TURTLE HISTORIES AND ALTERNATE UNIVERSES: EXPLORATORY MODELING WITH NETLOGO AND MATHEMATICA

E. BAKSHY*, Northwestern University, Evanston, IL & University of Michigan, Ann Arbor, MI
U. WILENSKY, Northwestern University, Evanston, IL

ABSTRACT

This paper presents the design of a development platform integrating NetLogo, a multi-agent programmable modeling environment with the *Mathematica* scientific computing environment. We will discuss the affordances of such environments, which can simplify and enrich the research process for agent-based modelers. More specifically, we will demonstrate the advantages of having real-time exchange of complex data structures between agent-based modeling environments and symbolic mathematical software such as *Mathematica*. Together, such tools can provide researchers with a highly interactive, self-documenting workflow that neither tool can provide alone. This paper will give an overview of how the integrated environment can be used for common tasks in agent-based modeling, the construction of interfaces for exploring simulation dynamics, and the effective design patterns for representing simulation results.

Keywords: Agent-based modeling, exploratory analysis, NetLogo, *Mathematica*

INTRODUCTION

The behavioral dynamics of agent-based models contain vast quantities of information for which analysis can often be daunting to researchers. Nevertheless, for the purposes of model verification, validation and replication, it is essential for researchers to carefully and extensively study their models and analyze the behavior at several different levels (Wilensky & Rand, 2007). This paper presents a framework for representing, measuring, and visualizing the behavior of agent-based models. We will discuss some limitations of software systems used in the development and analysis of agent-based models, and demonstrate the ways in which our framework attempts to address these issues. We propose that many of these tasks can be resolved through the integration of agent-based modeling environments and scientific computing environments, such as NetLogo (Wilensky, 1999) and *Mathematica* (Wolfram, 2003). Our approach is, in some respects, similar to that of Macal & Howe (2005), which provides an

*Corresponding author address: Eytan Bakshy, School of Information, University of Michigan, 1075 Beal Ave., Ann Arbor, MI 48109; e-mail: ebakshy@umich.edu

extensive engine-level integration between Repast (2007) and *Mathematica*. We will further elaborate upon the affordances of such environments in the context of current issues faced by agent-based modelers.

Mathematica is an interactive programming environment which can support many of the tasks common to agent-based modelers. These tasks include pre-processing and analysis of external data used to motivate or calibrate models, model prototyping, interactive model exploration, data collection, storage, analysis, and documentation among other tasks. In contrast to using several special purpose or compiled programming languages for each of these tasks, the integration of such tools with high-level agent-based modeling environments like NetLogo can bridge the gap between model development, inquiry, and analysis.

Mathematica essentially consists of two processes, the kernel and the front-end. The kernel stores and executes all program code and data, which are represented in a uniform fashion as *expressions*. The front-end allows users to manipulate, retrieve, and graphically represent expressions stored in the kernel. Users interact with a *notebook*, which can contain text, program code, data output, and graphics. Typically, users enter commands into the notebook, which are executed by the kernel, and its output is displayed below, giving a line-by-line documentation of a user's session. In *Mathematica 6* (2007), notebooks can contain dynamic elements, which can display the state of expressions in the kernel in realtime, as well as relay information from interface objects back to the kernel.

