New Technologies in Simulation Games

Kenneth L. Simons
System Dynamics Group
M.I.T., E40-294
Cambridge, MA 02139

April 1990

Advances in computer software allow modellers to design, with relative ease, sophisticated, realistic educational tools. With these advances, new issues arise about how to make this educational software productive and stimulating, without limiting the freedom of the user or creating simply a computerized workbook.

Such simulation games have great educational potential for people who play video and home computer games, and sometimes for students in classrooms. The games must address three information levels: (1) real-world details, (2) simulation of a model, and (3) conceptual understanding of structure and dynamics. The systems viewpoint on the particular model must be clearly explained; otherwise users will have much fun but learn little. Feedback during the game teaches this systems understanding without requiring textbook readings. Such feedback requires new modes of "expert" computer analysis which need to be developed. Other tools need to be developed to help in creation of simulation games and to give the games abilities that they do not yet have, such as access to databases of models, pictures, and text, and connections between simulation games.

Simulation Games as Stand-Alone Educational Tools

Simulation games are especially useful for situations in which someone learns without a teacher. Simulation games generally are not ideal for classroom education about dynamic systems. Students can learn better and think more freely if they use discussion, chalkboards, and modeling software such as STELLA (Richmond, Peterson, and Vescuso 1987). Modeling software forces them to build their own models. In contrast, simulation games supply models to students. The modeling process probably engages students in thought and learning more than simulation games do.

Some particular situations in which simulation games may be preferable to modeling software include: (1) The students do not know how to model and to use modeling software, and do not have time to learn. This is often true in one-day workshops. (2) The students work without teachers, and they do not have the resources or the motivation to build their own models. (3) A simulation game is full of details that are best taught by playing the game, rather than model-building aided by a text and classroom lectures.

Everyone who plays computer games for fun is a potential learner without a teacher. People typically play games for entertainment, not for learning, but it is logical to make these games as educational as possible while retaining the fun and interest they elicit. Simulation games that are sufficiently fun and flashy can address the broad market of people who play computer games. Thus, simulation games that are made to be stand-alone educational tools have enormous educational potential.

Levels of Abstraction: from Macro to Micro

Ideally, simulation games should be designed at three levels of abstraction, from macro to micro. (Fig. 1.) At the macro level, the user (the learner) examines structure and dynamics of the simulated system: (1) the key variables involved, (2) important ways in which these variables interact, and (3) how the interactions cause the observed dynamics. Causal loop diagrams with explanations can effectively present structure and dynamics. Software should be able to give detailed explanations of all variables; for example if the user clicks with the mouse on a variable name, its explanation appears. For advanced users, the game should have stock-and-flow diagrams and equations, with appropriate explanations, and if possible it should include a copy of the model to be used with a modeling program such as STELLA. This allows examination of the model's assumptions and keeps it from being a "black box."

For complex models, the macro level is a crucial part of the learning process. Without it, a simulation game is great fun, but the user learns almost nothing. All too often participants or users say that simulation games were great fun and, often, great educational tools, *when in fact the games have taught the participants very little about the dynamics of the simulated systems and how to control them.* This is fine if the purpose of a game is to excite people or to make them have fun, but it does not achieve educational potential.

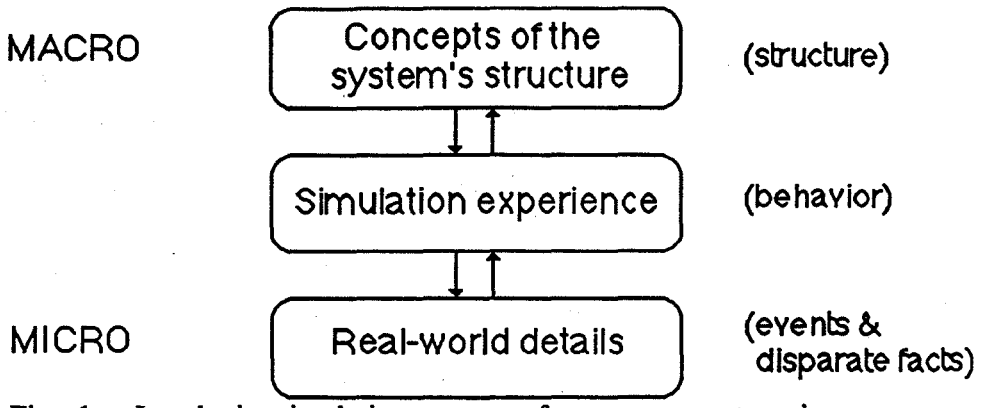

Fig. 1. Levels in simulation game software: macro to micro.

