NetLogo: A Simple Environment for Modeling Complexity

Seth Tisue
seth@tisue.net

Uri Wilensky
uri@northwestern.edu

Center for Connected Learning and Computer-Based Modeling
Northwestern University, Evanston, Illinois

Presented at the International Conference on Complex Systems, Boston, May 16–21, 2004

Abstract

NetLogo [Wilensky, 1999] is a multi-agent programming language and modeling environment for simulating complex phenomena. It is designed for both research and education and is used across a wide range of disciplines and education levels. In this paper we focus on NetLogo as a tool for research and for teaching at the undergraduate level and higher. We outline the principles behind our design and describe recent and planned enhancements.

1 Overview

NetLogo is a multi-agent programming language and modeling environment for simulating natural and social phenomena. It is particularly well suited for modeling complex systems evolving over time. Modelers can give instructions to hundreds or thousands of independent "agents" all operating concurrently. This makes it possible to explore connections between micro-level behaviors of individuals and macro-level patterns that emerge from their interactions.

NetLogo enables users to open simulations and "play" with them, exploring their behavior under various conditions. NetLogo is also an authoring environment that is simple enough to enable students and researchers to create their own models, even if they are not professional programmers.

We designed NetLogo for both education and research. There has been considerable research on the use of multi-agent modeling in K-12 settings (e.g. [Wilensky, 1995] [Resnick, 1996] [Wilensky & Resnick, 1999] [Wilensky, 2003] [Ionnidou et al., 2003] [Wilensky & Reisman, in press]). In this paper, however, we focus on NetLogo as a powerful research tool that is also suitable for learners at the undergraduate level and above.

Historically, NetLogo is the next generation of the series of multi-agent modeling languages including StarLogo [Resnick & Wilensky, 1993] [Resnick, 1994]. NetLogo is a standalone application written in Java so it can run on all major computing platforms. After five years of development, NetLogo is a mature product that is stable and fast. It is freeware—anyone can download it for free and build models without restriction. It comes with extensive documentation and tutorials and a large collection of sample models.

As a language, NetLogo is a member of the Lisp family that supports agents and concurrency. Mobile agents called "turtles" move over a grid of "patches," which are also programmable agents. All of the agents can interact with each other and perform multiple tasks concurrently.

In the following sections, we offer more detail on all of these topics. We begin with a guided tour of the application, then back up to outline its history, up to the most recent developments. We then give a more detailed account of the language itself and explain how we have implemented it. We conclude with a summary of work in progress and future plans.

2 Tour

Beginning NetLogo users typically start by exploring NetLogo's Models Library. This collection has more than 140 pre-built simulations that can be explored and modified. These simulations address many content areas in the natural and social sciences, including biology and medicine, physics and chemistry, mathematics and computer science, and economics and social psychology.

NetLogo is being used to build an endless variety of simulations. Members of our user community have turned turtles into molecules, wolves, buyers, sellers, bees, tribespeople, birds, worms, voters, passengers, metals, bacteria, cars, robots, neutrons, magnets, planets, shepherds, lovers, ants, muscles, networkers, and more. Patches have been made into trees, walls, terrain, waterways, housing, plant cells, cancer cells, farmland, sky, desks, fur, sand, you name it. Turtles and patches can be used to visualize and study mathematical abstractions, too, or to make art and play games. Themes addressed include cellular automata, genetic algorithms, positive and negative feedback, evolution and genetic drift, population dynamics, path-finding and optimization, networks, markets, chaos, self-organization, artificial societies and artificial life. The models all share our core themes of complex systems and emergence.

Figure 1 shows NetLogo's user interface after opening and running a model from the Models Library. Model controls are on the left. On the right is the graphics window, in which the "world" of the model is made visible. In the model shown, the turtles represent diffusing particles. They wander randomly. When the model begins, there is a single green patch in the center. When a particle encounters a green patch, it "sticks" and turns green itself. Over time a beautiful, branching aggregate emerges.

