CS620 - Modelling and Simulation
Virtual University of Pakistan
Prepared by: Muhammad Ali Imran Jalali
VU ID: BC220214205
FROM Module 1 to 130

INTRODUCTION TO SIMULATION

- A **simulation** is the imitation of the operation of a real-world process or system. Whether done by hand or on a computer, simulation involves the generation of an artificial history of a system and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system.
- The **behavior** of a system as it evolves is studied by developing a simulation model. This model usually takes the form of a set of assumptions concerning the operation of the system. These assumptions are expressed in mathematical, logical, and symbolic relationships between the entities, or objects of interest, of the system.
    - Once developed and validated, a model can be used to investigate a wide variety of **"what if"** questions about the real-world system. Potential changes to the system can first be simulated, to predict their impact on system performance.
    - Simulation can also be used to study systems in the design stage before such systems are built. Thus, simulation modeling can be used both as an analysis tool for predicting the effect of changes to existing systems and as a design tool to predict the performance of new systems under varying sets of circumstances.
- In some instances, a model can be developed which is simple enough to be "solved" by mathematical methods. Such solutions might be found by the use of differential calculus, probability theory, algebraic methods, or other mathematical techniques. The solution usually consists of one or more numerical parameters, which are called **measures of the performance** of the system. However, many real-world systems are so complex that models of these systems are virtually impossible to solve mathematically. In these instances, numerical, computer-based simulation can be used to imitate the behavior of the system over time.
  From the simulation, data are collected as if a real system were being observed. This simulation-generated data is used to estimate the measures of performance of the system.

The availability of special-purpose simulation languages, massive computing capabilities at a decreasing cost per operation, and advances in simulation methodologies have made simulation one of the most widely used and accepted tools in operations research and systems analysis.

Circumstances under which simulation is the appropriate tool to use have been discussed by many authors. Simulation can be used for the following purposes:

1. Simulation enables the study of, and experimentation with, the interactions of a complex system or a subsystem within a complex system.

2. Informational, organizational, and environmental changes can be simulated, and the effect of these alterations on the model's behavior can be observed.
3. The knowledge gained during the designing of a simulation model could be of great value toward suggesting improvement in the system under investigation.
4. Changing simulation inputs and observing the resulting outputs can produce valuable insight into which variables are the most important and into how variables interact.
5. Simulation can be used as a pedagogical device to reinforce analytic solution methodologies.
6. Simulation can be used to experiment with new designs or policies before implementation, to prepare for what might happen.
7. Simulation can be used to verify analytic solutions.
8. Simulating different capabilities for a machine can help determine the requirements on it.
9. Simulation models designed for training make learning possible without the cost and disruption of on-the-job instruction.
10. An animation shows a system in simulated operation so that the plan can be visualized.
11. The modern system (factory, wafer fabrication plant, service organization, etc.) is so complex that its internal interactions can be treated only through simulation.

This section is based on an article by Banks and Gibson [1997], who gave ten rules for evaluating when a simulation is not appropriate.

Rule 1: The first rule indicates that simulation should not be used when the problem can be solved by common sense. An example is given of an automobile tag facility serving customers who arrive randomly at an average rate of 10/hour and are served at a mean rate of 12/hour. To determine the minimum number of servers needed, simulation is not necessary. Just compute 100/12 = 8.33 indicating that nine or more servers are needed.

Rule 2: The second rule says that simulation should not be used if the problem can be solved analytically. For example, under certain conditions, the average waiting time in the example above can be found from curves that were developed by Hillier and Lieberman [2002].

Rule 3: The next rule says that simulation should not be used if it is easier to perform direct experiments. An example of a fast-food drive-in restaurant is given where it was less expensive to stage a person taking orders using a hand-held terminal and voice communication to determine the effect of adding another order station on customer waiting time.

Rule 4: The fourth rule says not to use simulation if the costs exceed the savings. There are many steps in completing a simulation, as will be discussed, and these must be done thoroughly. If a simulation study costs $20,000 and the savings might be $10,000, then a simulation would not be appropriate.

Rule 5 & Rule 6: Rules five and six indicate that simulation should not be performed if the resources or time are not available. If the simulation is estimated to cost $20,000 and there is only $10,000 available, the suggestion is not to venture into a simulation study. Similarly, if a decision is needed in two weeks and a simulation will take a month, the simulation study is not advised.

Rule 7: Simulation takes data, sometimes lots of data. If no data is available, not even estimates, simulation is not advised.

Rule 8: The next rule concerns the ability to verify and validate the model. If there is not enough time or if the personnel are not available, simulation is not appropriate.

Rule 9: If managers have unreasonable expectations, if they ask for too much too soon, or if the power of simulation is overestimated, then simulation might not be the most appropriate tool.

Rule 10: Last, if system behavior is too complex or can't be defined, simulation is not appropriate. Human behavior is sometimes extremely complex to model.

Simulation is intuitively appealing to a client because it mimics what happens in a real system or what is perceived for a system that is in the design stage. The output data from a simulation should directly correspond to the outputs that could be recorded from the real system. Additionally, it is possible to develop a simulation model of a system without dubious assumptions (such as the same statistical distribution for every random variable) of mathematically solvable models. For these and other reasons, simulation is frequently the technique of choice in problem-solving.

Simulation Models vs. Optimization Models

In contrast to optimization models, simulation models are "run" rather than solved. Given a particular set of input and model characteristics, the model is run and the simulated behavior is observed. This process of changing inputs and model characteristics results in a set of scenarios that are evaluated. A good solution, either in the analysis of an existing system or in the design of a new system, is then recommended for implementation.

Simulation has many advantages, but some disadvantages. These are listed by Pegden, Shannon, and Sadowski [1995].

Some advantages of Simulation:

1. New policies, operating procedures, decision rules, information flows, organizational procedures, and so on can be explored without disrupting the ongoing operations of the real system.

2. New hardware designs, physical layouts, transportation systems, and so on can be tested without committing resources for their acquisition.

3. Hypotheses about how or why certain phenomena occur can be tested for feasibility.

4.· Time can be compressed or expanded to allow for a speed-up or slow-down of the phenomena under investigation.

5. Insight can be obtained about the interaction of variables.

6. Insight can be obtained about the importance of variables to the performance of the system.

Disadvantages of Simulation

1. Model building requires special training. It is an art that is learned over time and through experience. Furthermore, if two models are constructed by different competent individuals, they might have similarities, but it is highly unlikely that they will be the same.

2. Simulation results can be difficult to interpret Most simulation outputs are essentially random variables (they are usually based on random inputs), so it can be hard to distinguish whether an observation is a result of system interrelationships or randomness.

3. Simulation modeling and analysis can be time-consuming and expensive. Skimping on resources for modeling and analysis could result in a simulation model or analysis that is not sufficient for the task.

4. Simulation is used in some cases when an analytical solution is possible, or even preferable, as was discussed earlier. This might be particularly true in the simulation of some waiting lines where closed-form queueing models are available.

Counteracting the disadvantages of simulation: In defense of simulation, these four disadvantages, respectively, can be offset as follows:

1. Vendors of simulation software have been actively developing packages that contain models that need only input data for their operation. Such models have the generic tag "simulator" or "template."

2. Many simulation software vendors have developed output-analysis capabilities within their packages for performing very thorough analyses.

3. Simulation can be performed faster today than yesterday and will be even faster tomorrow, because of advances in hardware that permit rapid execution of scenarios and because of advances in many simulation packages. For example, some simulation software contains constructs for modeling material handling that uses such transporter as fork-lift trucks, conveyors, and automated guided vehicles.

4. Closed-form models are not able to analyze most of the complex systems that are encountered in practice. In many years of consulting practice by two of the authors, not one problem was encountered that could have been solved by a closed-form solution.

The applications of simulation are vast. The Winter Simulation Conference (WSC) is an excellent way to learn more about the latest in simulation applications a theory. There are also numerous tutorials at both the beginning and the advanced levels. WSC is sponsored by six technical societies and the National Institute of Standards and Technology (IST). The technical societies are the American Statistical Association (ASA), Association for Computing Machinery/Special Interest Group on Simulation (ACMIGSI), Institute of Electrical and Electronics Engineers: Computer Society (IEECS), Institute of Electrical and Electronics Engineers: Systems, Man, and Cybernetics Society (IEE, SMCS), Institute of Industrial Engineers (IlE), Institute for Operations Research and the Management Sciences: College on Simulation (INFORMS/CS) and The Society for Computer Simulation (SCS). Note that IEEE is represented by two bodies. Information about the upcoming WSC can be obtained from www.wintersim.org. WSC programs with full papers are available from www.informs-cs.org/wscpapers.html. Some presentations, by area, from a recent WSC a listed next:

Manufacturing Applications

- Dynamic modeling of continuous manufacturing systems, using analogies to electrical systems
- Benchmarking of a stochastic production planning model in a simulation testbed
- Paint line color change reduction in automobile assembly
- Modeling for quality and productivity in steel cord manufacturing
- Shared resource capacity analysis in biotech manufacturing
- Neutral information model for simulating machine shop operations

Semiconductor Manufacturing

- Constant time interval production planning with application to work-in-process control
- Accelerating products under due-date oriented dispatching rules
- Design framework for automated material handling systems in 300mm wafer fabrication factories
- Making optimal design decisions for next-generation dispensing tools
- Application of cluster tool modeling in a 300mm wafer fabrication factory.
- Resident-entity based simulation of batch chamber tools in 300mm semiconductor manufacturing

Construction Engineering and Project Management

- Impact of multitasking a merge bias on procurement of complex equipment
- Application of lean concepts and simulation for drainage operations maintenance crews
- Building a viral shop model for steel fabrication
- Simulation of the residential lumber supply chain

The applications of simulation are vast. The Winter Simulation Conference (WSC) is an excellent way to learn more about the latest in simulation applications a theory. There are also numerous tutorials at both the beginning and the advanced levels. WSC is sponsored by six technical societies and the National Institute of Standards and Technology (IST). The technical societies are the American Statistical Association (ASA), Association for Computing Machinery/Special Interest Group on Simulation (ACMIGSI), Institute of Electrical and Electronics Engineers: Computer Society (IEECS), Institute of Electrical and Electronics Engineers: Systems, Man, and Cybernetics Society (IEE, SMCS), Institute of Industrial Engineers (IlE), Institute for Operations Research and the Management Sciences: College on Simulation (INFORMS/CS) and The Society for Computer Simulation (SCS). Note that IEEE is represented by two bodies. Information about the upcoming WSC can be obtained from www.wintersim.org. WSC programs with full papers are available from www.informs-cs.org/wscpapers.html. Some presentations, by area, from a recent WSC a listed next:

Manufacturing Applications

- Dynamic modeling of continuous manufacturing systems, using analogies to electrical systems
- Benchmarking of a stochastic production planning model in a simulation testbed
- Paint line color change reduction in automobile assembly
- Modeling for quality and productivity in steel cord manufacturing
- Shared resource capacity analysis in biotech manufacturing
- Neutral information model for simulating machine shop operations

Semiconductor Manufacturing

- Constant time interval production planning with application to work-in-process control
- Accelerating products under due-date oriented dispatching rules
- Design framework for automated material handling systems in 300mm wafer fabrication factories
- Making optimal design decisions for next-generation dispensing tools
- Application of cluster tool modeling in a 300mm wafer fabrication factory.
- Resident-entity based simulation of batch chamber tools in 300mm semiconductor manufacturing

Military Applications

- Multinational Intra-Theatre Logistics Distribution
- Examining Future Sustainability of Canadian Forces Operations
- Feasibility Study for Replacing the MK19 Automatic Grenade Launching System
- Training Joint Forces for Asymmetric Operations
- Multi-Objective Unmanned Aerial Vehicle Mission Planning
- Development of Operational Requirements Driven Federations

Logistics, Transportation, and Distribution

- Operating Policies for a Barge Transportation System
- Dispensing Plan for Emergency Medical Supplies in the Event of Bioterrorism
- Analysis of a Complex Mail Transportation Network
- Improving the Performance of Container Terminals
- Yard Crane Dispatching Based on Real-Time Data
- Unit Loading Device Inventory in Airline Operations
- Inventory Systems with Forecast Based Policy Updating
- Dock Allocation in a Food Distribution Center
- Operating Policies for a Barge Transportation System

Business Processing

- A New Policy for the Service Request Assignment Problem
- Process Execution Monitoring and Adjustment Schemes
- In-Store Merchandising of Retail Stores
- Sales Forecasting for Retail Small Stores

Health Care

- Interventions to Reduce Appointment Lead-Time and Patient No-Show Rate
- Supporting Smart Thinking to Improve Hospital Performance
- Verification of Lean Improvement for Emergency Room Process
- Reducing Emergency Department Overcrowding
- Inventory Modeling of Perishable Pharmaceuticals
- Implementation of an Outpatient Procedure Center
- Infectious Disease Control Policy
- Balancing Operating Room and Post-Anesthesia Resources
- Cost-Effectiveness of Colorectal Cancer Screening Tests

Conclusion

Simulation can be performed faster today than yesterday and will be even faster tomorrow, because of advances in hardware that permit the rapid execution of scenarios and because of advances in many simulation packages.

For example, some simulation software contains constructs for modeling material handling that uses such transporters as fork-lift trucks, conveyors, and automated guided vehicles

Closed-form models are not able to analyze most of the complex systems that are encountered in practice.

In many years of consulting practice, not one problem was encountered that could have been solved by a closed-form solution. We have seen such a wide variety of areas where simulation has applications.

Military Applications

- Multinational Intra-Theatre Logistics Distribution
- Examining Future Sustainability of Canadian Forces Operations
- Feasibility Study for Replacing the MK19 Automatic Grenade Launching System
- Training Joint Forces for Asymmetric Operations
- Multi-Objective Unmanned Aerial Vehicle Mission Planning
- Development of Operational Requirements Driven Federations

Logistics, Transportation, and Distribution

- Operating Policies for a Barge Transportation System
- Dispensing Plan for Emergency Medical Supplies in the Event of Bioterrorism
- Analysis of a Complex Mail Transportation Network
- Improving the Performance of Container Terminals
- Yard Crane Dispatching Based on Real-Time Data
- Unit Loading Device Inventory in Airline Operations
- Inventory Systems with Forecast Based Policy Updating
- Dock Allocation in a Food Distribution Center
- Operating Policies for a Barge Transportation System

Business Processing

- A New Policy for the Service Request Assignment Problem
- Process Execution Monitoring and Adjustment Schemes
- In-Store Merchandising of Retail Stores
- Sales Forecasting for Retail Small Stores

Health Care

- Interventions to Reduce Appointment Lead-Time and Patient No-Show Rate
- Supporting Smart Thinking to Improve Hospital Performance
- Verification of Lean Improvement for Emergency Room Process
- Reducing Emergency Department Overcrowding
- Inventory Modeling of Perishable Pharmaceuticals
- Implementation of an Outpatient Procedure Center

- Infectious Disease Control Policy
- Balancing Operating Room and Post-Anesthesia Resources
- Cost-Effectiveness of Colorectal Cancer Screening Tests

Conclusion

Simulation can be performed faster today than yesterday and will be even faster tomorrow, because of advances in hardware that permit the rapid execution of scenarios and because of advances in many simulation packages.

For example, some simulation software contains constructs for modeling material handling that uses such transporters as fork-lift trucks, conveyors, and automated guided vehicles.

Closed-form models are not able to analyze most of the complex systems that are encountered in practice.

In many years of consulting practice, not one problem was encountered that could have been solved by a closed-form solution. We have seen such a wide variety of areas where simulation has applications.

To model a system, it is necessary to understand the concept of a **system and the system boundary**.

SYSTEM

- A **system** is defined as a group of objects that are joined together in some regular interaction or interdependence toward the accomplishment of some purpose. An example is a production system

  manufacturing automobiles. The machines, component parts, and workers operate jointly along an assembly line to produce a high-quality vehicle.

SYSTEM ENVIRONMENT

A system is often affected by changes occurring outside the system. Such changes are said to occur in the system environment [Gordon, 1978].

- In modeling systems, it is necessary to decide on the boundary between the system and its environment. This decision may depend on the purpose of the study.

- In the case of the factory system, for example, the factors controlling the arrival of orders may be considered to be outside the influence of the factory and therefore part of the environment. However, if the effect of supply on demand is to be considered, there will be a relationship between factory output and the arrival of orders, and this relationship must be considered an activity of the system.
- Similarly, in the case of a bank system, there could be a limit on the maximum interest rate that can be paid. For the study of a single bank, this would be regarded as a constraint imposed by the environment.

- In a study of the effects of monetary laws on the banking industry, however, the setting of the limit would be an activity of the system [Gordon, 1978]

- **System:** An organized entity made up of interrelated and interdependent parts.
- **Boundaries:** Barriers that define a system and distinguish it from other systems in the environment.
- **Homeostasis:** The tendency of a system to be resilient towards external factors and maintain its key characteristics.
- **Adaptation:** The tendency of a self-adapting system to make the internal changes needed to protect itself and keep fulfilling its purpose.
- **Reciprocal Transactions:** Circular interactions that systems engage in such that they influence one another.
- **Feedback Loop:** The process by which systems self-correct based on reactions from other systems in the environment.
- **Throughput:** Rate of energy transfer between the system and its environment during the time it is functioning.
- **Microsystem:** The system closest to the client.
- **Mesosystem:** Relationships among the systems in an environment.
- **Exosystem:** A relationship between two systems that has an indirect effect on a third system.
- **Macrosystem:** A larger system that influences clients, such as policies, administration of entitlement programs, and culture.
- **Chronosystem:** A system composed of significant life events that can affect adaptation.

To understand and analyze a system, several terms need to be defined.

ENTITY: An entity is an object of interest in the system.

ATTRIBUTE: An attribute is a property of an entity.

ACTIVITY: An activity represents a time period of a specified length.

EXAMPLE: If a bank is being studied, customers might be one of the entities, the balance in their checking accounts might be an attribute, and making deposits might be an activity.

The collection of entities that compose a system for one study might only be a subset of the overall system for another study [Law, 2007]. For example, if the aforementioned bank is being studied to determine the number of tellers needed to provide for paying and receiving, the system can be defined as that portion of the bank consisting of the regular tellers and the customers waiting in line. If the purpose of the study is expanded to determine the number of special tellers needed (to prepare cashier's checks, to conduct commercial transactions, etc.), the definition of the system must be expanded.

STATE: The state of a system is defined to be the collection of variables necessary to describe the system at any time, relative to the objectives of the study.

- In the study of a bank, possible state variables are the number of busy tellers, the number of customers waiting in line or being served, and the arrival time of the next customer.

EVENT: An event is defined as an instantaneous occurrence that might change the state of the system.

ENDOGENOUS VS. EXOGENOUS EVENTS: The term endogenous is used to describe activities and events occurring within a system, and the term exogenous is used to describe activities and events in the environment that affect the system.

- In the bank study, the arrival of a customer is an exogenous event, and the completion of the service of a customer is an endogenous event.

Table 1 lists examples of entities, attributes, activities, events, and state variables for several systems. Only a partial listing of the system components is shown. A complete list cannot be developed unless the purpose of the study is known. Depending on the purpose, various aspects of the system will be of interest, and then the listing of components can be completed.

Table 1 Examples of Systems and Their Components

System	*Entities*	*Attributes*	*Activities*	*Events*	*State Variables*
Banking	Customers	Checking-account balance	Making deposits	Arrival; departure	Number of busy tellers; number of customers waiting
Rapid rail	Riders	Origin; destination	Traveling	Arrival at station; arrival at destination	Number of riders waiting at each station; number of riders in transit
Production	Machines	Speed; capacity; breakdown rate	Welding; stamping	Breakdown	Status of machines (busy, idle, or down)
Communications	Messages	Length; destination	Transmitting	Arrival at destination	Number waiting to be transmitted
Inventory	Warehouse	Capacity	Withdrawing	Demand	Levels of inventory; backlogged demands

SYSTEM CATEGORIES: Systems can be categorized as discrete or continuous. "Few systems in practice are wholly discrete or continuous, but since one type of change predominates for most systems, it will usually be possible to classify a system as being either discrete or continuous" [Law, 2007].

DISCRETE SYSTEM: A discrete system is one in which the state variable(s) change only at a discrete set of points in time. The bank is an example of a discrete system: The state variable, the number of customers in the bank, changes only when a customer arrives or when the service provided by a customer is completed. **Figure 1** shows how the number of customers changes only at discrete points in time.

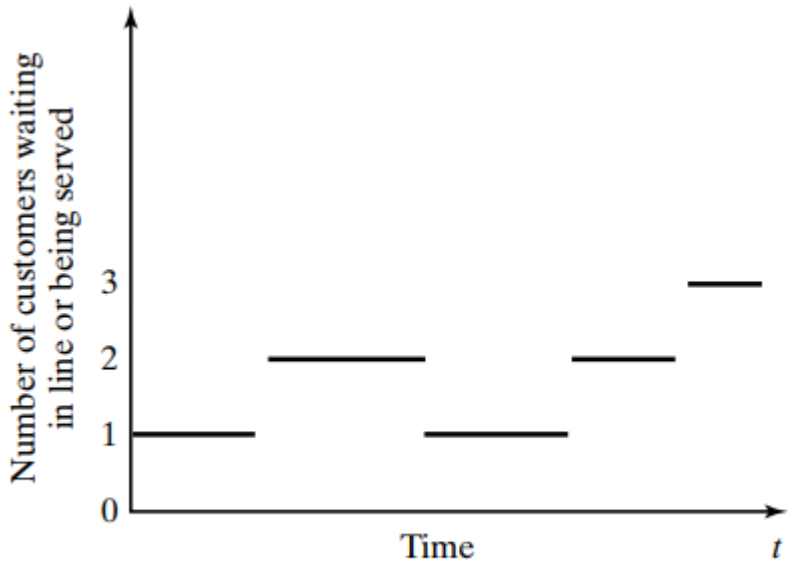


Figure 1 Discrete-system state variable.

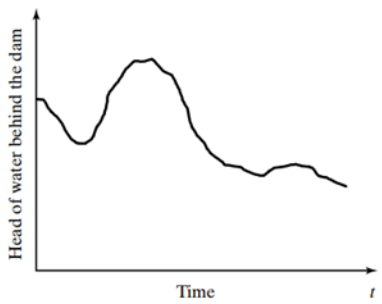


Figure 2 Continuous-system state variable.

CONTINUOUS SYSTEM: A continuous system is one in which the state variable(s) change continuously over time. An example is the head of water behind a dam. During and for some

time after a rainstorm, water flows into the lake behind the dam. Water is drawn from the dam for flood control and to make electricity. Evaporation also decreases the water level. Figure 2 shows how the state variable head of water behind the dam changes for this continuous system.

Why Model?

Sometimes it is of interest to study a system to understand the relationships between its components or to predict how the system will operate under a new policy. To study the system, it is sometimes possible to experiment with the system itself. However, this is not always an option. A new system might not yet exist; it could be in a hypothetical form only or at a design stage. Even if the system exists, it might be impractical to experiment with it.

BANK EXAMPLE: In the case of a bank, reducing the number of tellers to study the effect on the length of waiting lines might infuriate the customers so greatly that they will move their accounts to a competitor. Consequently, studies of systems are often accomplished with a model of a system.

Example: Consulting job for the simulation of a redesigned port in Western Australia

At US$300 million for a loading/unloading berth, it is not advisable to invest that amount only to find that the berth is inadequate for the task.

Model: A model is defined as a representation of a system to study that system. For most studies, it is only necessary to consider those aspects of the system that affect the problem under investigation. These aspects are represented in a model of the system; the model, by definition, is a simplification of the system.

On the other hand, the model should be sufficiently detailed to permit valid conclusions to be drawn about the real system. Different models of the same system could be required as the purpose of investigation changes.

Similar to the components of a system including entities, attributes, and activities, models are developed to represent them. However, the model contains only those components that are relevant to the study.

Types of Models: Models can be classified as being mathematical or physical.

Mathematical Model: A mathematical model uses the symbolic notation and mathematical equations to represent a system. A simulation model is a particular type of mathematical model of a system.

Physical model: A physical model is a larger or smaller version of an object such as the enlargement of an atom or a scaled-down version of the solar system.

Classification of Simulation Models: Simulation models may be further classified as being static or dynamic, deterministic or stochastic, and discrete or continuous.

Static Simulation Model: A static simulation model, sometimes called a Monte Carlo simulation, represents a system at a particular point in time.

Dynamic Simulation Model: Dynamic simulation models represent systems as they change over time. The simulation of a bank from 9:00 A.M. to 4:00 P.M. is an example of a dynamic simulation.

Deterministic Simulation Models: Simulation models that contain no random variables are classified as deterministic. Deterministic models have a known set of inputs, that will result in a unique set of outputs. Deterministic arrivals would occur at a dentist's office if all patients arrived at their scheduled appointment times.

Stochastic Simulation Model: A stochastic simulation model has one or more random variables as inputs. Random inputs lead to random outputs. Since the outputs are random, they can be considered only as estimates of the true characteristics of a model. The simulation of a bank would usually involve random interarrival times and random service times. Thus, in a stochastic simulation, the output measures—the average number of people waiting, the average waiting time of a customer—must be treated as statistical estimates of the true characteristics of the system.

Discrete and continuous Models: Discrete and continuous models are defined analogously to discrete and continuous systems discussed earlier. However, a discrete simulation model is not always used to model a discrete system, nor is a continuous simulation model always used to model a continuous system.

Tanks and pipes might be modeled discretely, even though we know that fluid flow is continuous. In addition, simulation models may be mixed, both discrete and continuous. The choice of whether to use a discrete or continuous (or both discrete and continuous) simulation model is a function of the characteristics of the system and the objective of the study. Thus, a communication channel could be modeled discretely if the characteristics and movement of each message were deemed important.

Conversely, if the flow of messages in aggregate over the channel were of importance, modeling the system via continuous simulation could be more appropriate. The models emphasized in this course are dynamic, stochastic, and discrete.

Types of Models: Models can be classified as being mathematical or physical.

Mathematical Model: A mathematical model uses the symbolic notation and mathematical equations to represent a system. A simulation model is a particular type of mathematical model of a system.

Physical model: A physical model is a larger or smaller version of an object such as the enlargement of an atom or a scaled-down version of the solar system.

Classification of Simulation Models: Simulation models may be further classified as being static or dynamic, deterministic or stochastic, and discrete or continuous.

Static Simulation Model: A static simulation model, sometimes called a Monte Carlo simulation, represents a system at a particular point in time.

Dynamic Simulation Model: Dynamic simulation models represent systems as they change over time. The simulation of a bank from 9:00 A.M. to 4:00 P.M. is an example of a dynamic simulation.

Deterministic Simulation Models: Simulation models that contain no random variables are classified as deterministic. Deterministic models have a known set of inputs, that will result in a unique set of outputs. Deterministic arrivals would occur at a dentist's office if all patients arrived at their scheduled appointment times.

Stochastic Simulation Model: A stochastic simulation model has one or more random variables as inputs. Random inputs lead to random outputs. Since the outputs are random, they can be considered only as estimates of the true characteristics of a model. The simulation of a bank would usually involve random interarrival times and random service times. Thus, in a stochastic simulation, the output measures—the average number of people waiting, the average waiting time of a customer—must be treated as statistical estimates of the true characteristics of the system.

Discrete and continuous Models: Discrete and continuous models are defined analogously to discrete and continuous systems discussed earlier. However, a discrete simulation model is not always used to model a discrete system, nor is a continuous simulation model always used to model a continuous system.

Tanks and pipes might be modeled discretely, even though we know that fluid flow is continuous. In addition, simulation models may be mixed, both discrete and continuous. The choice of whether to use a discrete or continuous (or both discrete and continuous) simulation model is a function of the characteristics of the system and the objective of the study. Thus, a communication channel could be modeled discretely if the characteristics and movement of each message were deemed important.

Conversely, if the flow of messages in aggregate over the channel were of importance, modeling the system via continuous simulation could be more appropriate. The models emphasized in this course are dynamic, stochastic, and discrete.

Discrete-event system simulation: It is the modeling of systems in which the state variable changes only at a discrete set of points in time. The simulation models are analyzed by numerical rather than analytical methods.

- **Analytical methods:** These methods employ the deductive reasoning of mathematics to "solve" the model. For example, differential calculus can be used to compute the minimum-cost policy for some inventory models.
- **Numerical methods:** These methods employ computational procedures to "solve" mathematical models. In the case of simulation models, which employ numerical methods, models are "run" rather than being "solved"—that is, an artificial history of the system is generated from the model assumptions, and observations are collected to be analyzed and to estimate the true system performance measures.

Real-world simulation models are rather large, and the amount of data stored and manipulated is vast, so these "runs" are usually conducted with the aid of a computer. In summary, in discrete-event system simulation, the models of interest are analyzed numerically, usually with the aid of a computer.

STEPS IN A SIMULATION STUDY: **Figure 3** shows a set of steps to guide a model builder in a thorough and sound simulation study. The steps in a simulation study are as follows:

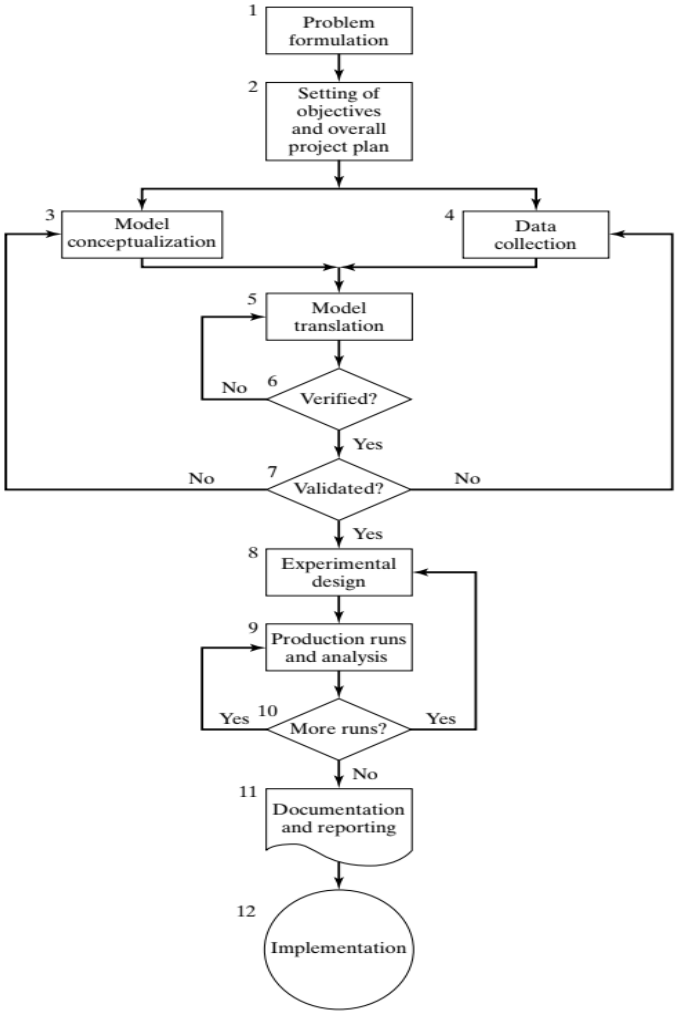


Figure 3 Steps in a simulation study.

1. Problem formulation: Every study should begin with a statement of the problem. If the statement is provided by the policymakers or those that have the problem, the analyst must ensure that the problem being described is clearly understood. If a problem statement is being developed by the analyst, the policymakers must understand and agree with the formulation.

Although not shown in Figure 3, there are occasions where the problem must be reformulated as the study progresses. In many instances, policymakers and analysts are aware that there is a problem long before the nature of the problem is known.

2. Setting of objectives and overall project plan: The objectives indicate the questions to be answered by simulation. At this point, a determination should be made concerning whether simulation is the appropriate methodology for the problem as formulated and the objectives as stated.

Assuming that it is decided that simulation is appropriate, the overall project plan should include a statement of the alternative systems to be considered and of a method for evaluating the effectiveness of these alternatives. It should also include the plans for the study in terms of the number of people involved, the cost of the study, and the number of days required to accomplish each phase of the work, along with the results expected at the end of each stage.

STEPS IN A SIMULATION STUDY: **Figure 3** shows a set of steps to guide a model builder in a thorough and sound simulation study. The steps in a simulation study are as follows:

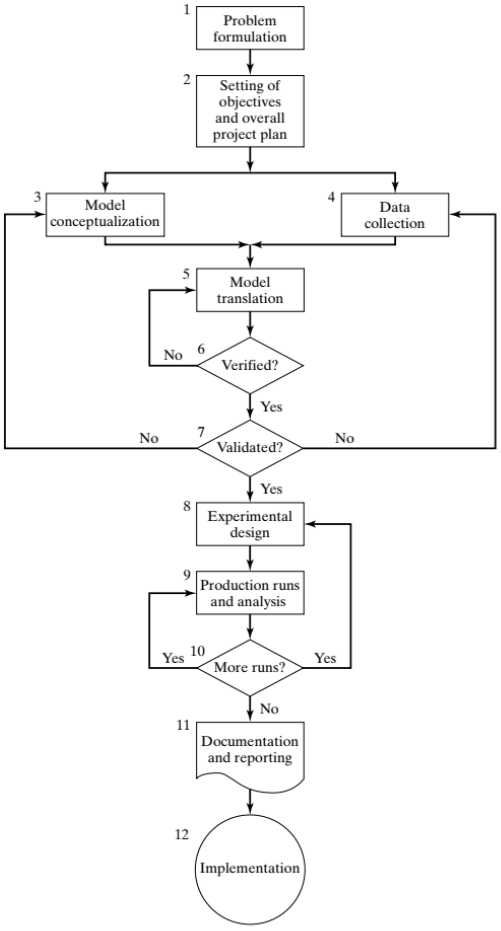


Figure 3 Steps in a simulation study.

1. Problem formulation: Every study should begin with a statement of the problem. If the statement is provided by the policymakers or those that have the problem, the analyst must ensure that the problem being described is clearly understood. If a problem statement is being developed by the analyst, the policymakers must understand and agree with the formulation.

Although not shown in Figure 3, there are occasions where the problem must be reformulated as the study progresses. In many instances, policymakers and analysts are aware that there is a problem long before the nature of the problem is known.

2. Setting of objectives and overall project plan: The objectives indicate the questions to be answered by simulation. At this point, a determination should be made concerning whether simulation is the appropriate methodology for the problem as formulated and the objectives as stated.

Assuming that it is decided that simulation is appropriate, the overall project plan should include a statement of the alternative systems to be considered and of a method for evaluating the effectiveness of these alternatives. It should also include the plans for the study in terms of the number of people involved, the cost of the study, and the number of days required to accomplish each phase of the work, along with the results expected at the end of each stage.

STEPS IN A SIMULATION STUDY

3. Model conceptualization: The construction of a model of a system is probably as much art as science. Pritsker [1998] provides a lengthy discussion of this step. "Although it is not possible to provide a set of instructions that will lead to building successful and appropriate models in every instance, there are some general guidelines that can be followed" [Morris, 1967]. The art of modeling is enhanced by an ability to abstract the essential features of a problem, to select and modify basic assumptions that characterize the system, and then to enrich and elaborate the model until useful approximation results. Thus, it is best to start with a simple model and build toward greater complexity. However, the model complexity need not exceed that required to accomplish the purposes for which the model is intended. Violation of this principle will only add to model-building and computer expenses. It is not necessary to have a one-to-one mapping between the model and the real system. Only the essence of the real system is needed.

Involving the Model User in Conceptualization: It is advisable to involve the model user in model conceptualization. Involving the model user will both enhance the quality of the resulting model and increase the confidence of the model user in the application of the model.

STEPS IN A SIMULATION STUDY

3. Model conceptualization: The construction of a model of a system is probably as much art as science. Pritsker [1998] provides a lengthy discussion of this step. "Although it is not possible to provide a set of instructions that will lead to building successful and appropriate models in every instance, there are some general guidelines that can be followed" [Morris, 1967]. The art of modeling is enhanced by an ability to abstract the essential features of a problem, to select and modify basic assumptions that characterize the system, and then to enrich and elaborate the model until useful approximation results. Thus, it is best to start with a simple model and build toward greater complexity. However, the model complexity need not exceed that required to accomplish the purposes for which the model is intended. Violation of this principle will only add to model-building and computer expenses. It is not necessary to have a one-to-one mapping between the model and the real system. Only the essence of the real system is needed.

Involving the Model User in Conceptualization: It is advisable to involve the model user in model conceptualization. Involving the model user will both enhance the quality of the resulting model and increase the confidence of the model user in the application of the model.

6. Verified? Verification pertains to the computer program that has been prepared for the simulation model. Is the computer program performing properly? With complex models, it is difficult, if not impossible; to translate a model successfully in its entirety without a good deal of debugging; if the input parameters and logical structure of the model are correctly represented in the computer, verification has been completed. For the most part, common sense is used in completing this step.

7. Validated? Validation usually is achieved through the calibration of the model, an iterative process of comparing the model against actual system behavior and using the discrepancies between the two, and the insights gained, to improve the model. This process is repeated until model accuracy is judged acceptable. In the previously mentioned example of a bank, data were collected concerning the length of the waiting lines under current conditions. Does the simulation model replicate this system measure? This is one means of validation.

8. Experimental design: The alternatives that are to be simulated must be determined. Often, the decision concerning which alternatives to simulate will be a function of runs that have been completed and analyzed. For each system design that is simulated, decisions need to be made concerning the length of the initialization period, the length of simulation runs, and the number of replications to be made of each run. (Sanchez [2007] discusses this topic extensively.)

9. Production runs and analysis: Production runs and their subsequent analysis, are used to estimate measures of performance for the system designs that are being simulated. (Software to aid in this step, include AutoStat (in AutoMod), OptQuest (in several pieces of simulation software), and SimRunner (in ProModel).

10. More runs? Given the analysis of runs that have been completed, the analyst determines whether additional runs are needed and what design those additional experiments should follow.

11. Documentation and reporting: There are two types of documentation: program and progress.

Program documentation: It is necessary for numerous reasons. If the program is going to be used again by the same or different analysts, it could be necessary to understand how the program operates. This will create confidence in the program, so that model users and policymakers can make decisions based on the analysis. Also, if the program is to be modified by the same or a different analyst, this step can be greatly facilitated by adequate documentation. One experience with an inadequately documented program is usually enough to convince an analyst of the necessity of this important step. Another reason for documenting a program is so that model users can change parameters at will to learn the relationships between input parameters and output measures of performance or to discover the input parameters that “optimize” some output measure of performance.

Musselman [1998] discusses progress reports that provide the important, written history of a simulation project.

Logs and Reports: Project reports give a chronology of work done and decisions made. This can prove to be of great value in keeping the project on course.

Musselman suggests frequent reports (monthly, at least) so that even those not involved in the day-to-day operation can be kept abreast. The awareness of these others can often enhance the successful completion of the project by surfacing misunderstandings early when the problem can be solved easily. Musselman also suggests maintaining a project log to provide a comprehensive record of accomplishments, change requests, key decisions, and other items of importance.

On the reporting side, Musselman suggests frequent deliverables. These may or may not be the results of major accomplishments. His maxim is that "it is better to work with many intermediate milestones than with one absolute deadline." Possibilities before the final report include a model specification, prototype demonstrations, animations, training results, intermediate analyses, program documentation, progress reports, and presentations. He suggests that these deliverables should be timed judiciously over the life of the project.

The results of all the analyses should be reported clearly and concisely in a final report. This will allow the model users (now the decision-makers) to review the final formulation, the alternative systems that were addressed, the criteria by which the alternatives were compared, the results of the experiments, and the recommended solution(s) to the problem. Furthermore, if decisions have to be justified at a higher level, the final report should provide a vehicle of certification for the model user/decision-maker and add to the credibility of the model and the model-building process.

12. Implementation: The success of the implementation phase depends on how well the previous eleven steps have been performed. It is also contingent upon how thoroughly the analyst has involved the ultimate model user during the entire simulation process. If the model user has been involved during the entire model-building process and if the model user understands the nature of the model and its outputs, the likelihood of a vigorous implementation is enhanced [Pritsker, 1995].

Conversely, if the model and its underlying assumptions have not been properly communicated, implementation will probably suffer, regardless of the simulation model's validity.

Simulation-model building process in Phases: The simulation-model building process shown in Figure 3 can be broken down into four phases.

Phase 1: The first phase, consisting of steps 1 (Problem formulation) and 2 (Setting of objective and overall design), is a period of discovery or orientation. The initial statement of the problem is usually quite "fuzzy," the initial objectives will usually have to be reset, and the original project plan will usually have to be fine-tuned. These recalibrations and clarifications could occur in this phase or perhaps will occur after or during another phase (i.e., the analyst might have to restart the process).

Phase 2: The second phase is related to model building and data collection and includes steps 3 (Model conceptualization), 4 (Data collection), 5 (Model translation), 6 (Verification), and 7 (Validation). A continuing interplay is required among the steps. Exclusion of the model user during this phase can

have dire implications at the time of implementation.

Phase 3: The third phase concerns the running of the model. It involves steps 8 (Experimental design), 9 (Production runs and analysis), and 10 (More runs). This phase must have a comprehensively conceived plan for experimenting with the simulation model. A discrete-event stochastic simulation is, in fact, a statistical experiment. The output variables are estimates that contain random error, and therefore a proper statistical analysis is required. Such a philosophy is in contrast to that of the analyst who makes a single run and draws an inference from that single data point.

Phase 4: The fourth phase, implementation, involves steps 11 (Documentation and reporting) and 12 (Implementation). Successful implementation depends on the continual involvement of the model user and the successful completion of every step in the process. Perhaps the most crucial point in the entire process is Step 7 (Validation) because an invalid model is going to lead to erroneous results, which, if implemented, could be dangerous, costly, or both

Simulation Case Studies

- In this module, we shall see an overview of various case studies

Variety of Examples

- We shall be taking a look at a couple of examples.
- The variety will include hands-on as well as spreadsheet examples.
- Key objective: Take a look at Discrete Event Simulation and also descriptive statistics for the prediction of system performance

General Steps

- Every simulation that we observe will be based on 3 key steps
- The idea is to examine the examples using a methodological approach

First Step

- Determine the characteristics of each of the inputs to the simulation.
- Quite often, these are modeled as probability distributions, either continuous or discrete.

Second Step

- Construct a simulation table.
- Each simulation table is different, for each is developed for the problem at hand.

Example: Simulation Table

- An example of a simulation table is shown in Table.
- In this example, there are p inputs, xij,j= I, 2, .. . , p,
- One response yi for each of repetitions (or, trials)
- i =1, 2, . . . , n.
- Initialize the table by filling in the data for repetition 1

Table

Repetitions	Inputs						Response
	x_{i1}	x_{i2}	...	x_{ij}	...	x_{ip}	y_i
1							
2							
3							
.							
.							
.							
n							

Step 3

- For each repetition, i, generate a value for each of the p inputs and evaluate the function
- Calculate a value of the response yi.
- The input values may be computed by sampling values from the distributions chosen in step 1
- A response typically depends on the inputs and one or more previous responses.
- Also, if the program is to be modified by the same or a different analyst, this step can be greatly facilitated by adequate documentation.
- One experience with an inadequately documented program is usually enough to convince an analyst of the necessity of this important step.

Overview of Examples

- Queuing
- Inventory
- Reliability
- Network analysis.
- The two queuing examples provide
  - single-server
  - two-server system

First Example Overview

- The first of the inventory examples involves a problem that has a closed-form solution;

- What it means is that the simulation solution can be compared to the mathematical solution.

Second Inventory Example

- The second inventory example pertains to the classic order-level model.

More examples

- Next, there is an example that introduces the concept of random normal numbers
- Also, there is a model for the simulation of lead-time demand.
- The examples conclude with the analysis of a network.

Summary

- In this module, we have examined how simulation examples give us a clear picture of how to do simulations
- We also took a look at the type of examples that we shall be examining in our next module

Basics of Spreadsheet Simulation

- In this module, we shall see an overview of spreadsheet simulation
- We shall also take a look at simulating randomness

How to Simulate Randomness

- Almost every simulation we encounter needs at least one random variable
- E.g. Coin Toss (Head vs. Tail)

Another Example: Queue

- **Arrival and departure rate**

Taking a Step Back: What is Random?

- In the real world, the number of factors involved in any event is so numerous that it can be very difficult to predict such events.
- In a computer, this can be very hard to generate.
- Hence the concept of "pseudorandomness"

General Steps

- Generate a number between 0 and 1 – referred to as the "random number"
- "Random variable" is any randomly generated quantity with a specified statistical distribution.
- A method of generating a sequence of random numbers is called a Random Number Generator (RNG)

Important Statistical Properties

- The number should be uniformly distributed between 0 and 1
- Subsequent numbers should be statistically independent of all previous numbers

VBA in MS Excel

- We use the RAND formula = RAND()
- More complex formulas can be used e.g. IF(RAND() <= 0.5, 0.1)
- Here IF checks the first condition and if true, returns the second one
- Another option is RANDBETWEEN()

- Rnd01
- Discrete Uniform(min, max)
- DiscreteEmp(rCumProb, rValues)
- Uniform(low, high)
- NORMSINV(Rnd01())
- For manual simulation, we can use coin toss or dice (Impractical for anything large, however)

Summary

- In this module, we have examined the basics of simulation in a spreadsheet

How can we simulate a coin toss in a spreadsheet model? The solution shows how to use a uniform random number generator in a spreadsheet simulation.

Example 1: Coin Tossing

We want to simulate a sequence of 10 coin tosses. The coin is a "fair" coin meaning that the chance of getting ahead is the same as getting a tail. Each has a probability of 0.5 of occurring. You should run the simulation multiple times and compare the results with your expectations from real-life coin tossing. This is an example of a Monte Carlo simulation; no events or clock times are being tracked.

The spreadsheet solution, shown in Table 1, in fact, consists of four solutions, the first two using the RAND() random number generator, and the last two using the supplied VBA function Rnd01(). It is left as an exercise to devise a justification for the statistical equivalence of the 4 solutions, even though on any click of the button 'Generate New Trial', the 4 sets of results are different.

The spreadsheet logic for coin tossing is simple. The second solution (Column E) is:

=IF(RAND()<=0.5,"H","T")

while the fourth solution (Column I) is:

=IF(Rnd01()<=0.5,"H","T")

Each call to RAND() or Rnd01() produces a new random number. The 'One Trial' worksheet records the frequency for heads versus tails in the 10 tosses and graphs the results in a histogram.

