8 Emergence

8.1 Introduction and Objectives

The most important and unique characteristic of ABMs is that complex, often unexpected, system dynamics emerge from how we model underlying processes. Emergence, therefore, is the most basic concept of agent-based modeling. The key question about emergence is this: what dynamics of the system and what model outcomes *emerge*—arise in relatively complex and unpredictable ways—from what behaviors of the agents and what characteristics of their environment? What other model dynamics and outcomes are instead *imposed*—forced to occur in direct and predictable ways—by the model's assumptions?

By "unpredictable" here we refer to outcomes that are difficult or impossible to predict just by thinking. The concept of emergence has sometimes been given mystic connotations such as being unexplainable in principle, but with ABMs we are focusing on just the opposite: on explaining things via simulation. Epstein and Axtell (1996) described this kind of explanation aptly with their famous question "Can you grow it?": can you make your model system look and behave like the real one?

There is no simple way to classify some particular outcome of an ABM as emergent vs. imposed, but we can think about these qualitative criteria for a model result being emergent:

- It is not simply the sum of the properties of the model's individuals;
- It is a different type of result than individual-level properties; and
- It cannot easily be predicted from the properties of the individuals.

As an example, we can address this emergence question for the Butterfly model. The first step is to identify the specific model outcomes we are considering. The specific numerical results of the Butterfly model that we have been focusing on are the widths of corridors that groups of hilltopping butterflies use as they migrate. These outcomes result, obviously, from the one decision that model butterflies make: whether to move uphill or randomly at each time step. Do the Butterfly model's corridor width results meet the three criteria for being emergent? Corridor width is not simply the sum of individual butterfly properties, and is a different type of property from the properties of individuals. The individual butterflies have only one property, their location, and the width of a migration corridor is a different property than the location of an individual. (And remember that we discovered in chapter 5 that butterfly corridors changed qualitatively when we switched from a simple cone landscape to a realistically complex landscape, so corridor width emerges from the environment as well as from butterfly behavior.)

But we must admit that the corridor width of a group of butterflies is, in a way, just the sum over time of where the individual butterflies were. Is corridor width easily predicted from the properties of the individuals? Not completely, but even without the model we would expect corridor width to decrease as the value of the parameter q (the probability of the butterfly moving directly uphill instead of randomly) increases, and to get very large as q approaches zero. So we could describe the butterfly migration model as having its key outcome emerge from the individual decision of whether to move uphill or randomly (and from the landscape), but because this behavior is so simple, there is at least one model result—how corridor width varies with q (figure 5.3)—that is quite predictable.

The extent to which ABM results are emergent can vary widely, and having more emergence does not necessarily make a model better. The best level of emergence is often intermediate. Models with highly imposed results are often not very interesting, and sometimes their problem can be solved with a simpler mathematical model. On the other hand, a model with too many of its results emerging in complex ways from too many complex individual behaviors and mechanisms can be very hard to understand and learn from. The Butterfly model is not a bad ABM just because some of its results are not surprising; it is a perfect model for the problem it was designed for, but that problem did not involve highly complex agent behavior.

Learning objectives for this chapter are to:

- Develop an understanding of what makes the results of ABMs more vs. less emergent, by looking at two of NetLogo's library models.
- Start using the most important technique in agent-based science: designing and analyzing simulation experiments to test hypotheses and develop understanding about how an ABM works and how well it reproduces the system it models.
- Master the use of BehaviorSpace, NetLogo's extremely useful tool for automatically running simulation experiments and producing output from them.
- For the first of many times, analyze output produced by NetLogo models by importing results to other software for graphing and statistical analysis.

8.2 A Model with Less-Emergent Dynamics

The Biology category of NetLogo's Models Library includes a model called Simple Birth Rates. It is indeed a very simple model, designed to simulate how the difference in birth rate between two co-occurring species (red and blue turtles) affects the number of individuals in each population. The two species differ only in the number of offspring each individual produces each time step; the probability of death is the same for both species and is adjusted so that the total number of individuals is approximately constant. (If you study business, think of these as two chains of fast-food restaurant that go out of business at the same rate but differ in how often they open new shops.) Open this model, read its Information tab, and play with it a bit.

Let us focus on one question that this model can address: how does the time until "extinction" of one species depend on the relative birth rates of the two species? If we hold the birth rate of the red species (global variable `red-fertility`) constant at 2.0 and vary `blue-fertility` from 2.1 to 5.0, how does the number of ticks until extinction of the red species change? In other words, what shape do we expect the graph at figure 8.1 to have?

