

# Contents

---

Preface *xi*  
Acknowledgments *xvii*

## **Part I Agent-Based Modeling and NetLogo Basics 1**

- 1 Models, Agent-Based Models, and the Modeling Cycle 3**
  - 1.1 Introduction, Motivation, and Objectives 3
  - 1.2 What Is a Model? 4
  - 1.3 What Does the Modeling Cycle Involve? 7
  - 1.4 What Is Agent-Based Modeling? How Is It Different? 10
  - 1.5 Summary and Conclusions 12
  - 1.6 Exercises 13
- 2 Getting Started with NetLogo 15**
  - 2.1 Introduction and Objectives 15
  - 2.2 A Quick Tour of NetLogo 16
  - 2.3 A Demonstration Program: Mushroom Hunt 18
  - 2.4 Summary and Conclusions 29
  - 2.5 Exercises 32
- 3 Describing and Formulating ABMs: The ODD Protocol 35**
  - 3.1 Introduction and Objectives 35
  - 3.2 What Is ODD and Why Use It? 36
  - 3.3 The ODD Protocol 37
  - 3.4 Our First Example: Virtual Corridors of Butterflies 43
  - 3.5 Summary and Conclusions 46
  - 3.6 Exercises 47
- 4 Implementing a First Agent-Based Model 49**
  - 4.1 Introduction and Objectives 49
  - 4.2 ODD and NetLogo 49

4.3	Butterfly Hilltopping: From ODD to NetLogo	50
4.4	Comments and the Full Program	57
4.5	Summary and Conclusions	60
4.6	Exercises	61
<b>5</b>	<b>From Animations to Science</b>	<b>63</b>
5.1	Introduction and Objectives	63
5.2	Observation of Corridors	64
5.3	Analyzing the Model	69
5.4	Time-Series Results: Adding Plots and File Output	69
5.5	A Real Landscape	71
5.6	Summary and Conclusions	74
5.7	Exercises	75
<b>6</b>	<b>Testing Your Program</b>	<b>77</b>
6.1	Introduction and Objectives	77
6.2	Common Kinds of Errors	78
6.3	Techniques for Debugging and Testing NetLogo Programs	81
6.4	Documentation of Tests	91
6.5	An Example and Exercise: The Culture Dissemination Model	92
6.6	Summary and Conclusions	94
6.7	Exercises	95
<b>Part II</b>	<b>Model Design Concepts</b>	<b>97</b>
<b>7</b>	<b>Introduction to Part II</b>	<b>99</b>
7.1	Objectives of Part II	99
7.2	Overview of Part II	100
<b>8</b>	<b>Emergence</b>	<b>103</b>
8.1	Introduction and Objectives	103
8.2	A Model with Less Emergent Dynamics	104
8.3	Simulation Experiments and BehaviorSpace	105
8.4	A Model with Complex Emergent Dynamics	111
8.5	Summary and Conclusions	116
8.6	Exercises	116
<b>9</b>	<b>Observation</b>	<b>119</b>
9.1	Introduction and Objectives	119
9.2	Observing the Model via NetLogo's View	120
9.3	Other Interface Displays	123
9.4	File Output	124
9.5	BehaviorSpace as an Output Writer	128
9.6	Export Primitives and Menu Commands	128
9.7	Summary and Conclusions	129
9.8	Exercises	129
<b>10</b>	<b>Sensing</b>	<b>131</b>
10.1	Introduction and Objectives	131
10.2	Who Knows What: The Scope of Variables	132
10.3	Using Variables of Other Objects	135
10.4	Putting Sensing to Work: The Business Investor Model	136

10.5	Summary and Conclusions	145
10.6	Exercises	146
<b>11</b>	<b>Adaptive Behavior and Objectives</b>	<b>149</b>
11.1	Introduction and Objectives	149
11.2	Identifying and Optimizing Alternatives in NetLogo	150
11.3	Adaptive Behavior in the Business Investor Model	154
11.4	Nonoptimizing Adaptive Behavior: A Satisficing Example	155
11.5	The Objective Function	158
11.6	Summary and Conclusions	159
11.7	Exercises	160
<b>12</b>	<b>Prediction</b>	<b>161</b>
12.1	Introduction and Objectives	161
12.2	Example Effects of Prediction: The Business Investor Model's Time Horizon	162
12.3	Implementing and Analyzing Submodels	164
12.4	Analyzing the Investor Utility Function	167
12.5	Modeling Prediction Explicitly	169
12.6	Summary and Conclusions	170
12.7	Exercises	171
<b>13</b>	<b>Interaction</b>	<b>173</b>
13.1	Introduction and Objectives	173
13.2	Programming Interaction in NetLogo	174
13.3	The Telemarketer Model	175
13.4	The March of Progress: Global Interaction	180
13.5	Direct Interaction: Mergers in the Telemarketer Model	180
13.6	The Customers Fight Back: Remembering Who Called	182
13.7	Summary and Conclusions	185
13.8	Exercises	186
<b>14</b>	<b>Scheduling</b>	<b>189</b>
14.1	Introduction and Objectives	189
14.2	Modeling Time in NetLogo	190
14.3	Summary and Conclusions	198
14.4	Exercises	199
<b>15</b>	<b>Stochasticity</b>	<b>201</b>
15.1	Introduction and Objectives	201
15.2	Stochasticity in ABMs	202
15.3	Pseudorandom Number Generation in NetLogo	204
15.4	An Example Stochastic Process: Empirical Model of Behavior	210
15.5	Summary and Conclusions	211
15.6	Exercises	213
<b>16</b>	<b>Collectives</b>	<b>215</b>
16.1	Introduction and Objectives	215
16.2	What Are Collectives?	216
16.3	Modeling Collectives in NetLogo	216
16.4	Example: A Wild Dog Model with Packs	218
16.5	Summary and Conclusions	228
16.6	Exercises	229

<b>Part III Pattern-Oriented Modeling</b>	231
<b>17 Introduction to Part III</b>	233
17.1 Toward Structurally Realistic Models	233
17.2 Single and Multiple, Strong and Weak Patterns	234
17.3 Overview of Part III	236
<b>18 Patterns for Model Structure</b>	239
18.1 Introduction and Objectives	239
18.2 Steps in POM to Design Model Structure	240
18.3 Example: Modeling European Beech Forests	241
18.4 Example: Management Accounting and Collusion	245
18.5 Summary and Conclusions	246
18.6 Exercises	247
<b>19 Theory Development</b>	249
19.1 Introduction and Objectives	249
19.2 Theory Development and Strong Inference in the Virtual Laboratory	250
19.3 Examples of Theory Development for ABMs	252
19.4 Exercise Example: Stay or Leave?	255
19.5 Summary and Conclusions	259
19.6 Exercises	260
<b>20 Parameterization and Calibration</b>	263
20.1 Introduction and Objectives	263
20.2 Parameterization of ABMs Is Different	264
20.3 Parameterizing Submodels	265
20.4 Calibration Concepts and Strategies	266
20.5 Example: Calibration of the Woodhoopoe Model	272
20.6 Summary and Conclusions	275
20.7 Exercises	276
<b>Part IV Model Analysis</b>	279
<b>21 Introduction to Part IV</b>	281
21.1 Objectives of Part IV	281
21.2 Overview of Part IV	282
<b>22 Analyzing and Understanding ABMs</b>	285
22.1 Introduction and Objectives	285
22.2 Example Analysis: The Segregation Model	286
22.3 Additional Heuristics for Understanding ABMs	291
22.4 Statistics for Understanding	295
22.5 Summary and Conclusions	296
22.6 Exercises	297
<b>23 Sensitivity, Uncertainty, and Robustness Analysis</b>	299
23.1 Introduction and Objectives	299
23.2 Sensitivity Analysis	301
23.3 Uncertainty Analysis	307
23.4 Robustness Analysis	312

23.5	Summary and Conclusions	313
23.6	Exercises	314
<b>24</b>	<b>Where to Go from Here</b>	<b>317</b>
24.1	Introduction and Objectives	317
24.2	Keeping Your Momentum: Reimplementation	318
24.3	Your First Model from Scratch	318
24.4	Modeling Agent Behavior	319
24.5	ABM Gadgets	320
24.6	NetLogo as a Platform for Large Models	321
24.7	An Odd Farewell	323
	References	325
	Index	333
	Index of Programming Notes	339

# Models, Agent-Based Models, and the Modeling Cycle

## 1.1 Introduction, Motivation, and Objectives

Why is it important to learn how to build and use agent-based models (ABMs) or “individual-based” models, as they are called in some fields? The short answer to this question is that we need ABMs to solve problems that traditional models and methods are too simple for. For example, An (2001) showed that a simple ABM could explain the previously misunderstood dynamics of a medical syndrome that kills many people. Ecologists have learned that traditional models of the basic relations between predator and prey populations are unrealistic because they ignore individual behaviors that ABMs can represent (Abrams 1993; Railsback and Harvey 2013). In perhaps the most prominent example, the 2008 global financial crisis has been attributed in part to policy models that simplify away the essential complexities that ABMs can address (Buchanan 2009).

