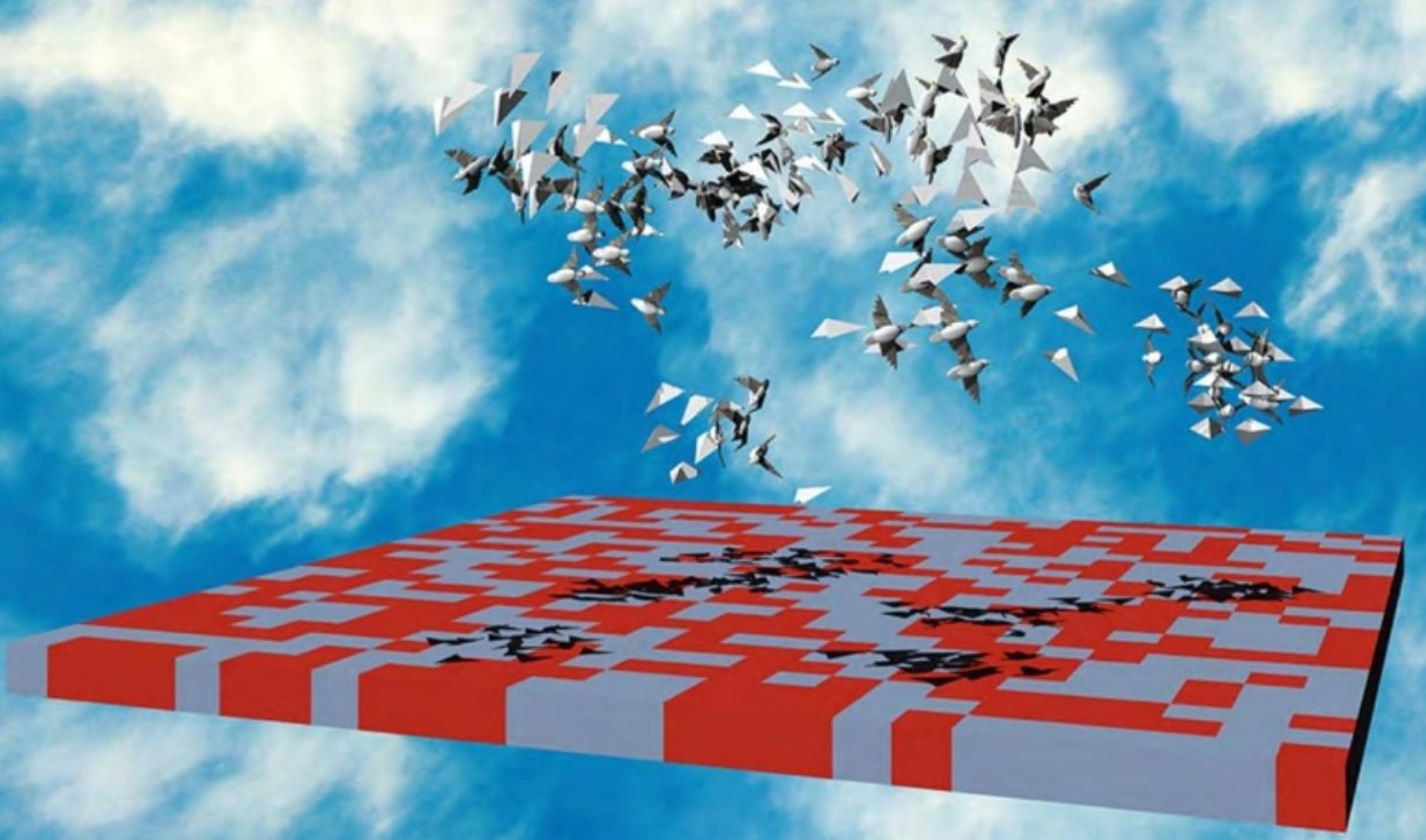
AN INTRODUCTION TO

AGENT-BASED MODELING

MODELING NATURAL, SOCIAL, AND ENGINEERED COMPLEX SYSTEMS WITH NETLOGO



URI WILENSKY AND WILLIAM RAND

An Introduction to Agent-Based Modeling

Modeling Natural, Social, and Engineered Complex Systems with NetLogo

Uri Wilensky and William Rand

The MIT Press
Cambridge, Massachusetts
London, England

This book was set in Times LT Std 10/13pt by Toppan Best-set Premedia Limited, Hong Kong. Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data
Wilensky, Uri, 1955–
An introduction to agent-based modeling : modeling natural, social, and engineered complex systems with NetLogo / Uri Wilensky and William Rand.
pages cm
Includes bibliographical references and index.
ISBN 978-0-262-73189-8 (pbk. : alk. paper) 1. System analysis—Data processing. 2. Computer simulation. 3. Multiagent systems. 4. NetLogo (Computer program language) I. Rand, William, 1976– II. Title.
T57.62.W54 2015
003'.3—dc23

2014023747

10 9 8 7 6 5 4 3

Contents

agent-based modeling (ABM) will play a central role in that toolkit.[1] Through this book, we will provide an introduction to ABM for anyone interested in answering questions about the complex systems that are embedded in natural, social, and engineered contexts.

Our book will draw on applications in a wide variety of fields to help illustrate the power of ABM methodology. Along the way, we will guide you through a series of hands-on examples that will help you to understand how you can use this tool in your own work. The guiding principle we used while writing this textbook (and also for NetLogo, the language we will be using) is, "Low Threshold, No Ceiling."[2] By that, we mean that there is very little prerequisite knowledge to start using the material, but at the same time there is no limit to what can be accomplished once it is mastered.

ABM is a powerful tool in helping us to understand complex systems. Our textbook, though titled "An Introduction," provides the tools necessary to enable you to build and use agent-based models to investigate your own questions.

Who We Wrote This For

Because the field of ABM is applicable to so many domains, this textbook can be used in a wide variety of contexts. It can serve as a main text for an interdisciplinary undergraduate course on complex systems or a computer-science class on agent-based modeling. It can be used as a supplementary text in a very wide range of undergraduate classes, including any class where agent-based modeling can be profitably applied. This can be quite a broad list of content areas. The material described in this textbook has been used in natural science classes such as physics, chemistry, and biology; social science classes such as psychology, sociology, and linguistics; and engineering classes such as materials science, industrial engineering, and civil engineering. We have strived to balance the examples across content areas, in order for at least one example to hit squarely on the content area of a given course. Naturally, to achieve this goal, we must sacrifice depth in any one content area. As the field of ABM progresses, we expect that in-depth domain-specific textbooks will appear.

Though we have targeted a high-level undergraduate or entry-level graduate audience, we expect that the book will be useful to other audiences as well. The prerequisite knowledge is not great; a motivated individual learner in a wide range of academic settings could use this text. Similarly, since ABM methods may be new to many graduate students, we expect that our book could be quite useful as a supplementary text to graduate classes in a wide range of subject domains. ABM methods are increasingly used in research labs, in the business community, and in policy circles. We anticipate that professionals in these areas can benefit from the methods and examples herein. The material for this textbook

1. If ABM is used in a plural sense (ABMs) or with an article (an ABM) then ABM stands for agent-based model and not agent-based modeling.

2. In the appendix, we discuss the history of the Logo programming language from which this slogan originated. It can also be rendered as "Low Threshold, High Ceiling."

has arisen from and has been tested for over two decades of both undergraduate and graduate classes taught in Computer Science and in Learning Sciences by Uri Wilensky and additional classes taught by William Rand, as well as hundreds of workshops, seminars, and summer school courses conducted by both authors.

In the title, we specifically highlight "natural, social, and engineered complex systems." Natural systems are studied in fields such as biology and physics: complex systems that are naturally occurring. Social systems consist of individuals interacting with each other. Social systems can be natural and/or engineered. Engineered systems are those that have been designed by humans to achieve a particular goal.

There are few prerequisites for this book. We do not assume any mathematical knowledge beyond basic algebra, and we do not assume any prior programming knowledge. For chapters 6, 7, and 8, we do assume that you have a very basic knowledge of statistics—for example, that you understand what a normal distribution is. Moreover, as we discuss later, we expect you to start with a rudimentary familiarity with NetLogo, and **we suggest that you work through the first three tutorials included in the NetLogo user manual of the NetLogo software.** The NetLogo software can be downloaded from ccl.northwestern.edu/netlogo/.[3]

Working through this book will require reading and writing of computer code. This may be quite unfamiliar to some readers. Though many people believe that computer programming is too hard for them to learn, research by Constructionist educators and decades of experience has shown the authors that virtually all students can learn to program in NetLogo. We hope that you won't be scared off by all the code in the textbook, and that you will take the time to learn to read and write the code. We are very confident that you can do it and that taking the time to learn it will pay rich dividends.

NetLogo and the Textbook

Many different agent-based modeling languages exist. As of this writing, NetLogo remains the most widely used. Of the others currently in use, Swarm, developed at the Santa Fe Institute, Repast, developed at Argonne National Laboratory, and MASON, developed at George Mason University, are the next most pervasive among scientists and researchers. Most ABM toolkits (including NetLogo) are open source and freely available. AnyLogic is a commercial package that has also been successful. Other software packages for building ABMs in current use include Ascape, Breve, Cormas, MASS, PS-I, and SeSam. It is also possible to write an agent-based model in any language. When building your own model in a standard non-agent-based programming language, the time to run the model may be faster, but often the time for the lifecycle of development will be considerably slower.

3. This textbook uses the desktop application version of the NetLogo software. At the time of publication of this book, a browser-based version of NetLogo will also be available. Most examples used in the book will work in the browser-based version, though some will require adaptation for that version.

We make use of NetLogo examples throughout this textbook, both to provide hands-on illustrations of the principles being discussed and also as a form of "pseudo-code."[4] However, this textbook is not a NetLogo manual. Before reading much farther, we suggest you **download the NetLogo software,** open source and available for free from **http://ccl.northwestern.edu/netlogo,** and work through the introductory material in the NetLogo manual (available through the Help menu of the NetLogo software), including the three tutorials that introduce the user to basic model development and syntax. This is a necessary prerequisite to comprehending the material in the book beyond chapter 1. The NetLogo user manual (found under the NetLogo Help menu) is the authoritative reference for NetLogo and it is regularly updated and improved. It will often be advisable to consult the manual, the interface and programming guides, the dictionary and FAQ throughout this textbook. NetLogo comes with an extensive library of models from which you can learn common code patterns. The "Code Examples" models are meant to be simple models intended to show you common code patterns. There are many online resources for help with NetLogo (see http://ccl.northwestern.edu/netlogo/resources.shtml). The NetLogo users group (reachable through the NetLogo HELP menu) is a forum where people post NetLogo questions. The community is quite responsive, and if you post a question there you will likely receive a prompt response. It is a good idea to subscribe to this group early on in working through this textbook. Another resource for asking and answering NetLogo programming questions is Stack Overflow (http://stackoverflow.com/questions/tagged/netlogo), where you will find many questions and answers about NetLogo programming.

We use the NetLogo examples in order to concretize the concepts of agent-based modeling that we present. By presenting computer code side-by-side with conceptual discussions, we hope to provide both a larger picture of the use of agent-based models as well as a specific illustration.

We have selected NetLogo as our ABM language for this text for several reasons. Before explaining the specifics, it is important to say that we believe our book will be just as useful for those using other ABM languages. Even if we had written a completely language-agnostic book, we would still have needed to include pseudo-code to illustrate our points. Since NetLogo was designed to be easily readable, we believe that NetLogo code is about as easy to read as any pseudo-code we could have used. NetLogo also has the big advantage over pseudo-code of being executable, so the user can run and test the examples. Moreover, there are a large number of agent-based models written in NetLogo in a wide variety of domains, so a literate agent-based modeler requires at least a passing understanding of NetLogo.

4. Pseudo-code is an intermediate form between text and computer code that is often used to describe computational algorithms.

