NEW PARADIGMS FOR COMPUTING, NEW PARADIGMS FOR THINKING

MITCHEL RESNICK

Epistemology and Learning Group
Learning and Common Sense Section
The Media Laboratory
Massachusetts Institute of Technology
20 Ames Street Room E15–312
Cambridge, MA 02139
mres@media.mit.edu

ABSTRACT

New computational paradigms (such as object-oriented programming, constraints, logic, and parallelism) can significantly influence not only what people do with computers, but also how they think about and make sense of the world. This paper examines how one particular computational paradigm—parallelism—can support people in new types of mathematical explorations and also help them gain new insights into the workings of real-world, decentralized systems like bird flocks and traffic jams. The paper describes a new parallel programming environment called StarLogo, and it provides examples of how StarLogo activities can suggest and encourage new ways of thinking.

This paper was prepared for the NATO Advanced Workshop on "The Design of Computational Media to Support Exploratory Learning" (diSessa, Hoyles, Noss, & Edwards, in press).

In Y. Kafai and M. Resnick (Eds.), *Constructionism in Practice: Rethinking Learning and Its Contexts.* Presented at the National Educational Computing Conference, Boston, MA, June 1994. The MIT Media Laboratory, Cambridge, MA.

1. Introduction

Educators are increasingly interested in providing students with computational tools that support exploration and experimentation. But designing such tools (and designing contexts for using such tools) is easier said than done. Doing it well requires intertwining many different threads of thought.

One thread involves an understanding of the learner: What are the learner's preconceptions and expectations? How will the learner integrate new experiences into existing frameworks? In what ways can learners construct new concepts and new meanings—and in what ways can new computational media provide scaffolding to support this process?

A second thread in the design fabric involves an understanding of domain knowledge. If a new computational tool or activity is intended to help students learn about a particular area of mathematics or science, the designer had better know something about that area of mathematics or science. But there is a deeper point. The best computational tools don't simply offer the same content in new clothing; rather, they aim to recast areas of knowledge, suggesting fundamentally new ways of thinking about the concepts in that domain, allowing learners to explore concepts that were previously inaccessible. A classic example is Logo's turtle geometry, which opens up new ways of thinking about geometry and makes possible new types of geometry explorations (Papert 1980; Abelson & diSessa 1980). The design of such tools requires a deep understanding of a particular domain.

A third thread involves an understanding of computational ideas and paradigms. Just as sculptors need to understand the qualities of clay (or whatever material they are using), designers of computational tools need to understand their chosen medium. The point is not simply for computational designers to make nicely crafted artifacts (though that, of course, is important). Computation is not just a medium for designers; it is a set of powerful ideas for students to learn and explore. Well-designed computational tools and activities can provide students with new ways of thinking about computational ideas. Again, Logo provides a classic example. Since Logo is a procedural programming language, it introduces students to ideas about procedural abstraction—ideas that are important not just in the world of computer science, but in many other disciplines (Harvey, 1985).

In this paper, I will focus especially on this third thread (though recognizing that it must be intertwined with the first two). In recent years, computer scientists have developed and experimented with a wide range of new programming paradigms: object orientation, logic, constraints, parallelism. Each of these paradigms offers new design possibilities, new ways to create things with computers. But each also offers new epistemological possibilities, new ways to think about computation and other phenomena in the world. An old saying goes something like this: If a person has only a hammer, the whole world looks like a nail. Adding new tools to the carpenter's toolkit changes the way the carpenter looks at the world. So, too, with computational paradigms: new paradigms can change the way computer users think about the world.

Here, I focus particularly on one new computational paradigm: parallelism. I will describe a new parallel programming language, called StarLogo, that I designed explicitly as an environment for exploratory learning. I will discuss how StarLogo can expand the

space of design possibilities for students while also offering students new ways of looking at the world.

2. Parallelism

During the past two decades, there has been ever-increasing interest in parallel computation and parallel programming languages. Some researchers have added "parallel constructs" to existing programming languages, yielding new language dialects like Concurrent Pascal (Brinch Hansen, 1975) and MultiLisp (Halstead, 1985). Other researchers have created entirely new programming models, designed explicitly with parallelism in mind (e.g., Sabot, 1988).

