset, returning only those agents on the current or specified patch, respectively.

k) Indentation is not required in NetLogo, nor is it interpreted in any fashion by NetLogo. However, it can be useful to the programmer, in keeping clear the logical structure of code.

l) In some instances, parentheses are required – either to force a desired order of operations on a computation, or to help NetLogo understand statements that can have a variable amount of information associated with them. However, parentheses are often used primarily to make explicit (for the programmer's benefit) the logical structure of an expression.

m) Since many punctuation characters are allowed in NetLogo variable and procedure names, we have to be careful when we want to use those characters for their mathematical, logical, or structural purpose. For example, **a + b** is an expression that NetLogo will try to evaluate by taking the value of **a**, adding it to the value of **b**, and returning the result. However, **a+b** (without spaces around the plus sign) is interpreted by NetLogo to refer to a variable with the name **a+b**, and not to the sum of the variables **a** and **b**.

n) Use the above summaries, along with the NetLogo Dictionary, to help you read and understand the **move** procedure.

Task 4: Dissipating and Diffusing Heat

Several pages back, we hinted that we would need to have the patches do some processing in our model, and not simply serve as heat buckets. Let's consider two simple questions:

1. What would happen if every heatbug kept radiating heat into the environment, and that heat simply accumulated in the patches?
2. In the absence of some insulating structure, should we expect that heat radiated into one patch will remain there, and not spread – by convection or conduction – to the neighboring patches?

Obviously, we need to do something so that the heat doesn't continue to build up without limit, and so that the heat can flow between the patches. Fortunately, NetLogo makes it relatively easy to take care of both of these concerns. First, the **diffuse** statement can be used to spread some quantity stored in a patch variable around between the patches, with the end effect being that the differences between the patches tend to smooth out. Second, patches are themselves stationary agents: we can ask them to perform tasks in the same way that we ask turtles to perform tasks.

Before we write code the for our patches, there's some information we need that we don't yet have. How quickly should heat diffuse between patches? How quickly should heat dissipate from the system?

In this case, the easy answer is also a pretty useful one: let's make these experimental parameters of our model. We can add two more sliders, to control the rates of heat diffusion and heat dissipation. The user will then be able to set the sliders as desired, and see how the heatbugs respond.

1. Switch to the **Interface** tab.
2. Create two more sliders, with the following attributes:

Global variable	Minimum	Increment	Maximum	Value
dissipation-rate	**0**	**0.01**	**1.00**	**0.10**
diffusion-rate	**0**	**0.05**	**1.00**	**0.70**

3. Now we need to write the **go** procedure. This procedure will use NetLogo's built-in **diffuse** statement to diffuse the heat across the patches; then it will ask the patches to dissipate some of their heat; finally, it will ask the heatbugs to move.

   Write as much as you can of the **go** procedure, before checking your code against the following:

```
to go
    diffuse temperature diffusion-rate
    ask patches [
        set temperature (temperature * (1 - dissipation-rate))
    ]
    ask heatbugs [
        move
    ]
    tick
end
```

4. You should be able to read this procedure, and match it – virtually line for line – against the previous description of what the procedure should do. The only part that wasn't mentioned before was the **tick** statement. **tick** updates the NetLogo time tick counter, and – depending on the model settings – tells NetLogo to update the display.
5. Save your model.

Task 5: Running the Simulation

1. Switch to the **Interface** tab.
2. Create a **Go** button. This button should call the **`go`** procedure, and should be a "forever" button. (Based on the code of the **`go`** procedure, should the **Go** button be an Observer button, a Turtles button, a Patches button, or a Links button?)
3. Make sure the **View updates** checkbox, in the toolbar at the top of the **Interface** tab, is checked.
4. Select **on ticks** from the drop-down menu below **View updates**.
5. Save your model.
6. Click the **Setup** button.
7. Click the **Go** button. What happens?
8. While the model is running, experiment with different values of the **`random-move-prob`**, **`dissipation-rate`**, and **`diffusion-rate`** sliders. Do you notice any changes in the aggregate behavior of the heatbugs?
9. Try setting up and re-running the simulation with different values of the ideal and radiant heat range sliders. What changes in aggregate behavior do you notice?

Task 6: Making Heat Visible

Sometimes we choose to incorporate strictly visual elements in a model, even when they don't have any effect on agent behavior, and even if they don't present output data in a form we can easily collect and analyze. Such elements can be useful in helping us, as modelers and programmers, understand what's going on as the simulation is running. They can also enrich the explanatory power of a model that will be shown to others.