Let’s now look closely at one real model and the difference it has made.

### 1.1.1 A Motivational Example: Rabies Control in Europe

Rabies is a viral disease that kills great numbers of wild mammals and can spread to domestic animals and people. In Europe, rabies is transmitted mainly by red foxes. When an outbreak starts in a previously rabies-free region, it spreads in “traveling waves”: alternating areas of high and low infection rates.

Rabies can be eradicated from large areas, and new outbreaks can be controlled, by immunizing foxes: European governments have eradicated rabies from central Europe by manufacturing rabies vaccine, injecting it into baits, and spreading the baits from aircraft. However, this program is extremely expensive and works only if new outbreaks are detected and contained. Key to its cost-effectiveness are these questions: What percentage of wild foxes need to be vaccinated to eliminate rabies from an area, and what is the best strategy for responding to outbreaks?

Models have long been applied to such epidemiological problems, for wildlife as well as people. Classical differential equation models of the European rabies problem predicted that

70% of the fox population must be vaccinated to eliminate rabies. Managers planned to respond to new outbreaks using a “belt vaccination” strategy (which has worked well for other epidemics, including smallpox): not vaccinating the outbreak location itself but a belt around it, the width of which was usually determined by the limited emergency supply of vaccine. The 70% vaccination strategy did succeed, but the rabies problem has several characteristics suggesting that an agent-based modeling approach could make important contributions: the spread of rabies has important patterns in space as well as time, and is driven by individual behavior (in this case, the use of stationary territories by most foxes but long-distance migration by young foxes). Hence, Florian Jeltsch and colleagues developed a simple ABM that represented fox families in stationary home ranges and migration of young foxes (Jeltsch et al. 1997). This model accurately simulated the spread of rabies over both space and time.

Dirk Eisinger and Hans-Hermann Thulke then modified the ABM specifically to evaluate how the distribution of vaccination baits over space affects rabies control (Thulke and Eisinger 2008, Eisinger and Thulke 2008, Eisinger et al. 2005). Their ABM indicated that eradication could be achieved with a vaccination rate much lower than 70%, a result that could save millions of euros and was confirmed by the few case studies where actual vaccination coverage was monitored. The reason for the lower vaccination rate predicted by the ABM is that the “wave” spread of rabies emerges from local infectious contacts that actually facilitate eradication. The ABM of Eisinger and Thulke also indicated that the belt vaccination strategy for outbreaks would fail more often than an alternative: compact treatment of a circle around the initial outbreak. Because the ABM had reproduced many characteristics of real outbreaks and its predictions were easy to understand, rabies managers accepted this result and began—successfully—to apply the compact vaccination strategy.

The rabies example shows that agent-based modeling can find new, better solutions to many problems important to our environment, health, and economy—and has already done so. The common feature of these problems is that they occur in systems composed of autonomous “agents” that interact with each other and their environment, differ from each other and over space and time, and have behaviors that are often very important to how the system works.

### 1.1.2 Objectives of Chapter 1

This chapter is your introduction to modeling and agent-based modeling. We get started by clarifying some basic ideas about modeling. These lessons may seem trivial at first, but they are in fact the very foundation for everything else in this course. Learning objectives for this chapter are to develop a firm understanding of

- What models are, and what modeling is—why do we build models anyway?
- What the modeling cycle involves—what is the iterative process of designing, implementing, and analyzing models and using them to solve scientific problems?
- What agent-based models are—how are ABMs different from other kinds of models, and why would you use them?

## 1.2 What Is a Model?

A model is a purposeful representation of some real system (Starfield et al. 1990). We build and use models to solve problems or answer questions about a system or a class of systems. In science, we usually want to understand how things work, explain patterns that we have observed, and predict a system’s behavior in response to some change. Real systems often are

too complex or develop too slowly to be analyzed using experiments. For example, it would be extremely difficult and slow to understand how cities grow and land uses change just with experiments. Therefore, we try to formulate a simplified representation of the system using equations or a computer program that we can then manipulate and experiment on. (To formulate a model means to design its assumptions and algorithms.)

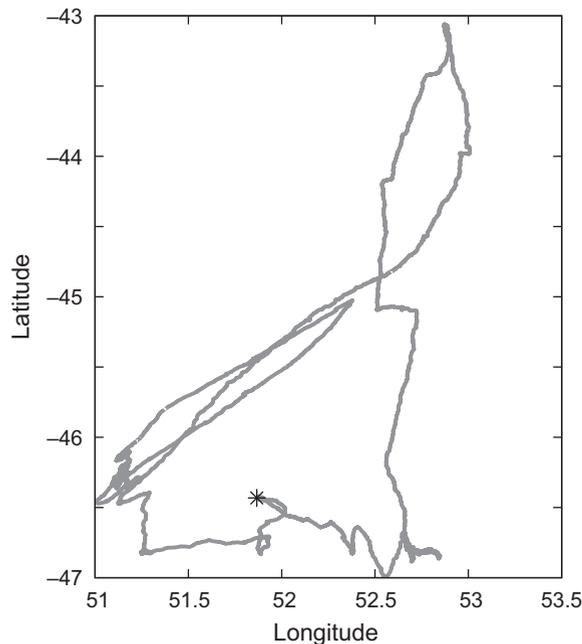
But there are many ways of representing a real system (a city or landscape, for example) in a simplified way. How can we know which aspects of the real system to include in the model and which to ignore? To answer this question, the model's purpose is decisive. The question we want to answer with the model serves as a filter: all those aspects of the real system considered irrelevant or insufficiently important for answering *this* question are filtered out. They are ignored in the model, or represented only in a very simplified way.

Let us consider a simple, but not trivial, example: Did you ever search for mushrooms in a forest? Did you ask yourself what the best search strategy might be? If you are a mushroom expert, you would know how to recognize good mushroom habitat, but let us assume you are a neophyte. And even the mushroom expert needs a smaller-scale search strategy because mushrooms are so hard to see—you often almost step on them before seeing them.

You might think of several intuitive strategies, such as scanning an area in wide sweeps but, upon finding a mushroom, turning to smaller-scale sweeps because you know that mushrooms occur in clusters. But what does “large” and “small” and “sweeps” mean, and how long should you search in smaller sweeps until you turn back to larger ones?

Many animal species face similar problems, so it is likely that evolution has equipped them with good adaptive search strategies. (The same is likely true of human organizations searching for prizes such as profit and peace with neighbors.) Albatross, for example, behave like mushroom hunters: they alternate more or less linear long-distance movements with small-scale searching (figure 1.1).

The common feature of the mushroom hunter and the albatross is that their sensing radius is limited—they can only detect what they seek when they are close to it—so they must move.



**Figure 1.1**  
Flight path of a female wandering albatross (*Diomedea exulans*) feeding in the southern Indian ocean. The flight begins and ends at a breeding colony (indicated by the star) in the Crozet Islands. Data recorded by H. Weimerskirch and colleagues for studies of adaptive search behavior in albatross (e.g., Weimerskirch et al. 2007).

And often the items searched for are not distributed randomly or regularly but in clusters, so search behavior should be adaptive: it should change once an item is found.