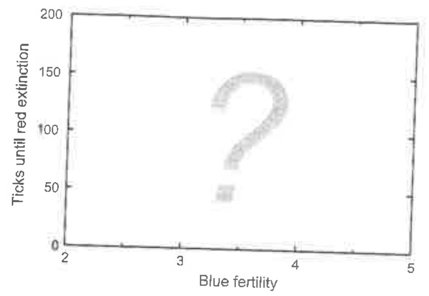


Figure 8.1
The problem we address with the Simple Birth Rates model: what shape will this graph have?

This model includes a button and procedure called `run-experiment` designed to help with this problem: it runs the model until one species is extinct, tells us how long it took, and then starts another run.

If we think about this problem for a minute, we can form a pretty strong expectation of the answer. When birth rates of the two species are close, they should coexist for a long time. As the difference in birth rate increases, the time to red extinction should decrease rapidly. But no matter how rapidly the blue reproduce, it seems unlikely that the reds will go extinct immediately (the model is initialized with 500 of them), so the time to red extinction will probably never get as low as one time step. We thus expect the curve on figure 8.1 to start high when `blue-fertility` is 2.1, drop rapidly, then flatten out at a value above 1 as `blue-fertility` increases.

Are these expectations correct? To find out, we need to do a simulation experiment.

8.3 Simulation Experiments and BehaviorSpace

To see how the time to extinction of red turtles in the Simple Birth Rates model varies with the fertility parameter of blue turtles, we need to run the model over a wide range of `blue-fertility` values and record the time (tick number) at which the number of red turtles reaches zero. But there is a complication: this model, like most ABMs, is stochastic, producing different results each time we run it because the turtles that die each tick are chosen randomly (see the procedure `grim-reaper`). (We will look more at stochasticity in chapter 15.)

Because the model produces different results each run, to really understand it we need to estimate the average results and the variability around those averages. Modelers often do this the same way that scientists study real systems that are highly variable: by executing *experiments* that include *replicates* of several different *scenarios* (statisticians often use the word "treatment" as we use "scenario"). In simulation modeling, a scenario is defined by a model and one set of parameters, inputs, and initial conditions; if the model had no stochastic elements, it would produce exactly the same results each time it executed the same scenario. Replicates are model runs in which only the stochastic elements of the model change. To be specific, *replicates* usually refer to model runs among which only the random numbers differ. (There are also other ways to replicate model scenarios, e.g., by randomizing input data or initial conditions.)

Now, to analyze how blue fertility affects the time to extinction of red turtles, we will use a simulation experiment that varies `blue-fertility` from 2.1 to 5.0 in increments of 0.1.

producing a total of 30 scenarios. Further, we will run 10 replicates of each scenario. Each model run will continue until there are no more red turtles and then output the tick number at which this extinction occurred. These results need to be recorded in a file that we can then import to spreadsheet or statistical software and, finally, calculate the mean and standard deviation of time to red extinction and graph how they vary with `blue-fertility`.

This experiment is an example *sensitivity experiment*: a simulation experiment in which we vary one parameter over a wide range and see how the model responds to it (see also chapter 23). This experimental design is very useful for understanding how a model, and the system it represents, responds to one factor at a time.

Executing this experiment as we did in chapter 5—moving a slider to change the parameter value, hitting `setup` and `go` buttons, and writing down results—would take a long time, but now we will show you how to do such experiments very easily via BehaviorSpace. At this point you should read the BehaviorSpace Guide, which is under the "Features" heading of the NetLogo User Manual.

You can think of BehaviorSpace as a separate program, built into NetLogo, that runs simulation experiments on your model and saves the results in a file for you to analyze. By filling in a simple dialog, you can program it to do any or all of these functions:

- Create scenarios by changing the value of global variables;
- Generate replicates (called "repetitions" in NetLogo) of each scenario;
- Collect results from each model run and write them to a file;
- Determine when to stop each model run, using either a tick limit (e.g., stop after 1000 ticks) or a logical condition (e.g., stop if the number of red turtles is zero); and
- Run some NetLogo commands at the end of each model run.

The information that you enter in BehaviorSpace to run experiments is saved as part of the model's NetLogo file. Therefore, the first thing you need to do now, before using BehaviorSpace with the Simple Birth Rates model, is to save your own copy of this model with a unique name. From now on, use your own versions of the model instead of the one in the Models Library. Then:

- Open up BehaviorSpace from NetLogo's Tools menu.