Table 1 Simulation Table for Coin Tossing

	B	C	D	E	F	G	H	I
11		Solution #1A		Solution #1B		Solution #2A		Solution #2B
12								
13			Solution	Solution			Solution	Solution
14	Toss	RAND()	using RN from ColC	using RAND()	Toss	Rnd01()	using RN from ColG	using Rnd01()
15	1	0.7317089	T	T	1	0.9871114	T	T
16	2	0.8285837	T	T	2	0.0225598	H	H
17	3	0.0701168	H	T	3	0.0008356	H	T
18	4	0.2147295	H	H	4	0.2127686	H	T
19	5	0.8613179	T	H	5	0.8586159	T	H
20	6	0.4159098	H	H	6	0.5503362	T	T
21	7	0.7492317	T	H	7	0.5243730	T	H
22	8	0.9671661	T	T	8	0.8884351	T	H
23	9	0.4823089	H	T	9	0.2111118	H	T
24	10	0.6738897	T	H	10	0.7680427	T	H

The first and third solutions are two steps, an approach that is useful when a generated random number is used more than once. Columns C and G contain the generated random number, and columns D and H the result ("H" or "T").

How to Simulate a Random Service Time

The following example shows how to generate random samples from an arbitrary discrete distribution.

The example is simple, having just 3 possible values of service time. The key principle, namely, the transformation from a random number to a service time of a prescribed probability, easily generalizes to any discrete distribution with a finite number of possible values.

Example 2: Random Service Times

An automated telephone information service spends either 3, 6, or 10 minutes with each caller. The proportion of calls for each service length is 30%, 45%, and 25%, respectively. We want to simulate these service times in a spreadsheet. Our actual purpose is to learn how to generate random samples
from a discrete distribution, in preparation for the queuing, inventory, and other examples to follow.
This also is an example of a Monte Carlo simulation.

Table 2 shows the input specification and a portion of the simulation table (the first 4 callers), taken from the spreadsheet model.

Table 2 Input Specification and Simulation Table for Service Times

	A	B	C	D
4		**Service Time**	**Probability**	**Cumulative Probability**
5				
6				
7		3	0.30	0.30
8		6	0.45	0.75
9		10	0.25	1.00
10				
11		**Number of Callers=**		25
12		**Simulation Table**		
13		**Step**	**Activity**	
14				
15		**Caller**	**Service Time**	
16		1	6	
17		2	6	
18		3	10	
19		4	6	

The input specification defines the probabilities for the service times, and also includes the cumulative probabilities. Cumulative probabilities always increase to 1.0; as we shall see, the generation method uses the cumulative

probabilities. The method always starts with a random number and ends with the desired value, in this case, a random service time with the specified distribution. The simulation table in the spreadsheet shows the resulting frequency distribution (or histogram) of the generated service times for 25 callers.

How to Simulate a Random Arrival Time

In this section, our goal is to learn how to generate random interarrival times (times between successive arrivals) and compute arrival times of customers or other entities to a system. The interarrival times, as well as the service times in Example 2, are examples of simulation-generated time durations, called ***activity times*** or ***activities***. These are distinguished from ***arrival times***, which are times on a simulation clock representing the time of arrival to a facility.

A simulation clock is a key component of every dynamic discrete-event simulation. In all of the Excel spreadsheet solutions, the clock times are labeled just above the appropriate column titles of the simulation table, so that you can easily distinguish between the activity times (service or interarrival times, for example) and the computed clock times. In all cases, a clock time represents the time of occurrence of an event, such as arrival, a service beginning, or a service completion in a queueing model.

Example 3: Random Arrival Times

Telephone calls to the telephone information service, where service times are defined in Example 2, occur at random times defined by a discrete distribution for which the interarrival times have values 1, 2, 3, or 4 minutes, all with equal probability. Our purpose here is to show how to generate both interarrival times and arrival times.

Since this example has one event, namely, the arrival event, it is our first example of a dynamic, event-based model (despite its simplicity).

Dynamic simply means time-based, with system state changing over time; dynamic, event-based means that the model tracks the progression of the event occurrences over time.

The interarrival times can be generated randomly in the same manner as done in Example 2 for service times. The spreadsheets, however, supply a second VBA function to simplify random generation from a discrete uniform distribution. ***Uniform*** means that all the values have equal probabilities; in this example, all the interarrival times may have one of the values 1, 2, 3, or 4 minutes, each occurring with a probability of 0.25. For such discrete uniform distributions, the spreadsheets supply the VBA function, DiscreteUniform(), which takes two arguments: either the cell containing the minimum value followed by the cell containing the maximum value of the desired range of integers, or the actual low and high values, as in:

= DiscreteUniform(G5,G6)
or
= DiscreteUniform(1,4)

The DiscreteUniform(low, high) function is implemented by a single line of VBA code, as follows:
DiscreteUniform = low + Int((high – low + 1)*Rnd())
which becomes, for this example:
DiscreteUniform = 1 + Int(4 * Rnd())

where Rnd() generates a random number between 0 and 1 (in VBA, it is always less than one but may equal zero), and Int() is a VBA function that truncates (rounds down) its argument. (Note that 4 * Rnd() generates real values between 0.0 and 3.99999, so truncating and then adding 1 yield the desired range 1 to 4. Why do we expect equal probabilities?)

Table 3 Input Specification and Simulation Table for Arrival Times.

	A	B	C	D
4		**Interarrival Times (minutes)**		
5				
6		**Minimum**	1	
7		**Maximum**	4	
8				
9		**Number of Callers=**		25
10		**Simulation Table**		
11		**Step**	**Activity**	**Clock**
12			**Interarrival**	
13		**Caller**	**Time**	**Arrival Time**
14		1		0
15		2	2	2
16		3	1	3
17		4	3	6

Table 3 shows the input specification for interarrival times and a portion of the simulation table, taken from the spreadsheet model in "Example2.3ArrivalTimes.xls". As shown in Step 1 (row 14),
we assume that the first arrival occurs at simulation time 0. (The assumed arrival time for the first customer is model-dependent; it could be any other specified time, constant or random.) The interarrival times are randomly and independently generated using the DiscreteUniform(1, 4)function. The arrival times are computed by a simple formula, by adding the current customer's interarrival time to the previous customer's arrival time. (You should verify the spreadsheet formula for arrival times.)

Example 3 is the first that contains an event, namely, the arrival event, and the event time associated with that event, namely, the CLOCK time for the time of arrival. Note that Table 3 distinguishes the activity times from the CLOCK times.

The simulation table provides the structure, and the steps outlined below provide a set of general guidelines for carrying out a spreadsheet simulation. We have seen several examples of a simulation table, with Table 3 in Example 3 being the best so far—since it contains a clock time for the arrival the event, making it one of the simplest possible dynamic, event-based simulations.

First, the model developer must define the model inputs, the system states, and the model outputs.

Inputs: Inputs are the exogenous variables that are (usually) defined independently of other system characteristics; examples include the probability of a head-on a coin toss, the service time, and the inter-arrival time distributions in a queuing system, or the demand distribution in an inventory model. These input specifications are used to generate random activity times and other random variables. Other inputs, such as policy or decision parameters, may be constants.

Outputs: Outputs are used to compute measures of system performance (also known as ***model responses***, or simple ***responses***).

For example, an output could be an individual customer's waiting time in a queue, or the cost of an individual transaction in an inventory system.

Associated Responses: The associated responses might be average waiting time in a queue or the average cost per unit time in an inventory system, respectively.

Simulation Table Design: Each simulation table is custom designed for the problem at hand. Each column in the table is one of the following types:
1. An activity time associated with a model input;
2. Any other random variable defined by a model input;
3. A system state;
4. An event, or the clock time of an event;
5. A model output;
6. Sometimes, a model response.
For the spreadsheet simulations here, the above-listed items are sufficient.

Model Responses: In general, model responses, also known as ***measures of system performance,*** are computed outside the simulation table after the simulation has been completed. For example, in a waiting line model, the delay in a queue of each customer is a ***model output***, while the average delay overall customers is a ***model response***. The model output is tracked in the simulation table; the response is computed afterward.

Activity Time: An activity time is a time defined by a constant or a statistical distribution, such as a service time, or an interarrival time. An activity is a duration of time, in contrast to event times, which are clock times.
System States: System states include, for example, server status (busy or idle) in a queueing model and inventory levels in an inventory model. In general, the set of system states includes any information needed for the simulation to proceed through time (or from one step to the next), as well as any quantity of information that makes the simulation logic simpler or more transparent and thus easier to understand. The greatest simplicity possible, as long as the model

logic is correct, assists in ease in understanding that can be of tremendous value, especially when trying to verify that a model's logic is indeed correct.

Table 4 A Generic Simulation Table

	Activities and System States						
							Outputs
Step	x_{i1}	x_{i2}	$\cdots$	x_{ij}	$\cdots$	x_{ip}	y_i
1							
2							
3							
.							
.							
.							
n							

Each row in the simulation table represents a step in the simulation and is usually associated with the occurrence of one or more events, or the progress of one entity (such as a customer) through the system. The simulation table is designed so that a given step depends solely on model inputs and/or one or more previous steps or previously computed values in the current step.

In general, a modeler develops and runs a spreadsheet or manual simulation by following these guidelines:

This module develops a common framework for the modeling of complex systems by using discrete event simulation. It covers the basic building blocks of all discrete-event simulation models: entities and attributes, activities, and events.

Discrete-Event Simulation: In discrete-event simulation:

System: A system is modeled in terms of its state at each point in time; of the entities that pass through the system and;

Entities: The entities that represent system resources; and of the activities and events that cause the system state to change.

Discrete-event models are appropriate for those systems for which changes in system state occur only at discrete points in time.

Concepts In Discrete-event Simulation

Here we are exclusively dealing with dynamic, stochastic systems (i.e., involving time and containing random elements) that change discretely. This module expands on these concepts and proposes a framework for the development of a discrete-event model of a system. The major concepts are briefly defined and then illustrated by examples:

System A collection of entities (e.g., people and machines) that interact together over time to accomplish one or more goals.

Model An abstract representation of a system, usually containing structural, logical, or mathematical relationships that describe a system in terms of state, entities and their attributes, sets, processes, events, activities, and delays.

System state A collection of variables that contain all the information necessary to describe the system at any time.

Entity Any object or component in the system that requires explicit representation in the model (e.g., a server, a customer, a machine).

Attributes The properties of a given entity (e.g., the priority of a waiting customer, the routing of a job through a job shop).

List A collection of (permanently or temporarily) associated entities, ordered in some logical fashion (such as all customers currently in a waiting line, ordered by "first come, first served," or by priority).

Event An instantaneous occurrence that changes the state of a system (such as an arrival of a new customer).

Event notice A record of an event to occur at the current or some future time, along with any associated data necessary to execute the event; at a minimum, the record includes the event type and the event time.

Event list A list of event notices for future events, ordered by time of occurrence; also known as the future event list (FEL).

Concepts In Discrete-event Simulation

Activity A duration of time of specified length (e.g., a service time or interarrival time), which is known when it begins (although it may be defined in terms of a statistical distribution).

Delay A duration of time of unspecified indefinite length, which is not known until it ends (e.g., a customer's delay in a last-in-first-out waiting line which, when it begins, depends on future arrivals).

Clock A variable representing simulated time, called CLOCK in the examples to follow.

Different simulation packages use different terminology for the same or similar concepts—for example, **lists** are sometimes called **sets**, **queues**, or **chains**.

Sets or lists: Sets or lists are used to hold both entities and event notices.

Entities on a List: The entities on a list are always ordered by some rule, such as first-in-first-out or last-in-first-out, or are ranked by some entity attribute, such as priority or due date.

Event List: The future event list is always ranked by the event time recorded in the event notice.

Activity: An activity typically represents a service time, an interarrival time, or any other processing time whose duration has been characterized and defined by the modeler.

Duration of an Activity: An activity's duration may be specified in several ways:

a. ***Deterministic*—**for example, always exactly 5 minutes;
b. ***Statistical*—**for example, as a random draw from the set {2, 5, 7} with equal probabilities;
c. ***A function depending on system variables and/or entity attributes*—**for example, loading time for an iron ore ship as a function of the ship's allowed cargo weight and the loading rate in tons per hour.

However, it is characterized, the duration of an activity is computable from its specification at the instant it begins. Its duration is not affected by the occurrence of other events (unless, as is allowed by some simulation packages, the model contains logic to cancel or postpone an activity in progress).

Keeping Track of Activities: To keep track of activities and their expected completion time, at the simulated instant that an activity duration begins, an **event notice** is created having an event time equal to the activity's completion time.

Example: For example, if the current simulated time is CLOCK = 100 minutes and an inspection time of exactly 5 minutes is just beginning, then an event notice is created that specifies the type of event (an end-of-inspection event), and the event time (100 + 5 = 105 minutes).

Delays: In contrast to an activity, a delay's duration is not specified by the modeler ahead of time, but rather is determined by system conditions. Quite often, a delay's duration is measured and is one of the desired outputs of a model run. Typically, a delay ends when some set of logical conditions becomes true or one or more other events occur. For example, a customer's delay in a waiting line may be dependent on the number and duration of service of other customers ahead in line as well as the availability of servers and equipment.

Delay or Conditional Wait: A ***delay*** is sometimes called a ***conditional wait***, an activity an ***unconditional wait***.

Primary Event: The completion of an activity is an event, often called a ***primary event***, that is managed by placing an event notice on the FEL.

In contrast, delays are managed by placing the associated entity on another list, perhaps representing a waiting line, until when the system conditions permit the processing of the entity.

Secondary Event: The completion of a delay is sometimes called a ***conditional or secondary event***, but such events are not represented by event notices, nor do they appear on the FEL.

Dynamic Nature of the System:

The systems considered here are ***dynamic***, that is, *changing over time*. Therefore, *system state*, *entity attributes* and the *number of active entities*, the *contents of sets*, and the *activities* and *delays* currently in progress are all ***functions of time*** and are constantly changing over time. Time itself is represented by a variable called ***CLOCK***.

Example: Call Center, Revisited

Consider the Able–Baker call center system of Example from the Simulation Examples in a Spreadsheet chapter. A discrete-event model has the following components:
System state

LQ(t), the number of callers waiting to be served at time *t*;
LA(t), 0 or 1 to indicate Able as being idle or busy at time *t*;
LB(t), 0 or 1 to indicate Baker as being idle or busy at time *t*.
Entities Neither the callers nor the servers need to be explicitly represented, except in terms of the state variables, unless certain caller averages are desired (compare Examples 4 and 5).
Events
Arrival event;

Service completion by Able;
Service completion by Baker.

Example: Call Center, Revisited (Continued…)

Activities
Interarrival time, defined in Table from the Simulation Examples in a Spreadsheet chapter;
Service time by Able, defined in Table from the Simulation Examples in a Spreadsheet chapter;
Service time by Baker, defined in Table from the Simulation Examples in a Spreadsheet chapter.
Delay
A caller waits in the queue until Able or Baker becomes free

The definition of the model components provides a static description of the model. In addition, a description of the dynamic relationships and interactions between the components is also needed.

Some questions that need answers include:

1. How does each event affect system state, entity attributes, and set contents?

2. How are activities defined (i.e., deterministic, probabilistic, or some other mathematical equation)? What event marks the beginning or end of each activity? Can the activity begin regardless of the system state, or is its beginning conditioned on the system being in a certain state? (For example, a machining "activity" cannot begin unless the machine is idle, not broken, and not in maintenance.)

3. Which events trigger the beginning (and end) of each type of delay? Under what conditions does a delay begin or end?

4. What is the system state at time 0? What events should be generated at time 0 to "prime" the model—that is, to get the simulation started?

Discrete Event Simulation

A *discrete-event simulation* is the modeling over time of a system all of whose state changes occur at discrete points in time—those points when an event occurs. A discrete-event simulation (hereafter called a ***simulation***) proceeds by producing a sequence of system snapshots (or system images) that represent the evolution of the system through time.

The Event Scheduling/Time Advance Algorithm

The mechanism for advancing simulation time and guaranteeing that all events occur in correct chronological order is based on the future event list (FEL). This list contains all event notices for events that have been scheduled to occur at a future time. Scheduling a future event means that, at the instant an activity begins, its duration is computed or drawn as a sample from a statistical distribution; and that the end-activity event, together with its event time, is placed on the future event list. In the real world, most future events are not scheduled but merely happen—such as random breakdowns or random arrivals. In the model, such random events are represented by the end of some activity, which in turn is represented by a statistical distribution.

At any given time t, the FEL contains all previously scheduled future events and their associated event times (called $t_1, t_2, \ldots$ in Figure 1). The FEL is ordered by event time, meaning that the events are arranged chronologically—that is, the event times satisfy

$$t < t_1 \leq t_2 \leq t_3 \leq \cdots \leq t_n.$$

CLOCK	System state	Entities and attributes	Set 1	Set 2	...	Future event list (FEL)	Cumulative statistics and counters
t	$(x, y, z, \ldots)$					$(3, t_1)$ – Type 3 event to occur at time t_1 $(1, t_2)$ – Type 1 event to occur at time t_2	

Figure 1 Prototype system snapshot at simulation time t.

Time t is the value of CLOCK, the current value of simulated time. The event associated with time t_1 is called the imminent event; that is, it is the next event that will occur. After the system snapshot at simulation time CLOCK = t has been updated, the CLOCK is advanced to simulation time CLOCK = t_1, the imminent event notice is removed from the FEL, and the event is executed. Execution of the imminent event means that a new system snapshot for time $t1$ is created, one based on the old snapshot at time t and the nature of the imminent event. At time t_1, new future events may or might not be generated, but if any are, they are scheduled by creating event notices and putting them into their proper position on the FEL. After the new system snapshot for time $t1$ has been updated, the clock is advanced to the time of the new imminent event and that event is executed. This process repeats until the simulation is over.

The sequence of actions that a simulator (or simulation language) must perform to advance the clock and build a new system snapshot is called the ***event-scheduling/time-advance algorithm***, whose steps are listed in Figure 2 (and explained thereafter).

Old system snapshot at time t

CLOCK	System state	...	Future event list	...
t	(5, 1, 6)		$(3, t_1)$ – Type 3 event to occur at time t_1 $(1, t_2)$ – Type 1 event to occur at time t_2 $(1, t_3)$ – Type 1 event to occur at time t_3 . . . . $(2, t_n)$ – Type 2 event to occur at time t_n	

Event-scheduling/time-advance algorithm

Step 1. Remove the event notice for the imminent event (event 3, time t_1) from FEL.

Step 2. Advance CLOCK to imminent event time (i.e., advance CLOCK from t to t_1).

Step 3. Execute imminent event: update system state, change entity attributes, and set membership as needed.

Step 4. Generate future events (if necessary) and place their event notices on FEL, ranked by event time. (Example: Event 4 to occur at time t^*, where $t_2 < t^* < t_3$.)

Step 5. Update cumulative statistics and counters.

New system snapshot at time t_1

CLOCK	System state	...	Future event list	...
t_1	(5, 1, 5)		$(1, t_2)$ – Type 1 event to occur at time t_2 $(4, t^*)$ – Type 4 event to occur at time t^* $(1, t_3)$ – Type 1 event to occur at time t_3 . . . . $(2, t_n)$ – Type 2 event to occur at time t_n	

Figure 2 Advancing simulation time and updating system image.

List Processing: The length and contents of the FEL are constantly changing as the simulation progresses, and thus its efficient management in a simulation software package will have a major impact on the model's computer runtime. The management of a list is called list processing.

Cancellation Of An Event: The major list processing operations performed on an FEL are removal of the imminent event, the addition of a new event to the list, and occasionally the removal of some event (called ***cancellation of an event***). As the imminent event is usually at the top of the list, its removal is as efficient as possible.

The addition of a new event (and cancellation of an old event) requires a search of the list. The efficiency of this search depends on the logical organization of the list and on how the search is conducted. In addition to the FEL, all the sets in a model are maintained in some logical

order, and the operations of addition and removal of entities from the set also require efficient list-processing techniques.

The removal and addition of events from the FEL are illustrated in Figure 2. Event 3 with event time $t1$ represents, say, a service completion event at server 3. Since it is the imminent event at time t, it is removed from the FEL in Step 1 (Figure 2) of the event-scheduling/time-advance algorithm.

When event 4 (say, an arrival event) with event time $t*$ is generated at step 4, one possible way to determine its correct position on the FEL is to conduct a top-down search:

If $t^* < t_2$,	place event 4 at the top of the FEL.
If $t_2 \leq t^* < t_3$,	place event 4 second on the list.
If $t_3 \leq t^* < t_4$,	place event 4 third on the list.
$\vdots$	
If $t_n \leq t^*$,	place event 4 last on the list.

The least efficient way to maintain the FEL is to leave it as an unordered list (additions placed arbitrarily at the top or bottom), which would require at Step 1 of Figure 2 a complete search of the list for the imminent event before each clock advance. (The imminent event is the event on the FEL with the lowest event time).

Exogenous Events: The system snapshot at time 0 is defined by the initial conditions and the generation of the so-called exogenous events. The specified initial conditions define the system state at time 0. For example, in Figure 2, if $t = 0$, then the state (5, 1, 6) might represent the initial number of customers at three different points in the system. An exogenous event is a happening "outside the system" that impinges on the system. An important example is an arrival to a queueing system. At time 0, the first arrival event is generated and is scheduled on the FEL (meaning that its event notice is placed on the FEL). The interarrival time is an example of an activity. When the clock eventually is advanced to the time of this first arrival, a second arrival event is generated. First, an interarrival time, call it $a*$, is generated; it is added to the current time, CLOCK = t; the resulting (future) event time, $t + a* = t*$, is used to position the new arrival event notice on the FEL. This method of generating an external arrival stream is called ***bootstrapping***; it provides one example of how future events are generated in Step 4 of the event-scheduling/time-advance algorithm. Bootstrapping is illustrated in Figure 3

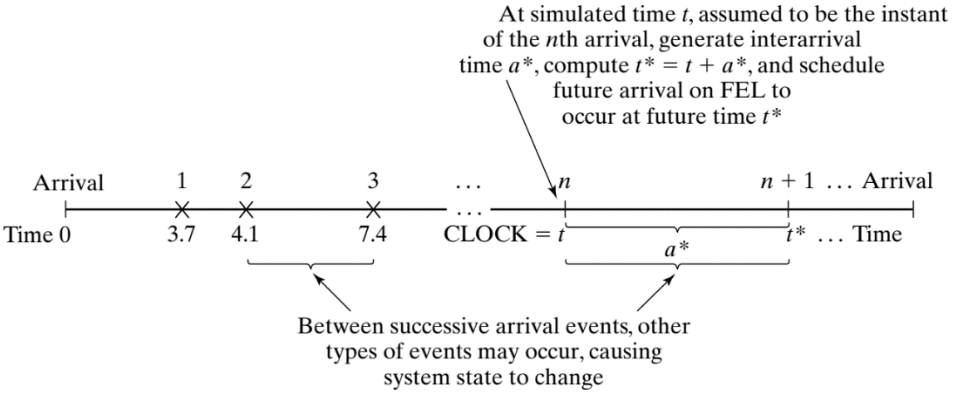


Figure 3 Generation of an external arrival stream by bootstrapping.

A second example of how future events are generated (Step 4 of Figure 2) is provided by a service completion event in a queueing simulation. When one customer completes service, at current time CLOCK = t, if the next customer is present, then a new service time, $s*$, will be generated for the next customer. The next service completion event will be scheduled to occur at future time $t* = t + s*$, by placing onto the FEL a new event notice, of type *service completion*, with event time $t*$.

Beginning service is a conditional event because its occurrence is triggered only on the condition that a customer is present and a server is free. Service completion is an example of a primary event. Note that a conditional event, such as beginning service, is triggered by a primary event occurring and by certain conditions prevailing in the system. Only primary events appear on the FEL.

A third important example is the alternate generation of runtimes and downtimes for a machine subject to breakdowns. At time 0, the first runtime will be generated, and an end-of-runtime event will be scheduled. Whenever an end-of-runtime event occurs, downtime will be generated, and an end-of-downtime event will be scheduled on the FEL. When the CLOCK is eventually advanced to the time of this end-of-downtime event, a runtime is generated, and an end-of-runtime event is scheduled on the FEL. In this way, runtimes and downtimes continually alternate throughout the simulation. A runtime and downtime are examples of activities, and the ***end of runtime*** and ***end of downtime*** are primary events.

Every simulation must have a stopping event, here called *E*, which defines how long the simulation will run. There are generally two ways to stop a simulation:
1. At time 0, schedule a stop simulation event at a specified future time T_E. Thus, before simulating, it is known that the simulation will run over the time interval [0, T_E]. Example: Simulate a job shop for T_E = 40 hours.
2. Run-length T_E is determined by the simulation itself. Generally, T_E is the time of occurrence of some specified event *E*. Examples: T_E could be the time of the 100th service completion at a certain service center. T_E could be the time of breakdown of a complex system. T_E could be the time of disengagement or total kill (whichever occurs first) in a combat simulation. T_E could be the time at which a distribution center ships the last carton in a day's orders.

In case 2, T_E is not known ahead of time. Indeed, it could be one of the statistics of primary interest to be produced by the simulation.

When using a simulation package or even when doing a manual simulation, a modeler adopts a world view or orientation for developing a model. The most prevalent world views are the ***event scheduling world view***, as discussed in the previous section, ***the process-interaction world view***, and the activity-scanning world view. Even if a particular package does not directly support one or more of the world views, understanding the different approaches could suggest alternative ways to model a given system.

When using a package that supports the process-interaction approach, a simulation analyst thinks in terms of processes.

The analyst defines the simulation model in terms of entities or objects and their life cycle as they flow through the system, demanding resources and queueing to wait for resources.

More precisely, a process is the life cycle of one entity. This life cycle consists of various events and activities. Some activities might require the use of one or more resources whose capacities are limited. These and other constraints cause processes to interact, the simplest example being an entity forced to wait in a queue (on a list) because the resource it needs is busy with another entity. The process-interaction approach is popular because it has intuitive appeal and because the simulation packages that implement it allow an analyst to describe the process flow in terms

of high-level block or network constructs, while the interaction among processes is handled automatically.

In more precise terms, a process is a time-sequenced list of events, activities, and delays, including demands for resources, that define the life cycle of one entity as it moves through a system. An example of a "customer process" is shown in Figure 4. In this figure, we see the interaction between two customer processes as customer $n + 1$ is delayed until the previous customer's "end service event" occurs. Usually, many processes are active simultaneously in a model, and the interaction among processes could be quite complex.

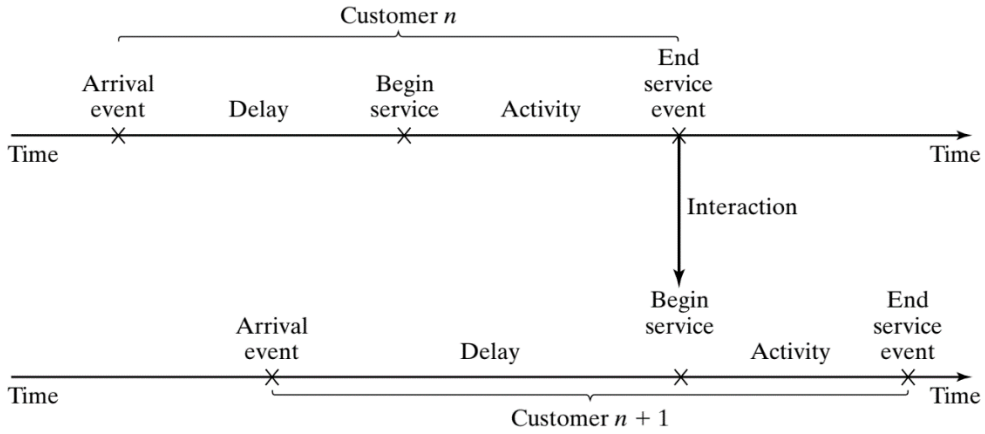


Figure 4 Two interacting customer processes in a single-server queue.

With the activity-scanning approach, a modeler concentrates on the activities of a model and those conditions, simple or complex, that allow an activity to begin. At each clock advance, the conditions for each activity are checked, and, if the conditions are true, then the corresponding activity begins. Proponents claim that the activity-scanning approach is simple in concept and leads to modular models that are more easily maintained, understood, and modified by other analysts at later times. They admit, however, that the repeated scanning to discover whether an activity can begin results in slow runtime on computers.

The pure activity-scanning approach has been modified (and made conceptually somewhat more complex) by what is called the three-phase approach, which combines some of the features of event scheduling with activity scanning to allow for variable time advance and the avoidance of scanning when it is not necessary, but keeps the main advantages of the activity-scanning approach.

In the three-phase approach, events are considered to be activities of duration zero-time units.

With this definition, activities are divided into two categories, which are called *B* and *C*.

B activities: activities bound to occur; all primary events and unconditional activities.

C activities: activities or events that are conditional upon certain conditions being true.

The B-type activities and events can be scheduled ahead of time, just as in the event-scheduling approach. This allows variable time advance. The FEL contains only B-type events. Scanning to learn whether any C-type activities can begin or C-type events occur happens only at the end of each time advance, after all, B-type events have been completed.

In summary, with the three-phase approach, the simulation proceeds with repeated execution of the 3 phases until it is completed:

Phase A Remove the imminent event from the FEL and advance the clock to its event time.

Remove from the FEL any other events that have the same event time.
Phase B Execute all B-type events that were removed from the FEL. (This could free several resources or otherwise change the system state.)
Phase C Scan the conditions that trigger each C-type activity and activate any whose conditions are met. Rescan until no additional C-type activities can begin and no events occur.

In the conducting of an event-scheduling simulation, a simulation table is used to record the successive system snapshots as time advances.

Example 3: Single-Channel Queue
Consider the grocery store with one checkout counter that is simulated in Example 5 from the Simulation Examples in a Spreadsheet chapter by an ad hoc method. The system consists of those customers in the waiting line plus the one (if any) checking out. A stopping time of 60 minutes is set for this example. The model has the following components:
System state (*LQ(t), LS(t)*), where *LQ(t)* is the number of customers in the waiting line, and *LS(t)* is the number being served (0 or 1) at time *t*.
Entities The server and customers are not explicitly modeled, except in terms of the state variables.
Events
Arrival (A);
Departure (D);
Stopping event (E), scheduled to occur at time 60.

Event notices
(A, *t*), representing an arrival event to occur at future time *t*;
(D, *t*), representing a customer departure at future time *t*;
(E, 60), representing the simulation stop event at future time 60.
Activities
Interarrival time, defined in Table 9 from the Simulation Examples in a Spreadsheet chapter;
Service time, defined in Table 10 from the Simulation Examples in a Spreadsheet chapter.
Delay Customer time spent in the waiting line.
The event notices are written as (***event type, event time***). In this model, the FEL will always contain either two or three event notices. The effect of the arrival and departure events is shown in detail in Figures 5 and 6.

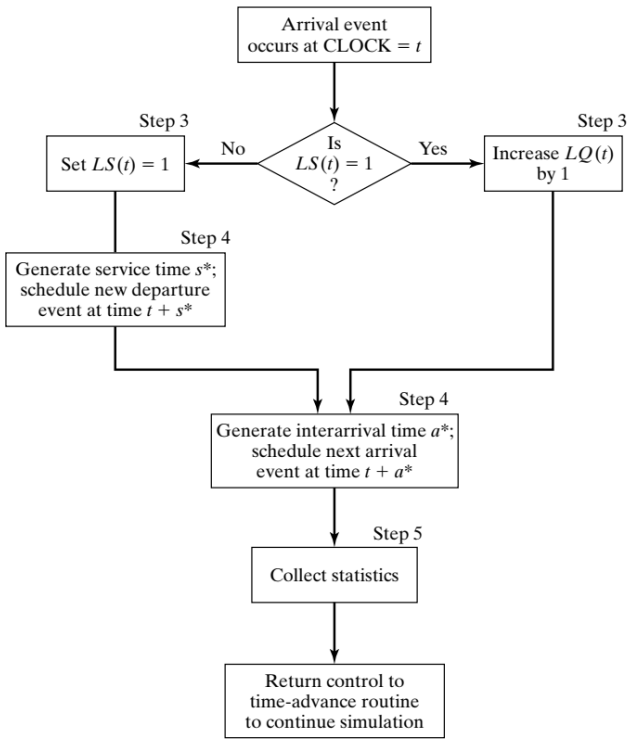


Figure 5 Execution of the arrival event.

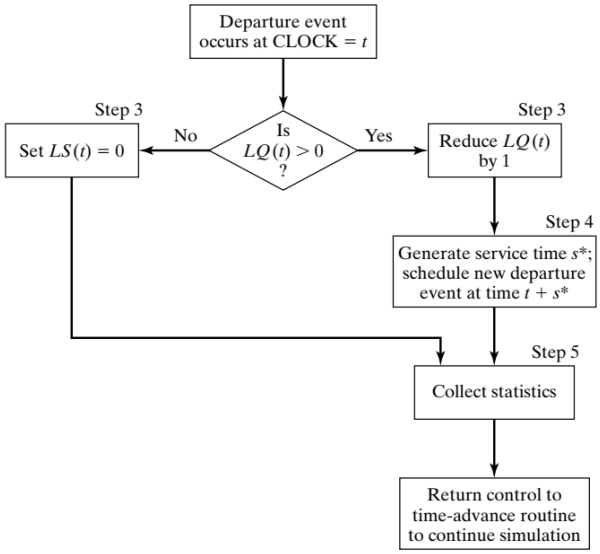


Figure 6 Execution of the departure event.

The simulation table for the checkout counter is given in Table 1. The reader should cover all system snapshots except one, starting with the first, and attempt to construct the next snapshot from the previous one and the event logic in Figures 5 and 6.

Interarrival Times	1	1	6	3	7	5	2	4	1 · · ·
Service Times	4	2	5	4	1	5	4	1	4 · · ·

Execution Of The Arrival Event

Initial conditions are that the first customer arrives at time 0 and begins service. This is reflected in ***Table 1*** by the system snapshot at time zero (CLOCK = 0), with $LQ(0) = 0$, $LS(0) = 1$, and both a departure event and arrival event on the FEL.

Also, the simulation is scheduled to stop at time 60. Only two statistics, server utilization, and maximum queue length will be collected.

Server Utilization

Server utilization is defined by total server busy time (B) divided by total time (T_E).

Total busy time, B, and maximum queue length, MQ, will be accumulated as the simulation progresses.

A column headed "Comments" is included to aid the reader ($a*$ and $s*$ are the generated interarrival and service times, respectively).

Table 1 Simulation Table for Checkout Counter

CLOCK	System state $LQ(t)$	System state $LS(t)$	Future event list	Comment	Cumulative statistics B	Cumulative statistics MQ
0	0	1	(A, 1) (D, 4) (E, 60)	First A occurs ($a^* = 1$) Schedule next A ($s^* = 4$) Schedule first D	0	0
1	1	1	(A, 2) (D, 4) (E, 60)	Second A occurs: (A, 1) ($a^* = 1$) Schedule next A (Customer delayed)	1	1
2	2	1	(D, 4) (A, 8) (E, 60)	Third A occurs: (A, 2) ($a^* = 6$) Schedule next A (Two customers delayed)	2	2
4	1	1	(D, 6) (A, 8) (E, 60)	First D occurs: (D, 4) ($s^* = 2$) Schedule next D (Customer delayed)	4	2
6	0	1	(A, 8) (D, 11) (E, 60)	Second D occurs: (D, 6) ($s^* = 5$) Schedule next D	6	2
8	1	1	(D, 11) (A, 11) (E, 60)	Fourth A occurs: (A, 8) ($a^* = 3$) Schedule next A (Customer delayed)	8	2
11	1	1	(D, 15) (A, 18) (E, 60)	Fifth A occurs: (A, 11) ($a^* = 7$) Schedule next A Third D occurs: (D, 11) ($s^* = 4$) Schedule next D (Customer delayed)	11	2
15	0	1	(D, 16) (A, 18) (E, 60)	Fourth D occurs: (D, 15) ($s^* = 1$) Schedule next D	15	2
16	0	0	(A, 18) (E, 60)	Fifth D occurs: (D, 16)	16	2
18	0	1	(D, 23) (A, 23) (E, 60)	Sixth A occurs ($a^* = 5$) Schedule next A ($s^* = 5$) Schedule next D	16	2
23	0	1	(A, 25) (D, 27) (E, 60)	Seventh A occurs: (A, 23) ($a^* = 2$) Schedule next Arrival Sixth D occurs: (D, 23)	21	2

As soon as the system snapshot at time CLOCK = 0 is complete, the simulation begins. At time 0, the imminent event is (A,1).

The CLOCK is advanced to time 1, and (A,1) is removed from the FEL.

Because $LS(t) = 1$ for $0 \leq t \leq 1$ (i.e., the server was busy for 1 minute), the cumulative busy time is increased from $B = 0$ to $B = 1$. By the event logic in ***Figure 6***, set $LS(1) = 1$ (the server becomes busy). The FEL is left with three future events, (A, 2), (D, 4), and (E, 60). The simulation CLOCK is next advanced to time 2, and an arrival event is executed. The interpretation of the remainder of **Table 1** is left as an exercise.

The simulation in **Table 1** covers the time interval [0,23]. At simulated time 23, the system empties, but the next arrival also occurs at time 23.

The server was busy for 21 of the 23-time units simulated, and the maximum queue length was two.

This simulation is, of course, too short to draw any reliable conclusions.

Note that the simulation table gives the system state at all times, not just the listed times.

For example, from time 11 to time 15, there is one customer in service and one in the waiting line.

The Dump-Truck Problem

Six dump trucks are used to haul coal from the entrance of a small mine to the railroad. Figure 7 provides a schematic of the dump-truck operation. Each truck is loaded by one of two loaders. After loading, the truck immediately moves to the scale, to be weighed as soon as possible. Both the loaders and the scale have a first-come-first-served waiting line (or queue) for trucks. Travel time from a loader to the scale is considered negligible. After being weighed, a truck begins a travel time (during which time the truck unloads) and then afterward returns to the loader queue.

The distributions of loading time, weighing time, and travel time are given in Tables 3, 4, and 5, respectively.

The purpose of the simulation is to estimate the loader and scale utilization (percentage of the busy time). The model has the following components:
System state
[*LQ(t), L(t), WQ(t), W(t)*], where
LQ(t) = number of trucks in loader queue;
L(t) = number of trucks (0, 1, or 2) being loaded
WQ(t) = number of trucks in weigh queue;
W(t) = number of trucks (0 or 1) being weighed, all at simulation time *t*.
Entities
The six dump trucks (*DT*1, . . . , *DT*6).
Event notices
(*ALQ*, *t*, *DTi*), *DTi* arrives at loader queue (*ALQ*) at time *t*;

(*EL*, *t*, *DTi*), *DTi* ends loading (*EL*) at time *t*;
(*EW*, *t*, *DTi*), *DTi* ends weighing (*EW*) at time *t*.

Lists
Loader queue, all trucks waiting to begin loading, ordered on a first-come-first-served basis;
Weigh queue, all trucks waiting to be weighed, ordered on a first-come-first-served basis.
Activities
Loading time, weighing time, and travel time.
Delays
Delay at loader queue, and delay at scale.

Table 3 Distribution of Loading Time for the Dump Trucks

Loading Time	Probability	Cumulative Probability	Random Number Interval
5	0.30	0.30	$0.0 \leq R \leq 0.3$
10	0.50	0.80	$0.3 < R \leq 0.8$
15	0.20	1.00	$0.8 < R \leq 1.0$

Table 4 Distribution of Weighing Time for the Dump Trucks

Weighing Time	Probability	Cumulative Probability	Random Number Interval
12	0.70	0.70	$0.0 \leq R \leq 0.7$
16	0.30	1.00	$0.7 < R \leq 1.0$

Table 5 Distribution of Travel Time for the Dump Trucks

Travel Time	Probability	Cumulative Probability	Random Number Interval
40	0.40	0.40	$0.0 \leq R \leq 0.4$
60	0.30	0.70	$0.4 < R \leq 0.7$
80	0.20	0.90	$0.7 < R \leq 0.9$
100	0.10	1.00	$0.9 < R \leq 1.0$

Review of Terminology and Concepts

Discrete Random Variables: Let X be a random variable. If the number of possible values of X is finite, or countably infinite, X is called a discrete random variable. The possible values of X may be listed as $x_1, x_2, \ldots$. In the finite case, the list terminates; in the countably infinite case, the list continues indefinitely.

Example 1
The number of jobs arriving each week at a job shop is observed. The random variable of interest is X, where
X = number of jobs arriving each week
The possible values of X are given by the range space of X, which is denoted by R_X.
Here $R_X = \{0, 1, 2, \ldots\}$.
Let X be a discrete random variable. With each possible outcome x_i in R_X, a number $p(x_i) = P(X = x_i)$ gives the probability that the random variable equals the value of xi.
The numbers $p(x_i)$, $i = 1, 2, \ldots$, must satisfy the following two conditions:

a. $p(x_i) \geq 0$, for all i

b. $\sum_{i=1}^{\infty} p(x_i) = 1$

The collection of pairs $(x_i, p(x_i))$, $i = 1, 2, \ldots$ is called the probability distribution of X, and $p(x_i)$ is called the probability mass function (pmf) of X.

Example 2

Consider the experiment of tossing a single die. Define X as the number of spots on the up face of the die after a toss. Then $R_X = \{1, 2, 3, 4, 5, 6\}$. Assume the die is loaded so that the probability that a given face lands up is proportional to the number of spots showing. The discrete probability distribution for this random experiment is given by:

x_i	1	2	3	4	5	6
$p(x_i)$	1/21	2/21	3/21	4/21	5/21	6/21

The conditions stated earlier are satisfied—that is, $p(x_i) \geq 0$ for $i = 1, 2, \ldots, 6$ and $\sum_{i=1}^{\infty} p(x_i) = 1/21 + \cdots + 6/21 = 1$. The distribution is shown graphically in Figure 1.

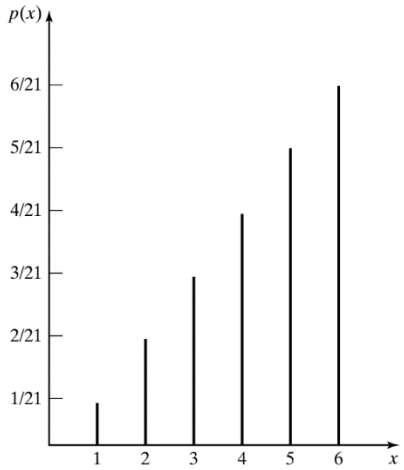


Figure 1 Probability mass function for loaded-die example.

Review of Terminology and Concepts

Continuous Random Variables

If the range space R_X of the random variable X is an interval or a collection of intervals, X is called a continuous random variable. For a continuous random variable X, the probability that X lies in the interval $[a, b]$ is given by:

$$P(a \leq X \leq b) = \int_a^b f(x)\,dx \qquad (1)$$

The function $f(x)$ is called the probability density function (pdf) of the random variable X. The pdf satisfies the following conditions:

a. $f(x) \geq 0$ for all x in R_X
b. $\int_{R_X} f(x)dx = 1$
c. $f(x) = 0$ if x is not in R_X

As a result of Equation (1), for any specified value x_0, $P(X = x_0) = 0$, because

$$\int_{x_0}^{x_0} f(x)\,dx = 0$$

$P(X = x_0) = 0$ also means that the following equations hold:

$$P(a \leq X \leq b) = P(a < X \leq b) = P(a \leq X < b) = P(a < X < b) \qquad (2)$$

The graphical interpretation of Equation (1) is shown in Figure 2. The shaded area represents the probability that X lies in the interval $[a, b]$.

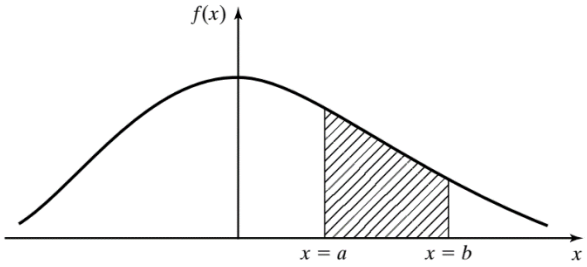


Figure 2 Graphical interpretation of $P(a < X < b)$.

Example 3

The life of a device used to inspect cracks in aircraft wings is given by X, a continuous random variable assuming all values in the range $x \geq 0$. The pdf of the device's lifetime, in years, is as follows:

$$f(x) = \begin{cases} \frac{1}{2}e^{-x/2}, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

This pdf is shown graphically in Figure 3. The random variable X is said to have an exponential distribution with a mean of 2 years.
The probability that the life of the device is between 2 and 3 years is calculated as:

$$\begin{aligned} P(2 \leq X \leq 3) &= \frac{1}{2}\int_2^3 e^{-x/2}\,dx \\ &= -e^{-3/2} + e^{-1} \\ &= -0.223 + 0.368 \\ &= 0.145 \end{aligned}$$

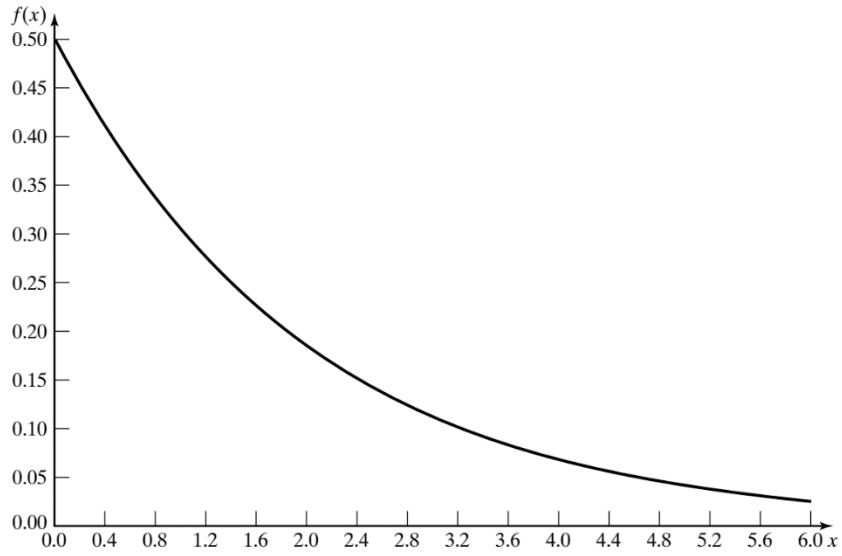


Figure 3 pdf for inspection-device life.