We have several other significant reasons for choosing NetLogo as the language for our book. Uri Wilensky, the first author of this textbook, is also the author and developer of NetLogo and has conducted agent-based modeling research, development and teaching with NetLogo (and its precursors) for over two decades. He has taught NetLogo in his classes and in workshops for that same period. The second author has years of experience teaching NetLogo workshops and conducting agent-based modeling research with NetLogo. As such, we are intimately familiar with the language nuances and details. More important, NetLogo was designed with great attention to learnability. As we said, its core design principle is "low threshold, no ceiling."

Achieving both of these goals completely is not possible and to some degree, they trade-off against each other, but NetLogo has gone a considerable distance in achieving both. No other extant ABM language is close to NetLogo's low threshold. As such, it is an ideal language for learning ABM and is used widely in classrooms all over the world. Yet, NetLogo also achieves a high ceiling. NetLogo is in use by a large number of scientists and professionals and is regularly employed in cutting edge research (see http://ccl.northwestern.edu/netlogo/references.shtml for a partial list of research papers employing NetLogo). So after completing this text, you should be well prepared to use NetLogo in your research, teaching, and/or professional life. Learning NetLogo will make you a better ABM modeler/researcher regardless of the language that you may eventually use.

Learning Objectives

There are ten main objectives of our textbook, which roughly parallel the nine chapters and the appendix of the textbook. We will phrase these objectives in terms of questions a reader should be able to answer at the end of our textbook:

0. Why does agent-based modeling provide us with a unique and powerful insight into complex systems?
1. What is agent-based modeling and how is it used?
2. What are some simple agent-based models that we can create?
3. How do I extend an agent-based model that was created by someone else?
4. How do I create my own agent-based model?
5. What are the basic components of agent-based modeling?
6. How can I analyze the results of an agent-based model?
7. How can I tell if the implemented agent-based model corresponds to the concept of the model that I developed in words? How can I tell if the results of my agent-based model tell me anything about the real world? How can I make sure that someone else can repeat my results?

8. What are some advanced ways of including data and using output from agent-based models? What are some of the open research questions in agent-based modeling?
9. From what computational scientific roots did agent-based modeling arise?

Features

There are four main features that are present in nearly every chapter of the book: in-line development, textboxes, explorations, and references. In-line development is what we call the approach of developing a model or an extension to a model in the main text of the book instead of asking the reader to do it later offline. It is expected for some of these chapters (2, 3, 4, 5, 6, and 7 especially) that the reader will read the textbook while sitting at their computer, entering the code from the textbook and manipulating the models as they read along. **All models developed in the textbook can be found in their entirety in the IABM Textbook folder of the NetLogo models library.** This folder is organized by chapters of the textbook. These completed models are provided as a resource, but we encourage students to follow along with the textbook, developing the models themselves before opening the models in the IABM Textbook folder. Most other models referred to in the textbook reside in the NetLogo models library, which can be accessed through the "models library" menu item from NetLogo's "file" menu. The models library is itself comprised of subfolders. The principal subfolder is "sample models," which is organized by subject, such as biology, mathematics, social science, and others. **All models used in the book and supplementary materials can be found on the website for the book, www.intro-to-abm.com, updated regularly.**

The textboxes serve to provide additional information and concepts. There are three kinds of textboxes: (1) definitional, (2) exploration, and (3) advanced concept. *Definitional* textboxes (set in blue) define words and terms critical to the discussion. *Exploration* textboxes (set in green) provide additional explorations that are relevant to the material being discussed at that point. *Advanced concept* textboxes (set in orange) highlight concepts that are beyond the scope of the book but that might be of interest to the reader.

At the end of every chapter (except chapter 0), there are explorations. An instructor may want to draw on these explorations for student homework. These explorations are roughly in order of difficulty. Some of them are quite open-ended, while others are more constrained.

At the end of the book there is a references section, the text references organized alphabetically, and the software/model references organized by chapter. To distinguish the text from the software references, the software references are italicized in the text.

Organization

We will present the basics of ABM in nine chapters and an appendix. The nine chapters and the appendix can be thought of as three sections: (1) What ABMs are, (2) How to

complex patterns can "self-organize" without a central leader, and (4) the same phenomenon can be modeled in many different ways, depending on which aspect of it you want to emphasize.

Chapter 4 takes us to the next step of starting to develop models from scratch. We develop a model, named Wolf Sheep Simple, in five distinct stages. The model that we develop illustrates many of the basic concepts that go into ABM creation and design, and it includes many core features of ABMs, such as different agent types, environmental and agent interactions, and competition for resources. Central to this chapter is the delineation and elaboration of the key design principle of ABM.

In chapter 5, we zoom out to take a look at the broader picture, cataloguing and describing the constituent parts of an ABM. After classifying these components into the five categories of Agents, Environments, Interactions, User Interface/Observer, and Schedule, we examine each of these component classes in turn. Within each of these classes, we discuss common issues that should be considered before and while building an agent-based model. For example, what kind of topology should the environment have? Or what properties and actions do the agents in the model have? It can be quite useful to make a plan for each component class and their interactions at an early point in the design plan, and to revisit that plan throughout the design process.

Together, these four chapters provide an in-depth treatment of how to construct agent-based models and provide the basic tools needed to design, construct, and select the components that will go into a model. After reading these four chapters, you should have sufficient knowledge to design and construct basic agent-based models.

Chapters 6, 7, and 8: How to Analyze and Use ABMs

One feature of many ABMs is that they create large amounts of data. Chapter 6 discusses how to examine these data to find meaningful relationships and conduct analyses that are useful for understanding the behavior of the model. We discuss how to explore and describe model results using statistics, graphs, and geographic and network methods. We also discuss the importance of multiple model runs and how to sweep a parameter space to determine the range of behaviors exhibited by a model.

In chapter 7, we examine three key concepts in modeling: verification, validation, and replication. Verification is the process of comparing an implemented model to its associated conceptual model and investigating whether the implemented model is faithful to the conceptual model. By verifying a model, we show that the model implementer has comprehended the intended micro-level rules and mechanisms and has correctly implemented them in the model code. This is true even if the model author may have intended different emergent behavior or predicted different emergent results from those that arise in the implemented model. Validation is a comparison of an implemented model to the real world, to see if the results of the implemented model give us insight into the corresponding real world phenomenon. Validation enables us to use the model to make statements about the

real world. Replication is the reproduction of a model result published by one scientist or model developer by another scientist or model developer. Replication is a central process in the creation of scientific knowledge, and replication procedures for agent-based models are explained. Overall, this chapter discusses how implemented ABMs relate to both other models and the real world.

In chapter 8, we address advanced topics with regard to agent-based modeling and how the methods that we have described in the first eight chapters can be further refined. We focus on ways to incorporate data into ABMs from advanced sources such as social network analysis, geographic information systems, and real-time sensors. We also discuss how to make ABMs more powerful, by incorporating techniques such as machine learning, system dynamics modeling, and participatory simulation. We conclude this chapter with a discussion of future challenges for ABM.

In these last three chapters we discuss how to analyze ABMs, how to verify and validate the results of ABMs, and how to use advanced techniques in ABM. These are the important practices of ABM after creating your model and for examining models authored by others.

Acknowledgments

Many people have contributed greatly to this book. Wilensky would like to first and foremost thank Seymour Papert, his dissertation advisor and colleague, who, in many ways, invented the idea of an "agent," who first created a Logo turtle, and who was inspirational regarding the power of computation to transform science and learning. Wilensky is also deeply grateful to Seth Tisue, lead developer of NetLogo for over a decade. Seth's dedication to high quality and elegant code, and his passion and dedication to parsimony, have made NetLogo a much better product. Isaac Asimov, a neighbor of Wilensky's in childhood, was inspirational in describing psychohistory and in catalyzing early notions of emergence. Walter Stroup was an invaluable colleague and joint developer of the HubNet participatory simulation module. Stroup and Corey Brady were influential in advocating for the importance of combining ABM with networked participation.

Rand would like to acknowledge his dissertation chairs, Rick Riolo and John Holland, who encouraged him to pursue his interests in agent-based modeling and helped him to develop many of his formative thoughts on complex systems and ABM. Rand would also like to thank Scott Page, who was not only a member of his dissertation committee but also a continual support to his work in ABM, not the least of which was providing him with a chance to try out the textbook with a sophomore level class at the University of Michigan. In addition, the graduate workshop hosted by Scott Page and John Miller at the Santa Fe Institute was instrumental in helping Rand discover how to teach ABM. Roland Rust had the critical insight to realize that ABM would become increasingly useful for

business and worked with Rand to found the Center for Complexity in Business. Finally, and most important, Rand would also like to thank his coauthor for his wonderful support and for having the courage to allow a postdoctoral fellow to embark on such an unusual project as writing a textbook.

In addition, we received valuable feedback from a number of internal and external reviewers over the course of the writing of this textbook. Chief among them are the teaching assistants for Wilensky's agent-based modeling class at Northwestern University, in which these materials were tested over several years: Forrest Stonedahl, Josh Unterman, David Weintrop, Aleata Hubbard, Bryan Head, Arthur Hjorth, and Winston Chang carefully reviewed the materials, observed the students using the materials, and gave extensive helpful comments for improving the text, the model code, and the explorations. Many other members of the Northwestern Center for Connected Learning and Computer-Based Modeling also gave very helpful feedback, including Seth Tisue, Corey Brady, Spiro Maroulis, Sharona Levy, and Nicolas Payette. Dor Abrahamson, Paulo Blikstein, Damon Centola, Paul Deeds, Rob Froemke, Ed Hazzard, Eamon Mckenzie, Melanie Mitchell, Michael Novak, Ken Reisman, Eric Russell, Pratim Sengupta, Michael Stieff, Forrest Stonedahl, Stacey Vahey, Aditi Wagh, Michelle Wilkerson-Jerde, and Christine Yang contributed illuminating examples. Forrest Stonedahl, Corey Brady, Bryan Head, David Weintrop, Nicolas Payette, and Arthur Hjorth suggested useful explorations for the chapters that have enriched student experience. Wilensky's students at Tufts University, Ken Reisman, Ed Hazzard, Rob Froemke, Eamon McKenzie, Stacey Vahey, and Damon Centola, shared their abundant enthusiasm, generated interesting applications, and furthered the educational uses. Northwestern's dean of engineering, Julio Ottino, was always supportive and encouraging, and confident in the importance of an ABM textbook.

Many colleagues engaged us in many stimulating conversation that influenced our thinking and writing. Danny Hillis provided the context for the nascent beginnings of massively parallel simulations at Thinking Machines Corporation. Luis Amaral, Aaron Brandes, Dirk Brockman, Joanna Bryson, Dan Dennett, Gary Drescher, Michael Eisenberg, Rob Goldstone, Ken Kahn, John Miller, Marvin Minsky, Josh Mitteldorf, Richard Noss, Scott Page, Rick Riolo, Roland Rust, Anamaria Berea, and Bruce Sherin are colleagues with whom we have had ongoing stimulating conversations that helped further the ideas in this book. Mitchel Resnick was an important colleague and collaborator in developing the idea of ABM for education, antecedent software development, and the role of levels and emergence.

The development of NetLogo was supported by more than fifteen years of funding from the National Science Foundation. Additional support came from the Spencer Foundation, Texas Instruments, the Brady Fund, the Murphy Society, the Johns Hopkins Center for Advanced Modeling in the Social, Behavioral and Health Sciences, and the Northwestern Institute on Complex Systems. We are especially grateful to NSF program officers Nora Sabelli and Janet Kolodner for their years of advice and support. We are greatly indebted

to the NetLogo development team at the CCL led by Seth Tisue, who worked meticulously to guarantee the quality of the NetLogo software. Spiro Maroulis and Nicolas Payette contributed greatly to expanding NetLogo's network capabilities. Ben Shargel and Seth Tisue led the design of BehaviorSpace, and James Newell led design of NetLogo 3D, assisted by Esther Verreau and Seth Tisue. Paulo Blikstein, Corey Brady, and Bob Tinker were influential in advocating for combining ABM with physical computing devices. Many members of the CCL made significant contributions to the NetLogo models library.