The BehaviorSpace dialog that opens lets you create new experiments, edit previously saved ones, or copy or delete experiments. By "experiment" (or "experiment setup"), this dialog refers to a set of instructions defining the scenarios, replicates, outputs, and other experiment characteristics, that you want. You can create and save a number of different experiment setups for the same model.

- Create a new experiment setup by clicking on the New button.

This will open the Experiment dialog, which you will now fill out by modifying the default values. As you fill out the Experiment dialog, you can save your settings by clicking on the OK button at the bottom (to resume work on the dialog, reopen the experiment by clicking on Edit in the BehaviorSpace dialog).

- The first thing you should do in the Experiment dialog is give the experiment a new name. The name should describe the experiment, so it should be something like "Blue_fertility_effect_on_red_extinction."

- Next, you can specify the scenarios to run by filling in the field labeled "Vary variables as follows." Note that NetLogo automatically inserts the global variables that are in sliders or other Interface elements. From the BehaviorSpace documentation (and the example provided right below the field), you know that you can create scenarios varying `blue-fertility` from 2.1 to 5.0 in increments of 0.1 by entering:

```
["blue-fertility" [2.1 0.1 5.0]]
```

- In the "Vary variables as follows" field, it is a very good idea to also include the model variables you want to hold constant, fixing their values so they do not get changed accidentally by moving a slider:

```
["red-fertility" 2]
["carrying-capacity" 1000]
```

- Set the "Repetitions" value to 10.

Now you can tell NetLogo what results you want by filling in the field labeled "Measure runs using these reporters." You put NetLogo statements in this field that report outputs; these statements will be executed and the results written to the output file. In this case, the result that you want is the tick number at which red turtles become extinct. To get this, you can tell BehaviorSpace to stop the model when the number of red turtles is zero, and output the tick number when this happens. The reporter that gives the tick number is just the NetLogo primitive `ticks`, so:

- Put `ticks` in the "Measure runs . . ." field and uncheck the box for "Measure runs at every step." (See the programming note below to learn exactly when BehaviorSpace writes output.)

How do you tell BehaviorSpace to stop the model when reds go extinct? You need to put a condition in the "Stop condition" field that is true when the model should stop:

- Fill in the statement `red-count = 0`. (From the program, you will see that `red-count` is a global variable containing the current number of red turtles.)

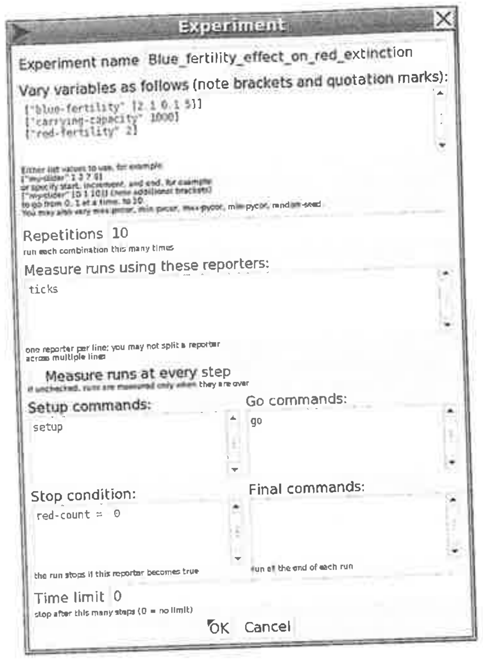
Nothing else in the Experiment dialog needs to be changed now, but make sure you understand what the boxes for "Setup commands," "Go commands," "Final commands," and "Time limit" do. The Experiment dialog should now look like figure 8.2.

- Now click OK to close the dialog. Close the BehaviorSpace dialog and save your file.

You are now ready to run the experiment.

- Open BehaviorSpace again from the Tools menu, select your experiment setup, and click "Run."

Try both the "spreadsheet" and "table" output formats; they each produce files in .csv format designed to be imported into spreadsheets or other software. You are also given a choice of how


Experiment

Experiment name Blue_fertility_effect_on_red_extinction

Vary variables as follows (note brackets and quotation marks):

```
["blue-fertility" [2.1 0.1 5]]
["carrying-capacity" 1000]
["red-fertility" 2]
```

Either list values to use, for example
["my-slider" 1 2 7 8]
or specify start, increment, and end, for example
["my-slider" [0 1 10]] (note additional brackets)
to go from 0, 1 at a time, to 10
You may also vary max-pxcor, min-pxcor, max-pycor, min-pycor, random-seed

Repetitions 10
run each combination this many times

Measure runs using these reporters:

```
ticks
```

one reporter per line; you may not split a reporter across multiple lines

Measure runs at every step
if unchecked, runs are measured only when they are over

Setup commands:

```
setup
```

Go commands:

```
go
```

Stop condition:

```
red-count = 0
```

the run stops if this reporter becomes true

Final commands:

run at the end of each run

Time limit 0
stop after this many steps (0 = no limit)

Figure 8.2
Experiment setup for Simple Birth Rates.

many of your computer's processors to use, if you have multiple processors (discussed in the second programming note below). After the experiment starts, you can deselect the updates of View and plots and monitors to speed up execution.