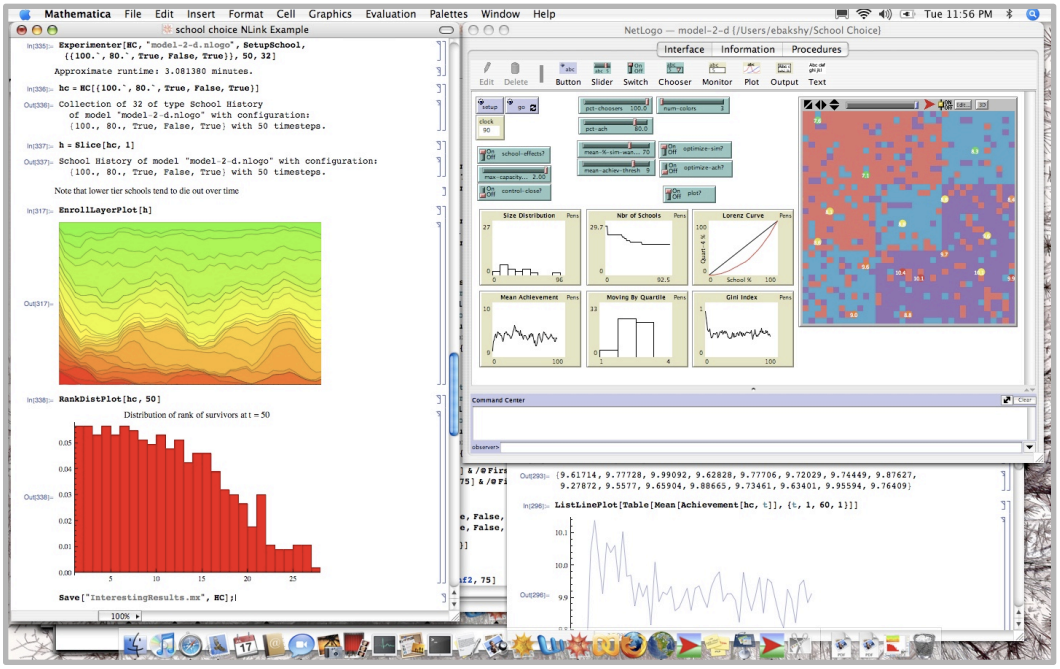

FIGURE 1 The NetLogo-Mathematica Modeling environment

The integrated NetLogo-Mathematica environment, depicted in Figure 1, includes many aspects that make it particularly well suited for conducting research with agent-based models. *Mathematica*'s data connectivity supports automatic format recognition and type conversion of files, as well as support for SQL database connectivity. In conjunction with pattern matching and rule-based programming functionality, such routines can reduce the amount of time spent preparing and organizing data for use with NetLogo. This integration makes accessible, for use with NetLogo models, *Mathematica*'s functions for statistics, non-linear optimization, linear algebra, graph theory, and a number of other functions suited for the execution and analysis of agent-based models. These methods can be combined with high-level graphical interface constructs to rapidly create custom tools for exploratory analysis of models. The environment's document-centered interface lets users combine comments, code, visualizations, and annotations in a single working notebook that can be viewed side by side with the NetLogo graphical interface. Finally, because all definitions, data, and graphics are serializable, the storage and retrieval of complex data structures representing model data (e.g., simulation histories) can be accomplished with minimal effort. These technical aspects of *Mathematica*, combined with the NetLogo-Mathematica interface, provide a flexible foundation upon which agent-based research frameworks can be built.

OVERVIEW OF THE NETLOGO-MATHEMATICA INTERFACE

The NetLogo-Mathematica toolkit provides a high-level interface to NetLogo from the *Mathematica* kernel via the J/Link Java interface. Once installed, one can load the package and launch NetLogo with no additional configuration.† At its core, the interface comprises of two simple functions: `NLCommand[]`, which executes a NetLogo command, and `NLReport[]`, which returns data from NetLogo. Other high-level primitives for repetitive tasks and acquiring structured interaction topologies, such as patches or grids (via `NLGetPatches[]`), and links or networks (via `NLGetGraph[]`) are included in the toolkit as well.

