# Decentralized Modeling and Decentralized Thinking

Mitchel Resnick
MIT Media Laboratory
mres@media.mit.edu

Published in:
*Modeling and Simulation in Precollege Science and Mathematics (pp. 114-137)*,
edited by W. Feurzeig and N. Roberts. Springer: New York.

## 1. Introduction: The Era of Decentralization

It seems fair to say that we live in an Era of Decentralization. Almost every time you pick up a newspaper, you can see evidence of the growing interest in decentralized systems. On the front page, you might read an article about the transition of the former Communist states from centrally-planned economies to market-based economies. Turn to the business page, and you might find an article about the shift in corporate organizations away from top-down hierarchies toward decentralized management structures. The science section might carry an article about new distributed models of the mind, and it might include a technology column about the role of the Internet in promoting distributed approaches to computing. And in the book review, you might discover how the latest literary theories are based on the idea that literary meaning itself is decentralized, always constructed by individual readers, not imposed by a centralized author.

But even as the influence of decentralized ideas grows within our culture, there is a deep-seated resistance to such ideas. At some deep level, people seem to have strong attachments to centralized ways of thinking. When people see patterns in the world, they often assume that there is some type of centralized control, even when it doesn't exist. For example, most people assume that birds in a flock play a game of follow-the-leader: the bird at the front of the flock leads, and the others follow. But that's not so. In fact, most bird flocks don't have leaders at all. Rather, each bird follows a set of simple rules, reacting to the movements of the birds nearby it. Orderly flock patterns arise from these simple, local interactions. The bird in front is not a "leader" in any meaningful sense—it just happens to end up there. The flock is organized without an organizer, coordinated without a coordinator. Yet most people continue to assume the existence of a "leader bird."

This assumption of centralized control, a phenomenon I call the *centralized mindset*, is not just a misconception of the scientifically naive. It seems to affect the thinking of nearly everyone. Until recently, even scientists assumed that bird flocks must have leaders. It is only in recent years that scientists have revised their theories, asserting that bird flocks are leader-less and self-organized (Heppner and Grenander, 1990; Reynolds, 1987). A similar bias toward centralized theories can be seen throughout the history of science.

In this chapter, I discuss how computer-modeling activities can help people move beyond the centralized mindset, helping them gain new insights into (and appreciation for) the workings of decentralized systems. In particular, I will discuss a programmable modeling environment, called StarLogo, that I developed to help pre-college students model and explore decentralized systems. By presenting "stories" of students' activities with StarLogo, I hope to shed light on the nature of the centralized mindset and on ways of moving beyond it. At the same time, I have a more general goal: to present (and defend) a set of principles to guide the uses of computer modeling in science education.

## 2. Learning Through Modeling

My "decentralized modeling" research project has been guided by five core principles. In my view, these principles apply not only to my own research project, but to all applications of computer modeling in science education.

• *Principle 1: Encourage construction of models (not just manipulation of pre-existing models).* In many educational applications of computer modeling, students do little more than twiddle parameters on pre-constructed models. For example, they are given a model of a spring with a mass on the end, along with sliders for controlling the spring constant and mass. That type of activity can have some value. But students are likely to make much deeper connections with the concepts underlying the model if they are given the opportunity to construct models on their own (Papert, 1991). Accordingly, I designed StarLogo as a *programmable* modeling environment, with which students can construct their own models.

• *Principle 2: Rethink what is learned (not just how it is learned).* The activity of computer modeling provides a new opportunity for students to learn through exploration and experimentation. But often overlooked is the potential to use modeling to rethink not just the process but the content of science education. Too often, educators use computer modeling as a new way to teach the same old things (e.g., the motion of springs). In my work, the emphasis has been on using modeling to help students explore ideas and concepts that were previously inaccessible. For example, ideas about decentralized systems and self-organizing systems have traditionally been taught at the graduate level, using advanced mathematics. StarLogo was designed to make these ideas accessible to pre-college students, without any advanced mathematics.

• *Principle 3: Support true computational models (not just computerization of traditional mathematical models).* For several hundred years, mathematicians and scientists have used differential equations to model dynamical systems. Many computer-modeling tools re-implement this approach on the computer, using the computer to numerically solve differential equations. These tools are certainly very useful, and some of them do a very good job of hiding the formal mathematics under graphical descriptions of the differential equations. But the most fundamental contributions of computer modeling are likely to come from tools that are based on totally new representations, tailored explicitly for the computer. That is the case with StarLogo, which is based on hundreds of individual objects acting in parallel. This type of representation was not possible in the paper-and-pencil era, and it makes possible new ways for even young students to explore the workings of dynamical systems.

• *Principle 4: Facilitate personal connections (not just mathematical abstractions)*. In designing new types of learning tools, it is important to consider two types of connections (Resnick, Bruckman, and Martin, 1996). First, there are epistemological connections: how will the tool connect to important domains of knowledge and encourage new ways of thinking? But equally important are personal connections: how will the tool connect to users' interests, passions, and experiences? Many computer-modeling tools are "impersonal": students must manipulate either mathematical abstractions or aggregate quantities. StarLogo aims to be more "personal," encouraging students to think about the actions and interactions of individual (and familiar) objects.

• *Principle 5: Focus on stimulation (not just simulation)*. Many computer models try to imitate some real-word system or process as accurately as possible. Computer simulations of nuclear reactors are used to predict when the reactors might fail; computer simulations of meteorological patterns are used to predict tomorrow's weather. In these cases, the more accurate the simulation, the better. But, for educational applications of computer modeling, real-world fidelity should not be the top goal. Instead, the real world should serve only as an inspiration, a departure point for thinking about some set of ideas or concepts. The goal is not to simulate particular systems and processes in the world; it is to probe, challenge, and disrupt the way people think about systems and processes in general. That is the goal of StarLogo: to stimulate people to develop new ways of thinking about decentralized systems.

## 3. The Centralized Mindset

Before exploring how computer modeling can help people move beyond the centralized mindset, it is worth examining the nature of the centralized mindset. In some ways, the pervasiveness of the centralized mindset might seem surprising. After all, aren't we living in an Era of Decentralization? Actually, it isn't so surprising if we look at the growing interest in decentralization from a different perspective: Why are people becoming more interested in decentralized ideas *now*? Why didn't it happen before? Why have people resisted decentralized approaches in the past? What underlies this persistence of resistance? What made people cling onto centralized approaches so tightly, for so long?

The centralized mindset can be seen throughout the history of science. Until the mid-19th century, almost everyone embraced the idea that living systems were designed by some God-like entity. Even scientists were convinced by the so-called "watchmaker argument" (or the "argument from design"), proposed by theologian William Paley in his book *Natural Theology* (Paley, 1802). Paley noted that watches are very complex and precise objects. If you found a watch on the ground, you could not possibly believe that such a complex object had been created by random chance. Instead, you would naturally conclude that the watch must have ad a maker. For Paley, the same logic applies to living systems: they, too, must have a maker.

It is not surprising that scientists accepted Paley's argument in the early 19th century, since there were no viable alternative explanations for the complexity of living systems. What *is* surprising is how strongly scientists held onto centralized beliefs even after

Darwin provided a viable (and more decentralized) alternative. Science historian Ernst Mayr (1982) notes that biologists put up "enormous resistance" to Darwin's theories for a full 80 years after publication of *Origin of Species*, generally preferring more centralized alternatives.

The history of research on slime-mold cells, as told by Evelyn Fox Keller (1985), provides another example of centralized thinking. During their life cycle, slime-mold cells sometimes gather together into clusters. For many years, scientists believed that the aggregation process was coordinated by specialized slime-mold cells, known as "founder" or "pacemaker" cells. According to this theory, each pacemaker cell sends out a chemical signal, telling other slime-mold cells to gather around it, resulting in a cluster. In 1970, Keller and Segel (1970) proposed an alternative model, showing how slime-mold cells can aggregate without any specialized cells. Nevertheless, for the following decade, other researchers continued to assume that special pacemaker cells were required to initiate the aggregation process. As Keller (1985) writes, with an air of disbelief: "The pacemaker view was embraced with a degree of enthusiasm that suggests that this question was in some sense foreclosed." By the early 1980's, researchers began to accept the idea of aggregation among homogeneous cells, without any pacemaker. But the decade-long resistance serves as some indication of the strength of the centralized mindset.

People also view the workings of the economy in centralized ways, assuming singular causes for complex phenomena. Children, in particular, seem to assume strong governmental control over the economy. Of course, governments *do* play a large role in most economies, but children assume that governments play an even larger role than they actually do. In interviews with Israeli children between 8 and 15 years old, psychologist David Leiser (1983) found that nearly half of the children assumed that the government sets all prices and pays all salaries. Even children who said that employers pay salaries often believed that the government provides the money for the salaries. A significant majority of the students assumed that the government pays the increased salaries after a strike. And many younger children had the seemingly contradictory belief that the government is also responsible for organizing strikes. As Leiser writes: "The child finds it easier to refer unexplained phenomena to the deliberate actions of a clearly defined entity, such as the government, than to impersonal 'market forces.'"

In some ways, it is not surprising that people have such strong commitments to centralized approaches. Many phenomena in the world *are*, in fact, organized by a central designer. These phenomena act to reinforce the centralized mindset. When people see neat rows of corn in a field, they assume (correctly) that the corn was planted by a farmer. When people watch a ballet, they assume (correctly) that the movements of the dancers were planned by a choreographer. When people see a watch, they assume (correctly) that it was designed by a watchmaker.

Moreover, most people participate in social systems (such as families and school classrooms) where power and authority are very centralized. These hierarchical systems serve as strong models. Many people are probably unaware that other types of organization are even possible. In an earlier research project, I developed a programming language (called MultiLogo) based on "agents" that communicated with one another. In using the language, children invariably put one of the agents "in charge" of the others. One student explicitly referred to the agent in charge as "the teacher." Another referred to it as "the mother" (Resnick, 1990).

Perhaps most important, our intuitions about systems in the world are deeply influenced by our conceptions of ourselves. The human mind is composed of thousands of interacting entities (e.g., Minsky, 1987), but each of us experiences our own self as a singular entity. This is a very convenient, perhaps necessary, illusion for surviving in the world. When I do something, whether I'm painting a picture or organizing a party, I feel as if "I" am playing the role of the "central actor." It feels like there is one entity in charge: me. So it is quite natural that I should expect most systems to involve a central actor, or some entity that is in charge. The centralized mindset might be viewed as one aspect (and a lasting remnant) of the egocentrism that Piaget identified in early childhood.

## 4. Tools for Decentralized Thinking

In some ways, people already have a great deal of experience with decentralized systems: they observe decentralized systems in the natural world, and they participate in decentralized social systems in their lives. But, of course, observation and participation do not necessarily lead to strong intuitions or deep understanding. People observed bird flocks for thousands of years before anyone suggested that flocks are leader-less. Observation and participation are not enough. People need a richer sense of engagement with decentralized systems. One way to do that is to give people opportunities to *design* decentralized systems.

At first glance, this approach to the study of decentralized systems might seem like a contradiction. After all, how can you design decentralized phenomena? By definition, decentralized patterns are created without a centralized designer. But there are ways to use design in the study of decentralized systems. Imagine that you could design the behaviors of lots of individual components—then observe the patterns that result from all of the interactions. This is a different sort of design: You control the actions of the parts, not of the whole. You are acting as a designer, but the resulting patterns are not designed.

Over the years, computer scientists have developed a variety of computational tools that can be used for this type of "decentralized design." Cellular automata represent one example (Toffoli & Margolus, 1987). In cellular automata, a virtual world is divided into a grid of "cells." Each cell holds a certain amount of "state." (On the computer screen, different states are usually represented by different colors.) In the simplest cases, each cell might hold just a single piece of state, indicating whether the cell is "alive" or "dead." There is a transition rule that determines how each cell changes from one generation to the next. Transition rules are typically based on the states of a cell's "neighbors." For example, a cell might become "alive" if the majority of its neighboring cells are alive. Each cell executes the same rule, over and over. Cellular automata have proved to be an extraordinarily rich framework for exploring self-organizing phenomena. Simple rules for each cell sometimes lead to complex and unexpected large-scale structures.

To engage students in thinking about decentralized systems, I wanted to provide an environment similar to cellular automata, but more connected to students' interests and experiences. While cellular automata are well-suited for computer hackers and mathematicians, they seem ill-suited for people who have less experience (or less interest in) manipulating abstract systems. The objects and operations in cellular automata are not familiar to most people. The idea of writing "transition rules" for "cells" is not an idea that most people can relate to.

Instead, I decided to create an environment based on the familiar ideas of "creatures" and "colonies." The goal was to enable students to investigate the ways that colony-level behaviors (such as bird flocks or ant foraging trails) can arise from interactions among individual creatures.[1] Logo seemed like a good starting point for my computational system (Papert, 1980; Harvey, 1985). The traditional Logo "turtle" can be used to represent almost any type of object in the world: an ant in a colony, a car in a traffic jam, an antibody in the immune system, or a molecule in a gas. But traditional versions of the Logo language are missing several key features that are needed for explorations of colony-type behaviors. So I developed a new version of Logo, called StarLogo, that extends Logo in three major ways (Resnick, 1991, 1994).

First, *StarLogo has many more turtles*. While commercial versions of Logo typically have only a few turtles, StarLogo has *thousands* of turtles, and all of the turtles can perform their actions at the same time, in parallel.[2] For many colony-type explorations, having a large number of turtles is not just a nicety, it is a necessity. In many cases, the behavior of a colony changes qualitatively when the number of creatures is increased. An ant colony with 10 ants might not be able to make a stable pheromone trail to a food source, whereas a colony with 100 ants (following the exact same rules) might.

Second, *StarLogo turtles have better "senses."* The traditional Logo turtle was designed primarily as a "drawing turtle," for creating geometric shapes and exploring geometric ideas. But the StarLogo turtle is more of a "behavioral turtle." StarLogo turtles come equipped with "senses." They can detect (and distinguish) other turtles nearby, and they can "sniff" scents in the world. Such turtle-turtle and turtle-world interactions are essential for creating and experimenting with decentralized and self-organizing phenomena. Parallelism alone is not enough. If each turtle just acts on its own, without any interactions, interesting colony-level behaviors will generally not arise.

Third, *StarLogo reifies the turtles' world*. In traditional versions of Logo, the turtles' world does not have many distinguishing features. The world is simply a place where the turtles draw with their "pens." Each pixel of the world has a single piece of state information—its color. StarLogo attaches a much higher status to the turtles' world. The

[1] I am using the terms "creature" and "colony" rather broadly. On a highway, each car can be considered a "creature," and a traffic jam can be considered the "colony."

[2] The initial version of StarLogo was implemented on a massively-parallel computer, the Connection Machine. We have since implemented StarLogo on traditional sequential computers by simulating parallelism. To download a copy of StarLogo, see http://www.media.mit.edu/~starlogo/

world is divided into small square sections called *patches*. The patches have many of the same capabilities as turtles, except that they can not move. Each patch can hold an arbitrary variety of information. For example, if the turtles are programmed to release a "chemical" as they move, each patch can keep track of the amount of chemical that has been released within its borders. Patches can execute StarLogo commands, just as turtles do. For example, each patch could diffuse some of its "chemical" into neighboring patches, or it could grow "food" based on the level of chemical within its borders. Thus, the environment is given a status equal to that of the creatures inhabiting it.

StarLogo programs can be conceptualized as turtles moving on top of (and interacting with) a cellular-automata grid. All types of interactions are possible: turtle-turtle, turtle-patch, and patch-patch interactions. StarLogo places special emphasis on *local* interactions—that is, interactions among turtles and patches that are spatially near one another. Thus, StarLogo is well-suited for explorations of self-organizing phenomena, in which large-scale patterns arise from local interactions. In addition, the massively parallel nature of StarLogo makes it well-suited for explorations of probabilistic and statistical concepts—and studies of people's thinking about these concepts (Wilensky, 1993).

Figure 1 shows a StarLogo simulation of slime-mold cells aggregating into clusters. In this example, each cell emits a chemical pheromone, and it also moves in the direction where the pheromone is strongest (that is, it "follows the gradient" of the pheromone). At the same time, the patches cause the pheromone to diffuse and evaporate. With this simple strategy, the cells quickly aggregate into clusters—demonstrating that aggregation can arise from a decentralized mechanism.

In some ways, the ideas underlying StarLogo parallel the ideas underlying the early versions of Logo itself. In the late 1960's, Logo aimed to make then-new ideas from the computer-science community (like procedural abstraction and recursion) accessible to a larger number of users. Similarly, StarLogo aims to make 1990's ideas from computer science (like massive parallelism) accessible to a larger audience. And whereas Logo introduced a new object (the turtle) to facilitate explorations of particular mathematical/scientific ideas, such as differential geometry (Abelson & diSessa, 1980), StarLogo introduces another new object (the patch) to facilitate explorations of other mathematical/scientific ideas (such as self-organization).

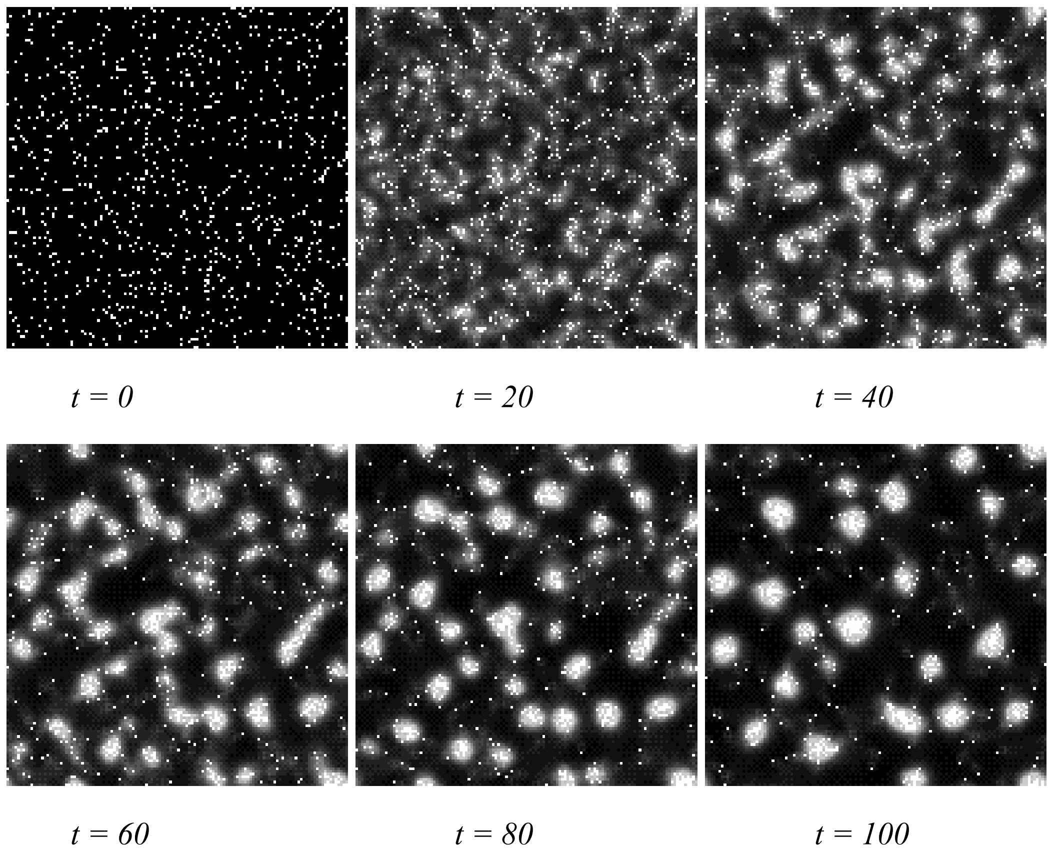

*Figure 1: Slime-mold cells aggregating into clusters*

## 5. StarLogo Stories

This section presents stories of student projects with StarLogo—describing the models that students constructed and what they learned in doing so. The students typically came to MIT for eight to ten sessions, each lasting 60 to 90 minutes. Most students worked together in pairs. I worked directly with the students, suggesting projects, asking questions, challenging assumptions, helping with programming, and encouraging students to reflect on their experiences as they worked with StarLogo. Computer interactions were saved in computer files, and all discussions were recorded on audio tape. In the early sessions, I typically showed students existing StarLogo programs. The students experimented with the programs, trying different parameters and making slight modifications of the programs. As the sessions progressed, I encouraged students to develop their own projects ideas and construct their own models, based on personal interests.

### 5.1 Traffic Jams

Ari and Fadhil were students at a public high-school in the Boston area. Both enjoyed working with computers, but neither had a very strong mathematical or scientific background. At the time Ari and Fadhil started working with StarLogo, they were also taking a driver's education class. Each had turned 16 years old a short time before, and they were excited about getting their driver's licenses. Much of their conversation focused on cars. When I gave Ari and Fadhil a collection of articles to read, a *Scientific American* article titled "Vehicular Traffic Flow" (Herman & Gardels, 1963) captured their attention.

Traffic flow is rich domain for studying collective behavior. Interactions among cars in a traffic flow can lead to surprising group phenomena. Consider a long road with no cross streets or intersections. What if we added some traffic lights along the road? The traffic lights would seem to serve no constructive purpose. It would be natural to assume that the traffic lights would reduce the overall traffic throughput (number of cars per unit time). But in some situations, additional traffic lights actually *improve* overall traffic throughput. The New York City Port Authority, for example, found that it could increase traffic throughput in the Holland Tunnel by 6 percent by deliberately stopping some cars before they entered the tunnel (Herman & Gardels, 1963).

Traditional studies of traffic flow rely on sophisticated analytic techniques (from fields like queuing theory). But many of the same traffic phenomena can be explored with simple StarLogo programs. To get started, Ari and Fadhil decided to create a one-lane highway. (Later, they experimented with multiple lanes.) Ari suggested adding a police radar trap somewhere along the road, to catch cars going above the speed limit. But he also wanted each car to have its own radar detector, so that cars would know to slow down when they approached the radar trap.

After some discussion, Ari and Fadhil decided that each StarLogo turtle/car should follow three basic rules:

- If there is a car close ahead of you, slow down.
- If there are not any cars close ahead of you, speed up (unless you are already moving at the speed limit).
- If you detect a radar trap, slow down.

Ari and Fadhil implemented these rules in StarLogo. They expected that a traffic jam would form behind the radar trap, and indeed it did (Figure 2). After a few dozen iterations of the StarLogo program, a line of cars started to form to the left of the radar trap. The cars moved slowly through the trap, then sped away as soon as they passed it. Ari explained: "First one car slows down for the radar trap, then the one behind it slows down, then the one behind that one, and then you've got a traffic jam."

I asked Ari and Fadhil what would happen if only *some* of the cars had radar detectors. Ari predicted that only some of the cars would slow down for the radar trap. Fadhil had a different idea: "The ones that have radar detectors will slow down, which will cause the other ones to slow down." Fadhil was right. The students modified the StarLogo program so that only 25 percent of the cars had radar detectors. The result: the traffic flow looked exactly the same as when all of the cars had radar detectors.

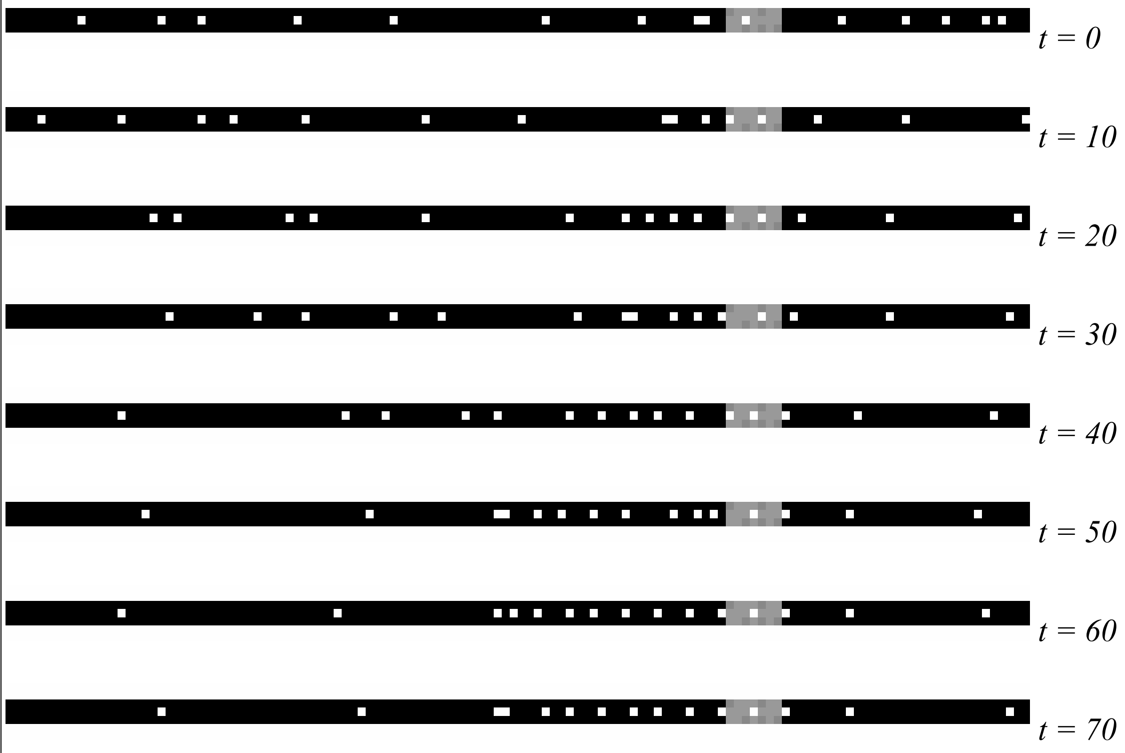

*Figure 2: Traffic jam caused by radar trap (shaded area)*
*(Cars move left to right)*

What if *none* of the cars had radar detectors—or, equivalently, if the radar trap were removed entirely? With no radar trap, the cars would be controlled by just two simple rules: if you see another car close ahead, slow down; if not, speed up. The rules could not be much simpler. At first, Fadhil predicted that the traffic flow would become uniform: cars would be evenly spaced, traveling at a constant speed. Without the radar trap, he reasoned, what could cause a jam? But when the students ran the program, a traffic jam formed (Figure 3). Along parts of the road, the cars were tightly packed and moving slowly. Elsewhere, they were spread out and moving at the speed limit.

Ari and Fadhil were surprised. And when I showed Ari and Fadhil's program to other high-school students, they too were surprised. In general, the students expected the cars to end up evenly spaced along the highway, separated by equal distances. Several of them talked about the cars reaching an "equilibrium," characterized by equal spacing. No one expected a traffic jam to form. Some of their predictions:

> *Emily: [The cars will] just speed along, just keep going along...they will end up staggered, in intervals.*
>
> *Frank: Nothing will be wrong with it. Cars will just go...There's no obstacles. The cars will just keep going, and that's it.*

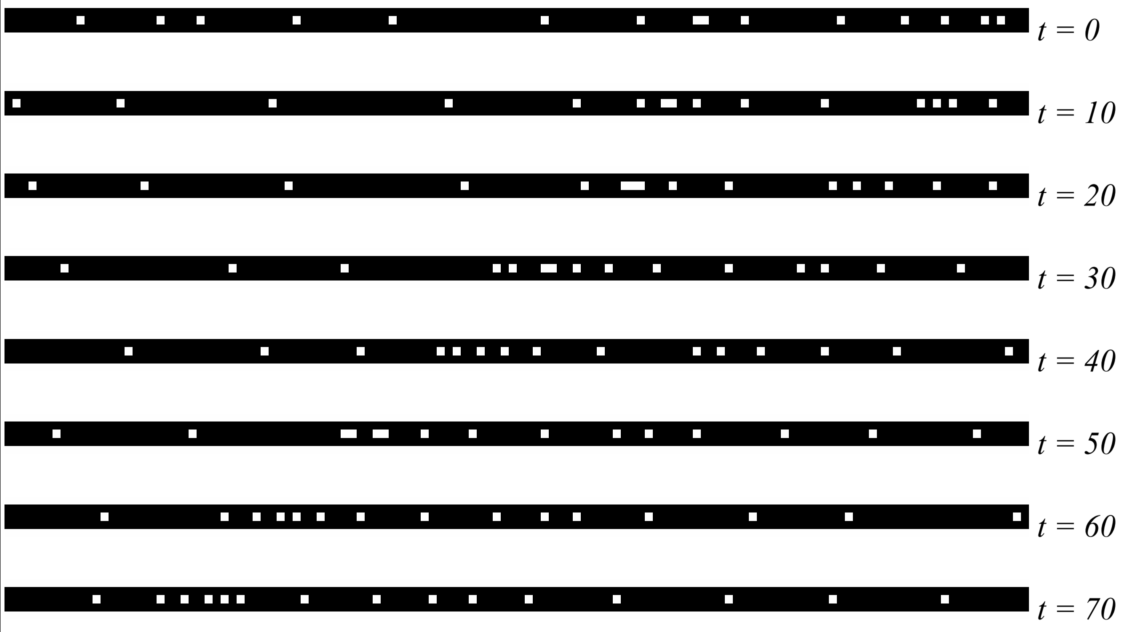

*Figure 3: Traffic jam without radar trap*
*(Cars move left to right, but jam moves right to left)*

> *Ramesh: They will probably adjust themselves to a uniform distance from each other.*

When I ran the simulation, and traffic jams began to form, the students were shocked. In their comments, most students revealed a strong commitment to the idea that some type of "seed" (like an accident or a broken bridge) is needed to start a traffic jam. Perhaps Frank expressed it best: "I didn't think there would be any problem, since there was nothing there." If there is *nothing there*—if there is no seed—there should not be a traffic jam. Traffic jams do not just happen; they must have localizable causes. And the cause must come from outside the system (not from the cars themselves). Some researchers who study systems talk about *exogenous* (external) and *endogenous* (internal) factors affecting the behavior of a system. In the minds of the students, patterns (such as traffic jams) can be formed only by exogenous factors.

Fadhil suggested that the jams were caused by differences in the initial speeds of the cars. So the students changed the StarLogo program, starting all of the cars at the exact same speed. But the jams still formed. Fadhil quickly understood. At the beginning of the program, the cars were placed at random positions on the road. Random positioning led to uneven spacing between the cars, and uneven spacing could also provide the "seed" from which a traffic jam could form. Fadhil explained: "Some of the cars start closer to other cars. Like, four spaces between two of them, and two spaces between others. A car that's only two spaces behind another car slows down, then the one behind it slows down."

Next, they changed the program so that the cars were evenly spaced. Sure enough, no traffic jams formed. All of the cars uniformly accelerated up to the speed limit. But Ari and Fadhil recognized that such a situation would be difficult to set up in the real world. The distances between the cars had to be just right, and the cars had to start at exactly the same time—like a platoon of soldiers starting to march in unison.

## 5.2 Termites and Wood Chips

Termites are among the master architects of the animal world. On the plains of Africa, termites construct giant mound-like nests rising more than 10 feet tall, thousands of times taller than the termites themselves. Inside the mounds are intricate networks of tunnels and chambers. Each termite colony has a queen. But, as in ant colonies, the termite queen does not “tell” the termite workers what to do. On the termite construction site, there is no construction foreman, no one in charge of the master plan. Rather, each termite carries out a relatively simple task. Termites are practically blind, so they must interact with each other (and with the world around them) primarily through their senses of touch and smell. From local interactions among thousands of termites, impressive structures emerge.

The global-from-local nature of termite constructions makes them well-suited for StarLogo explorations. Callie, one of the high-school students, worked on a simple form of termite construction: she programmed a set of termites to collect wood chips and put them into piles. At the start of the program, wood chips were scattered randomly throughout the termites’ world. The challenge was to make the termites organize the wood chips into a few, orderly piles.

Callie and I worked together on the project. We started with a very simple strategy, programming each individual termite to obey the following rules:

- If you are not carrying anything and you bump into a wood chip, pick it up.
- If you are carrying a wood chip and you bump into another wood chip, put down the wood chip you’re carrying.

At first, Callie and I were both skeptical that this simple strategy would work. There was no mechanism for preventing termites from taking wood chips away from existing piles. So while termites are putting new wood chips on a pile, other termites might be taking wood chips away from it. It seemed like a good prescription for getting nowhere. But we pushed ahead and implemented the strategy in a StarLogo program, with 1000 termites and 2000 wood chips scattered in a 128x128 grid.

We tried the program, and (much to our surprise) it worked quite well. At first, the termites gathered the wood chips into hundreds of small piles. But gradually, the number of piles declined, and the number of wood chips in each pile increased (see Figure 4). After 2000 iterations, there were about 100 piles, with an average of 15 wood chips in each pile. After 10,000 iterations, there were fewer than 50 piles left, with an average of 30 wood chips in each pile. After 20,000 iterations, only 34 piles remained, with an average of 44 wood chips in each pile. The process was rather slow, and it was frustrating to watch, as termites often carried wood chips away from well-established piles. But, all in all, the program worked quite well.

Why did it work? As we watched the program, it suddenly seemed obvious. Imagine what happens when the termites (by chance) remove all of the wood chips from a particular pile. Because all of the wood chips are gone from that spot, termites will never again drop wood chips there. So the pile has no way of restarting.

As long as a pile exists, its size is a two-way street: it can either grow or shrink. But the *existence* of a pile is a one-way street: once it is gone, it is gone forever. Thus, a pile is somewhat analogous to a species of creatures in the real world. As long as the species exists, the number of individuals in the species can go up or down. But once all of the individuals are gone, the species is extinct, gone forever. In these cases, zero is a "trapped state": once the number of creatures in a species (or the number of wood chips in a pile) goes to zero, it can never rebound.

Of course, the analogy between species and piles breaks down in some ways. New species are sometimes created, as offshoots of existing species. But in the termite program, there is no way to create a new pile. The program starts with roughly 2000 wood chips. These wood chips can be viewed as 2000 "piles," each with a single wood chip. As the program runs, some piles disappear, and no new piles are created. So the total number of piles decreases monotonically.

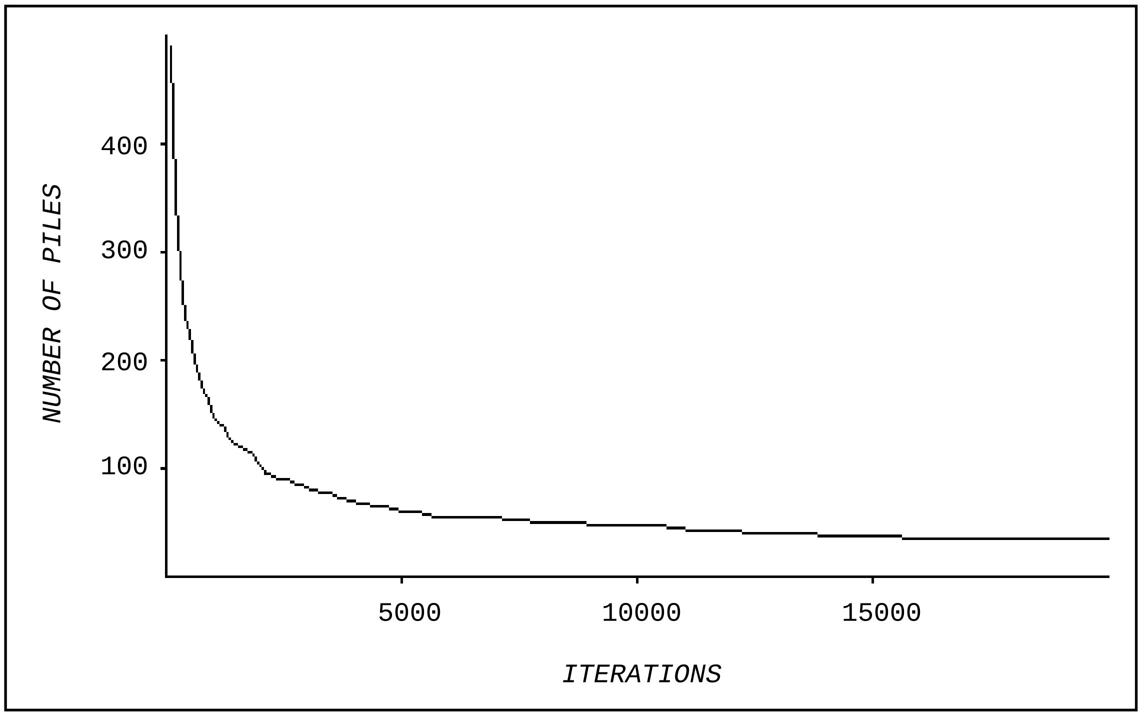

*Figure 4: The number of piles decreases monotonically*

## 5.3 Rabbits and Grass

The great baseball manager Casey Stengel once said: "If you don't know where you're going, you might end up somewhere else." My experiences with computer-based modeling activities have taught me a corollary: "Even if you think you know where you're going, you'll probably end up somewhere else."

That's what happened to Benjamin, a high-school student, when he set out to create an StarLogo program that would simulate evolution by natural selection. I had given Benjamin a *Scientific American* article (Dewdney, 1989) about a computer program called *Simulated Evolution* (Palmiter, 1989). Benjamin, who had just finished his junior year in high school, decided that he wanted to create a StarLogo program similar to the commercial program described in the article. His goal was to create a set of computer "creatures" that would interact and evolve.

At the core of Benjamin's simulation were turtles and food. His basic idea was simple: turtles that eat a lot of food reproduce, and turtles that don't eat enough food die. Eventually, he planned to add "genes" to his turtles. Different genes could provide turtles with different levels of "fitness" (perhaps different capabilities for finding food). But Benjamin never got around to the genes. Rather, on the road to evolution, Benjamin got sidetracked into an interesting exploration of ecological systems (in particular, predator-prey systems).

Benjamin began by making food grow randomly throughout the StarLogo world. (During each time step, each StarLogo patch had a random chance of growing some food.) Then he created some turtles. The turtles had very meager sensory capabilities. They could not "see" or "smell" food at a distance. They could sense food only when they bumped directly into it. So the turtles followed a very simple strategy: Wander around randomly, eating whatever food you bump into.

Benjamin gave each turtle an "energy" variable. Every time a turtle took a step, its energy decreased a bit. Every time it ate some food, its energy increased. Then Benjamin added one more rule: if a turtle's energy dipped to zero, the turtle died. With this program, the turtles do not reproduce. Life is a one-way street: turtles die, but no new turtles are born. Still, even with this simple-minded program, Benjamin found some surprising and interesting behaviors.

Benjamin ran the program with 300 turtles. But the environment could not support that many turtles. There wasn't enough food. So some turtles began to die. The turtle population fell rapidly at first, then it leveled out at about 150 turtles. The system seemed to reach a steady state with 150 turtles: the number of turtles and the density of food both remained roughly constant.

Then Benjamin tried the same program with 1000 turtles. If there wasn't enough food for 300 turtles, there certainly wouldn't be enough for 1000 turtles. So Benjamin wasn't surprised when the turtle population began to fall. But he *was* surprised with how *far* the population fell. After a while, only 28 turtles remained. Benjamin was puzzled: "We started with more, why should we end up with less?" After some discussion, he realized what had happened. With so many turtles, the food shortage was even more critical than before. The result: mass starvation. Benjamin still found the behavior a bit strange: “The turtles have less (initial energy as a group), and less usually isn’t more.”

Next, Benjamin decided to add reproduction to his model. His plan: whenever a turtle’s energy increases above a certain threshold, the turtle should “clone” itself, and split its energy with its new twin. That can be accomplished by adding another demon procedure to the program.

Benjamin assumed that the rule for cloning would somehow “balance” the rule for dying, leading to some sort of “equilibrium.” He explained: “Hopefully, it will balance itself out somehow. I mean it will. It will have to. But I don’t know what number it will balance out at.” After a little more thought, Benjamin suggested that the food supply might fall at first, but then it would rise back and become steady: “The food will go down, a lot of them will die, the food will go up, and it will balance out.”

Benjamin started the program running. As Benjamin expected, the food supply initially went down and then went up. But it didn’t “balance out” as Benjamin had predicted: it went down and up again, and again, and again. Meanwhile, the turtle population also oscillated, but out of phase with the food.

On each cycle, the turtles “overgrazed” the food supply, leading to a scarcity of food, and many of the turtles died. But then, with fewer turtles left to eat the food, the food became more dense. The few surviving turtles thus found a plentiful food supply, and each of them rapidly increased its energy. When a turtle’s energy surpassed a certain threshold, it cloned, increasing the turtle population. But as the population grew too high, food again became scarce, and the cycle started again.

Visually, the oscillations were striking. Red objects (turtles) and green objects (food) were always intermixed, but the density of each continually changed. Initially, the screen was dominated by red turtles, with a sparse scattering of green food. As the density of red objects declined, the green objects proliferated, and the screen was soon overwhelmingly green. Then the process reversed: the density of red increased, with the density of green declined.

Depending on the particular parameters, the oscillations took on different forms. In Benjamin’s initial program, the oscillations were damped: With each cycle, the peaks were a little less high, the troughs a little less deep. In the first cycle, the turtle population dwindled to just 26 turtles, then it rose to 303 turtles. In the next cycle, the population shrank to 47 turtles, then up to 244 turtles. Eventually, the turtle population stabilized between 130 and 160 turtles.

Benjamin recognized that this result depended critically on the parameters in his StarLogo program. He wondered: What would happen if the food grew just half as quickly? He figured that this new world would support fewer turtles, but how many fewer? In the original version of his StarLogo program, each patch had a 1 in 1000 chance of growing food. Benjamin changed it to 1 in 2000.

When Benjamin ran the program, he was in for another surprise: all of the turtles died. But Benjamin, who had just finished graphing the oscillations from the previous experiment, quickly realized what had happened. "The oscillation must be between some number and negative something," he said. That is: the trough of the oscillation must drop below zero. And once the population drops below zero, it can never recover. There is no peak after a negative trough. Extinction is forever: it is a "trapped state."

The problem lay in the initial conditions. Benjamin had started the simulation with 1000 turtles. If there were fewer initial turtles, the first trough wouldn't sink so deep. Benjamin came up with an ingenious solution. "I'll start with just one (turtle)," he explained. "It will definitely survive. I'll put money on it." Benjamin started the program again, this time with a single turtle. For a while, the single turtle roamed the world by itself. Benjamin cheered it on: "Come on. Hang on there. Come on. Get some food." Finally, the turtle cloned, and then there were two. "He's going to live," exclaimed Benjamin.

The turtle population rose to about 130 turtles, leveled off, then fell. As before, the turtle population went up and down in a damped oscillation. Eventually, the population stabilized at about 75 turtles. So with food growing at half the rate as before, the turtle population stabilized at about half the level as before. The "equilibrium population" seemed to be proportional to the rate of food growth.

Before running the program, Benjamin had predicted that the equilibrium population would be more drastically affected by the reduction in food growth. He expected the population to stabilize with considerably fewer than 75 turtles. But after watching the program run, he developed a explanation for the proportional relationship. Looking at the dots of food on the screen, he noted that the "food density" at equilibrium looked about the same as in the previous experiment, despite the change in the rate of food growth. That made sense to him: a certain food density is needed to keep the turtles just on the brink between death and reproduction. To reach a relatively steady state, the system needed to maintain that special food density. Given that the food was growing just half as quickly as before, it made sense that the system could support only half as many turtles.

Benjamin's reasoning is an example of what Hut and Sussman (1987) dubbed "analysis by synthesis." Traditionally, synthesis and analysis have been seen in opposition to one another, two alternate ways of solving problems. But with computer-based explorations, the two approaches get mixed and blurred. It is very unlikely that Benjamin could have developed his explanation without actually viewing (and manipulating) the simulation. Only by building and creating (synthesis) was Benjamin able to develop a well-reasoned explanation for the behavior of the turtles (analysis).

The oscillating behavior in Benjamin's project is characteristic of ecological systems with predators (in this case, turtles) and prey (in this case, food). Traditionally, scientific (and educational) explorations of predator-prey systems are based on sets of differential equations, known as the Lotka-Volterra equations (Lotka 1925; Volterra 1926). For example, the changes in the population density of the prey ($n_1$) and the population density of the predator ($n_2$) can be described with the following differential equations:

$$dn_1/dt = n_1(b - k_1n_2)$$

$$dn_2/dt = n_2(k_2n_1 - d)$$

where $b$ is the birth rate of the prey, $d$ is the death rate of the predators, and $k_1$ and $k_2$ are constants. It is straightforward to write a computer program based on the Lotka-Volterra equations, computing how the population densities of the predator and prey vary with time (e.g., Roberts 1983).

This differential-equation approach is typical of the way that scientists have traditionally modeled and studied the behaviors of all types of systems (physical, biological, and social). Scientists typically write down sets of differential equations then attempt to solve them either analytically or numerically. These approaches require advanced mathematical training; usually, they are studied only at the university level.

The StarLogo approach to modeling systems (exemplified by Benjamin's project) is sharply different. StarLogo makes systems-related ideas more accessible to younger students by providing them with a stronger personal connection to the underlying models. Traditional differential-equation approaches are "impersonal" in two ways. The first is obvious: they rely on abstract symbol manipulation (accessible only to students with advanced mathematical training). The second is more subtle: differential equations deal in aggregate quantities. In the Lotka-Volterra system, for example, the differential equations describe how the overall *populations* (not the individual creatures) evolve over time. There are now some very good computer modeling tools—such as Stella (Roberts et al., 1983) and Model-It (Jackson et al., 1996)—based on differential equations. These tools eliminate the need to manipulate symbols, focusing on more qualitative and graphical descriptions. But they still rely on aggregate quantities.

In StarLogo, by contrast, students think about the actions and interactions of individual objects or creatures. StarLogo programs describe how individual creatures (not overall populations) behave. Thinking in terms of individual creatures seems far more intuitive, particularly for the mathematically uninitiated. Students can imagine themselves as individual turtles/creatures and think about what they might do. In this way, StarLogo enables learners to "dive into" the model (Ackermann, 1996) and make use of what Papert (1980) calls "syntonic" knowledge about their bodies. By observing the dynamics at the level of the individual creatures, rather than at the aggregate level of population densities, students can more easily think about and understand the population oscillations that arise. Future versions of StarLogo will enable users to zoom in and out, making it easier for users to shift back and forth in perspective from the individual level to the group level.

I refer to StarLogo models as "true computational models" (Resnick, 1997), since StarLogo uses new computational media in a more fundamental way than most computer-based modeling tools. Whereas most tools simply implement traditional mathematical models on a computer (e.g., numerically solving traditional differential-equation representations), StarLogo provides new representations that are tailored explicitly for the computer. Of course, differential-equation models are still very useful—and superior to StarLogo-style models in some contexts. But too often, scientists and educators see traditional differential-equation models as the *only* approach to modeling. As a result, many students (particularly students alienated by traditional classroom mathematics) view modeling as a difficult or uninteresting activity. What is needed is a more pluralistic approach, recognizing that there are many different approaches to modeling, each with its own strengths and weaknesses. A major challenge is to develop a better understanding of when to use which approach, and why.

## 6. Decentralized Thinking

As students began working with StarLogo, they almost always assumed centralized causes in the patterns they observed, and they almost always imposed centralized control when they wanted to create patterns. But as students continued to work on StarLogo projects, most of them began to develop new ways of thinking about decentralization. In almost all cases, they developed an appreciation for and a fascination with decentralized systems. At one point, while we were struggling to get our termite program working, I asked Callie if we should give up on our decentralized approach and program the termites to take their wood chips to pre-designated spots. She quickly dismissed this suggestion:

> *Mitchel: We could write the program so that the termites know where the piles are. As soon as a termite picks up a wood chip, it could just go to the pile and put it down.*
>
> *Callie: Oh, that's boring!*
>
> *Mitchel: Why do you think that's boring?*
>
> *Callie: Cause you're telling them what to do.*
>
> *Mitchel: Is this more like the way it would be in the real world?*
>
> *Callie: Yeah. You would almost know what to expect if you tell them to go to a particular spot and put it down. You know that there will be three piles. Whereas here, you don't know how many mounds there are going to be. Or if the number of mounds will increase or decrease. Or things like that... This way, they [the termites] made the piles by themselves. It wasn't like they [the piles] were artificially put in.*

For Callie, pre-programmed behavior, even if effective, was "boring." Callie preferred the decentralized approach since it made the termites seem more independent ("they made the piles by themselves") and less predictable ("you don't know how many mounds there are going to be").

Over time, other students shared Callie's fascination with decentralization, though they often struggled in their efforts to use decentralized strategies in analyzing and constructing new systems. As I worked with students, I assembled a list of "guiding heuristics" that students used as they began to develop richer models of decentralized phenomena. These heuristics are not very "strong." They are not "rules" for making sense of decentralized systems. Rather, they are loose collections of ideas associated with decentralized thinking. Pedagogically, they serve as good discussion points for provoking people to think about decentralization. They also serve as a type of measuring stick for conceptual change: as students worked on StarLogo projects, they gradually began to integrate these heuristics into their own thinking and discourse. In this section, I discuss five of these guiding heuristics.

**• Positive Feedback Isn't Always Negative**

When people think about the scientific idea of positive feedback, they typically think of the screeching sound that results when a microphone is placed near a speaker. Positive feedback is viewed as a destructive force, making things spiral out of control. By contrast, negative feedback is viewed as very useful, keeping things under control. Negative feedback is symbolized by the thermostat, keeping room temperature at a desired level by turning the heater on and off as needed.

When I asked high-school students about positive feedback, most were not familiar with the term, but they were certainly familiar with the concept. When I explained what I meant by positive feedback, students quickly generated examples involving something getting out of control, often with destructive consequences. One student talked about scratching a mosquito bite, which made the bite itch even more, so she scratched it some more, which made it itch even more, and so on. Another student talked about stock-market crashes: a few people start selling, which makes more people start selling, which makes even more people start selling, and so on.

Despite these negative images, positive feedback often plays a crucial role in decentralized phenomena. Economist Brian Arthur (1990) points to the geographic distribution of cities and industries as an example of a self-organizing process driven by positive feedback. Once a small nucleus of high-technology electronics companies started in Santa Clara County south of San Francisco, an infrastructure developed to serve the needs of those companies. That infrastructure encouraged even more electronics companies to locate in Santa Clara County, which encouraged the development of an even more robust infrastructure. And thus, Silicon Valley was born.

For some students who used StarLogo, the idea of positive feedback provided a new way of looking at their world. One day, one student came to me excitedly. He had been in downtown Boston at lunch time, and he had a vision. He imagined two people walking into a deli to buy lunch.

> *Once they get their food, they don't eat it there. They bring it back with them. Other people on the street smell the sandwiches and see the deli bag, and they say, 'Hey, maybe I'll go to the deli for lunch today!" They were just walking down the street, minding their own business, and all of the sudden they want to go to the deli. As more people go to the deli, there's even more smell and more bags. So more people go to the deli. But then the deli runs out of food. There's no more smell on the street from the sandwiches. So no one else goes to the deli.*

**• Randomness Can Help Create Order**

Like positive feedback, randomness has a bad image. Most people see randomness as annoying at best, destructive at worst. They view randomness in opposition to order: randomness undoes order, it makes things disorderly.

In fact, randomness plays an important role in creating order in many self-organizing systems. People often assume that "seeds" are needed to initiate patterns and structures. In general, this is a useful intuition. The problem is that most people have too narrow a conception of "seeds." They think only of preexisting inhomogeneities in the environment—like a broken bridge on the highway, or a piece of food in an ant's world.

This narrow view of seeds causes misintuitions. In self-organizing systems, seeds are neither preexisting nor externally imposed. Rather, self-organizing systems often create *their own* seeds. It is here that randomness plays a crucial role. Random fluctuations act as the "seeds" from which patterns and structures grow. Randomness creates the initial seeds, then positive feedback makes the seeds grow. For example, the differing velocities of cars on a highway create the seeds from which traffic jams can grow.

**• A Flock Isn't a Big Bird**

In trying to make sense of decentralized systems and self-organizing phenomena, the idea of *levels* is critically important. Interactions among objects at one level give rise to new types of objects at another level. Interactions among slime-mold cells give rise to slime-mold clusters. Interactions among cars give rise to traffic jams. Interactions among birds give rise to flocks.

In many cases, the objects on one level behave very differently than objects on another level. For some high-school students, these differences in behavior were very surprising (at least initially). For example, the students working on the StarLogo traffic project were shocked by the behavior of the traffic jams: the jams moved backwards even though all of the cars within the jams were moving forward.

Confusion of levels is not restricted to scientifically naive high-school students. I showed the StarLogo traffic program to two visiting computer scientists. They were not at all surprised that the traffic jams were moving backwards. They were well aware of that phenomenon. But then one of the researchers said: "You know, I've heard that's why there are so many accidents on the freeways in Los Angeles. The traffic jams are moving backwards and the cars are rushing forward, so there are lots of accidents." The other researcher thought for a moment, then replied: "Wait a minute. Cars crash into other cars, not into traffic jams." In short, he believed that the first researcher had confused levels, mixing cars and jams inappropriately. The two researchers then spent half an hour trying to sort out the problem. It is an indication of the underdeveloped state of decentralized thinking in our culture that two sophisticated computer scientists needed to spend half an hour trying to understand the behavior of a ten-line decentralized computer program written by a high-school student.

## • A Traffic Jam Isn't Just a Collection of Cars

For most everyday objects, it is fair to think of the object as a collection of particular parts: a chair has four particular legs, a particular seat, and so on. But not so with objects like traffic jams. Thinking of a traffic jam as a collection of particular parts leads to confusion. The cars composing a traffic jam are always changing, as some cars leave the front of the jam and other join from behind. Even when all of the cars in the jam are replaced with new cars, it is still the same traffic jam. A traffic jam can be thought of as an "emergent object"—it emerges from the interactions among lower-level objects (in this case, cars).

As students worked on StarLogo projects, they encountered many emergent objects. In the termite example, the wood-chip piles can be viewed as emergent objects. The precise composition of the piles is always changing, as termites take away some wood chips and add other wood chips. After a while, none of the original wood chips remains, but the pile is still there.

## • The Hills are Alive

In *Sciences of the Artificial* (1969), Herbert Simon describes a scene in which an ant is walking on a beach. Simon notes that the ant's path might be quite complex. But the complexity of the path, says Simon, is not necessarily a reflection of the complexity of the ant. Rather, it might reflect the complexity of the beach. Simon's point: don't underestimate the role of the environment in influencing and constraining behavior. People often think of the environment as something to be *acted upon*, not something to be *interacted with*. People tend to focus on the behaviors of individual objects, ignoring the environment that surrounds (and interacts with) the objects.

A richer view of the environment is important in thinking about decentralized and self-organizing systems. In designing StarLogo, I explicitly tried to highlight the environment. Most creature-oriented programming environments treat the environment as a passive entity, manipulated by the creature that move within it. In StarLogo, by contrast, the "patches" of the world have equal status with the creatures that move in the world. By reifying the environment, I hoped to encourage people to think about the environment in new ways.

Initially, some students resisted the idea of an active environment. When I explained a StarLogo ant-foraging program to one student, he was worried that pheromone trails would continue to attract ants even after the food sources at the ends of the trails had been fully depleted. He developed an elaborate scheme in which the ants, after collecting all of the food, deposited a second pheromone to neutralize the first pheromone. It never occurred to him to let the first pheromone evaporate away. In his mind, the ants had to take some positive action to get rid of the first pheromone. They could not rely on the environment to make the first pheromone go away.

## 7. New Media, New Mindsets

There is an old saying that goes something like this: "Give a person a hammer, and the whole world looks like a nail." Indeed, the ways we see the world are deeply influenced by the tools and media at our disposal. If we are given new tools and media, not only can we accomplish new tasks, but we begin to view the world in new ways.

Often, we hardly recognize how our tools and media are influencing our ways of viewing the world. For several centuries now, scientists have described the world in terms of differential equations. Is that because differential equations are the best way to represent and describe the world? Or is it because the common media of the era (paper and pencil) are well suited to manipulations of differential equations? Could we say: "Give a scientist paper and pencil, and the whole world looks like differential equations"?

New computational media now hold the promise for radically reshaping how people model (and think about) the world. But this shift won't happen automatically. Computer modeling will make bring profound change to the classroom only if modeling tools take full advantage of new computational representations. Just as sculptors need to understand the qualities of clay (or whatever material they are using), designers of computer-modeling tools need to understand their chosen medium. StarLogo, for example, leverages two new computational paradigms—massive parallelism and object-oriented programming. These new paradigms offer new design possibilities: new ways to create decentralized models. But even more importantly, these new paradigms offer new epistemological possibilities: a new decentralized framework for making sense of many phenomena in the world.

Adding new tools to the carpenter's toolkit changes the way the carpenter looks at the world. So, too, with computational ideas and paradigms. That is the central challenge for computer-modeling activities in education: not only to help students create models in new ways, but also to help students develop fundamentally new ways of thinking about the systems and phenomena that they are modeling.

## Acknowledgments

Brian Silverman, Andy Begel, Hal Abelson, Seymour Papert, Uri Wilensky, and Larry Latour have provided inspiration and ideas for the StarLogo project. The LEGO Group and the National Science Foundation (Grants 9153719-MDR, 9358519-RED, 9553474-RED) have provided generous financial support. Portions of this chapter were previously published elsewhere (Resnick, 1994, 1996, 1997), and are reprinted with permission.

## References

Abelson, H., & diSessa, A. (1980). *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. Cambridge, MA: MIT Press.

Ackermann, E. (1996). Perspective-Taking and Object Construction: Two Keys to Learning. In Y. Kafai & M. Resnick (eds.), *Constructionism in Practice* (pp. 25-35). Mahwah, NJ: Lawrence Erlbaum.

Arthur, W.B. (1990). Positive Feedbacks in the Economy. *Scientific American*, *262* (2), 92-99.

Dewdney, A.K. (1989). Simulated Evolution: Wherein bugs learn to hunt bacteria. *Scientific American*, *260* (5), 138-141.

Harvey, B. (1985). *Computer Science Logo Style*. Cambridge, MA: MIT Press.

Heppner, F., and Grenander, U. (1990). "A Stochastic Nonlinear Model for Coordinated Bird Flocks." In S. Krasner (Ed.), *The Ubiquity of Chaos*. Washington, D.C.: AAAS Publications.

Herman, R., & Gardels, K. (1963). Vehicular Traffic Flow. *Scientific American*, *209* (6), 35-43.

Hut, P., and Sussman, G.J. (1987). Advanced Computing for Science. *Scientific American*, *255* (10).

Jackson, S., Stratford, S., Krajcik, J., & Soloway, E. (1996). A Learner-Centered Tool for Students Building Models. *Communications of the ACM*, *39* (4), 48-49.

Keller, E.F. (1985). *Reflections on Gender and Science*. New Haven: Yale University Press.

Keller, E.F., and Segel, L. (1970). "Initiation of Slime Mold Aggregation Viewed as an Instability." *Journal of Theoretical Biology*, *26*, 399-415.

Leiser, D. (1983). Children's Conceptions of Economics—The Constitution of a Cognitive Domain. *Journal of Economic Psychology*, *4*, 297-317.

Lotka, A.J. (1925). *Elements of Physical Biology*. New York: Dover Publications (reprinted 1956).

Mayr, E. (1982). *The Growth of Biological Thought*. Cambridge, MA: Harvard University Press.

Minsky, M. (1987). *The Society of Mind*. New York: Simon & Schuster.

Paley, W. (1802). *Natural Theology—or Evidences of the Existence and Attributes of the Deity Collected from the Appearances of Nature*. Oxford: J. Vincent.

Palmiter, M. (1989). *Simulated Evolution*. Bayport, NY: Life Science Associates.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books.

Papert, S. (1991). Situating Constructionism. In I. Harel & S. Papert (Eds.), *Constructionism*. Norwood, NJ: Ablex Publishing.

Resnick, M. (1990). MultiLogo: A Study of Children and Concurrent Programming. *Interactive Learning Environments*, *1* (3), 153-170.

Resnick, M. (1991). Animal Simulations with StarLogo: Massive Parallelism for the Masses. In J.A. Meyer & S. Wilson (Eds.), *From Animals to Animats*. Cambridge, MA: MIT Press.

Resnick, M. (1994). *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. Cambridge, MA: MIT Press.

Resnick, M. (1996). Beyond the Centralized Mindset. *Journal of the Learning Sciences*, *5* (1), 1-22.

Resnick, M. (1997). Learning Through Computational Modeling. *Computers in the Schools*.

Resnick, M., Bruckman, A., & Martin, F. (1996). Pianos Not Stereos: Creating Computational Constructions Kits. *Interactions*.

Reynolds, C. (1987). "Flocks, Herds, and Schools: A Distributed Behavioral Model." *Computer Graphics*, *21* (4), 25-36.

Roberts, N., Anderson, D., Deal, R., Garet, M., & Shaffer, W. (1983). *Introduction to Computer Simulation: A System Dynamics Modeling Approach*. Reading, MA: Addison-Wesley.

Simon, H. (1969). *The Sciences of the Artificial*. Cambridge, MA: MIT Press.

Toffoli, T., & Margolus, N. (1987). *Cellular Automata Machines*. Cambridge, MA: MIT Press.

Volterra, V. (1926). Fluctuations in the Abundance of a Species Considered Mathematically. *Nature*, *188*, 558-560.

Wilensky, U. (1993). *Connected Mathematics: Building Concrete Relationships with Mathematical Knowledge*. Doctoral dissertation. Cambridge, MA: MIT Media Lab.