In the middle level between macro and micro is the simulation experience. This is the essence of a simulation game. It is where events play themselves out according to the model that underlies the simulation. If the model is of urban development, businesses and slums spring up; unemployment figures change; the number of homeless increases or decreases; and the city's attractiveness to outsiders fluctuates. Many simulation games are now based only on simulation, with no macro-level understanding and few micro-level details. This is especially true of computer games intended for fun. With complex models, this will probably leave the user without an understanding of the system.

The educational value of a simulation comes from repeated simulation of a model. The user tries certain policies or parameters, looks at the results, and then simulates again with different policies or parameters. Through this closed-loop process, the user gradually builds an understanding of the system. The ability to simulate again and again is crucial to simulation game software for education.

At the micro level, computer software can provide a simulation game with the richness needed to make the situation seem real. For example, many people have used a model of the Kaibab deer plateau (Sterman 1979) and a STELLAStack simulation game with this model (Richmond and Peterson, 1988). The model includes deer and predator populations, the amount of vegetation ("food") available for deer, and policy options such as hunting of deer and transportation of predators. One point of the model is to devise policies that will keep the deer herd from overpopulation and starvation. People learn with the model, but they generally do not understand the concept of thousands of deer starving, or what a starving deer looks like, or what it means that there is very little "food" left. This experience is too far removed from most people's lives for them to understand. A simulation game can provide them with the details they need for this understanding, for example photos of starving deer and of trees stripped bare, and conversations with rangers and park visitors.

Similarly, a simulation game can provide information about what certain parts of a model mean in practice: how hunting permits work, what kinds of food deer eat, what it would take to get truck or airlift predators to the Kaibab plateau, etc.

Micro-level details are a valuable part of educational software. Explanation of these details as part of a simulation provides a rich and powerful way to teach. For instance, in a biology class, students learn (1) many details of organs, hormones, and chemicals, and (2) how these separate entities lead to the dynamics of chemicals and hormones in the body, breathing, etc. Simulation game software, if well-designed, addresses details and dynamics.

When used properly, real details keep a simulation game interesting. Without them, the game will likely be dry and boring, so that the user may soon quit. It is also important not to swamp the user in a wash of detail which removes all attention from dynamics. A game designer should ask the key question: "Do people who use my simulation game learn to understand and control its important dynamics?"

For the present, the main factor that controls use of micro-detail in simulation games is how much the development process can afford. Detail is becoming easier to include with new computer technologies, but in any case it can be a big job. The developer of a simulation game as educational software must make a decision as to how detailed the game will be, so that he can finish the product given his available resources.

All three levels of abstraction can be used simultaneously, as is often desirable. During a simulation, the details of the micro-level will help the game seem realistic, flashy, and interesting. Macroscopic understanding of the system can give the learner more chances to learn than just at the end of a simulation, since explanations of system structure and dynamics become tools that the learner uses to help formulate policy.

For example, during the Kaibab plateau game, there could be pictures of the plateau and of deer and predators, mail that arrives from the Sierra Club and from park rangers, and billing notices from consultants who helped conduct a deer count. At the same time, the learner could continually consult a causal loop diagram that helps her think through the effects of her policies, and graphs of what has happened so far to confirm the effects she expected from earlier policies.

Fun or Learning? — A Dilemma Resolved

There is a developer's dilemma between (1) a fun computer game that teaches little and (2) boring educational software that functions like a workbook or a textbook. A game's developer should not become trapped

in the dilemma, but avoid it altogether. To do so, he can make software in which the learner controls the game, but in the midst of or after each simulation the computer explains the dynamics and structure of what just happened, so that the learner can generate ideas for her own improvement. As the learner continues to play and to receive instructional feedback from the computer, she will learn about the system in the game, its dynamics, and the reasons for the results of her policies.

In preliminary tests of a computerized learning laboratory, Senge discovered that businessmen very much enjoyed a simulation game of their business, but that they learned little. Play did not yield learning. To resolve his learning lab problem, Senge created a structured set of discussions and exercises which participants follow. A teacher runs the two- to three-day seminars. While participants do spend time in which they play with the simulation and try out their own ideas, most of their time is spent on structured discussion and exercises. The success of this method is under study, to see whether the participants learn about the system represented by the model, its dynamics, and appropriate policy decisions for those systems (Senge 1987, 1989; Kim 1989).