`NLCommand[]` is often used to programmatically initialize a model and execute the main loop. The function performs automatic type conversion, expression splicing, and concatenation, which allows users to easily access or modify NetLogo data using any combination of numerical, string, boolean, color, and list expressions. Additionally, numbers, strings, and lists are automatically converted back to native *Mathematica* types when requested from NetLogo using `NLReport[]`. For example, `NLCommand["set foo", {{True, 12, 8.4}, {False,`

---

† requires NetLogo 4.0 or greater and licensed version of *Mathematica* 6.0

`13, 8.9}}]` will set the NetLogo global variable, `foo`, to the NetLogo list expression, `[[true 12 8.4] [false 13 8.9]]`. Similarly, `NLReport["foo"]` will return a *Mathematica* expression containing the original nested list of boolean, integer, and floating-point types. These two basic functions are often sufficient to collect data and create plots on the fly that might otherwise require the use of file I/O or complex graphical interface programming.

We present the NetLogo-Mathematica environment in the context of several modeling tasks. First, we will discuss the validation in a suite of statistical mechanics models with traditional analytical descriptions. Second, we will show how the interactive visualization features of *Mathematica* 6 can be used to replay model dynamics in an agent-based model of cultural dissemination. Finally, we will present effective design patterns for representing simulation results, and show how they may be used to perform an exploratory analysis of the parameter space of a forest fire model.

COMPARING AGENT-BASED MODELS WITH ANALYTICAL MODELS

The NetLogo-Mathematica kit was first used to solve the following problem: are the agent interaction rules in GasLab suite of NetLogo models (Wilensky, 1997a) sufficient to generate velocity distributions found via traditional analytical treatments of ideal gases? In this example, the model's initial conditions are set using `NLCommand[]`. We define the function `Resample[]` to execute the NetLogo model for 50 "ticks," and return a list of speeds back to *Mathematica*

```
Resample[]:= Module[{},
 NLCommand["repeat 50 [go]"];
 NLReport["[speed] of particles"]
];
```

To collect a sufficient number of moments of velocities, the distributions are resampled forty times using the list constructor, `Table[Resample[],{40}]`, which will generate a list of 40 elements, each element being the a consecutive resampling of the simulation.