Why would we want to develop a model of this problem? Because even for this simple problem we are not able to develop quantitative mental models. Intuitively, we find a search strategy that works quite well, but then we see others who use different strategies and find more mushrooms. Are they just luckier, or are their strategies better?

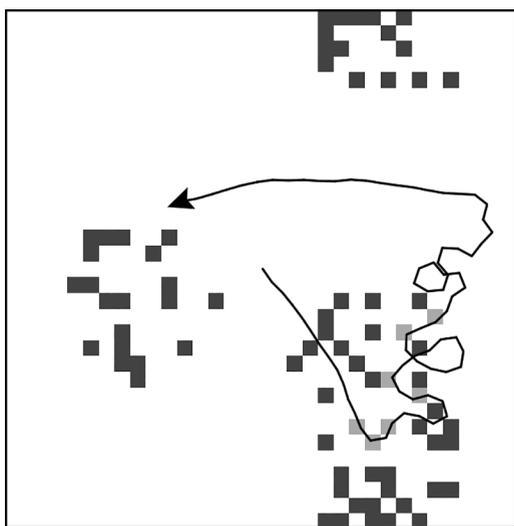
Now we understand that we need a clearly defined purpose before we can formulate a model. Imagine that someone simply said to you, “Please, model mushroom hunting in the forest.” What should you focus on? On different mushroom species, different forests, identification of good and bad habitats, effects of hunting on mushroom populations, etc.? However, with the purpose “What search strategy maximizes the number of mushrooms found in a certain time?” we know that

- We can ignore trees and vegetation; we only need to take into account that mushrooms are distributed in clusters. Also, we can ignore any other heterogeneity in the forest, such as topography or soil type—they might affect searching a little, but not enough to change the general answer to our question.
- It will be sufficient to represent the mushroom hunter in a very simplified way: just a moving “point” that has a certain sensing radius and keeps track of how many mushrooms it has found and perhaps how long it has been since it found the last one.

So now we can formulate a model that includes clusters of items and an individual “agent” that searches for the items in the model world. If it finds a search item, it switches to smaller-scale movement, but if the time since it found the last item exceeds a threshold, it switches back to more straight movement to increase its chances of detecting another cluster of items. If we assume that the ability to detect items does not change with movement speed, we can even ignore speed.

Figure 1.2 shows an example run of such a model, our simple Mushroom Hunt model. In chapter 2 you will start learning NetLogo, the software platform we use in this book, by programming this little model.

This searching problem is so simple that we have good idea of what processes and behaviors are important for modeling it. But how in general can we know whether certain factors



**Figure 1.2**  
Path of a model agent searching for items that are distributed in clusters.

are important with regard to the question addressed with a model? The answer is we can't! That is exactly why we have to formulate, implement (program in the computer), and analyze a model, because then we can use mathematics and computer logic to rigorously explore the consequences of our simplifying assumptions.

Our first formulation of a model must be based on our preliminary understanding of how the system works, what the important elements and processes are, and so on. These preliminary ideas might be based on empirical knowledge of the system's behavior, on earlier models addressing similar questions, on theory, or just on . . . imagination (as in the mushroom hunting example). However, if we have no idea whatsoever how the system works, we cannot formulate a model! For example, even though scientists are happy to model almost everything, so far there seems to be no explicit model of human consciousness, simply because we have no clue what consciousness really is and how it emerges.

Because the assumptions in the first version of a model are experimental, we have to test whether they are appropriate and useful. For this, we need criteria for whether the model can be considered a good representation of the real system. These criteria are based on patterns or regularities that let us identify and characterize the real system in the first place. Stock market models, for example, should produce the kinds of volatility and trends in prices we see in real markets. Often we find that the first version of a model is too simple, lacks important processes and structures, or is simply inconsistent. We thus go back and revise our simplifying assumptions.

### 1.3 What Does the Modeling Cycle Involve?

When thinking about a model of a mushroom hunter (or albatross), we intuitively went through a series of tasks. Scientific modeling means going through these tasks in a systematic way and using mathematics and computer algorithms to rigorously determine the consequences of the simplifying assumptions that make up our models.

Being scientific always means iterating through the tasks of modeling several times, because our first models can always be improved in some way: they are too simple or too complex, or they make us realize that we are asking the wrong questions. It is therefore useful to view modeling as iterating through the “modeling cycle” (figure 1.3). *Iterating* does not mean that we always go through the full cycle; rather, we often go through smaller loops, for example, between problem formulation and verbal formulation of the model. The modeling cycle consists of the following tasks:

1. *Formulate the question.* We need to start with a very clear research question because this question then serves as the primary compass and filter for designing a model. Often, formulating a clear and productive question is by itself a major task because a clear question requires a clear focus. For complex systems, getting focused can be difficult. Very often, even our questions are only experimental and later we might need to reformulate the question, perhaps because it turns out not to be clear enough, or too simple, or too complex.  
The question in our Mushroom Hunt model is, What search strategy maximizes the rate of finding items if they are distributed in clusters?
2. *Assemble hypotheses for essential processes and structures.* Agent-based modeling is “naive” (DeAngelis et al. 1994) in the sense that we are not trying to aggregate agents and what they are doing in some abstract variables like abundance, biomass, overall wealth, demographic rates, or nutrient fluxes. Instead, we naively and directly represent the agents and their behavior. We create these agents, put them in a virtual environment, and then

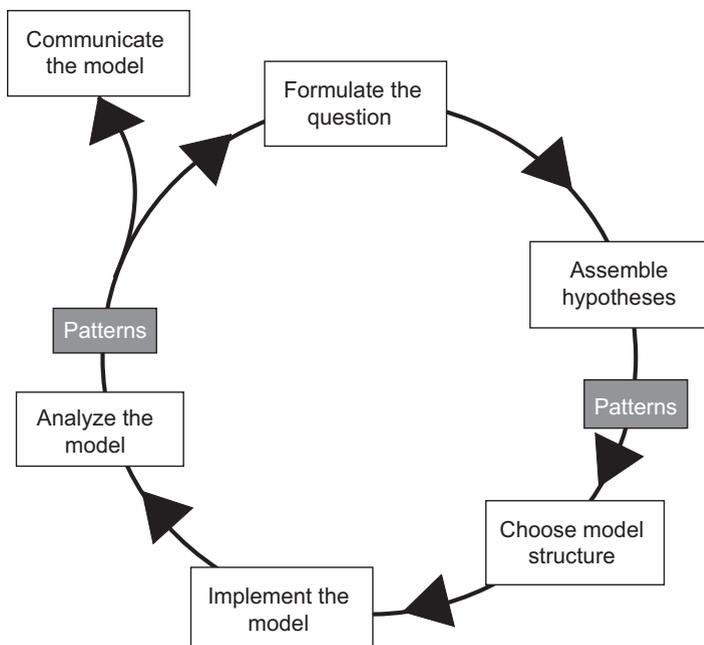


Figure 1.3  
The modeling cycle (from  
Grimm and Railsback 2005).

let the virtual world run and see what we can learn from it. (It is important, though, to ask ourselves: Is it possible to answer our question using a more aggregated and thus simpler model?)

Usually, we have to formulate many hypotheses for what processes and structures are essential to the question or problem we address. We can start top-down and ask ourselves questions such as, What factors have a strong influence on the phenomena of interest? Are these factors independent or interacting? Are they affected by other important factors? We might draw so-called influence diagrams, or flow charts, or just caricatures of our system and question. But whatever technique we prefer, this task has to combine our existing knowledge and understanding with a brainstorming phase in which we wildly hypothesize, followed by, most importantly, a simplification phase.

We have to force ourselves to simplify as much as we can, or even more. The modeling cycle must be started with the most simple model possible, because we want to develop understanding gradually, while iterating through the cycle. A common mistake of beginners is to throw too much into the first model version—usually arguing that all these factors are well known and can't possibly be ignored. The modeling expert's answer to this is, yes, you might be right, but—let us focus on the absolute minimum number of factors first. Put all the other elements that you think might need to be in the model on your “wish list” and check their importance later.