A teacher's structured guidance will not be available to people who learn on their own, and a long and highly structured study plan will turn away all but especially motivated people. For a general audience of people who will work on their own, it is important to come up with independent software that (1) captivates people and sustains their interest as long as the learning takes, and (2) leads them to successfully learn. In addition, it is desirable to (3) let the user direct her own learning. Learner-directed learning (a) keeps people's attention via their direct involvement and (b) helps make the software responsive to the user's needs.

To achieve these goals, a simulation game can be made to work as follows: It may have a brief introduction, but the user takes control of the game right away. She becomes an active participant in the simulation and makes decisions to try to keep the simulated system working well. The simulation evolves throughout the game play, which may last for a minute or for several hours. (Long simulations have the disadvantage that they limit the number of repeated trials for which the learner has time.) The game gives her feedback (1) as she plays it, (2) at the end of each simulation, or (3) both while she plays *and* after each simulation. This feedback shows her the structure of the simulated system, and how the structure results in its dynamics. Ideally, it addresses the specific dynamics that the user has caused. Also ideally, this feedback exists as a tool that the user can access at any time. This tool to explain the system becomes an integral part of the game because it is inherently useful to help the user "win" the game. In a sense, it is the user's consultant during the game.

Feedback to the Learner — Problems in Computer Analysis

To create intelligent feedback is a difficult problem, if it is to address what the learner does each time she plays a game. The computer must analyze why the user's policies give certain results. It must pick out certain feedback loops that are most relevant to the issue in question, then explain these loops and their dynamics to the user. It should explain why her policies give the results that she sees during the game.

The analysis is easiest when the user can set only a few parameters in a model, and she sets those parameters only at the beginning of the simulation. In this case, it is relatively easy to identify particular ranges of parameters that give certain types of results. For example, what policies to hunt deer and predators, chosen at the beginning of a game, result in starvation of large numbers of deer or predators, versus what policies always work well. In this example, two policy decisions (hunting of deer and hunting of predators) could be represented as variables on the two axes of a two-dimensional graph. In certain regions on this graph, the results could be classified into result groups such as "dynamics of sort A happen," "dynamics of sort B happen," and "dynamics of sort C happen." For each of these groups, the computer could give a different explanation of what happened in terms of feedback loops and the dynamics that result from these loops.

The above method works fine with a small number of result groups, but it becomes far too complex for (1) games in which the learner makes decsions about many different variables that affect the results, and (2) games in which people play continuously, instead of setting parameters only at the beginning. (1) For games that have many different variables, result groups are not on a two-dimensional graph, but in many-dimensional space, with as many dimensions as there are policies. With many dimensions, it is exponentially harder to divide sets of policy decisions into result groups. No one wants to have to find all possible results for a game with 20 decisions which all significantly affect the game's dynamics, and more importantly, no one wants to write explanations of feedback and dynamics for each of the many results. (2) Continuous-play games complicate the problem further because the learner may not follow any clear policy. She may switch policies in mid-game, and do so repeatedly.

Thus, for all but the simplest games, another method of analysis is needed. Perhaps the solution to this problem is to wait until computer experts manage to create high-powered expert systems or artificial intelligence systems that do the job. However, the following description

suggests a method of analysis that may be useful for the process. To this author's knowledge, these methods have not been put into practice.

The method is to analyze which feedback loops are important for long-term and short-term processes. The computer can calculate the gain of each feedback loop, because it knows each variable's numeric value and equation. It can list which feedback loops are especially strong positive or negative loops at any given time. In addition, it can look for changes in variables that happen slowly, but that continue over the long term — for example, a creeping but steady increase in how much a government monitors individuals.

Once the computer has its own list of what feedback loops and variables are changing quickly or steadily, it can compare these with a list of what variables or sectors the game's designer considers important. The computer may find strong dynamics in one sector of the model that happens to be of little importance in explanation, so the designer would tell the computer to ignore that sector. In some cases, the designer may tell the computer to note when certain variables do *not* change rapidly, because they should change when all goes well. The analysis can be done during the play of the game. Once the computer has completed some amount of analysis, at the end of or in the midst of a simulation, it can provide feedback to the learner.

The game should give this feedback in a form that allows the learner to explore all sorts of information about the model, but that keeps relevant factors most prominent. Even if the computer has found a lot of information that is important to present, it should not deluge the learner with that information. Rather, it should carefully organize its presentation in logical parts, perhaps according to sectors in the model. It should not force the learner to look at all this information, but let her choose what to see.