In this screen shot, we see only NetLogo's "Interface" tab. The other tabs contain documentation and the actual model code. The Interface tab is also an

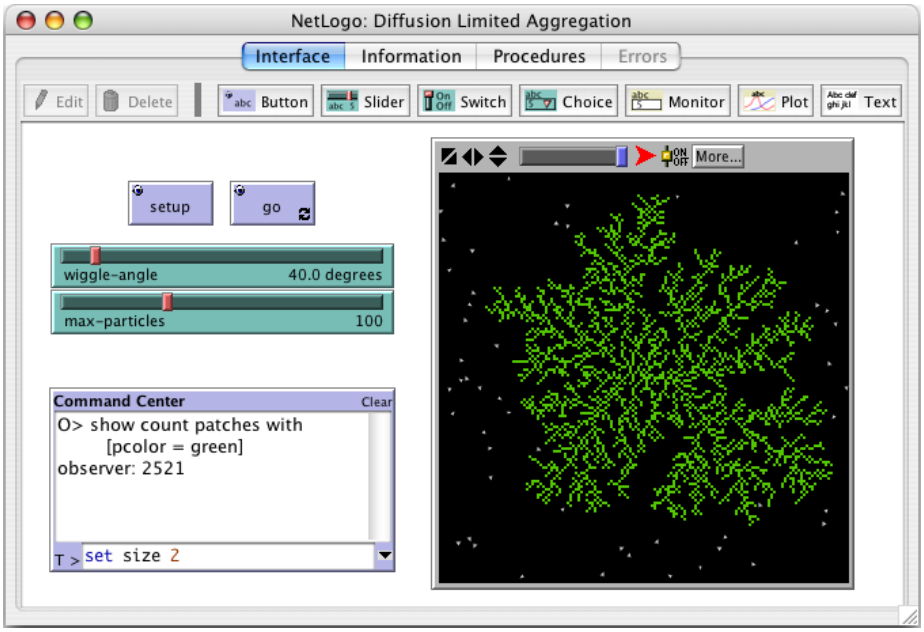

Figure 1: NetLogo's user interface, with model Diffusion Limited Aggregation

interface builder. No firm distinction is made between using a model and editing it—you can move, modify, or create interface elements at any time. Agents can be inspected and altered and the code for the model can be changed without restarting the simulation.

Figure 2 shows the Procedures tab and most of the code for the model. Language elements are automatically color-coded to enhance the visibility of the code's structure.

NetLogo can exchange data with other applications. The language includes commands that let you read or write any kind of text file. There are also facilities for exporting and importing data in standard formats. The complete state of the world can be saved and restored in a format that can easily be opened and analyzed with other software. Graphed data can be exported for rendering and analysis with other tools. The contents of the graphics window, or of the model's whole interface, can be saved as an image. A standard utility can be used to turn the images into a movie. Finished models can be published on the web as Java applets.

NetLogo includes a still evolving tool called BehaviorSpace that allows "parameter sweeping," that is, systematically testing the behavior of a model across a range of parameter settings.

The most visible area of change in NetLogo 2.0 was graphics. Now, turtles can be any size and shape and be positioned anywhere. Turtles and patches can also be labeled with text. Turtle shapes are vector-based to ensure smooth

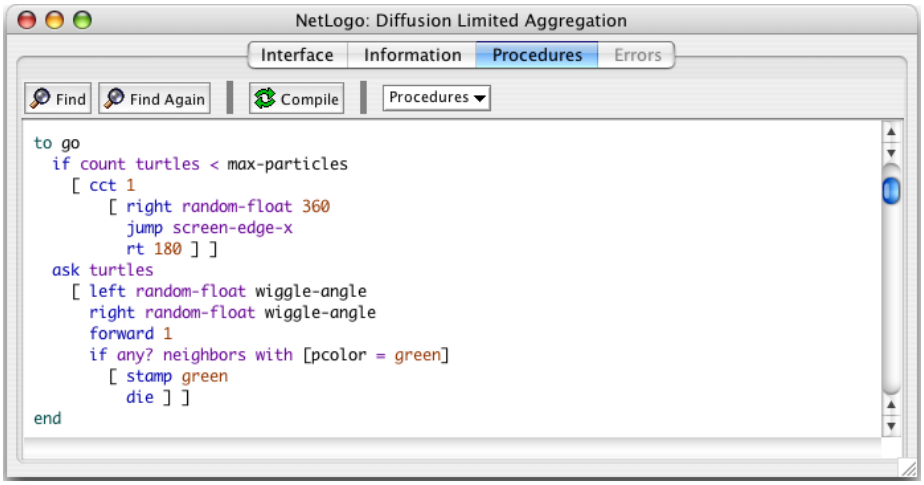

Figure 2: Main procedure of the aggregation model.

appearance at any scale. These changes have led to dramatic visual enhancement of models. An example of graphics that weren't possible before is the use of turtles to represent both nodes and edges in a network as in Figure 3.