In this model, we don't yet have an easy way to see what's going on in the environment. We know that the heatbugs are reacting to the heat in the environment, and we know that they're adding heat to the environment, but we're only able to observe this heat indirectly, through the heatbugs' actions. Let's do something about that now.

One commonly used approach for visualizing data that varies across a landscape is to color that landscape according to the data. NetLogo supports this approach by allowing us not only to set the patch colors, but also to scale them as shades of a basic color, using patch data values to control this scaling.

For our model, let's use shades of violet, with the darkest shades representing little or no heat, and the lightest shades (nearly white) representing a lot of heat. But how much is a lot? We generally won't know, in advance, how hot the hottest patches might get. How should we calibrate our shading, so that we use the full range of the shades of violet, without having some patches that turn out to be hotter those that we've already colored with the lightest shade?

Arguably, the most direct way to solve this problem would be to find the highest patch temperature after each tick, use the lightest shade for that value, and make the shade proportionately darker for patches with lower temperatures. In fact, the NetLogo code for this approach is very simple. For example, the expression **`max [temperature] of patches`** uses just a few words to get the key piece of information we need: the highest temperature on the terrain. (This approach isn't necessarily the method with the best performance, but it will do for now.)

After we know the highest temperature, there's another built-in NetLogo reporter that makes it easy to scale colors according to data values. With the **`scale-color`** reporter, we specify a basic color, an expression that will be used for scaling, a minimum value, and a maximum value. Based on where the value of the expression falls, between the minimum and maximum, a color somewhere between the darkest and lightest shades of the specified color will be returned.

Let's try it!

1. Create a new procedure with the following code:

```
to update-color
    let high-temp (max [temperature] of patches)
    ask patches [
        set pcolor (scale-color blue temperature 0 high-temp)
    ]
end
```

2. Modify your **`go`** procedure by inserting a new line above the line with the **`tick`** statement, and typing **`update-color`** in the new line.
3. Check your code.
4. Save your model.
5. Run the simulation. What do you see?

Task 7: Plotting Output Data

Some models are purely exploratory, with no real expectation of useful results; some are even built purely for entertainment. We might not invest much effort into capturing or presenting output data from these types of models – at least, not at first. But in general, we should be thinking, starting fairly early in the modeling process, about the kind of output that we would like to capture from our simulation models. "Capture" is being used in a broad sense here: we might simply want to generate a plot, with no intention of analyzing the underlying data any further; we might want to capture each data point in detail, so that we can crunch the numbers with statistical software; we might simply want to grab frames from the simulation display, and string them together in a QuickTime movie. In any case, the general question is this: How do we want to examine and present the behavior of the simulated world we've built?

NetLogo lets us capture and present simulation output in a wide variety of ways. One of the most powerful and direct ways is to plot it on a graph – which is just what we'll do now. But what should we plot? What sort of measurements do you think would be interesting? What do you think is possible?

In the fundamental description of a heatbug's behavior, there's something that might sound a little bit silly: heatbugs are motivated by happiness – more specifically, they're always attempting to reduce their unhappiness. Given that, it might be interesting to plot the average unhappiness of the entire population over time.

Just as we were able to find the maximum heat level in the terrain with a single line of code, the average value of a turtle variable is easy to compute. But here we have a complication: we're implicitly using the concept of unhappiness in the **`move`** procedure, but we're not storing each heatbug's level of unhappiness in a variable; without that, computing the average is a bit more complicated. We could deal with this in a few different ways; we'll do it by defining and setting the value of a new turtle variable.

1. Find the **`heatbugs-own`** statement, near the top of the **Procedures** tab. To the two variables already listed within the square brackets, add a third: **`unhappiness`**.
2. Insert a new line at the start of the **`move`** procedure, just after the line that reads **`to move`**. In the new line, write the following code:

```
    set unhappiness (abs (temperature - ideal-temp))
```