Interlinkages Between A Simulation Game and Other Games and Software

In future there may be an enormous amount of simulation game software, computerized encyclopediae, and other educational and information-storage software. (Fig. 2.) In an ideal learner-directed learning environment, the user should be able to switch from one simulation to a related one by expressing interest in the latter situation. For example, the learner might be studying the dynamics of deer on the Kaibab plateau when she comes across a reference to the dynamics of forest growth. She could then tell the computer to "teach me about forest growth." Simple versions of such software interconnections are now

feasible, but normally prohibitively expensive to develop. As software develops to manage large amounts of information, interconnected learning environments may become commonplace.

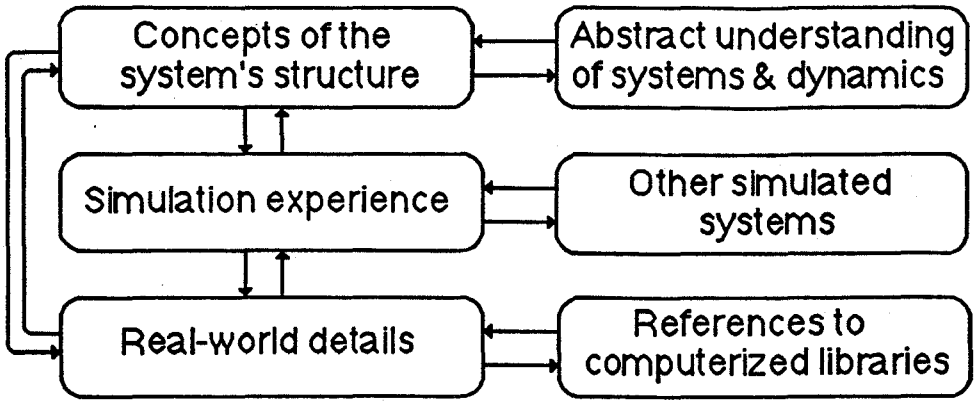

Fig. 2. Simulation games and other software could be interlinked to form a large educational system that covers many topics.

Simulation game software should also interlink in appropriate ways to (1) computerized libraries containing facts, pictures, historical events and patterns, maps, film, sounds, music, etc.; and (2) educational tools designed to teach about systems and their dynamics on an abstract level that applies to all dynamic situations. Access to large libraries of information is growing exponentially as electronic data storage becomes practical for billions of bytes at a time, and as software tools evolve to catalogue and access data and to interconnect different computers and different information storage formats. Educational software to teach about systems and their dynamics will surely develop soon, since (a) many people would like to interlink this software to their simulation games, and (b) such software would be easy to create. In its most rudimentary form, the software could even take the form of a computerized book with illustrations and an index. Such software would be a good thesis topic for a student with a working knowledge of system dynamics.

An Example Simulation Game

The following is an idealized example of a simulation game. It takes into account the various goals mentioned in this paper as important for some or all simulation games:

1. Fun *and* education for the learner.
2. Simultaneous use of three levels of abstraction: (a) participative simulation, (b) micro-details, and (c) macroscopic explanations of system structure and dynamics.

3. Suitability for use at home, where there is no teacher.
4. Learner-directed learning.
5. Instructional feedback about the dynamics of the game available throughout the simulation.
6. Interlinkages with other models, educational software, and databases.

Not all simulation games should have all these goals. Games for workshops and class assignments may purposely be more academic. At present, some of these goals cannot be met because current computer technologies make the development of a full-featured game an expensive and lengthy process which only half-heartedly meets some of the above goals.

Here is the example: Someone is using a game about urban development and housing problems. She plays the role of mayor. In front of her on a 40-inch screen is a map of simulated Boston showing areas under development, slums, business districts, and factories. To the right of the map a message from a city councilman appears, along with a picture of the man. The message says that the councilman is annoyed at having to spend much money on housing projects in Dorchester, and that he wants instead to direct the money to public schools. He asks for the mayor's political support in an upcoming council vote. The "mayor" uses the computer's mouse to click on Dorchester, and pictures of Dorchester appear. Two balding men in tattered clothes and blankets huddle over a warm air vent, and one has his hand extended in a plea for loose change. His cracked voice comes over the computer's speaker: "Can you spare a quarter, please?" In another picture, this one a video, a youth gang loiters on a corner. A child comes out of a nearby school and runs past them toward home, terrified by their taunts and jeers. These pictures serve to make the simulation real, to bring a sense of importance that can so often be lost in academic work.