3 History

NetLogo originates in a blend of StarLisp [Lasser & Omohundro, 1986] and Logo [Papert, 1980] (itself a member of the Lisp family). From Logo, it inherits the "turtle." In traditional Logo, the programmer controls a single turtle; a NetLogo model can have thousands of them. NetLogo also follows Logo's philosophy of ease of use, providing a "low threshold" of entry for new users. From StarLisp, a parallel Lisp of the 1980's, it inherits multiple agents and concurrency.

NetLogo's current design is based on our experience with our earlier environment, StarLogoT [Wilensky, 1997]. We redesigned both the language and the user interface. NetLogo includes almost all of StarLogoT's features and many new ones. Many of the new features of NetLogo are aimed at research users.

NetLogo has been under development since 1999. Version 2.0.1 (May 2003) is mature, stable, and reliable. Even though our user base has expanded, the rate of incoming bug reports has slowed to a trickle. Models now run much faster than in earlier versions—our users now find it fast enough for most purposes.

We have much evidence that acceptance of NetLogo in the research and education communities is wide and growing. The software has been downloaded tens of thousands of times. Currently, there are about 50 downloads per day. Our announcements list has over 5,000 members. The NetLogo discussion group (http://groups.yahoo.com/group/netlogo-users/) has over 1,500 members and averages over 100 posts per month. Traffic on the discussion group increased

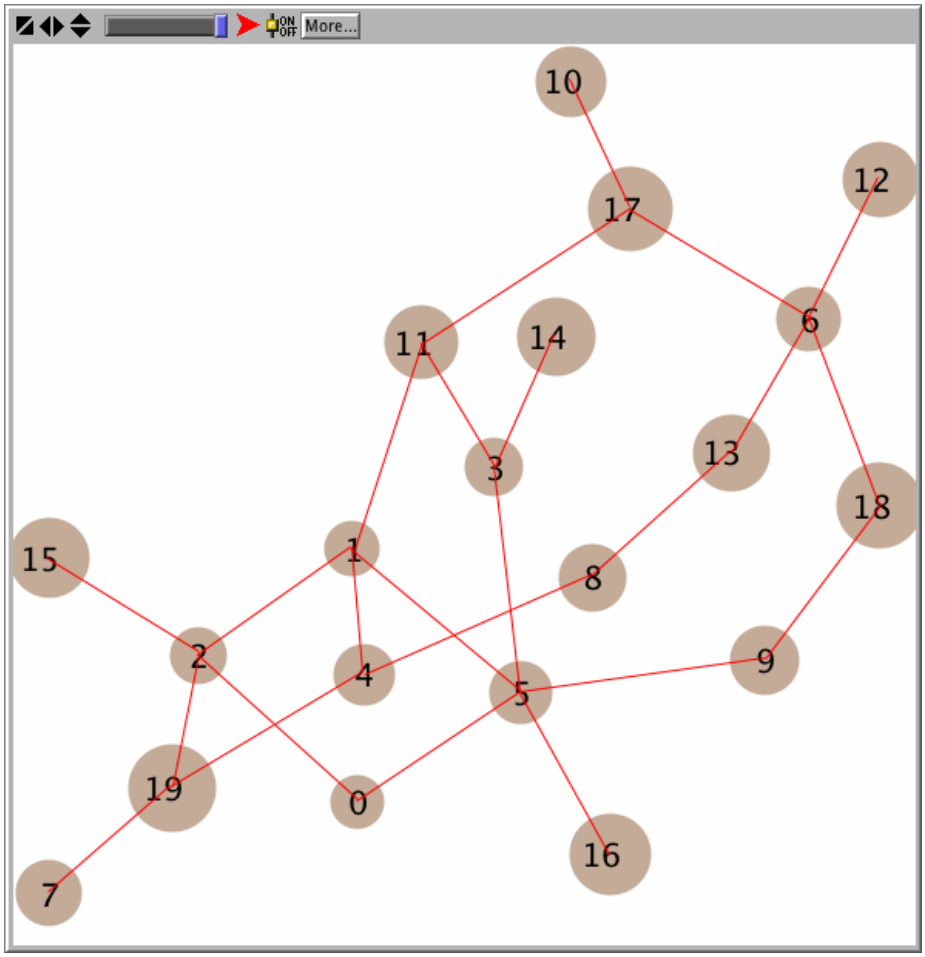

Figure 3: Nodes and edges, both represented using turtles

fivefold from 2002 to 2003. Several organizations have conducted workshops on NetLogo for researchers and teachers. Beginning this year, we will be holding our own annual workshop at Northwestern. A number of university classes are now taught, in whole or in part, using NetLogo. Some of these classes and workshops have rich collections of associated materials available online. The NetLogo web site has an area where users can upload models to share with the user community. More than 80 models have been uploaded so far.