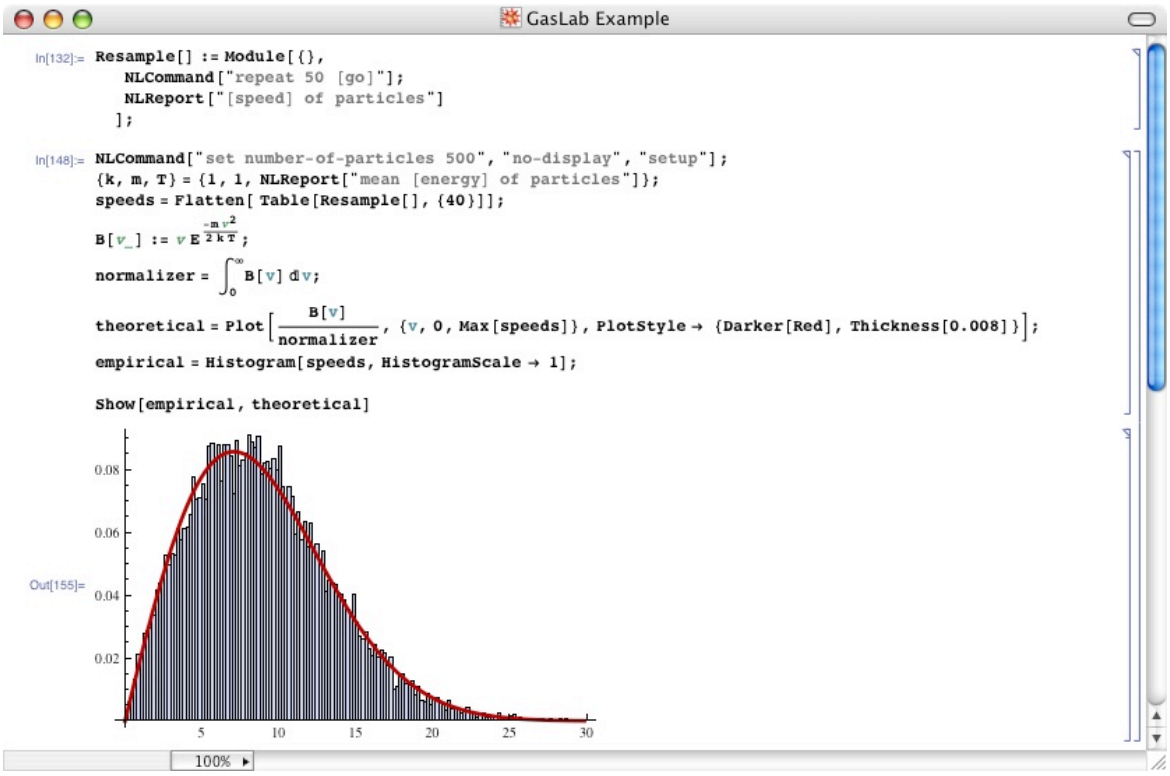

FIGURE 2 Comparing simulated results with analytical distributions

To compare the observed distribution with the analytic description of the speed distribution, we must find the mean energy of the system, which can be sampled directly from NetLogo: `NLReport["mean [energy] of particles"]`. With this data, we compare the observed distribution with the Maxwell-Boltzmann distribution. The two distributions are in close agreement with one another, as illustrated in Figure 2. This first example is a fairly straightforward example of data collection and visualization. We now turn to a more complex example involving multi-dimensional, time-varying data.

VISUALIZATION AND INTERFACE CONSTRUCTION

Programmable agent-based modeling environments like NetLogo allow developers to rapidly construct realtime visualizations of their model. However, an ABM environment typically supports a single modality of visual representation viewed forwards in time. This can make the analysis of complex systems, whose components evolve in a parallel fashion at multiple levels, quite difficult. The NetLogo-Mathematica environment provides a convenient

way to store the complete simulation "history" in memory, and rapidly prototype interactive, multi-modal visualizations for understanding this data.

The following example shows how `NLGetPatches[]`, the NetLogo-Mathematica function for retrieving patch-based data, can be combined with interactive visualization procedures in the analysis of patch-based models. The time-varying patch data will be represented in several ways over time using `Manipulate[]`. In this example, we use a modified version of a NetLogo implementation (Centola, 2007) of Axelrod's model of cultural dissemination (Axelrod, 1997). In NetLogo, each patch agent owns a variable called `culturalFeatures`, which is an array of *k* features. At every time step, we execute the model and retrieve the matrix of patches, with each entry of the matrix being a list of cultural features, using `NLGetPatches["culturalFeatures"]`.