But academic understanding is important too. The mayor examines a causal diagram that illustrates important feedback loops, using words, arrows, and pictures. She tells the computer to show Boston's history of development and housing. The feedbacks come to life: the arrows in strong positive loops pulse with a hot red glow; the ones in strong negative loops fight against the positive loops with cold blue pulses. Counters that indicate population, unemployment, and available housing whirl madly as the year goes from 1780 to 1990. A picture of a factory expands to indicate growth in the number of factories in Boston. At the bottom, graphs display historical trends of the most important variables. The mayor clicks on a button for a summary report, and a document appears. It uses simple text and sketches of feedback loops to explain what happened. "In the 1700's, as Boston became a center for trade, more and more people moved into the area. People attracted businesses, which

created more jobs, attracted more people, and made the area a bigger trading center. This spiral growth led to...." After she reviews the explanation, the mayor switches to the present. She looks at causal loop structures and at graphs over the past ten years of unemployment, housing costs, and her (declining) popularity. Chicago has had the same problems, hasn't it? She reviews the dynamics of that city, especially recent dynamics, just as she did with the historical dynamics of Boston. After a little more thought, she is ready to answer the councilman, and to go on with the business of running Boston.

Tools to Develop Simulation Games

Computer technologies needed to create simulation games have advanced rapidly. These technologies are complicated and build upon each other, and their final form is hard to predict until programmers actually create them. Computer-to-user interfaces are a technology that has, in the last few years, suddenly made development of simulation games much more feasible. Until recently, building sophisticated interfaces required knowledgeable programmers and lots of time. Recent software such as HyperCard (Apple Computer 1988) has simplified interface development and made it feasible even for inexperienced programmers. The programmer selects and moves boxes, buttons, pictures, and other elements, and he can tell each element to perform simple or complicated programs, with code that he or someone else has written. Although software like HyperCard has faults, it is a great advance over what was before. It has helped to shift emphasis from programming of interfaces toward other issues such as educational value.

A number of other technologies are important in simulation game design. (Table 1.) Modeling software, such as STELLA and Dynamo (Pugh 1986), allows people to efficiently formulate and try out models. Ideally, the software should include advanced features to help analyze and test models, including abilities to examine stock-and-flow and causal-loop structures, to calculate feedback loop gains, to check dimensional consistency, and to perform automated policy, parameter, and Monte Carlo analyses. Software that converts models to computer code should let simulation game developers attach to their games the models that they create with modeling software. In the games they should retain use of advanced features of the modeling software. STELLAStack is one such program that allows a game designer to attach STELLA models to his games, and other such programs exist or will soon exist. Multimedia tools, which facilitate use of graphics, video, sound, and animation, are another forefront of software development. At present, multimedia software is

typically unreliable, and industry standards have not been defined. It is still hard to include video and animation in simulation games. Hypertext and database-access tools are also still in developmental stages. Hypertext creates linkages between text, so that a user can explore topics on his own, rather than read from front to back. For example, a user who noticed a reference to cougars could click on the word "cougars" to learn about the species. Database-access tools allow access to information such as encyclopedia entries, weather data, street maps, music, and paintings. These tools are likely to become more prevalent as costs decrease for storing huge amounts of data, and as such information storage systems become commonplace.

Technologies Important to Simulation Game Design:
• Interface environments & tools to build interfaces.
• Model construction and analysis software.
• Programs to incorporate models and model analysis into other software.
• Multimedia tools.
• Hypertext and database tools.

Table 1. Technologies for Simulation Game Design.

These features need to be put into (1) coherent game design environments, or (2) toolboxes with the tools that programmers need. Game design environments should allow easy construction of interrelated screens which incorporate graphics, text, and automated objects such as buttons, dials, tables, and graphs. Existing design environments either fail to provide a complete set of easy-to-use tools to create polished simulation games, or restrict a designer to a limited format. In the letter case are, for example, Dynamo and MicroWorlds (Diehl), and in the former case is STELLAStack. This author (Simons 1989) has created a prototype design environment which uses HyperCard. The environment allows someone to quickly create a polished simulation game with any STELLA model, and to customize it as they wish, using a variety of standard tools in the process. The prototype took about 500 hours to construct, and a full-fledged version would likely require 1000 hours with experienced HyperCard programmers, if they take advantage of tools and knowledge that others have created. While such a software-engineering environment is valuable, toolboxes of commonly used routines and objects can be almost as powerful when used by knowledgeable programmers, and they allow more flexibility for development of future toolboxes and design environments.