Many of the materials in this textbook had previously been used in Wilensky's agent-based modeling class in the Computer Science Department at Northwestern University. Early versions of the textbook were used as the basis for a summer workshop taught by Rand as part of the Summer CommuniCy (under the leadership of Klaus Liepelt) at Mittweida University in Germany, and in various classes at the University of Michigan and University of Maryland. We have also piloted these materials in hundreds of NetLogo workshops in the United States and abroad. We would like to thank the students who have been involved in these classes over the years for their feedback and support.

We are indebted to several undergraduate students for proofreading the manuscript, including Ziwe Fumodoh, Nickolas Kaplan, Claire Maby, Elisa Sutherland, Cristina Polenica, and Kendall Speer. For the second printing, we are grateful to Nicolas Payette and Ken Kahn for finding and fixing mistakes in the first printing. We thank them for their many corrections. Of course, all mistakes that remain in the manuscript are our responsibility.

We are deeply grateful to our spouses, Donna Woods and Margaret Rand, for their patience and support during the many days and nights that we were busy writing and were away from family. Wilensky also thanks his children, Daniel and Ethan, for their patience when their dad was too busy writing to play. Rand also thanks his children, Beatrice and Eleanor, who were born during the writing of this book.

We would like to acknowledge the support of the Northwestern Institute on Complex Systems (NICO), which encouraged this project and supported William Rand during the early writing of this textbook. The University of Maryland Robert H. Smith School of Business Center for Complexity in Business supported William Rand during the second half of the writing of this textbook.

0 Why Agent-Based Modeling?

Some look at things that are, and ask why? I dream of things that never were and ask why not?

— John F. Kennedy (paraphrasing George Bernard Shaw)

I think the next century will be the century of complexity.

—Stephen Hawking

We shape our tools and then our tools shape us.

—Marshall McLuhan

This book is an introduction to the methodology of agent-based modeling (ABM) and how it can help us more deeply understand the natural and social worlds and engineer solutions to societal problems. Before we discuss why agent-based modeling is important, we briefly describe what agent-based modeling is. An *agent* is an autonomous computational individual or object with particular properties and actions. *Agent-based modeling* is a form of computational modeling whereby a phenomenon is modeled in terms of agents and their interactions. We will describe ABM more comprehensively in chapter 1. As you will see in this textbook, ABM is a methodology that can be promiscuously applied—there are few, if any, content areas where ABM is not applicable. It can enable us to explore, make sense of, and analyze phenomena and scenarios across a wide variety of contexts and content domains. In the past two decades, scientists have increasingly used agent-based modeling methods to conduct their research.

The main argument of this introductory chapter is that ABM is a transformative representational technology that enables us to better understand familiar topics, and at younger ages; make sense of and analyze hitherto unexplored topics; and enable a democratization of access to computational tools for making sense of complexity and change. As such, we believe that developing ABM literacy is a powerful professional and life skill for students, and we should strive for universal ABM literacy for all, from young students to professionals.

in this simple case, things fundamentally change. The algorithms that are taught after this transformation are different. People's mental representation will alter, as will their sense of systematicity in the field. Psychologically important landmark values (i.e., V vs. 0) will be different. Even social embedding, such as "who can do what," changes (e.g., scribes or special human calculators for the emperor vs. modern carpenters or business people doing their own calculations). In our terminology, we will say that we have a new structuration of a discipline (Wilensky & Papert, 2010; Wilensky et al., 2005). We will proceed to flesh out this term through concrete examples. But, for now, we introduce a preliminary formal definition: By *structuration* we mean the encoding of the knowledge in a domain as a function of the representational infrastructure used to express the knowledge. A change from one structuration of a domain to another resulting from such a change in representational infrastructure we call a *restructuration*.

There have been many examples of restructurations in human history. Of course, our thought experiment is based on a historical reality. Before the transition from the use of Roman to Hindu Arabic numerals in Europe around the turn of the first millennium, most Europeans were able to use Roman numerals fluently. However, because Roman numerals were not very well suited to large numbers and to multiplication and division of such numbers, people went to special "experts" to perform multiplication and division for them. European mathematicians first started employing Hindu-Arabic numerals at the end of the tenth century, quickly realizing its advantages in working with large numbers. In 1202, the mathematician Fibonacci wrote a text outlining the Hindu-Arabic system that resulted in gradual adoption by scientists. Still, "universal" adoption of Hindu-Arabic numerals in Europe was not achieved until the sixteenth century—a restructuration that took more than half a millennium! Why did it take so long for a representational infrastructure quickly recognized as superior to gain widespread adoption? The case of Italian shopkeepers may help explain this quandary. Medieval Italian shopkeepers kept two sets of books for their accounting: one set, in which they did their real calculations, was kept in Hindu-Arabic; the other set, which was presented to the inspecting authorities, was kept in Roman, since a Roman representation was required by the government. The shopkeepers had to laboriously translate the first set into the second. That they deemed such translation worthwhile is a testimony to the value of the restructuration. The fact that the authorities did not officially recognize the Hindu-Arabic books was a major obstacle to the structuration's more rapid spread. We call this resistance to the spread of structurations "structurational inertia" (Wilensky & Papert, 2010). Just as an object's inertia keeps it from changing its motion, so structurational inertia keeps structurations from changing, impeding the spread of restructuration.

Our Roman-to-Arabic numeracy example is just one of many that we could have chosen. In his book *Changing Minds* (2001), DiSessa describes the historical restructuration of simple kinematics from a text-based to an algebraic representation. He illustrates this restructuration through a story of the seventeenth-century scientist Galileo. In his book

Dialogues Concerning Two New Sciences (1638), Galileo struggles to handle a problem involving the relationship between distance, time and velocity. He laboriously describes four theorems relating these three quantities. The reader is invited to peruse and decipher these theorems. The surprising realization is that all four of these theorems are in fact variations of the single equation $D=VT$, or distance equals velocity times time. How could it be that Galileo, inventor of the telescope, and one of the great "fathers of modern science," struggled so mightily with an equation with which most middle-schoolers are facile? The explanation is both simple and profound: Galileo did not have algebraic representation. He had to write these theorems in Italian, and natural language is not a well-suited medium for conveying these kinds of mathematical relationships. Thus, the restructuration of kinematics from the representational system of natural language to that of algebra transformed what was a complex and difficult idea for as powerful an intellect as Galileo's into a form that is within the intellectual grasp of every competent secondary student.

The development of Arabic numerals and the transformation of kinematics via algebra were empowering and democratizing, enabling significant progress in science and widening the range of people who could make sense of previously formidable topics and skills. The vista opened to the imagination is dramatic: If the problems with which we struggle today could be so transformed, think of the new domains we could enter and conquer. If algebra could make accessible to students what was hard for Galileo, what domains that are hard for us today to understand could we restructurate to make more accessible?

Complex Systems and Emergence

What might be the analogy today? What areas are widely thought to be difficult for people to comprehend and potentially ripe for restructuration? One such area is complex systems. Its very name suggests that it is a difficult area for comprehension.

What we perceive as difficult has cognitive dimensions, but difficulty is also greatly affected by our current needs. As commerce developed in the Middle Ages, there arose an increasing need for arithmetic with large numbers, so the difficulty of doing it with Roman numerals became more salient. As science developed the need to account more precisely for heavenly bodies, the difficulty of describing their motions became more transparent. In the current day, the world we live in has become increasingly complex, in part because, in earlier periods of history, we did not have to pay as much attention to complex interactions; we could get by with understanding simple systems and local effects. Yet, as technology and science have advanced, we have become more affected by complex interactions. We are now aware that changes to the rain forest in Brazil can have dramatic effects on the climate of faraway countries; that unwise financial decisions in one country can have significant economic impact on the rest of the world; that a single case of a new disease in China can spread around the globe in short order; or that a four-minute video uploaded

by a Korean pop star can turn him into a worldwide sensation in a matter of days. As such, the difficulty of making sense of complex systems has become more salient.

However, even if the level of complexity in our life remained constant over the ages, our continual quest for knowledge would ultimately lead us to study complex systems. As we gain facility and more complete understanding of simple systems, we naturally progress to trying to make sense of increasingly complex systems. Simple population dynamics models, for example, make the implicit assumption that all members of a species are the same, but later, it becomes important to explore the manifold complexity of the food web and how each individual interacts with every other individual. Thus, our need to understand more complex systems is also a natural result of the growth of human knowledge.

As we gain knowledge, we create more sophisticated tools and these tools enable us to ask and answer new questions. As described earlier, the advent of powerful computation enables us to model, simulate, and more deeply probe complex systems.

For the reasons stated, the field of complex systems has arisen and grown. Complex systems theory develops principles and tools for making sense of the world's complexity and defines complex systems as systems that are composed of multiple individual elements that interact with each other yet whose aggregate properties or behavior is not predictable from the elements themselves. Through the interaction of the multiple distributed elements an "emergent phenomenon" arises. The phenomenon of *emergence* is characteristic of complex systems. The term "emergent" was coined by the British philosopher and psychologist G. H. Lewes, who wrote:

> Every resultant is either a sum or a difference of the co-operant forces; their sum, when their directions are the same—their difference, when their directions are contrary. Further, every resultant is clearly traceable in its components, because these are homogeneous and commensurable. It is otherwise with emergents, when, instead of adding measurable motion to measurable motion, or things of one kind to other individuals of their kind, there is a co-operation of things of unlike kinds. The emergent is unlike its components insofar as these are incommensurable, and it cannot be reduced to their sum or their difference. (Lewes 1875)

Since Lewes's time, scholars have struggled with how to best define emergence—some definitions succinct, others more involved. For our purposes, we define emergence as *the arising of novel and coherent structures, patterns, and properties through the interactions of multiple distributed elements*. Emergent structures cannot be deduced solely from the properties of the elements, but rather, also arise from interactions of the elements. Such emergent structures are system properties yet they often feedback to the very individual elements of which they are composed.

Important features of emergence include the global pattern's spontaneous arising from the interaction of elements, and the absence of an orchestrator or centralized coordinator—the system "self-organizes." Structure (or rules) at the micro-level leads to ordered pattern at the macro-level. Because the macrostructures are emergent, composed of many

elements, they are dynamic, and perturbing them often results in them dynamically reforming. Another way of thinking about such structures is to view them not as entities, but rather, as processes holding the structure in place, which are often invisible until the structure is disturbed. However, a reformed structure, while recognizably the same structure, will not be identical, since for most emergent structures, randomness plays a role in each reformation. From a micro-level perspective, this suggests that the formation rules need not be deterministic. Indeed, in many complex systems, probabilistic and random processes contribute to, and are even essential to, the creation of order.

In complex systems, order can emerge without any design or designer. The idea of order without design has been controversial throughout the history of science and religion. In modern times, the supposed impossibility of order without design has been a linchpin of the intelligent design movement against naturalistic evolution, as supporters argue that life's manifold and irreducible complexity could not arise "by chance" without a designer. Yet, complex systems theory is ever finding more complex systems that may at first seem irreducible but are found to be self-organized or evolved rather than intelligently designed by a designer.

Understanding Complex Systems and Emergence

We have said that understanding complex systems and emergence is hard for people. Emergence, in particular, presents two fundamental and distinct challenges. The first difficulty lies in trying to figure out the aggregate pattern when one knows how individual elements behave. We sometimes call this *integrative* understanding, as it parallels the cumulative integration of small differences in calculus. A second difficulty arises when the aggregate pattern is known and one is trying to find the behavior of the elements that could generate the pattern. We sometimes call this *differential* understanding (aka *compositional* understanding), as it parallels the search in calculus for the small elements that produce an aggregate graph when accumulated. Let's now consider two examples that illustrate these concepts.