Review of Terminology and Concepts

Cumulative distribution function

The cumulative distribution function (cdf), denoted by *F(x)*, measures the probability that the random variable *X* assumes a value less than or equal to *x*, that is, $F(x) = P(X \leq x)$.
If *X* is discrete, then

$$F(x) = \sum_{\substack{\text{all} \\ x_i \leq x}} p(x_i) \qquad (3)$$

If *X* is continuous, then

$$F(x) = \int_{-\infty}^{x} f(t)\,dt \qquad (4)$$

Some properties of the *cdf* are listed here:

a. F is a nondecreasing function. If $a < b$, then $F(a) \leq F(b)$.
b. $\lim_{x \to \infty} F(x) = 1$
c. $\lim_{x \to -\infty} F(x) = 0$

All probability questions about X can be answered in terms of the cdf. For example,

$$P(a < X \leq b) = F(b) - F(a) \quad \text{for all } a < b \qquad (5)$$

For continuous distributions, not only does Equation (5) holds but also the probabilities in Equation (2) are equal to *F(b) - F(a)*.

Example 4

The die-tossing experiment described in Example 2 (Module 44) has a cdf given as follows:

x	$(-\infty, 1)$	$[1, 2)$	$[2, 3)$	$[3, 4)$	$[4, 5)$	$[5, 6)$	$[6, \infty)$
$F(x)$	0	1/21	3/21	6/21	10/21	15/21	21/21

where $[a, b) = \{a \leq x < b\}$. The ***cdf*** for this example is shown graphically in Figure 4.

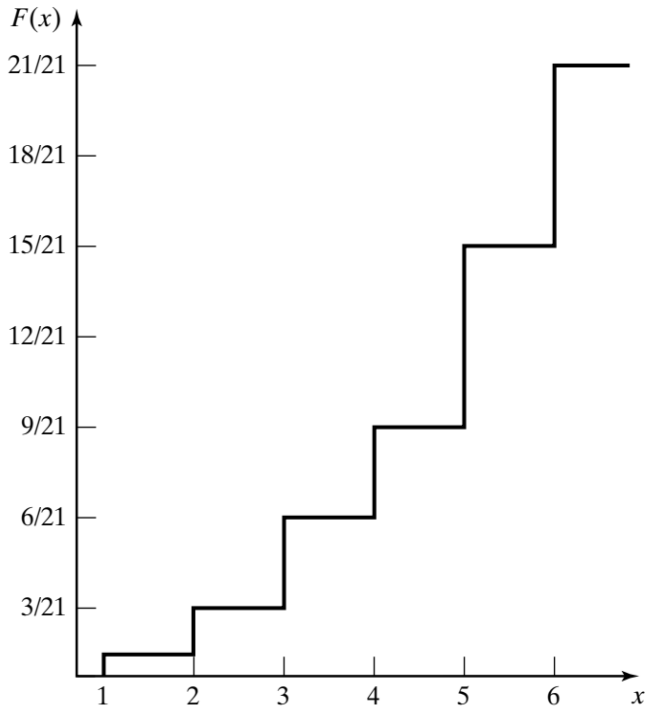


Figure 4 cdf for loaded-die example.

If X is a discrete random variable with possible values $x1, x2, \ldots$, where $x1 < x2 < \ldots$, the cdf is a step function. The value of the cdf is constant in the interval $[x_{i-1}, x_i)$ and then takes a step, or jump, of size $p(x_i)$ at x_i. Thus, in Example 4, $p(3) = 3/21$, which is the size of the step when $x = 3$.

Example 5

The ***cdf*** for the device described in Example 3 (Module 45) is given by

$$F(x) = \frac{1}{2}\int_0^x e^{-t/2}dt = 1 - e^{-x/2}$$

The probability that the device will last for less than 2 years is given by

$$P(0 \le X \le 2) = F(2) - F(0) = F(2) = 1 - e^{-1} = 0.632$$

The probability that the life of the device is between 2 and 3 years is calculated as

$$\begin{aligned} P(2 \le X \le 3) &= F(3) - F(2) = (1 - e^{-3/2}) - (1 - e^{-1}) \\ &= -e^{-3/2} + e^{-1} = -0.223 + 0.368 = 0.145 \end{aligned}$$

as found in Example 3.

Review of Terminology and Concepts

Expectation

An important concept in probability theory is that of the expectation of a random variable. If X is a random variable, the expected value of X, denoted by $E(X)$, for discrete and continuous variables, is defined as follows:

$$E(X) = \sum_{\text{all } i} x_i p(x_i) \qquad \text{if } X \text{ is discrete} \qquad (6)$$

and

$$E(X) = \int_{-\infty}^{\infty} x f(x)\, dx \qquad \text{if } X \text{ is continuous} \qquad (7)$$

The expected value $E(X)$ of a random variable X is also referred to as the mean, μ, or the first moment of X. The quantity $E(X^n)$, $n \geq 1$, is called the nth moment of X, and is computed as follows:

$$E(X^n) = \sum_{\text{all } i} x_i^n p(x_i) \qquad \text{if } X \text{ is discrete} \qquad (8)$$

And

$$E(X^n) = \int_{-\infty}^{\infty} x^n f(x)\, dx \qquad \text{if } X \text{ is continuous} \qquad (9)$$

The variance of a random variable X denoted by $V(X)$ or var(X) or σ^2, is defined by

$$V(X) = E[(X - E[X])^2]$$

A useful identity in computing $V(X)$ is given by

$$V(X) = E(X^2) - [E(X)]^2 \qquad (10)$$

The mean $E(X)$ is a measure of the central tendency of a random variable. The variance of X measures the expected value of the squared difference between the random variable and its mean.

Thus, the variance, $V(X)$, is a measure of the spread or variation of the possible values of X around the mean $E(X)$. The standard deviation, σ, is defined to be the square root of the variance, σ^2. The mean, $E(X)$, and the standard deviation, $\sigma = \sqrt{V(X)}$, are expressed in the same units.

Example 6

The mean and variance of the die-tossing experiment described in Example 2 are computed as follows:

$$E(X) = 1\left(\frac{1}{21}\right) + 2\left(\frac{2}{21}\right) + \cdots + 6\left(\frac{6}{21}\right) = \frac{91}{21} = 4.33$$

To compute $V(X)$ from Equation (10), first compute $E(X^2)$ from Equation (8) as follows:

$$E(X^2) = 1^2\left(\frac{1}{21}\right) + 2^2\left(\frac{2}{21}\right) + \cdots + 6^2\left(\frac{6}{21}\right) = 21$$

Thus,

$$V(X) = 21 - \left(\frac{91}{21}\right)^2 = 21 - 18.78 = 2.22$$

and

$$\sigma = \sqrt{V(X)} = 1.49$$

Example 7

The mean and variance of the life of the device described in Example 3 are computed as follows:

$$E(X) = \frac{1}{2}\int_0^\infty xe^{-x/2}\,dx = -xe^{-x/2}\Big|_0^\infty + \int_0^\infty e^{-x/2}\,dx$$

$$= 0 + \frac{1}{1/2}e^{-x/2}\Big|_0^\infty = 2 \text{ years}$$

To compute $V(X)$ from Equation (10), first compute $E(X^2)$ from Equation (9) as follows:

$$E(X^2) = \frac{1}{2}\int_0^\infty x^2e^{-x/2}\,dx$$

Thus,

$$E(X^2) = -x^2e^{-x/2}\Big|_0^\infty + 2\int_0^\infty xe^{-x/2}\,dx = 8$$

giving

$$V(X) = 8 - 2^2 = 4 \text{ years}$$

Review of Terminology and Concepts

The mode

The mode is used in describing several statistical models that appear in statistical models in simulation.

Mode in Discrete Case: In the discrete case, the mode is the value of the random variable that occurs most frequently.

Mode in Continuous Case: In the continuous case, the mode is the value at which the pdf is maximized.

Bimodal Distribution: The mode might not be unique; if the modal value occurs at two values of the random variable, the distribution is said to be ***bimodal***.

Useful Statistical Models

Numerous situations arise in the conduct of a simulation where an investigator may choose to introduce probabilistic events.

In a queuing system, interarrival and service times are often probabilistic.

Inventory model: In an inventory model, the time between demands and the lead times (time between placing and receiving an order) can be probabilistic.

Reliability model: In a reliability model, the time to failure could be probabilistic. In each of these instances, the simulation analyst desires to generate random events and to use a known statistical model if the underlying distribution can be found.

Queueing systems

In queueing examples, interarrival- and service-time patterns, the times between arrivals and the service times are often probabilistic. However, it is possible to have a constant interarrival time (as in the case of a line moving at a constant speed in the assembly of an automobile), or a constant service time (as in the case of robotized spot welding on the same assembly line). The following example illustrates how probabilistic interarrival times might occur

Example 8

Mechanics arrive at a centralized tool crib as shown in Table 1. Attendants check-in and check out the requested tools to the mechanics. The collection of data begins at 10:00 A.M. and continues until 20 different interarrival times are recorded. Rather than record the actual time of day, the absolute time from a given origin could have been computed. Thus, the first mechanic could have arrived at time 0, the second mechanic at time 7:13 (7 minutes, 13 seconds), and so on.

Table 1 Mechanics' Arrival Data

Arrival Number	Arrival (hour:minutes::seconds)	Interarrival Time (minutes::seconds)
1	10:05::03	—
2	10:12::16	7::13
3	10:15::48	3::32
4	10:24::27	8::39
5	10:32::19	7::52
6	10:35::43	3::24
7	10:39::51	4::08
8	10:40::30	0::39
9	10:41::17	0::47
10	10:44::12	2::55
11	10:45::47	1::35
12	10:50::47	5::00
13	11:00::05	9::18
14	11:04::58	4::53
15	11:06::12	1::14
16	11:11::23	5::11
17	11:16::31	5::08
18	11:17::18	0::47
19	11:21::26	4::08
20	11:24::43	3::17
21	11:31::19	6::36

Example 9

Another way of presenting interarrival data is to find the number of arrivals per time period. Here, such arrivals occur over approximately 1.5 hours; it is convenient to look at 10-minute time intervals for the first 20 mechanics. That is, in the first 10-minute time period, one arrival occurred at 10:05::03. In the second time period, two mechanics arrived, and so on. The results are summarized in Table 2.

Table 2 Arrivals in Successive Time Periods

Time Period	Number of Arrivals	Time Period	Number of Arrivals
1	1	6	1
2	2	7	3
3	1	8	3
4	3	9	2
5	4	—	—

This data could then be plotted in a histogram, as shown in Figure 5.

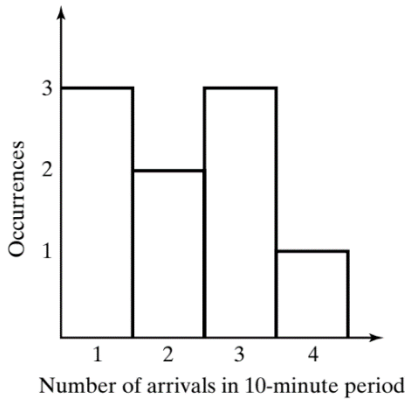


Figure 5 Histogram of arrivals per time period.

The distribution of time between arrivals and the distribution of the number of arrivals per time period is important in the simulation of waiting-line systems. "***Arrivals***" occur in numerous ways; as machine breakdowns, as jobs coming into a job shop, as units being assembled on a line, as orders to a warehouse, as data packets to a computer system, as calls to a call center, and so on.

Agent-Based Modelling

Before we discuss why agent-based modeling is important, we briefly describe what agent-based modeling is.

Agent: An *agent* is an autonomous computational individual or object with particular properties and actions.

Agent-based modeling: It is a form of computational modeling whereby a phenomenon is modeled in terms of agents and their interactions.

In the past two decades, scientists have increasingly used agent-based modeling methods to conduct their research.

- ABM is a species of computation, growing up alongside the maturation of computer technology.
- **The combination of large data, cheap computation, and high connectivity allow agent-based models to be constructed with millions of individual agents** whose properties and behaviors have been validated. Moreover, computational representations are dynamic and executable, allowing for greater interactivity between the user and representation. Perhaps, even more important, agent-based representations have particular advantages in that they are easy for people to understand.
- **Agent-based representations are easier to understand than mathematical representations of the same phenomenon.** This is because agent-based models are constructed out of individual objects and simple rules for their movement of behavior, as opposed to equational models that are constructed from mathematical symbols. In our natural discourse, we commonly describe our experience in terms of the interactions of individuals, as opposed to in terms of the rates of change of aggregates as in differential equations.
- **Individual Agents:** In thinking about individual agents, we can make sense of them by projecting our bodily experience onto the agents. Thus, the language and concepts we use in agent-based modeling are much closer to natural language and our natural thinking.

Steps for Conceptualization: As a first step to presenting this conceptualization, we look back historically at changes to representational tools and practices used in science and the significant benefits they had for both scientists and learners.

A Thought Experiment
Imagine a country (let's call it Foo) where people represented numbers as the Romans did, using symbols such as MCMXLVIII. Fooian scientists worked laboriously to quantify and more accurately calculate basic science such as planetary motion and dynamical forces.

Number of Different Approaches: Business people had great difficulties with their financial calculations, tradespeople struggled with their measurements, and consumers labored to assess their purchases. Educational researchers and policymakers in this imaginary country were very concerned with the difficulty of learning to handle numbers, and they worked hard to make these skills accessible to more of their citizens. They engaged in several different approaches.

The Sort of Innovation: A first step is to name the sort of innovation associated with the shift from Roman to Hindu-Arabic representations of numbers. It is not sufficient, for example, to say that we have a new approach to learning and working with numbers.

Structuration: By *structuration,* we mean the encoding of the knowledge in a domain as a function of the representational infrastructure used to express the knowledge.

Restructuration: A change from one structuration of a domain to another resulting from such a change in representational infrastructure we call a *restructuration*.

Examples of Restructuration: 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.

COMPLEX SYSTEMS AND EMERGENCE

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.

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.

From Simple to Complex System

As we gain facility and a more complete understanding of simple systems, we naturally progress to trying to make sense of increasingly complex systems.

Simple Population Dynamics Models

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

complex systems is also a natural result of the growth of human knowledge.

Complex Systems Theory: 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.

Emergent Phenomenon: 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.

Emergence: Since Lewes's time, scholars have struggled with how to best define emergence — some definitions are 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*.

Features of Emergence: 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."

Macrostructures Are Emergent: Structure (or rules) at the micro-level leads to ordered patterns 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.

Understanding Complex Systems and Emergence
We have said that understanding complex systems and emergence is hard for people.

Fundamental And Distinct Challenges: 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.

  Integrative Understanding: 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.

  ***Differential or Compositional* Understanding:** 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.

Example 1: Integrative Understanding

Figure 0.1 presents a system composed of a few identical elements following one rule.

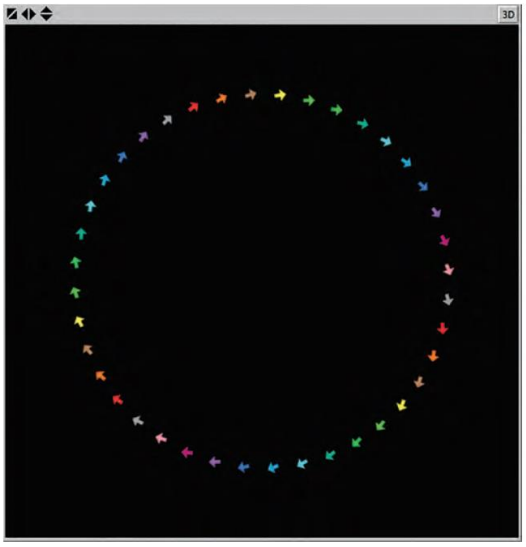


Figure 0.1
Some arrows moving clockwise around a circle of radius 20.

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 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 pattern that we see?

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 large, very dense flocks of starlings that resemble insect swarms. How and why do these flocks emerge?

Figure 0.3
Flock of geese flying in a classic V formation.

Deterministic-centralized Mindset

In the 1980s and early 90s, Wilensky and Resnick interviewed people from a wide range of backgrounds, asking them to explain how such patterns arise. The results suggested a phenomenon, or cognitive pattern, called the deterministic-centralized mindset, or DC mindset. The pattern stemmed from two main empirical findings:

(1) Most subjects did not see any role for randomness in creating these structures, randomness was seen as destructive of pattern, not a force for creating a pattern (the D component); and

(2) Most subjects described these patterns as arising from the actions of a centralized controller or orchestrator (the C component).

Thinking In Levels: 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's two 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 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.

Appearance Of Stability: 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.

> **Complex Phenomena:** Wilensky and Resnick also showed a host of examples where this level's slippage interfered with both the integrative and differential understanding of complex phenomena in the natural and human social worlds.

Agent-Based Modeling as Representational Infrastructure for Restructurations: 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 of science we have relied on simplified descriptions of complexity

Agent-based modeling

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.

Gas Molecule Agent

A gas molecule agent, for instance, might have properties such as “mass” with a value of 30 atomic mass units, “speed” with a value of 10 meters per second, and “heading” with a value of the angle it is facing.

A sheep agent

A sheep agent, by contrast, might have properties such as “speed” with a value of 3 mph, weight with a value of 30 lbs., and fleece with a value of “full” (a discrete-textual rather than numerical value).

Rules of Behavior

In addition to their properties, agents also have rules of behavior. A gas molecule agent might have the rule to collide with another molecule; a sheep agent might have a rule to eat grass if there is grass available nearby.

Universal Clock

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).

Agents And Rules

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.

ABM’s Enable Restructurations of Complex Systems

We contend that ABM 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?

NetLogo

The computer language used in this text, NetLogo (*Wilensky, 1999*), was developed by Uri Wilensky for these express purposes. 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.

Turtle:

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.

Netlogo Used for Constructing And Exploring Models

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 secondary schools.

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.

Lotka-Volterra Differential Equations

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:

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

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

The first equation says that the number of predators increases as predators interact with prey (by fixed constant K1) and decreases by a constant mortality rate (M).

The second equation says that the number of prey increases by a constant birthrate (B) and decreases in interaction with predators (by a fixed constant K2).

The solution to these equations resembles the classic sinusoidal curves that show the cycling of the predator populations with one ascendant when the other is at a trough.

An Agent-based representation of Predator and Prey

An agent-based representation of predator and prey, such as the one illustrated in figure 0.6, would typically employ simple algorithmic models.

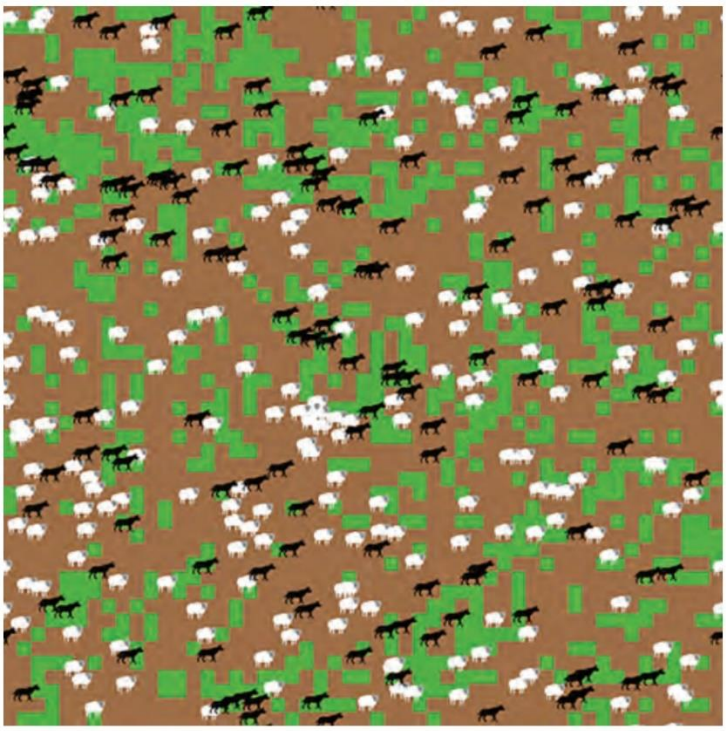


Figure 0.6
An agent-based Wolf Sheep Predation model. (See *Wilensky, 1997c*)

They might give each predator and prey a store of energy that is depleted when they move and increased when they eat. If their energy dips too low, they die. If it is high enough, they might reproduce.

And when they move, if they encounter food (which, for the predator, is the prey), they eat it. These instructions are explicitly stated in an easy-to-read language that instructs each agent in how to behave and accompanies the visual model.

Explicit Mechanism

This kind of representation makes its mechanisms explicit; as such, they are usually quite understandable by even young children. They also can be challenged more easily and tested. We will explore such predator-prey models in detail.

Advantages To Agent-based Model Representations

There are many advantages to agent-based representations. First, it does not require knowing calculus. Requiring calculus to be able to reason about predator-prey interactions sets the entry threshold quite high.

In the United States, only a small percentage of adults have ever taken calculus, and far fewer are familiar with differential equations. So calculus

serves as a gatekeeper limiting access to important content to a small minority of adults and almost all children. Yet the content of the predator-prey model can be made quite accessible to a non-calculus audience through ABM. This gatekeeper problem may seem less formidable when thinking about an audience of scientists who have likely had calculus. Even with this audience in mind, it is still often difficult to uncover the mechanisms behind the equations, to challenge them, and to propose alternate mechanisms and new

equations.

Agent-based models can serve as powerful complements to equation-based models. They are particularly effective entry points into scientific domains.

Disadvantage Of ABM Representation

But they also have some disadvantages as compared to equations.

- For an expert, an equation can more compactly represent a phenomenon than can an ABM.
- Moreover, when the equation is solvable, it enables the direct calculation of results without the need to run a model.

When a model requires a large number of agents, its execution time can be so long as to be impractical as a way of calculating results. The corresponding equation-based models must often make many simplifying assumptions to gain this increase in speed.

These simplifications are most justifiable when the agents are sufficiently homogenous that it can be advantageous to treat them as average quantities as opposed to the heterogeneous individuals often used in ABM.

Disadvantages of ABM

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.

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.

- **Classic heat equation**
- **Spread of a forest fire**

- 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.

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

ABM Approach To Modeling Forest Fires

Contrast these two equations 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.

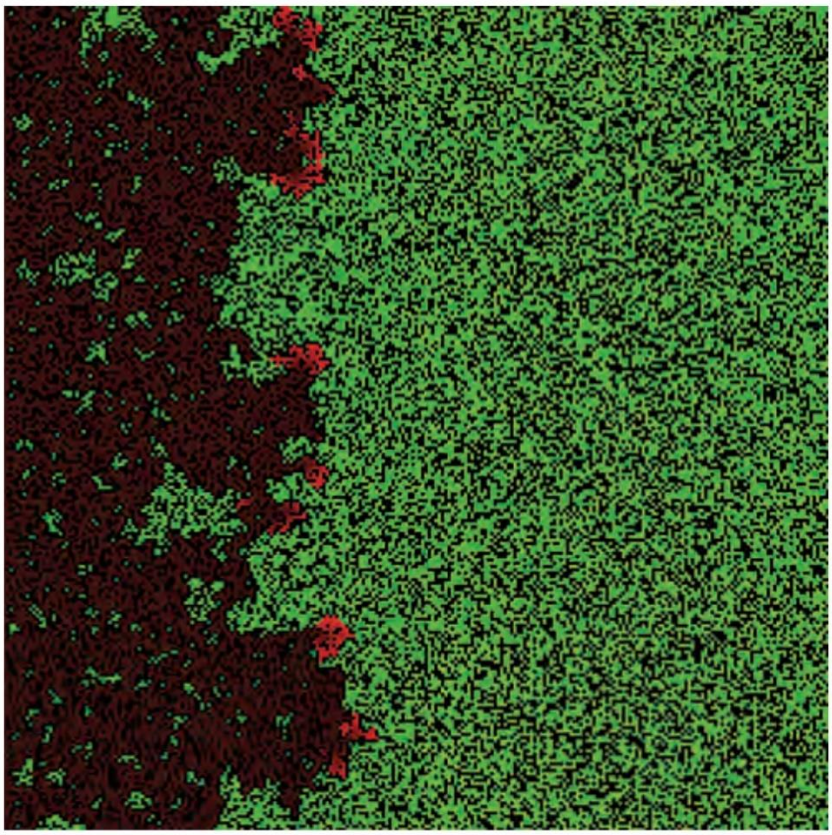


Figure 0.7
An agent-based model of a forest fire. (See *Wilensky, 1997b*)

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.

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.

ABM Provides Several Representational Benefits

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.

ABM uses Discrete rather than Continuous Representations

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.

Ants (Foraging For Food)

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 have 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 above paragraph of this section is in itself a model of ant behavior.

Model of Ant Behavior

Model: 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.

Textual Description: 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.

Computational Model

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, algorithmically manipulates those inputs, 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 into a working computational simulation (written in some form of computer "code").

Besides textual models, other forms of conceptual models describe processes, objects, or events but are not computational; conceptual models can also be **diagrammatic** or **pictorial**.

Agent

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 multivalued state)*

Agent-based modeling (ABM)

- Agent-based modeling (ABM) is a computational modeling paradigm that enables us to describe how any agent will behave.
- 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.

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.

Complexity and Complex Systems

- The study of complex systems has become an important scientific frontier.
- By the term complex system, we mean a system composed of many distributed interacting parts.
- The field of complex systems or complexity science arose in the mid-1980s and was born out of a variety of disparate fields from economics to physics to ecology.
- Complex systems science provides a set of tools and frameworks for viewing phenomena.
- Any phenomenon can be viewed as a complex system, and choosing when to do so depends on assessing when it is most useful to use the lens and/or methodologies of complex systems.
- One important methodology of complexity science is agent-based modeling.

Creating the Ant Foraging Model (Hypothesis)

In the next few paragraphs, we will describe some hypotheses about how ants accomplish this behavior and what mechanisms are at work that enables ants to find food in this way.3
Perhaps the trails form as follows:

- After an ant finds food, it goes back to the nest, drops the food off, and communicates to the queen ant, and this queen ant tells the other ants where the food is and sends them off to collect it.
- Suppose a researcher notices that the food-finding ant never communicates with any kind of "boss" ant before leaving the nest again, so the researcher might reason that food gathering was not working through a centralized leader or controller but rather through distributed control.

Another Hypothesis

- Another hypothesis that could be advanced is that perhaps the ant did not communicate with a central leader,
- but rather simply communicated with other ants, and those other ants were able to "diffuse" the information about the food source throughout the nest.
- There is reason to believe such a hypothesis might be true since bees work similarly;
- when a bee finds a food source, it returns to the hive and conducts a complicated dance that tells the other bees where to find the food (Gould & Gould, 1988).
- However, though ants do have some methods of communicating information directly (Hölldobler & Wilson, 1998)
- Most species do not communicate the exact route to a food source in this way.

Basic Hypothesis

- 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?

Implemented Model

- 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 to operate per the model described earlier.

Set of Rules

- 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 is no 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.

NetLogo

- These rules are easily implemented in a computer language.
- There are many computer languages in use for many purposes, but only a few are specially 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.
- NetLogo is so easy to use that, rather than describe algorithms and models in pseudo-code, throughout this course 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.)

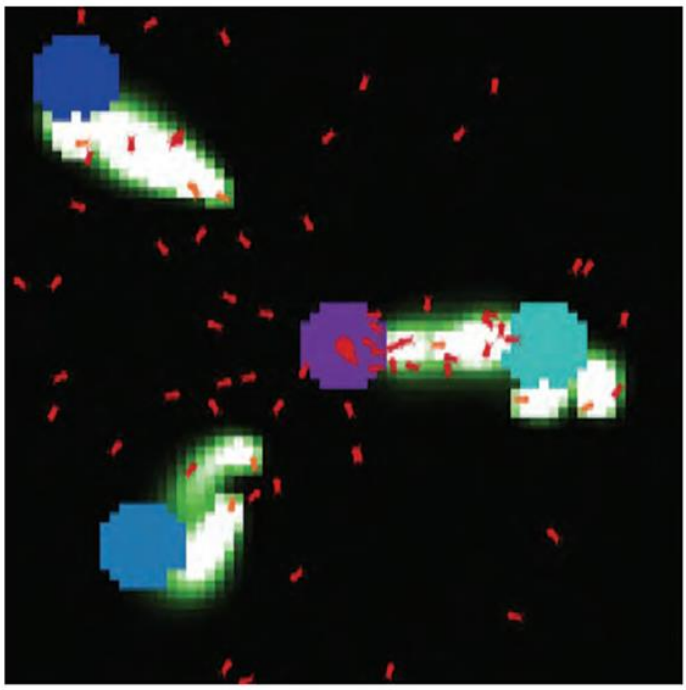


Figure 1.2
NetLogo "Ants" model of foraging behavior.[8]

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 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 optimally.
- 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.
- 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.

Exploration & Exploitation

- 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 sometimes 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.

Emergent Phenomenon

- 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 anyone 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).

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.

Eight Main Uses For Agent-based Models

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

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 world.
- But all models are coarse-grained descriptions of reality; and 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 (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

- Models are explanatory in that they point out the essential mechanisms underlying a phenomenon.
- They can function as 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.

The key function of ABMs

- 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 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 their results to the Ants model and to real-world observations to determine the most plausible explanation.

Models Facilitate Experimentation

Models facilitate ***experimentation***.

- Models can be run repeatedly to note variations in their dynamics and their outputs.
- Some models have very little variation from run to run.
- Others exhibit ***path dependency*** and can therefore vary tremendously from run to run.

Experimentation

Model parameters can be varied to see their effect on model behavior and outputs. For example, in the Ants model, we could change the evaporation rate of the pheromone and see what effect that has on the performance of the ants. Alternatively, we could model the return to nest behavior in more detail, by placing a sun in the environment and having the ants calculate their path back based on the sun.

Effects of modifications on the system

Thus agent-based models enable us to easily examine different mechanisms and attributes of a system and see what effect those modifications have on the overall behavior of the system.

Model's Behavior Classification

By varying these different attributes or parameters and observing their effect on the behavior of the system, we can classify model behavior into output regimes, or characteristic output behaviors of the model.

Models Provide Us with Analogies

Since models are simplifications of reality, they enable us to find similarities with other such simplifications even if the modeled phenomena are very different. In this way, we can apply reasoning gained in one domain

- of knowledge to other knowledge domains. For instance, in 1996, Schoonderwoerd, extending Dorigo's work on ant colonies as optimization devices, found that the problem of ants efficiently finding food sources is similar to the problem of packets on networks efficiently seeking out their destinations (Schoonderwoerd et al., 1996).

Communication And Education

- Agent-based models can be used as a vehicle for communication and education.
- We can show the model to people who have never seen an ant colony before and they can explore how ant colonies behave.
- Models provide us with an educational tool that encapsulates knowledge that may not be readily available from real-world observation.
- Moreover, an agent-based model allows us to expand beyond static knowledge, enabling learners to conduct experiments much as scientists do.
- If someone has a hypothesis as to how a particular mechanism works, they can implement that mechanism and see whether it can account for observed behavior.

Thought Experiments

Models can sometimes present new phenomena that are not necessarily about some real-world phenomenon but are ***thought experiments*** on possible computations.

Some might not use the word "model" for this kind of computational artifact, but many scientists do refer to them as models. Some classic examples of these kinds of models are cellular automata, fractals, and particle swarms.

Prediction

A common conception of computer modeling is that its major purpose is *prediction*.

- Indeed, we often use modeling to think about possible future scenarios, and to the extent that any modeling tool can be used for discussing the future, agent-based modeling can be used in this way.
- However, like any modeling tool, an agent-based model's predictions depend on the accuracy of its input data.

- This is especially true when the events we are interested in are the results of complex systems, for which small changes in inputs can
  often lead to very different results.
- Many times, while modelers may claim to want to use a model to predict the outcome of a system, such as an ant colony, they use the model to *describe* past patterns of behavior, such as what food source was first eaten, and to *explain* future patterns that might arise, such as how ants might move around on the landscape.
- Sometimes modelers start motivated by generating predictions but end up finding greater power for explanation and description.
- Agent-based models, in particular, are distinctive from other modeling approaches in that they were designed to understand and explain complex phenomena that otherwise could not be explained through traditional approaches.

The Reality of Agents

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.

What is an Agent?

An agent is an autonomous individual or object with particular properties, actions, and possibly goals.

What is the environment?

- The environment is the landscape on which agents interact and can be geometric, network-based, or drawn from real data.

What about the interactions?

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.

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.

Differences between Agent-based Modeling (ABM) and equation-based modeling (EBM)

Many differences between ABM and equation-based modeling (EBM) have been discussed.

- 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 an 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 assume that the population size is large and that spatial effects are unimportant.

Randomness

One important feature of agent-based modeling, and computational modeling in general, is that it is easy to incorporate randomness into your models.

Benefits of Randomness

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.

Example of Randomness

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 modify their heading by a small amount based on a random number.

Ant Model and Randomness

- 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.

- Finally, there are often times when we simply do not know enough about how a complex system works 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.

ABM Benefits

- 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.

Examples

Some problems with a 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.

General Rules

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).

ABM Tradeoffs

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 appropriate tool at the appropriate time, and sometimes agent-based modeling is not the right tool for the job.

For example, ABM can be computationally intensive. Simulating thousands or millions of individuals can require great computing power.

Equation-based models, by contrast, are often very simple to run and essentially just require repetitive mathematical calculations. This is true only for simple equation-based models; numerically solving complicated equation-based models may take as much computational time as agent-based models.

Still, ABM may be Better

The computational expense of running an ABM is a price one pays for having

the benefits of rich individual-level data. The additional computational power needed for running ABMs is the same power that allows the tracking and development of rich histories of individuals.

Examples

For example, in the Ants model, we could write a simple equation that describes the rate at which food is gathered based on its distance from the nest.

However, through ABM, we can observe the behavior of the individual ants and understand how they form trails to the food source.

Other cases

- ABM does require us to gain an understanding of the micro behavior of a system, but without modeling the micro behaviors, ABM would not provide as elaborate a model as it does.
- While ABM does require us to know or guess about the individual-level mechanisms operating, it does not require knowledge of the aggregate-level mechanism.

Now We will learn to construct a few simple agent-based models. These simple

models, sometimes referred to as “ toy models, ” are not meant to be models of real phenomena, but instead are intended as “ thought experiments. ” They are offered, as Seymour Papert puts it, as “ objects to think with ” (1980). Our purpose is to show that it is relatively easy to create simple agent-based models, yet these simple models still exhibit interesting and surprising emergent behavior. We will construct three such models: “ Life, ” “ Heroes and Cowards, ” and “ Simple Economy. ”

Game of Life: History

In 1970, the British mathematician John Horton Conway (described in Conway, 1976) created a cellular automaton that he called the “ Game of Life. ” Martin Gardner (1970) popularized this game in his Scientific American column. Subsequently, millions of readers played the game.

Game of Life

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 squares1 (or “ cells ”) on aside. 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 as 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

(2) If it has more than three (3) alive neighbors, it also dies

(3) If it has exactly two (2) alive neighbors, the cell remains in the state it is in

(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.

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.

Neighbors

n agent-based modeling, we often model local interactions. This is accomplished by having cells in the grid talk to their neighbor cells. Most often, we are using two-dimensional grids or lattices with square cells.

In a square lattice, there are two commonly used neighborhoods: the **von Neumann neighborhood** and the **Moore neighborhood**.

A cell’s ***von Neumann neighborhood*** consists of the four cells that share an edge with it, the cells to the north, south, east, and west (the green squares in the right figure that follows).

A ***Moore neighborhood*** consists of the eight cells that touch it, adding the cells to the northeast, southeast, northwest, and southwest (the entire green area in the left figure that follows).

In NetLogo, the primitive NEIGHBORS refers to a patch’s Moore neighbors, and NEIGHBORS4 refers to a patch’s von Neumann neighbors.

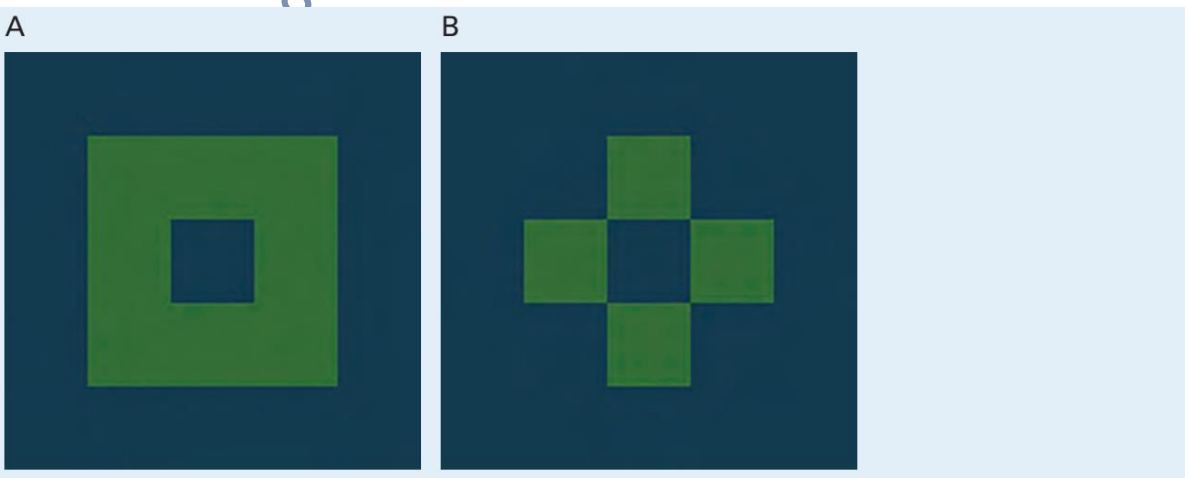


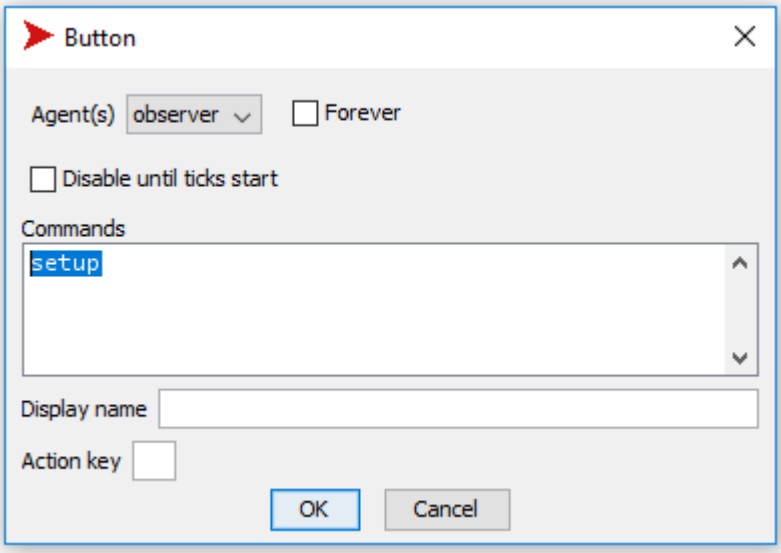
Figure 2.10
(A) Moore Neighborhood. (B) von Neumann Neighborhood. This will be further elaborated in chapter 5.

Overview

- Observing how Game of Life works
- Examining emergent patterns

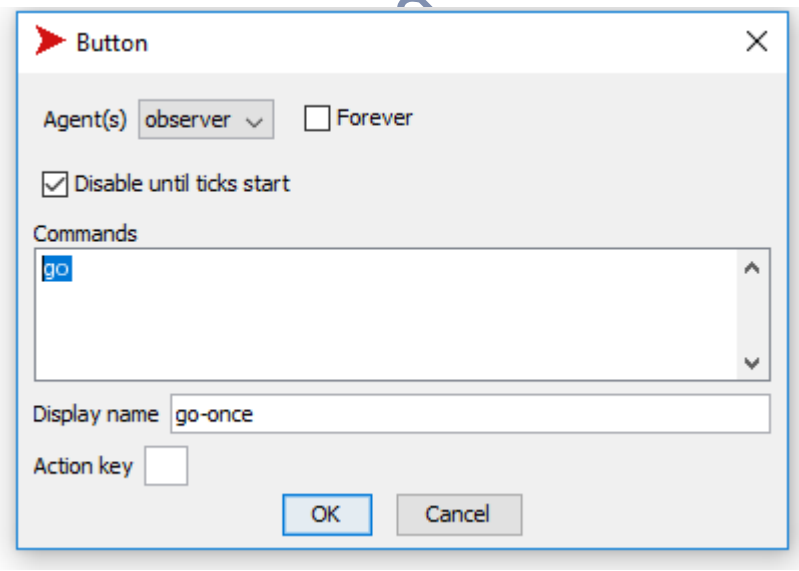
User Interface

- Adding a Button



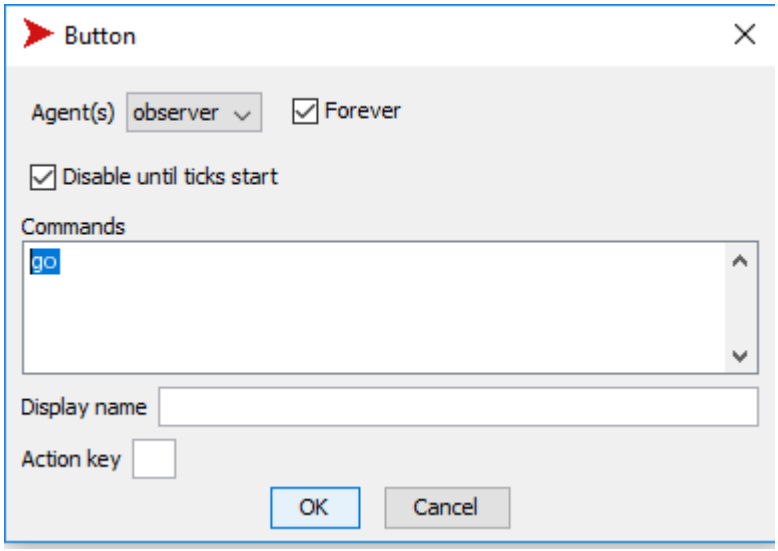
Go-Once button

- Add another button



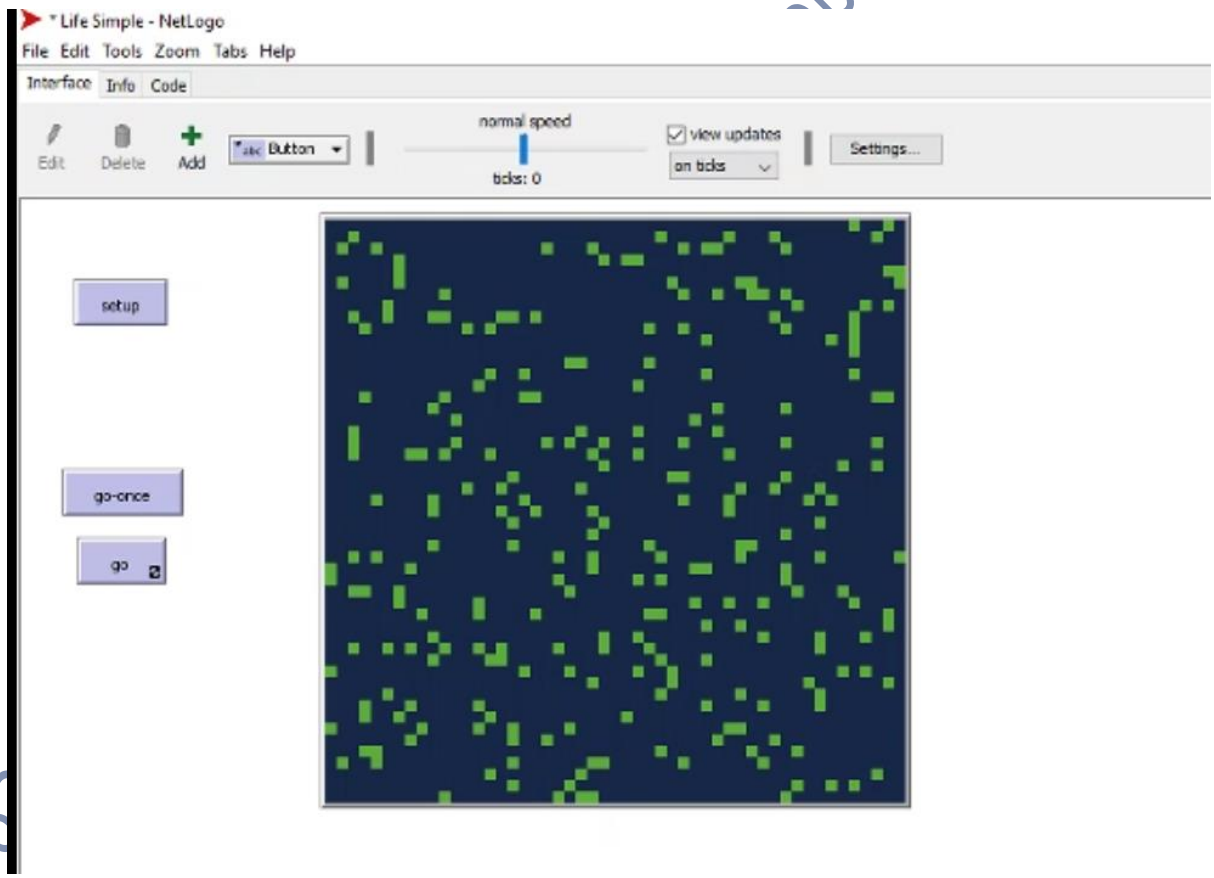
Go button

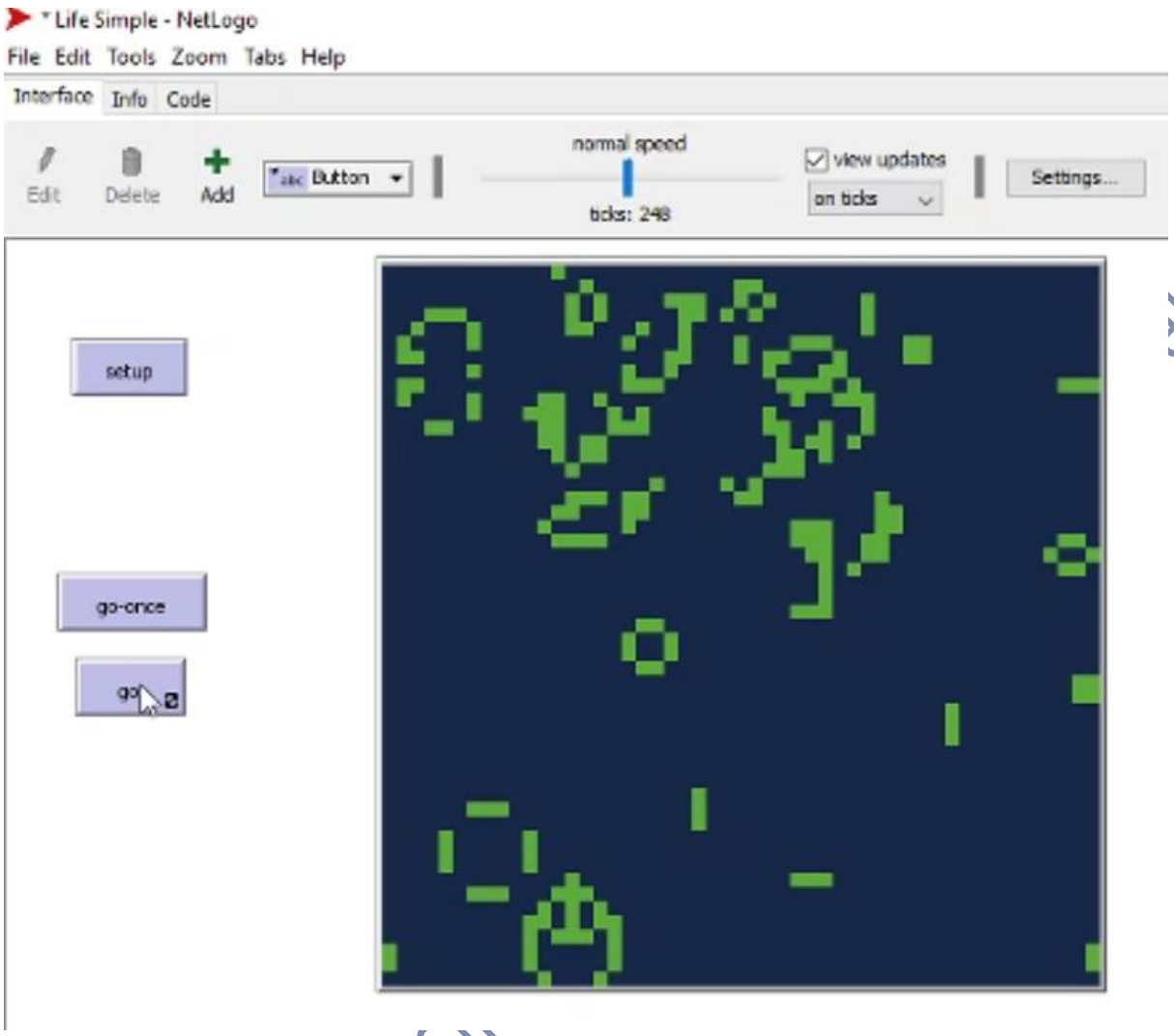
- Add yet another button



Understanding the Output

Simulation of Life





- **Emergent Pattern: Glider**

- **Another emergent pattern**

Summary

- We have examined outcomes from the game of life
- We have seen how emergent patterns emerge unexpectedly

In the past two decades, there has been increasing interest in and use of agent-based modeling in the social sciences. Several of the models that we have already discussed, such as Ants, the Game of Life and Heroes, and Cowards have been applied to understanding social systems.