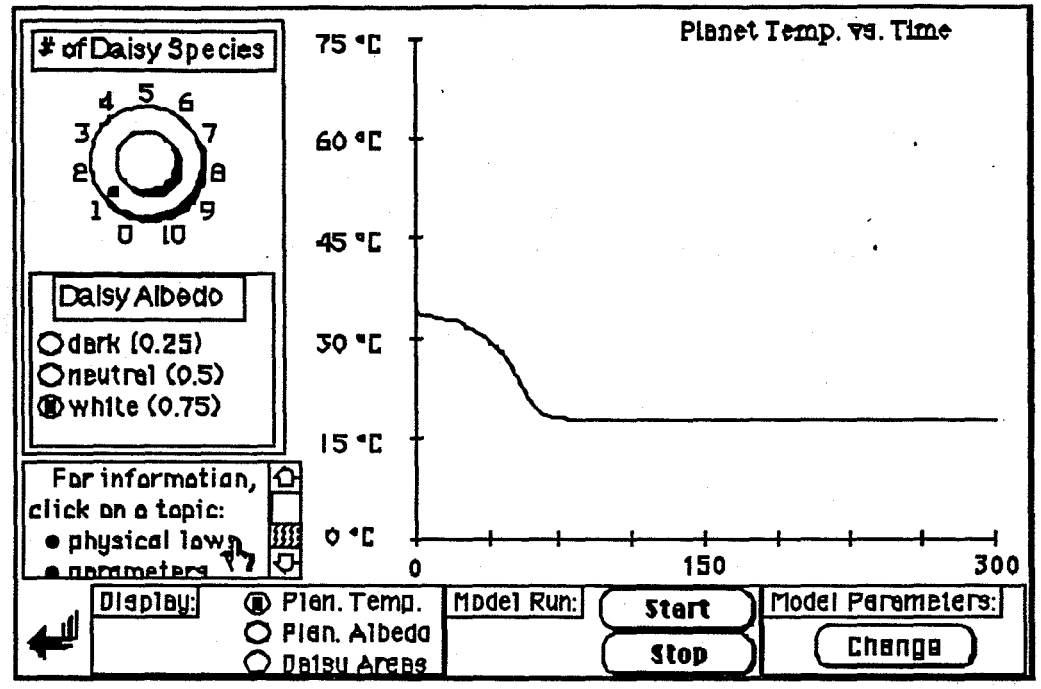

Fig. 3. A screen from a simulation game that might be made with SIMGAME, an environment in which to develop simulation games. A prototype version of SIMGAME has been created at M.I.T. by this author. Individual elements can be placed on interlinked pages via simple commands.

References

Apple Computer, Inc. 1988. *HyperCard® User's Guide*. Manual for computer software. Cupertino, California: Apple Computer.

Diehl, E. *MicroWorlds*. Computer software. Cambridge, MA: MicroWorlds.

Graham, A. K., and P. M. Senge. 1990. Computer-Based Case Studies and Learning Laboratory Projects. *System Dynamics Review* 6 (1): 100-105.

Kim, D. H. 1989. Learning Laboratories: Designing a Reflective Learning Environment. In *Proceedings of the 1989 International Conference of the System Dynamics Society*, Stuttgart.

Pugh, A. L. 1986. *Professional DYNAMO Plus*. Computer software and manual. Cambridge, MA: Pugh-Roberts Associates.

Richmond, B., S. Peterson, and P. Vescuso. 1987. *An Academic User's Guide to STELLA*. Macintosh computer software and manual. Lyme, NH: High Performance Systems.

Richmond, B., and S. Peterson. 1988. *STELLAStack*. Macintosh computer software and manual. Lyme, NH: High Performance Systems.

Senge, P. M. 1987. Catalyzing Systems Thinking within Organizations. System Dynamics Group Working Paper D-3877-9.

_____. 1989. Organizational Learning: New Challenges for System Dynamics. System Dynamics Group Working Paper D-4023.

Simons, K. L. 1989. SIMGAME User Manual. System Dynamics Group Working Paper D-4115.

_____. 1989. SIMGAME Development Manual. System Dynamics Group Working Paper D-4116.

_____. 1989. SIMGAME Code. System Dynamics Group Working Paper D-4117.

Sterman, J. D. 1979. Kaibab Plateau Model. System Dynamics Group Working Paper D-3032-10.