Example 1: Integrative Understanding

Figure 0.1 presents a system composed of a few identical elements following one rule. Each element is a small arrow. We imagine a clock ticking and at each tick of the clock the arrows follow their rule. We initialize the system so that each individual arrow starts on a circle (of radius 20 units). We start them all facing clockwise on the circle. Now, we give them one movement behavior (or rule). At every tick of the clock, they move forward 0.35 units then turn right one degree. As the clock ticks, they continue to move and turn, move and turn, moving clockwise along the circle.

Now suppose that we slightly alter these rules. Instead of moving forward 0.35 units, we have them move 0.5 units while still turning one degree. What will be the aggregate

Some arrows moving clockwise around a circle of radius 20.

pattern that we see? Before reading further, take a moment to imagine what the pattern will be.

Most people do not predict the resulting pattern. We have heard people predict that the arrows will move onto a larger circle, a smaller circle, a flower shape, and many others. In fact, the pattern that emerges is a pulsating circle. All the arrows stay in a circle, but the circle changes its radius, first expanding, then contracting and repeating this cycle forever.

Example 2: Differential Understanding

Now let's consider the flip side of these difficulties. There are many coherent, powerful, and beautiful patterns we observe in the world. What accounts for their prevalence? How do they originate?

The secret to understanding the formation of these patterns is to understand that they are emergent, arising from the interactions of distributed individual elements.

One such prevalent (and often beautiful) pattern is the flocking of birds. Birds fly together in many different formations, from the classic V formation of goose flocks to the

flocks of starlings (thousands of birds) acting as a swarm."

2. For beautiful video of massive starling swarms, called murmurations, see: http://www.huffingtonpost.com/2013/02/01/starling-murmuration-bird-ballet-video_n_2593001.html, http://www.youtube.com/watch?v=PnywhC36UVY, http://www.youtube.com/watch?v=XH-groCeKbE, or http://www.youtube.com/watch?v=iRNqhi2ka9k.

Traffic Jam by Osvaldo Gago, 2005.

by a pecking order; the jam occurs at a specific place because the accident or radar trap was there, rather than at random locations along the road.

To be sure, accidents and radar traps cause some traffic jams. Most, however, arise from the random entry of cars into, say, a highway and the resultant statistical distribution of cars and speeds.[3] Similarly, science has established that bird flocks are not centrally organized; rather than the same bird staying at the apex of the V formation, different birds occupy that spot. The composition of the formation, hence, is not deterministic, but rather, emergent from birds' independent movements as they head in a particular direction, trying to avoid other birds and yet not get too far from their neighbor birds.[4] We will further explore models of flocking in chapter 7.

In further analyses of the interviews, Wilensky and Resnick (1999) identified a key component of the DC mindset and an obstacle to thinking about emergent phenomena: the problem of "thinking in levels." Emergent phenomena can be described as existing on at least two levels: the level of the individual elements (cars, birds, people, etc.); and the level of system or aggregate patterns (flocks, traffic jams, housing patterns, etc.). Most

3. For a video of a fascinating experiment to create traffic jams, see http://www.newscientist.com/article/dn13402-shockwave-traffic-jam-recreated-for-first-time.html.

4. For a NetLogo model of birds flocking, see *Wilensky (1998)*.

people fail to distinguish between these levels, instead "slipping" between levels to attribute the properties of one level to the other. Consider a V- flock, which appears to be stable and to have a consistent shape. The *appearance* of stability often leads people to conclude that the individual elements of the flock (the birds) *are* stable and have a consistent place in the flock. As we have seen, however, this is a misunderstanding based on a slippage between levels, and is an example of a failure in differential understanding. In this example, the shape of the flock is salient; the birds' behavior is less so. The natural direction of levels slippage is from aggregate to individual. We are seduced into transferring a property of the aggregate to the individual elements. With the case of traffic, we are much more familiar with the ways individual cars move then we are of aggregate traffic patterns. When we typically think about traffic, we are seated inside a car, very aware of its movements and how it responds to the movements of other cars. When we encounter a jam, we are likely to think of it as behaving like a car; we imagine it as responding to the stopping of a car in an accident and moving forward like a car, rather than moving backward as jams actually do. Here, the direction of levels slippage is opposite to that of bird flocks: the properties of the individual elements, the cars are transferred *to* the aggregate pattern, the jam. This is an example of a failure of integrative understanding.

Wilensky and Resnick also showed a host of examples where this levels slippage interfered with both integrative and differential understanding of complex phenomena in the natural and human social worlds. Furthermore, this mindset is not just a problem of the scientifically naïve. Trained scientists also fall prey to the DC mindset.[5] Wilensky and Resnick presented a host of examples across an array of content and contexts (e.g., economic markets, predator-prey relations, slime-mold behavior, human housing patterns, growth of crystals, insect foraging) where levels slippage interfered with understanding. Many of these examples (and a host of new ones) will appear in this book. Indeed, in the past two decades, researchers have found that emergent phenomena are endemic to the natural and social worlds and that using an emergent lens to make sense of complex patterns is a vital need in a twenty-first-century world.

Agent-Based Modeling as Representational Infrastructure for Restructurations

Returning to Roman-to-Arabic numerical restructuration analogy, we suggest that new computer-based representations can help restructurate our knowledge in many domains. With the aid of new computer-based modeling environments, we can simulate complex patterns and better understand how they arise in nature and society. Whereas in many areas

5. Keller and Segal (1985) described the scientific study of slime molds and how it was shaped by the DC Mindset. At certain stages of their life cycle, slime molds gather into clusters. Early in the study of slime molds it was assumed that there was a "founder" or "pacemaker" that controlled the aggregation process, but later it was discovered that there was no need for a specialized coordinator. Yet the centralized view was embraced and vehemently defended for more than a decade, despite strong evidence to the contrary.

of science we have relied on simplified descriptions of complexity—often using advanced mathematical techniques that are tractable and allow us to calculate answers—we can now use computation to simulate thousands of individual system elements, called "agents." This allows new, more accessible and flexible ways to study complex phenomena—we simulate to understand.

Agent-based modeling is a computational methodology that enables one to model complex systems. As the name suggests, agent-based models are composed of *agents:* computational entities that have properties, or *state variables and values* (e.g., position, velocity, age, wealth, etc.). Agents usually also have a graphical component so you can see them on the computer screen. An agent can represent any element of a system. A gas molecule agent, for instance, might have properties such as "mass" with value 30 atomic mass units, "speed" with value 10 meters per second, and "heading" with a value of the angle it is facing. A sheep agent, by contrast, might have properties such as "speed" with value 3 mph, weight with value 30 lbs., and fleece with a value of "full" (a discrete-textual rather than numerical value). In addition to their properties, agents also have rules of behavior. A gas molecule agent might have a rule to collide with another molecule; a sheep agent might have a rule to eat grass if there is grass available nearby. In an agent-based model, we imagine a universal clock. When the clock ticks, all agents invoke their rules. If the conditions of the rules are satisfied, (e.g., they are at the edge of a box, or grass is nearby), they enact the behavior (i.e., bounce or eat grass). The goal of agent-based modeling is to create agents and rules that will generate a target behavior. Sometimes the rules are not well known, or you just want to explore the system's behavior. In that case, ABM can be used to help you better understand a phenomenon through experimentation with rules and properties.

A working hypothesis of representational theorists is that anything that is perceived as difficult to understand can be made more understandable by a suitable representation. We contend that ABM's enable restructurations of complex systems so that the (a) understanding of complex systems can be democratized and (b) the science of complex systems can be advanced. This hypothesis begets a design challenge: Can we design a suitable representational language that supports both parts of the claim, enabling scientists to author scientific models in this language while simultaneously enabling a wider audience to gain access to (and understand) complex systems?

The computer language used in this text, NetLogo (*Wilensky, 1999*), was developed by Uri Wilensky for these express purposes.[6] It is a general-purpose agent-based modeling language designed to be "low-threshold"—that is, novices can quickly employ it to do meaningful and useful things—but also "high-ceiling"—meaning that scientists and researchers can use it to design cutting-edge scientific models. The language borrows much of its syntax from the Logo language, which was designed to be accessible to children.

6. NetLogo is freely available from ccl.northwestern.edu/netlogo.

Like Logo, NetLogo calls its prototypical agent a "turtle." However, while in Logo, the user directs the turtle to draw geometric figures, in NetLogo, this is generalized to thousands of turtles. Instead of drawing with pens, they typically draw with their bodies, moving according to rules, and the configuration of their bodies presents a visualization of the modeled phenomenon. NetLogo was first developed in the late 1990s, and it is now in use by hundreds of thousands of users worldwide. Thousands of scientific papers have utilized NetLogo to construct and explore models in a wide variety of disciplines. It has also been employed by policymakers to model policy choices, business practitioners to model business decisions, and students to model subject matter in their coursework across virtually the entire curriculum. Many NetLogo-based courses have sprung up in both universities and in secondary schools.

As of yet, no textbook has been written that gives a general and systematic introduction to NetLogo in all of its features and shows how to use it to model phenomena across many different domains. It is our hope that this textbook will serve to enhance and further democratize ABM literacy. We envision it being used as a primary textbook in an agent-based modeling course, but it can also serve as a supplementary textbook in virtually any university course whose subject matter is amenable to agent-based modeling.

We further maintain that virtually every university subject can benefit from a basic familiarity with agent-based modeling. Some subject domains have embraced agent-based modeling from the start, such as chemistry, biology, and materials science. Others embraced it in a second wave, such as psychology, sociology, physics, business, and medicine. Recently, we have seen the growth of agent-based modeling in economics, anthropology, philosophy, history, and law. While different fields have different degrees of structurational inertia, there is no end to the domains of application for ABM. However, differences in structurational inertia render some fields more easily adaptable to ABM restructurations than others. To illustrate the potential power of widespread agent-based modeling literacy and restructuration, we will look briefly at two examples derived from different content domains: predator-prey interactions and the spread of forest fires.

Example: Predator-Prey Interactions

Let us start with the study of predator-prey interactions. This domain is often first introduced qualitatively in high school, then quantitatively at the university level. In its quantitative form, the population dynamics of a single predator and prey are introduced by the classic Lotka-Volterra differential equations, a pair of coupled differential equations that proceed as follows:

$$\frac{dPred}{dt} = K_1 * Pred * Prey - M * Pred$$

$$\frac{dPrey}{dt} = B * Prey - K_2 * Pred * Prey$$

is the more valid. But the equational model is not more valid; it, too, is a model, and a highly simplified one at that. In fact, we now know from the works of biologists such as Gause (1936) that the equational model is less accurate than an ABM in the isolated predator-prey situation for which it was intended. In particular, the equational model underrepresents extinctions, since the model uses real numbers to represent the population densities. This means that the prey population, for example, can dip to 0.5, or 0.1, or 0.01 and yet still come back. In the real world, however, populations are discrete. When the model goes below one prey (or a pair), it reaches a functional point of no return.

Example: Forest Fires

Our second example is about the spread of a forest fire. This domain is not usually present in the K-12 or university curriculum, but when taught, it typically falls under the subject matter of physics, described in terms of two classic partial differential equations. The first is the classic heat equation, which describes the distribution of heat in a given region over time, where theta represents the thermal diffusivity of the material through which the heat is traveling.

$$\frac{dH(x,t)}{dt} = \theta \frac{d^2H(x,t)}{dx^2}$$

The second equation physicists use to describe the spread of a forest fire treats the fire as if it were a potentially turbulent fluid, thus using the Reynolds equation of fluid flow.

$$\frac{dU_i}{dt} + U_j \frac{dU_i}{dx_j} = -\frac{1}{\rho}\frac{dP}{dx_i} + v\frac{d^2U_i}{dx_j dx_j} - \frac{d}{dx_j}\overline{u_i' u_j'}$$

Needless to say, these equational representations are well beyond students in the K–12 years and, we would guess, the vast majority of undergraduate science majors. Understanding what they mean and how to compute them requires significant knowledge of higher-level physics as well as the machinery of partial differential equations.