Programming note: How BehaviorSpace works

BehaviorSpace is an extremely important tool, but if you do not understand exactly how it works you will find it frustrating and you may misinterpret its results. You need to understand several details.

At the start of each run, BehaviorSpace changes the variable values as specified in its "Vary variables . . ." box. BehaviorSpace sets these values *before* it executes the `setup` procedure. Therefore, if `setup` sets a variable that BehaviorSpace is supposed to be controlling, the value given by BehaviorSpace will be overwritten by the value set in `setup`. Consequently, any variable that you want BehaviorSpace to control must be defined and initialized only by an interface element (typically, a slider or chooser).

When the "Measure runs at every step" box is *not* checked, results are calculated and written to the output file only after a model run has stopped. The run can be stopped by a `stop` statement in the `go` procedure (as we have in the Butterfly model), or by using the

"Stop condition" box in BehaviorSpace's experiment editing menu, or by using the "Time limit" box in the experiment menu.

When the "Measure runs at every step" box *is* checked, BehaviorSpace produces its first output *when the* `setup` *procedure completes*. This reports the state of the model after it is initialized and before simulations begin.

When the "Measure runs at every step" box is checked, BehaviorSpace then produces output *each time the end of the* `go` *procedure is reached*. If a `stop` statement terminates the run before the end of the `go` procedure (as in the Butterfly model), no output will be written for that last time step.

If you specify a "stop condition" or "time limit" in the BehaviorSpace experiment instead of in your procedures, BehaviorSpace checks this condition each time the `go` procedure has completed a time step and output for that step has been written. Hence, if you use a stop condition `ticks = 1000` NetLogo will completely finish the `go` procedure on which `ticks` reaches 1000, then write its BehaviorSpace output, then terminate the run.

There are therefore two ways to get output every time step as you expect it. One is to remove the `stop` statement from the `go` procedure and instead put it in the BehaviorSpace experiment (e.g., by setting the time limit to 1000). The second is to reorganize the `go` procedure so it starts with `tick`, followed by a statement that stops the model if the desired number of ticks is exceeded (not just met). This issue is considered further in section 14.2.6.

Programming note: BehaviorSpace, multiple processors, and output files

BehaviorSpace can execute multiple model runs at the same time if your computer has multiple processors. When an experiment starts, you tell BehaviorSpace how many of your processors you want it to use, and it will execute one model run at a time on each processor. But there are two consequences for file output.

First, files produced by BehaviorSpace will be affected because the lines of output for different model runs may be intermingled instead of in the expected order, for example:

[run number]	blue-fertility	[step]	ticks
1	2.1	0	0
2	2.2	0	0
1	2.1	1	1
1	2.1	2	2
2	2.2	1	1

This problem can be solved after you import the BehaviorSpace output to a spreadsheet, by simply sorting the output lines by run number and step.

Second, any output files that you code yourself are likely to cause unavoidable and fatal run-time errors because the multiple runs will each try to open the same file at the same time, which is not allowed. (The same problem could happen with input files.) The only solutions are to get rid of your output files, have each model run write to a different file, or run BehaviorSpace on only one processor.

When BehaviorSpace has finished running the experiment, you can import the results to software of your choice, calculate the mean and standard deviation over the 10 replicates of the time to red extinction for each value of `blue-fertility`, and draw a graph similar to figure 8.3.

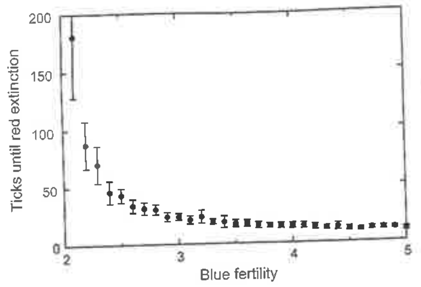


Figure 8.3
Example results of the Simple Birth Rates experiment. The dots are the mean time to red extinction, and error bars are +/− one standard deviation over the 10 replicates.