The reason for this advice is this: just our preliminary understanding of a system is not sufficient for deciding whether things are more or less important for a model. It is the very purpose of the model to teach us what is important. So it is wise to have a model implemented as soon as possible, even if it is ridiculously simple. The simpler the model is, the easier it is to implement and analyze, and the sooner we are productive. The real productive phase in a modeling project starts when we get the modeling cycle running: assumptions—implementation—model output—analyses—interpretation—revised assumptions, and so on.

It is difficult to formalize this task of the modeling cycle. One important help is heuristics for modeling: rules of thumb that are often, but not always, useful for designing models. We point out these heuristics throughout this book; use the index to find them. Compilations of modeling heuristics can be found in Starfield et al. (1990) and Grimm and Railsback (2005, chapter 2). And—in part III of this book we present *pattern-oriented modeling*, a very important strategy for formalizing both this and the next step in the modeling cycle.

For the Mushroom Hunt model, we assume that the essential process is switching between relatively straight large-scale “scanning” movement and small-scale searching, depending on how long it has been since the hunter last found an item.

3. *Choose scales, entities, state variables, processes, and parameters.* Once we choose some simplifying assumptions and hypotheses to represent our system of interest, it is time to sit down and think through our model in detail. We thus produce a written formulation of the model. Producing and updating this formulation is essential for the entire modeling process, including delivery to our “clients” (our thesis committee, journal reviewers, research sponsors, etc.). In chapter 3, we will start using a very helpful protocol for doing this.

This step, for the Mushroom Hunt model, includes specifying how the space that hunters move through is represented (as square grids with size equal to the area the hunter can search in one time step), what kinds of objects are in the model (one hunter and the items it searches for), the state variables or characteristics of the hunter (the time it has hunted and the number of items it has found, and the time since last finding an item), and exactly how the hunter searches. (Full details are provided when we implement the model in chapter 2.)

4. *Implement the model.* This is the most technical part of the modeling cycle, where we use mathematics and computer programs to translate our verbal model description into an “animated” object (Lotka 1925). Why animated? Because, in a way, the implemented model has its own independent dynamics (or “life”), driven by the internal logic of the model. Our assumptions may be wrong or incomplete, but the implementation itself is—barring software mistakes—always right: it allows us to explore, in a logical and rigorous way, the consequences of our assumptions and see whether our initial model looks useful.

This task is often the most daunting one for neophytes in modeling, because they usually have no training in how to build software. Thus, our claim that the implementation is always “right” might sound ironic to beginners. They might struggle for months to get the implementation right—but only if they don’t take advantage of existing software platforms for agent-based modeling. With the platform that we use in this book, NetLogo, you can often implement simple ABMs within a day or two, including the time to test your code and show that it is accurate. So please don’t panic!

5. *Analyze, test, and revise the model.* While new modelers might think that designing a model and implementing it on the computer takes the most work, this task—analyzing a model and learning from it—is the most time-consuming and demanding one. With tools like NetLogo, you will learn to quickly implement your own ABMs. But doing science with ABMs requires much more. Much of this book will be devoted to this task: How can we learn from our models? In particular, we will try to put forward the research program of “individual-based ecology” (Grimm and Railsback 2005) and apply it to other sciences. This program is dedicated to learning about the real world: we do not just want to see what happens when we create some agents and make up their behaviors—we want to see what agent behaviors can explain and predict important characteristics of real systems. To answer the mushroom hunting question, we could analyze the model by trying

a variety of search algorithms and parameter values to see which produce the highest rate of finding items.

6. *Communicate the model.* Here, “communicate” means to publish or otherwise transfer the knowledge produced so far to clients, other scientists, and perhaps even the public. Of course, many other kinds of communication take place throughout the modeling cycle. One very important benefit of this task is that it enforces discipline on the modeling cycle by making us stop the other tasks long enough to thoroughly document what we have done and learned so far, and to think and get feedback on what we should do next.

## 1.4 What Is Agent-Based Modeling? How Is It Different?

Historically, the complexity of scientific models was often limited by mathematical tractability: when differential calculus was the only approach we had for modeling, we had to keep models simple enough to “solve” mathematically and so, unfortunately, we were often limited to modeling quite simple problems—or forced to address complex problems with models that were too simple.

With computer simulation, the limitation of mathematical tractability is removed so we can start addressing problems that require models that are less simplified and include more characteristics of the real systems. ABMs are less simplified in one specific and important way: they represent a system’s individual components and their behaviors. Instead of describing a system only with variables representing the state of the whole system, we model its individual agents. ABMs are thus models where individuals or agents are described as unique and autonomous entities that usually interact with each other and their environment locally. Agents may be organisms, humans, businesses, institutions, and any other entity that pursues a certain goal. Being unique implies that agents usually are different from each other in characteristics such as size, location, resource reserves, and history. Interacting locally means that agents usually do not interact with all other agents but only with their neighbors—in geographic space or in some other kind of “space” such as a network. Being autonomous implies that agents act independently of each other and pursue their own objectives. Organisms strive to survive and reproduce; traders in the stock market try to make money; businesses have goals such as meeting profit targets and staying in business; regulatory authorities want to enforce laws and provide public well-being. Agents therefore use *adaptive behavior*: they adjust their behavior to the current states of themselves, of other agents, and of their environment.

Using ABMs lets us address problems that concern *emergence*: system dynamics that arise from how the system’s individual components interact with and respond to each other and their environment. Hence, with ABMs we can study questions of how a system’s behavior arises from, and is linked to, the characteristics and behaviors of its individual components.

ABMs are useful for problems of emergence because they are *across-level* models. Traditionally, some scientists have studied only systems, modeling them using approaches such as differential equations that represent how the whole system changes. Other scientists have studied only what we call agents: how plants and animals, people, organizations, etc. change and adapt to external conditions. ABMs are different because they are concerned with two (and sometimes more) levels and their interactions: we use them to both look at what happens to the *system* because of what its *individuals* do and what happens to the *individuals* because of what the *system* does. So throughout this course there will be a focus on modeling the behavior of agents and, at the same time, observing and understanding the behavior of the system made up by the agents.

ABMs are also often different from traditional models by being “unsimplified” in other ways, such as representing how individuals, and the environmental variables that affect them, vary over space, time, or other dimensions. ABMs often include processes that we know to be important but that are too complex to include in simpler models.

What kinds of problems can we address with ABMs? Every year more and more important problems are addressed with ABMs because they demand across-level approaches and are too complex to address without models or with only simple models. Here are some examples:

- What factors caused the housing market collapse of 2008, and what policy changes could prevent a recurrence (Gilbert et al. 2009, Geanakoplos 2012)?
- What causes the complex and seemingly unpredictable dynamics of a stock market? Are market fluctuations caused by dynamic behavior of traders, variation in stock value, or simply the market’s trading rules (LeBaron 2001, Duffy 2006)?
- How can we manage tropical forests in a sustainable way, maintaining both economic uses and biodiversity levels critical for forests’ stability properties (Huth et al. 2004, Bohn and Huth 2017)?
- How do shorebird populations respond to loss or alteration of the mudflats they feed in, and how can the effects be mitigated cost-effectively (Goss-Custard et al. 2006, Garcia et al. 2016)?
- How is development and maintenance of human tissue regulated by signals from the genome and the extracellular environment and by cellular behaviors such as migration, proliferation, differentiation, and cell death? How do diseases result from abnormalities in this system (Peirce et al. 2004, An 2015, Martin et al. 2015)?
- How do pharmaceuticals interact with human organs and within-organ processes to produce both benefits (Hunt et al. 2013) and undesirable side effects (Smith et al. 2016)?
- What drives patterns of land use change during urban sprawl, and how are they affected by the physical environment and by management policies (Parker et al. 2003, Brown et al. 2004, Groeneveld et al. 2017)?

The ability of ABMs to address complex, multilevel problems comes at a cost, of course. Traditional modeling requires mathematical skills, especially differential calculus and statistics. But to use simulation modeling we need additional skills. This course is designed to give you three very important skills for using ABMs:

- A new “language” for thinking about and describing models. Because we cannot define ABMs concisely or accurately in the languages of differential equations or statistics, we need a standard set of concepts (e.g., emergence, adaptive behavior, interaction, sensing) that describe the important elements of ABMs.
- The software skills to implement models on computers and to observe, test, control, and analyze the models. Producing useful software is more complex for ABMs than for most other kinds of models.
- Strategies for designing and analyzing models. There is almost no limit to how complex a computer simulation model can be, but if a model is too complex it quickly becomes too hard to parameterize, validate, or analyze. We need a way to determine what entities, variables, and processes should and should not be in a model, and we need methods for analyzing a model, after it is built, to learn about the real system.