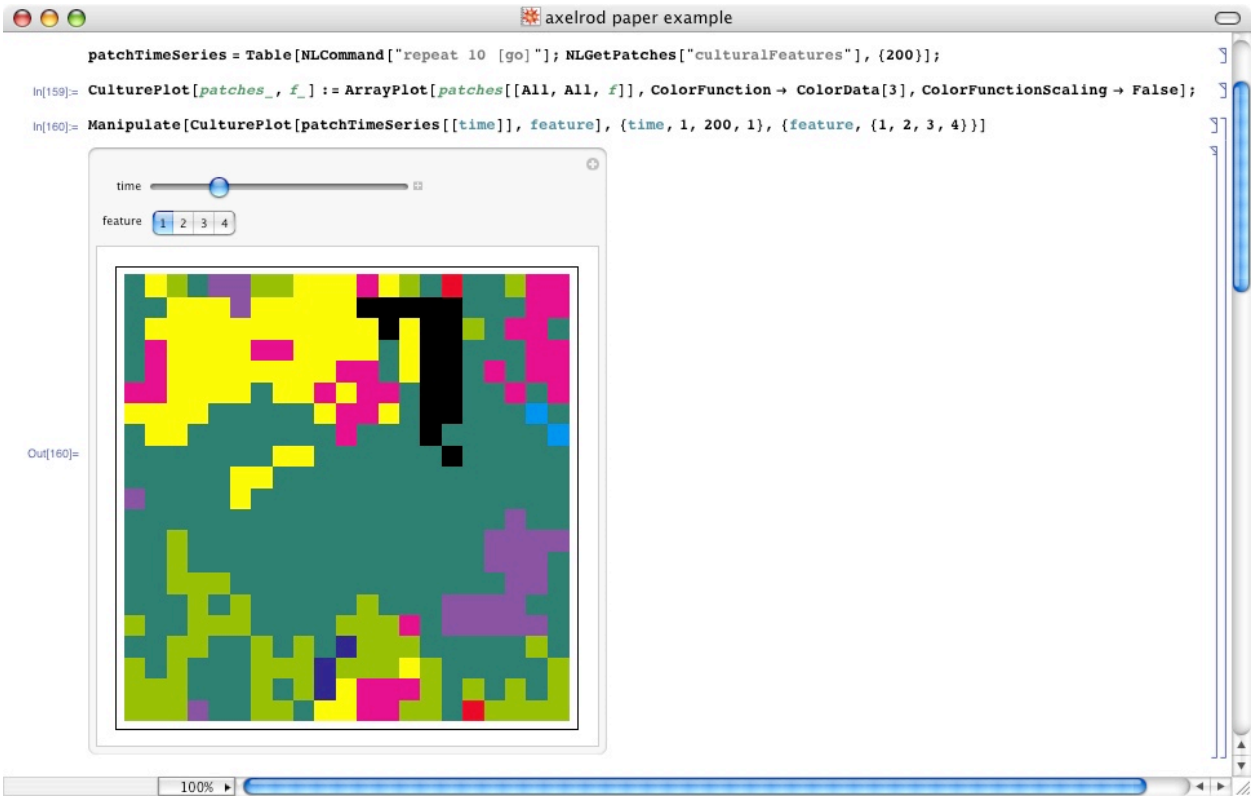

FIGURE 3 An interactive interface for exploring model dynamics

The interface in Figure 3 is generated by a single call to `Manipulate[]`:

```
Manipulate[CulturePlot[patchTimeSeries[[time]], feature],
 {time, 1, 200,1}, {feature, {1, 2, 3, 4}}]
```

The code specifies an interface which lets the user to view any of the four features at time steps 1 through 200, which may be animated both forwards and backwards. Dynamic visualization constructs such as `Manipulate[]` are just one example of a host of other high-level tools for constructing interfaces. These functions can also be used to display the progress of a parameter-space exploration in realtime, or compare aggregate "between realization" visualizations with individual simulation time-series. Such interfaces are particularly useful for collaborative analysis of models, since they allow a team of scientists, including those with less programming experience, to readily find patterns and test hypotheses in a model.

DESIGN PATTERNS FOR EXPLORATORY ANALYSIS OF AGENT-BASED MODELS

The traditional cycle of rigorous analysis of NetLogo models commonly involves writing software to specify a region of parameter space to explore in an automated fashion, either through the use of shell scripts, or specialized tools such as BehaviorSpace (Wilensky, 2003). Users must specify ahead of time the ranges and increments of parameters they would like to vary, and how many times each model run is repeated. Other structured data, such as lists or graphs can be difficult to format and read in by most tools for analysis, so most data written to disk is in the form of pre-aggregated scalar data. In this section we will propose a method for effectively executing and storing structured simulation data on a call-by-need basis. This approach is similar to memoization in dynamic programming, where "subproblems" (measures on a parameterization of a model) are stored in memory to speed up the execution of larger problems, such as finding critical points in a model's behavior or developing visualizations involving potentially thousands of runs.