Contrast this with the ABM approach to modeling forest fires (illustrated in figure 0.7), which would typically model the environment as a grid of cells with trees occupying certain cells. Modeling the spread of fire consists simply of giving rules to the cells that are on fire as to when to spread to neighboring tree cells. This representation is so simple, we have seen elementary school students comprehend and explore it. They can experiment to see how different densities of trees in the forest affect the fire spread and they can modify the basic model to ascertain the effects of wind, or wood type, or fire source. We will explore an ABM of a forest fire in chapter 3 and consider such extensions. Of course, a very simple ABM of forest fire spread would not correctly model a particular fire, but it does give insight into the dynamics of any fire and once we know the details of a particular fire, we can add in whatever data or rules that apply to the situation. This enables

An agent-based model of a forest fire. (See *Wilensky, 1997b*)

even scientists to experiment much more fluidly with different models of spread, iteratively refining their models. ABM methods are starting to be used to model and fight real forest fires (see, e.g., www.simtable.com for a company that does agent-based modeling of emergency management including wildfires).

The restructuration of these systems using ABM provides several representational benefits. They make use of discrete rather than continuous representations, which are more easily comprehensible, more closely match real-world situations and require much less formal mathematics to employ. They are easier to explore and much easier to modify. They present immediate feedback with visualizations that allow researchers and practitioners to understand and critique them at two levels, the level of the overall aggregate pattern, such as the fire spread or predator population levels, but also the level of the behavior of the individual animals, and the fire spread to particular trees. Though these two examples highlight some of the advantages of agent-based restructurations, the full potential of ABM restructuration is not yet evident either in these examples, or in science as a whole.

The two examples we have given here come from the natural sciences. We believe the potential of ABM restructurations may be even more important in the social sciences. This is because the core representational infrastructure in the social sciences consists of words

and texts. Words and texts do not as easily specify the precision of an idea and can thus be interpreted in fundamentally different ways by different people. Moreover, words and texts are not dynamic representations, so they cannot give you immediate feedback as to the consequences of the assumptions embedded in them. By capturing social science theories in dynamic ABM representations, we make their assumptions explicit, and they become demonstrations of the consequences of their assumptions. If someone wants to disagree with your model, he or she must show how either an assumption is incorrect or missing or show how the logic of the interactions is flawed. The model serves as an object-to-think-with and a test bed for alternate assumptions. This can be particularly powerful when it comes to issues of policy where one can rapidly test many different alternative potential scenarios and examine their consequences. As such, ABMs serve as powerful complements to text-based explanations.

Over the past twenty years, the authors of this textbook have been working on improving the infrastructure, NetLogo, and also on restructurating domains. We have been involved with agent-based restructurations at all levels of schooling, in a wide variety of domains including most of the natural and social sciences and engineering. Restructurations have been performed in a range of fields so diverse as to include cognitive and social psychology, linguistics, biology, chemistry, physics, and many more. Agent-based models are now used in the professions to do research in medicine and law and by policymakers to help them explore effects of alternative policies.

There is still much work to do to establish the representational infrastructure and the science of ABM. What is needed is widespread literacy in agent-based modeling. We are hopeful that this textbook will move us forward and enable a large number of students to learn about and master this new representational infrastructure. We envision a series of textbooks that use agent-based modeling to restructurate many specific subjects. It is our hope that this textbook will help to spread literacy in agent-based modeling, to catalyze these restructurations, and that the widespread use of agent-based representations will take considerably less than five hundred years.

1 What Is Agent-Based Modeling?

Go to the ant, O sluggard; consider her ways, and be wise.

—Proverbs 6:6

Ants

The ant opens her eyes and looks around. There are many of her siblings nearby, but there is no food. The ant is hungry, so she heads out from the ant colony and starts to wander around. She sniffs a little to the left and a little to the right, and still she cannot smell any food. So she keeps wandering. She passes by several of her sisters, but they do not interest her right now; she has food on her mind. She keeps wandering. Sniff! Sniff! Mmmm … good! She gets a whiff of some of that delightful pheromone stuff. She heads in the direction of the strongest pheromone scent; in the past there has been food at the end of that trail. Sure enough, as the ant proceeds along the trail, she arrives at some delicious food. She grabs some food and heads back to the colony, making sure to drop some pheromone along the way. On the way back, she runs into many of her sisters, each sniffing her way along pheromone trails, repeating the same process they will carry out all day.

What we have just fancifully described is an ant foraging for food. The opening paragraph of this section is in itself a model of ant behavior. By a *model*, we mean an abstracted description of a process, object, or event. Models can take many distinct forms. However, certain forms of models are easier to manipulate than other forms. The textual description in our first paragraph cannot easily be manipulated to answer questions about the ant colony's behavior—for example, what would we see if all the ants in the colony followed the behavior we described? It is difficult to extrapolate from the textual description of one ant to a description of an ant colony. The textual model is not sufficiently generative to answer such questions. It is a fixed description—always "behaving" the same, thus not bestowing any insight into the range of variation in behaviors. And it is not combinatorial—we cannot use the description to understand the interaction of the ants with each other or with the environment.

One way to make the preceding model more generalizable and gain insight into the behavior of an ant colony would be to implement the model in a computational form. A *computational model* is a model that takes certain input values, manipulates those inputs in an algorithmic way, and generates outputs. In computational form, it would be easy to run the "Ants" model with large numbers of ants and to observe the model's outputs given many possible different inputs. We use the term *model implementation* to refer to this process of transforming a textual model[1] into a working computational simulation (written in some form of computer "code"). Besides textual models, there are other forms of *conceptual models* that describe processes, objects, or events but are not computational; conceptual models can also be diagrammatic or pictorial.

This description lends itself particularly well to being implemented, since it results from a particular standpoint, that of an individual ant, or ant *agent*. By the word *agent*, we mean an autonomous individual element of a computer simulation. These individual elements have properties, states, and behaviors.

The ant is an agent. It has properties such as its appearance and its rate of movement. It has characteristic behaviors such as moving, sniffing, picking up food, and dropping pheromone. It has states such as whether or not it is carrying food (a binary state) and whether it can sense how much pheromone is in the environment around it (a multi-valued state).

Agent-based modeling (ABM) is a computational modeling paradigm that enables us to describe how any agent will behave.[2] The methodology of ABM encodes the behavior of individual agents in simple rules so that we can observe the results of these agents' interactions. This technique can be used to model and describe a wide variety of processes, phenomena, and situations, but it is most useful when describing these phenomena as *complex systems*. Our aim in this textbook is to facilitate your creating, modifying, and analyzing the outputs of agent-based models. We begin by describing the genesis of the ant foraging conceptual model and how it was implemented as an agent-based model.

Creating the Ant Foraging Model

Many biologists and entomologists have observed ants in the wild (Hölldobler & Wilson, 1998; Wilson, 1974) and have described how ants seemed to form trails to and from food sources and their nest. In the next few paragraphs we will describe some hypotheses about how the ants accomplish this behavior and what mechanisms are at work that enable ants to find food in this way.[3] Perhaps the trails form as follows: After an ant finds food, it goes

1. Or, more generally, a conceptual model in any form.

2. We will use the acronym ABM in several ways in this textbook. Sometimes we will use it to refer to the practice of agent-based modeling. Other times, we will use an ABM to refer to an agent-based model, or ABMs to refer to agent-based models. We rely on context to disambiguate our use of the term.

3. Our account of ant behavior is inspired by the actual history of the scientific study of ants. It has been simplified here for the sake of exposition. For a more detailed account, see Theraulaz and Bonabeau (1999).

Now that we have a basic hypothesis (in the form of a textual model) that describes the behavior of the ant, how do we implement the model so that we can test out this hypothesis and see if our computational model adequately accounts for what we observe in nature?[6] The first step is to create a more algorithmic description of the preceding textual model. This is just another model itself, but one that is more easily translated into an implemented model. Here is one set of rules that an individual ant could follow in order to operate in accordance with the model described earlier. We describe the rules from the point of view of an individual ant.

1. If I am not carrying food, I check if there is food where I am; if there is, I pick it up; if there isn't food right here, I try to sense a pheromone trail nearby; if I find one, then I face "uphill" along the pheromone gradient toward its strongest source.
2. If I am carrying food, I turn back toward the nest and drop pheromone on the ground below me.
3. I turn randomly a small amount and move forward a step.

These rules are easily implemented in a computer language. There are many computer languages in use for many purposes, but only a few are especially tailored to work with agent-based modeling. One of these is NetLogo (Wilensky, 1999a). NetLogo is both a modeling language and an integrated environment designed to make agent-based models easy to build. In fact, NetLogo is so easy to use that, rather than describe algorithms and models in pseudo-code,[7] throughout this textbook we will use NetLogo instead. Describing the preceding rules in the NetLogo language is straightforward. Here, for example, is one translation of rules 1–3 into NetLogo code:

```
if not carrying-food? [ look-for-food ] ;; if not carrying food, look for it
if carrying-food? [ move-towards-nest ] ;; if carrying food turn back towards the nest
wander                                  ;; turn a small random amount and move forward
```

This code snippet is not the complete implemented model but it does describe the core components that go into the Ants model. To complete the model we need to describe each of the subcomponents (such as `move-towards-nest`, and `look-for-food`, and `wander` and `look-for-food` will need to describe the ant's sniffing for pheromone). Each of which is a small amount of code. The result will be a fully implemented computational model. (See figure 1.2.)

Let us quickly try to "read" the code snippet and understand what it is doing. The `if` primitive takes as input a predicate (i.e., a statement that is either true or false) and takes

6. The process of comparing data from a computational model to real-world data is called *validation*. It will be discussed in depth in chapter 7.

7. Pseudo-code is an intermediate form between text and computer code that is often used to describe computational algorithms.

NetLogo "Ants" model of foraging behavior.

one action if the predicate is true and another if it is false. The first "if" in the preceding code asks if the ant is carrying food. If she is not carrying food, then she takes the first action, that is, to look for food. She does this by checking to see if there is food where she is standing, and if there is not she looks around to see if there is a pheromone trail nearby that she can follow to find food (this is all described in the `look-for-food` procedure). If she is carrying food, then she takes the second action, which is to turn back toward the nest. To head back to the nest, she first determines if she is at the nest, if she is, then she drops her food, otherwise she turns so as to follow the trail back to the nest, dropping some pheromone along the way, since she just came from a food source. Regardless of whether she is looking for food or returning to her nest, the ant wanders a little bit along the direction she is heading. This is to simulate that ants, as with almost all animals, do not usually follow a straight line, but instead take small steps in other directions along the way.

By the end of chapter 3, you will be able to make substantive modifications to this model, and by chapter 4, you will be able to construct such a model on your own.

8. http://ccl.northwestern.edu/netlogo/models/Ants (*Wilensky, 1997*).

Box 1.2
Exploring the Ant Foraging Model

To explore this model in more depth, open the Ants model (Wilensky, 1997) (found in the Biology section of the NetLogo models library). Once you open this model, you will see a set of controls that you can manipulate to change the parameters of the model. Try varying some of the parameters (e.g., POPULATION, DIFFUSION-RATE, EVAPORATION-RATE), and explore the accompanying material that discusses this model and the real-world experiments it is based upon. As you manipulate the model, consider the following questions:

1. How does the evaporation rate affect the ability of the ants to form trails to the food? What happens if there is no evaporation?
2. How does the rate of diffusion affect the kind of trails the ants form?
3. How does the number of ants affect the colony's ability to consume the food?

Results and Observations from the Ant Model

When running the Ants model, the results are initially surprising for someone who has only seen the rules for the individual ants. The "aggregate" or "macro" behavior of the model shows apparently systematic food gathering behavior. It is as if the ant colony has a clear plan for how to gather the food. Yet, we have seen that the Ants model rules do not contain any systematic foraging plan. If we look closely at the model running, we observe that initially the ants wander around at random. Then some ants will wander into a nearby food source. Once they find that food source, they will start to bring food back to the nest, laying a pheromone trail beneath them. If just one lone ant finds the food source, the pheromone trail will not be strong enough for other ants to follow it; but as more and more ants find the food source, the trail will become stronger and stronger. Eventually, the actions of many ants will create a strong pheromone trail from the nest to the food source, and so any ant can easily find the trail to the food source.