Why ABM in Other Sciences?

Agent-based methods may be particularly valuable in the social sciences where agents are heterogeneous and mathematical descriptions often do not offer sufficient descriptive power.

Organizations

Several prominent communities have organized around using complex systems methods and agent-based modeling in the social sciences. Among these are:

- the Complex Systems Social Sciences Association (CSSSA) and
- the World Congress on Social Simulation (WCSS),

both of which also organize conferences. Several such events are included as part of other meetings.

Other Organizations

In recent years, organizations such as

- the American Education Research Association (AERA),
- Marketing Science,
- Eastern Economics Association, and
- American Association of Geographers (AAG)

The above-mentioned organizations have hosted sessions on the intersection of agent-based modeling and social science.

Economics Using ABM

One area that has been receiving increasing attention from the ABM community in economics. As economies consist of heterogeneous actors such as buyers and sellers, there is a natural mapping of ABM methods to economics. In 1996, the economists, Josh Epstein and Robert Axtell published a book that depicted a world they called SugarScape, which was populated by economic agents.

Simple Economy Description

In this section, we will build a very simple economic model that has some surprising results.

Suppose you have a fixed number of people, say 500, each starting out with the same amount of money, say $100. At every tick, each person gives one of his or her dollars to any other person at random. What will happen to the distribution of money? An important constraint is that the total amount of money remains fixed, so no one can have less than zero money. If you run out of money, you cannot give any away until you get some back. Refining the question a

little further, we ask: Is there a stable limiting distribution of the money? If so, what is it? For instance, will all the wealth be concentrated in a few hands or will it be equitably distributed?

Simple Economy Intuition

Many people, when posed with this question, have an intuition that there is a limiting distribution and that it is relatively flat. The reasoning behind the intuition is that since no

person starts with an advantage and the selection of people to whom money is transferred is random, no person should have much of an advantage over any others. Thus, the resulting wealth distribution should be relatively flat: Everyone should wind up with

roughly the same amount of money he or she started with.

Alternate Intuition

Other people have an intuition that wealth should be normally distributed.

Overview

- Go in more detail about the "Simple Economy" model
- Understand the code for Simple Economy

Simple Economy

```
turtles-own [ wealth ]

to setup
  clear-all
  create-turtles 500 [
    set wealth 100
    set shape "circle"
    set color green
    set size 2

    ;; visualize the turtles from left to right in ascending order of wealth
    setxy wealth random-ycor
  ]
  reset-ticks
end
```

```
to go
  ;; transact and then update your location
  ask turtles with [ wealth > 0 ] [ transact ]
  ;; prevent wealthy turtles from moving too far to the right
  ask turtles [ if wealth <= max-pxcor [ set xcor wealth ] ]
  tick
end

to transact
  ;; give a dollar to another turtle
  set wealth wealth - 1
  ask one-of other turtles [ set wealth wealth + 1 ]
end

to-report top-10-pct-wealth
  report sum [ wealth ] of max-n-of (count turtles * 0.10) turtles [ wealth ]
end

to-report bottom-50-pct-wealth
  report sum [ wealth ] of min-n-of (count turtles * 0.50) turtles [ wealth ]
end
```

Summary

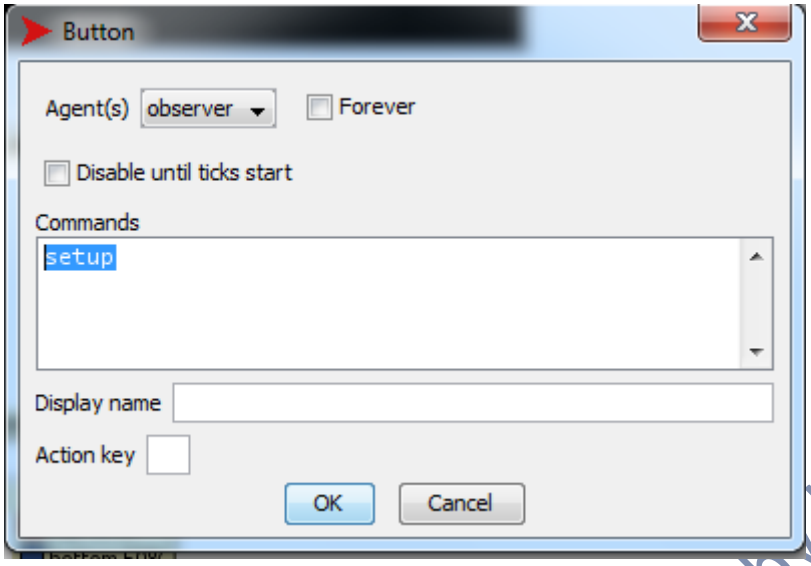
- We have examined the implementation of the Simple Economy Model
- We have seen how turtles can be used to develop a model

Overview

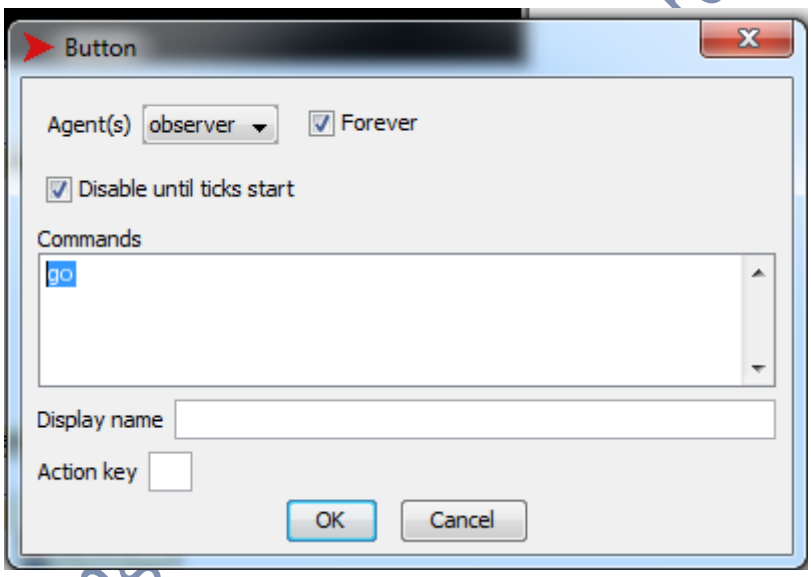
- Understanding the User Interface of Simple Economy
- Understand the execution of Simple Economy

Simple Economy

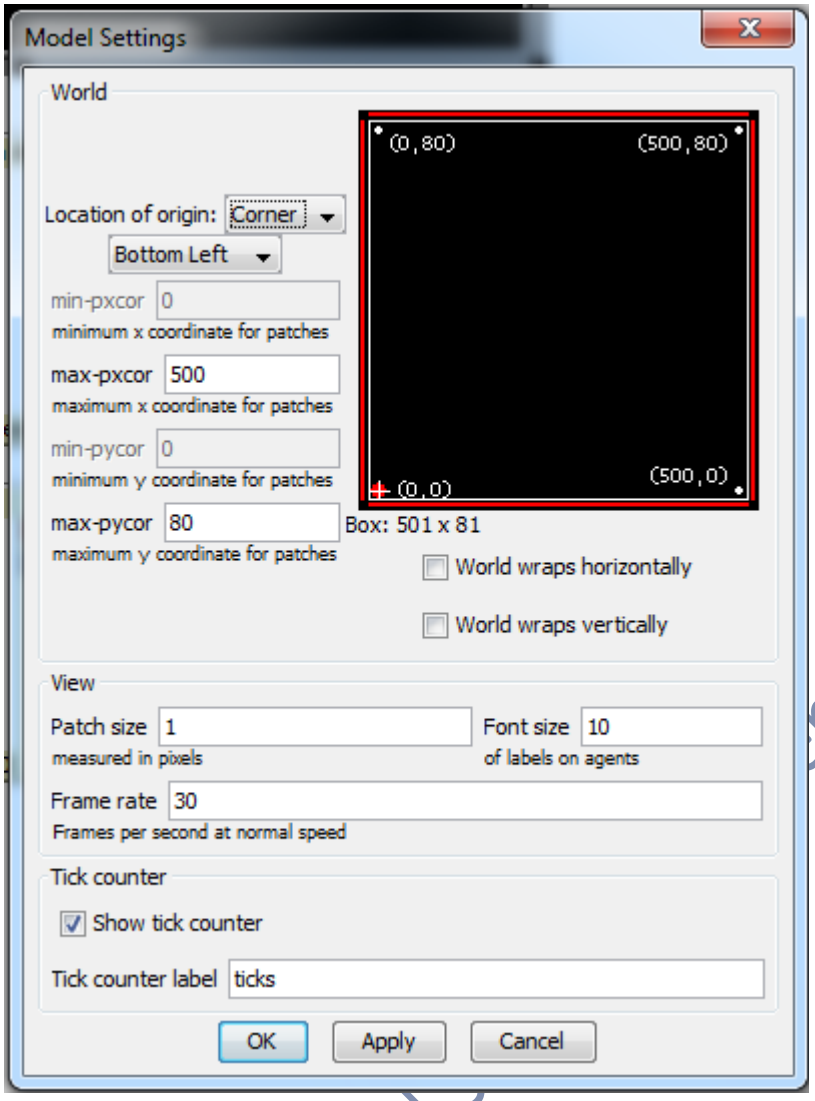
- First we add a button



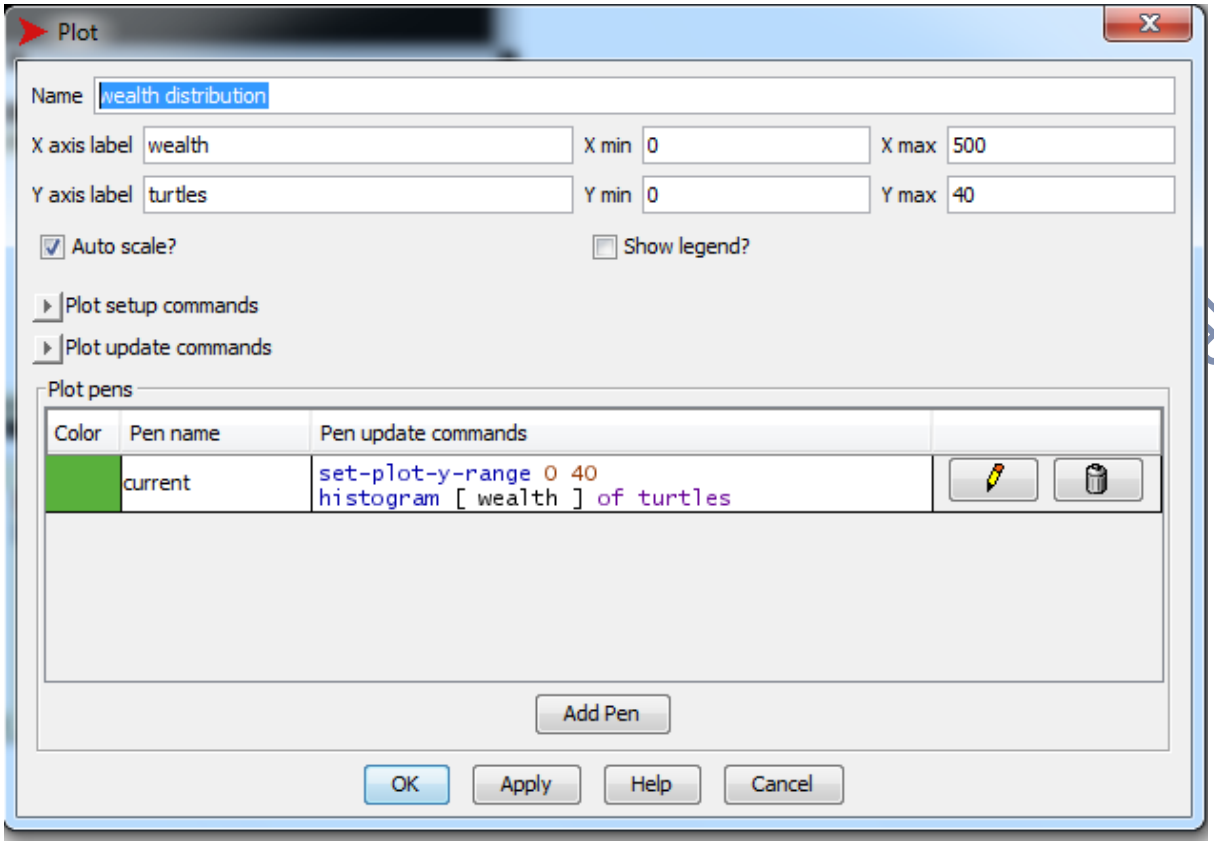
- Another button



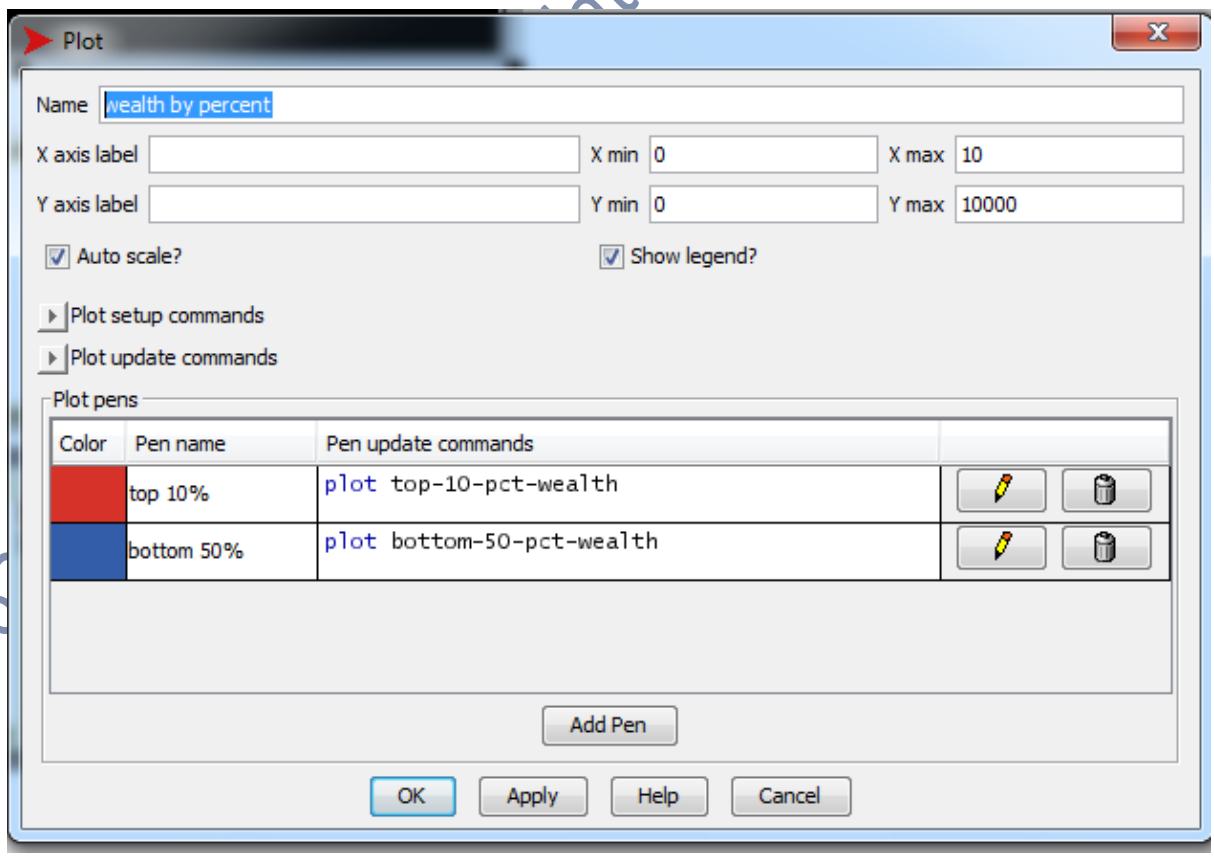
- Next, let us adjust the world settings



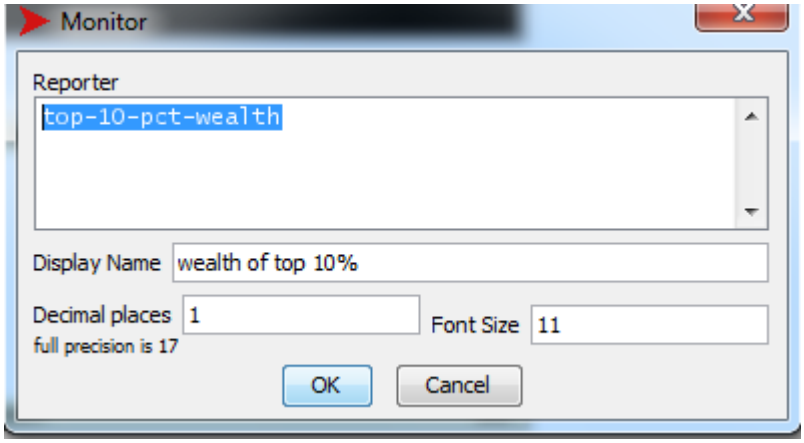
- Now, we add a plot



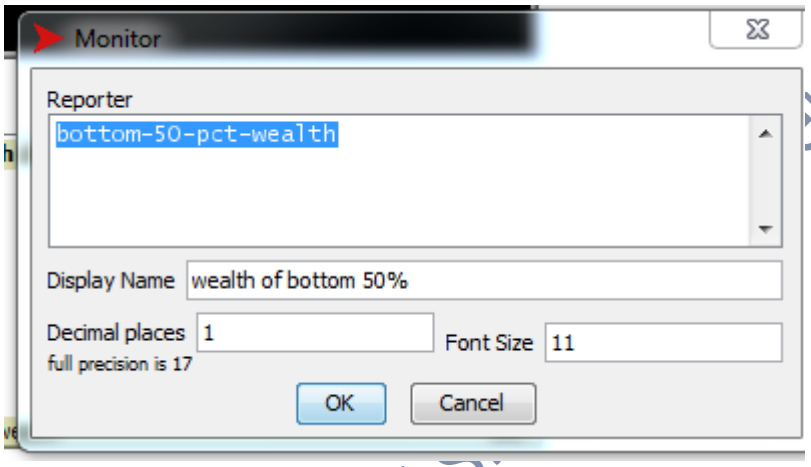
- **Now, we add another plot**



- **Now, we add a monitor**



- **Now, we add another monitor**



Summary

- We have examined the design of the User interface for the Simple Economy Model

Overview

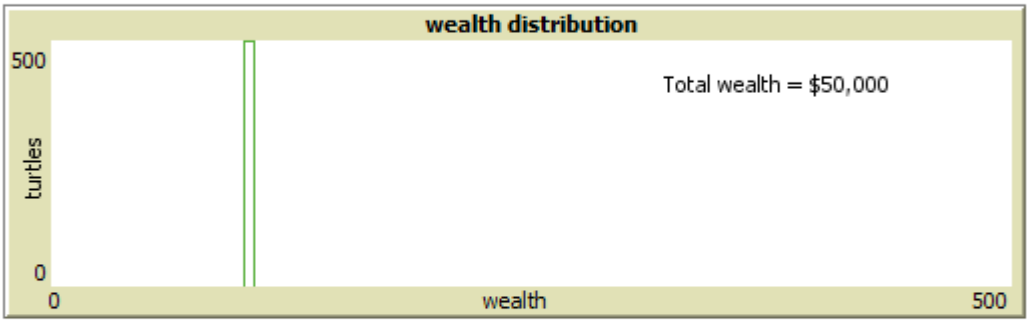
- Understand the Complexity dynamics of the Simple Economy Model

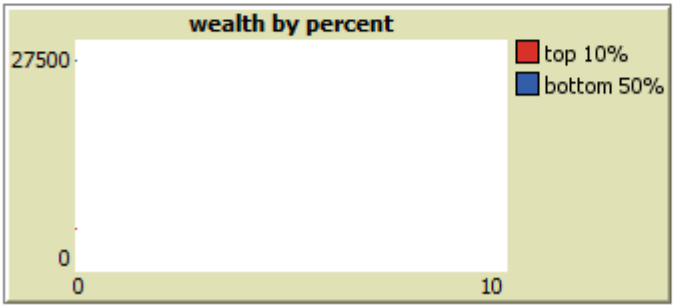
Simulation Run

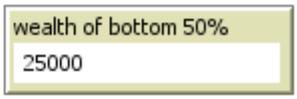
- Let us see how this works in NetLogo

Running the Model

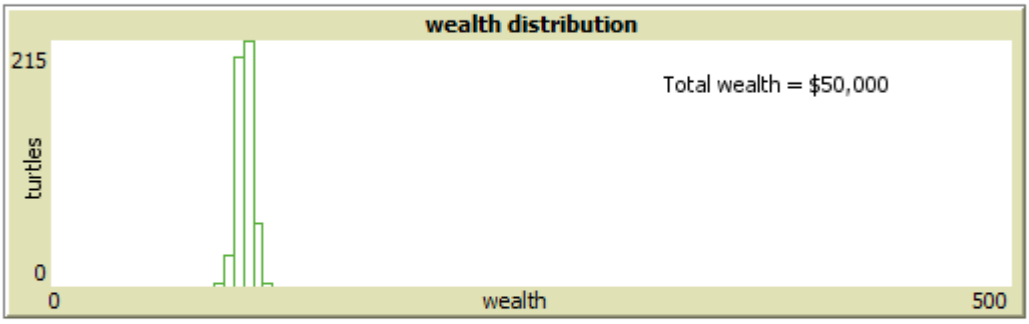
- Let us see the initial conditions

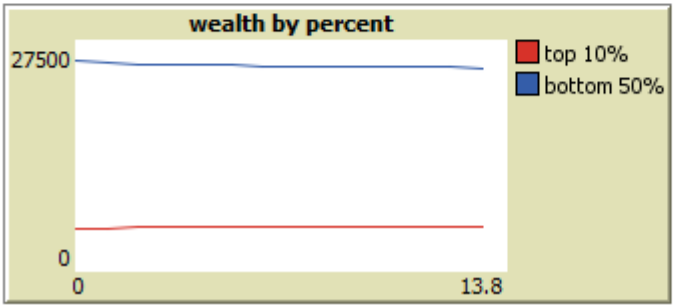


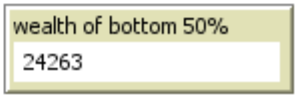


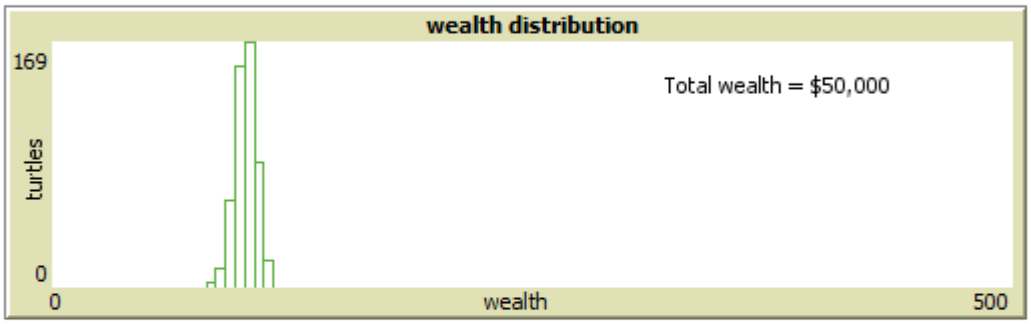


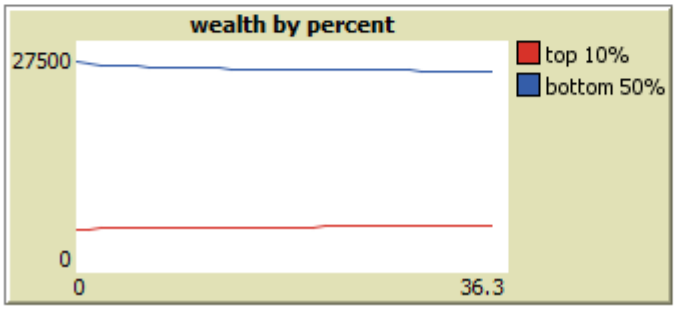
- After running

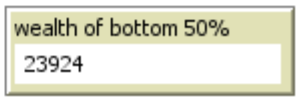




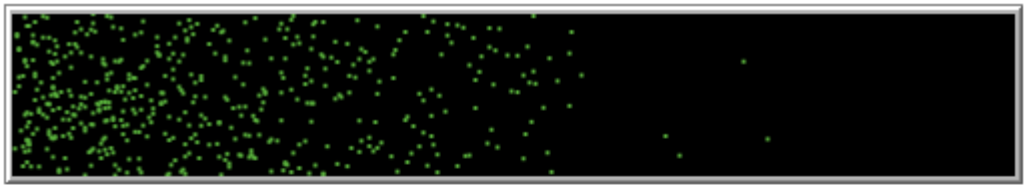


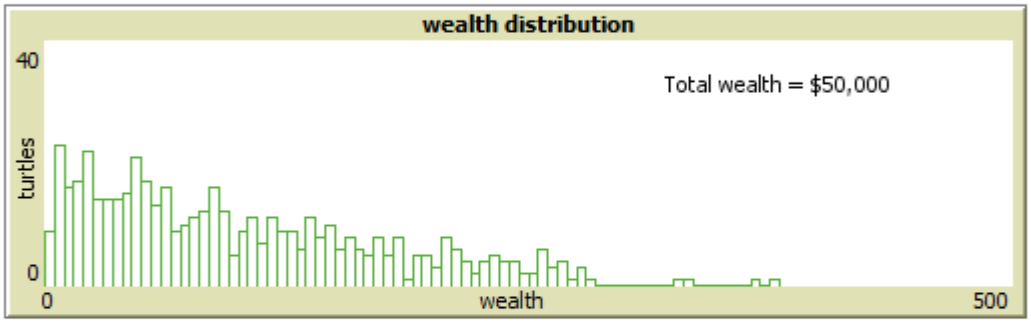


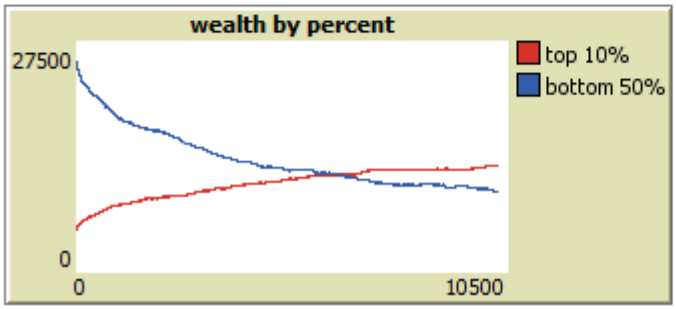


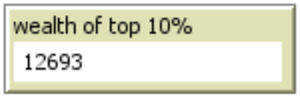


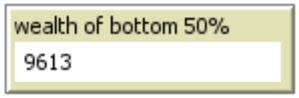
- **After long run: 10,000**



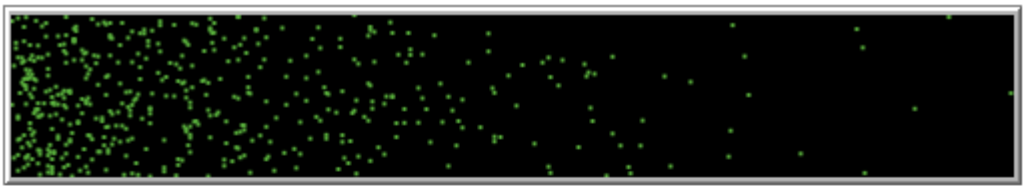


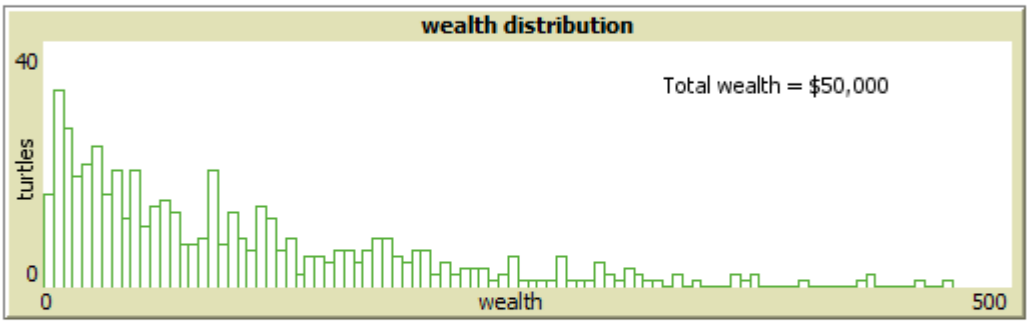


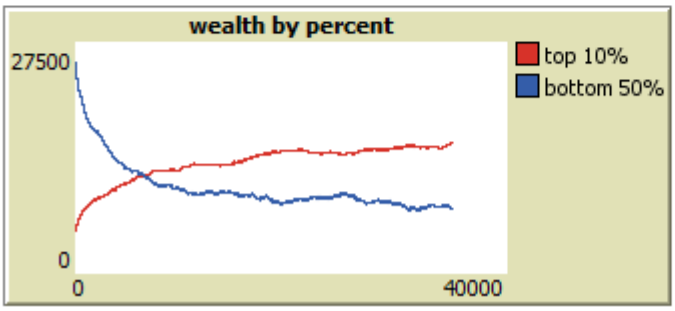


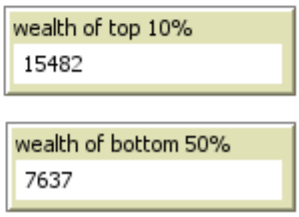


- **After long run: 35,000**









Understanding Results

- We can see that the distribution is not at all flat.
- There are a few very wealthy individuals and many poor agents.
- At this time step in this model run, the wealth of the top 10 percent of the agents is a total of $12,633, or an average of $253 per person
- Whereas the total wealth of the entire bottom 50 percent is only $10,166, or an average of $41 per person.
- The crossover point at which the wealth of the top 10 percent of all agents exceeded the wealth of the bottom half, or 50 percent, of all agents was at 5,600 ticks.
- If we run it for 25,000 more ticks, the gap continues to grow and now the top 10 percent have more than twice as much in total (and more than 10 times as much per individual) than the bottom 50 percent.
- The distribution of money eventually converges to a stationary distribution.
- This distribution has been shown to be exponential, which means that there is great inequality in monetary wealth.
- The key condition that creates this stationary distribution is the conservation of money.

Summary

- We have examined the Execution of the Simple Economy Model
- We have seen how surprising results can appear in relatively simple models

Overview

- Understand some commonly used NetLogo concepts

Understanding Complex Results

- Understanding complex results often requires advanced modeling techniques
- E.g. in Simple economy, we have seen complex results
- Some of these results need correlation with the real world
- Empirical studies have shown that wealth distribution in most countries is not distributed exponentially,
- Instead it follows a power law known as the Pareto distribution (Pareto, 1964).
- This distribution is even more unequal than an exponential distribution.
- Power law distributions arise in a surprising number of contexts
- They are typically associated with preferential attachment processes (Barabási, 2002)
- The amount received is proportional to the existing amount.

Lists and Agentsets

- To model such complexity we often need advanced programming concepts
- Some of the most used constructs (data structures) in NetLogo are:
- Agentsets
- Lists
- An agentset is a set of agents.
- Agentsets can have turtles, patches, or links, but they cannot mix agent types.
- You can ask an agentset to perform some commands.
- NetLogo comes with three special agentsets built in: “ turtles, ” “ patches, ” and “ links”
- What makes agentsets so powerful is that you can create your own agentsets
- For example, an agentset of all the red turtles,
- Or an agentset of all the patches in the upper right quadrant.
- Agentsets are always in random order.
- So if you ask an agentset, several different times, to execute some commands, each time the order in which the agents take turns executing the commands will be different.
- Lists are ordered collections of data.
- We can have a list of numbers
- A list of words, even a list of lists, also even a list of agents.
- Since they are in a list, these agents will have a particular order, so you can use the list to execute their commands in any order you might like.

Summary

- We have examined the basis of complex results in the Simple Economy Model
- We have seen how concepts from one domain can be used in a completely different domain

Part of the inhumanity of the computer is that, once it is competently programmed and working smoothly, it is completely honest.

—Isaac Asimov

The perfect journey is never finished, the goal is always just across the next river, round the shoulder of the next mountain. There is always one more track to follow, one more mirage to explore.

—Rosita Forbes

Sailors on a becalmed sea, we sense the stirring of a breeze.

—Carl Sagan

Characteristic Features

There are four characteristic features of agent-based modeling:

1. Simple rules can be used to generate complex phenomena

2. Randomness in individual behavior can result in consistent patterns of population behavior

3. Complex patterns can " self-organize " without any leader orchestrating the behavior

4. Different models emphasize different aspects of the world

1. Simple rules can be used to generate complex phenomena

Many of the models that we will look into have very simple rules that do not require complex mathematical formulas or a deep understanding of the knowledge domain that they are attempting to model. Nonetheless, they are able to reproduce complex phenomena that are observed in the real world. For instance, a model of fire spread may have only a simple rule to describe fire spread from one tree to another, but it still may have interesting things to say about how likely a fire is to spread across an entire forest.

2. Randomness in individual behavior can result in consistent patterns of population behavior

It is common for people when they see an ordered population level behavior such as a flock of birds to assume that there must be deterministic processes that govern the behavior

of the individuals (Wilensky & Resnick, 1999). In the case of the birds, people tend to believe that there must be specific social rules or communications that tell each bird how

to place itself in the flock. However, nature has some surprises for us: Many times the individual level rules are quite simple (see point 1) and do not necessarily tell the bird where to position itself in the flock. Instead, the rules often contain a certain amount of nondeterminism and are robust to perturbations in the initial conditions. Despite the stochastic nature of these systems, they can still result in the generation of predictable high-level behavior like the flocking of birds.

Characteristic Features of ABM

3. Complex patterns can "self-organize" without any leader orchestrating the behavior

Similarly, it is common for people when they see a flock of birds, to assume that there must be a leader who orchestrates the behavior — a leader bird who tells each follower bird what to do (Resnick & Wilensky, 1993 ; Resnick, 1994a; Wilensky & Resnick, 1999).1 However, nature again surprises: a population of individuals, each following very simple rules, can " self-organize, " generating complex and beautiful patterns without any orchestrator or centralized controller — these patterns are called "emergent" (Wilensky & Reisman, 2006).

4. Different models emphasize different aspects of the world

Even after we have completed a good working model of a particular phenomenon, we have not finished with the modeling process. Every model foregrounds certain aspects of

the world and backgrounds other aspects. There can be many models of the same phenomenon, and they may each have something interesting to say about the way the world works. For instance, a model of residential location preferences may emphasize how likely a person is to move into a neighborhood based on how she likes the other people in the neighborhood. Such a model may have very interesting things to say about how urban populations develop, but it may say nothing about school districts, location of retail businesses, or development of parks, all of which are also affected by residential location preferences.

The Fire Model

Critical Threshold or Tipping Point: Many complex systems tend to exhibit a phenomenon known as a " critical threshold " (Stauffer & Aharony, 1994) or a " tipping point " (Gladwell, 2000).

Tipping Point: Essentially, a tipping point occurs when a small change in one parameter results in a large change in an outcome.

Tipping Point Example: One model that clearly contains a tipping point is the early agent-based model of a ***forest fire***.

Forest Fire Model: This model is easy to understand, yet exhibits some interesting behavior.

- Besides being interesting in its own right, the model of forest fire spread is highly relevant to other natural phenomena such as the spread of a disease, percolation of oil in rock, or diffusion of information within a population (Newman, Girvan & Farmer, 2002).
- This simple model is highly sensitive to one parameter. When observing the resultant outcome of whether or not a fire will burn from one side of a forest to another (percolate), the output is mainly dependent on the percentage of the ground that is covered by trees (see figure 3.1). As this parameter increases, there will be little to no

effect on the system for a long time, but then all of a sudden, the fire will leap across the world. This is a "tipping point" in the system.

Figure 3.1
A forest fire consuming trees in a forest.

Benefits of Knowing that a System has a Tipping Point: Knowing that a system has tipping points can be useful for a variety of reasons.

1. First, if you know a system exhibits a tipping point, you know that continuing to put effort into the system, even if you are not seeing any results yet, may yet bear fruit.
2. Second, if you know where the tipping point is, and if you know how close you are to it, then you can determine whether or not it is worth putting additional effort into the system.
3. If you are far away from the tipping point, then it might not be worthwhile trying to change the state of the system, whereas if you are close to the tipping point it may take only a small amount of effort to make a big change in the state of the system.

Description of the Fire Model: The Fire model arose from a number of independent efforts to understand *percolation phenomena*.

Percolation: In percolation, a substance (such as oil) moves through another material (such as rock), which has some porosity.

Broadbent and Hammersley (1957) first posed this problem, and since then many mathematicians and physicists have worked on it. Influenced by cellular automata models, they introduced a percolation model using the example of a porous stone immersed in a bucket of

water. The question they focused on was: What is the probability that the center of the stone becomes wet?

A fire moving through a forest can be thought of as a kind of percolation where the fire is like the oil and the forest is like the rock, with the empty places in the forest analogous to the porosity of the rock. A similar question to Broadbent and Hammersley's is: If you start with some burning trees on one edge of the forest, how likely is the fire to spread all

the way to the other side of the forest? Many scientists created and studied such fire models. In 1987, the Danish physicist and complex systems theorist Per Bak and his colleagues showed that the spread of the fire depended on a critical parameter, the density of the forest. Because this parameter arises naturally, the complexity of the fire can arise spontaneously and was therefore a possible mechanism to explain the world's naturally arising complexity. Bak and his colleagues called this phenomenon "self-organizing criticality" and demonstrated it in a number of contexts including, famously, the emergence of avalanches in sand piles.

Herein, we will explore a version of the Fire model adapted by Wilensky (1997a) that is developed in the NetLogo modeling language and is distributed in the Earth Science section of the NetLogo models library.

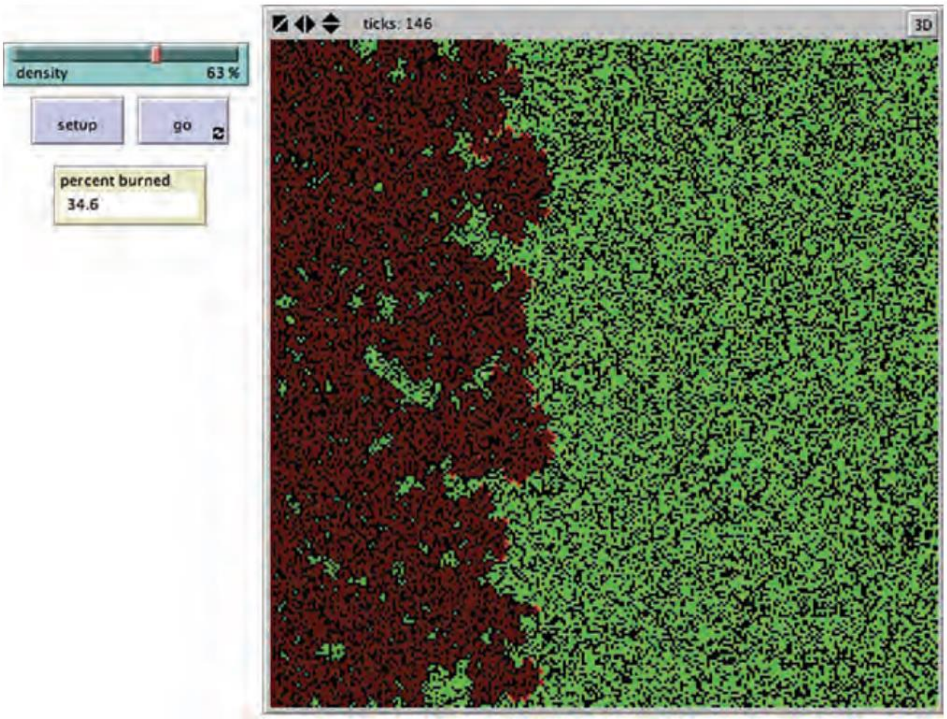


Figure 3.2
NetLogo Fire Simple Model. Based on NetLogo Fire model (Wilensky, 1997). http://ccl.northwestern.edu/netlogo/models/Fire.

This version of the Fire model contains only patches, no turtles. The patches can have four distinct states. They can be (1) green, indicating an unburned tree, (2) red, indicating a burning tree, (3) brown, indicating a burned tree, or (4) black, indicating empty space (illustrated in figure 3.2). When the model is first set up the left edge of the world is all red, indicating that it

is “on fire.” When the model starts running, the fire will ignite any “neighboring” tree — that is, a tree to its right, left, below, or above it that is not already

burned and not already on fire. This will continue until the fire runs out of trees that it can

ignite. The only “control parameter” in this model is the density of trees in the world. This density parameter is not an exact measure of the number of trees in the world. Rather, it

is a probability that determines whether or not each patch in the world contains a tree.

Because the density is probabilistic and not deterministic, even if you run the model multiple times with the same density setting, you will get different results.

Understand the code of the Forest Fire Model

Let us examine the simple rules that govern the Fire model. The code that initializes the model is as follows:

```
to setup
    clear-all
       ;; make some green trees
    ask patches [
       if (random 100) < density
           [ set pcolor green ]
       ;; make a column of burning trees at the left-edge
       if pxcor = min-pxcor
           [ set pcolor red ]
    ]
    ;; keep track of how many trees there are
    set initial-trees count patches with [pcolor = green]
    reset-ticks
end
```

Just as in the Life Simple model we saw earlier, the Fire Simple model will use only stationary agents, patches, and no moving agents, turtles. Besides these basic types of agents, agent-based models can have many other types, including user created agent types.

Explanation of code of the Forest Fire Model

- The first line of the procedure is a command, CLEAR-ALL (or CA for short in the NetLogo language), which resets the world to its initial state — it resets the model’ s clock, kills all moving agents, and restores the default values to the stationary agents.
- The rest of this code issues commands to the patches. First, it populates the world with trees, and second, it makes a column of burning trees (by setting their color to red, the indication that they are on fire) at the left edge of the world. We have made many

modeling choices here. Foremost among these are (1) modeling empty space in the forest as black patches, (2) modeling nonburning trees in the forest as green patches, and (3) modeling fire as burning trees, which are represented as red patches.

- Once these modeling choices are made, we can write the code to populate the model. To set up the trees, we have asked the patches to perform:

```
if (random 100) < density
    [ set pcolor green ]
```

To help us in understanding the setup code, suppose that the density is set to 50. This code tells each patch to roll a hundred-sided "die" (or, an alternative metaphor, spin a spinner with 100 equal sectors). Let us take the point of view of a patch. If I' m a patch, I throw the die. If that die comes up less than 50, then "I become a tree," otherwise "I don't do anything." So, half of the time I will become a tree (turn green) and the other half of the time I'll remain black. Note that each patch throws its own independent die so theoretically all of them (or none of them) could become trees. But, on average, half will become trees. If the density were higher or lower, the same code would work to populate the model with roughly the appropriate tree density. This trick of taking the point of view of the agent, what we called ***agent-centric thinking*** earlier, is one we will employ frequently and is a good habit to form for facility with ABM.

Once the model is set up or initialized, we must define what the model does at each "tick" of the clock. This is typically done in a procedure named GO.