A model realization can be thought of as a collection of data representing the execution of a deterministic program, or model. Agent-based models typically exhibit some degree of stochasticity. That is, the execution of the model involves psuedorandom processes, which may result in models that have identical initial conditions but produce a variety of possible outcomes. Thus, it is often important that there is a way to encode multiple realizations of the same parameterization of the model, but with different random seeds. With this method, any particular realization can be reproduced, given that the random seed is properly initialized and stored. These model realizations can be parameterized by their configuration settings and a labeling of their realization. A realization can be represented in *Mathematica* using some form, such as:

```
Realization[{p1, ..., pn}, repetitionNumber] = <model data>
```

In *Mathematica*, we will take advantage of the fact that the system can "remember" its value using the idiom:

```
f[x_] := f[x] = function to be evaluated at x
```

Each time the function `f[]` is evaluated at some *x*, its value is calculated and stored as part of the definition of the function. This is a convenient mechanism for storing the results of often computationally intensive realizations of models for later use. Below the typical structure of a NetLogo-Mathematica Realization object is specified:

```
Realization[{var1_, ..., varN_}, repetition_] :=
  Realization[{var1, ..., varN}, repetition] =
   Module[{intermediate data structures},
    (a) setup model using parameters;
    (b) execute model and store intermediate results;
    (c) return result structure
   ];
```

(a) NetLogo variables are initialized according to the model's parameters using NLCommand[]. Side effects of the initialization, such as NetLogo-generated random seeds, or initial placement of agents may be recorded in intermediate data structures
(b) The main NetLogo loop is executed, and agent variables or aggregate measures are recorded to intermediate data structures. At this point, we may process NetLogo data using *Mathematica* and insert new values into agents. This is typical for models in which agents utilize *Mathematica* functionality to carry out their rules.
(c) The intermediate structures are combined into a single expression representing the simulation.

In addition, users may define several functions that operate on the Realization data. These operators come in three common varieties:

- Directly accessing an element of the resultant expression, such as a time-series array of some measure, or the distribution of agents' variables
- Aggregating agent data, such as calculating the Gini index of the stored population or the clustering coefficient of a network
- Visualization, such as plotting the time dynamics or network structure of a realization

Together, these functions and structures can provide a flexible framework for dealing with modeling tasks ranging from exploratory analysis, sensitivity analysis (Miller, 1998), to validation and docking (Axtell et. al., 1996, Wilensky & Rand, 2007).

EXPLORATORY ANALYSIS WITH REALIZATION OBJECTS

Here we will consider an instance of this Realization object prototype. This function executes the NetLogo forest fire model (Wilensky, 1997b) with a particular density and reports back the fraction of trees burned.

```
PctBurned[density_, rep_] := PctBurned[density, rep] = Module[{},
    NLCommand["set density ", density, "setup"];
    NLCommand["while [any? turtles] [go]"];
    NLReport["(burned-trees / initial-trees)"]
    ];
```

We may attempt to find the phase transition by plotting the result of this function with the entire range of densities, from zero percent to one hundred percent in increments of ten:

```
ListPlot[Table[PctBurned[density,1],{density,0,100,10}]]
```

Finding that a phase transition occurs approximately between forty and eighty percent density, we can execute the model over this range in increments of five percent, and observe its variance over ten additional repetitions using a box and whisker plot:

```
BoxWhiskeryPlot[Transpose[Table[Table[PctBurned[density,rep],{rep,10}],
 {density,30,60,5}]]]
```

Finally, we can plot the transition averaged over twenty runs at a higher resolution in increments of one:

```
ListPlot[Table[Mean[Table[PctBurned[density,rep],{rep,20}]],
 {density,30,60,1}]]
```