The ants as a group appear to exploit food sources in an optimal manner. That is, they first gather food from the nearest food source, then the second nearest, and so on. This appears to be a conscious plan of the ant colony; but as we know from the ant rules, this is not the case. In fact, as you observe the model closely, you will note that sometimes ants operate almost at cross-purposes with other ants, creating additional pheromone trails to farther food sources and distracting some of the ants that are currently harvesting the closer food source. The ants do not have a centralized controller; instead, the nearest food sources are the most likely to be found first by the ants randomly wandering from the nest. The nearest food sources also require the least amount of pheromone since the pheromone only has to cover the shortest path from the nest to the food. Once a sufficient number of ants have found a particular food source, the pheromone trail to it stabilizes and thus

attracts more ants to it.[9] When the food source has been completely consumed, the ants no longer lay pheromone; hence, the trail dissipates, and the ants are released to search for other food sources.

This optimal exploitation of food sources could be placed within a larger context. In many ways, the colony of ants seems to balance exploration and exploitation (Dubins & Savage, 1976. In any situation in which an entity is operating in an unknown environment, the entity must spend some time exploring the environment to understand how its actions affect its rewards, and some time exploiting the environment, that is taking actions that it knows have produced the best rewards in the past. By allocating a large number of ants to exploit the current nearest food source while other ants continue to explore, the ant colony as a whole successfully balances exploration and exploitation.

However, these "trails" that the ants build to the food source, the "optimal" behavior that they exhibit, and the "balance" between exploration and exploitation are not coded into any one ant. There is nothing that tells the ant to build a trail; there is nothing that tells the ant to go to the nearest food source first; there is nothing that tells some ants to explore while others exploit. The "trails," "optimal," and "balance" behavior of the ants is not coded into any of the ants but is instead an *emergent* phenomenon of the model (Holland, 1998; Anderson, 1972; Wilensky & Resnick, 1999). Having run and used the model, it may seem obvious that these low-level rules can create these rich and optimal global patterns. But the history of science is full of wrong turns in which scientists believed that a complex phenomenon needed a complex organizational structure and a leader (Resnick, 1994; Wilensky & Reisman, 2006; Wilensky & Resnick, 1999). In contrast, through ABM we understand that this complexity can self-organize without a leader.

What Good Is an Ant Model?

The Ant model is our first example of an agent-based model. Now that we have gone through the process of understanding how a model of ant foraging works, what knowledge have we gained from that process and how can we profitably use the model? It may seem at first that the only thing the model does is provide a visualization of one particular textual model. We describe eight main uses for agent-based models: (1) description, (2) explanation, (3) experimentation, (4) providing sources of analogy, (5) communication/education, (6) providing focal objects or centerpieces for scientific dialogue, (7) as thought experiments, and (8) prediction.

A model is *descriptive* of a real-world system. Granted, it is a simplification of the real world and does not contain all of the details and inconsistencies that are present in the real

9. The increase in pheromone attracts an increasing number of ants to the vicinity of the pheromone, which in turn increases the amount of pheromone. This process exhibits positive feedback, which will be discussed further in chapter 7.

Box 1.3

world. But all models are coarse-grained descriptions of reality; and, in fact, models that are not coarse-grained descriptions are useless as descriptions because they are indistinguishable from the real world and therefore do not assist in our understanding of complex systems[10] (Korzybski, 1990). If your model includes all aspects of the real phenomenon, it is more efficient to simply observe reality, since it saves you the time of building the model. The function of a model is to help us to understand and examine phenomena that exist in the real world in more tractable and efficient ways than by simply observing reality. Even if you have never observed a real ant colony, the Ants model helps—it can help you know what to look for and generate hypotheses that you can confirm or disconfirm by observation.

Models are *explanatory* in that they point out the essential mechanisms underlying a phenomenon. They can function as a proof that hypothesized mechanisms are sufficient to account for an observation. Models provide us with a proof of concept that something is possible. For example, once we built the Ants model and observed the results, we proved that an ant colony could exhibit characteristics such as "trails," "optimality," and "balance" without a centralized controller (Resnick & Wilensky, 1993; Resnick, 1994; Wilensky & Resnick, 1999). These characteristics are all emergent outcomes of low-level mechanisms. A key function of ABMs is to explicate the power of the emergence. In general, it is difficult for people to understand how such simple rules can lead to complex observed phenomena, and ABMs make this connection explicit. Even if this were not the way ants actually worked, the model illustrates that this is one mechanism that could be used. We can also compare and contrast alternative hypotheses. We could, for example, build the other food-gathering hypotheses discussed earlier as computational models and compare

10. The Argentine author Borges has a fanciful short story (1946) based on the premise of a map as big as the terrain it maps.

that make them agent-based models? Now that we have described the process of designing one particular model, let us take a step back and describe more formally what agent-based modeling is and how it can be used.

What Is Agent-Based Modeling?

The profitable uses of the Ants model that we described earlier are not particular to that one model. These are generally applicable affordances of the methodology used to develop the model, which is agent-based modeling. The core idea of *Agent-Based Modeling* is that many (if not most) phenomena in the world can be effectively modeled with agents, an environment, and a description of agent-agent and agent-environment interactions. An *agent* is an autonomous individual or object with particular properties, actions, and possibly goals. The environment is the landscape on which agents interact and can be geometric, network-based, or drawn from real data. The interactions that occur between these agents or with the environment can be quite complex. Agents can interact with other agents or with the environment, and not only can the agent's interaction behaviors change in time, but so can the strategies used to decide what action to employ at a particular time. These interactions are constituted by the exchange of information. As a result of these interactions, agents can update their internal state or take additional actions. The goal of this textbook is to explore in detail all of the different aspects and uses of agents and their interactions.

Agent-Based Models vs. Other Modeling Forms

What makes agent-based models distinct from other models? The most common form of scientific models is the equation form. Parunak, Wilensky, and colleagues (Parunak et al., 1998; Wilensky, 1999b; Wilensky & Reisman, 2006) discuss the many differences between ABM and equation-based modeling (EBM). One distinction is that because ABM models individuals it can model a heterogeneous population, whereas equational models typically must make assumptions of homogeneity. In many models, most notably in social science models, heterogeneity plays a key role. Furthermore, when you model individuals, the interactions and results are typically discrete and not continuous. Continuous models do not always map well onto real-world situations. For instance, equation-based models of population dynamics treat populations as if they are continuous quantities when in fact they are populations of discrete individuals. When simulating population dynamics it is very important to know if you have a sustainable population. After all, a wolf population cannot continue if there are fewer than two wolves left; in reality, a millionth of a wolf cannot exist and certainly cannot reproduce, but it can result in increased wolf population in EBMs. The mismatch between the continuous nature of EBMs and the discrete nature of real populations causes this "nano-wolf" problem (Wilson, 1998). As a result, for EBMs

to work correctly, they must make the assumption that the population size is large and that spatial effects are unimportant (Parunak et al., 1998; Wilensky & Reisman, 2006; Wilkerson-Jerde & Wilensky, 2010, in press).

Another advantage of ABM over EBM is that it does not require knowledge of the aggregate phenomena: One does not need to know what global pattern results from the individual behavior. When modeling an outcome variable with EBM, you need to have a good understanding of the aggregate behavior and then test out your hypothesis against the aggregate output. For example, in the wolf-sheep (predator-prey) example, to build the EBM, you need to have an understanding of the relationship between (aggregate) wolf populations and sheep populations. To encode this aggregate knowledge such as in the classic Lotka-Volterra equations (Lotka, 1925; Volterra, 1926), you must have knowledge of differential equations.[11] In contrast, ABM enables you to write simple rules for simple entities, requiring knowledge only of commonsense behaviors of individual wolves and sheep and yet still observe the aggregate result by running the model. Thus, even if you have no hypothesis as to how the aggregate variables will interact, you can still build a model and generate results.

Because agent-based models describe individuals, not aggregates, the relationship between agent-based modeling and the real world is more closely matched. It is therefore much easier to explain what a model is doing to someone who does not have training in the particular modeling paradigm. This is beneficial because it means that no special training is required to understand an agent-based model. It can be understood by all of the stakeholders in a modeling process. Moreover, with some ABM languages like NetLogo, the syntax is so readable that stakeholders without knowledge of how to build a model can often read the model code and understand what is going on. This helps improve the verifiability of the model.[12] This "glass box" approach to modeling (Tisue & Wilensky, 2004) enables all interested parties to talk about the model all the way down to its most basic components.

Finally, the results generated by ABMs are more detailed than those generated by EBMs. ABMs can provide both individual and aggregate level detail at the same time. Since ABMs operate by modeling each individual and their decisions, it is possible to examine the history and life of any one individual in the model, or aggregate individuals and observe the overall results. This "bottom-up" approach of ABMs is often in contrast with the "top-down" approach of many EBMs, which tell you only how the aggregate

11. Differential equations are often represented with another modeling approach, systems dynamics modeling, which provides discrete approximations to the equations. We discuss system dynamics modeling in more depth in chapter 8.

12. A model is considered *verified* if the implemented model matches the conceptual model. Of course, since conceptual models and implemented models are distinct entities, it is impossible to say a model is completely verified, but it is possible to say that model is verified to a certain extent.

system is behaving and do not tell you anything about individuals. Many EBMs assume that one aspect of the model directly influences, or causes, another aspect of the model, while ABMs allow indirect causation via emergence to have a larger effect on the model outcomes.

Randomness vs. Determinism

One important feature of agent-based modeling, and of computational modeling in general, is that it is easy to incorporate randomness into your models.[13] Many equation-based models and other modeling forms require that each decision in the model be made deterministically. In agent-based models this is not the case; instead, the decisions can be made based on a probability. For instance, in the Ants model, as the ants move around the landscape, their decisions are not completely determined; instead, at each time step they change their heading a small amount based on a random number. As a result, each ant follows a unique, irregular path. In reality, ants might be affected by small changes in elevation, the presence or absence of twigs and stones, and even the light of the sun. To build a complete model of all of these factors might be very tedious and would probably be very specific to a particular environment. Moreover, since our real goal in building this model is to understand how ants gather food and not how they move about the landscape, there is no guarantee that a more deterministic model will provide us with a better answer to this question. Thus, using the random number serves as an approximation that may turn out to be just as correct in answering our driving question.

The "random" Ants model is easier to describe than a deterministic one. If we are explaining the Ants model to a person who has never seen it before, we can say that at each time step the ants change their heading by a small random amount. We do not have to describe how the model takes into account all of the environmental factors that could be involved. This simplification also speeds up model development, since we do not need to spend time formalizing all of these details. If at a future time we decide that the ants do need to make more deterministic decisions about their environment, we can incorporate that knowledge at that time. Thus, though randomness in a model acts as an approximation to real world concepts, the model can later be made less approximate by the incorporation of additional knowledge.

Finally, there are often times when we simply do not know enough about how a complex system works in order to build a completely deterministic model. In many of these cases the only type of model that we can build is a model with some random elements. Agent-based modeling and other modeling forms that allow you to incorporate random features are essential to studying these kinds of systems.

13. Since computers are deterministic machines the randomness that they possess is not true randomness but rather "pseudo-randomness." This will be discussed further in chapter 5.

When Is ABM Most Beneficial?

Agent-based modeling has some benefits over other modeling techniques, but, as with any tool, there are contexts in which it is more useful than others. ABM can be used to model just about any natural phenomenon (e.g., you could describe any phenomenon by describing the interaction of its subatomic particles). However, there are some contexts for which the cost of building an ABM exceeds the benefits, and there are other times when the benefits are extraordinary given the costs. It is sometimes difficult to discern the difference between these two scenarios. However, there are a few general guidelines that can help to identify situations where ABM will be particularly valuable. These are meant as guidelines and not particular prescriptions or "rules" about when to use ABM. Most often, you will have to judge based on the particular situation.