4 Language

As a language, NetLogo adds agents and concurrency to Logo. Although Logo isn't limited to graphical applications, it is best known for its "turtle graphics," in which a virtual being or "turtle" moves around the screen drawing figures by leaving a trail behind it. NetLogo generalizes this concept to support hundreds or thousands of turtles all moving around and interacting. The world in which the turtles move is a grid of "patches," which are also programmable. Collectively, the turtles and patches are called "agents". All agents can interact with each other and perform multiple tasks concurrently. NetLogo also includes a third agent type, the "observer". There is only one observer. In most models, the observer gets the ball rolling by issuing instructions to the turtles and patches. Different "breeds" of turtle may be defined, and different variables and behaviors can be associated with each breed.

Some models use the patch world just as a lattice. For example, in a cellular automaton, there are no turtles, only patches. And in some other models, turtles move on the lattice (from patch center to patch center). But the patches are not just lattice sites—they are square sections of a continuous two-dimensional space. Turtle coordinates are floating point values, so a turtle may be positioned anywhere within a patch. For example, in the aggregation model shown above, the aggregate is made up of lattice sites, but particles move freely on the plane.

There are many language elements for talking about space and spatial relations: `towards`, `distance`, `neighbors`, `forward` and `back`, `left` and `right`, `size`, `heading`, `patch-ahead`, `diffuse`, and so on. Some of these come from Logo—others are new.

An important NetLogo language feature, not found in its predecessors, is "agentsets," or collections of agents. For example, the set of all turtles and the set of all patches are agentsets. You can also make custom agentsets on the fly, for example the set of all red turtles, or a column of patches (the set of patches with a given X coordinate). Agentsets are responsible for much of NetLogo's expressive power.

One of our core design goals for NetLogo is that results be scientifically reproducible, so it is important that models operate deterministically. NetLogo is a "simulated parallel" environment. In true parallel computing, programs must be constructed very carefully to avoid nondeterminism. We think this is too great a burden for novice programmers, so concurrency in NetLogo operates deterministically. That means that if you "seed" the random number generator

the same way, then a NetLogo model always follows the same steps in the same order and produces the exact same results, regardless of what computer you run it on. Java's underlying platform-independent math libraries help assure consistency.

In addition to special constructs to support multi-agent modeling, NetLogo also includes standard programming constructs such as procedures, loops, conditionals, recursion, strings, lists, and so forth. Both integer arithmetic and double-precision IEEE floating point arithmetic are supported. The `run` and `runresult` commands can be used to execute code constructed on the fly. However, some advanced features present in many Lisp-like languages are missing for now, such as symbols, user-definable control structures, and true functional values.

For further information on the NetLogo language, consult the NetLogo User Manual [Wilensky, 1999], particularly the Programming Guide and Primitives Dictionary sections.

5 Implementation

NetLogo is written in Java, version 1.4. Java was chosen because both the core language and the GUI libraries are cross-platform, and because modern Java virtual machines have relatively high performance. NetLogo 1.3 supported earlier Java versions going back to Java 1.1, but for NetLogo 2.0 we decided to require Java 1.4. Regrettably, switching to Java 1.4 meant dropping support for users of Windows 95 and MacOS 8 and 9, since no Java 1.4 implementation is available for those operating systems. However, we continue to offer support and bugfixes for NetLogo 1.3, so those users aren't left out in the cold.

NetLogo is a hybrid compiler/interpreter. To improve performance, we don't interpret the user's code directly. Instead, our compiler annotates and restructures it into a form that can be interpreted more efficiently.

At one time, NetLogo was a closed platform. Users couldn't alter or extend it, or control it from external code. This is now changing—NetLogo is becoming extensible. It has always been a full-fledged programming language, so users may write procedures in NetLogo and then use them just like built-in commands. But now in NetLogo 2.0.1, we have an application programmer's interface (API) for extensions so that users can add new elements to the language by implementing them directly in Java. For example, you might let agents make sounds and music using Java's MIDI capabilities, or communicate with remote computers, and many other things.