CONCLUSION

The environment and techniques presented in this paper can provide researchers with a rich environment in which they can rigorously debug, analyze, and make inferences from agent-based models. It provides an integrated workflow which enables users to focus on experimentation rather than the implementation. We hope that the methods proposed here can be of use to the agent-based modeling community and promote a more intimate understanding of phenomena observed in our models and a more robust treatment of our results. We have used these tools in several of our research projects at the CCL. In the context of an NSF-funded research project on modeling educational policy, we have profitably applied the NetLogo-Mathematica interface tools to explore a large-scale model of school choice calibrated with empirical data. The integrated environment has enabled the iterative construction of the model, including the model calibration, analysis of runs, and even model documentation. In addition,

the environment has aided in the collaboration with members outside of our immediate research group by enabling us to rapidly examine new hypotheses and analyze the data in multiple ways. In this respect, the NetLogo-Mathematica integrated environment provides a powerful addition to the model builder's toolkit.

ACKNOWLEDGMENTS

The preparation of this paper was supported by the National Science Foundation DHB grant #0624318. We would like to thank Spiro Maroulis, Bill Rand, Seth Tisue, and Lada Adamic for their feedback on this work.

REFERENCES

Axelrod, R. The Dissemination of Culture: A Model with Local Convergence and Global Polarization The Journal of Conflict Resolution, Vol. 41, No. 2 (Apr., 1997), pp. 203-226

Axtell, R., Axelrod, R, Epstein, J. M, & Cohen, M. D. (1996), "Aligning Simulation Models: A Case of Study and Results", Computational Mathematical Organization Theory, 1(2), pp. 123-141

Macal, C.M. & Howe, T.R. (2005). Linking Repast and Computational Mathematics Systems: Mathematica and MATLAB. Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms, Gleacher Center. Chicago, IL, USA. 13-15 October 2005. Argonne National Laboratory and the University of Chicago. Pp. 5-23.

Miller, J. H. (1998) Active Nonlinear Tests (ANTs) of Complex Simulation Models Management Science, Vol. 44, No. 6 pp. 820-830

Tisue, S., & Wilensky, U. (2004). NetLogo: Design and implementation of a multi-agent modeling environment. Paper presented at the Agent 2004 conference, Chicago, IL, October 2004.

Wilensky, U. (2001). Modeling nature's emergent patterns with multi-agent languages. Paper presented at the Eurologo 2001 conference, Linz, Austria.

Wilensky, U. & Rand, W. (2007). Making Models Match: Replicating an Agent-Based Model" Journal of Artificial Societies and Social Simulation, 2007, 10(4). http://jasss.soc.surrey.ac.uk/10/4/2.html .

Wolfram, S., (2003). The Mathematica Book: 5th Edition, Wolfram Media, Champaign, IL.

Models and Software

Centola, D. (2007). Axelrod Culture Model. http://hsd.soc.cornell.edu/curricular/Axelrod/AxelrodModel.html. Networks and Social Dynamics Group, Cornell University, Ithaca, NY

Wilensky, U. (1997a). NetLogo GasLab Gasinabox model.http://ccl.northwestern.edu/netlogo/models/GasLabgasinabox. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Wilensky, U. (1997b). NetLogo Fire model. http://ccl.northwestern.edu/netlogo/models/Fire. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.
Wilensky, U. (1999). NetLogo [Computer software] (Version 4.0). Evanston, IL: Center for Connected Learning and Computer-Based Modeling. http://ccl.northwestern.edu/netlogo.

Repast, 2007, [Computer software] Repast home page; available at http://repast.sourceforge.net/
Mathematica, 2007, [Computer software] Mathematica documentation (Version 6.0); Champaign, IL: Wolfram Research Inc. http://reference.wolfram.com/mathematica/.

Wilensky, U. (1999a). NetLogo [Computer software] (Version 4.0). Evanston, IL: Center for Connected Learning and Computer-Based Modeling. http://ccl.northwestern.edu/netlogo.
Wilensky, U. (2003). Behavior Space [Computer Software] (Version 3.0). Evanston, IL: Center for Connected Learning and Computer Based Modeling, Northwestern University. http://ccl.northwestern.edu/netlogo/behaviorspace.