In most of this research, the primary goal is to improve the speed of computation. A recent article in a major computer-science journal quotes a user saying: "Nobody wants parallelism. What we want is performance" (Pancake, 1991). In other words, many people see parallelism as a "necessary evil" in order to improve the speed at which programs execute. If they could, many language developers would hide parallelism from the user. Indeed, some researchers have developed "parallelizing compilers," which allow programmers to continue writing programs in traditional sequential style, putting the burden on the compiler to "parallelize" the code to improve performance.

In adding parallelism to Logo, I had a very different set of goals. I was not particularly concerned with performance or speed. Rather, I was interested in providing new ways for programmers to model, control, and think about actions that actually happen in parallel. Many things in the natural world (such as ants in a colony) and the manufactured world (such as rides in an amusement park) really do act in parallel. The most natural way to model and control such situations is with a parallel programming language. In these cases, parallelism isn't a "trick" to improve performance; it is the most natural way of expressing the desired behavior.

My interest in parallelism was motivated, in part, by my research on LEGO/Logo, a computer-controlled construction kit for children (Resnick, Ocko, & Papert 1988; Resnick 1993). When children build and program LEGO machines, they often want different machines to run different programs at the same time. For example, after building an amusement park with a LEGO Ferris wheel and a LEGO merry-go-round, a child might want the two rides to run different programs at the same time. That is a very natural thing to want. In fact, many children are surprised that traditional versions of Logo can't do such a simple thing. Running simultaneous programs is a simple thing to think about, shouldn't it be a simple thing to do in Logo?

To address this problem, I developed a parallel version of Logo called MultiLogo (Resnick 1990). To enable users to create and execute multiple processes at the same time, MultiLogo adds one new programming construct: the "agent." Each agent is like a separate version of Logo—that is, each agent can control a computational process. By using multiple agents, users can control multiple processes. In this way, MultiLogo users can control the simultaneous actions of multiple LEGO machines in the world, or multiple Logo turtles on the screen. (A commercial version of Logo, called Microworlds, developed by Logo Computer Systems Inc., now offers many of the same capabilities as MultiLogo.)

3. Beyond the Centralized Mindset

MultiLogo works well for situations with a few objects (LEGO machines or graphic turtles) acting in parallel. But what if you want to simulate a large flock of birds? Or a highway full of cars at rush hour? In these cases, you need hundreds (or even thousands) of objects acting in parallel and interacting with one another. For situations like these, I created a new version of Logo, called StarLogo (Resnick 1994).

One reason for my interest in these "massively parallel" situations is that people seem to have great difficulty thinking about and understanding such situations. When people see patterns in the world, they tend to assume some type of centralized control. For example, when people see a flock of birds, they typically assume that the bird in the front is leading and the others are following. But that's not the case. 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. Orderly flock patterns arise from these simple, local interactions. The flock is organized without an organizer, coordinated without a coordinator (Heppner & Grenander, 1990).

Many other systems work the same way, with patterns determined not by some central authority but by local interactions among decentralized components. As ants forage for food, their trail patterns are determined not by the dictates of the queen ant, but by local interactions among thousands of worker ants. Macroeconomic patterns arise from local interactions among millions of buyers and sellers. In immune systems, armies of antibodies seek out bacteria in a systematic, coordinated attack—without any "generals" organizing the overall battle plan.

StarLogo is intended to help people model such systems—and, in the process, develop new ways of thinking about such systems. Too often, when people observe systems in the world, they assume centralized control where it doesn't exist. The continuing resistance to evolutionary theories is an example: many people still insist that someone or something must have explicitly designed the orderly structures that we call Life. Similarly, when people construct new systems (whether it is managers creating new organizational structures or engineers creating new technological structures), they often impose centralized control when it is not needed or is counter-productive.

The thesis underlying StarLogo is that people, by designing and playing with decentralized systems in StarLogo, will move beyond this centralized mindset, developing new ways of thinking about and understanding systems. Toward that end, StarLogo extends Logo in three major ways.

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. For many colony-type explorations, having hundreds 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 turtles 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. There is even a built-in primitive to make turtles follow

the gradient of a scent—that is, to make turtles turn in the direction where the scent is strongest. Such turtle-turtle and turtle-world interactions are essential for creating and experimenting with self-organizing phenomena. Parallelism alone is not enough. If each turtle just acts on its own, without any interactions, interesting colony-level behaviors will never 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 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 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.

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 (such as procedural abstraction and recursion) accessible to a larger number of users. Similarly, StarLogo aims to make 1990's ideas from computer science (such 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), StarLogo introduces another new object (the patch) to facilitate explorations of other mathematical/scientific ideas (such as self-organization).

The next three sections present three examples of StarLogo explorations. These examples aim to illustrate how new approaches to programming can encourage and facilitate new approaches to thinking.

4. New Turtle Geometry

While I was designing StarLogo, my primary goal was to develop a language for exploring biological phenomena, like ant-colony foraging. But once StarLogo was up and running, I stumbled upon some unexpected ways to use the new abundance of turtles.

At one point, I created 5000 turtles, then typed the command `setxy 0 0`, making all of the turtles move to the middle of the screen, to the Cartesian point (0,0). Only a single turtle was visible on the screen. But in fact, that single turtle was at the top of a very tall "pile" of turtles, with 4999 turtles underneath it—somewhat like the pile of turtles in Dr. Seuss's *Yertle the Turtle.*

The pile of turtles seemed like a neat trick. Then, suddenly, I realized an even better trick. I typed the command `forward 50`. The 5000 turtles exploded outward from the center of the screen. Since the turtles had random headings, they all moved in random directions. But their overall pattern was anything but random. After each turtle had moved forward five steps, they formed a circle of radius 5, centered on the point (0,0). After each turtle had moved another five steps, they formed a circle of radius 10. So the overall effect was an expanding circle. The circle grew until it reached a radius of 50.

This approach to drawing a circle represents a new form of "turtle geometry." In traditional turtle geometry, the Logo turtle uses a "pen" to draw various geometric shapes and patterns. For example, the command `repeat 360 [forward 1 right 1]` makes the Logo turtle draw a circle. The turtle takes a step forward, then turns a degree to the right, then another step forward, and so on. After 360 steps, the turtle returns to its starting point, having completed a circle. (Actually, it draws a regular polygon with 360 sides. But if you increase the number of steps, and decrease the turning angle, the polygon gets closer and closer to a circle, approaching a circle as its limit.) The StarLogo form of turtle geometry is quite different. Rather than a single turtle drawing geometric shapes and patterns, a collection of turtles use their own "bodies" to form geometric shapes and patterns. Rather than turtles *drawing* circles, the turtles *are* the circle.

The StarLogo approach works only if there are a large number of turtles. If there were only 50 turtles (instead of 5000), the "circle of turtles" would have many "holes" in it. All of the turtles would lie on the same circle, but they wouldn't appear as a "complete" circle. To give the appearance of a complete circle, turtles must be distributed around the entire circumference of the circle. The approach with 5000 turtles relies on the statistical properties of a random distribution. Each of the 5000 turtles has a random heading. There is a chance, of course, that all 5000 turtles could have headings between 0 and 90, so they would form only a quarter-circle. But statistically, the 5000 turtles are almost certain to arrange themselves around the entire circumference (with gaps that are no more than a few thousandths of the circumference).

In this way, StarLogo serves as a type of microworld for exploring ideas about statistics and randomness (Wilensky, 1993). Indeed, the circle example forced some high-school students to rethink their notions of randomness. I asked some students to predict what would happen when 5000 turtles (all starting at the center, with random headings) moved forward 50 steps, and one student responded: "Each turtle has a random heading, so they'll go all over the place." In his mind, randomness was clearly associated with disorder ("all over the place"). Even after seeing the turtles move outward in an expanding circle, one of the students remained bothered: "If the turtles have random headings, why are they always forming a circle?"

This "expanding-from-the-center" strategy is just one way to create a circle in StarLogo. An alternative approach (connected to a different set of mathematical ideas) is the "constraint-plus-noise" strategy. In this approach, you create several dozen turtles, and you assign two "buddies" to each turtle. For example, turtle number 14 could have turtle 13 and turtle 15 as its two buddies. You then write a program so that each turtle tries to keep a "desired distance" away from each of its two buddies. If it gets too close to one of its buddies, it moves away a bit. If it gets too far from one of its buddies, it moves towards it a bit. The StarLogo code looks like this:

```
to constraint-rule
setheading toward :buddy1
forward 0.1 * ((distance :buddy1) - :desired-distance)
setheading toward :buddy2
forward 0.1 * ((distance :buddy2) - :desired-distance)
end
```

The turtles start scattered randomly on the screen (Fig. 1a). Then, using the parallelism of StarLogo, all of the turtles run the `constraint-rule` at the same time.

The turtles quickly settle into a stable state, with each turtle the same distance from each of its two buddies (Fig. 1b). The collection of turtles looks like a jumbled mess, with no clear pattern. But then you add one more rule (run in parallel with the `constraint-rule`): Each turtle should try to move away from all of the other turtles (not just its buddies).

```
to noise-rule
seth toward pick-random-turtle
back 1
end
```

Once this new rule is added, the turtles begin to push away from one another. The overall movement has a distinctly organic feel, as if the group of turtles is trying to spring to life. At first, the turtles seem to be tied in knots. But they gradually push out the kinks and knots, expanding to a full circle (Fig. 1c-1f).

This constraint-plus-noise approach is a general-purpose strategy that can be used in many problem-solving and design situations. But most people are unfamiliar with this strategy and this way of thinking; they have never had the appropriate tools to try it out and play with it.

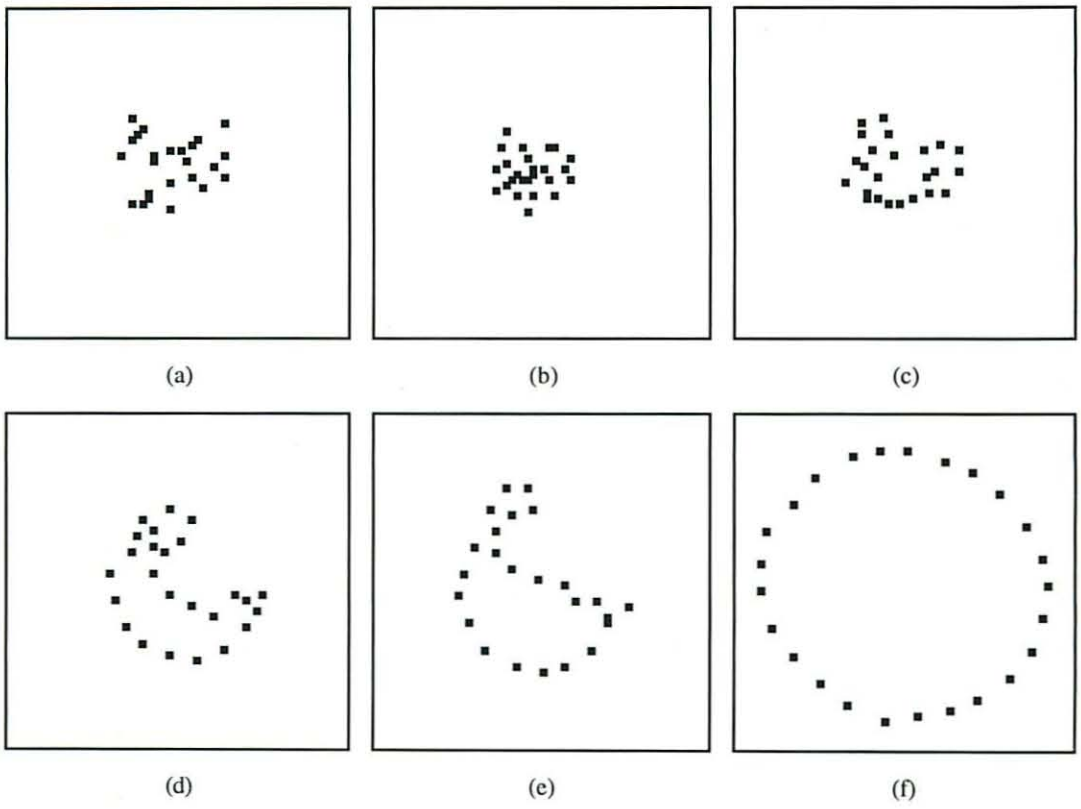

Figure 1: When the constraint-rule is turned on, the turtles keep a uniform distance from their "buddies" (b). When the noise-rule is turned on, the turtles expand into a circle (c-f).

Are the StarLogo approaches to turtle geometry better than traditional turtle geometry? That is the wrong question to ask. The point is not to provide *better* ways of doing geometry, but to provide *more* ways of doing and thinking about geometry. Each way of thinking about something strengthens and deepens each of the other ways of thinking about it. Understanding something in several different ways produces an overall understanding that is richer and of a different nature than any one way of understanding. Thus, the new StarLogo turtle geometry has the potential to supplement and reinforce other ways of thinking about geometry.

5. Rugby Turtles

At a recent workshop on computers and learning, the workshop organizers proposed three mathematical "challenge problems." Many different computational tools (Microworlds Logo, Boxer, StarLogo, Cabri, etc.) were available for use. The idea was to see whether different people approached these problems differently. In particular, would the choice of computational tools significantly affect the problem-solving strategies?

Workshop participants worked on the problems in the evenings (sometimes late into the night!). One of the problems captured my interest. It came from a British math textbook and involved the game of rugby:

> After a try has been scored, the scoring team has the opportunity to gain further points by "kicking a conversion." The kick can be taken from anywhere on an imaginary line that is perpendicular to the try line and goes through the point that the try was scored [Fig. 2]. Where should the kick be taken from to maximize the angle AKB?

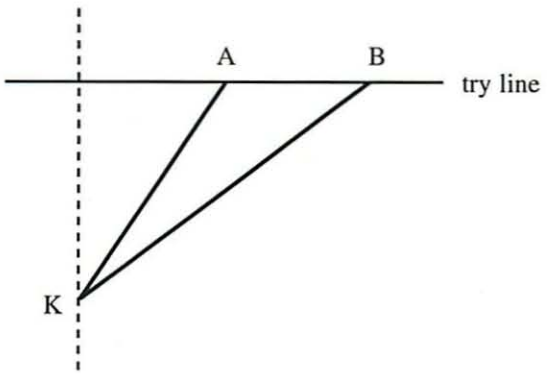

Figure 2

People at the workshop approached this problem in many different ways. Some started with paper and pencil—sketching diagrams, writing equations, taking derivatives. Other participants went right to the computer. Perhaps the most common response was to use one of the "new geometry construction kits" such as Cabri.

As soon as I read the problem, a very different approach sprang to my StarLogo-biased mind. I immediately recognized a massively parallel approach to the problem. Uri Wilensky was also interested in this approach, and we decided to work on the problem together using StarLogo. Rather than seeing the problem as a geometry problem, we viewed it as a probability and statistics problem.

We imagined hundreds of rugby players standing at different points along the line perpendicular to the try line, and we imagined each player kicking hundreds of balls in random directions. To find the point K that maximizes the angle AKB, we simply needed to figure out which of my hundreds of rugby players scored the most conversions (by successfully kicking balls between the goalposts A and B). The reason is clear: if each player is kicking balls in random directions, the player with the largest AKB angle will score the most conversions. This strategy is an example of what is sometimes called a Monte Carlo approach.

It was quite easy to write the StarLogo program to implement this strategy. We used turtles to represent the rugby balls. To start, the program put thousands of turtles/balls with random headings at random positions along the perpendicular line. Then, the program "kicked" all of these turtles/balls, moving them forward in straight lines. Finally, the program took all of the turtles that successfully went through the goalposts and moved them back to their starting points on the perpendicular line. The point with the most surviving turtles is the point that maximizes the angle AKB.

This example illustrates how new computational tools can suggest and make possible new problem-solving approaches to traditional problems. This approach even has some advantages over more traditional strategies. If extra constraints were added to the problem, such as a wind blowing across the field, or a limitation on the distance a rugby player can kick a ball, it would be quite easy to adjust the StarLogo program to take the new constraints into account. It would be more difficult to adjust traditional geometric analyses.

6. 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. So when I gave Ari and Fadhil a collection of articles to read, it is not surprising that a *Scientific American* article titled "Vehicular Traffic Flow" (Herman & Gardels, 1963) captured their attention.

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 cars would know to slow down when they approached the radar trap.

After some discussion, Ari and Fadhil programmed each driver to follow three simple rules: (1) If you see another car close ahead of you, slow down. (2) If there are no

cars close ahead of you, speed up (unless you are already at the speed limit). (3) If you detect a radar trap, slow down.

Both students expected that a traffic jam would form behind the radar trap, and indeed it did. As the cars slowed down for the trap, the cars behind them were forced to slow down, as so on, creating a queue with roughly equal distances between the cars. When the cars moved beyond the trap, they accelerated smoothly until they reached the speed limit.

I asked the students what would happen if they ran the same program without the radar trap. The cars would be controlled by just two rules: if you see another car close ahead, slow down; if not, speed up. They predicted that the traffic flow would become uniform; cars would be evenly spaced, traveling at a constant speed. After all, without the radar trap, what could cause a jam? When we ran the program, however, a traffic jam formed. Along parts of the road, the cars were tightly packed and moving slowly. Elsewhere, they were spread out and moving at the speed limit (Fig. 3).

At first, the students were shocked. Their comments revealed the workings of a centralized mindset: They argued that traffic jams need some sort of centralized "seed" (like a radar trap or accident) in order to form. They couldn't believe that simple interactions among cars could create a jam. But that's what was happening in their simulation. If a few cars, by chance, happened to get near one another, they slowed down. This, in turn, made it likely that even more cars behind them would have to slow down, leading to a jam. By continuing to modify and play with their StarLogo program (adjusting initial speeds and positions, changing acceleration rates), Ari and Fadhil eventually developed better intuitions about traffic jams, recognizing how decentralized interactions can indeed cause the formation of larger-scale traffic patterns.

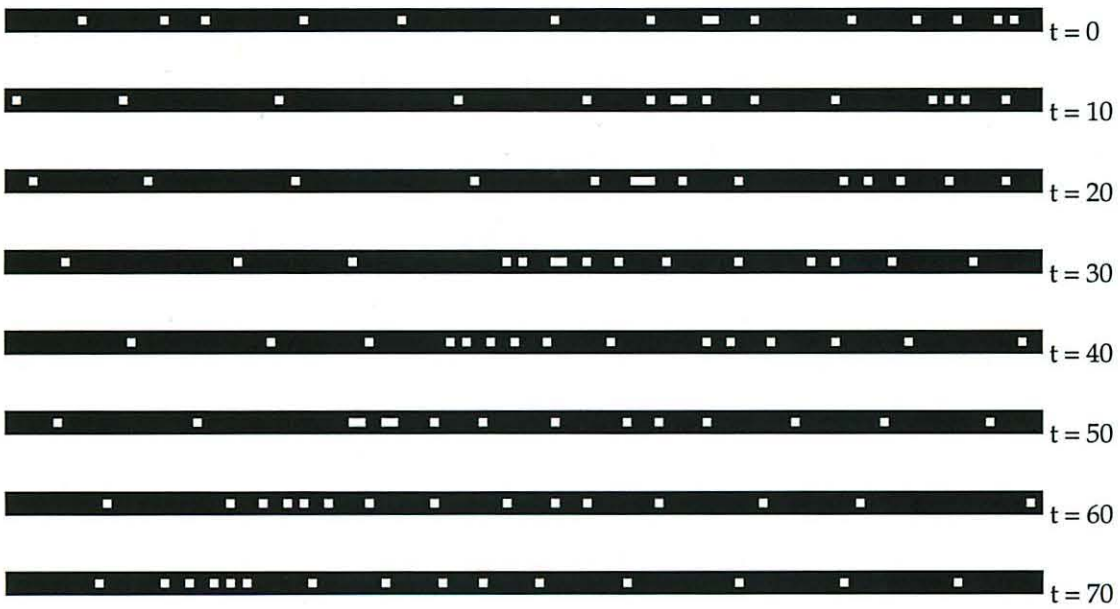

Figure 3: Traffic jam without radar trap
(Cars move left to right, but jam moves right to left)

7. New Ways of Thinking

The centralized mindset is not just a misconception of scientifically naive high-school students. It seems to affect the thinking of nearly everyone. For many years, even scientists assumed that bird flocks must have leaders. It is only recently that scientists have revised their theories, asserting that bird flocks are leader-less and self-organized. A similar bias toward centralized theories can be seen throughout the history of science, with scientists remaining committed to centralized explanations, even in the face of discrediting evidence.

The history of research on slime-mold cells, as told by Evelyn Fox Keller (1985), provides a striking example of centralized thinking. At certain stages of their life cycle, slime-mold cells 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 a colleague proposed an alternative model (Keller & Segel, 1970), showing how slime-mold clusters can form if every individual cell follows the same set of simple rules, involving the emission and sensing of chemicals. Nevertheless, for the following decade, other researchers continued to assume that special pacemaker cells were required to initiate the aggregation process. As Keller 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.

For many years, there has been a self-reinforcing spiral. People saw the world in centralized ways, so they constructed centralized tools and models, which further encouraged a centralized view of the world. Until recently, there was little pressure against this centralization spiral. Even if someone wanted to experiment with decentralized approaches there were few tools or opportunities to do so.

The centralization spiral is now starting to unwind. New computational tools based on the paradigm of massive parallelism are supporting and encouraging new ways of thinking. In some cases (as seen in the turtles-in-a-circle and rugby examples), these new tools can encourage new approaches to mathematical problems and new ways of conceptualizing mathematical ideas. In other cases (such as the traffic example), the tools can support explorations into the workings of real-world systems. Overall, these new tools provide an opportunity for students (and others) to move beyond the centralized mindset, suggesting an expanded set of models and metaphors for making sense of the world.

Acknowledgments

Hal Abelson, Seymour Papert, Brian Silverman, Randy Sargent, Uri Wilensky, Ryan Evans and Andy Begel have provided encouragement, inspiration, and ideas for the StarLogo project. Thanks to Andy diSessa, Celia Hoyles, Richard Noss, and Laurie Edwards for organizing the NATO workshop for which I wrote this paper, and thanks to Andy diSessa, Yasmin Kafai, and Greg Kimberly for comments on a draft of this

paper. The LEGO Group, the National Science Foundation (Grants 9153719-MDR, 8751190-MDR, and RED-9358519), and Nintendo Co. have provided financial support for this research.

References

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

Brinch Hansen, P. (1975). The Programming Language Concurrent Pascal. *IEEE Trans. Software Eng.*, 1 (2): 199-207.

diSessa, A., Holyes, C., Noss, R., with Edwards, L., eds. (in press). *The Design of Computational Media to Support Exploratory Learning*. Heidelberg: Springer Verlag.

Halstead, R. (1985). Multilisp: A Language for Concurrent Symbolic Computation. *ACM Trans. of Prog. Languages and Systems*, 7 (4): 501-538.

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

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

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

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

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

Pancake, C. (1991). Software Support for Parallel Computing: Where Are We Headed? *Communications of the ACM*, 34 (11): 53-64.

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

Resnick, M., Ocko, S., & Papert, S. (1988). LEGO, Logo, and Design. *Children's Environments Quarterly*, 5 (4): 14-18.

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

Resnick, M. (1993). Behavior Construction Kits. *Communications of the ACM*, 36 (7): 64-71.

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

Sabot, G. (1988). *The Paralation Model*. Cambridge, MA: MIT Press.

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