This section on implementation has been necessarily brief. We are publishing a separate paper [Tisue & Wilensky, in press] that addresses design and implementation issues in greater depth.

6 Conclusion

We have already touched upon some goals for future NetLogo versions, such as increased speed and greater extensibility. The following are enhancements on which we are already working and have made significant progress:

- 3-D NetLogo, including language extensions and 3-D graphics.
- Support for different lattices and world topologies.
- Usability improvements to the code and shapes editors.
- Easier, more flexible randomized agent scheduling.

We are also working on integration with aggregate modeling engines, improved support for network models, and integration with network analysis and visualization tools.

There are ongoing efforts within our research group to further explore NetLogo's potential for research and education. Of particular relevance to NetLogo's future as a research tool are three major ongoing projects:

- Integrated Simulation and Modeling Environments (ISME), a three-year project in collaboration with the University of Texas which uses NetLogo to enact "participatory simulations" [Wilensky & Stroup, 1999] in both classroom and research contexts.
- Procedural Modeling of Cities, a new three-year project in which agents "grow" virtual cityscapes for use in architecture, urban planning, and entertainment.
- Modeling School Reform, a new project to build models of the potential effects of educational policy decisions, to assist school leaders and policy makers.

These projects will drive substantial expansion of NetLogo's ability to support large, ambitious, multi-leveled models.

7 Acknowledgments

Portions of this paper were loosely adapted from the NetLogo User Manual [Wilensky, 1999]. We thank Ben Shargel for the name and first implementation of BehaviorSpace, and Owen Densmore for contributing the network layout model used to produce Figure 3. We gratefully acknowledge the support of the National Science Foundation.

References

[Ionnidou et al., 2003] Ioannidou, Andri; Alexander Repenning; Clayton Lewis; Gina Cherry; and Cyndi Rader, "Making Constructionism Work in the Classroom", International Journal of Computers for Mathematical Learning, Volume 8, Issue 1, pp. 63-108.

[Lasser & Omohundro, 1986] Lasser, Clifford; Omohundro, Stephen M.; *The Essential Star-lisp Manual*, Thinking Machines Corporation.

[Papert, 1980] Papert, Seymour, *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books.

[Resnick, 1994] Resnick, Mitchel, *Turtles, Termites and Traffic Jams: Explorations in Massively Parallel Microworlds*, MIT Press.

[Resnick, 1996] Resnick, Mitchel, "Beyond the Centralized Mindset", Journal of the Learning Sciences 5(1): 1-22.

[Resnick & Wilensky, 1993] Resnick, Mitchel; and Uri Wilensky, "Beyond the Deterministic, Centralized Mindsets: New Thinking for New Sciences", American Educational Research Association.

[Tisue & Wilensky, in press] Tisue, Seth; and Uri Wilensky, "NetLogo: Design and Implementation of a Multi-Agent Modeling Language", SwarmFest 2004.

[Wilensky, 1995] Wilensky, Uri, "Paradox, Programming and Learning Probability", Journal of Mathematical Behavior Vol. 14, No. 2, p 231-280.

[Wilensky, 1997] Wilensky, Uri, StarLogoT, Center for Connected Learning and Computer-Based Modeling, Northwestern University. http://ccl.northwestern.edu/cm/starlogot/

[Wilensky, 1999] Wilensky, Uri, NetLogo (and NetLogo User Manual), Center for Connected Learning and Computer-Based Modeling, Northwestern University. http://ccl.northwestern.edu/netlogo/

[Wilensky, 2003] Wilensky, Uri, "Statistical Mechanics for Secondary School: The GasLab Multi-Agent Modeling Toolkit", International Journal of Computers for Mathematical Learning, Volume 8, Issue 1.

[Wilensky & Reisman, in press] Wilensky, Uri; and Kenneth Reisman, "Thinking Like a Wolf, a Sheep or a Firefly: Learning Biology through Constructing and Testing Computational Theories", Cognition & Instruction.

[Wilensky & Resnick, 1999] Wilensky, Uri; and Mitchel Resnick, "Thinking in Levels: A Dynamic Systems Approach to Making Sense of the World", Journal of Science Education and Technology, vol. 8, no. 1.

[Wilensky & Stroup, 1999] WILENSKY, Uri; and STROUP, Walter; "Learning through Participatory Simulations: Network-based Design for Systems Learning in Classrooms", Proceedings of the Computer Supported Collaborative Learning Conference, Stanford University.