The code for the GO procedure is then as follows:

```
to go
    ;; stop the model when done
    if all? patches [pcolor != red]
       [ stop ]

    ;; ask the burning trees to set fire to any neighboring non-burning trees
    ask patches with [pcolor = red] [ ;; ask the burning trees
       ask neighbors4 with [pcolor = green] [ ;; ask their non-burning neighbor trees
       set pcolor red ]   ;; to catch on fire
    set pcolor red - 3.5    ;; once the tree is burned, darken its color
]
tick  ;; advance the clock by one "tick"
end
```

Now run the Fire Simple model several times. If we run it with a low density of trees, we will see, as expected, very little spread of the fire. If we run it with a very high density of trees, we will see, as expected, the forest being decimated by the inexorable march of the fire. What should we expect at medium densities? Many people surmise that if the density is set to 50 percent, then the fire will have a 50 percent probability of reaching the right edge of the forest. If we try it, however, we see that at 50 percent density, the fire does not spread much. If we raise it to 57 percent, the fire burns more, but still doesn't usually reach the other side of the forest (see figure 3.3).

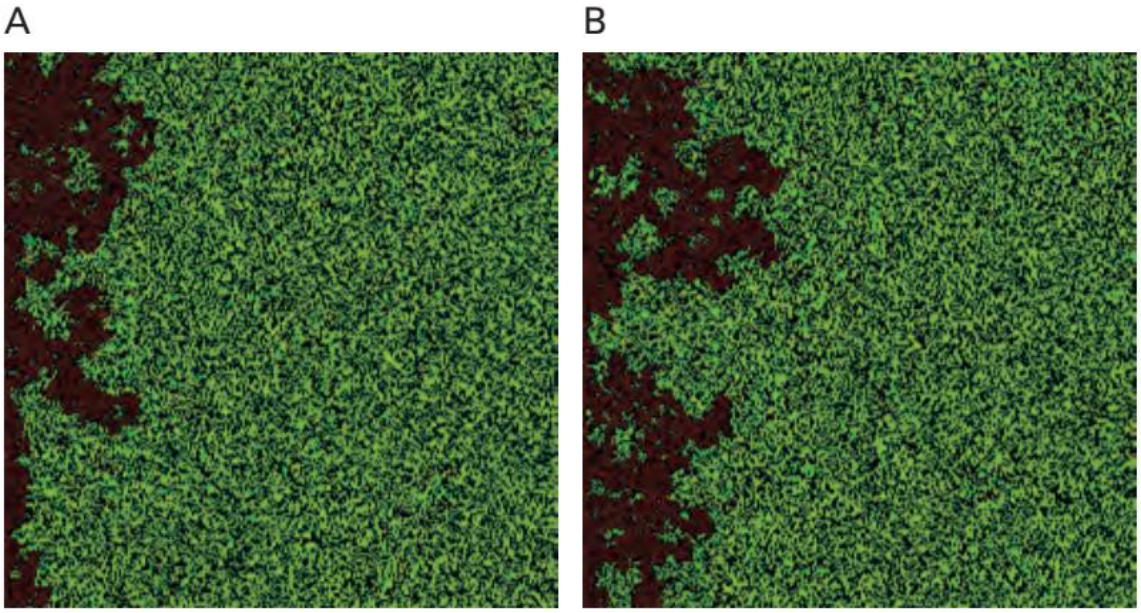


Figure 3.3
Two typical runs of the Fire Simple model with density set to 57 percent.

However, if we raise the density to 61 percent, just 2 percent more density, the fire inevitably reaches the other side (see figure 3.4).

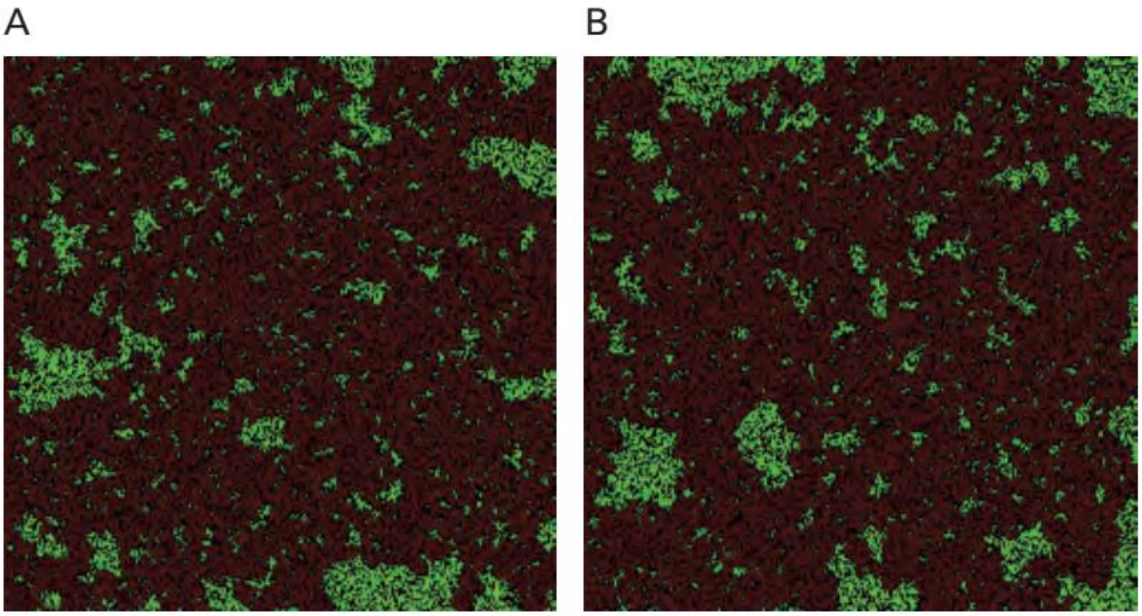


Figure 3.4
Two typical runs of the Fire Simple model with density set to 61 percent.

This is unexpected. We would expect a small change in density to have a relatively small effect on the spread of the fire. But, it turns out, the Fire model has a "critical parameter" of 59 percent

density. Below 59 percent density the fire does not spread that much; above it, it spreads dramatically farther. This is an important and prevalent property of complex systems: They exhibit nonlinear behavior where a small change in input can lead to a very large change in output.

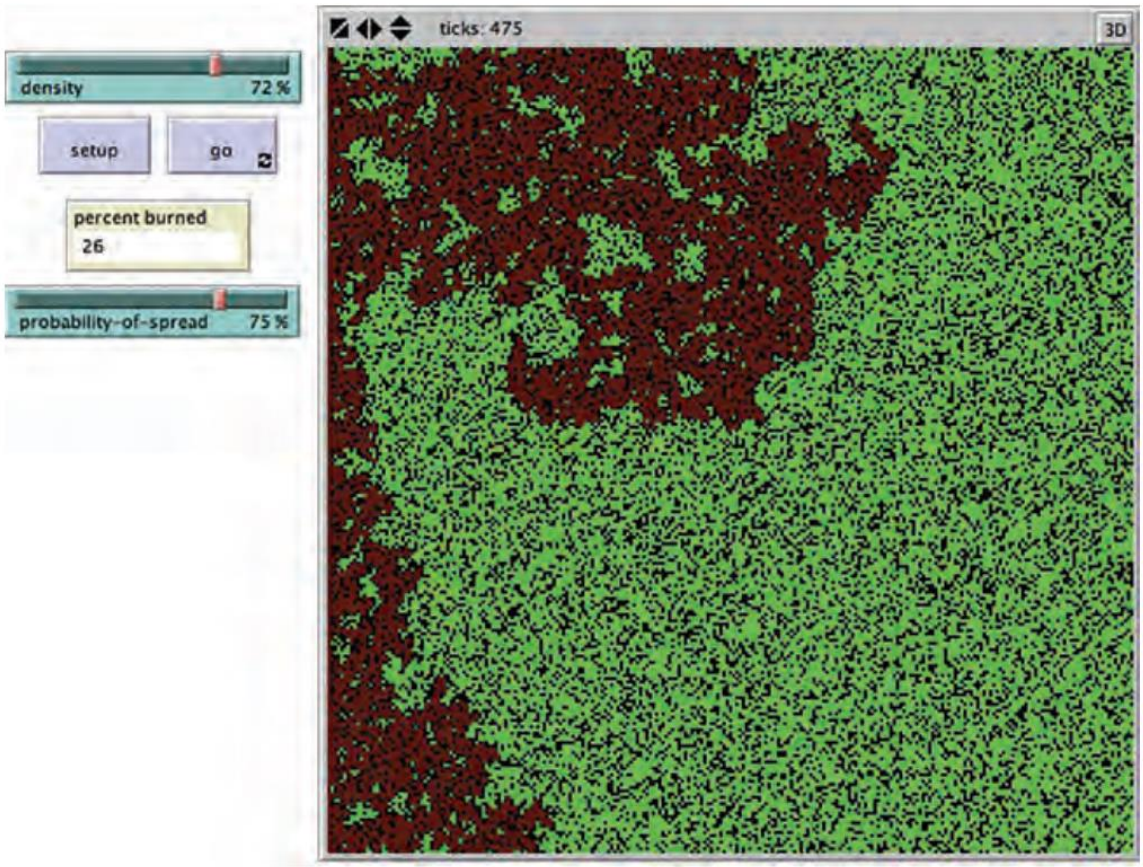


Figure 3.5
NetLogo Fire Simple model after first extension.

Second Extension: Adding Wind

Sometimes we can take an ABM for which we used a probability to model a suite of mechanisms and refine it so that some of those mechanisms are modeled in a more physical way. Our first extension hides a host of possible mechanisms in the PROBABILITYOF-SPREAD. We can pull out one of those mechanisms and model it in a more refined way. Wind is a good example of a process that we can model more specifically. We can think of the effect of wind on a fire as increasing the chance of fire spread in the direction it is blowing, decreasing the chance of fire spread in the direction it is not blowing, and having little effect on fire spread that occurs perpendicular to the direction of its movement. Of course, this too is an oversimplification — there are often local effects of wind, such as turbulence. As we have seen, all models are simplifications, and playing the modeling game means we will accept this one for now.

To implement wind in our model, we will create two sliders. One will control the speed of the wind from the south (a negative value will indicate a wind from the north) and one will control the speed of the wind from the west (a negative value will indicate a wind from the east). We create these two sliders and set them up to go from – 25 to 25. How do we use these new parameters in the code? If we think back to our first modification, we want these new parameters to affect the PROBABILITY-OF-SPREAD.

This effect will be based on the direction that the fire is attempting to spread. Since we have set the wind speed to vary from – 25 to 25, we could conceive of those numbers as percentages by which to modify the probability of spread which is expressed as a percent. In order to do this, let us first create a local variable called PROBABILITY that will initially be set to PROBABILITY-OF-SPREAD. A local variable is one that has a value only within a limited context, usually inside the procedure in which it is defined. If we only need to reference a variable in one procedure, then it is best to use a local variable. But, remember, you will not be able to see the value of the variable elsewhere in your program or in the command center. We define a local variable with the “let” primitive and can modify it with the “set” primitive. We can modify PROBABILITY to take into account WIND-SPEED by increasing or decreasing it by the WIND-SPEED in the direction the fire is burning. When we put this all together, we get the following code:

```
to go

    if all? patches [pcolor = red]
         [ stop ]

    ;; each burning tree (red patch) checks its 4 neighbors.
    ;; If any are unburned trees (green patches), change their probability
    ;; of igniting based on the wind direction
    ask patches with [pcolor = red] [
         ;; ask the unburned trees neighboring the burning tree
         ask neighbors4 with [pcolor = green] [
             let probability probability-of-spread     ;; define a local variable

             ;; compute the direction you (the green tree) are from the burning tree
             ;; (NOTE: "myself" is the burning tree (the red patch) that asked you
             ;; to execute commands)
             let direction towards myself
             ;; the burning tree is north of you
             ;; so the south wind impedes the fire spreading to you
             ;; so reduce the probability of spread
             if (direction = 0 ) [
                  set probability probability - south-wind-speed ]

             ;; the burning tree is east of you
             ;; so the west wind impedes the fire spreading to you
             ;; so reduce the probability of spread
             if (direction = 90 ) [
                  set probability probability - west-wind-speed ]

             ;; the burning tree is south of you
             ;; so the south wind aids the fire spreading to you
             ;; so increase the probability of spread
             if (direction = 180 ) [
                  set probability probability + south-wind-speed ]

             ;; the burning tree is west of you
             ;; so the west wind aids the fire spreading to you
             ;; so increase the probability of spread
             if (direction = 270 ) [
                  set probability probability + west-wind-speed ]
             if random 100 < probability [ set pcolor red ]
         ]
         set pcolor red - 3.5
    ]
    tick
end
```

This code is a little tricky. Essentially, what it does is modify the probability of spread, increasing it in the direction of the wind and decreasing it in the opposite direction. It calculates the change in the probability by first determining which direction the fire is trying to spread and then determining which of the winds will affect it. Once this probability is calculated, it is then used to determine whether or not the fire spreads to the neighboring tree.

This modification can lead to quite interesting patterns of spread. For example, set the density at 100 percent, and make the wind blow strong from the south and west. At the same time, set the PROBABILITY-OF-SPREAD fairly low, say around 38 percent. This

creates a triangular spread, and, all else being equal, the fire should spread to the northeast.

(See figure 3.6.)

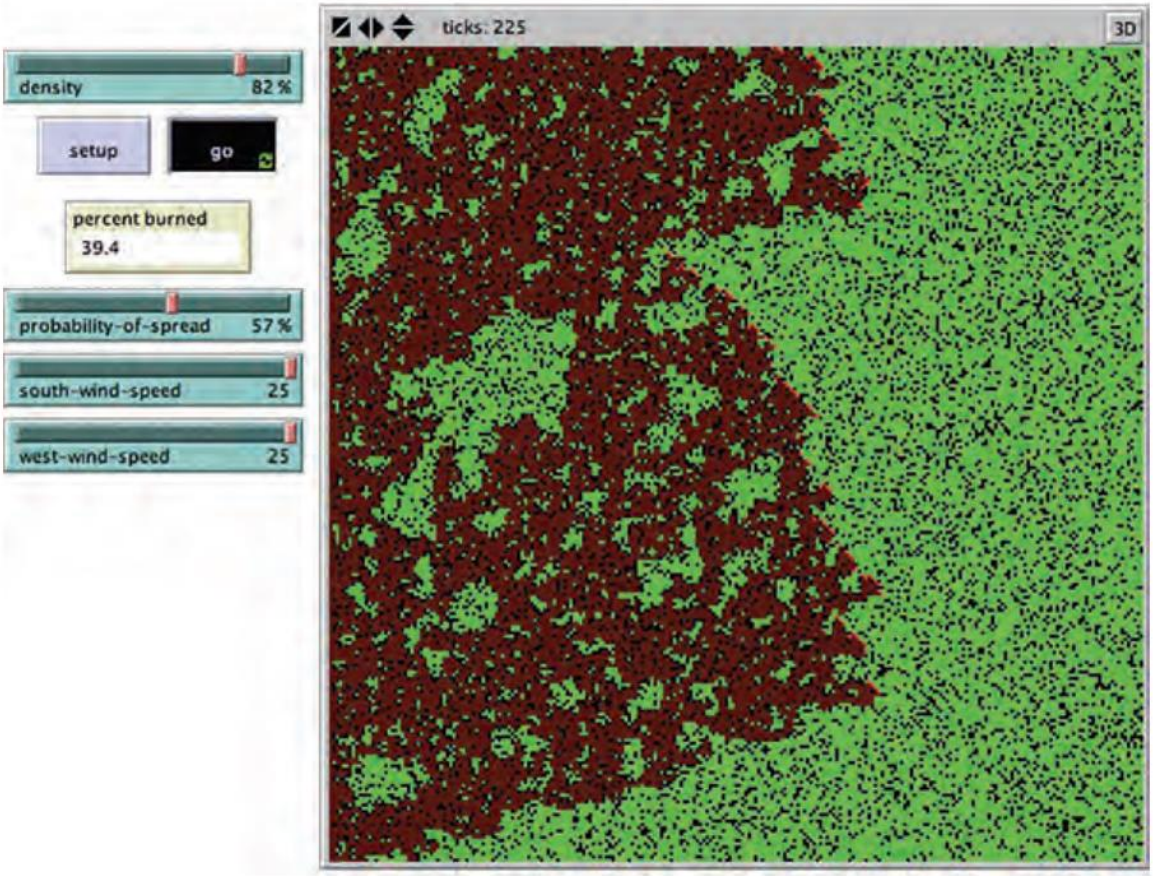


Figure 3.6
NetLogo Fire model after second extension.

Third Extension: Allow Long-Distance Transmission

We have modeled the effect of wind as pushing the fire to spread in one direction. Another possible effect of wind is to enable the fire to jump over long distances and start fires where there are no surrounding burning trees. That might be an interesting process to model. In order to make sure the revised model can replicate our old results, we add a switch to control the jumping. We add a Boolean (TRUE or FALSE) switch labeled BIG-JUMPS? Switches in NetLogo provide a way to have direct control over variables that can only be true or false.

This particular switch allows us to turn on and off the jumping behavior. With the switch

off, the revised model should produce the same results as Fire Simple Extension 2. One way to model fire jumping due to wind would be to ignite a new fire at some distance in the direction of the wind. We can do this by modifying the code inside our earlier GO procedure as follows picking up after we do all of the changes in probability based on wind:

```
…
if random 100 < probability [
                    set pcolor red

                    ;; if big-jumps is on, then sparks can fly farther
                    if big-jumps? [
                              let target patch-at ( west-wind-speed / 5 ) ( south-
                              wind-speed / 5 )
                              if target != nobody and [pcolor] of target = green [
                              ask target [
                                  set pcolor red ;; ignite the target patch
                              ]
                              ]
                    ]
          ]
…
```

The first part of this code turns the current patch to red, which is the same as the previous version, but the second part after IF BIG-JUMPS? is what has changed. If BIG-JUMPS? is true then it looks for a target patch that is some distance away in the direction of the wind. If there is an unburned (green) tree at that location, then that tree is also set on fire (due to the spark having landed there). A more detailed model could include explicit spark agents, which travel according to the wind, and catch trees on fire when they land, but in this case, we just model the effects of flying sparks without modeling the sparks themselves.

Insert this code and see what effect it has on the model. You start to see lines in the direction of the wind as it jumps across gaps in the forest (as shown in figure 3.7). This extension can have visually dramatic effects. Explore different sets of parameters and observe the different patterns of spread you can get. This modification increases the probability that the fire will reach the right edge of the world (our measure), but the resultant pattern is no longer the same; there are big chunks of the world that are not burned anymore. As new patterns emerge, we may need to reevaluate the questions that drove us to create the model, which may change the measures we collect about the model.

Each extension can lead to many more questions, which in turn call for new extensions.

In our third fire extension, it is of course also possible to have big jumps be probabilistic in the same way that the neighboring spreads are, or to have the sparks from the big jumps land in a random rather than a determined location.

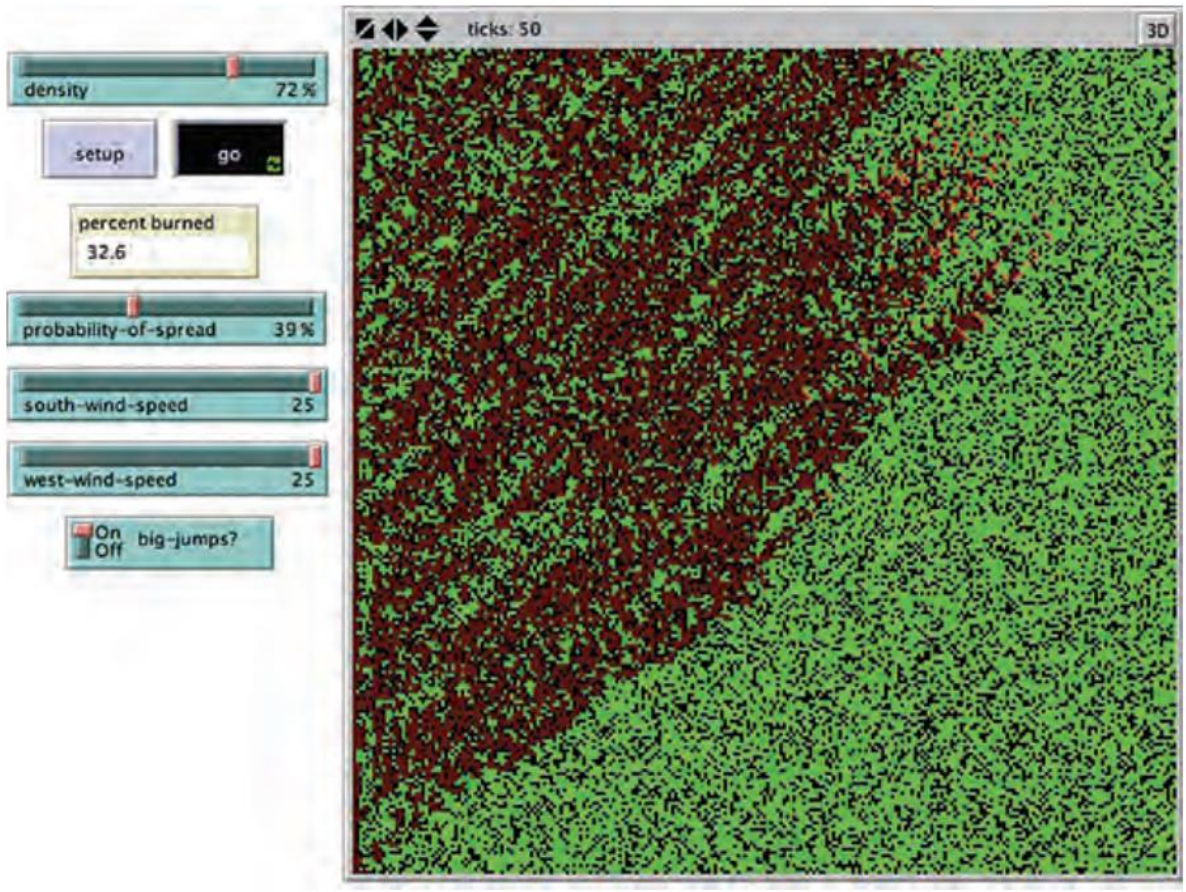


Figure 3.7
NetLogo Fire model after third extension.

Summary of the Fire Model

We have made three different changes to the model, which have affected our tipping point in three different ways.

1. The final change to the model points out that the measure we were observing (whether or not the fire made it to the right edge of the world) might not even be the measure we want to observe.
2. As we change the fire spread mechanisms, other measures such as "percent burned" may become more important. These new measures may or may not have tipping points. This illustrates an important feature of tipping points.

3. They are defined by an input and an output measure. A model does not have a tipping point in and of itself — the tipping point is relative to the choice of inputs and output measures.

The Diffusion-Limited Aggregation (DLA) Model

As we discussed earlier, the ABM perspective enables a new and different understanding of complex systems. As demonstrated with the Fire Simple model, it is possible to think about inanimate objects (such as trees) as agents. By reconceptualizing atoms and molecules as agents, and then developing procedural rules to describe how these agents interact we can gain a deeper understanding of many different types of physical phenomena.

For instance, the formation of complex beautiful patterns in nature has mystified and amazed humans for many years (see figure 3.9), yet many of these patterns can be generated using simple rules. Often in ABMs we will see that the complexity of a resulting pattern bears little direct relationship to the complexity of the underlying rules.

Figure 3.9
A DLA copper aggregate formed from a copper sulfate solution in an electrode position cell (Kevin R. Johnson, 2006, http://commons.wikimedia.org/wiki/File:DLA_Cluster.JPG).

In fact, in some cases the results will seem to be exactly the opposite; simpler rules will often create more complex patterns. The focus of our interest should not necessarily be on the complexity of the rules, but instead on the interaction those rules produce. Many agent-based models are interesting not because of what each agent does, but because of what the agents do together.

In this section, we will look at a very simple model where all that the agents do is move

randomly around the world, and eventually stop moving when a basic condition is met;

despite the simplicity of the rules, interesting complex phenomena can emerge.

Description of Diffusion-Limited Aggregation

In many physical processes from the creation of clouds, to snowflakes, to soot, smoke, and dust, particles aggregate in interesting ways.

- Diffusion-limited aggregation (DLA) is an idealization of this process, and was first examined as a computational model in the early eighties (Witten & Sander, 1981, 1983).
- DLA models were able to generate patterns that resemble many found in nature, such as crystals, coral, fungi, lightning, and growth of human lungs, as well as social patterns such as growth of cities (Garcia-Ruiz et al., 1993; Bentley & Humphreys, 1962; Batty & Longley, 1994).

Netlogo Model of DLA

A NetLogo version of diffusion-limited aggregation was one of the first models written for the NetLogo models library (Wilensky, 1997b). This model starts with a large number of red particle agents and one green patch in the center. The green patch at the center of the screen is stationary, but when you press GO, all of the red particles move randomly by "wiggling" left and right (resulting in a small random turn) and then moving forward one step. After a particle moves, if any of its neighbors (i.e., nearby patches) are green, the particle dies, turning the patch it is on green (the particle can do so because all turtles have direct access to the variables of the patch they are on). If you let the model run long enough, you will get an interesting fractal-like pattern forming from the green patches (see figure 3.10).

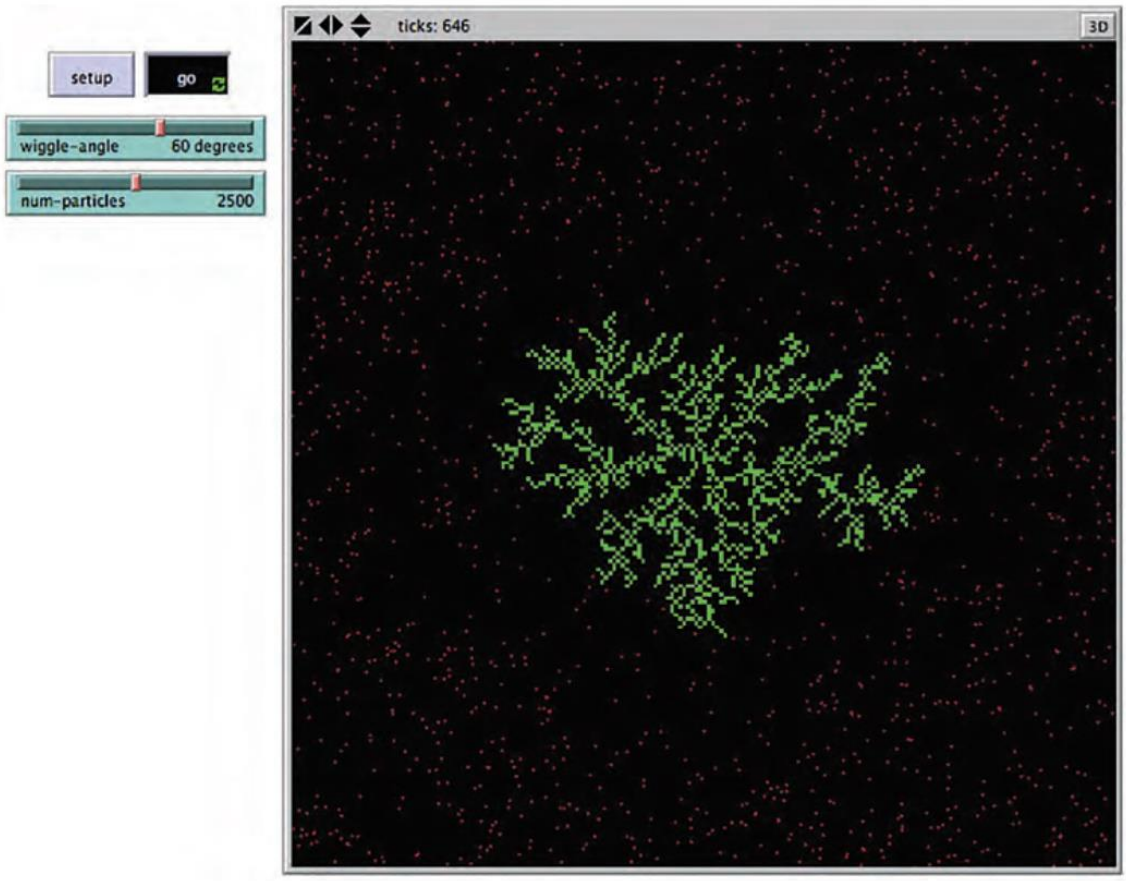


Figure 3.10
NetLogo model of diffusion limited aggregation. http://ccl.northwestern.edu/netlogo/models/DLA (based on *Wilensky, 1997b*).

Here is the NetLogo code for the GO procedure for DLA Simple. WIGGLE-ANGLE is a global variable, the value of which is set by a slider in the interface.

```
to go
    ask turtles
       ;; turn a random amount right and left
       [ right random wiggle-angle
           left random wiggle-angle
           forward 1
           ;; if you are touching a green patch, turn your own patch green and
           ;; then die
           if ( any? neighbors with [pcolor = green] )
              [ set pcolor green
                  die ] ]
    tick
end
```

As the code shows, if any of the neighbors are green the particle stops moving, changes the color of the patch that it is on to green and dies. Turtles have direct access to the variables of the patch they are on — that is how the turtle can set the color of the patch it

is on (the PCOLOR in NetLogo).

We will now create three extensions of the DLA Simple model.

First Extension: Probabilistic Sticking

The DLA Simple model has very simple rules. The code consists of only two procedures, and both of them are very small, yet these two procedures can generate many interesting results. In these extensions we will examine how adding a few more simple rules can enable us to generate more interesting patterns.

After you run the model for a while, you will realize that it always produces thin and wispy types of structures. Often the “stems” and “trunks” of the structures are only a single patch wide. This is because as soon as a particle touches anything green it will stop moving.

It is much more likely that it will touch something toward the edge of the structure rather than near the interior of the structure.

However, we can change this. One way to think of this system is that if a particle comes into contact with a stationary object, then it becomes stationary itself with 100 percent probability. What if we decrease this probability? That is exactly what we will do in this extension. We will allow the user to control the probability of a particle becoming stationary.

In the original code, if any of the neighbors are green, the particle stops moving, changes the color of the patch that it is on to green, and dies. What we need to do is add another test to this rule, but we also have to make one other change to the code at the same time.

Since stopping will be probabilistic, it is now possible that a particle could be on top of another green particle, and, in that case, we do not want the particle to stop. So, we also need to make sure we are not on a green particle before we stop, which involves adding an additional condition to the model:

```
to go
    ask turtles
       [ right random wiggle-angle
           left random wiggle-angle
           forward 1
           if (pcolor = black) and ( any? neighbors with [pcolor = green] )
    and ( random-float 1.0 < probability-of-sticking )
              [ set pcolor green
                                    die ] ]
    tick
end
```

Now, a particle will only stick if a random number it generates is less than a given probability, but where is this probability defined? If you add this code into the GO procedure and then move to the Interface tab, NetLogo will generate an error because we have not defined this parameter yet. So we will do that now. We go back to the Interface tab and create a slider called PROBABILITY-OF-STICKING, and we give it a minimum value of 0.0, a maximum value of 1.0 and an increment of 0.01.3 Now we are finished with this extension, and we can run the model with a PROBABILITY-OF-STICKING of 0.5, as seen in figure 3.11.

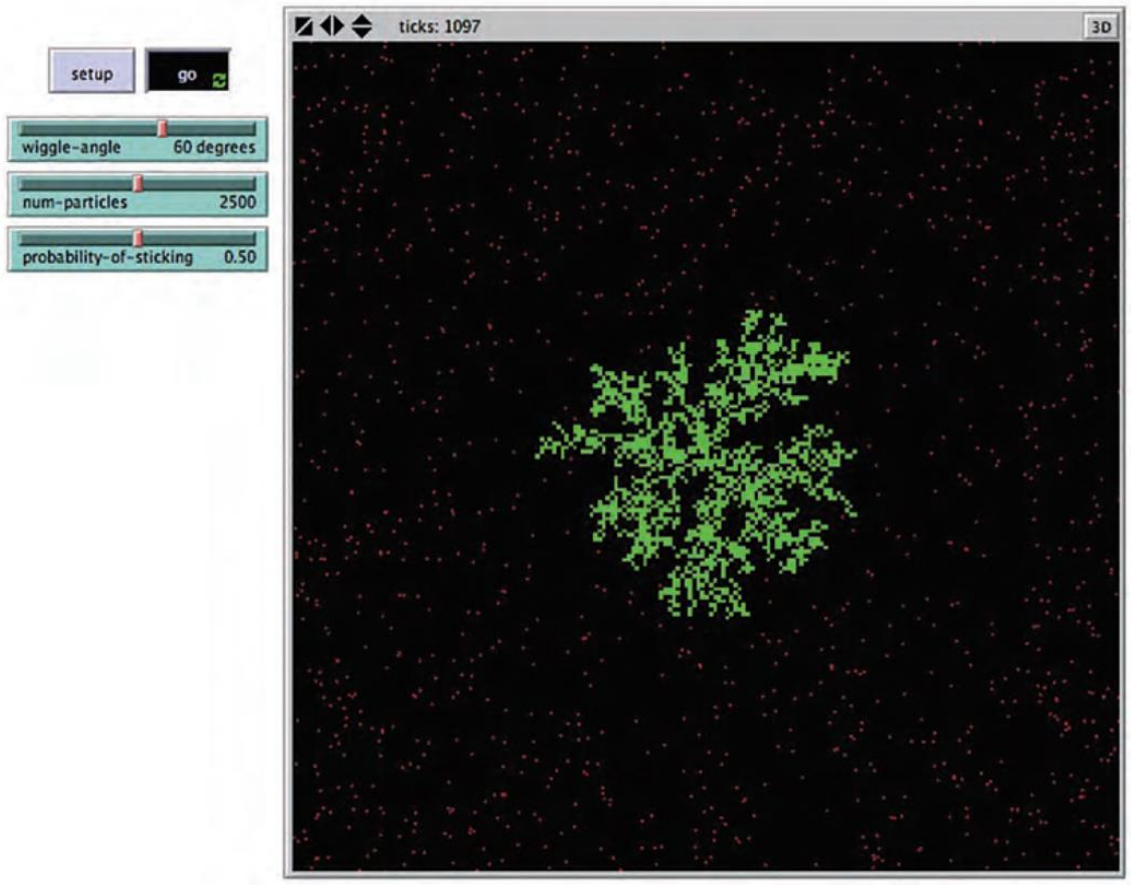


Figure 3.11
DLA model after first extension.

You will notice that the branches of the structure become thicker when the probability is 0.5. This is because particles have a smaller probability of getting stuck on the outside

of the structure and can wander deeper in to the structure before stopping. If you want you can still generate the original results by setting the probability to 1.0. As with the extensions to the Fire Simple model, it is often helpful to make sure that when extending

a model you can still generate the previous results. Not only does this allow you to inspect whether any errors were introduced by your code change, but also, as you continue to add new parameters and mechanisms, you will often want to go back to your original results for comparison.

Second Extension: Neighbor Influence

The addition of our probability parameter enables us to explore a whole new range of structures, while not interfering with our ability to add additional simple rules to this model. One commonly explored extension to the DLA model is to explore how the probability of sticking is related to the number of neighbors that are already stationary (the green patches) (Witten & Sander, 1983). If a particle is moving, it is more likely that it will stick somewhere if two or

three of its neighbors are at rest than if just one of its neighbors is at rest. Thus, we want the probability of stopping to increase as the number of stopped neighbors increases.

Let us begin this extension by adding a switch to our interface, which we will call NEIGHBOR-INFLUENCE? This switch will determine whether or not we are taking the number of neighbors in to account when determining if a particle should stop moving.

NEIGHBOR-INFLUENCE? is a Boolean variable, that is, it has one of two values, TRUE or FALSE. It is conventional to append a question mark character to the end of the variable name to point out to the reader that the variable is Boolean. After we add this switch, we can go back and look at our GO procedure. We left the GO procedure looking like this:

```
to go
    ask turtles
       [ right random wiggle-angle
           left random wiggle-angle
           forward 1
           if ( pcolor = black ) and ( any? neighbors with [pcolor = green] ) and
                 ( random-float 1.0 < probability-of-sticking )
                 [ set pcolor green
                 die ] ]
    tick
end
```

We need to modify the PROBABILITY-OF-STICKING based on the NEIGHBORINFLUENCE? switch. To accomplish this, we must first create a variable, which we will call LOCAL-PROB If NEIGHBOR-INFLUENCE? is turned off then LOCAL-PROB will be the same as PROBABILITY-OF-STICKING. However, if NEIGHBOR-INFLUENCE? Is turned on, then we will reduce the probability of sticking for small numbers of green neighbors. We accomplish this by multiplying the PROBABILITY-OF-STICKING by the fraction of its eight neighbors that are green. For example, if PROBABILITY-OF-STICKING is set to 0.5 and if a particle encounters a neighborhood with four green neighbors, then we multiply PROBABILITY-OF-STICKING by 4/8 to get a sticking probability for that encounter of 0.25. We do this in order to create a relationship such that if many neighbors are green the probability approaches the PROBABILITY-OF-STICKING and if few neighbors are green the probability approaches zero.4 As a further exploration, try some other functions and see how it affects this result; to facilitate this, you may want to make the number of neighbors that affect a particle a parameter. The new code looks like this:

```
        to go
                ask turtles
[ right random wiggle-angle
    left random wiggle-angle
    forward 1
    let local-prob probability-of-sticking

    ;; if neighbor-influence is TRUE then make the probability proportionate
    ;; to the number of green neighbors, otherwise use the slider as before
    if neighbor-influence? [
    ;; increase the probability of sticking the more green neighbors there are
                        set local-prob probability-of-sticking *
(count neighbors with [pcolor = green] / 8)
    ]

    if (pcolor = black) and ( any? neighbors with [pcolor = green] )
and ( random-float 1.0 < local-prob )
    [ set pcolor green
          die ] ]
    tick
end
```

When we run the model with this change in the code, we notice

(1) that it takes much longer to form a structure because there is a much lower probability of a mobile particle stopping, and

(2) the structures that emerge are very thick and almost bloblike. In addition, there are many fewer one patch-wide branches, because the probability of stopping when in contact with only one other stationary patch is very small. You can see the results in figure 3.12 .

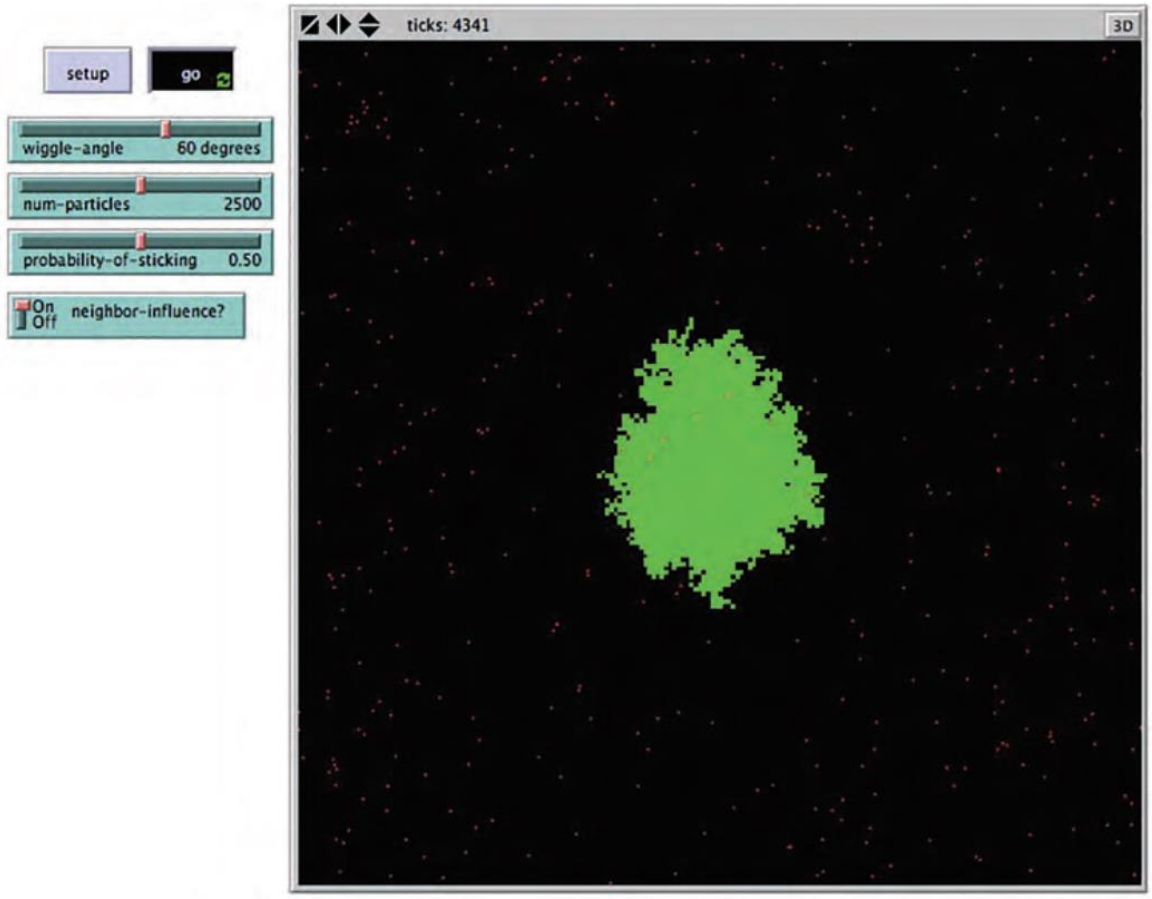


Figure 3.12
DLA model after second extension.

Third Extension: Different Aggregates

The previous two extensions have concentrated on deciding how a particle should stop when it is moving — there is another way that we can control how a particle stops. We can simply give it more places to stop at the beginning. If we look at the SETUP procedure in the previous version of the model, we can see that it creates one green patch in the middle of the world:

```
to setup
    clear-all
    ;; ask the middle patch to turn green
    ask patch 0 0
       [ set pcolor green ]
    create-turtles num-particles
       [ set color red
           set size 1.5   ;; make the particle bigger so it's easier to see
           setxy random-xcor random-ycor ]
end
```

Patch 0 0 is the unique name for the patch that is at x-coordinate 0, and y-coordinate 0, which is at the center of the NetLogo world by default. However, what if we want to create multiple green patches at the start? If there are multiple green patches, then there will be more places for the moving particles to come to rest, and we can generate different patterns of aggregation. To begin with, we need to create a slider so we can control the number of different aggregates that we create; we will call this slider NUM-SEEDS. We give this slider a minimum of 1 (since we need at least one seed), a maximum of 10, and an increment of 1.

Once we have this slider, we can ask NUM-SEEDS patches to turn green in the setup:


```
to setup
    clear-all
    ;; start with NUM-SEEDS green patches as "seeds"
    ask n-of num-seeds patches
       [ set pcolor green ]
    create-turtles num-particles
       [ set color red
           set size 1.5
           setxy random-xcor random-ycor ]
    reset-ticks
end
```

The N-OF reporter selects a random set of NUM-SEEDS patches. We then ask these

randomly chosen patches to turn green, making them seeds for the aggregates. This is different from what we did in the Fire model; in that model, we asked each of the patches to determine if they should become a tree (a seed) using a random variable. The Fire model method will generate roughly the fraction of trees specified by the slider; the N-OF method will always generate exactly the number of seeds specified by the slider. In agent-based modeling, the probabilistic method used in the Fire model is often considered more realistic since nature does not specify exact numbers. However, the N-OF method gives us more precise control over the model's behavior.

This code works, but note that it no longer re-creates the exact results of the original model. Even if we set NUM-SEEDS to 1, we will no longer always create a seed at the center of the world. Instead, the seed will be randomly placed somewhere in the world. In this case it is simpler to deviate from the original version and not much is lost by doing so. There is really nothing special about the center of the world, and having the seed start anywhere randomly will be sufficiently similar to the original result for testing and exploration purposes. If we really desire to have the seed start at

the center of the world, we could add another parameter; this is left as an exercise for

the reader. The final version of the simple DLA model, after all three extensions, is shown in figure 3.13. With multiple seeds, the patterns look somewhat like frost forming on a cold window.

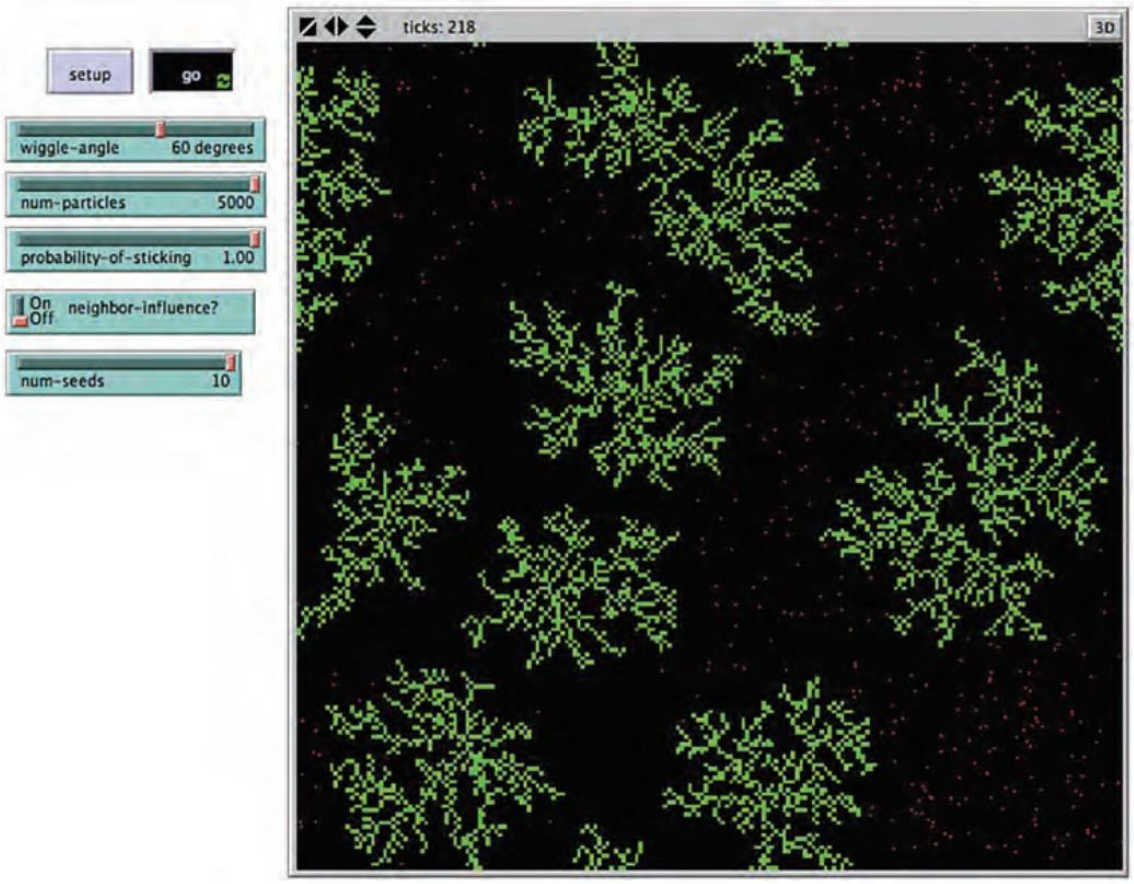


Figure 3.13
DLA model after third extension.

