Software Review: NetLogo, a Multi-agent Simulation Environment

Elizabeth Sklar
Department of Computer and Information Science
Brooklyn College
City University of New York
2900 Bedford Avenue
Brooklyn, NY 11210
sklar@sci.brooklyn.cuny.edu

1 Introduction

NetLogo [41, 40] is a freely downloadable, agent-based software package that was created at the Center for Connected Learning and Computer-Based Modeling (CCL) at Northwestern University, directed by Uri Wilensky. It is the latest in a series of multi-agent simulation environments that includes StarLogo [28, 33], which was developed at the MIT Media Lab by Wilensky and Mitchel Resnick. In these environments, agents are represented as "turtles" and are programmed with a variant of Seymour Papert's LOGO [8, 19]. Designed to be simple enough for children to program, NetLogo illustrates what can happen when populations of turtles are given sets of rules to obey. Despite the entry-level programming interface, NetLogo is capable of quite sophisticated modeling and allows experienced programmers to add their own Java extensions. As a result, NetLogo has been widely used by a broad audience, from elementary school children to academics in the social, computer, and "hard" sciences; and the online community page includes models constructed by a wide range of representatives from each of these segments of the population.

The NetLogo home page (http://ccl.northwestern.edu/netlogo/) includes a download area, models pages, sample downloadable extensions, user manuals, a FAQ, and links to various resources. The software package comes with a large number of demonstration models, each of which can be executed and modified, since the source for each demo is included in the package. These demos are also available on the "models" portion of the home page, where they can be run in a Web browser. They are contributed by community members and are tested and supported by the development team at CCL. Model application areas range from biology to computer science to art to artificial life and include simulations such as predator-prey, spread of disease, rumor mill, traffic jams, small worlds, fractals, erosion, dining philosophers, cellular automata, crystallization, and radioactivity. In addition, on the "community models" portion of the home page, an even larger and more diverse set of models is available. These, too, have been contributed by community members, but they are not checked by CCL staff; they are supported by their authors, whose contact details are listed on the Web page.

NetLogo was first released in 1999, and the most recent release (as of this writing) is version 3.1.2 (August 9, 2006). NetLogo runs on virtually any of today's most popular platforms—Microsoft Windows, Mac OS X, and Linux. For each version released, the Web page (URL) enumerates the operating systems on which NetLogo has been successfully installed and executed. NetLogo is written in Java, which aids its portability and compatibility. One of the most convenient features of NetLogo allows programmers to save models as Java applets, permitting seamless publication of simulations from NetLogo's built-in integrated development environment to a Web page.

NetLogo, Uri Wilensky, Learning Sciences & Computer Science, Center for Connected Learning and Computer Based Modeling, Northwestern University.

© 2007 Massachusetts Institute of Technology

2 Background

Agent-based modeling is an approach to simulation in which the system under investigation is represented as a set of agents [42]. The most fundamental properties of any agent are that it is *autonomous* and *self-interested*. The agent is equipped with a high-level set of goals; and every time it has a choice of action, it chooses the action that it believes is the one that best achieves one of its goals (or subgoals). This implies the agent has some internal set of rules guiding its decisionmaking and cannot be directly told what to do by another agent; if one agent wishes to change the behavior of another agent, it can only do so indirectly, by effecting some action that alters the best way for the second agent to achieve its own goals. Due to the autonomy of the individual agents, multi-agent systems provide a natural means of structuring simulations of systems where there are multiple loci of control.

Agent-based modeling and simulation contrasts with traditional approaches, which are typically built up from sets of interrelated differential equations. These *equation-based models* generate useful predictions about the behavior of populations, but agent-based models are a natural way to describe systems characterized by many levels of interactions, and agent-based models can capture emergent phenomena that equation-based models cannot [37]. Multi-agent systems are most commonly used for analyzing, designing, modeling, and simulating a diverse range of *complex systems*. Examples of phenomena studied using agent-based modeling include flocking [29], ant colony optimization [4, 7], and crowd flow [12, 14, 21, 24, 3, 11]. Another area in which agent-based modeling has made an impact is in the development and analysis of economic models [39, 22, 17, 18, 5, 34, 6, 38, 10, 15, 16].

NetLogo is an interesting contrast to other types of agent-based modeling environments, such as Swarm, RePast, and AgentSheets. Perhaps the most widely used of these is Swarm [36], developed at the Santa Fe Institute. Swarm has an active user group and an annual meeting (SwarmFest) where research using Swarm is discussed. Despite these advantages, several research groups have built their own tools because it is argued that Swarm is difficult to use. Repast [27], for example, is a Swarm-like tool developed at the University of Chicago for carrying out agent-based simulations in the social sciences (and is a tool we have been using in our work). Both Repast and Swarm require model developers to have a significant programming background, though Repast is friendlier to install and use than Swarm. Ascape [2], developed at the Brookings Institute, is an attempt to make it more straightforward for nontechnical developers to create models. The system does seem simpler than Swarm, and, according to [25], the Ascape code for the Sugarscape [35] model is more compact than in Swarm or Repast; however, as the Ascape manual makes clear, anyone intending to use Ascape "will need to have (or acquire) a working knowledge of Java programming." The Multi-Agent Modeling Language (MAML) [20] is another attempt to make modeling easier, by providing a series of macros that can quickly define the skeleton of a Swarm model (which then has to be fleshed out in detail using C code). Simplifying model construction is also the aim of the Strictly Declarative Modeling Language (SDML) [30], developed in SmallTalk at Manchester Metropolitan University (UK), which allows agent models to be specified as sets of declarative rules.

In addition to these research tools, AgentSheets [1] is a commercial package for building agent-based models. It is designed to allow non-programmers to create virtual worlds and other kinds of interactive content. It uses a rule-based language, which can be difficult for nontechnical users to utilize effectively. Furthermore, it does not provide the flexibility or extensibility of NetLogo.

MicroWorlds [23] is another commercial package created for non-programmers; the intended audience ranges from elementary school children as young as 4 years old ("MicroWorlds JR") to middle school students. MicroWorlds is also based on Logo and is the flagship product of Logo Computer Systems Inc. (LCSI), founded in 1981 by Seymour Papert. The MicroWorlds interface can be entry-level, where users click on elements to construct "worlds" of their own and can animate the inhabitants of their world by selecting from a small set of actions (like moving forward). More advanced users can write Logo programs to control the elements in the world. The biggest difference between MicroWorlds and NetLogo is that the intended (and actual) audiences are different; MicroWorlds is targeted at elementary school students, whereas NetLogo is being used by an older and much more sophisticated audience. NetLogo is free and written in Java, so it is overall a more extensible and flexible environment.

3 The Netlogo Interactive Environment

NetLogo employs a graphical user interface, which contains three tabs: Interface, Information, and Procedures. The initial tab, which is shown upon starting up NetLogo, is shown in Figure 1; this is the Interface tab. The user clicks on the buttons at the top of the Interface tab to go to the other tabs, each of which is described in this section.

3.1 Interface

The interface tab is essentially a visual editor in which the programmer can create and edit graphical elements: button (invokes a *procedure*, as described in Section 3.3), slider (sets the value of a numeric variable, within a range specified by the programmer when creating the slider), switch (sets the value of a binary variable to either *on* or *off*), chooser (sets the value of a string variable from a list of choices designated by the programmer), monitor (displays the value of a variable, updated as the simulation runs), plot (displays plots, updated as the simulation runs), output (displays dynamic textual output, updated as the simulation runs, and scrolls as needed), and text (displays a static label, set by the programmer). The content of each of these elements is controlled by global variables and methods which are specified in the Procedures tab.

By default, there is a two-dimensional canvas, called the *world*, in which agents, or turtles, are illustrated. The programmer can change the size of this canvas, which by default is 35 × 35 units, or *patches*, in size. The size of each patch is also settable by the programmer; the default size is 12 pixels. The origin is by default at the center of the world, so that the edge coordinates (in patches) extend from (−17, 17) in the upper left corner to (17, −17) in the lower right corner; however, the placement of the origin can also be set by the programmer (other choices are corner, edge, and custom). One of the newest features of NetLogo is the ability to view the world in three dimensions, although only two-dimensional (x, y) control is allowed (i.e., there is no z coordinate).

The basic unit of simulation in NetLogo is a *turtle*, that is, an agent. Each turtle resides on a patch; multiple turtles can be located on a single patch. All turtle movements are computed in terms of patches, so, for example, the command `forward 4` moves the turtle forward (in a direction based on its heading) 4 patches, over 4 time steps in the simulation.

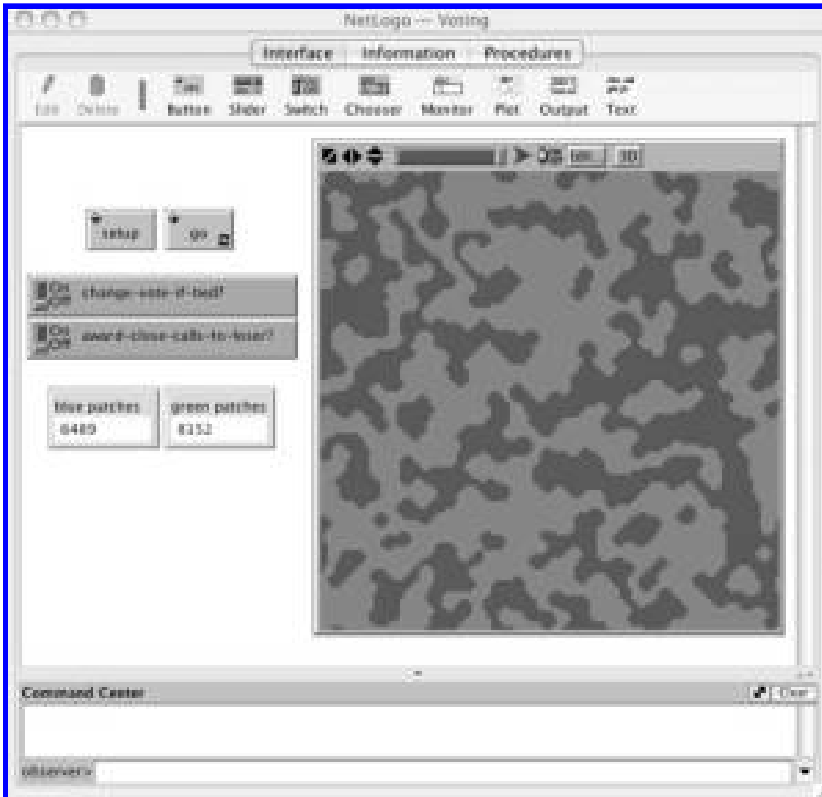
Typically, the NetLogo programmer instantiates a population of turtles and writes a series of *procedures* (i.e., methods) that control the behavior of the turtles. One of the primary assumptions is that the turtles will represent physical entities whose behaviors result in movements around the two-dimensional world. However, this is not a requirement for all simulations; and some of the models in the demo set and posted on the community page offer other ways to use turtles, though discussion of these types of models is beyond the scope of this review.

Programmers set the shape, size, and color for each turtle. The shape can be selected from a pre-defined set of nearly 40 shapes, or a new one can be designed using the handy Shapes Editor tool. This is a pixel editor that allows definition of shapes that fit within a circular region with a diameter of 20 pixels and use up to 16 colors. The tool shows multiple views of the shape as it is edited, giving the programmer an idea of how the new shape will look when it is drawn in various sizes in the NetLogo world.

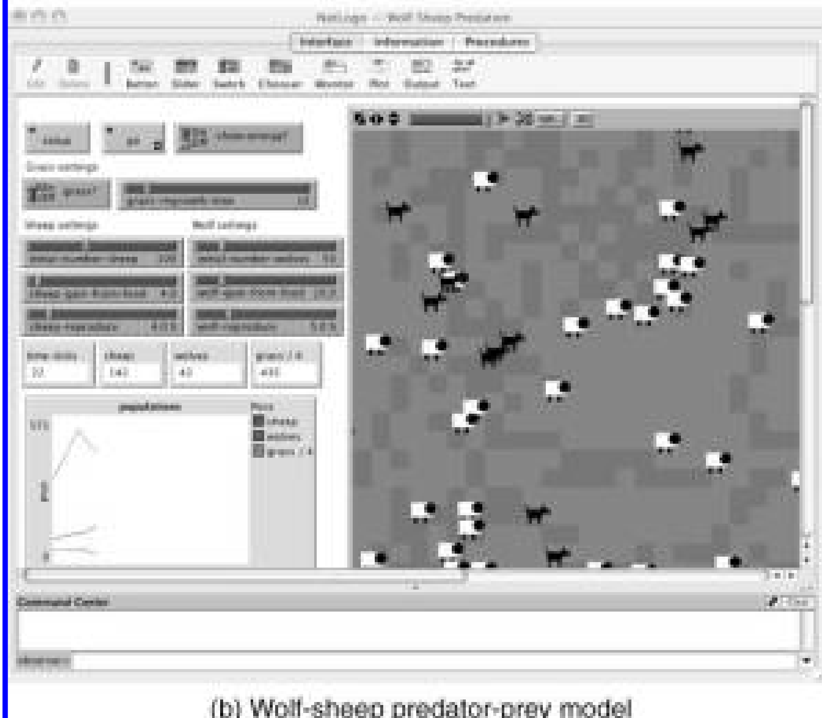
3.2 Information

The information tab is designed to enable and encourage documentation of the NetLogo model. The programmer can click on an Edit button, and then the content of the information tab is displayed in a plain text editor. There is one "smart" feature of this editor. Section titles can be underlined, using plain text dashes or hyphens, and when the information tab is displayed (outside of the edit mode), these titles are shown with a blue background and separated from the text that follows by a blank line.

By default, the information tab is initialized with the following sections: What is it?, How it works, How to use it, Things to notice, Things to try, Extending the model, NetLogo features, Related models, and Credits and References. Following each section heading, there is a one-sentence description of the intended



(a) Voting model



(b) Wolf-sheep predator-prey model

content. For example, under Extending the model, the description says: "This section could give some ideas of things to add or change in the procedures tab to make the model more complicated, detailed, accurate, etc."

3.3 Procedures

The Procedures tab enables the programmer to make use of an extensive language of *primitives* that control the behavior of NetLogo interface elements and turtles. The primitives can be grouped according to functionality. Some operate on data structures (*Turtle*, *Agentset*, *List*, and *String*). Some control basic program flow and functionality. Others provide access to the interface components (e.g., *Patch* and *World*). Input from and output to a range of file types is handled by primitives. A new, experimental set of primitives, *Links*, offer network display capabilities. All of the categories of primitives are described below.

3.3.1 Data Structures

The primitives in the *Turtle* category control movement of the agents within the two-dimensional world as well as the appearance (color, shape, and size) of turtles. They also include methods for constructing and destroying turtles and for inquiring about turtle properties.

Typically, a NetLogo simulation operates on one or more populations of agents, each of which is called an *agentset*. The primitives in the *Agentset* category allow programmers to perform atomic operations on sets of turtles, such as sorting, grabbing a set of values, and computing statistics on a set of values.

NetLogo allows a programmer to define and instantiate her own *breed*, in addition to using the generic turtle agent. For example, if a market simulation is modeling behavior of buyers and sellers in an auction, then the programmer can define:

```
breed [buyers sellers]
```

in order to distinguish between the two types of agents. Then, the programmer can declare properties for each:

```
buyers-own [purchase-price]
sellers-own [sale-price]
```

Both the *Turtle* and *Agentset* primitives allow operations on breeds. The latest versions of NetLogo (after version 3.0) have provided more flexibility with breeds than earlier releases, though all turtle-specific primitives cannot be applied seamlessly to breeds. All turtles, when instantiated, are assigned a unique ID number. A breed is essentially a relabeling of a turtle agent, so when breeds are instantiated, they are assigned ID numbers out of the same pool as the turtles. The primitive

```
turtle <id-number>
```

Figure 1. NetLogo interface. Two demo models are shown. The *voting* model (a) demonstrates how voters influence each other. Each square in the canvas on the right side of the window represents a voter. Voters can select either the *blue* or the *green* party; the color of each square indicates their choice. The choices are initialized randomly, about half for each party. When the simulation runs, each agent interacts with its neighbors (the eight surrounding squares). Agents can change their own votes depending on their neighbors' choices. Users can modify two parameters that affect voting behaviors and can run simulations to experiment with the results obtained from various combinations of parameter settings. The *wolf-sheep predation* model (b) demonstrates a predator-prey scenario in which wolves eat sheep and sheep eat grass. All expend energy to "live." In addition, sheep and wolves reproduce, and grass grows. When the grass is gone, the sheep die out; when the sheep are gone, the wolves die out. As with the voting model, users can set a number of parameters and experiment with various settings, observing phenomena such as slow grass regrowth leading to all grass being eaten, which leads to more rapid decline in the sheep population.

allows the programmer to specify a particular turtle, just like looking up an element in an array. However, if the turtle was created as a member of a breed, the breed name cannot be used here; the turtle command name must be used. This can be confusing to novice programmers. Yet, once understood, the issue is not critical and the intended functionality works as expected.

The primitives in the *Patch* category control various patch properties (e.g., color) and report instances of turtles located on patches. Aggregate properties of sets of four or eight neighboring patches can be calculated and returned easily.

The primitives in the *List* category operate on NetLogo's list data structure. This is a Lisp-like list structure, which stores a variable number of items of the same data type; lists of lists are allowed. The built-in list primitives provide functionality to return values from a list such as first, last, all but the first, all but the last, a specific item, a randomly chosen item, or a sublist. They also provide sorting, reversing, shuffling and filtering, as well as item substitution and removal. The *String* category offers primitives that operate on NetLogo's string data structure. These include comparison and concatenation operators, and functions that return substrings, remove characters from a string, and calculate the length of a string.

3.3.2 Program Control Flow and Basic Functionality

The *Control/Logic* category contains primitives that enable standard programming structures, such as looping (both counter- and condition-controlled) and branching (if-else statements), and that provide Boolean logic functions (and, or, not, and xor). The *Mathematical* primitives include all the standard arithmetic and comparison operators, plus trigonometric functions (sin, arcsin, etc.), statistical functions (mean, standard deviation, variance, etc.), and some useful number functions, such as floor, ceiling, and round. There are also a useful set of random number generators that produce numbers from either normal, Poisson, exponential, or gamma distributions. The programmer can set the random number seed.

3.3.3 Display Canvas

The *World* category contains primitives that pertain to the *world* display canvas (mentioned earlier), for example, for getting the size of the canvas in (x, y) coordinates and for clearing variables, patches, and turtles. The *Perspective* category contains primitives that allow the user to observe individual turtles, either by putting a *spotlight* on the turtle in the display, or by setting the center of the display to the turtle's location, shifting the display window as the turtle moves. The *Color* category contains primitives that control the colors of turtles (of any shape) and patches. Colors can be expressed in both RGB (red, green, and blue) and HSB (hue, saturation, and brightness) modes. Colors can be scaled at runtime in order to specify proportional colors within a range. For example, a population of 100 turtles can each be given a slightly lighter or darker shade of blue by specifying that each is assigned a color value n% of blue, where $0 < n < 100$. The *Plotting* category of primitives controls the display of the *plot* interface elements. Plots can be *x-y* line plots, scatterplots, bar charts, or histograms. The plotting functions include primitives to associate data sets with series being plotted, set axis limits, set plot type, and export a plot to an external file (in PNG format). The axes of a plot can be updated automatically, increasing as necessary when a simulation runs in order to keep data being plotted from overflowing outside the bounds of the plot.

3.3.4 Input and Output

The *Input/Output* and *Files* categories provide primitives to interface with data files and system variables such as the date and time. These include user interface primitives that return the (x, y) coordinates of the mouse in the *world* display canvas, or pop up file selection windows for the user to choose a file to open or for saving results, or write to the output portion of the interface tab. The file-related primitives allow standard functionalities, such as opening and closing files, reading from and writing to text files and checking for the end of file. The *Movie* category provides a set of

primitives that permit the capture of a simulation in a QuickTime file. The entire interface or just the display canvas can be captured.

3.3.5 Links

Links are a new, experimental part of NetLogo provided with version 3.1 (April 2006). This is essentially a data structure that connects two turtles together. The idea is to provide a means for drawing networks inherent within NetLogo. The prior lack of the ability to draw network structures easily was one of NetLogo's shortcomings, so this is a welcome new feature. Although it is experimental, it appears quite promising.

3.4 HubNet

One of the features of NetLogo that has been used in classrooms is the *HubNet* capability. This technology allows a group of users to interface with NetLogo, each controlling their own turtle through a remote device, such as a computer or a hand-held programmable calculator. This type of activity is called *participatory simulation*. Detailed discussion of this technique is beyond the scope of this article; interested readers are directed to http://ccl.northwestern.edu/partsims.html.

HubNet uses a client-server architecture. The HubNet server runs NetLogo, and the clients connect to the server and interact using a NetLogo interface. The server can send messages to one or more clients; the clients can send messages to the server. The authoring of HubNet functionality is not as well documented as the rest of NetLogo's features; potential users are requested (in the NetLogo manual) to contact the NetLogo authors for help implementing HubNet facilities.

3.5 Extensions

One of the aspects that render NetLogo so useful across a wide spectrum of audiences and applications is its *extensions* feature. Since NetLogo is written in Java, it is possible for programmers to write their own Java classes to interface with the NetLogo elements. This feature has improved greatly with more recent releases (version 3.0 and later) and includes API[1] documentation in the standard javadoc [13] format so that it is friendly to access and easy to read. The extensions portion of the NetLogo Web page contains sample extensions that can be downloaded, though users are cautioned that extensions are an "experimental" feature of NetLogo. Sample extensions include primitives for adding MIDI sound to models, for providing speech synthesis capabilities via the Mac OS X text-to-speech facility, for reading data via a live, Internet-based external feed and connecting to the GoGo board [9].

4 Summary

NetLogo is a well-written, easy-to-install, easy-to-use, easy-to-extend, and easy-to-publish-online environment. The entry level is simple enough and the tutorials provided in the package are straightforward and clear enough that anyone who can read and is comfortable using a keyboard and mouse could create their own models in a short time, with little or no additional instruction. The ceiling for university instructors and researchers is high—virtually unlimited—particularly for prototyping simulations of complex systems. The only downfall of the environment is that when it comes to running intense computations over many iterations, the system is simply too slow; for example, simulating thousands of generations of agents in an evolutionary computation experiment is better done elsewhere. However, any graphical, Java-based environment would be inappropriate for such a task. NetLogo's real strength in this area is the ease with which advanced programmers can rapidly prototype, debug, and visualize algorithms, saving time in the development cycle and providing a demonstration vehicle for presentations and posting on Web pages.

1 Application Programmer Interface.

This reviewer, who has used NetLogo for both research and teaching at several levels, highly recommends it for instructors from elementary school to graduate school and for researchers from a wide range of fields. Based on our experience, NetLogo has proven to be a very useful tool for prototyping simulations developed for research applications in the area of modeling real-world phenomena using multi-agent systems [31] and studying interaction mechanisms in multi-agent systems [32, 26]. It has also been used successfully to teach programming to inner-city high school students as part of a summer computing workshop and to undergraduates as the basis for an introductory computer science course for non-majors. Finally, it has been employed by doctoral students for rapidly constructing small demonstration projects as part of a graduate seminar in artificial life.

References

1. Agentsheets. http://www.agentsheets.com/ (accessed September 9, 2006).
2. Ascape. http://www.brook.edu/es/dynamics/models/ascape/default.htm (accessed September 9, 2006).
3. Borgers, A., & Timmermans, H. (1986). A model of pedestrian route choice and demand for retail facilities within inner-city shopping areas. *Geographical Analysis*, *18*, 115–128.
4. Botee, H. M., & Bonabeau, E. (1998). Evolving ant colony optimization. *Advances in Complex Systems*, *1*, 149–159.
5. Cheng, S.-F., Leung, E., Lochner, K. M., O'Malley, K., Reeves, D. M., Schvartzman, L. J., & Wellman, M. P. (2005). Walverine: A Walrasian trading agent. *Decision Support Systems*, *39*(2), 169–184.
6. Cliff, D., & Bruten, J. (1997). *Minimal-intelligence agents for bargaining behaviours in market-based environments* (Technical Report HP-97-91). Bristol, UK: Hewlett-Packard Research Laboratories.
7. Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*. Cambridge, MA: MIT Press.
8. Feurzeig, W., & Papert, S. (1968). Programming-languages as a conceptual framework for teaching mathematics. In *Proceedings of NATO Science Conference on Computers and Learning* (pp. 37–42).
9. GoGo Board. http://padthai.media.mit.edu:8080/cocoon/gogosite/home.xsp?lang=en (accessed September 9, 2006).
10. Greenwald, A., & Kephart, J. (1999). Shopbots and pricebots. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 506–511). San Francisco: Morgan Kaufmann.
11. Helbing, D., Farkas, I., & Viscek, T. (2000). Simulating dynamical features of escape panic. *Nature*, *407*, 487–490.
12. Helbing, D., Schweitzer, F., Keltsch, J., & Molnar, P. (1997). Active walker model for the formation of animal and trail systems. *Physical Review E*, *56*, 2527–2539.
13. Javadoc. http://java.sun.com/j2se/javadoc/ (accessed September 9, 2006).
14. Kayser, D. R., Aberle, L. K., Pochy, R. D., & Lam, L. (1992). Active walker models: Tracks and landscapes. *Physica A*, *191*, 17–24.
15. Kephart, J. (2002). Software agents and the route to the information economy. *Proceedings of the National Academy of Sciences*, *99*, 7202–7213.
16. Kephart, J., & Greenwald, A. (2002). Shopbot economics. *Autonomous Agents and Multi-Agent Systems*, *5*, 255–287.
17. LeBaron, B. (2000). Agent-based computational finance: Suggested readings and early research. *Journal of Economic Dynamics and Control*, *24*, 679–702.
18. LeBaron, B., Arthur, W. B., & Palmer, R. (1999). Time series properties of an artificial stock market. *Journal of Economic Dynamics and Control*, *23*, 1487–1516.
19. Logo. http://www.logofoundation.org/ (accessed September 9, 2006).
20. Marchal, D. (2002). Simulating pedestrian crowd behaviour in virtual cities (Technical report). Department of Computer Science, UCL, University of London.
21. Marks, R. E. (1992). Breeding hybrid strategies: Optimal strategies for oligopolists. *Journal of Evolutionary Economics*, *2*, 17–38.

22. Microworlds. http://www.microworlds.com/ (accessed September 9, 2006).

23. Multi-Agent Modeling Language. http://www.maml.hu/maml/initiative/index.html (accessed September 9, 2006).

24. Musse, S. R., Babski, C., Capin, T., & Thalmann, D. (1998). Crowd modelling in collaborative virtual environments. *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (pp. 115–123). New York: ACM Press.

25. Parker, M. T. (2000). *Ascape: Abstracting complexity* (Technical report). The Brookings Institution.

26. Reich, J., & Sklar, E. (2006). Robot-sensor networks for search and rescue. In *Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR).*

27. Repast. http://repast.sourceforge.net (accessed September 9, 2006).

28. Resnick, M. (1997). *Turtles, termites and traffic jams: Explorations in massively parallel microworlds.* Cambridge, MA: MIT Press.

29. Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, *21*, 25–34.

30. A strictly declarative modeling language. http://sdml.cfpm.org/ (accessed September 9, 2006).

31. Sklar, E., & Davies, M. (2005). Multiagent simulation of learning environments. In *Fourth International Conference on Autonomous Agents and Multi Agent Systems* (pp. 953–959). New York: ACM Press.

32. Sklar, E., Schut, M., Diwold, K., & Parsons, S. (2006). Exploring coordination properties within populations of distributed agents. In *AAAI Spring Symposium on Distributed Plan and Schedule Management*.

33. Starlogo. http://education.mit.edu/starlogo/ (accessed September 9, 2006).

34. Stone, P., & Greenwald, A. (2005). The first international trading agent competition: Autonomous bidding agents. *Electronic Commerce Research*, *5,* 229–265.

35. Sugarscape. http://www.brook.edu/es/dynamics/sugarscape/default.htm (accessed September 9, 2006).

36. Swarm. http://www.swarm.org/wiki/Swarm_main_page (accessed September 9, 2006).

37. Tang, Y., Parsons, S., & Sklar, E. (2006). Modeling human education data: From equation-based modeling to agent-based modeling. *Multi-agent Based Simulation (MABS) Workshop at Autonomous Agents and MultiAgent Systems.*

38. Tesauro, G., & Das, R. (2001). High-performance bidding agents for the continuous double auction. *Proceedings of the 3rd ACM Conference on Electronic Commerce* (pp. 206–209). New York: ACM Press.

39. Tesfatsion, L. (2002). Agent-based computational economics: Growing economies from the bottom up. *Artificial Life*, *8*, 55–82.

40. Wilensky, U. (1999). Netlogo. http://ccl.northwestern.edu/netlogo/ (accessed September 9, 2006).

41. Wilensky, U. (2002). Modeling nature's emergent patterns with multi-agent languages. *Proceedings of EuroLogo 2002*. Linz, Austria.

42. Wooldridge, M. (2002). *An Introduction to Multiagent Systems*. New York: Wiley.