3. In the **`if`** statement that immediately follows the new line, change

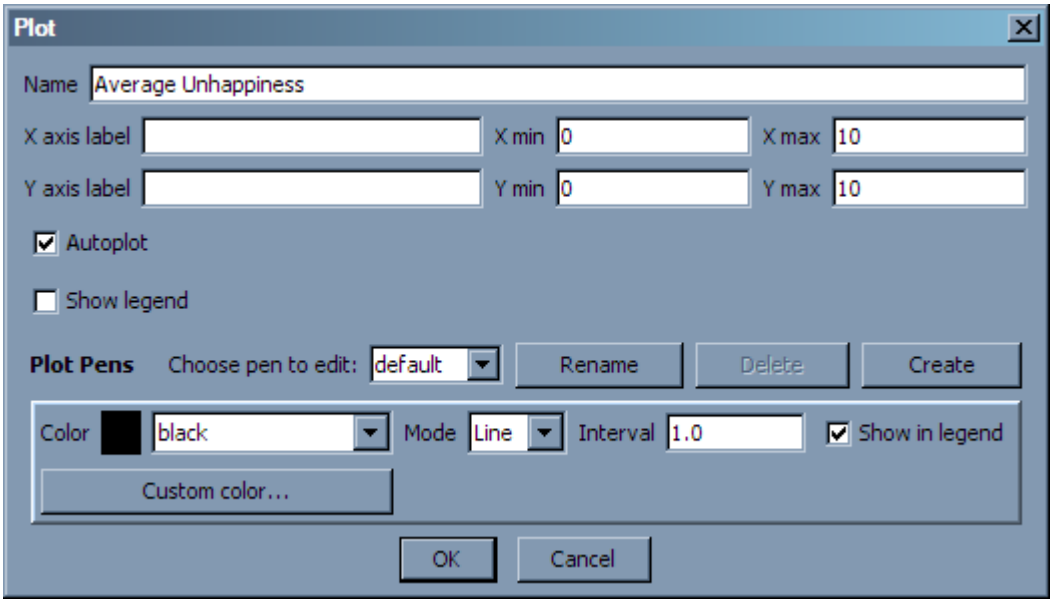
```
    if (temperature != ideal-temp) [
```

   to

```
    if (unhappiness > 0) [
```

4. Check your code.
5. Save and test your model; the aggregate behavior should be unchanged.
6. Switch to the **Interface** tab.
7. Add a plot to the user interface, following the same steps used for adding sliders and buttons – but select **Plot** from the menu, instead of **Slider** or **Button**.
8. Name the plot "Average Unhappiness". (Note that the name of a plot is like the name of a shape: it's not a variable or procedure name – and in our code, it's always specified in quotes. Thus, a plot name can have spaces in it.)
9. Check the **Autoplot** checkbox.
10. Uncheck **Show legend**.

    The **Plot** properties dialog should look something like this:



11. Click **OK**.
12. Switch back to the **Procedures** tab.

13. Create a new procedure after the end of the existing code, with the following code:

```
to update-plot
    set-current-plot "Average Unhappiness"
    plot mean [unhappiness] of heatbugs
end
```

    (In general, when there's only one plot in a model's **Interface** tab, there's no need to use **`set-current-plot`**. However, it's a good habit to get into, since your code will require fewer changes if you should add additional plots.)

14. Modify the **`go`** procedure by adding a single line just before the **`tick`** statement. Type **`update-plot`** in this new line.

15. Check, save, and run.

16. What do you observe about the average unhappiness?

17. What happens to the average unhappiness if you change one or more of **`random-move-prob`**, **`dissipation-rate`**, and **`diffusion-rate`**, while the model is running?

Discussion

1. How would you characterize the types of aggregate behavior the heatbugs display?

2. Did you notice any qualitative changes to the aggregate behavior with different values of the input parameters?

3. Did displaying the temperature in the patches via color coding make it easier for you to observe the heatbugs' behavior, or was it a distraction?

4. Did the plot of average unhappiness show you what you expected to see? In other words, did you anticipate well what this plot would look like?

5. In every tick, the model is reviewing the temperature in all 10,000 patches, to find the hottest spot. What do you think will happen if we expanded the model to 500x500 (250,000 patches)? Can you think of any other way to find the highest temperature in the terrain?

6. No heatbug is allowed to move onto a patch where another heatbug is already resident. However, it's possible that the other heatbug will actually move from that patch before processing of that tick is complete. Should we deal with this issue in our models? If so, how? (Keep in mind that NetLogo shuffles the agentset before processing an **`ask`**; thus, the order in which the heatbugs move changes each time.)

Optional Tasks

1. What do you think will happen if you perturb the heatbugs system, while the model is running, by suddenly heating or cooling the entire world by a large amount? How do you think that will show up in the average unhappiness plot?

   Add a **Heat Up** and/or a **Cool Down** button, with the necessarily additions or changes to the **Procedures** tab contents, to test your hypothesis.

2. Can you think of any ways to make the heatbugs' movement a bit "smarter", while still limiting visibility and interaction to their immediate neighborhoods? What changes do you think this will produce in the aggregate behavior?

   Implement and test your modified heatbug movement behavior. Do the results match what you expected?