Full-fledged ABMs assume that agents are different from each other; that they interact with only some, not all, other agents; that they change over time; that they can have different “life cycles” or stages they progress through, possibly including birth and death; and that they make

autonomous adaptive decisions to pursue their objectives. However, as with any model assumption, assuming that these individual-level characteristics are important is experimental. It might turn out that for many questions we do not explicitly need all, or even any, of these characteristics. And in fact full-fledged ABMs are quite rare. In ecology, for example, many useful ABMs include only one individual-level characteristic: local interactions. Thus, although ABMs are defined by the assumption that agents are represented in some way, we still have to make many choices about what type of agents to represent and in what detail.

Because most model assumptions are experimental, we need to test our model: we must implement the model and analyze its assumptions. For the complex systems we usually deal with in science, just thinking is not sufficient to rigorously deduce the consequences of our simplifying assumptions: we have to let the computer show us what happens. We thus have to iterate through the modeling cycle.

## 1.5 Summary and Conclusions

Agent-based modeling is no longer a completely new approach, but it still offers many exciting new ways to look at old problems and lets us study many new problems. In fact, the use of ABMs is even more exciting now that the approach has matured: the worst mistakes have been made and corrected, agent-based approaches are no longer considered radical and suspicious, and we have convenient tools for building models. People like you are positioned to take advantage of what the pioneers have learned and the tools they have built, and to get directly to work on interesting problems.

In this first chapter our goal is to provide some extremely fundamental and important ideas about modeling and agent-based modeling. Whenever you find yourself frustrated with either your own model or someone else's, in "big-picture" ways (What exactly does this model do? Is it a good model or not? Should I add this or that process to my model? Is my model "done?"), it could be useful to review these fundamental ideas. They are, in summary:

- A model is a purposeful *simplification* of a system for solving a particular *problem* (or category of problems).
- We use ABMs when we think it is important for a model to include the system's individuals and what they do.
- Modeling is a cycle of formulating a precise question; assembling hypotheses for key processes and structures; formulating the model by choosing appropriate scales, entities, state variables, processes, and parameters; implementing the model in a computer program; and analyzing, testing, and revising.

Understanding this modeling cycle is so important that a review of modeling practice (Schmolke et al. 2010) concluded that explicitly thinking about and documenting each step in the cycle is the primary way we can improve how models are developed and used. Schmolke et al. proposed a very useful format ("TRACE") for documenting the entire cycle of developing, implementing, and analyzing a model. TRACE has been further developed by Augusiak et al. (2014) and Grimm et al. (2014).

It is very important that you have a basic understanding of these ideas from the start, but for the rest of part I we focus on obtaining a fundamental knowledge of how to implement models on the computer. In the rest of this course, however, we will come back to modeling ideas. As soon as you have some ability to program and analyze your own models and some understanding of how to use these modeling concepts, you will rapidly become a real modeler.

## 1.6 Exercises

1. One famous example of how different models must be used to solve different problems in the same system is grocery store checkout queues. If you are a customer deciding which queue to enter, how would you model the problem? What exact question would your model address? What entities and processes would be in the model? Now, if instead you are a store manager deciding how to operate the queues for the next hour or so, what questions would your model address and what would it look like? Finally, if you are a store designer and the question is how to design the checkout area so that 100 customers can check out per hour with the fewest employees, what things would you model? (Hint: Think about queues in places other than stores.)
2. For the following questions, what should be in a model? What kinds of things should be represented, what variables should those things have to represent their essential characteristics, and what processes that change things should be in the model? Should the model be agent-based? If the question is not clear enough to decide, then reformulate the question to produce one that is sufficiently clear.
  - a) How closely together should a farmer plant the trees in a fruit orchard?
  - b) How much of her savings should an employee put in each of the five investment funds in her retirement program?
  - c) Should a new road have one, two, or three lanes in each direction?
  - d) Is it acceptable to allow a small legal harvest of whales?
  - e) To complete a bachelor's degree in physics as soon as possible, what classes should a student register for this semester?
  - f) How many trees per year should a timber company harvest?
  - g) Banks make money by investing the money that their customers deposit, but they must also keep some money available as cash. A bank can fail if its customers withdraw more cash than the bank has available, or if their investments do not make enough money to meet expenses. Government regulators require banks to keep a minimum percentage of total deposits as cash that is not invested. To minimize bank failures, what should this minimum percentage be?
  - h) To maximize profit, how many flights per day should Saxon Airlines schedule between Frankfurt (their international hub) and Leipzig?
  - i) To minimize system-wide delays and risk of accidents, how many flights per day should the European Aviation Administration allow between Frankfurt and Leipzig?
  - j) Two new movies will open in theaters next weekend. One is based on a comic book series and features a superhero, special effects, car chases, and violence. The other is a romantic comedy starring a beautiful actress and a goofy but lovable actor. Which movie will make the most money next weekend? Over the next four weeks? Over the next five years?
  - k) Using your own studies, research, or experience in general, what other problems or questions might be the basis of a model or agent-based model?

## Index

- across-level models, 10–11
- actions (elements of a model schedule), 41, 190–199
- adaptation and adaptive behavior, 10, 41–42, 149–150; evolutionary approaches, 320; memory in, 182–85; modeling of, 154–59; non-optimizing approaches and satisficing, 155–56; prediction in, 161–64, 169; optimization approaches, 150–53; rule-based approaches, 210; stay-or-leave decisions, 256; stochastic approaches, 210–11; theory for, 249–55, 319–20; with trade-offs among objectives, 150, 154–55, 251–52
- Agent Monitors, 26–27, 80, 84, 122, 153
- agent-based models: across-level nature of, 10; and theory, 249–55, 319–20; criticisms of, 91, 275; definition of, 10–12; difference between individual-based models and, xii; full-fledged, 11–12; typical questions of, 11
- agents: decision alternatives as, 150–52; definition of, 10; NetLogo terminology for, 16–17
- agentsets: built-in, 31, 150–51; differences between lists and, 183; initialization of, 112; local, 151; merging of, 140–41; for modeling collectives, 216, 218, 223; removing agents from, 141; sensing of, 135, 139–40; sorting, 192–93; subsetting (filtering) of, 150–52
- alternatives analysis and model uncertainty, 310–12
- and, 93, 152
- anonymous procedures, 184, 193
- arithmetic operators, 26
- ask: and changes in code context, 18, 53; randomization by, 183, 190; used to make agents execute code, 21; used to set variables of other agents, 135–36
- attributes. *See* state variables
- basic principles (element of ODD model description protocol), 42
- Bayesian updating theory to model prediction, 169
- beech forest model BEFORE, 241–45
- beginners' mistakes and difficulties, 8, 9, 22, 39, 66, 68, 319
- behavior. *See* adaptation and adaptive behavior
- BehaviorSpace, 105–11, 154; in calibration experiments, 271, 273–74; on computer clusters, 322; global variables for, 133; and the `go` procedure's organization, 192; random number control and, 209, 299–301; for sensitivity and uncertainty analysis, 302, 309–10; using without putting variables on the interface, 302; for writing output, 128
- Bernoulli distribution, 206–7
- boolean conditions, reporters, and variables, 25, 66, 185, 212; generated by a Bernoulli distribution, 200–201; in `if` and `ifelse` statements, 25, 185; in looping, 196; controlled by switches on Interface tab, 66, 93
- brainteasers, 54, 114, 152
- Breeding Synchrony model, 194; robustness analysis of, 313

- breeds, breed, and breeds-own, 215, 217–18, 222–25
- Business Investor model: adaptive behavior in, 154–55; empirical model of behavior in, 210–11; with links for sensing a network, 144–45; objective function in, 158–59, 164–68; ODD description of, 137–39; prediction in, 162–64
- Butterfly Hilltopping model: analysis of, 69; emergence in, 103–4; implementation in NetLogo of, 50–59, 64–68, 69–74; observation of, 121–22; ODD description of, 43–46; real landscape data imported for, 71–74; testing software of, 87–91
- buttons: context of, 18; creation of on Interface, 19–20, 53, 55; Forever button, 20, 55; selecting and modifying, 55; step button, 83; for test and display procedures, 85, 122. *See also* syntax errors and checker
- calibration and parameterization: automated, 321; categorical vs. best-fit, 267; criteria for, 269–70; definition and purposes of, 263–64; and differences between traditional modeling and ABMs, 264–65; documentation of, 276; literature on, 266; measures of model fit for, 268–70; overfitting in, 266, 272; parameters for, 266–67, 294; purposes of, 264; statistics for, 295–96; strategies for, 266–64; with stochastic results, 271; of submodels, 43, 265–66; and theory testing, 263–64; time-series, 268–69
- carefully, 126–27
- checklist: of design concepts, 41–42; of NetLogo elements, 30–31
- clear-all, 22; and BehaviorSpace, 109, 302; variables on Interface not set to zero by, 66
- Code Examples in NetLogo Models Library, 16, 69, 125–26, 208
- collectives (model design concept), 41–42; definition of, 215; as emergent properties vs. explicit entities, 215; represented via breeds, 217–18; represented via patches or links, 216–17; Wild Dog Model example, 219–227; Woodhoopoe Model example, 257
- comma-separated values (.csv) files, 71, 125–28; CSV extension to write, 125; and European computers, 71, 126; produced by export primitives, 70, 128
- Command Center, 26–27, 83, 85
- comments in code, 57–59, 60; for temporarily deactivating code, 58, 66, 92, 293, 302; in version control, 64
- communication of models and science, 10, 35, 260, 300
- competition, 139, 173, 174; in the BEFORE forest model, 241, 243–44; and scheduling model actions, 186, 192–94, in the Telemarketer Model, 176–77, 180, 194; in the Wild Dog Model, 218, 221
- conceptual framework for agent-based modeling, 41–42, 99–100
- contexts in NetLogo programming, 17–18, 53, 57–58, 79, 81, 134, 136, 142; and breeds, 217; and foreach, 193; using local variables in multiple contexts, 74
- continuous time simulation. *See* discrete event simulation
- contour plots: in calibration, 273; using Excel, 164–66; of parameter interactions in sensitivity analysis, 304–6; in submodel analysis, 164–68; using R, 166–67
- Cooperation model, 174
- copying of code: benefits and ethics of, 125–26; as cause of errors, 78–79
- count, 27, 67, 196
- create-link(s)-to, 144, 182, 224, 227
- create-turtles (crt), 23, 54; and breeds, 217; context started by, 18; and initialization of turtle variables, 23, 144; order of creation vs. initialization in, 144
- csv files. *See* comma-separated values files
- currencies of model results, 292–93; in sensitivity analysis, 301, 306
- debugging of code. *See under* software
- decision-making traits. *See* adaptation and adaptive behavior
- deduced variables, 39, 245
- defensive programming, 205–7, 224, 309
- design concepts: as a conceptual framework, 99–100; in ODD protocol, 41–42
- detective work in modeling, 78, 95, 246, 295, 296, 312–13
- dictionary of NetLogo primitives, 17, 22; F1 key to access, 21–22
- discrete event simulation, 189, 194–98, 322
- display, 121
- display. *See* Interface tab in NetLogo; View
- distance, 114
- distance to nearest other agent set by, 114–15, 153
- documentation: of analyses, 171, 238, 282; of software tests, 58, 91–92. *See also* ODD protocol for model description; TRACE format for describing a modeling cycle

- emergence (model design concept), 10, 42, 103–4, 116; of collectives, 216; criteria for, 103; in Flocking model, 111–12
- entities: collectives as, 215–18; in ODD protocol, 38–39; selection of, 9, 239–40
- environment: emergence from, 103–4, 116, 173; modeling of, 17, 38–39, 40, 43, 133, 194, 321
- epidemic models, 315–16; for rabies 3–4
- error-message, 126–27
- evolutionary modeling of behavior, 320
- Excel spreadsheet software, 126, 164–66, 273
- execution speed, 109, 321–22. *See also* speed controller
- export- primitives, 70–71, 128
- extensions to NetLogo, 208–9, 320–21; CSV, 125; GIS, 71, 320; Networks, 173; R, 209, 321
- extinction: risk of in Wild Dog model, 218, 228, 296, 302–5, 310–12; in the Simple Birth Rates model, 105–10
- false. *See* boolean conditions, reporters, and variables
- file- primitives, 72, 87–88, 125–28
- files: input, 43, 72; output, 88, 124–28; problems with due to multiple processors and BehaviorSpace, 109–10; for software testing, 87–88, 89–90. *See also* BehaviorSpace; comma-separated values files; export- primitives
- filter, 184
- filtering agentsets, 151
- fitness, 41, 42, 45, 149–50, 256, 258. *See also* objectives
- Flocking model: analysis of using BehaviorSpace, 112–15; calibration exercise with, 278; collectives in, 216; emergence in, 111, 116; related scientific models, 112, 216, 253
- foreach, 86, 139, 193
- forest model. *See* beech forest model BEFORE
- forward (fd), 25–26
- fput, 86, 184
- Geographical Information Systems (GIS) extension for NetLogo, 320, 322
- global variables, 17, 18, 132–33; and clear-all, 22; controlling in BehaviorSpace, 107, 109; defining of, 23, 50; initialization of, 56, 67; movement to Interface of, 66; and the observer, 17; risks of using, 133; typical uses of, 17, 133
- globals, 23, 50, 132; and variables on Interface, 66
- go procedure, 18, 19, 21, 23–24, 190–92; and BehaviorSpace, 109, 192; as the model schedule, 23–24, 50, 190–92; number of times executed, 27, 55, 109; value of keeping simple, 24, 48, 124
- hatch, 18, 27, 134, 217, 224
- header information in output files, 125–27
- help sources, 29, 33, 64. *See also* instructors' guidance
- heuristic models of behavior, 258, 319, 320
- heuristics for developing and analyzing models, 9, 286–95, 296
- analyze simplified versions of your model, 293
  - analyze from the bottom up, 294
  - at an interesting point in parameter space, keep the controlling parameter constant and vary other parameters, 290
  - explore unrealistic scenarios, 294
  - find “tipping points” in model behavior, 288
  - look for striking or strange patterns in the model output, 290
  - run the model step by step, 289
  - try different visual representations of the model entities, 289
  - try extreme values of parameters, 288
  - use several “currencies” for evaluating your simulation experiments, 292
- hierarchies among agents, representing via scheduling, 192
- Hilltopping Butterfly model. *See* Butterfly Hilltopping model
- histogram and histograms, 123–24, 178, 257, 311
- hypothesis testing: in model analysis, 282, 285, 286–95; in software testing, 87; for theory development, 237, 250–55
- ifelse and ifelse-value, 24–25, 55–56, 82, 134–35, 185
- in-cone, 151
- in-radius, 21–22, 79–80, 142–43, 151–53
- individual-based ecology, xii, 9
- individual-based models. *See* agent-based models
- initialization, 18–19, 22, 43; and BehaviorSpace, 109, 302; of an empty agentset using no-turtles, 112; via the Interface, 66; of lists, 183–84; using nobody to designate a turtle, 181; via the setup procedure, 19, 50, 52; stochastic, 202, 209; of variables, 56, 67
- input data, 43, 50, 320–21. *See also* files
- instructors' guidance, xiv–xvi
- interaction: between agent and environment, 42, 59, 61; among agents, 42, 173–74; direct vs. mediated, 173; local vs. global, 12, 173, 174–75, 294;

- interaction (*continued*)
  - among model processes, 290, 295; parameter, 303–5, 306–7; programming of long-term, 181–82; among system levels, 10
- Interface tab in NetLogo: addition and modification of elements in, 19–20, 53, 55, 123–24; and `clear-all`, 22, 66; hiding code in, 70, 107–8, 122; input, 129; monitor, 119; movement of variables to, 66; observer context of, 18, 132–33; output, 68, 119. *See also* Agent Monitors; Command Center; Model Settings dialog; plots; speed controller on NetLogo Interface; View
- internal organization of systems, 233–35, 242–45
- `label` and `plabel`, 61, 83, 120–22
- landscapes. *See* spaces and landscapes
- learning (model design concept), 42, 98, 259
- length, 184
- `let`. *See* local variables
- `link-neighbors` primitives, 144–45, 151, 182, 223–26
- links in NetLogo, 17, 134, 144–45, 173, 181–82; for associating collectives with their members, 223–27; for representing collectives, 216–17
- `links-own`, 134
- lists: arrays and tables as alternatives to, 208; definition and initialization of, 85, 183–84; in calculating statistics, 135; for modeling memory, 85–86, 183–84; produced from an agentset by `of`, 135; sorting of, 193
- local randomness, 209–10, 309–10
- local variables: containing agentsets, 93, 151; created by `let`, 67; defined, 134–35; and inputs to reporters, 141; reasons for using, 73–74, 134–35, 136
- log-normal distribution, 206, 214
- logistic functions, 220, 221–22
- logistic regression, 203, 212
- `loop`, 196–97
- loops: in NetLogo, 196–97, 205–6, 309; vs. NetLogo's style, 29, 139–40
- management accounting and collusion models, 245–46
- management applications of models, 3–4, 38, 218–19, 227–28; and model uncertainty and robustness, 310–12, 314
- market models, 11, 40, 253–54
- Mathematica, 162, 321
- `max` and `min`, 73, 124, 135
- `max-n-of` and `min-n-of`, 151
- `max-one-of` and `min-one-of`, 114–15, 141, 152, 153
- `max-pxcor` and `max-pycor`, 16, 18, 51, 80
- mean, 17, 68, 123, 128, 135, 217
- memory, models of, 85–86, 182–85
- model, definition of, 4–7
- model fit measures, 265, 266, 267–69, 271
- Model Settings dialog, 18, 51–52, 72; errors caused by, 80
- modeling cycle, 7–12, 37, 63, 119, 238, 250–51, 281, 318–19; TRACE format for documenting, 12, 43, 91, 238, 282
- Models Library of NetLogo, 16, 63, 125–26, 208, 318; limitations of, 63, 318
- modular code, 141
- monitors. *See* Agent Monitors
- Mousetrap model, 194–98
- `move-to`, 17, 26, 54–56
- multiagent systems, xii
- multicriteria assessment of models, 234
- Mushroom Hunt model: as calibration exercise, 278; as example modeling problem, 5–10; as introductory programming example, 18–28; as modeling cycle example, 7–9; as searching behavior example, 157
- `myself`, 93, 114–15, 136, 141–42, 152–53, 175
- `n-of`, 21–22
- nearest other agent, 114–15, 152–53
- `neighbors`, 17, 80, 87, 90, 93, 140, 143
- NetLogo: checklist and mini-reference, 30–31; extensions, 73, 125, 173, 195, 208–9, 320–21; influence of on model and research design, 30; installation, 16; introduction to, 16–18; ODD protocol correspondence with, 49–50; personality and style of, 29; suitability for large models and alternatives to, xiii–xiv, 313–15; terminology, 17–18; Users Group, 33; versions, xvi, 16; why to use, xiii–xiv
- networks: in the Business Investor model, 144–45; in the Models Library, 173; representing in NetLogo, 144–45, 208, 217
- neural network models of behavior, 320
- nonspatial models, 110, 180, 220, 229, 294
- null submodels and theory, 252, 254, 255, 319
- objectives (model design concept), 41–42, 149–50; in the Business Investor model, 154; in the Woodhoopoe model, 258–59
- objectives of this book, xii–xiii
- observation (model design concept), 42, 119–20; of the Butterfly model, 64–68; to facilitate

- pattern-oriented modeling, 240; of NetLogo models, 120–28
- observer and observer context in NetLogo, 17–18, 50, 132–33
- ODD protocol for model description, 35–43; and standardization of terminology, xii; and corresponding elements in NetLogo, 49–50; and design of models, 36, 99–100; template document for, 37
  - of the BEFORE beech forest model, 242–44
  - of the Breeding Synchrony model, 74–75
  - of the Business Investor model, 137–39
  - of the Butterfly Hilltopping model, 43–45
  - of the Culture Dissemination model, 90–91
  - of the Harvester model, 96
  - of the Segregation model, 286–87
  - of the Telemarketer model, 175–79
  - of the Wild Dog model, 219–21
  - of the Woodhoopoe model, 257–59
- of, 18, 135–36, 175
- one-of, 54; agentset converted to an individual agent by, 182
- or, 152
- other, 113–15, 144, 151–53
- output. *See* files; Interface tab in NetLogo; observation
- overfitting in calibration, 266, 272, 276
- panic, when not to, 9, 18, 275. *See also* worry
- parameters: definition of, 23, 39; documentation of, 43; estimating values of, 263–72; as global variables and on the Interface, 23, 66, 133; interaction among, 304–7; for logistic functions, 221–22; model sensitivity to, 104, 299–300, 301–7; model uncertainty due to, 264, 307–12; parameter space, 271; of random number distributions, 202, 205–9, 212, 219; reference (standard) values of, 301, 302; World settings as, 18, 51–52, 80, 146; *See also* calibration and parameterization
- patch, 72–73
- patch-here, 67, 85–86, 140–41, 152–53
- patch-set, 140–41
- patches-own, 50, 51, 133
- pattern-oriented modeling, 233–35, 239–41, 249–52, 263–64
- patterns: characterization of system and problem by, 234–36; criteria for matching, 236, 260–61, 269–70; in software testing, 82–83
- pen-down and pen-mode, 27
- plabel, 83, 121–22
- plot, 70, 123
- plots, 69–70, 123–24; export of, 70–71, 128. *See also* contour plots
- Poisson distribution, 169, 207–8, 212, 221
- positive feedbacks, 192
- precision, 83, 121, 124
- prediction (model design concept), 42, 161–64, 169–70
- predictions from models: absolute vs. relative, 311–12; independent or secondary, 244–45
- primitives, 17; errors due to misunderstanding of, 79–80; provided by extensions, 320
- probability: Bayesian updating of, 169–70; logistic model of, 220–22; in objectives for adaptive behavior, 159, 168, 252; observed frequencies as, 202–3, 210–11; and parameters in sensitivity analysis, 301–2, 304; in prediction, 169–70, 171; random distributions and theory of, 204–8; in software testing, 87
- problem addressed by a model. *See* question addressed by a model
- processes in a model: description of, 40–41, 43; design of, 7–9, 39, 233–34, 239, 240–41, 249–252, existing models of, 320–21
- programming practices, 57–59, 60, 64–65, 67, 68–69, 77–78, 81–82, 86, 91–92, 94–95, 125–26, 205
- pseudocode, 41
- pseudorandom numbers, 204
- question (problem, purpose) addressed by a model, 4–7, 11, 38, 233–34, 318–19
- R statistical software, 89, 162, 166, 209, 306, 321
- rabies control models, 3–4
- random, 24–25, 207
- random-float, 55–56, 205
- random-normal, 204, 205–6; converted to log-normal distribution, 214
- random number distributions, 202–8
- random number generation and seeds, 209–10; and BehaviorSpace, 209–10; use of in uncertainty analysis, 309–10
- random-poisson, 207–8
- random-seed. *See* random number generation and seeds
- regimes of control, 288
- reimplementation: of existing models, 318–19; of submodels to test software, 89–91
- repeat, 196
- replication and repetitions, 105–7, 115, 203; in calibration, 271; and random-seed, 209; in sensitivity analysis, 302; in statistical analysis of results, 295

- reporters, 17, 67, 141–42, 175; in BehaviorSpace and Interface elements, 107, 128, 273; as boolean logical conditions, 185
- reset-ticks, 27, 191–92
- resize-world, 72, 80
- right, 25
- robustness analysis, 299–300, 312–13
- run-time errors, 80, 322. *See also* defensive programming
- satisficing decision traits, 155–58
- scale selection, xiii, 9, 39–40, 174, 191, 233, 265. *See also* pattern-oriented modeling; scheduling
- scale-color, 52–53, 83, 120, 122–23
- scenarios in simulation experiments, 106–7; contrast of, 115, 295; and uncertainty analysis, 310–12; usefulness of unrealistic, 59, 294–95, 313
- scheduling, 23, 40–41, 189–92; discrete event, 194–98; and the `go` procedure, 50; to represent hierarchies, 192–93; using time steps, 190–92; of View and observer actions, 121
- science and agent-based modeling, xii–xiii, 9–10, 36, 63, 77, 86, 99, 112, 149, 234, 250, 319–20
- scientific method. *See* strong inference and scientific method
- scope of variables, 132–35
- Segregation model, 286–91, 314
- sensing (model design concept), 42, 131–32, 136–37; of an agentset, 139–40; in networks of links, 144–45
- sensitivity analysis, 267, 300, 314, 321, 322; global, 306–7; parameter interactions in, 304–6; single-parameter, 301–4
- sensitivity experiments, 106, 116, 227–28, 288, 313
- set, 21
- set-current-plot, 124
- Settings, Model. *See* Model Settings dialog
- setup procedure, 19–24, 52–54, 71–74, 126; interaction of BehaviorSpace with, 109, 112, 302; and initialization element of ODD, 50, 52; and variables on Interface, 66. *See also* initialization
- show, 26–27, 58, 69, 83, 153
- Simple Birth Rates model, 104–11
- simulation experiments, 65, 251, 285, 296, 318; examples of, 65–69, 104–11, 112–15, 142–43, 154–55, 162–64, 179, 227–28; replication of, 203. *See also* calibration and parameterization; robustness analysis; sensitivity analysis; theory; uncertainty analysis
- skeletons of procedures, 21, 52, 60, 81–82
- sliders, 55, 66, 133; and BehaviorSpace, 107, 109
- software: alternatives to NetLogo, xiii–xiv, 321–22; common errors in, 78–81; testing and verification of, 81–91. *See also* programming practices
- sort-on, 193
- spaces and landscapes, 38–40; benefits of simplifying, 59, 293, 313; in the Business Investor model, 137; in the Butterfly model, 44, 51–53, 71–74; from imported data, 72–73, 320; of irregular polygons, 322; non-geographic, 10, 74, 137, 173; scales of, 39–40; wrapping, 22, 80
- spatial extent, 40, 51–52
- speed controller (slider) on NetLogo Interface, 26, 83, 121, 192
- spreadsheets: for preparing input and analyzing results, 71, 72, 109–10, 273; for reimplementing submodels, 87–91; to test and explore submodels, 162, 164–67, 170
- stability properties, 293
- standardization of agent-based modeling, xiii, 36
- state variables, 9, 38–39, 43. *See also* variables
- statistical analysis: for software testing, 87–88, 95; of model results, 203, 295–96, 306, 321
- stochasticity (model design concept), 42, 201–3; analysis of via replication, 203; in calibration, 271; in models of behavior, 210–11, 219; and sensitivity analysis, 301–3; in uncertainty analysis, 307; uses of, 202–3
- stop, 55, 68, 109; in looping, 196
- Stopping rules and stopping models, 55, 198; in BehaviorSpace, 107–8, 109
- strong inference and scientific method, 234, 250
- structure and structural realism of models, 7–9, 38–40, 233–34, 239–40, 264
- stylized facts, 234
- submodels: behavior submodels, 149; and corresponding procedures in NetLogo, 50; definition of, 40, 43; description of in ODD protocol, 43; implementation and analysis of, 164–70, 171, 294; independent reimplementations of, 89–91; parameterization of, 265–66
- syntax errors and checker, 20, 78, 79, 81–82
- Telemarketer model: analysis of, 179; addition of mergers to, 180–82; collectives in, 216; with customers remembering callers, 182–85; ODD description of, 175–79; potential bias in due to execution order, 186, 192, 194
- temporal extent, 39–40, 55, 198
- testing software. *See under* software
- theory in agent-based models: definition of, 249–50; development and testing of, 250–52, 319–20; examples of, 252–55
- theory potentially useful in adaptive traits: Bayesian updating of probabilities, 169–70;

- evolutionary modeling, 320; game theory, 245–46; objective optimization in decision analysis and behavioral ecology, 154, 251–52; probability, 169, 204; satisficing, 155–58; simple heuristics, 319–20; state- and prediction-based theory, 252
- tick and ticks, 27, 55, 107, 121, 191–92; and BehaviorSpace, 107
- tick-advance and non-integer ticks, 195–96
- time in models. *See* scheduling
- time steps, 39–40, 190; consequences of using, 191–92. *See also* scheduling; tick
- topology of NetLogo's world, 22. *See also* Model Settings dialog
- to-report, 67, 141–42
- TRACE format for describing a modeling cycle, 12, 43, 91, 171, 238, 276
- trade-off decisions, 150, 154–55, 162–64, 251–52
- troubleshooting tips, 68–69
- true. *See* boolean conditions, reporters, and variables
- turtles-here, 121–22, 134, 140, 143, 151; to identify members of a patch collective, 216
- turtles-on, 79–80, 151
- turtles-own, 25, 50, 134, 217
- uncertainty: in calibration data, 270; not quantified by replication, 203; reduction of via calibration, 260, 266; in sensing information, 131; structural vs. parameter, 264; and theory development, 260
- uncertainty analysis: definition of, 300; methods for, 307–312; of relative vs. absolute predictions, 310–12
- underdetermined models, 239
- uphill, 17, 56, 91
- User Manual for NetLogo, 15–17, 26
- user-file, 72
- utility (economic objective), 41; and Business Investor model, 138, 149–50, 154–59, 164–68
- validation of models, 244–45. *See also* pattern-oriented modeling; theory in agent-based models
- variables: of breeds, 217; built-in, 17, 23, 51, 120; choosing (*see* structure and structural realism of models); that contain agents and agentsets, 67; discrete vs. continuous, 26; local, 67, 73–74; names of, 58, 86; of other objects, 135–36; scope of, 132–35; types of, 17, 51. *See also* state variables
- version control, 64–65
- View, 120–23; changing size of, 52; in continuous-time models, 197–98; updating of, 121, 191–92. *See also* Interface tab in NetLogo
- wait, 121
- website for this book, xvi
- while, 72, 196–97; to catch normal distribution outliers, 205–6, 309
- Wild Dog model, 218–28; ODD description of, 219–21; sensitivity analysis of, 302–4, 304–6; uncertainty analysis of, 307–11
- with, 67, 141, 151–52
- with-local-randomness, 209–10, 212, 309–100
- with-max and with-min, 114–15, 151, 152–53
- Woodhoopoe model: calibration of, 272–75; ODD description of, 257–59; as theory development exercise, 256–59, 260–61
- word, 70, 83; example uses of in error statements, 207, 225; to produce .csv output files, 127; to produce output files from BehaviorSpace experiments, 110
- world and wrapping, 22, 80. *See also* Model Settings dialog
- worry, why not to yet, 41, 47, 64, 94, 112, 145, 270. *See also* panic

## Index of Programming Notes

- A simple way to import spatial data, 72–73
- A very brief introduction to output files, 88
- Ask-concurrent, 193–94
- BehaviorSpace, multiple processors, and output files, 109–10
- Clear-all, 22
- Copying code, 125–26
- CSV files, 71
- How BehaviorSpace works, 109
- Logical conditions are reporters, 185
- Modifying Interface elements, 55
- Moving variables to the Interface, 66
- Numerical recipes and NetLogo extensions—don't reinvent the wheel again!, 208–9
- Parameterizing and programming logistic function, 221–22
- Shortcut keys for editing, 20
- Sensing the members of an agentset, 139–40
- Tracking relationships and long-term interactions via variables and via links, 181–82
- Troubleshooting tips, 68–69

Updating display variables to represent agent states, 122–23  
Using `repeat`, `loop`, and `while` to execute code repeatedly, 196–97  
Using `-set` to merge agentsets, 140–41  
Version control, 64–65  
When is the View updated?, 121  
Writing reporters, 141–42