Does this graph look like you expected it to? While it seems reassuring that the model produced results that were quite as we expected, it also makes us wonder what the value of building an ABM was if it produces results we could easily anticipate. In fact, we chose this model to examine first because it is a clear example of an ABM whose dynamics are strongly imposed by its very simple, rigid agent behaviors: the agents make no decisions, they have no individual state variables, and their very few behaviors (reproducing, dying) are tightly specified, or imposed, by the global parameters. If the agents in this model adapted their fertility to (for example) local density of other turtles or food, or had to find mates within a certain time or distance before reproducing, the model would produce less predictable, more interesting results. While this model is fun to play with and provides a good BehaviorSpace exercise, it is not a good example of the kind of problem that requires an ABM to understand. So let's move on to a model that is, in contrast, a classic example of emergent dynamics.

8.4 A Model with Complex Emergent Dynamics

Another model in the Biology section of NetLogo Models Library is called Flocking. This is a NetLogo version of a well-known example (e.g., Reynolds 1987) of how complex and realistic dynamics can emerge from a few simple agent behaviors, in ways that could not be completely predicted. Schools of fish and flocks of birds can be thought of as emergent properties of how the individual animals move in response to each other. A school or flock can appear as a coherent entity, but it can also break up and re-form, and its shape changes continually. (There are fascinating videos of flocks of birds, especially starlings, on the web.)

In the Flocking model, the individuals have one behavior: adjusting their movement direction in response to the location and direction of other nearby individuals (their "flockmates," which are all other turtles within a radius equal to the `vision` parameter). They make this decision considering three objectives: moving in the same direction as their flockmates ("align"), moving toward the flockmates ("cohere"), and maintaining a minimum separation from all other individuals ("separate"). Parameters control the relative strength of these three objectives by limiting the maximum angle a turtle can turn to align, cohere, and separate. The `minimum-separation` parameter sets the distance at which turtles try to separate from each other.

You will quickly see that Flocking's results are in fact complex. The turtles form flocks that continually change characteristics, and these characteristics change as you vary the parameters. Further, parameters seem to interact (the effect of one parameter depends on the value of another); for example, see what happens when you vary `max-cohere-turn` when `minimum-separation` is low, then high. (To speed up the model, use the comments in the `go` procedure to make turtles move forward one unit instead of making five short moves.)

The Flocking model demonstrates two common (though not universal) characteristics of emergent dynamics. First, the most important and interesting results of the model seem qualitative and hard to describe with numbers. Whereas the state of the Simple Birth Rates model can be described by two numbers (how many red and blue turtles are alive), the characteristics of the emergent flocks clearly change as you vary parameters but in ways that are not easy to quantify. The second characteristic is that it takes some time before the flock characteristics emerge, both when the model starts and when you change parameter values in the middle of a run. ABMs often have a "warm-up" period in which their dynamics gradually emerge.

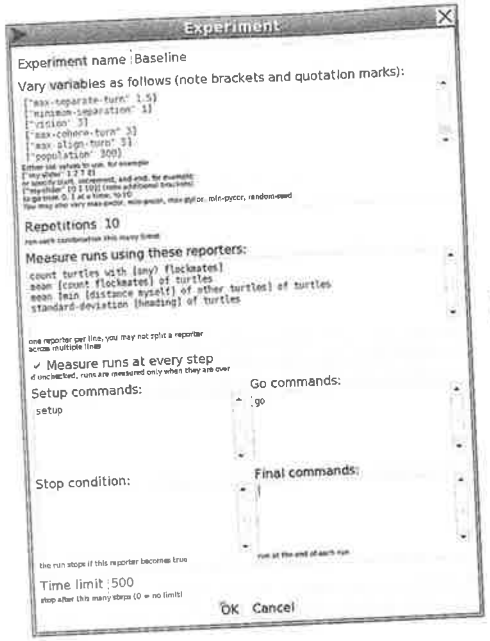
Let's think about how emergent Flocking's results are, using the criteria introduced in section 8.1. Are the results of this model different from just the sum of individual turtle properties, and of different types than the properties of the individuals, and not easily predicted from the rules given to the turtles? It is tempting to immediately answer "yes" to these questions, but there is one problem: we have not yet defined exactly *what* results of this model we are interested in. When we used the Simple Birth Rates model, we could clearly state what output we wanted to analyze: the time until red turtles went extinct. But we have not yet stated which results of Flocking we are analyzing.