Some problems with large number of homogenous agents are often better modeled (i.e., they will provide more accurate solutions to aggregate problems faster) using an aggregate solution like mean field theory or system dynamics modeling (Opper & Saad, 2001; Forrester, 1968). For instance, if you are concerned about the temperature in a room, then tracking every individual molecule and its history is not necessary. On the other hand, if a problem has only a handful of interacting agents, then you usually do not need to bring to bear the full power of ABM and instead can write detailed equations describing the interaction—two billiard balls colliding, for example, does not require ABM. As a rule of thumb, agent-based models are most useful when there are a medium number (tens to millions) of interacting agents (Casti, 1995).

Agent-based models are more useful when the agents are not homogenous. For instance, modeling all the trades and events on a stock market floor requires a more rich and detailed examination of individual-level behavior. Different stock trading agents have different risk thresholds and hence will not make the same decision given the same environmental state. Even in the Ants model, while the ants all had the same rules of behavior, they were not homogenous in location, heading, food-carrying state, and so on. ABM is very useful when agents are heterogeneous and the heterogeneity of the agents affects the overall performance of the system. Since ABM enables each individual to be tracked and described at the individual level, it is much more powerful than techniques such as systems dynamics modeling (Forrester, 1968; Sterman, 2000; *Richmond & Peterson, 1990*). System dynamics modeling requires the creation of a separate "stock" for each group of agents with different properties, and, when the space of properties is large, this becomes difficult to build, track, and integrate. ABM, on the other hand, requires you only to specify how agents' properties are defined and not to keep track of all possible agent types, which provides a more concise description of a complex system. Thus, using ABM is especially beneficial when the agents are heterogeneous.

Having heterogeneous agents also allows the interactions between agents to be quite complex. Since we can specify an almost infinite number of different agent types, we can specify just a few simple rules to describe how those agents interact with each other to

create a very rich tapestry of interactions. Moreover, since ABM allows individual agents to keep a history of interactions, they can change their behaviors, and even their strategies, based on past events.[14] For example, in an ABM of the evolution of cooperation, it is possible that agents can learn and hence modify their behavior as a result of continual interaction with a particular group of agents. They may learn to distrust that group, or alternatively they may learn to act more favorably toward that group. Thus ABM is very useful when modeling complex interactions of adaptive agents.

In the same way that ABM is useful when the interaction between agents is complex, it is also useful when the agents' interaction with the environment is complex. The environment in an ABM is often itself composed of stationary agents, and thus modeling agent-environment interactions has all of the power of modeling any agent-to-agent interaction. For example, in an ABM of fish ecology, a fisherman can recognize a particular location as a place that he has fished before and decide not to fish there again. This agent-environment interaction enables geographic and location-dependent information to be included in the model, and thus we get richer data than a geographic-independent model. In the fish model, it may be known that the average fish population is steady over time for a large area, but that could mean that the average fish population is steady in all of the subareas, or that different subareas trade off with each other, resulting in a larger fish population in some places and a smaller one in others. Thus, the rich description of environment and geography entailed by ABM allows for the generation of more detailed information. This enables ABM to generate spatial patterns of results as opposed to spatially homogenous aggregate results.

Another way that ABM provides more detailed information than equation-based or many other modeling approaches is through its rich conception of time. In ABM, one models agents and their interactions with each other. These interactions occur temporally; that is, some interactions occur before or after others. ABM thus enables you to move beyond a static snapshot of the system and toward a dynamic understanding of the system's behavior. In this way, ABM provides a rich and detailed account of the process of a system's unfolding in time, and not just the final state of the system. For example, in a stock market model you can actually observe individuals buying and selling stocks over time instead of modeling only the change in the stock's price. By enabling a detailed conception of time, ABM vastly expands on the detail of the resultant model.

Trade-offs of ABM

Agent-based modeling provides some benefits over other methods of modeling, but, in any particular situation, choosing a modeling methodology is a case of choosing the

14. Strategies are distinct from behavior because they express how to behave in a particular set of circumstances. Thus, a change in strategy often results in a change of behavior, but a change in behavior is not necessarily the result of a change in strategy.

(the code represents the conceptual model) and validated (the model has a correspondence to the real world). We also cover issues to consider in replicating an ABM. Since validation and replication usually require statistical comparison, there is a short introduction to the necessary statistics.

In chapter 8 we tie many of these threads together to discuss how ABM is applied in real-world settings and examine advanced uses of ABM. We highlight some of the principal examples from domains such as ecology, economics, land-use planning, computer science, and political science. We discuss what ABM methodology has contributed to scientific knowledge and what it will be used for in the future. We discuss how to incorporate these richer data sources into your ABM. These sources include GIS, Social Network Analysis, and sensor data (visual and nonvisual). We address how to export data from ABMs to advanced mathematical analysis packages. We also discuss how to make ABMs more powerful, by incorporating techniques such as machine learning, system dynamics modeling, and participatory simulation. We conclude with a discussion of future research trends and challenges within ABM and upcoming areas of applications of ABM to new knowledge domains.

In the appendix, we examine the origins and history of ABM, with an emphasis on its computational roots. This is provided to set a historical context for the rest of the book enabling us to understand how a variety of fields came together to create what we now call ABM. Some readers may wish to read the appendix at this time, though it can be completely skipped for the reader focused on model building. But before we embark on our ABM journey, allow us to take one last look at the Ants model.

Conclusion

The Ants model is interesting for biologists, and we have even discussed how it can be used to reason analogically about other systems, like computer networks and path planning. But what if we wanted to transform the Ants model to be more like some other system? There are many similarities between ant colonies and human organizational systems. They both exhibit problem-solving behavior. They both are results of organized structures that have evolved over the millennia. For example, what if we tried to reconceptualize the ants in the system as humans? Then we can visualize the ant colony as the central business district of a town. With this slight shift in perspective, we can start to see how the model could resemble a human city, with individuals that go off to work every day and return in the evening.

The last description suggests a major difference between the ant and the human systems. Humans tend to leave for work around the same time in the morning and return home around the same time in the evening. So we need to modify our model slightly: Instead of having the ants (now humans) leave randomly from the nest, we have them leave at random intervals around a start time, go off to find some food, and stay near

the food until a certain amount of time passes, then return to their homes. And humans do not live in a colony; they live in different locations around a city, so we need to give each human a different home that they start with, and then allow them to walk (with some randomness) to their work. But humans do not walk randomly (well, not much of the time); they instead take preplanned routes on roads. So now let us put down a road network for them to drive on their way to work. But if they are driving to work, then they will be limited by the speed of the traffic. So now we need to implement a vehicle simulation on the roads. And so it goes. … Slowly, our model of one specific ant blossoms into a model of many ants collecting food, which then metamorphoses into a model of urban commuting patterns. A powerful aspect of ABM is that it enables us to find universal patterns that characterize apparently quite different phenomena, to generate these patterns with simple rules, and to explore the effects of simple modifications to those rules.

What models will you build? What are the simple rules that describe the agents in your model? What are your agents? Are they humans, ants, cars, computers, deer, viruses, cells, coffee trees, hurricanes, air particles, electrons, snowflakes, sand grains, students, teachers, videogames, marketing strategies, innovations, or any of a vast number of objects, events, or things? Whatever it is you want to model, ABM provides you with tools and capabilities that enable you to simulate and analyze it as a complex system. As you progress through this book, you will be introduced to the tools and develop the skills that you need to explore the world around you in an agent-based way.

At this point, it is recommended that you work through the three NetLogo tutorials found in the NetLogo user manual that is available from the help menu of the NetLogo application. It will be necessary to work through the tutorials to do many of the explorations at the end of this chapter and to follow chapter 2.

Explorations

Beginner NetLogo Explorations

1. Complete the tutorials that are available in the NetLogo User Manual.
2. Look over the models in the Sample Models section of the NetLogo models library. The models are grouped by subject area. Pick out a model you find interesting and try running it in different ways. What set of parameters gives you the most interesting behavior? Is there a way to change a parameter in a small way and get a very different behavior? Explicitly describe the rules the agents are following.
3. Describe a phenomenon that you think it would be interesting to model using ABM. What are the agents in this model? What properties do they have? What kind of actions can the agents take? What kind of environment do the agents exist in? What is the order of events that occurs at each time step of the model? What types of output will this model generate? What do you expect to observe as a result of running this model?

Ants and Other Model Explorations

4. Examine the code for the NetLogo Ants model, described in the chapter. The wiggle procedure right now has the ant turn a random amount to the left and then back to the right a random amount. This approximates a random walk that is centered on moving straight ahead. If you changed the procedure so the walk was biased to the left or to the right, how would that change the results? What if the limits of how much the ant turned were changed? Make these modifications and observe the results.
5. *Termites model* Run the Termites model (found in the Biology section of the NetLogo models library). In this model there are only two objects: termites and wood chips. What are the termites doing in this model? Without looking at the code or the info window, can you describe the rules governing the termites' behavior? Hint: It might help to reduce the number of termites and wood chips and to slow down the speed slider.
6. *Daisyworld model* Some ABMs are used not as models of real-world events, but rather as thought experiments. Run the Daisyworld model (found in the Biology section of the NetLogo models library). This model defines a world in which the whole surface is covered by daisies, and it examines how different factors affect the global temperature of the world. Adjust the parameters of the model and observe how the model reacts. The standard parameters result in a temperature slightly below 50. Find a set of parameters that move the temperature closer to 12. This model rarely results in a constant temperature; it usually oscillates. Describe this oscillation.

Concept Explorations

7. *Modeling at different levels* Agent-based models can be written at different levels. For example, one model may have agents that are populations of wolves and sheep, whereas another model may have agents that are individual wolves and sheep. Write a description of how packs of wolves interact with flocks of sheep at the group level. Now write a description of how individual sheep interact with individual wolves. How are your descriptions different? What phenomena are you describing? At what times would the group level description be helpful? At what times would the individual level description be helpful?
8. *Emergence and ABM* Agent-based models often exhibit emergent properties. One characteristic of an emergent phenomenon is that the system exhibits a property that is not defined at the individual level. For instance, examine the Traffic Basic model (found in the Social Sciences section of the models library). Run the model several times and observe the results. What causes the traffic jams in the model? Does there appear to be any external event that causes them? Inspect the cars. Is there any property of these cars that describes a traffic jam? If one car moves slowly, is that enough to cause traffic jams?
9. *ABM for education and understanding* ABM provides us with a new way of understanding the world around us. ABM has many uses in research. But ABM also has great potential as a tool for education. For instance, molecules in a free gas can be thought of

as agents moving around and colliding with each other. Examine the GasLab Free Gas model (found in the Chemistry and Physics section of the models library). Do you think this model is easier to understand than a traditional equation-based approach to understanding Free Gas phenomena? What affordances does the ABM approach give us that traditional approaches lack? Are there ways that ABM can be more confusing than traditional approaches? If so how?

NetLogo Explorations

10. Create at least two different ways of distributing turtles randomly across the screen. In one method, use only turtle motion commands such as `forward`, `left`, and `right`. In another method, use `set` or `setxy`. Create buttons to launch these procedures. Compare and contrast your different methods. Is one of these more efficient? Is one of them more realistic? In what situations would each of them have advantages over the other?
11. Write a procedure to get a color to spread from patch to patch. (There are many ways to do this. Pick one you like.) Create a button to launch this procedure.
12. Write a procedure that makes the turtles chase after the mouse cursor. Create a button to launch this procedure.
13. Select a new shape for the turtles from the shapes editor, and then create a "cloud" of turtles (a bunch of turtles in the same local area) using your new shape. Create some green patches. Make the turtles follow the mouse cursor around the screen but avoid the green patches. Make the green color spread from green patches to other patches nearby. Create buttons to launch these procedures.
14. Create a "cloud" of turtles, half of them one color and half of them another color. Based on a probability have one color of turtles move up and the other color turtles move down. Label the turtles with their `who` number. Create buttons to launch these procedures. Create a monitor that keeps track of how many times one of the turtles has moved.
15. (a) Create a new model with a `setup` procedure that creates turtles. (b) Create a slider that controls the number of turtles created. (c) Write a `go` procedure that makes the turtles wander around the screen randomly. (d) Change the `go` procedure to make the turtles afraid of each other. (e) Make the turtles die when they reach the edge of the screen. (f) Create a plot that displays the number of turtles.
16. Write two procedures. In the `setup` procedure, turn the left side of the screen red, and the right side of the screen green and create two turtles. Give one turtle a shape from the shapes editor and make a new shape for the other turtle. In the `go` procedure, make the turtles move randomly about the screen. When a turtle is in an area of one color, create a circle of patches of the other color centered on the turtle.
17. Both turtles and patches can create visual images in the NetLogo view. Create a turtle and have it draw a circle (using the `pen-down` command). Create the outline of a circle with patches without using a turtle pen. Write a procedure that asks a turtle to draw a square given a starting location and side length. Write a similar procedure using patches.