The Segregation Model

Fundamental Approaches to ABM

There are two fundamental approaches to starting to build an agent-based model.

- The first approach, more common in science contexts, is to start with a known phenomenon to be modeled. This approach is called ***phenomena-based modeling***. Usually when engaged in phenomena-based modeling, we have an aggregate pattern, termed a *reference pattern* that we are trying to generate with agent rules.
- Another way to begin is to start with some simple rules and play them out to see what patterns develop. This approach is sometimes referred to as ***exploratory modeling***.

Frequently in pursuing ABM, we take some combination of these two approaches.

Schelling Model

In his investigations into the nature of nature of segregation, Thomas Schelling took a more exploratory modeling approach with his Neighborhood Tipping model (1971).

Schelling wondered what would happen if you assume that everyone in the world wanted to live in a place where at least a reasonable fraction of their neighbors were like themselves, i.e., they have an aversion to being an extreme minority (which he called weakly prejudiced; see Anas, 2002). He explored this model using a checkerboard as a grid, with pennies to represent one race and dimes to represent another. He counted by hand the number of pennies surrounding each dime and divide by the number of neighbors surrounding that dime. If this value exceeded a certain threshold value, he would move the dime to a random empty location on the board. He repeated this process hundreds of times and observed the outcome. (See figure 3.15.)

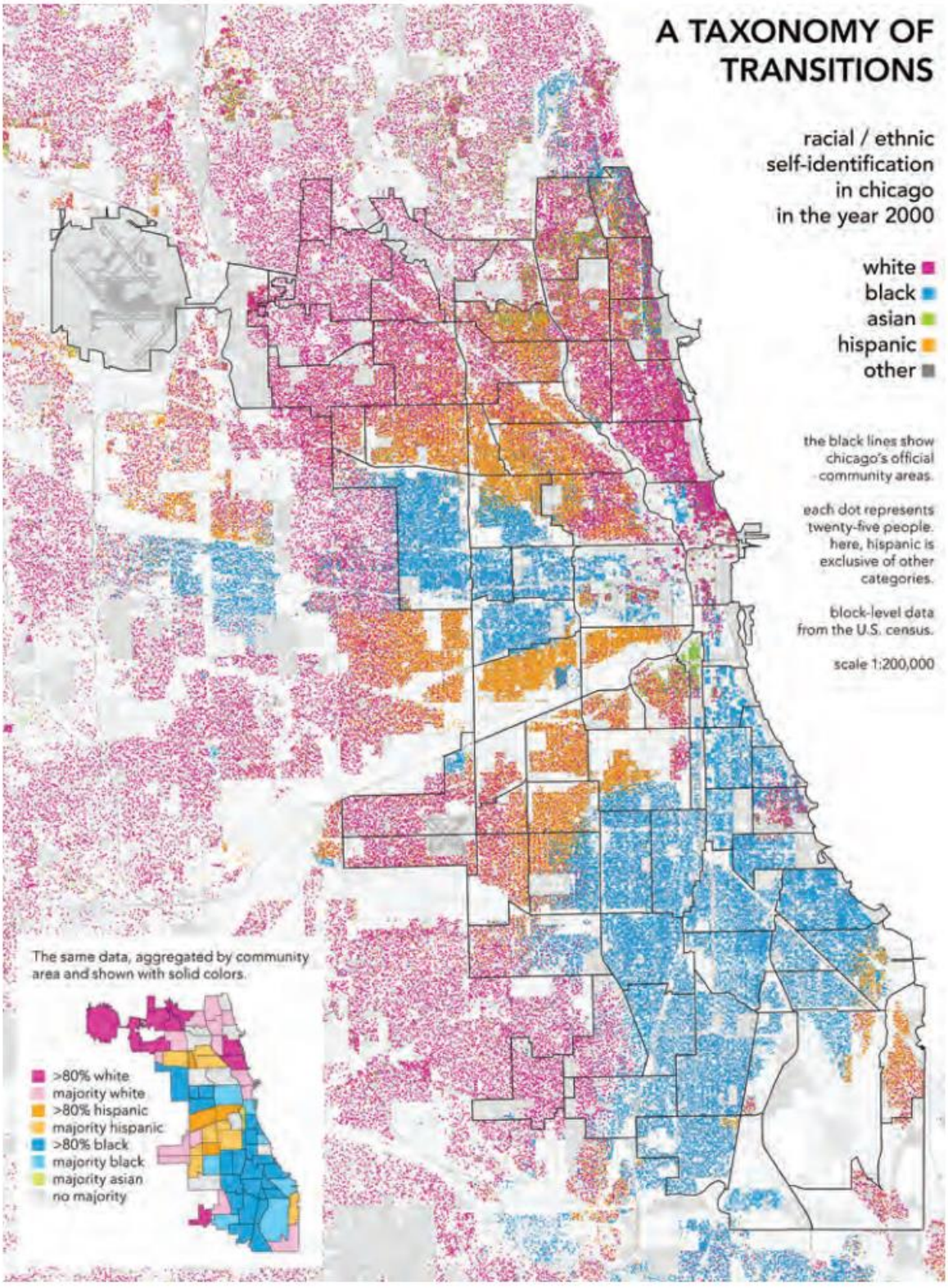


Figure 3.15
A map of Chicago with each dot representing twenty-five people. By representing the data at this finer level than traditional maps that aggregate demographics are mapped at a high level, Rankin illustrates nuances of segregation much like ABM enables us to explore nuances of individual-level behavior.

Schelling Model

For high levels of the threshold, the model confirmed what he predicted. The checkerboard would quickly segregate into areas of all pennies and all dimes. What surprised Schelling about this model (and many other people at the time) was that even at low levels of the threshold, he would still see groups of pennies and groups of dimes together in dense clusters.

This global segregation occurred despite the relative tolerance of each individual. There was no single individual in the Schelling model who wanted a segregated neighborhood, but the group as a whole moved toward segregation. Schelling described this as "macrobehavior" derived from "micromotives" (1978).

At the time when Schelling introduced this model, it was very controversial for several

reasons. First, it was widely believed at the time that housing segregation was caused by

individuals being prejudiced. Schelling' s work seemed to be "excusing" people from

prejudice, saying that prejudice itself is not the cause of segregation. The cause of segregation is an emergent effect of the aggregation of people ' s weak prejudice. But Schelling ' s point was not to excuse people for their prejudice, his model demonstrates that prejudice is not a " leverage point " of the housing system. Unless you can reduce prejudice to close to zero, so that everyone is perfectly comfortable being the sole member of their race in their neighborhood, this segregation dynamic will emerge. So, if your goal is to reduce housing segregation, it may not be effective to work directly on reducing individual ' s prejudice.

Another controversial aspect of Schelling ' s model is that it modeled the people as behaving with very simple rules. Critics argued that "people are not ants" ; they have complex cognition and social arrangements, and you cannot model them with such simple rules.

To be sure, Schelling ' s model was highly oversimplified model of people ' s housing

choices. Nevertheless, it did reveal a hitherto unknown important dynamic. Eventually,

Schelling carried the day. For his large body of work on exploring the relations of micromotives and macrobehavior and his work on conducting game-theory analyses of

conflict and cooperation, Schelling was awarded the Nobel Prize in economics in 2005.

To this day, it remains controversial how much of human behavior can be modeled with

simple rules.

SEGREGATION MODEL

A NetLogo version of the segregation model is distributed in the IABM Textbook section of the models library. This model is shown in figure 3.16 .

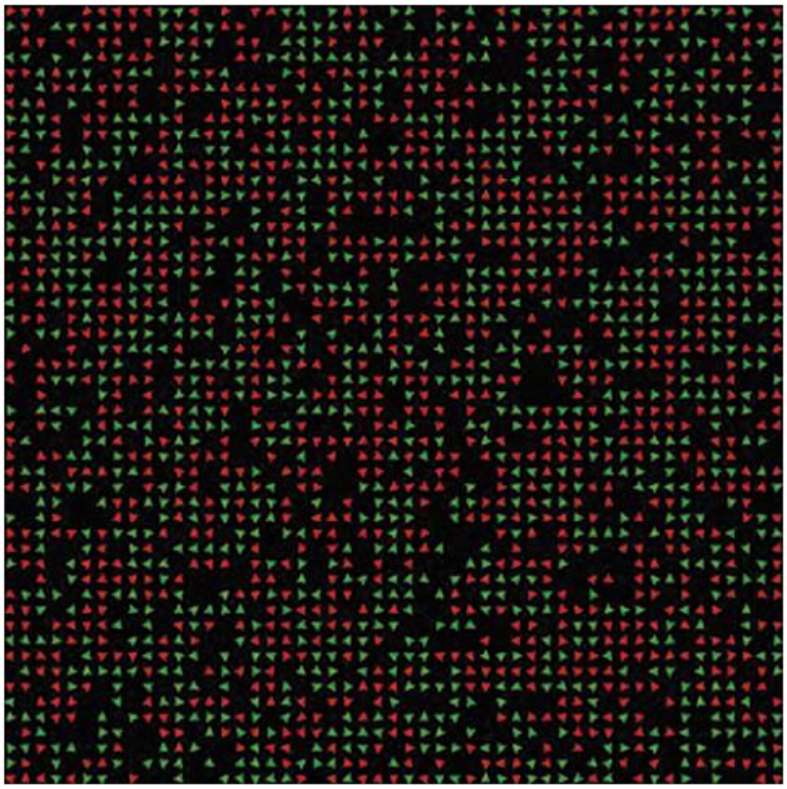


Figure 3.16
Initial state of the NetLogo Segregation model. Red and green turtles are distributed at random. http://ccl.northwestern.edu/netlogo/models/Segregation (*Wilensky, 1997d*).

When you press the SETUP button, an approximately equal number of red and green

turtles will appear at random locations in the world.

Each turtle will determine if it is happy or not, based on whether the percentage of neighbors that are the same color as itself meets the %-SIMILAR-WANTED threshold.

When you press the GO button, the model will check if there are any unhappy turtles.

Each unhappy turtle will move to a new location. It moves by turning a random amount and moving forward a random amount from 0 to 10. If the new location is not occupied then the turtle settles down, if it is occupied, it moves again. After all the unhappy turtles have moved, every turtle again determines whether or not it is happy and the process repeats itself. (See figure 3.17.)

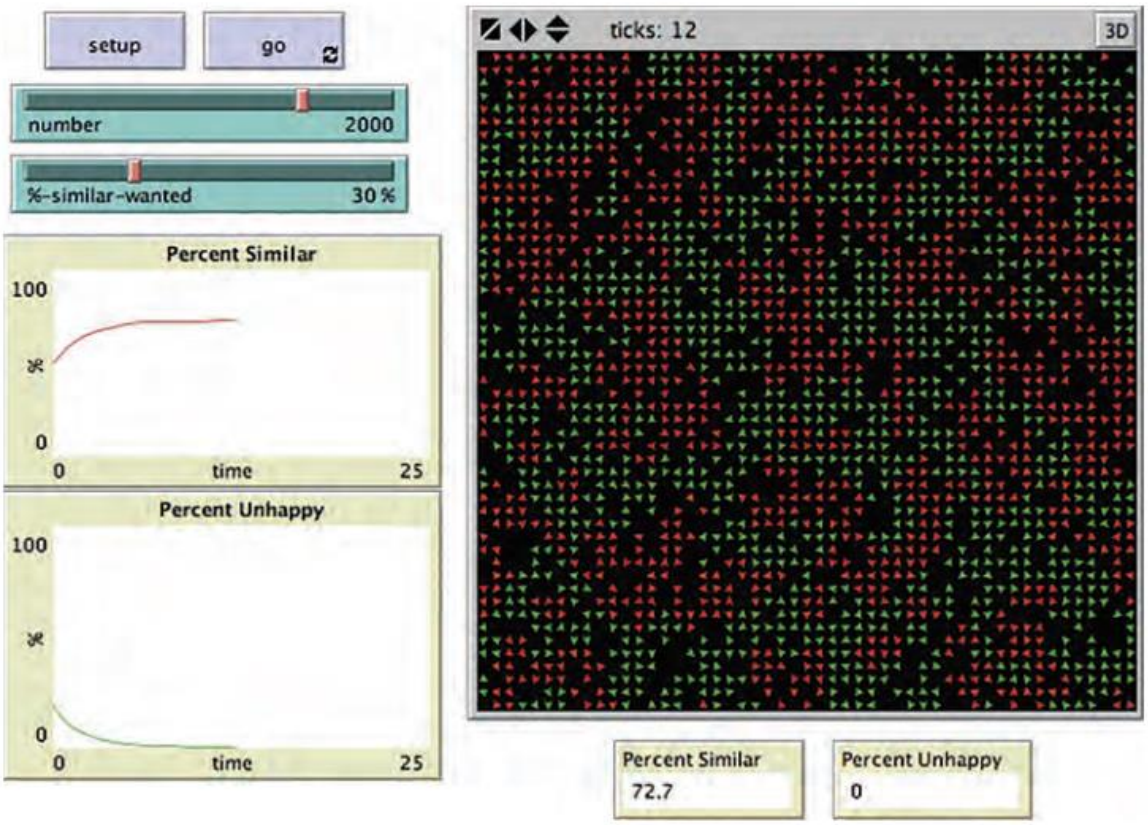


Figure 3.17
A run of the Segregation model with a "tolerance" level of 30 percent. Even though most agents are comfortable in an integrated neighborhood, the housing gets very segregated with 72.7 percent of agents surrounded by their same color agents.

The SETUP procedure for this model is as follows:

```
to setup
    clear-all
    ;; create a turtle on NUMBER randomly selected patches.
    ask n-of number patches
       [ sprout 1 ]

    ask turtles [
       ;; make approximately half the turtles red and the other half green
       set color one-of [red green]
    ]

    update-variables ;; update the turtles and the global variables
    reset-ticks
end
```

The code asks a random set of patches to each sprout a red turtle. The size of the patch set is determined by the NUMBER slider, so after sprouting, the number of turtles is exactly the value on the NUMBER slider, each turtle on its own patch. The code then asks the turtles to turn red or green with equal probability, resulting in an approximately equal number of red and green turtles.

SEGREGATION MODEL CONTINUED

The GO procedure is:

```
to go
    if all? turtles [happy?] [ stop ]
    move-unhappy-turtles
    update-variables
tick
end
```

This code stops the simulation when all the turtles are happy. If there are any unhappy turtles, it asks them to move, and then all turtles recalculate their happiness.

The happiness calculation takes place in the update-turtles procedure, which follows:

```
to update-turtles
   ask turtles [
      ;; in next two lines, we use "neighbors" to test the eight patches
      ;; surrounding the current patch

      ;; count the number of my neighbors that are the same color as me
      set similar-nearby count (turtles-on neighbors)
         with [color = [color] of myself]

      ;; count the total number of neighbors
      set total-nearby count (turtles-on neighbors)

      ;; I'm happy if there are at least the minimal number of
      ;;    same-colored neighbors
      set happy? similar-nearby >= ( %-similar-wanted * total-nearby / 100 )
   ]
end
```

This code asks each turtle to count how many of its neighbors are same-colored turtles and how many are differently colored turtles. It then sets its " happy? " variable to TRUE

if the percentage of same-colored neighboring turtles is at least equal to the value set by

the %-SIMILAR-WANTED slider.

The PERCENT SIMILAR monitor displays what percentage of a turtle ' s neighbors are

the same color as it, on average, while the PERCENT UNHAPPY monitor display what

percentage of all turtles are unhappy.

First Extension: Adding Multiple Ethnicities

One simple extension to this model is to add a third, fourth, and even fifth ethnicity. This can be done by modifying the SETUP code. Currently the model sets all the turtles to red initially, then asks half of them to become green. Recall the code that sets up the turtles:

```
;; create turtles on random patches
ask n-of number patches
   [ sprout 1 ]
ask turtles [
   ;; make half the turtles red and the other half green
   set color one-of [red green]
]
```

We need to figure out how to modify this code so that it works with more than two ethnicities. Since we are now allowing the number of colors to vary we need to go beyond red and green turtles, sometimes we will need blue, yellow, and orange turtles as well.

Therefore, we begin by defining the colors we will allow the turtles to have. We do this

by establishing a global variable *colors*:

```
globals [
percent-similar    ;; on average, what percent of a turtle's neighbors
                                                ;; are the same color as that turtle?
    percent-unhappy  ;; the percent of the turtles that are unhappy
    colors                              ;; a list of colors we use to color the turtles
]
```

The last line, "colors," is the only new global variable. Now we need to define what the value of this global is. We can do this in the SETUP procedure:

```
set colors [red green yellow blue orange ]
```

This line of code initializes the global variable *colors* to be a list of the five colors. Next, we would like to allow the model user to control of the number of ethnicities in the system, and we can do this by adding a NUMBER-OF-ETHNICITIES slider. We will initially set the bounds of this slider from 2 to 5. Now we have to make the code use this slider. We do this by modifying the turtle coloring code. We replace all of the SETUP code above with the following:

```
;; create a turtle on NUMBER randomly selected patches.
          ask n-of number patches
                        [ sprout 1 ]
```

```
;; assign a color to each turtle from the list of our colors
          ask turtles
          [ set color (item (random number-of-ethnicities) colors) ]
```

This code tells a number (equal to the interface slider NUMBER) of random patches to

sprout a turtle. Each turtle picks its color randomly from the list of colors we just initialized. But the turtles can only pick the colors in the list up to the number of ethnicities.

For example, if the NUMBER-OF-ETHNICITIES is 3, only the first three colors in the

list are used.

Now that we have code that allows multiple ethnicities, try running the model with different numbers of ethnicities and observe what effect this has on the segregation of the system. You will notice that as you increase the NUMBER-OF-ETHNICITIES, it takes longer for the system to settle — that is, for all the turtles to be happy. (See figure 3.18.)

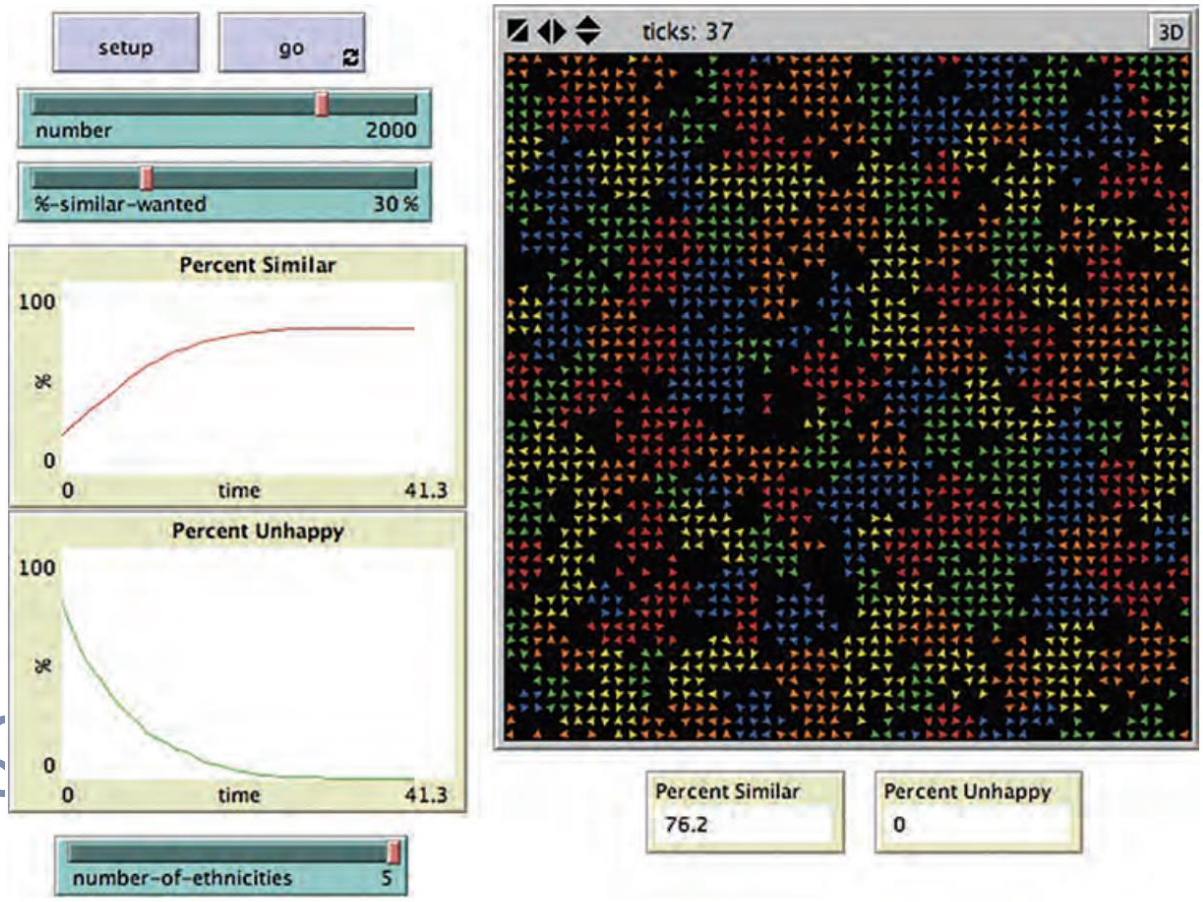


Figure 3.18
NetLogo Segregation model after first extension.

However, once the system has settled, the final PERCENT SIMILAR displayed in the monitor remains pretty much the same regardless of how many ethnicities there are. Can

you explain why this might be?

Second Extension: Allowing Diverse Thresholds

In the original version of Schelling ' s model, every agent in the world had exactly the same similarity threshold. This meant that every agent had the same level of tolerance for having other ethnicities in its neighborhood. Although relaxing this assumption might have been difficult to enact with a checkerboard, it is straightforward to do using an ABM language.

It's likely that different individuals in a real population would have different levels of tolerance. And with an ABM language we can easily give each individual agent its own personal characteristics. Agents can act on these personal characteristics and make decisions based on them. This means that even when agents in an ABM are running the exact same code, their behavior can be quite different.

So how can we make the individuals in the Segregation model more diverse? One way would be to give them a range of similarity thresholds rather than requiring them all to have the same value for that threshold. To do this, we need to first give the turtles the ability to have their own %-SIMILAR-WANTED. We do this by modifying the turtles-own properties. As we showed in chapter 2, any variable in a turtles-own declaration is a variable that is a property of every turtle, that means every turtle can have a different value for that variable.

```
turtles-own [
    happy?          ;; for each turtle, indicates whether at least
    ;; %-SIMILAR-WANTED percent of that turtle's neighbors
    ;; are the same color as the turtle
    similar-nearby  ;; how many neighboring patches have a turtle with
           ;;  my color?
    total-nearby  ;; sum of previous two variables
    my-%-similar-wanted       ;; the threshold for this particular turtle
]
```

Now, in addition to the other properties that the turtles have, they each have a MY-%-SIMILAR-WANTED property, but as of right now this value is not set. We need to modify

the SETUP procedure in order to initialize the turtles with their own MY-%-SIMILARWANTED values.

```
;; assign a color to each turtle from the list of our colors using number-of-
;; ethnicities to maximize the number of colors and assign an
;; individual level of %-similar-wanted
    ask turtles
        [ set color (item (random number-of-ethnicities) colors)
          set my-%-similar-wanted random %-similar-wanted[5] ]
```

We insert this new code directly after specifying the color that we changed in the first extension. This value is set to a random value between 0 and the value from the %-SIMILAR-WANTED slider. This means that the %-SIMILIAR-WANTED variable no

longer specifies the tolerance of each agent, but rather it specifies a maximum value that

any agent can have for its tolerance.

However, the code that determines whether or not a turtle is happy has not yet changed, and so all of these individual tolerance values will be ignored. Let us change the UPDATE-TURTLES code to use these new values. This requires only one small change

to the code:

```
set happy? similar-nearby >= ( my-%-similar-wanted * total-nearby / 100 )
```

All we do is replace %-SIMILAR-WANTED with MY-%-SIMILAR-WANTED to tell the turtle to use its own agent variable instead of the global variable.

Now that we have all of the code for the agents to use their own thresholds, run the model several times with different values of %-SIMILAR-WANTED. Do you see any different results? If you play with the model enough, you will notice that the PERCENT

SIMILAR monitor ends up with a lower value at the end of the run then it did after the first extension. This result is logical, since no individual can be less tolerant in this extension than before, but some individuals will be more tolerant allowing them to find

locations where they are happy even though there are more ethnically different individuals around them. In fact, the average of the MY-%-SIMILAR-WANTED values of the turtles will be approximately half of the global %-SIMILAR-WANTED slider. Comparing the original Segregation model with this extension of the model using twice the %-SIMILAR-WANTED value is left as further exploration for interested readers. (See figure 3.19.)

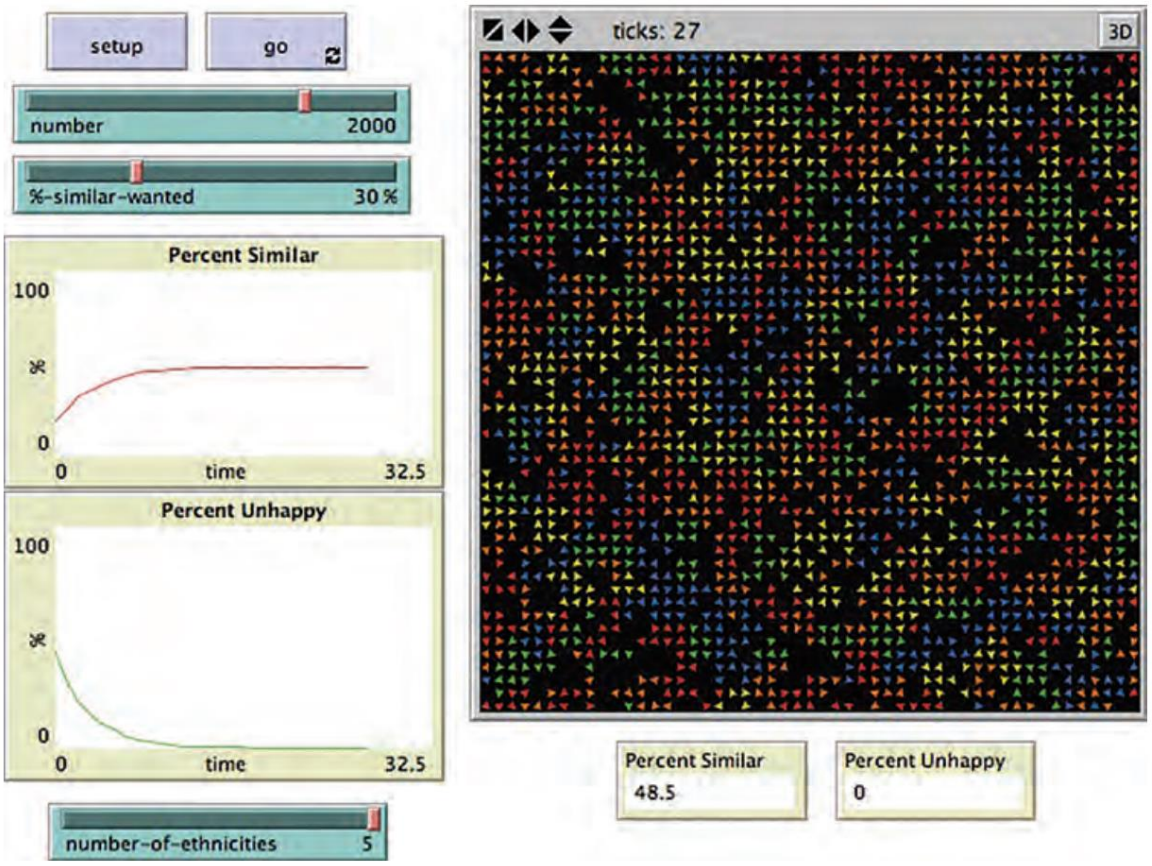


Figure 3.19
NetLogo Segregation model after second extension.

Third Extension: Adding Diversity-Seeking Individuals

Another simplification that the original segregation model makes about the world is that individuals are concerned about too much diversity but that no agents actively seek diversity in their neighborhoods. An easy way to overcome this simplification is to allow individuals to have a %-DIFFERENT-WANTED property just as they have a %-SIMILAR-WANTED property. We begin by creating a slider for %-DIFFERENTWANTED that is just like the %-SIMILAR-WANTED slider. After that we only need to make one additional change to the code. We've already been calculating the number of ethnically different turtles in the neighborhood and so we only have to modify the updateturtles procedure to make use of the new parameter:

```
;; count the number of my neighbors that are a different color than me
let other-nearby count (turtles-on neighbors)
           with [color != [color] of myself]

set happy? similar-nearby >= ( my-%-similar-wanted * total-nearby / 100 )
                  and other-nearby >= ( %-different-wanted * total-nearby / 100 )
```

With the addition of the new clause (shown in bold), the new happiness calculation tells the agents that they are only happy if the number of similar agents nearby is greater than their %-SIMILAR-WANTED threshold *and* the number of other agents nearby is greater than the %-DIFFERENT-WANTED threshold. So for an agent to be happy it cannot be too much in a minority or too great a majority. Now that we have added this code to the model, run it a few times and observe the results.
As you play around with these sliders, it becomes quickly clear that it is much easier for the PERCENT SIMILAR results to decrease after this extension has been implemented. The reason is that all of the agents actually seek out diversity now, so they are deliberately taking actions that decrease the PERCENT SIMILAR results. Moreover, as you manipulate the sliders, you can easily find states of the parameters where the system

never settles down. For instance, if you put both the %-SIMILAR-WANTED and %-

DIFFERENT-WANTED sliders over 50 percent, the model will probably never reach

equilibrium. This is because some agents (those with MY-%-SIMILAR-WANTED greater than 50) will be trying to satisfy impossible demands; they will never find a location where more than half the agents around them are similar to themselves and more than half the agents around them are different than themselves.6 In general you will find that it takes this new system much longer to settle down to an equilibrium state. This is because the agents are now pickier in terms of where they are happy. They are seeking both diversity and similarity. (See figure 3.20.)

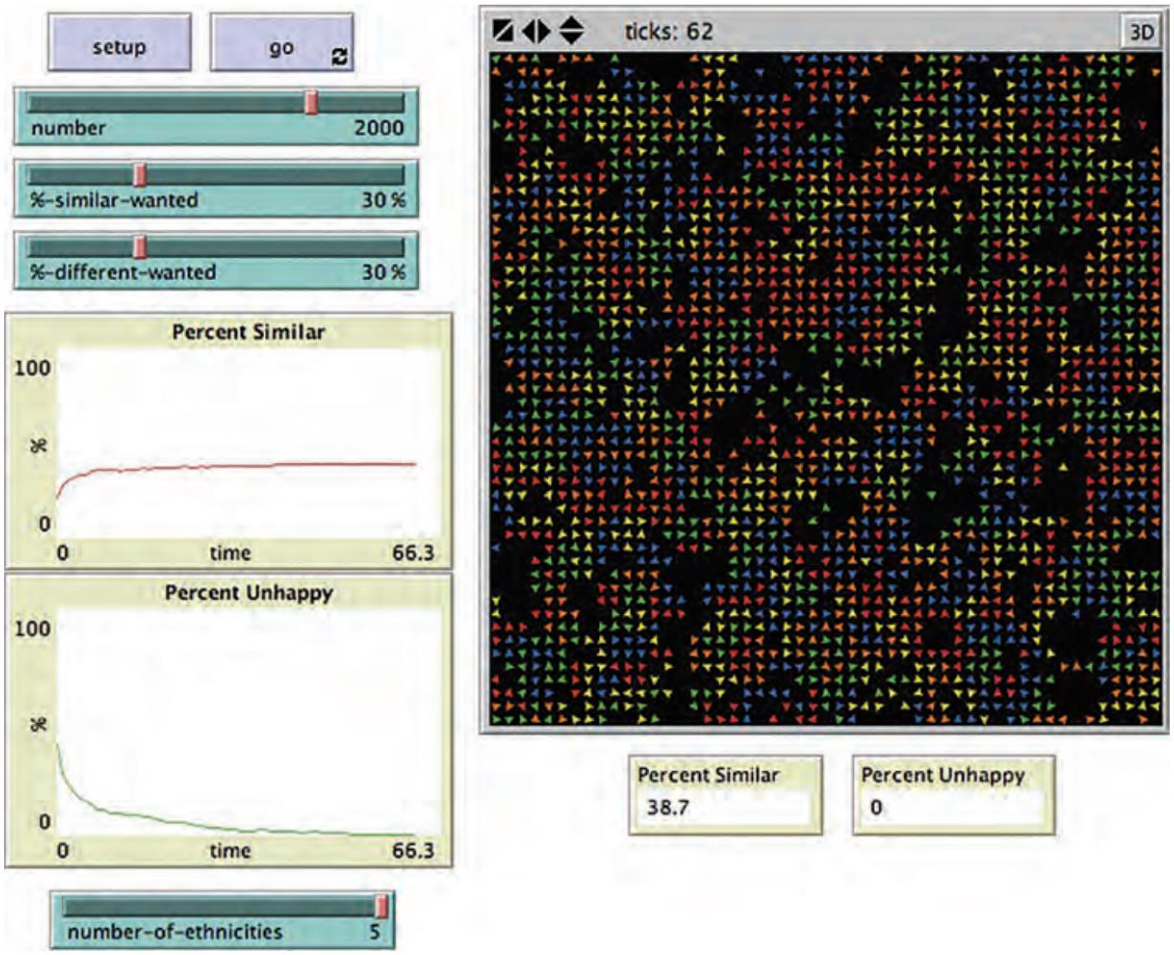


Figure 3.20
NetLogo Segregation model after third extension.

Advanced Urban Modeling Applications

Agent-based modeling has frequently been used for the modeling of urban landscapes.

"Residential Preference"

The Segregation model described earlier can be seen as one particular example of what are generally called "residential preference" models, which are a major component of urban modeling systems.

CITIES project

These models can be combined with commercial, industrial, and governmental models of urban policy to create an integrated model of a city. One such example is the CITIES project (see figure 3.21) carried out by the Center for Connected Learning and Computer-Based Modeling at Northwestern University. The goal of the CITIES project was to procedurally develop realistic cities that could be used as tools in education and exploration (Lechner et al., 2006).

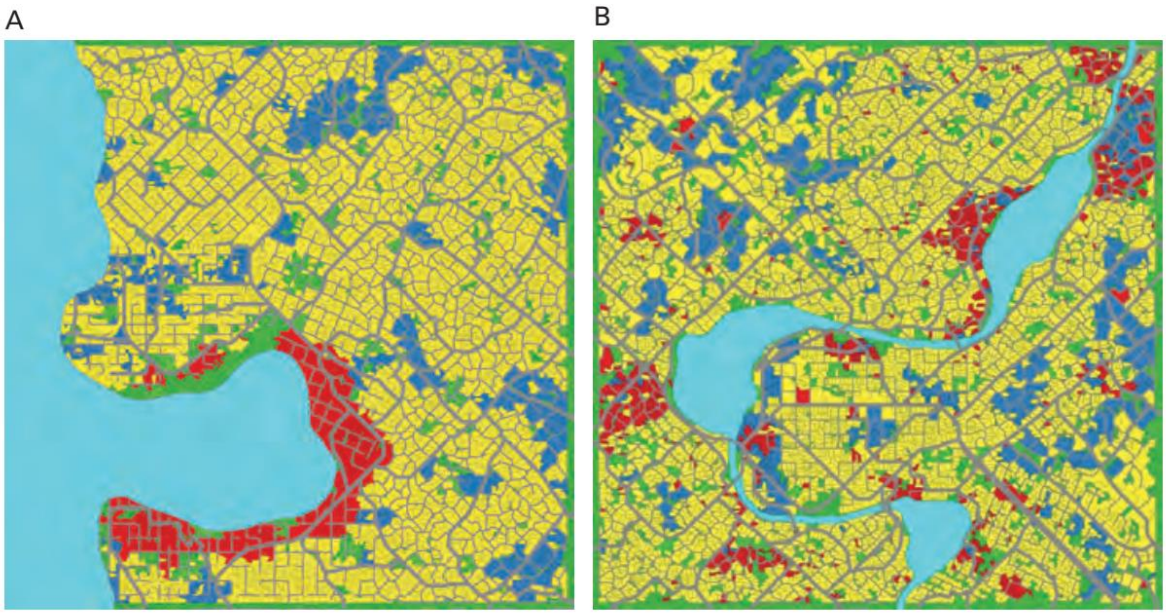


Figure 3.21
Some runs of the CITIES model in NetLogo (Lechner et al., 2006).

Goal Of Urban Modeling

Another goal of urban modeling is to explore the ecological impact of cities and their footprint on the environment.

SLUCE Project

The SLUCE project at the University of Michigan built several models of residential and tract development to explore land-use policies and their effect on the surrounding environs in southeastern Michigan (Brown et al., 2005; Rand et al., 2003).

Both of these projects can be seen as a subset of the larger category of land-use modeling. For a detailed review of agent-based models of land use, see Parker et al. (2003) and Batty (2005).

The El Farol Model

Occasionally, when working on agent-based models, you will find a model that reproduces phenomena that you are interested in but does not quite provide the output or data that you want. In this section we will take a model of an economic scenario, the El Farol model, and we will add additional reporters to this model. In this way we can gain new information from the model that was not previously available. This is possible because data collection in agent-based models is fairly simple, and the capabilities of ABMs for visually displaying information are diverse.

This section is a good example of how you do not need to understand every part of an agent-based model in order to work with it. The El Farol model uses “ lists ” and mathematical

regression, but the modifications we will make to this model do not require knowledge of how these particular methods work. Although it is almost always desirable to understand the basic rules of the model, for the purposes of a particular extension, it may be sufficient to develop that understanding from reading the Info tab rather than trying to puzzle out the details of the code. When extending models, it is useful, though not easy, to develop the skill of understanding only the code you will need to modify and know which code is safe to ignore. Our extensions to this model will not require much understanding of the mechanics of the model.

Description of the El Farol Model

W. Brian Arthur was an Irish economist who, at age thirty-seven, became the youngest

holder of an endowed chair at Stanford University. He was a pioneer of using complexity methods in economics and is on the founders' board of the Santa Fe Institute. In 1991,

Arthur posed the El Farol Bar problem as an exploration of bounded rationality and inductive reasoning.

" Optimal " Equilibrium

Traditional neoclassical economics presupposes that humans are completely rational: that is, they have access to perfect information and maximize their utility in every situation (Arthur, 1994). When each agent behaves this way, the aggregate can achieve an "optimal" equilibrium. But this is an idealization and people do not really conform to this normative description.

Therefore, Arthur suggested the El Farol Bar problem as an example of a system where agents do not perfectly optimize, yet a classical economic equilibrium is achieved. Arthur' s example is even more striking, because it is a situation in which it would be difficult for the "ideal" economic model to achieve equilibrium.

El Farol Bar

On Canyon Road in Santa Fe, New Mexico, there is a bar called the El Farol. This bar was popular with researchers from the Santa Fe Institute, including Arthur. One night each week, the El Farol had live Irish music, and Arthur enjoyed going on these nights, but occasionally it got crowded, and he did not enjoy it on those nights. Arthur wondered,

how do people decide if they should go to the bar or not? He imagined that there were one hundred citizens of Santa Fe who liked Irish music, and that each week they each tried to predict if the El Farol would be crowded. If they thought it was going to be crowded, specifically that more than sixty people would go to the bar, then they would stay at home, but if they thought it was not going to be crowded, then they would go to the El Farol.

Assuming that the attendance information at the bar each week was readily available, but that each citizen could only remember a limited number of weeks of the attendance,

Arthur's hypothesis

Arthur hypothesized that one way to model this situation would be to give each agent a bag of strategies.

Strategies

Each of these strategies would be some rule of thumb about what the attendance was that week, e.g., " twice last week ' s attendance, " " half the attendance of two weeks ago, " or " an average of the last three weeks attendance. " Each agent had a group of these strategies, and he would see how well these strategies would work had he used them in the previous weeks. The agents would use whichever strategy would have worked the best to predict this coming week ' s attendance, and they would decide

whether to attend the bar based on this prediction. When Arthur wrote this up as an agentbased model and examined the results, he found that the average attendance at the bar was around 60. So despite all of the agents using different strategies and not having perfect information, they managed to optimally utilize the bar as a resource.

Netlogo Model

This model is available in the NetLogo models library under Sample Models > IABM

Textbook > Chapter Three > El Farol (Rand & Wilensky, 2007; http://ccl.northwestern.edu/netlogo/models/ELFarol). This model has a few controls. You can adjust the MEMORYSIZE of the agents, which controls how many weeks of attendance they remember.

You can also change the NUMBER-STRATEGIES, which is the number of strategies each agent has in its bag of strategies. Finally, you can adjust the OVERCROWDINGTHRESHOLD, which is the number of agents needed to make the bar crowded. If you run the model as is, you will see the agents moving back and forth from the bar (the blue area) to their homes (green area), and the attendance is plotted on the left. The Interface tab of the model can be seen in figure 3.22.

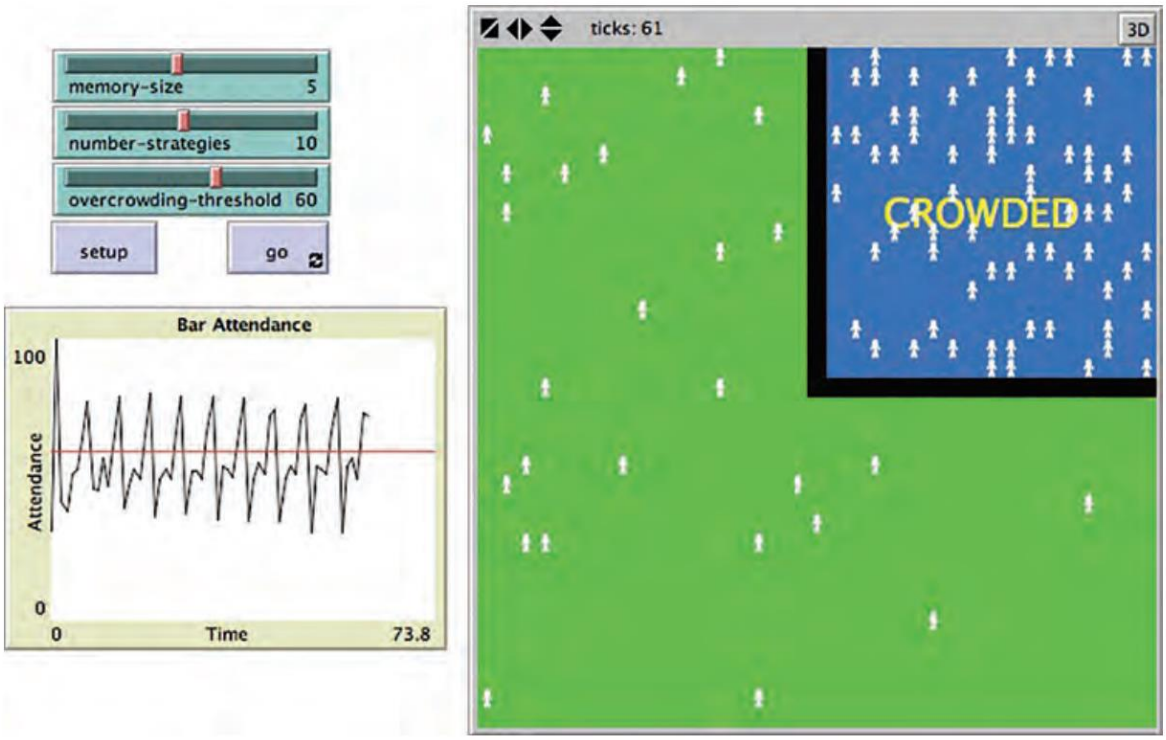


Figure 3.22
The El Farol model. Rand & Wilensky (2007). http://ccl.northwestern.edu/netlogo/models/ElFarol.

First Extension: Color Agents That Are More Successful Predictors

The original model tells you how many agents are attending the bar, and you can get the visual reference of them moving to and from the bar, but there is nothing in this model to indicate which agents are better at choosing good times to attend the bar. Are some agents better at deciding when the bar will not be crowded than others? In order to find this out, we first need each agent to keep track of how often they go to the bar when it is not crowded; to do this, we add a property to each agent. We will modify the turtles ' properties to add one additional item, REWARD:

```
turtles-own [
    strategies          ;; list of strategies
    best-strategy ;; index of the current best strategy
    attend?                ;; true if the agent currently plans to attend the bar
    prediction          ;; current prediction of the bar attendance
    reward                 ;; the amount that each agent has been rewarded
]
```

We are calling this new property REWARD, since the agents get rewarded if they attend

the bar when it is not crowded. Now we have to initialize this property, since each agent

will start with reward of 0. To do this we modify the create turtles part of the SETUP procedure, to initialize the REWARD to 0:

```
;; create the agents and give them random strategies
create-turtles 100 [
    set color white
    move-to-empty-one-of home-patches
    set strategies n-values number-strategies [random-strategy]
    set best-strategy first strategies
    set reward 0
    update-strategies
]
```

We also need to update the reward whenever the agent goes to the bar and it is not crowded. If we look at the GO procedure there is some code that determines if the bar is overcrowded or not and, if it is, the word CROWDED is displayed on a patch in the bar:

```
if attendance > overcrowding-threshold [
    ask crowded-patch [ set plabel "CROWDED" ] ;; label the bar as crowded
]
```

We want to update the reward of turtles who attend the bar when this condition is not true, so we can make this IF statement, an IFELSE statement, and then in the ELSE part

of the IFELSE we can ask all the turtles who attended the bar to increase their reward:

```
ifelse attendance > overcrowding-threshold [
    ask crowded-patch [ set plabel "CROWDED" ]
]
[   ;; if the bar is not overcrowded, reward the turtles that are attending
    ask turtles with [ attend? ] [
        set reward reward + 1
    ]
]
```

Now that we have given rewards to agents attending the bar when it is uncrowded, the final step is to modify the visualization to display these rewards appropriately. One way

to do that is to give each agent a different color based on how much reward it has accumulated. To make sure that we can keep track of relative differences in the model, we are going to color the agents relative to the maximum reward obtained. This means that we need to recolor every agent not just the ones that have gathered new rewards this time step.

If you look farther up in the GO procedure you will find some code where we ask each agent to predict the attendance that week. This is a good place to have the agents update

their color. NetLogo has a command called SCALE-COLOR that allow you to do this, we can set each agent ' s color to a shade of a color based on their reward. The SCALE-COLOR primitive takes four inputs: (1) a base color, which we will make red, (2) the variable that we are linking the color to, REWARD in this case, (3) a first range value, in this case we set it to slightly more than the maximum reward so far, and (4) a second range value, which we set to 0. If the first range value for SCALE-COLOR is less than the second one, then the larger the

number of the linked variable the lighter the color will be. If the second range value is less than the first one (as it is in this case), then the large the number of the linked variable, the darker the color will be. In this model, we set the first value to be the larger of the two so that everyone starts as white, and then turns a darker color as they accumulate rewards. This makes the model more consistent with the basic version, where all the agents are white:

```
ask turtles [
        set prediction predict-attendance best-strategy sublist history 0 memory-size
        ;; set the Boolean variable
        set attend? (prediction <= overcrowding-threshold)
        set color scale-color red reward (max [ reward ] of turtles + 1) 0
    ]
```

In this case, the darkest agents will be the one with the lowest reward, and the lightest agents will be the ones with the most reward (see figure 3.23).

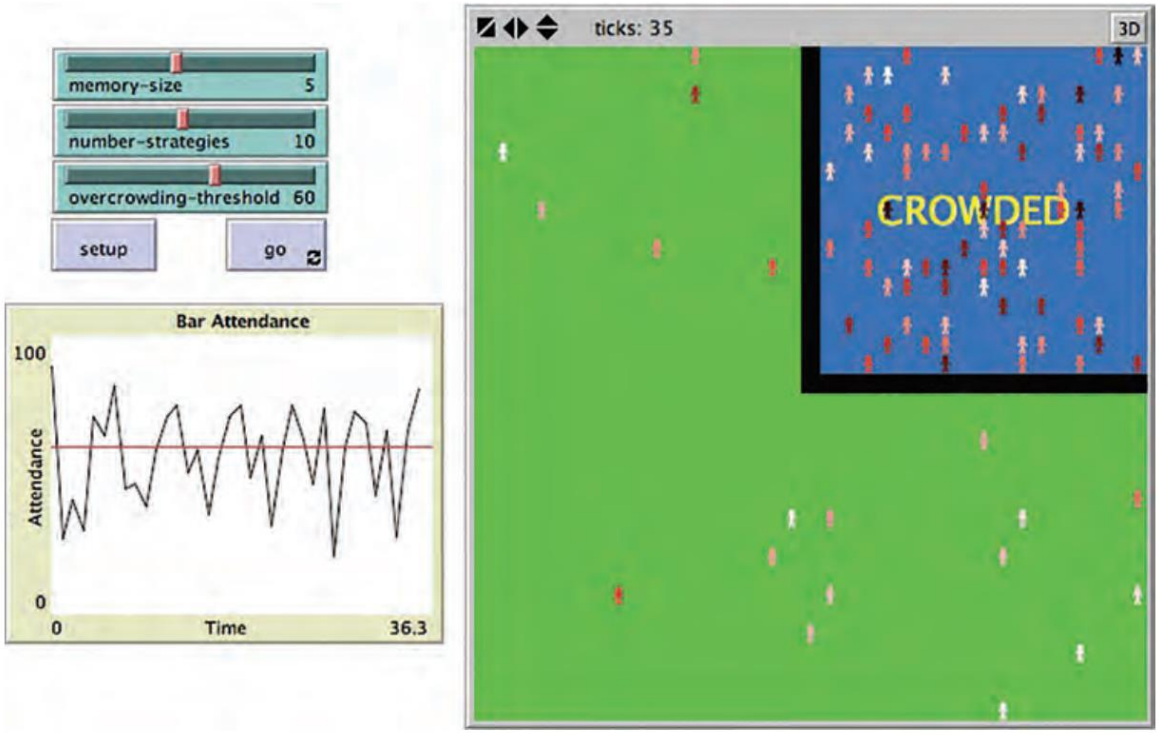


Figure 3.23
El Farol model: first extension.

Second Extension: Average, Min, and Max Rewards

The first extension enables us to visually see which agents are doing better at going to the bar when it is not crowded. However, we do not have any hard numbers for the agents.

We could simply inspect each agent by right clicking on it and selecting inspect agent.

This would enable us to see the reward for each agent, but a better way might be to constantly display some information about the agents. We can do this by using monitors, which display values about the agent-based models.

To begin with we add a monitor to the Interface tab. In the first monitor, we can calculate the maximum reward collected by any agent. In the reporter area of the monitor editing window, we can put the following code snippet:

```
max [ reward ] of turtles
```

We can also give this monitor the name " Max Reward. " After this we can add another monitor for the minimum reward with the following code:

```
min [ reward ] of turtles
```

And give this monitor the name " Min Reward. " Finally, we can create a third monitor and calculate the average or mean reward using the following code:

```
mean [ reward ] of turtles
```

We can give this monitor the name " Avg. Reward. " The model interface window now should look like figure 3.24 . When we run this model we now get data on the average, maximum, and minimum rewards over time. It quickly becomes apparent that though both the average and maximum reward increase over time, the max reward grows faster than the average reward meaning that some agents do better and better as time goes on. Though this extension did not take much coding, it gives you information and insight into what is going on in the model that was not available before.

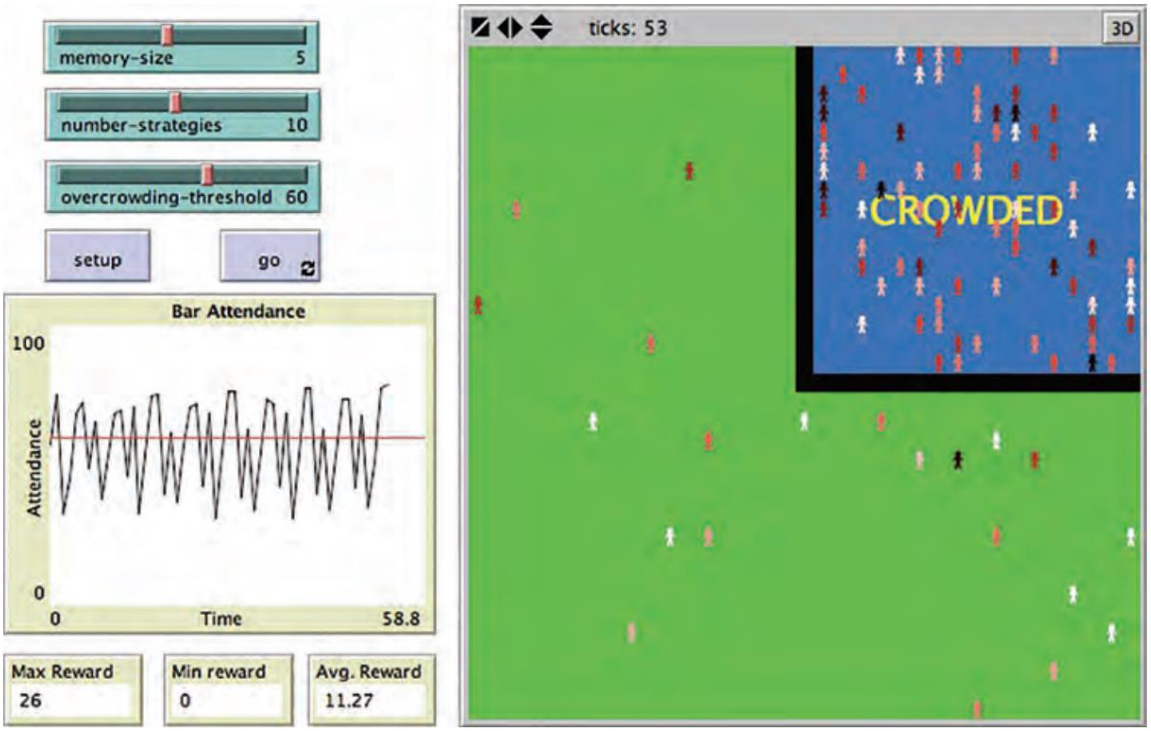


Figure 3.24
El Farol model: second extension.

Third Extension: Histogram Reward Values

The second extension provides more data about how the agents in the model are doing, but this data has been aggregated so it is not obvious how many agents occupy each reward level. For example, are the agents distributed uniformly across reward levels or are many agents at the minimum value and many agents at the maximum value with few agents in between? To find out an answer to this question, we are going to add another plot to the El Farol model. This plot will be a histogram of the distribution of rewards. This is similar to the plot we created in chapter 2 for the Simple Economy model, but in this model, we will need to use the plot widget's "Plot update commands" feature.

To do this we first need to create a new plot in the Interface tab. In the configuration for this plot, we can give the plot the name "Reward Distribution," and change the Mode of the pen to "Bar" by clicking on the pencil in the first row of the Plot pens table. We tell the pen to display a histogram of the distribution by putting the following code in "Pen update commands. "

```
histogram [ reward ] of turtles
```

This tells the pen to display a histogram of the REWARDs of the turtles. If we ran the model in this state, we would see a histogram drawn. However, the axes of the plot would not update correctly. Thus, we must enter the following under "Plot setup commands. "

```
set-plot-y-range 0 1
set-plot-x-range 0 (max [ reward ] of turtles + 1)
```

These commands are run every time the plot is updated. They set the X and the Y range

of the plot window to reflect the current values that we are plotting. By setting the Y-range to 1, we are letting NetLogo automatically decide how to increase this range if it needs to in order to show all the data. (See figure 3.25.)

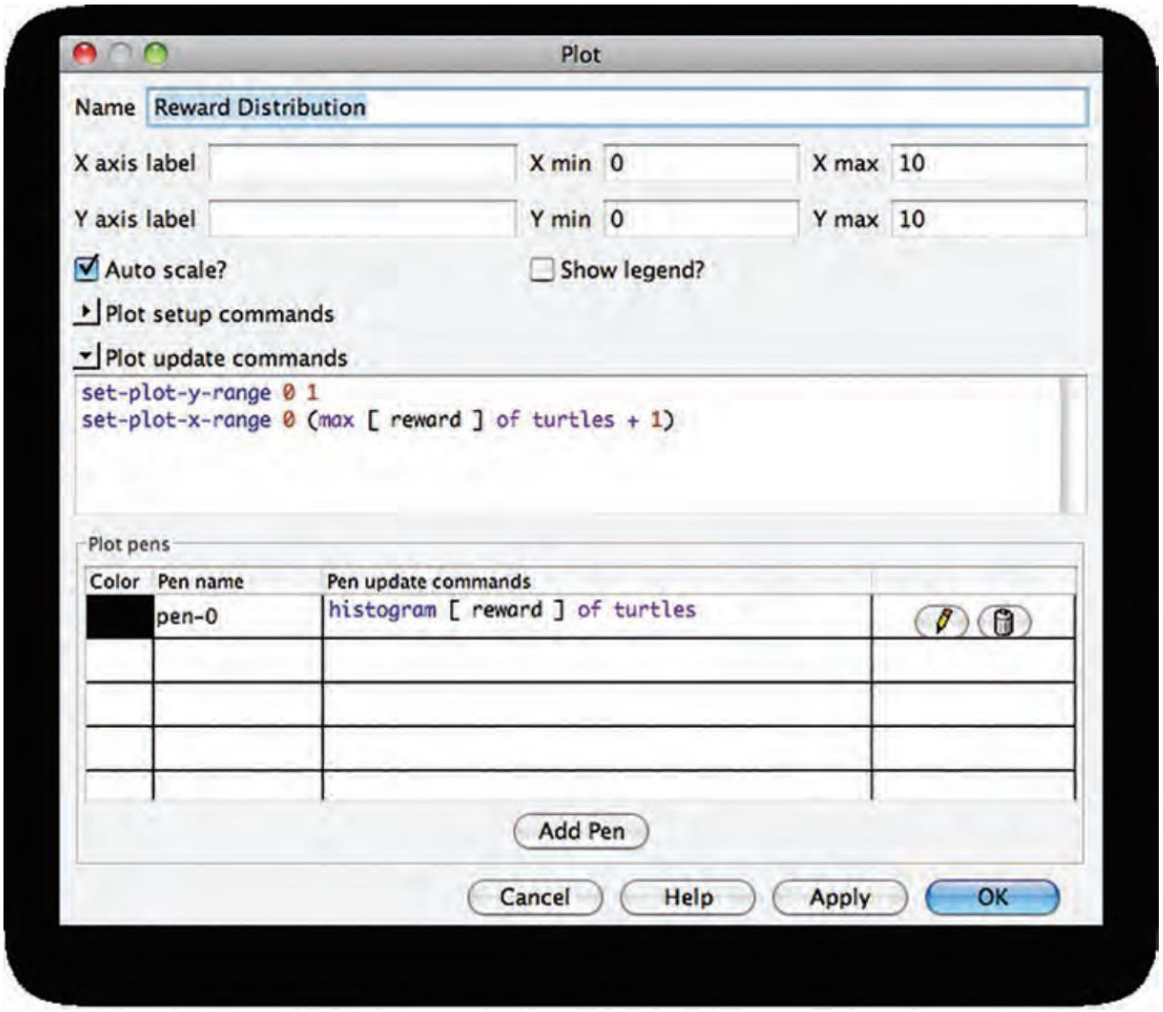


Figure 3.25
El Farol model: reward plot properties.

As you run the model over time, you will see it start out similar to a normal distribution,

but the distribution will quickly change with a few groups of individuals maintaining large rewards and many more having a lower reward. This lends credence to the hypothesis that there are a few agents that achieve a high reward level, but then there is a large gap between these high achieving agents, and the majority of agents, which achieve more average reward levels. (See figure 3.26.)

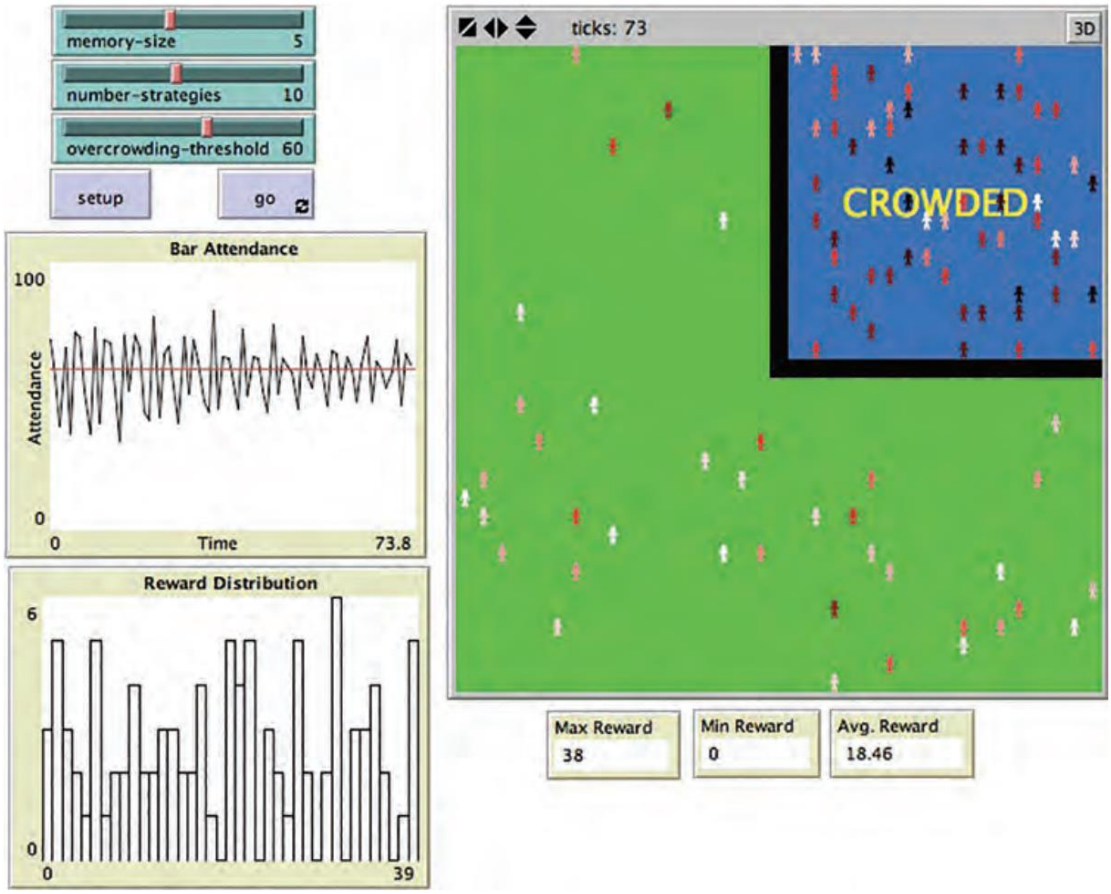


Figure 3.26
El Farol model: third extension.

Advanced Modeling Applications

The El Farol model has been investigated in many different ways. Partially because it is an exciting model in that it combines both ABM and machine learning (Rand, 2006 ; Rand & Stonedahl, 2007).

Machine learning

Machine learning enables the creation of powerful agent-based models where the agents not only change their actions over time, but also their strategies, i.e., the agents change the way they decide to take actions.

Minority Game

The El Farol model has also been idealized into the Minority Game (Challet, Marsili & Zhang, 2004), which has been studied by physicists and economists because it provides insights into complex systems. For instance, both El Farol and the Minority Game can be seen as a crude approximation for financial markets. In some sense, if everyone else in a financial market is

selling, you want to be buying, and if everyone else is buying then you want to sell; being in the minority is usually a good way to make money.

Economic Model

The El Farol model is also one example of an economic model. Another early agentbased economic model was the Artificial Stock Market developed by a group of Santa Fe Institute researchers (Arthur et al., 1997). This model attempted to simulate the stock market and allowed researchers to investigate how investors affect phenomena like booms and busts in the market.

Economic Models In NetLogo

There are several other economic models in the NetLogo models library, such as the Oil Cartel model and the Root Beer Supply Chain (see figure 3.27).

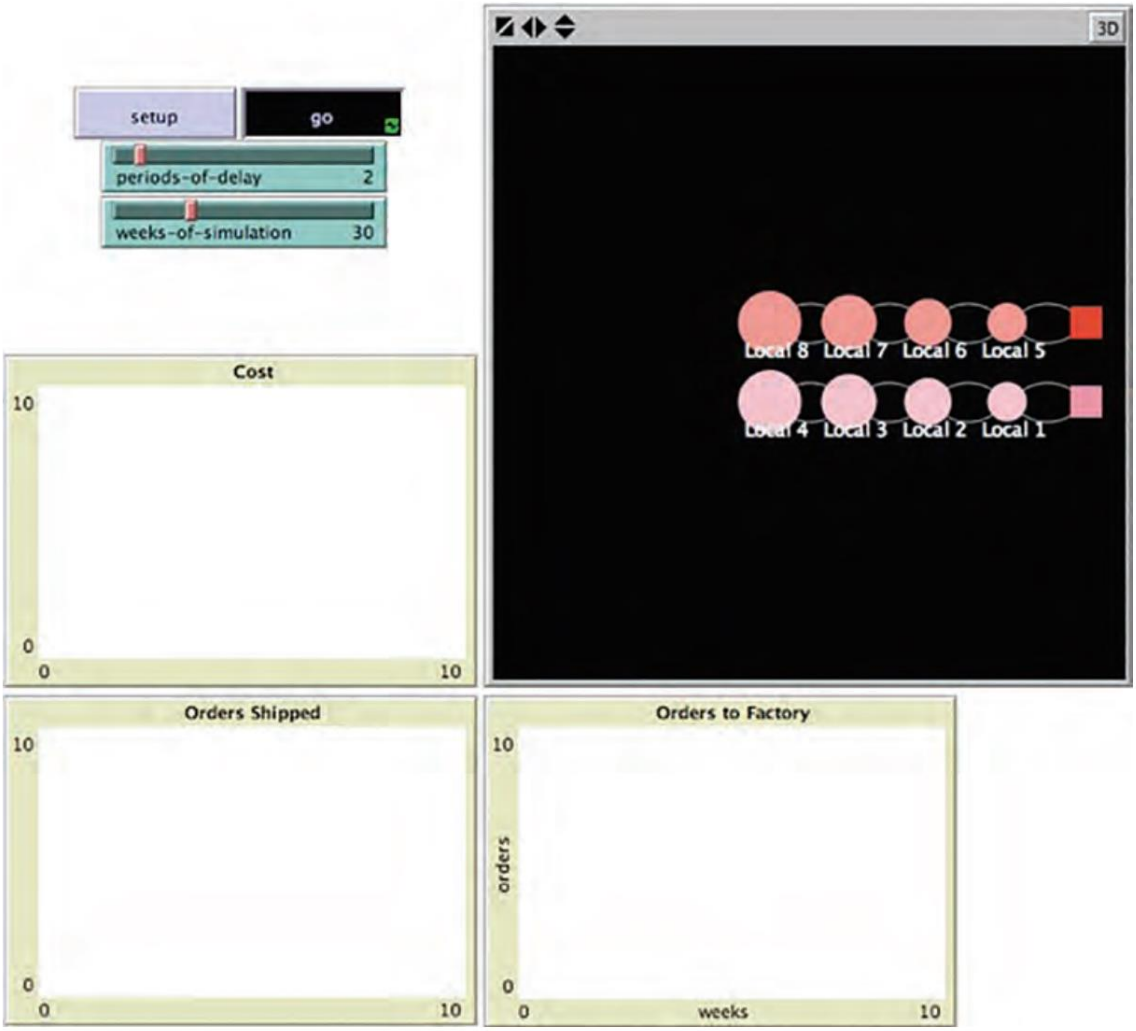


Figure 3.27
NetLogo HubNet Root Beer Game model. (*Wilensky & Stroup 2003*). http://ccl.northwestern.edu/netlogo/models/HubNetRootBeerGame.

Both of these models are interesting because they make use of the HubNet facility that is also provided with NetLogo.

HubNet

HubNet (Wilensky & Stroup, 1999c) is a NetLogo feature that enables Participatory Simulations, which is a simulation method where humans control agents in an agent-based model (see chapter 8 for more information on participatory simulations). This gives humans the ability to mix with nonhuman agents and to gain a deeper understanding of how their decisions affect and are constrained by a complex system.

After extending four NetLogo models, you should have the tools to extend many other NetLogo models.

Tools To Extend NetLogo Models

In the Fire model extensions and in the DLA extensions, we have seen that it is often useful to extend a model by replacing deterministic rules by probabilistic rules. Conversely, as we saw in modeling wind and long-distance transmission in the Fire model, it is sometimes useful to replace a probabilistic rule with an agent-based mechanism that generates the probabilities. And with each change to the model, we need to consider whether we also need to design a new metric for measuring model performance.

First Step of Devising a Question

- This module is intended to take you from the first step of devising a question or area you want to explore, all the way through designing, and building your model, to refining your question and revising your model, to analyzing your results and answering your question.

Sequence of Steps

This sequence is presented here in linear order, but in reality these steps fold back on each other and are part of an iterative exploration and refinement of the model and motivating question.

Three Main Sections

To facilitate this process, this module is broken into three main sections:

(1) Designing your model will take you through the process of determining what elements to include in your model,

(2) Building your model will demonstrate how to take a conceptual model and create a computational object, and

(3) Examining your model will address how to run your model, create some results, and analyze those results to provide a useful answer to your motivating question.

Model Of An Ecological System

Throughout this module, we will be designing, building, and examining a simple model

of an ecological system.

The basic question that we will be addressing is: “How do the population levels of two habitat-sharing animal species change over time?”

Wolf Sheep Simple model

For our purposes in this module, we will call this model the Wolf Sheep Simple model. Though we will discuss this model in the context of two biological species, the model could be generalized to other situations such as companies competing for consumers, electoral parties competing for votes, or viruses evolving in a computer system. More important, the components that we will be developing in this model are basic components utilized in most ABMs.

Designing Your Model

There are many ways of designing an agent-based model.1 Which you choose will depend on many factors including the type of phenomenon to be modeled, your level of knowledge of the content domain, your comfort with NetLogo coding and your personal modeling style.

Major Categories of Modeling

We consider two major categories of modeling: *phenomena-based modeling* and *exploratory modeling*.

PHENOMENA-BASED MODELING

Reference Pattern

In *phenomena-based modeling*, you begin with a known target phenomenon. Typically, that phenomenon has a characteristic pattern, known as a ***reference pattern***. Examples of reference patterns might include common housing segregation patterns in cities, spiral-shaped galaxies in space, leaf arrangement patterns on plants, or oscillating population levels in interacting species.

Goal Of Phenomena-Based Modeling

The goal of phenomena-based modeling is to create a model that will somehow capture the reference pattern. In ABM, this translates to finding a set of agents, and rules for those agents, that will generate the known reference pattern.

Once you have generated the reference pattern you have a candidate explanatory mechanism for that pattern and may also vary the model parameters to see if other patterns emerge, and perhaps try to find those patterns in data sets or by conducting experiments. You can also use phenomena-based modeling with other forms of modeling, such as equation-based modeling. In equation-based modeling, this would mean writing equations that will give rise to the reference pattern.

EXPLORATORY MODELING

The second core modeling form is exploratory modeling. This form is perhaps less common in equational contexts than it is in ABM. In exploratory modeling with ABM,

you create a set of agents, define their behavior, and explore the patterns that emerge.

We then refine our model in the direction of perceived similarities with these phenomena

and converge toward an explanatory model of some phenomenon.

Formulate Research Question

Another distinction in modeling methodology is to what degree we specify a question to be answered by a model.

- At one end of the spectrum, we formulate a specific research question (or set of questions) such as " How does a colony of ants forage for food? " or " How does a flock of geese fly in a V-shape? "
- At the other end, we may only begin with a sense of wanting to model ants or bird behavior, but without a clear question to be answered.
- As we explore the model design space, we will gradually refine our question to one that can be addressed by a specific model.

Designing The Conceptual Model

Yet a third dimension is the degree to which the process of designing the conceptual model is combined with coding your model.

In some cases, it is advisable to work out the entire conceptual model design in advance of any coding of the model. This is referred to as *top-down* design.

Top-Down Design

In a top-down design, the model designer will have worked out the types of agents in the model, the environment they reside in, and their rules of interaction before writing a single line of code.

Bottom-Up Design

In other cases, the conceptual model design and the coding of the model will coevolve, each influencing the evolution of the other. This is often referred to as *bottom-up* design. In bottom-up design, you choose a domain or phenomenon of interest with or without specifying a formal question. Using this approach, you would then start writing code relevant to that domain, building the conceptual model from the bottom up, accumulating the necessary mechanisms, properties and entities, and perhaps formulating some formal research questions along the way. For example, in a bottom-up design, you might start with a question about how an economic market would evolve, code some behaviors of buyers and sellers and, in so doing, realize you'll need to add brokers as agents in the model.

Design Principle

We will emphasize the top-down approach.

- The top-down design process starts by choosing a phenomenon or situation that you want to model or coming up with a question that you want to answer, and then designing agents and rules of behavior that model the elements of the situation. You then refine that *conceptual model* and continue to revise it until it is at a fine enough level of detail that you can see how to write the code for the model.

ABM Design Principle

Throughout the design process there is one major principle that we will use. We call this the *ABM design principle*: Start simple and build toward the question you want to answer.

Components Of Design Principle

There are two main components of this principle.

First Component: The first is to begin with the simplest set of agents and rules of behavior that can be used to explore the system you want to model. This part of the principle is illustrated by a quote from Albert Einstein, " The supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience " (1933). Or in another phrase he is reputed to have said: " Everything should be made as simple as possible, but not simpler. " In the case of ABM, this means making your model as simple as possible given that it must provide you with a stepping-stone toward your final destination.

Second Component: Second, always have your question in mind, which means not adding anything to your model that does not help you in answering your question. The statistician George Box provides a quote that illustrates this point, "All models are wrong, but some models are useful" (1979). What Box meant was that all models are by necessity incomplete because they simplify aspects of the world. However, some of them are useful because they are designed to answer particular questions and the simplifications in the model do not interfere with obtaining that answer.

Usefulness of ABM Design Principle

This core ABM design principle is useful in several ways.

- First, it reminds us to examine every candidate model agent and agent-rule and eliminate it if progress can be made without it. It is not uncommon for novice modelers to build a model in which certain components have no effect whatsoever. By starting small and slowly adding elements to your model, you can make sure that these extraneous components never get developed. By examining each additional component as to whether it is needed to answer the research question you are pursuing, you reduce the temptation to, paraphrasing William of Occam, " multiply entities unnecessarily. " In so doing, you reduce the chance of introducing ambiguities, redundancies, and inconsistencies into your model.

- Another virtue of the ABM design principle is that, by keeping the model simple, you make it both more understandable and easier to verify.
    - ***Verification*** is the process of ensuring that a computational model faithfully implements its target conceptual model.
    - **Simpler Conceptual Model:** A simpler conceptual model leads to a simpler model implementation, which makes it easier to verify the model. Starting simple and building up also facilitates the process of just-in-time results.

Identifying a Question: To apply our principle to the context of the Wolf Sheep Simple model, we need to start our design by reflecting on two habitat-sharing animal species and identifying simple agents and behaviors for our model. We will start by identifying a question we want to explore, which is required by the ABM design principle (top-down version). After that we will discuss what the agents in our models are and how they can act. Then we move on to the environment and its characteristics. As part of this process, we need to discuss what happens in an individual time step of the model. Finally, we discuss what measures we will be using to answer our question.

Choosing Your Questions

Choosing a question may seem to be a separate issue from model design.

Natural Progression

After all, the natural progression seems to be:

- first choose a question, and
- second build a model to answer that question.

Sometimes, that may indeed be the procedure we follow, but in many instances we will need to refine our questions when we start to think about it in an agentbased way.

Wolf Sheep Simple model

Our original question for the Wolf Sheep Simple model was:

- " How do the population levels of two species change over time when they coexist in a shared habitat? "

We will now evaluate whether this question is one that is amenable to ABM and refine our question within the ABM paradigm.

ABM For Making Sense Of Systems

Agent-based modeling is particularly useful for making sense of systems that have a

number of interacting entities, and therefore have unpredictable results. As we discussed

in early modules of the course, there are certain problems and questions that are more amenable to ABM solutions.

Consider A Different Modeling Method

If our primary question of interest violates our guidelines, it may be an indication that we should consider a different modeling method.

Assumption

For example, we might be interested in examining the dynamics of two very large populations under the assumption that the species are homogenous and well-mixed (no spatial component or heterogeneous properties) and that the population level of each species is simply dependent on the population level of the other species.

EBM vs. ABM

If that is the case then we could have used an equation-based model instead of an agent-based model since EBM' s work well for large homogeneous groups, and as we mentioned in previous chapters, there is a classic EBM for this situation known as the Lotka-Volterra differential equations (Lotka, 1925 ; Volterra, 1926).

Conceptualize The Agents

ABM will be more useful to us if we are thinking of the agents as heterogeneous with spatial locations. This affects how we conceptualize the agents. One aspect of the animals that is likely to be relevant to our question is how they make use of their resources. Animals make use of food resources by converting them to energy hence we will want to make sure that our agents have different amounts of energy and different locations in the world.

Agent's Interaction With Environment

A third guideline is to consider whether the aggregate results are dependent on the interactions of the agents and on the interaction of the agents with their environment. For example, if one species is consuming another then the results will be dependent on agent interaction. Predator-prey interactions are usually set in rich environments. Keeping the ABM design principle in mind, we start with the simplest environment — we enrich the environment a little by going beyond just predators and prey and including resources from the environment that the lowest level prey species consume.

Time Dependent Processes

Yet another guideline is that agentbased modeling is most useful for modeling time dependent processes. In the Wolf Sheep Simple model, our core interest lies in examining how population levels change over time. We might therefore refine our question to focus on conditions that lead to the two species coexisting together for some time. In this way our guidelines help us evaluate whether our question is well suited to ABM and, if so, to focus our question and conceptual model.

A Concrete Example

Now that we have identified our research question in detail it can be useful to consider a particular context for this research question.

Phenomena-based Agent-based Models

Earlier, we discussed reference patterns as a source of phenomena-based agent-based models. Sometimes that reference pattern is the original inspiration for the model.

Refined Research Question

Other times, as now, we have refined our research question enough that we seek out a reference pattern that will help us test whether our model is a valid answer to the question.

Predator-prey Relations

In the case of the predator-prey relations, there is a famous case of cohabiting small predator-prey populations in a small geographic area. This is the case of fluctuating wolf and moose populations in Isle Royale, Michigan. The wolves and moose of Isle Royale have been studied for more than five decades. This research represents the longest continuous study of any predator-prey system in the world. . . . Isle Royale is a remote wilderness island, isolated by the frigid waters of Lake Superior, and home to populations of wolves and moose. As predator and prey, their lives and deaths are linked in a drama that is timeless and historic. Their lives are historic because we have been documenting their lives for more than five decades. This research project is the longest continuous study of any predator-prey system in the world. (From the Wolves & Moose of Isle Royale Project Website, http://isleroyalewolf.org/) Figure 4.2 shows the wolf and moose populations in Isle Royale from 1959 through 2009.

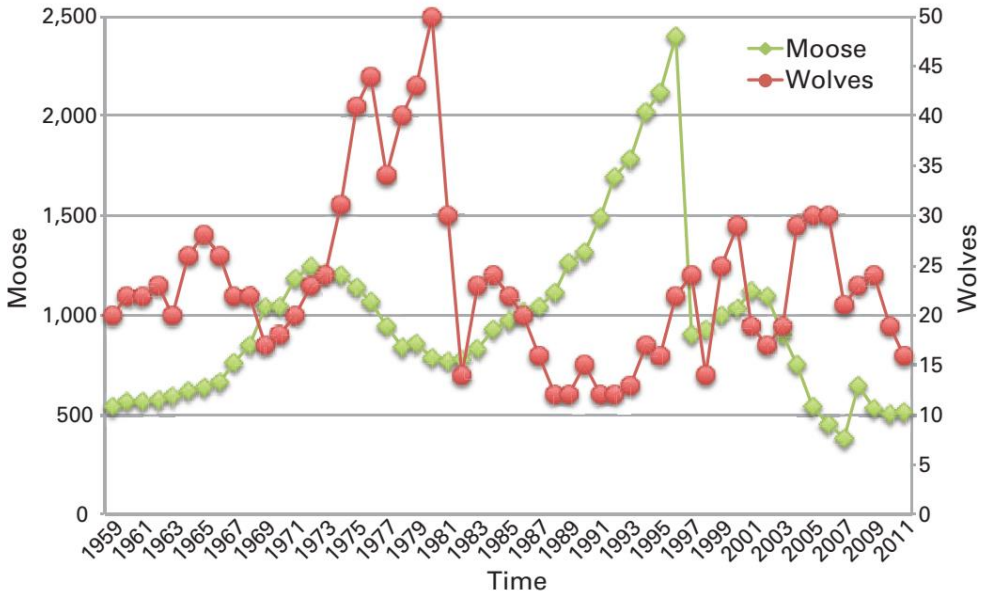


Figure 4.2
Five decades of fluctuating wolf and moose populations at Isle Royale. Note that when the wolf population peaks, the moose population is at a low point and, similarly, when the moose population peaks, the wolf population is at a low point.

This graph can serve as a reference pattern for our model. Our completed model will have to be able to generate a graph "similar" to this one to be a possible explanatory model of these phenomena. In general, this process is called *validation*.

As we can see in the data from Isle Royale, the wolf and moose populations in Isle Royale have been sustaining themselves for more than fifty years without either species going extinct. The populations also exhibit a rough oscillation, with moose at a low when wolves peak and vice versa. This data can serve as a reference pattern for our phenomenabased modeling. It allows us to further refine our research question to this: " Can we find model parameters for two species that will sustain *oscillating* positive population levels in a limited geographic area when one species is a predator of the other and the second species consumes resources from the environment? " In the models in this chapter, instead of modeling wolf and moose, we will model
wolf and sheep. The wolf and moose data set is well established, but our goal in this chapter is not to match this particular data, but to introduce you to classic examples of predator-prey modeling and to try to reproduce the oscillating sustained pattern of population levels.

Choosing Your Agents

Now that we have identified and contextualized our driving research question, we can begin to design the components that will help us answer it.

What Are The Agents In The Model?

The first question we should ask ourselves is: What are the agents in the model? When designing our agents, we want to choose those components of our model that are autonomous and have properties, states, and behaviors that could possibly have bearing on our question.

Agent Overload

But we must be careful to avoid agent overload. Depending on the perspective one takes, almost any model component could be considered an agent. However, a model that is designed with an excess of agent types can quickly become unmanageable.

Entities Relevant to Research Question

When choosing what are to be the agents in a model, it is important to concentrate on those autonomous entities which are most relevant to our research question.
A related issue is the " granularity " of the agent. Every entity is composed of multiple smaller entities.

What is The Right Level of Entity to Choose?

What is the right level of entity to choose? Should our agent be molecules or atoms? Body organs or cells? Some agents can be treated as mass properties. If we want to model a field of grass, we might not want to model every blade of grass, but instead choose " clumps " of grass as our agents.

Granularity Of Each Agent in temporal scale

It is important that the granularity of each agent be at roughly the same level. For instance in the temporal scale, if you are modeling the sheep actions at the level of days of activity, but the grass minute by minute, that can be difficult to reconcile.

Granularity of Each Agent In Physical Scale

In the physical scale, if there is more grass than the sheep can consume then you will not see many interesting behaviors since grass will not serve as a limiting condition.

Proto-agents

If we suspect that, in the future, some model entities might need to become full agents, we can choose to design them as proto-agents. By the term proto-agent, we mean agents that do not have individual properties, states, or behaviors but instead inherit some or all of their characteristics from a global agent type. For example, in the Wolf Sheep Simple model we

might desire to have a human hunter who interacts with the other two species. This hunter might simply eliminate a part of the populations every now and then.

Create a Simpler Proto-agent

There is no need to build the hunter as a full agent at the start; instead we can create a simpler proto-agent that has the ability to kill off a random percentage of the population. Eventually, if needed, this hunter could become a full agent and have full properties and behavior just like any other agent.

Three Agent Types

Given the preceding discussion, we start our Wolf Sheep Simple model design by choosing three agent types.

We model the predators, which we will call wolves, and the prey, which we will call sheep, and the resources the sheep consume, grass. We could have added many other agents to this model. For example, we could model the hunter described above or the precipitation levels or soil nutrition. However, by choosing just wolves, sheep, and grass, we stick to the ABM design principle.

Choosing Agent Properties

Agents have properties that distinguish them from other agents. It is important to determine these properties in advance so we can conceptualize the agent and design the agents ' interaction with each other and with the environment.

Wolf Sheep Simple model - Three Properties

In the Wolf Sheep Simple model we give the sheep and wolves three properties each:

(1) an energy level, which tracks the energy level of the agent,

(2) a location, which is where in the geographic area the agent is, and

(3) a heading, which indicates the direction the agent is currently moving or would be moving.

Energy Property

The energy property is not merely describing temporary energy (such as whether an animal is fresh or fatigued). Rather, " energy " incorporates some notion of the amount of " vitality " in a creature, abstracting away the messy details of metabolism, calorie storage, or starvation, and condensing it all into a single measure.

Adding Additional Properties

We could add additional properties and some of them might be useful for future extensions. For instance, we could add a movement speed and allow different agents to move at different

speeds, or an offense/defense capability that affects the ability of the individual to predate or resist predation.

Choosing Agent Behavior

In addition to designing the structure of the agents, it is important to determine what kind of behavior the agents can exhibit. These behaviors are necessary to describe how agents interact with each other and the environment.

Wolf Sheep Simple Model

In the Wolf Sheep Simple model, sheep and wolves share many common behaviors. They both have the ability to turn randomly, move forward, reproduce and die. However, sheep and wolves differ in that sheep have the ability to consume grass and wolves have the ability to consume sheep. This differentiates the two species/agent types from each other. Of course, once again there are many other behaviors we could prescribe for these agents. For instance, we could give sheep the ability to huddle in herds to defend against wolf attacks, or the ability to fight back. The wolves could have the ability to move at different speeds from a walk to a run or to chase sheep. Wolves and sheep also engage in a number of behaviors, such as sleeping, digesting food, and seeking shelter during a thunderstorm. However, again the behaviors we have described (moving, reproducing, eating, and dying) are reasonable choices for a simple model that can address our research question. For the grass clump agents, we give one simple behavior, the ability to grow.

Choosing Parameters of the Model

We could decide to write one set of completely specified rules to control the behavior of all of these agents and their environmental interactions during a time step, but it makes more sense to create some parameters that enable us to control the model, so that we can easily examine different conditions. A next step is to define what attributes of the model we will be able to control through parameters.

Several Possible Parameters Of Interest

There are several possible parameters of interest in the Wolf Sheep Simple model.

- For instance, we will want to be able to control the number of initial sheep and wolves. This will enable us to see how different values of the initial population levels affect the final population levels.
- Another factor to control is how much energy it costs an agent to move. Using this parameter, we can make the landscape more or less difficult to traverse, and thus simulate different types of terrain.

Parameters For Controlling The Energy

Related to the cost of movement is the energy that each species gains from food. Thus, we choose to have parameters for controlling the energy gained from grass and the energy gained from sheep.

Finally, since the sheep consume grass, in order to sustain the population over time we will want the grass to regrow. So we will need a parameter for the grass regrowth rate.

Homogenous Parameters

There are many other parameters that we could have included in this model. For instance, the parameters we chose are homogenous across the model. In other words, one sheep will gain the same from grass as any other sheep.

Heterogeneous Parameters

However, we could make this model more heterogeneous by drawing the energy gain for each sheep from a normal distribution, and have two model parameters that control the mean and variance of the energy gain.

Add Parameters to Control Aspects

We could also add parameters to control aspects that we are currently planning to specify as constant values in the code. For example, we did not create a parameter to control the speed of the agents.

Summary of the Wolf Sheep Simple Model Design

Now that we have gone through the major design steps, we can create a summary document that describes our model design. The Wolf Sheep Simple model can be described in the following way:

Driving Question Under what conditions do two species sustain oscillating positive population levels in a limited geographic area when one species is a predator of the other and the second species consumes limited but regenerating resources from the environment?

Agent
Agent Types Sheep, Wolves, Grass
Agent Properties Energy, Location, Heading (wolf and sheep), Grass-amount (grass)
Agent Behaviors Move, Die, Reproduce (wolf and sheep), Eat-sheep (wolf only), Eat-grass (sheep only), Regrow (grass)

Parameters Number of Sheep, Number of Wolves, Move Cost, Energy Gain From Grass, Energy Gain From Sheep, Grass Regrowth Rate
***Time Step*:**
1. Sheep and Wolves Move
2. Sheep and Wolves Die
3. Sheep and Wolves Eat
4. Sheep and Wolves Reproduce
5. Grass Regrows

Measures Sheep Population versus Time, Wolf Population versus Time It is quite useful while designing a model to write notes to yourself, as we have done in this section. You will find these notes invaluable after you have left the model for a while, since you will be able to

go back and recall why you made certain decisions and alternative choices that you considered. Also, for the purposes of explaining your model to others, it is very helpful to have such documentation about the model. Finally, we recommend that you date your notes as you create them so that you can track your model design process.

Set Of Questions

For instance, if you are using the top-down design process that we have just discussed, then you might look over the following set of questions and write down answers to them as a provisional guide for how to build your model:

1. What part of your phenomenon would you like to build a model of?
2. What are the principal types of agents involved in this phenomenon?
3. In what kind of environment do these agents operate? Are there environmental agents?
4. What properties do these agents have (describe by agent type)?
5. What actions (or behaviors) can these agents take (describe by agent type)?
6. How do these agents interact with this environment or each other?
7. If you had to define the phenomenon as discrete time steps, what events would occur in each time step, and in what order?
8. What do you hope to observe from this model?

Bottom-up Approach

A more bottom-up approach would not start with these questions. Instead, you might know only that you want to build some kind of ecological model and could begin by creating some sheep and have them spread out in the world.

Next, you might think of and start implementing some behaviors for the sheep such as moving, taking steps, eating, reproducing and dying.

Implementing Some Behaviors

In order for the sheep to eat, you might decide to add grass to the model.

Having sheep eat grass provokes the question, “What eats sheep?” Therefore, you modify the model to include wolves, as predators for the sheep. This process allows the wolf sheep model to be designed and even implemented without first considering the final goal of the model.

Examining a Model

We have found one set of parameters that exhibits the behavior of our reference pattern.

We note that these particular values for the parameters do not correspond to any real predator and prey populations. We did not calibrate our model from real-world data, so the parameter values themselves are not important. But, discovering that there exist model parameters that exhibit our reference pattern gives us insight into the natural phenomenon we were trying to model.

Parameters of two Species

Now that we have built our model, and we have found a set of parameters that allow the wolf and sheep populations to coexist and to oscillate, we have partially answered our research question: What parameters of two species will sustain oscillating positive population levels in a limited geographic area when one species is a predator of the other and the second species consumes limited but regrowing resources from the environment? We now know that it is possible for the rules we set up to produce the target reference pattern and therefore they could be a possible generative explanation for that pattern.

Observing the Behavior Of Model Once

However, just observing the behavior of this model once does not provide us with a robust answer. First, because many components of our model are stochastic in nature, there is no guarantee that the model when run again with the exact same parameters will exhibit the same behavior. Second, we have found one set of parameters that allow wolves and sheep to exist, but are there other parameter settings? A more general answer would allow us to make statements about ranges of parameters, and relationships between parameters, that allow both the wolves and the sheep to survive.

Variety of Parameter Settings

However, if we run the model many times with a variety of parameter settings, this will create a lot of data. Thus, to give us even a simple answer to our question, we really need to examine multiple runs across multiple different parameter settings and summarize this data in a useful way.

Multiple Runs

Whenever you have a model that has stochastic components, it is important to run the model several times so that you can be certain you have correctly characterized the behavior of the model.

Anomalous Behavior

If the model is run only once, you might happen to see anomalous behavior that is not what the model usually produces. For instance, in the Wolf Sheep Simple model it might be possible to run the model and arrive at a state in which there is simply one wolf and one sheep, and the sheep produces a second sheep often enough to keep the wolf fed, but not often enough to produce three sheep. However, due to the way the wolves and the sheep wander around the landscape, such an outcome is extremely unlikely, and it would not be typical of the model.

Run the Model Multiple Times

Thus, it is important to run the model multiple times so that you can characterize the normal/average behavior of the model and not the aberrant behavior of the model. On the other hand, there are times when the anomalous/aberrant model behavior *is* what you are interested in investigating, in which case you will need to run the model many times in order to find it and characterize it.

***BehaviorSpace* Tool**

Most ABM platforms provide a way to do this. NetLogo provides you with the BehaviorSpace tool (*Wilensky & Shargel, 2002*) that enables you to run a model for several iterations with the same (or different) parameter settings and collect the results.

Number of Time Steps

One additional consideration when performing multiple runs is how many time steps to run the model for. Since we want to be able to compare the results across different random number seeds, it is useful to hold the number of time steps we run the model constant.

Predator-Prey Models: Additional Context

One of the first uses of agent-based modeling was for modeling of ecosystems, where it was often called individual-based modeling (see the appendix for a brief discussion of the historical role of individual-based modeling). Modeling of ecological systems has been of interest to biologists and environmentalists for quite some time. If we can better understand how ecologies operate and what the effects of successful ecosystems on the global environment are, then we may be able to better intervene to assist in the sustenance of these systems. For more information on this topic, refer to Grimm and Railsback (2005) .

Predator-prey Interaction

One of the first attempts to study ecologies and population dynamics in a concrete way was the work of Lotka (1925) and Volterra (1926) . They developed a system of equations to describe a two species predator-prey interaction.

Lotka-Volterra Equations

These simple equations showed that you could meaningfully model ecological systems with just a few parameters. And once you have a model of a system you can start to explore options for perturbing the system into favorable states. The general result of the Lotka-Volterra equations is that predators and prey populations move in cycles. This is seen in figure 4.11.

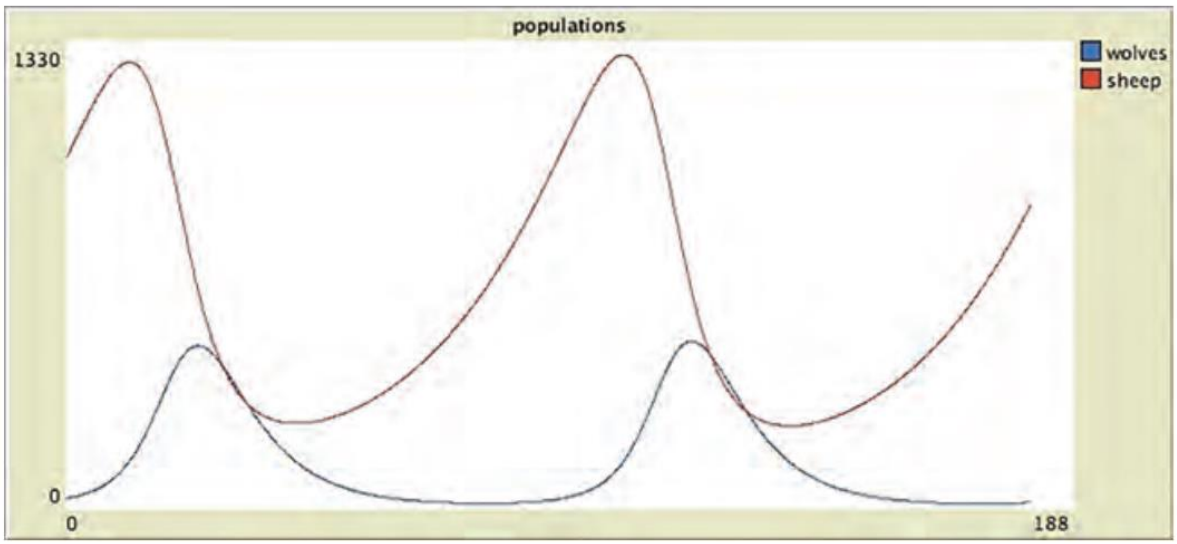


Figure 4.11
Lotka-Volterra relationship (Wolf Sheep Predation System Dynamics model) (*Wilensky, 2005*).

Depending on the parameter settings, if the two populations are equal at the start, then the predators will begin consuming the prey; this will result in a decrease of the prey population.

Prey and Predator Population

Eventually, the prey will die off to the point where predators cannot find any more prey to eat. This will in turn cause predators to die off because they have no prey to feed on. At this point the prey will be able to reproduce before the predators eat them, and the prey population will increase. As the prey population increases it will become easier for the predator population to find them and the predator population will increase as well.

This, in turn, results in the prey population ' s decreasing trend, and the whole process begins again.

Lotka-Volterra Equations

The Lotka-Volterra equations have been a standard way of describing the fluctuations in predator and prey populations. Because they are differential equations (see figure 4.12), they represent a continuous model of population change.

$$\frac{dx}{dt} = \alpha x - \beta xy$$

$$\frac{dy}{dt} = \delta xy - \gamma y$$

Figure 4.12
Classic Lotka Volterra differential equations. x is the number of prey, y is the number of predators, and α , β, γ, and δ are parameters describing the interactions of the two species.

But populations are not continuous; they are discrete. Sometimes a continuous approximation of a discrete process is fine, but in this case it may be an oversimplification. The general problem is that as the population of a species becomes very small, standard differential equation models do not allow the population to actually go to 0, which means that there is always a positive probability that the population will rebound. However, this is not what happens in the real world. There is no rebounding from 0.1 prey. If a species goes extinct, it goes extinct, and there is no probability that it will rebound.

“ Nano-sheep Problem ”

This phenomenon is so common in differential equation-based modeling that it is sometimes referred to as the “ nano-sheep problem ” in specific reference to the wolf-sheep/predator-prey model (Wilson, 1998). The problem is that nano-sheep (i.e., a millionth of a sheep) do not exist: Sheep either exist or do not exist, and therefore modeling them with a continuous distribution can be problematic. To rectify the nano-sheep problem, biologists have created agent-based (or to use the term biologists favor, individual-based) models of predator-prey relationships. Under certain conditions these models have been shown to replicate the Lotka-Volterra results, but without the nano-sheep problem. They also predict that this system is in fact highly susceptible to extinctions, something that the Lotka-Volterra equations fail to capture (Wilson et al., 1993 ; Wilson, 1998).

Gause Work

In 1934, Gause showed that indeed for isolated predator and prey (with no other competing species) the agent-based model was more accurate in its predictions — indeed, extinctions happen much more frequently than the Lotka-Volterra equations predict.

Advanced Modeling Applications

More broadly, the modeling of environmental and ecological systems has a long and rich history, representing some of the first applications of an agent-based modeling paradigm (DeAngelis & Gross, 1992). It also continues to be an exciting area for current research.

Modeling of Food Webs

One use of agent-based models in ecological systems is the modeling of food webs (Yoon et al., 2004). This combines ABM with another new complex systems methodology, network analysis (Schmitz & Booth, 1997).

Structure of Ecological Interactions

By understanding both the structure of ecological interactions via network analysis and the process of animal interactions via ABM, researchers gain a deeper insight into overall ecological systems (See, e.g., figure 4.13).

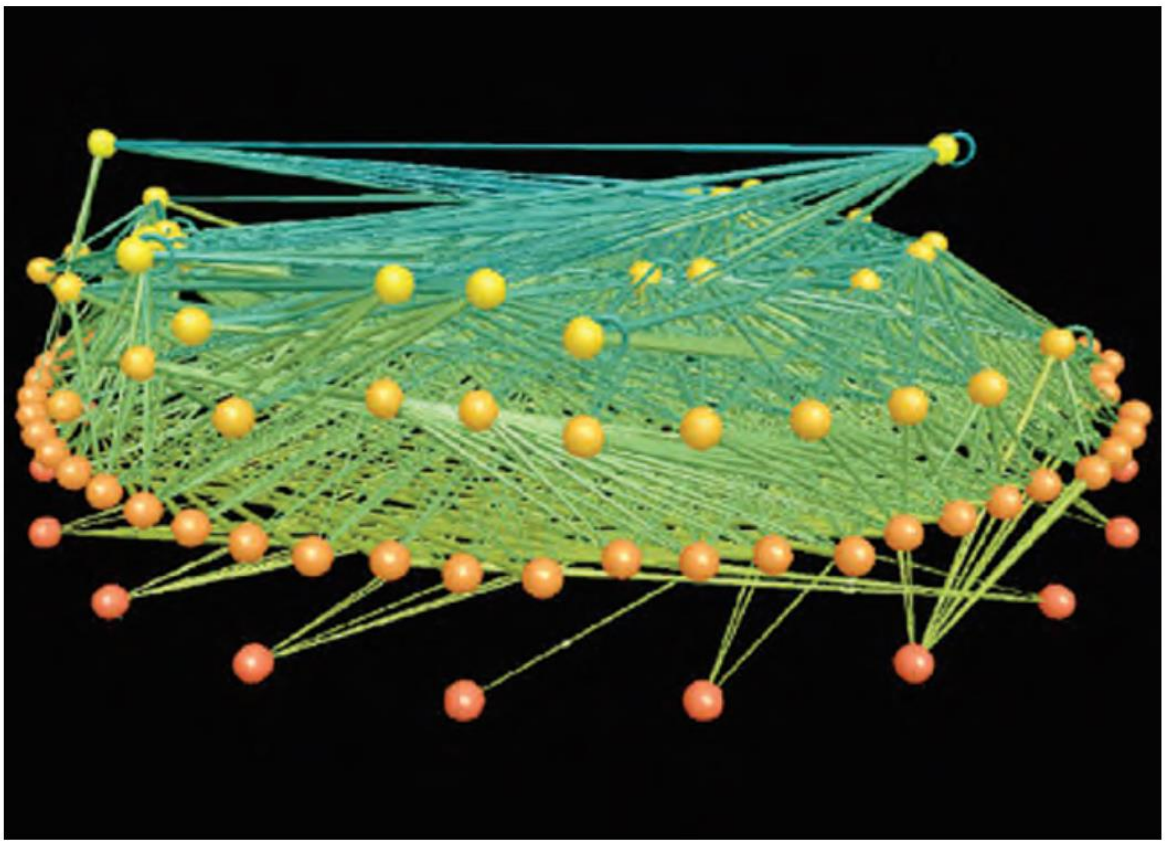


Figure 4.13
Little Rock Lake food web (foodwebs.org, 2006).

ABM and Ecological Modeling
Another particularly interesting application of ABM and ecological modeling has been in doing prescriptive design of engineered systems to ameliorate human interventions in the environment.

Modeling Of Fish Populations

For instance, Weber and his colleagues have done significant modeling of fish populations, and then examined the effect of various fish ladders near dams on salmon populations (Weber et al., 2006).

Ecological models can also be underpinnings of models of evolution. Agent-based modeling has been frequently used to model evolution of organisms (Aktipis, 2004; Gluckmann & Bryson, 2011; Hillis, 1991 ; Wilensky & Novak, 2010).

ABM Approach as Natural Selection

Evolution lends itself well to the ABM approach as natural selection and other evolutionary mechanisms can be thought of as computational algorithms. Models can be used to try to understand adaptation and speciation in the historical record. Figure 4.14 shows two examples of such models of evolution from the NetLogo models library.

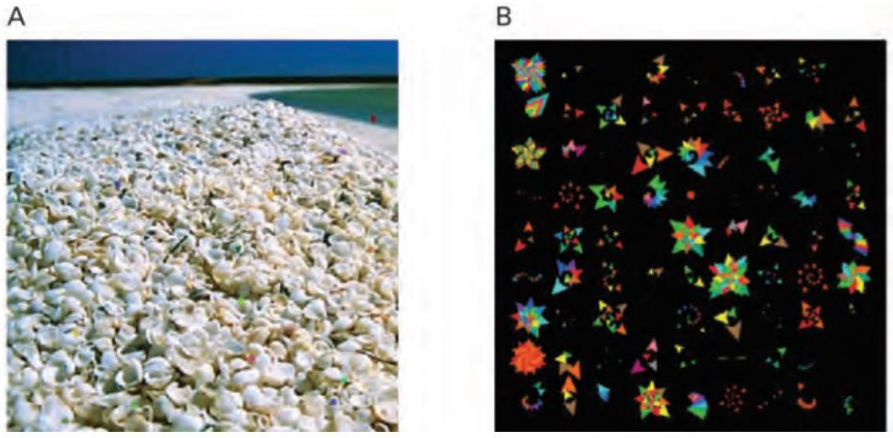


Figure 4.14
Agent-based models of evolution. (A) Evolution of camouflage of insects on a landscape. Different landscapes lead to the bugs evolving colorations that camouflage them in that landscape. (http://ccl.northwestern.edu/netlogo/models/BugHuntCamouflage.) (B) Evolution of computational biomorphs. Artificial flowers evolve through mating and blending their characteristics. (http://ccl.northwestern.edu/netlogo/models/SunflowerBiomorphs.)

The Components of Agent-Based Modeling

Components of ABM

The main components of any ABM are *agents*, *environment*, and *interactions*. Agents are the basic ontological units of the model, while the environment is the world in which the agent lives.

Distinction Between Agents and Environment

The distinction between agents and environment can be fluid, as the environment can sometimes be modeled as agents. Interactions can occur either between agents or between agents and the environment.

Agent Actions

Agent actions can also occur internally, directly affecting only the agent ' s internal state. Such is the case when an agent is trying to determine which action to take, as when the residents in the Segregation model are deciding whether or not they are unhappy. The environment is not merely passive; it can also act autonomously. For instance, in the models we discussed in chapter 4, the grass regrows on its own.

Additional Components

To this list of three basic components, we will also add two additional components.

Observer/User Interface

The first is called the Observer/User Interface. The observer is an agent itself,1 but one that has access to all of the other agents and the environment. The observer asks agents to carry out specific tasks.

User Interface

Users of agent-based models can interact with the agents by way of the User Interface, which enables the user to tell the observer what the model should do. The second component, the Schedule, is what the observer uses to tell the agents when to act.

Schedule

The Schedule often involves user interaction as well. In NetLogo models, the interface typically includes SETUP and GO buttons. The user presses SETUP and then GO to schedule these events to occur.

We will now describe each of these five components in further depth. Throughout this chapter, we will use a variety of models from the NetLogo models library to illustrate our examples. We start by examining the Traffic Basic model, a simple model of traffic flow (found in the Social Science section of the library). An early version of this model was designed by two high school students (Resnick, 1996; Wilensky & Resnick, 1999) to explore how traffic jams form. The students thought that they would have to include traffic accidents, radar traps, or some other form of traffic diversion to create a traffic jam, but they built an initial model without any such hindrances. To their surprise, traffic jams still formed despite the lack of impediments. This happens because as cars speed up and approach the cars in front of them, they eventually have to slow down, causing the cars behind them to slow down creating a ripple effect backward. Eventually, the car at the front of the jam will be able to move again, but by that time, there are many cars behind that car that cannot move, causing the traffic jam to move backward even as the traffic jam moves forward.2 This simple model (figure 5.1) again illustrates that emergent phenomena often do not correspond to our intuitions.

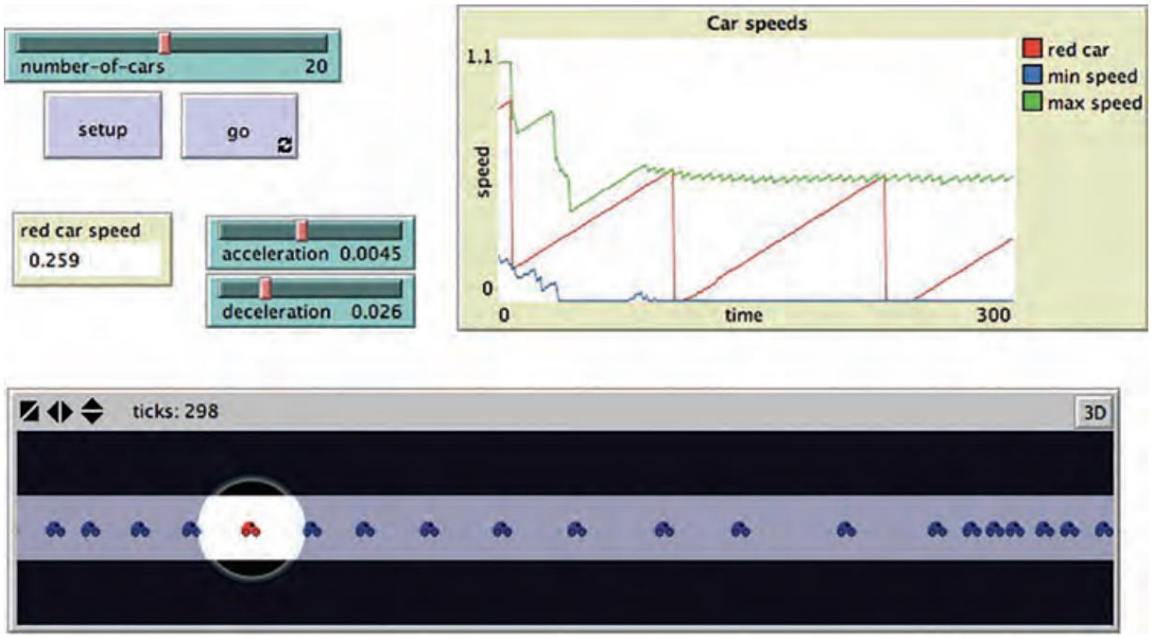


Figure 5.1
Traffic Basic model (Wilensky, 1997b).

Agents

Agents are the basic units of agent-based modeling. As such, it is important to choose the design of your agents carefully.

Properties and Actions

The two main aspects that define agents are the ***properties*** that they have and the ***actions*** (sometimes called behaviors or methods) that they can execute. ***Agent properties*** are the internal and external state of the agents — their data and description. ***Agent behaviors or actions*** are what the agents can do. Besides these two main agent attributes, there are also several issues that are related to agent design.

"Grain-size"

First is the issue of agent "grain-size": which is most effective for the chosen model? For example, if you are modeling a political system, do want to model the individual actors or, instead, the political institutions or even each nation' s government as a single entity?

Agent Cognition

A second factor to consider is agent cognition. How much capability do the agents have to observe the world around them and make a decision? Do they act in a stimulus-response fashion? Or do they plan out their actions?

Proto-agents and Meta-agents

Finally, we discuss some special types of agents:

- ***proto-agents***, which are not fully specified agents; and
- ***meta-agents***, composed of other agents.

Properties

Agent properties describe an agent ' s current state, the items that you see when you inspect an agent. Earlier, we briefly described how to use a patch monitor to see the current state of a patch, but you can also use monitors to inspect turtles and other types of agents as well. In this example, we are going to explore agent properties using a turtle monitor, but in NetLogo, you can monitor links and patches as well.

If you inspect one of the cars in Traffic Basic, you see (under the graphical image of the agent ' s local environment) the list of properties described in figure 5.2. This list contains two sets of properties. The first set is a standard set of properties of every turtle created in NetLogo: WHO, COLOR, HEADING, XCOR, YCOR, SHAPE, LABEL, LABEL-COLOR, BREED, HIDDEN?, SIZE, PEN-SIZE, and PEN-MODE. Patches and links also have a default set of properties. For patches, these are PXCOR, PYCOR, PCOLOR, PLABEL, and PLABEL-COLOR, while for links, these are END1, END2, COLOR, LABEL, LABEL-COLOR, HIDDEN?, BREED, THICKNESS, SHAPE, and TIE-MODE.

NetLogo

In NetLogo, turtles and links have COLOR as a property, while patches have PCOLOR as a property. Likewise, turtles have XCOR and YCOR as properties, while patches have PXCOR and PYCOR. To make things simpler, turtles can directly access the underlying properties of their current patch. For instance, for a turtle to set its color to the color of the underlying patch (effectively making it invisible), the turtle can just execute SET COLOR PCOLOR. If the patch property had the same name as the turtle property, this would not work, since there would be a confusion over which COLOR the code was describing. Only turtles can access patch properties directly, since a turtle can only be on one and exactly one patch. Links can span multiple patches and are always connected to multiple turtles, so you have to specify which patch you are referring to if you want to use a patch property in a link procedure. Similarly, you have to specify which link you are referring to when you want to use a link property in a turtle or patch procedure.

Link Properties

Moreover, patches cannot directly access turtle or link properties, because a patch could have 0, 1 or many links or turtles on it. Therefore, you must specify which turtle or link you are referring to when accessing their properties.

In the inspector window, the default properties of the agent appear first. These are

followed by the properties that the model author has specifically added to the agents for the model (e.g., in figure 5.2, SPEED, SPEED-LIMIT, SPEED-MIN). These authordefined properties should be described in the Info Tab associated with the model as well as in the comments in the Code Tab.

Figure 5.2
Agent properties for the Traffic Basic model.

Agent Properties for the Traffic Basic Model

For this model, SPEED describes the current speed of the car, SPEED-LIMIT is the maximum speed of the car and SPEED-MIN, its minimum speed.

When the Traffic Basic model starts up, all of these properties are set in the SETUPCARS procedure:

Properties

Agent properties describe an agent ' s current state, the items that you see when you inspect an agent. Earlier, we briefly described how to use a patch monitor to see the current state of a patch, but you can also use monitors to inspect turtles and other types of agents as well. In this example, we are going to explore agent properties using a turtle monitor, but in NetLogo, you can monitor links and patches as well.

If you inspect one of the cars in Traffic Basic, you see (under the graphical image of the agent ' s local environment) the list of properties described in figure 5.2. This list contains two sets of properties. The first set is a standard set of properties of every turtle created in NetLogo: WHO, COLOR, HEADING, XCOR, YCOR, SHAPE, LABEL, LABEL-COLOR, BREED, HIDDEN?, SIZE, PEN-SIZE, and PEN-MODE. Patches and links also have a default set of properties. For patches, these are PXCOR, PYCOR, PCOLOR, PLABEL, and PLABEL-COLOR, while for links, these are END1, END2, COLOR, LABEL, LABEL-COLOR, HIDDEN?, BREED, THICKNESS, SHAPE, and DE-MODE.

NetLogo

In NetLogo, turtles and links have COLOR as a property, while patches have PCOLOR as a property. Likewise, turtles have XCOR and YCOR as properties, while patches have PXCOR and PYCOR. To make things simpler, turtles can directly access the underlying properties of their current patch. For instance, for a turtle to set its color to the color of the underlying patch (effectively making it invisible), the turtle can just execute SET COLOR PCOLOR. If the patch property had the same name as the turtle property, this would not work, since there would be a confusion over which COLOR the code was describing. Only turtles can access patch properties directly, since a turtle can only be on one and exactly one patch. Links can span multiple patches and are always connected to multiple turtles, so you have to specify which patch you are referring to if you want to use a patch property in a link procedure. Similarly, you have to specify which link you are referring to when you want to use a link property in a turtle or patch procedure.

Link Properties

Moreover, patches cannot directly access turtle or link properties, because a patch could have 0, 1 or many links or turtles on it. Therefore, you must specify which turtle or link you are referring to when accessing their properties.

In the inspector window, the default properties of the agent appear first. These are

followed by the properties that the model author has specifically added to the agents for the model (e.g., in figure 5.2, SPEED, SPEED-LIMIT, SPEED-MIN). These authordefined properties should be described in the Info Tab associated with the model as well as in the comments in the Code Tab.

Figure 5.2
Agent properties for the Traffic Basic model.

Agent Properties for the Traffic Basic Model

For this model, SPEED describes the current speed of the car, SPEED-LIMIT is the maximum speed of the car and SPEED-MIN, its minimum speed.

When the Traffic Basic model starts up, all of these properties are set in the SETUPCARS procedure:

The Granularity of an Agent

One of the first considerations when designing an agent-based model is at what granularity should you create your agent — at what level of complexity is the agent you are modeling.

Level of Complexity for the Agents

So how do you choose the level of complexity for the agents in your model? The guideline is to choose agents such that they represent the fundamental level of interaction that pertains to your question about the phenomenon.

Tumor Model

For instance, if you look at the Tumor model (shown in figure 5.6) in the Biology section of the NetLogo models library, the agents are cells within the human body, since the research question the model examines is how tumor cells spread.

NetLogo Tumor model

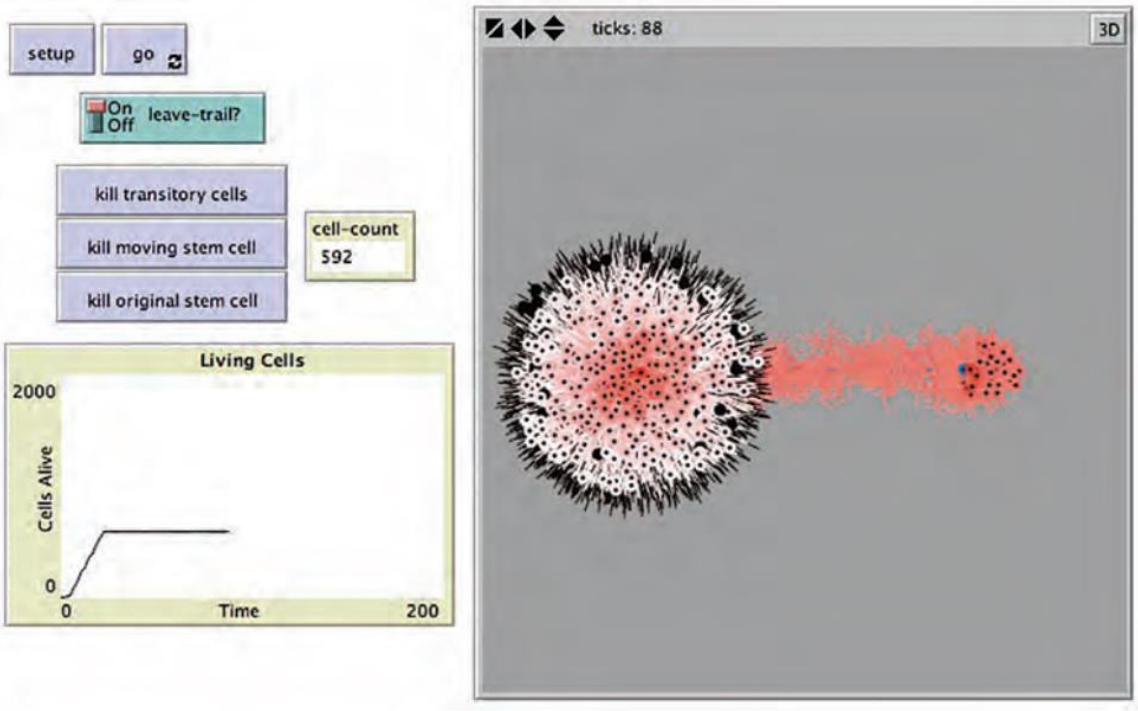


Figure 5.6
NetLogo Tumor model. http://ccl.northwestern.edu/netlogo/models/Tumor (*Wilensky, 1998b*).

However, even though the AIDS model in the Biology section of the NetLogo models library (see figure 5.7) is also concerned with disease, the basic agents are humans instead of cells.

NetLogo AIDS model

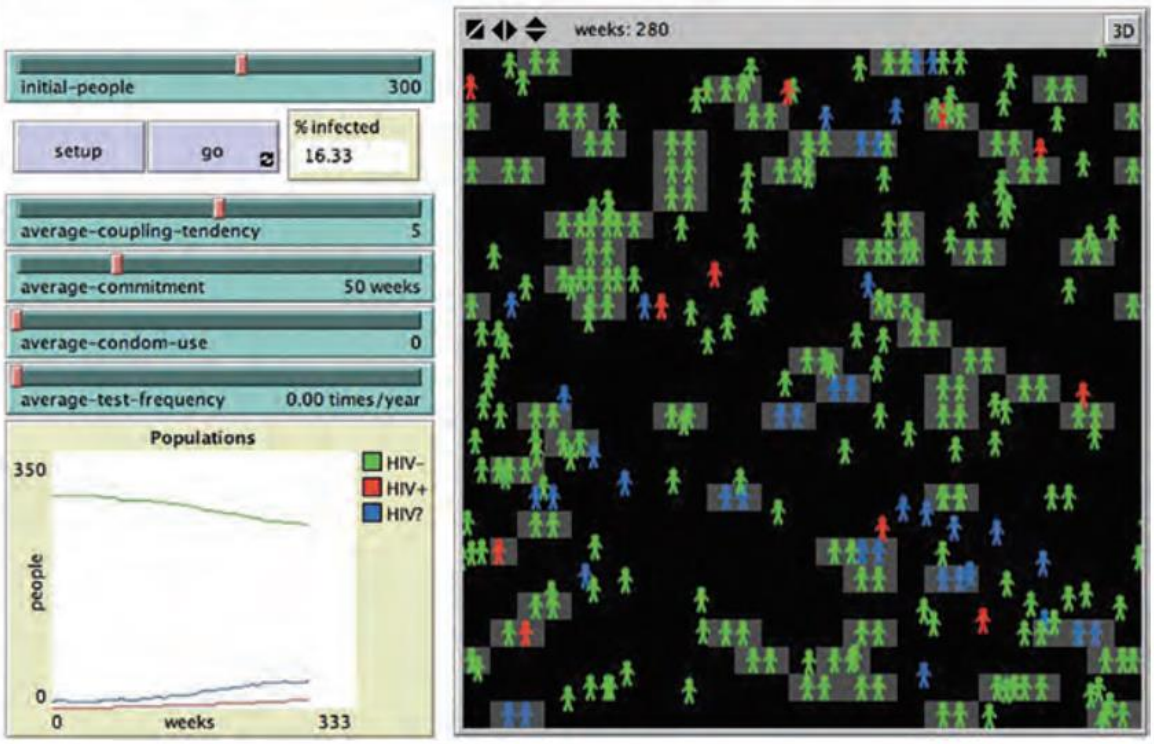


Figure 5.7
NetLogo AIDS model. http://ccl.northwestern.edu/netlogo/models/AIDS (*Wilensky, 1997a*).

Question Of Concern

This is because the AIDS model is more concerned with how the disease spreads between humans instead of how it spreads within the human body. In the Tumor model, the question of concern is how cells interact to create cancer.

Thus, the most important interactions happen at the cell level, and this makes cells a good choice for agents In the AIDS model, the question of concern is how humans interact to spread disease, with humans as the agents being the most appropriate choice.

Choices Of Granularity

These two models highlight the relationship between the question being investigated, the interactions being modeled, and the resulting choices of granularity for the agents.

Level Of Complexity

Both individual cells and groups of cells can be valid choices for the level of complexity of these agents, and deciding between them will depend on details of the question you are asking as well as considerations of computational complexity.

Agent Cognition

As we have described, agents have different properties and behaviors. Still, how do agents examine their properties and the world around them to decide what actions to take? This question is resolved by a decision-making process called agent cognition.

Types Of Agent Cognition

We will discuss several types of agent cognition:

- *reflexive agents*,
- *utility-based agents*,
- *goal-based agents*, and
- *adaptive agents* (Russell & Norvig, 1995).

Often, these types of cognition are thought of in increasing order of complexity, with reflexive agents being the simplest, and adaptive agents the most complex. In reality, though this is not a strict hierarchy of complexity.

Instead, these are descriptive terms that help us talk about agent cognition and can be mixed and matched; for instance, it is possible to have a utility-based adaptive agent.

Reflexive

Reflexive agents are built around very simple rules. They use if-then rules to react to inputs and take actions (Russell & Norvig, 1995). For instance, the cars in the Traffic Basic model are reflexive agents. If you look at the code that controls their actions, it looks like this:

```
let car-ahead one-of turtles-on patch-ahead 1  ;;choose a car on the patch ahead
ifelse car-ahead != nobody [  ;; if there is a car ahead
    slow-down-car car-ahead ;; set car speed to be slower than car ahead
]
[   ;; otherwise, speed up
    speed-up-car ;; increase speed variable by the acceleration
]
;; ….
fd speed
```

Put in words, this code says that if there are cars ahead, then slow down below the speed of the car in front; if there are not any cars ahead, then speed up. This is a reflexive action based on the state of the program, hence the name *reflexive agent*. This is the most basic form of agent cognition (and often a good starting point), but it is possible to make the agent cognition more sophisticated.

Utility-based Form of Agent Cognition

For instance, we could elaborate this model by giving the cars gas tanks and fuel efficiencies based on the speed they are going. We could then have the cars change their speeds in order to improve their fuel efficiencies. As a result, the agents might have to speed up and slow down at different times than they currently do to minimize their gas usage while still not causing accidents. Giving the agents this type of decision-making process

would give them a *utility-based* form of agent cognition in which they attempt to maximize a utility function — namely, their fuel efficiency (Russell & Norvig, 1995).

Implementation Of Model Of Agent Cognition

To implement this model of agent cognition, we need to start by replacing our SPEED-UP-CAR procedure with a new procedure that accounts for the car ' s fuel efficiency.

```
;; choose a car on the patch ahead
let car-ahead one-of turtles-on patch-ahead 1
ifelse car-ahead != nobody [ ;; if there is a car ahead
    slow-down-car car-ahead  ;; set car speed to be slower than car ahead
]
[ ;; otherwise, adjust speed to find ideal fuel efficiency
    adjust-speed-for-efficiency ]
```

We want the ADJUST-SPEED-FOR-EFFICIENCY procedure to maintain the same speed if the car is at the maximally fuel efficient speed. If it is not at its most efficient speed, the logic in this procedure should have the car speed up if it is moving too slow and slow down if the car is moving too fast. Additionally, the car would still need to slow down if it was about to crash into the car in front of it. As such, the true utility function includes an exception that gives a utility of 0 to any action that results in a crash. As a result, we can leave the first part of the code that slows the car before it crashes and just add ADJUST-SPEED-FOR_EFFICIENCY when there is no car directly ahead, as illustrated in the code below from the Traffic Basic Utility model, of the NetLogo models library.

```
;; car procedure
to adjust-speed-for-efficiency
 if (speed != efficient-speed) [ ;; if car is at efficient speed, do nothing
         if (speed + acceleration < efficient-speed) [
;; if accelerating will still put you below the efficient speed then accelerate
             set speed speed + acceleration
         ]                                        ;;
;; if decelerating will still put you above the efficient speed then decelerate
         if (speed - deceleration > efficient-speed) [
             set speed speed - deceleration
         ]
    ]
end
```

Utility Functions

In the language of utility functions, each car agent is minimizing a function f,

defined by:

$$f(v) = |v - v^*|$$

where v is the current velocity of the car and v* is the most efficient velocity. The constraint that is described in the code is that the v cannot be adjusted arbitrarily, but instead, can only be

changed by a limited increment every time step. By trying different values for EFFICIENT-SPEED, you can obtain quite different traffic patterns from the original model. For low values of EFFICIENT-SPEED, the system can achieve a free-flow state (with no jams) on a consistent basis.

Goal-based Agents

The third type of agent cognition we will discuss is goal-based cognition. Imagine that each car in the Traffic Grid model (see figure 5.8) from the social sciences section of the NetLogo models library has a home and a place of work, with the goal of moving from home to work in a reasonable amount of time. Now, not only do the agents have to be able to speed up and slow down, but they also have to be able to turn left and right. In this version of the model, the cars are ***goal-based agents***, since they have a goal (getting to and from work) that they are using to dictate their actions.

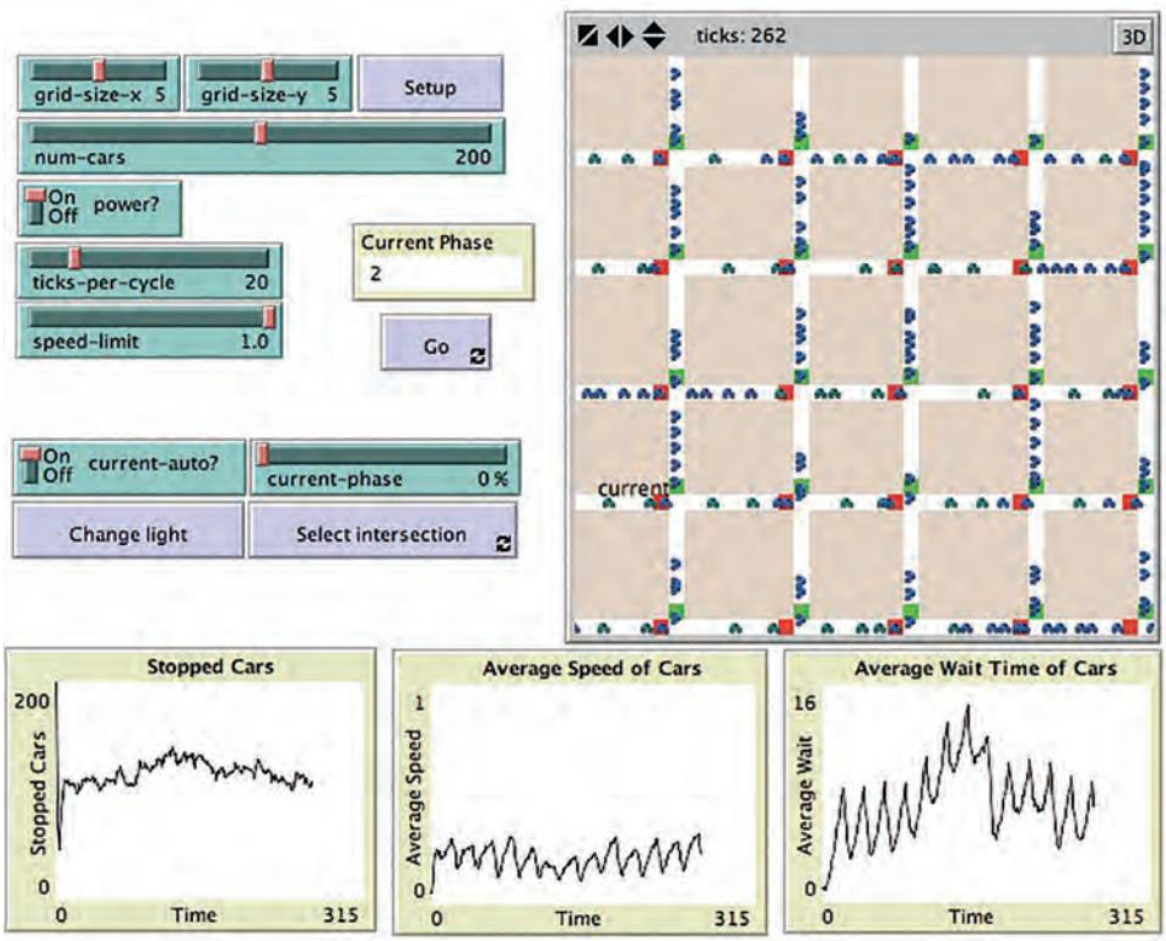


Figure 5.8
Traffic Grid model. http://ccl.northwestern.edu/netlogo/models/TrafficGrid (*Wilensky, 2002b*).

Implementing The Goal-based Version of the Model

To implement the goal-based version of the model, we start by defining the procedure that the cars will use to navigate between their two desired destinations. Instead of just moving straight all the time, as with the cars in the original model, the new model should have the car at each

step deciding which of its neighboring road patches is the closest to its destination, subsequently proceeding in that direction. We do this in the procedure called NEXT-PATCH. First, each car checks if it has arrived at its goal, and if so, switches its goal:

```
;; if I am going home and I am on the patch that is my home
;; I turn around and head towards work (my goal is set to "work")
    if goal = house and patch-here = house [
       set goal work
    ]
;; if I am going to work and I am on the patch that is my work
;; I turn around and head towards home (my goal is set to "home")
    if goal = work and patch-here = work [
       set goal house
    ]
```

The code above doesn ' t quite work. That ' s because the house and work are off-road and the cars remain on the road, so the condition " patch-here = house " or " patch-here = work " will never be satisfied. As we placed the house and work adjacent to the road, we can fix this by having the car check if it is right next to its goal.

```
;; if I am going home and I am next to the patch that is my home
;; I turn around and head towards work (my goal is set to "work")
if goal = house and (member? patch-here [neighbors4] of house) [
       stay set goal work
    ]
;; if I am going to work and I am next to the patch that is my work
;; I turn around and head towards home (my goal is set to "home")
    if goal = work and (member? patch-here [neighbors4] of work) [
        stay set goal house
    ]
```

Having established a goal, each car then chooses an adjacent patch to move to that is the closest to its goal. It does this by choosing candidate patches to move to (adjacent patches on the road) and then selecting the candidate closest to its goal.

The resultant NEXT-PATCH procedure is:

```
;; establish goal of driver
to-report next-patch
;; if I am going home and I am next to the patch that is my home
;; I turn around and head towards work (my goal is set to "work")
if goal = house and (member? patch-here [neighbors4] of house) [
        stay set goal work
    ]
;; if I am going to work and I am next to the patch that is my work
;; I turn around and head towards home (my goal is set to "home")
    if goal = work and (member? patch-here [neighbors4] of work) [
        stay set goal house
    ]
;; CHOICES is an agentset of the candidate patches which the car can
;; move to (white patches are roads, green and red patches are lights)
let choices neighbors with [pcolor = white or pcolor = red or pcolor = green]
          ;; choose the patch closest to the goal, this is the patch the car will move to
    let choice min-one-of choices [distance [goal] of myself]
;; report the chosen patch
    report choice
end
```

Adaptive Agent

One of the powerful advantages of agent-based modeling is that agents can change not only their decisions but also their strategies (Holland, 1996).

Adaptive Agent: An agent that can change its strategy based on prior experience is an ***adaptive agent***. Unlike conventional agents, which will also do the same thing when presented with the same circumstance, adaptive agents can make different decisions if given the same set of inputs.

Traffic Basic model

In the Traffic Basic model, cars do not always take the same action; they operate differently based on the cars around them by either slowing down or speeding up. However, regardless of what has happened to the cars in the past (i.e., whether they got stuck in traffic jams or not) they will continue to take the same actions in the same conditions in the future. To be truly adaptive, agents need to be able to change not only their actions in time, but also their strategies.

They must be able to change how they act because they have encountered a similar situation in the past and can react differently this time based on their past experience.

In other words, the agents learn from their past experience and change their behavior in the future to account for this learning.

Example Of Adaptive Cognition

For instance, the agent could have observed that in the past that, when there was a car five patches ahead of it, it braked too quickly, even though it could have waited longer to brake without hitting the other car. In the future, it could change its rule for braking to brake only when a car is four patches ahead. This agent is now an ***adaptive agent*** because it not only modifies its ***actions***, but also, its ***strategies***.

Other Kinds of Agents

We have already discussed breeds and various types of agents, but there are two other special kinds of agents that deserve at least a brief mention.

- The first type are ***meta-agents***, *agents made up of other agents*;
- The second is ***proto-agents***, *placeholder agents that allow you to define interactions for your fully defined agents with other entities that have not been fully developed.*

Meta-Agents Many things that we would consider agents in reality are actually composed of other agents. In reality, all " agents " are composed of other agents; to cite the oft-told parable, " It ' s turtles all the way down. " In other words, there is always a lower level of detail that you can use to describe an agent in your model, at least until you reach the most basic level so far described by physics. For instance, if we have selected a human to be the agent, he or she is actually composed of many subagents, with these subagents in turn being different depending on how you view the human. You could view the subagents of a human as the systems of the body — e.g., the immune and the respiratory systems — or you could view the subagents of a human as psychological aspects like the intellect and emotion. Moreover, these subagents are not the most basic level; in our example, the systems of the body are made up of organs, tissues, and cells.

Meta-agents and Sub-agents

At each of these levels, we are describing the relationship between ***meta-agents*** *(agents composed of other agents)* and ***subagents*** *(agents which compose other agents)*. Still, agents can also be both meta-agents and subagents at the same time. For instance, organs are composed of cells and are meta-agents. However, they also play a compositional role in the systems of the human body and are thus subagents.

We can use meta-agents in our ABMs by defining the subagents that make up our agents and providing them with their own actions and properties.

Meta-agent Appears to be a Single Agent

From the perspective of another meta-agent, a meta-agent appears to be a single agent. If a person meets another person, they do not (usually) directly interface with the heart or the lungs (unless the meeting takes place at the surgical table). Instead, they interface with the person as a whole.

Modeling Agents and Their Interactions

When we think of modeling agents and their interactions, we have to determine what level of granularity we want to describe the agent behaviors. However, we always have the option of refining our model by converting agents into meta-agents that describe the agents that constitute other agents. Sometimes, it can be useful to represent the agents in our models not as autonomous individuals, but instead, as meta-agents composed of other agents. NetLogo doesn ' t include explicit language support for these meta-agents, though there are commands for locking together the movement of several agents (e.g., the TIE command) that may be useful

in some circumstances. However, there is nothing to prevent you from designing models that have groups of agents representing single agents.

Proto-Agents

To truly be an agent within the agent-based modeling framework, an entity must have its own properties or actions. Sometimes, though, it can be desirable to create agents that, rather than having their own properties or behaviors, are instead placeholders for future agents. We call these agents ***proto-agents***, and their primary purpose is to enable us to specify how other agents would interact with them if they

were fleshed out into full agents.

Example of Proto-agent

For instance, if you were creating a model of residential location decision making, you would have residents as agents; however, since where a resident lives is greatly influenced by places of employment and services (e.g., grocery stores and restaurants) you might also want to include these " service centers " as agents as well. The same level of detail necessary for the residents, who are the focal agents of concern, may not be necessary for the service centers. Instead, they might be rendered as placeholders to represent where residents might potentially find jobs and transact business. However, as you continue to refine the model, you might give the service centers additional decision-making abilities. For instance, they might have a more elaborate model of market demand and decide where to locate using their own properties and beliefs about the future growth of the world. Keeping these agents as proto-agents early on means that when you add in the more richly detailed versions to the model, you do not have to go back and revise your resident agents. Instead, you can use the interaction events that they had already been using to interface with the service center proto-agents.

Environments

Another early and critical decision is how to design the ***environment*** of the agent-based model. The environment consists of the conditions and habitats surrounding the agents as they act and interact within the model. The environment can affect agent decisions, and, in turn, can be affected by agent decisions.

Environment Can Affect Agent

For instance, in the Ants model from chapter 1, the ants leave pheromone in the environment that changes the environment, and in turn changes the behavior of the ants. There are many different kinds of environments that are common in ABMs. In this section, we will discuss a few of the most common types of environments.

Implementation of an Environment

Before we discuss the types of environments, it is important to mention that the environment itself can be implemented in a variety of ways. First, the environment can be composed of

agents such that each individual piece of the environment can have a full set of properties and actions.

NetLogo

In NetLogo, this is the default view of the environment — the environment is represented by the agentset of patches. This allows different parts of the environment to have different properties and act differently based on their local interactions.

TOTAL: 1 to 130 Slides (Mid-Term)