This problem is a reminder that this course is about using ABMs for science. Flocking is a fascinating simulator: we specify some simple agent rules and observe the complex flocking dynamics that emerge. But when we do science we are instead trying to do the opposite: we identify some complex dynamics observed in the real world and then try to figure out what behaviors of the system's agents explain those emergent dynamics. Models similar to Flocking are in fact used for important science, as profiled in chapter 11 of Camazine et al. (2001) and section 6.2.2 of Grimm and Railsback (2005). In particular, Huth and Wissel (1992) combined ABMs with studies of real fish to address the question, "What assumptions about movement of individual fish explain the emergent characteristics of real schools of fish?"

We can use NetLogo's Flocking model to do simplified versions of Huth and Wissel's simulation experiments. Huth and Wissel (1992) started by specifying some specific model outputs that they could compare to real fish schools. These were statistics on properties such as how close the individuals are to the nearest other fish and how much variation there is in the direction they move. We can easily obtain similar statistical results from Flocking by using BehaviorSpace. (The model of Huth and Wissel is not directly comparable to Flocking; it uses slightly different behaviors and simulated only one small school of fish, whereas in Flocking there are typically several flocks.) We can, for example, calculate:

- The number of turtles who have flockmates;
- The mean number of flockmates per turtle;
- The mean distance between a turtle and the nearest other turtle; and
- The standard deviation in heading, over all turtles—an imperfect but simple measure of variability in direction.

Let's set up BehaviorSpace so it produces these results. For the Simple Birth Rates exercise we needed only one output at the end of each model run; now, let's get output from every tick of model runs that last 500 ticks. For now we do not want to vary any of the parameters, so we will set up no scenarios other than the "baseline" scenario with default parameter values. But we will run 10 replicates of this scenario. The Experiment dialog should look like figure 8.4. (Don't worry if you do not understand the reporters in the "Measure runs using . . ." field; you will learn about them in chapter 10.)



Experiment name Baseline

Vary variables as follows (note brackets and quotation marks):

```
["max-separate-turn" 1.5]
["minimum-separation" 1]
["vision" 3]
["max-cohere-turn" 3]
["max-align-turn" 5]
["population" 300]
```

Either list values to use, for example:
["my-slider" 1 2 7 8]
or specify start, increment, and end, for example:
["my-slider" [0 1 10]] (note additional brackets)
to go from 0, 1 at a time, to 10.
You may also vary max-pxcor, min-pxcor, max-pycor, min-pycor, random-seed

Repetitions 10
run each combination this many times

Measure runs using these reporters:

```
count turtles with [any? flockmates]
mean [count flockmates] of turtles
mean [min [distance myself] of other turtles] of turtles
standard-deviation [heading] of turtles
```

one reporter per line; you may not split a reporter across multiple lines

✓ Measure runs at every step
if unchecked, runs are measured only when they are over

Setup commands:
```
setup
```

Go commands:
```
go
```

Stop condition:

the run stops if this reporter becomes true

Final commands:

run at the end of each run

Time limit 500
stop after this many steps (0 = no limit)

Figure 8.4
Experiment setup for Flocking.

When you try to run this experiment, you may get a run-time error because the experiment uses the turtle variable `flockmates`, the agentset of other nearby turtles. Remember that BehaviorSpace calculates its first output when the `setup` procedure has finished, before the run actually starts. At this point, `flockmates` has not yet been given a value, so NetLogo does not know that it is an agentset. You can solve this problem by initializing `flockmates`: giving it a value that is an agentset, at the time each turtle is created. Simply add the statement `set flockmates no-turtles` to the `create-turtles` statement in `setup`:

```
crt population
[
  set color yellow - 2 + random 7 ; random shades look nice
  set size 1.5                    ; easier to see
  set flockmates no-turtles       ; New for BehaviorSpace
  setxy random-xcor random-ycor
]
```

(Look up `no-turtles` to understand this.) Just remember to ignore the first line of output, which will say that all turtles have zero flockmates, because it reports the model before the first tick is executed.

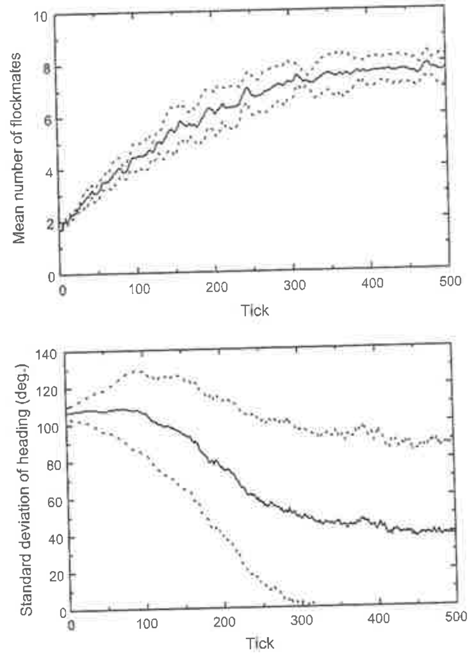


Figure 8.5
Results from Flocking experiment for (top) mean number of flockmates and (bottom) standard deviation of turtle headings. The solid lines are the mean, and dotted lines +/− one standard deviation, over 10 replicates.

Now you can graph and look at the results. For example, we found the mean number of flockmates to increase from less than two to over seven as the simulation proceeded (figure 8.5, top panel), while the standard deviation in heading decreased from over 100 to less than 40 degrees (bottom panel). Note that these results reflect the model's "warm-up" period when flocking patterns emerge. It appears that the number of flockmates and turtle heading outputs start to stabilize at about 400 ticks.

Now, let's conduct an experiment similar to one performed by Huth and Wissel (1992). What happens to the emergent schooling patterns if turtles adapt their direction considering only the one closest neighbor turtle? We can easily simulate this "one-flockmate" scenario with a small code change. Save a new version of Flocking and then change its `find-flockmates` procedure from this:

```
to find-flockmates  ;; turtle procedure
  set flockmates other turtles in-radius vision
end
```

to this:

```
to find-flockmates  ;; turtle procedure
  set flockmates other turtles in-radius vision
  set flockmates flockmates with-min [distance myself]
end
```

The new statement modifies the flockmates agentset so it contains only the nearest turtle. (In the unlikely case that the two nearest others are exactly the same distance from a turtle, `flockmates` would include both.)

NetLogo brainteaser

In the above code change that sets `flockmates` to the single nearest other turtle:

1. Why not simply use `set flockmates min-one-of other turtles [distance myself]` instead of two separate "set" statements? `min-one-of` selects the turtle with smallest distance from the calling turtle. (Try it!)
2. Why not simply use `set flockmates other turtles with-min [distance myself]`? (What seems strange if you try it?)

Answers:

1. `min-one-of` returns a single agent, not an agentset. This would turn the variable `flockmates` into a turtle instead of an agentset, but other code in this model requires `flockmates` to be an agentset. (The primitive `turtle-set` could be used to turn the result of `min-one-of` into an agentset.)
2. This statement always sets `flockmates` to an empty agentset. It first evaluates `turtles with-min [distance myself]`, which is simply the calling turtle. Then, `other` by definition removes the calling turtle from the agentset, leaving it empty. However, this statement *does* work if you put parentheses around `other turtles` to make NetLogo identify the other turtles before selecting from them the nearest one. We will study these agentset primitives in chapter 11.

With this change in turtle behavior, see how Flocking now looks. To see if these differences show up in the statistical output, repeat the BehaviorSpace experiment. You should see, in fact, that results are quite different. For example, the variation in turtle headings now changes little over time (figure 8.6) instead of decreasing sharply as it did in the baseline version (figure 8.5). Huth and Wissel (1992) used an experiment like this to conclude that real fish adjust their swimming direction considering several neighbors, not just one.

This experiment is an example of a second common kind of simulation experiment: contrasting alternative scenarios. The sensitivity experiment design we discussed in section 8.3 varies a parameter across a wide range, but scenario contrast experiments look at the differences between two (or several) distinctly different scenarios. (Scenario contrasts are more similar to the experimental designs typically used on real systems: specifying two or more treatments, then using replicates of each to determine how different the treatments' effects are.) Often, the different scenarios are different agent behaviors: in this example, the difference is defining flockmates as all other turtles within a range vs. only the closest other turtle. When we analyze scenario contrast experiments, we typically look at how different the results are between the two scenarios.

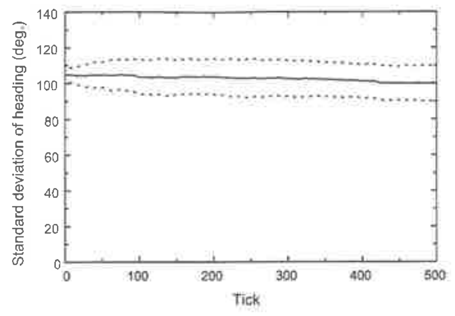


Figure 8.6
Flocking results for standard deviation in heading, with turtles using only one flockmate.

8.5 Summary and Conclusions

Simulation models, and especially ABMs, are useful because they can reproduce how complex dynamics of real systems emerge from the characteristics, behaviors, and interactions of the systems' members and their environment. ABMs can range from having relatively simple results that are closely imposed by simple, rigid rules for agent behavior, to producing many kinds of complex results that are very difficult to predict from agent behaviors. When we want to describe the emergence characteristics of an ABM, we can start by defining what the key outcomes of the model are and the agent behaviors that those outcomes emerge from. And we need to describe how model outcomes emerge from the agents' environment, if they do. In the Flocking and Simple Birth Rates models, results emerge only from agent behaviors; but the results of the hilltopping butterfly model depend qualitatively and quantitatively on behavior *and* on topography, an environment characteristic.

Making model results more emergent is not always better. Very simple ABMs with highly imposed dynamics may not be very different from an equation-based model of the same problem and hence not extremely exciting. But such simple ABMs can still be useful and appropriate for many problems, and can have advantages such as being easier and more intuitive to build and understand than equation-based models. At the other extreme, an ABM with such extremely emergent outcomes that they are completely unpredictable would be very hard to use for science because our goal is to figure out what underlying processes give rise to the emergent outcomes. Models such as Flocking that merely demonstrate emergence are not inherently scientific. However, they can be put to scientific use when we use them, with simulation experiments, to understand some specific emergent properties of a real system.

We observed two common characteristics of emergent dynamics in the Flocking model. First, important emergent outcomes may seem more qualitative than quantitative, but we can find numerical outputs to describe them. Second, it often takes a considerable "warm-up" period before emergent outcomes become apparent and stabilize.

In this chapter we demonstrated two very common and important kinds of simulation experiment. With the Simple Birth Rates model, we varied a parameter over a wide range and analyzed how model results changed in response; such *sensitivity* experiments analyze how sensitive the model is to a particular parameter or input. With the Flocking model, we compared two different versions of the model, a *contrast* of alternative model *scenarios* or versions. This kind of experiment is especially important for testing alternative model rules and determining which cause the ABM to be better at reproducing system behaviors observed in the real world.

In part III of this book we will delve more deeply into these simulation experiment issues. We will learn how to use qualitative patterns of emergent dynamics and contrasts of alternative model versions to develop theory for how agent-based systems work. In parts III and IV we will also learn how to use sensitivity experiments and other approaches for calibrating models to match real-world observations and to understand our model, and the system it represents, once it is built.

8.6 Exercises

1. Repeat the experiment on the Simple Birth Rates model in section 8.3, but using different values for the parameter `carrying-capacity`. How does the relationship between blue fertility and time to red extinction in figure 8.3 change when the carrying capacity is, for example, 500 and 1500 turtles?
2. "Risk analysis" is used in management of all kinds of systems. Conduct a new experiment on the Simple Birth Rates model using BehaviorSpace to determine how frequently the red turtles go extinct within 100 ticks. This output is an estimate of the probability ("risk") of red extinction within 100 ticks; how does it vary with blue fertility? You will clearly need different reporters to produce output and stop each run, but what else about the experiment could change?
3. Conduct a sensitivity experiment on the Flocking model. How do results such as those in figure 8.5 change if you use, for example, five values of `vision`, `max-align-turn`, or one of the other parameters?
4. Recent research on real starling flocks indicates that these birds do not respond to the flockmates within a limited distance but instead respond to the 6–7 nearest other birds, no matter how close or far they are (Ballerini et al. 2008). Conduct another scenario contrast experiment on the Flocking model. How do results of the experiment we did in section 8.4 change when you modify the `find-flockmates` procedure so flockmates are the nearest six turtles? (Hint: see the primitive `min-n-of`.) Include tests to prove that turtles always have six flockmates, and that a turtle's flockmates do not include itself.
5. Many models similar to Flocking assume that individuals adjust their direction not in response to all nearby others, but only in response to nearby others who are also within a particular vision angle (indicated by the arcs in figure 8.7, which has a vision angle of 60°). For example, if we assume turtles only see and respond to other turtles who are ahead, not behind, them, then flockmates could be set to all turtles that are within the vision distance *and* are at a vision angle less than 90° from the turtle's heading. Make this change to Flocking (which is extremely easy with the right primitive) and see how its results change as you vary the vision angle. Are the effects only qualitative or are they reflected in the model's statistical results?

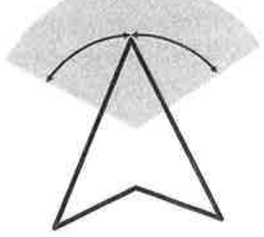


Figure 8.7