Compare and contrast the code for these two procedures. Which set of code is more compact? Are there advantages or disadvantages to using patches or turtles to accomplish this task?

18. Open the *random walk example* (found in the Code Examples folder in the models library). Inspect the code. What do you predict the turtle's path will look like? Run the model. Does the path look like you expected it would? Modify the model code so that the turtle's path is still random but is less "jagged," i.e., is smoother and straighter.

19. Most of this textbook addresses the use of ABM to model and scientifically explain phenomena. However, you can also use ABM to create powerful visualizations. As we have mentioned, ABM has even been used to create Academy Award–winning special effects. Examine the Particle Systems Basic model (found in the Computer Science section of the models library). This model creates interesting visual images from the manipulation of simple agents. Explore this model, and understand how the agents behave and what properties they have. Describe a non-agent-based model that would create similar results as the base model with the initial parameters. Now examine the NetLogo model again. Change the initial number of particles, the step size, and the gravity. Can you describe both an agent-based model and a non-agent-based model that creates these results? Which of these two models is easier to describe? Why?

20. *Computer modeling and chaos theory* Chaos theory was developed from traditional equation-based modeling, but one of its inspirations came from computer modeling. Edward Lorenz discovered that mathematical systems could produce very different results depending on the initial conditions that the systems have. He realized this because he tried to restart a computer model of the weather system halfway through a run with a new set of parameters that lacked a small amount of precision from his previous set of parameters. The resulting model behaved very different from his original model. Agent-based models can exhibit this same "sensitivity to initial conditions." For instance, examine the Sunflower model (found in the Biology section of the models library). This is an agent-based model of how rows of sunflower seeds are added to a sunflower. Run the model with the default settings. Now change one of the parameters. Run the model again. Repeat this process. As you keep manipulating the parameters, do you eventually get to the point where you can predict the behavior of the model with new parameters? Explain your answer. Why is this model predictable or unpredictable?

(A) Before: live cell (starred). (B) After: the starred cell dies.

The game is played on a large grid, such as a checkerboard or graph paper. Let's say we are playing on a square grid with 51 squares[1] (or "cells") on a side. Each cell can be either "alive" or "dead." This is called the "state" of the cell. Every cell is surrounded by eight "neighbor" cells. The grid is considered to "wrap around" so that a cell on the left edge has three (3) neighbor cells on the right edge and, similarly, a cell on the top edge has three (3) neighbor cells on the bottom edge. There is a central clock. The clock ticks establish a unit of time. In the game of Life, the unit of time is called a generation. More generally, in agent-based models, the unit of time is referred to a tick. Whenever the clock ticks, each cell updates its state according to the following rules:

Each cell checks the state of itself and its eight neighbors and then sets itself to either alive or dead. In the rule descriptions that follow, blue cells are "dead," green cells are "alive" and the yellow stars indicate the cells affected by the rule described.

(1) If the cell has less than two (2) alive neighbors, it dies (figure 2.1).
(2) If it has more than three (3) alive neighbors, it also dies (figure 2.2).
(3) If it has exactly two (2) alive neighbors, the cell remains in the state it is in (figure 2.3).
(4) If it has exactly three (3) alive neighbors, the cell becomes alive if it is dead, or stays alive if it is already alive (figure 2.4).

1. The standard setup for the NetLogo grid is to have an odd number of cells, so that there is always a center cell. Other setups are possible and are described in chapter 5.

(A) Before: starred cell surrounded by more than 3 live neighbors. (B) After: starred cell dies.

(A) Before: 3 starred cells each with two live neighbors. (B) After: each starred cell stays alive.

(A) Before: 2 starred dead cells have 3 live neighbors. (B) After: 2 starred dead cells become alive.

Since Gardner's publication of Conway's Game of Life, many people have explored it and have been surprised by the wide diversity of shapes and patterns that "emerge" from these simple rules. We will now use NetLogo to construct the Game of Life.

We will begin by reviewing the basic NetLogo elements. We start by opening the NetLogo application. The application opens with a blank interface with a large black square in it (see figure 2.5). The black square is the known as the "view" and is the area in which we will play the Game of Life. The surrounding white area is known as the "interface" and is where we can set up user-interface elements such as buttons and sliders. Right-click on the view and select "edit" from the drop-down menu. You will see the Model Settings dialog (figure 2.6) where we can configure some basic settings for our NetLogo model.

Every NetLogo model consists of three tabs.[2] The tab that you are looking at now is the *Interface* tab, where we work with widgets and observe model runs. Next we will work with the *Code* tab, where we write the model procedures.

A third very important tab is the *Info* tab. In this chapter, we will not work with the *Info* tab in detail, but it is an important part of any model. This is where model authors put the information about their models. Details on how the Info tab is structured can be found in the "Sections of the Info Tab" box in chapter five. The Info tab is a very useful resource for exploring a NetLogo model and we recommend that Textbook readers read them carefully when working with models and take the time to write good Info tabs for models you create.

2. In some versions of NetLogo, there is a fourth "Review" tab.

The NetLogo application at startup.

Now let us return to the Interface tab, and begin developing our Life-simple model. The view is composed of a grid of cells known, in NetLogo parlance, as patches. Click Settings in the toolbar of the Interface tab. By default, the view origin is at the center of the view and its current maximum x-coordinate and y-coordinate is 16. To begin developing our Life-simple model, we will change the values of `max-pxcor` and `max-pycor` to 25, which will give us a grid of 51 by 51 patches, for a total of 2,601 patches. This creates a much larger world enabling more space to create the elements of the Game of Life. To keep the view a manageable size on the screen, we will change the default patch size from 13 to 8. As the Game of Life is played in a wrapping grid, we keep the wrapping checkboxes checked (as they are by default, we will explain these in more detail in chapter 5). We can now click OK and save these new settings. (See figure 2.7.)

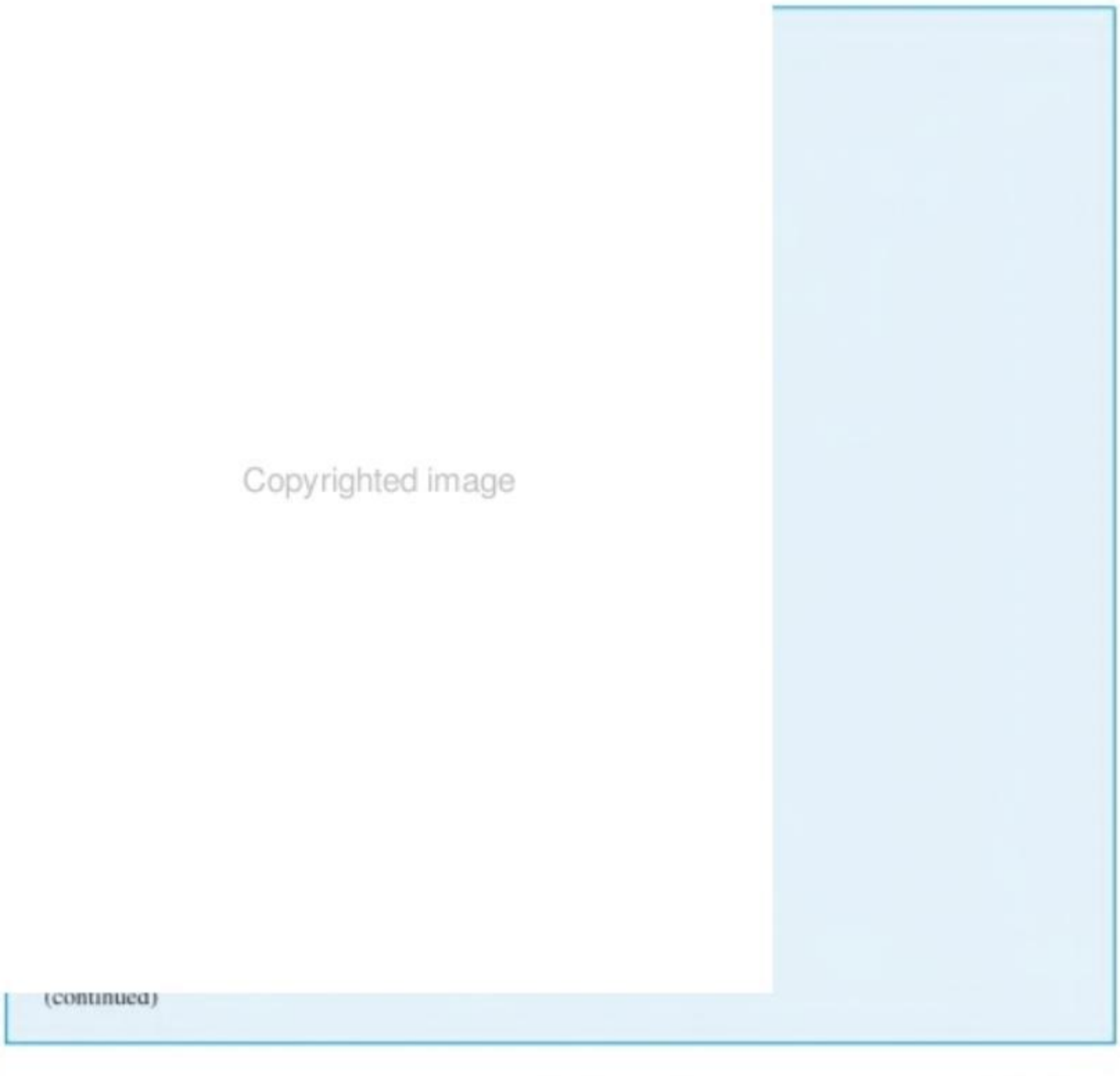
Figure 2.6
The NetLogo Model Settings dialog before adjusting the settings.

The Game of Life is played on a grid of cells, so we will consider each NetLogo patch as a different cell. Life has two kinds of cells in it, "live" cells and "dead" cells. We choose to model live cells as green patches and dead cells as blue patches. Once we have thought through what the model will look like we still need to create the model instructions. To do this, we will need to write NetLogo instructions (or code) in the Code tab. We select the Code tab and begin to write our code. NetLogo code takes the form of modules known as "procedures." Each procedure has a name and begins with the word `to` and ends with the word `end`.

Our Life model (Life-Simple in the IABM Textbook folder of the NetLogo models library) will consist of two procedures: `setup`, which initializes the game, and `go`, which advances the clock by one tick.

We create the `setup` procedure as follows:

Box 2.1
(continued)



(continued)

versions of our game sequentially without having to close NetLogo and open it back up. The second section issues commands to (or makes requests of) all of the patches. In NetLogo, we are polite in our interactions with agents, so to issue commands to the patches we use the form `ask patches`. We then enclose our requests to the patches in brackets. However, do not mistake our politeness—the patches have no choice but to do as they are asked. As a result, you will often find that in this book, we will slip interchangeably between the language of commands and requests as synonymous. In the second section of our `setup` code, there are two commands/requests. The first asks each patch to set its color (`pcolor` which stands for patch color) to blue, making all the cells dead. When creating agent-based models, it is useful to think of commands to agents in an agent-centric way. This will help you to understand how the model works, as described in the box that follows: