LEGO/Logo
A Learning Environment for Design

Stephen Ocko, Seymour Papert, and Mitchel Resnick
Media Laboratory
Massachusetts Institute of Technology

1. Introduction

Elementary-school science education is a largely unsuccessful enterprise. Students bring a wide range of misconceptions to the science classroom — and they tend to leave with their misconceptions intact (Driver et al. 1985; West and Pines 1985). Moreover, students do not gain much understanding of the nature of the scientific process; even when they conduct experiments in the classroom, few students seem to understand the purpose of experimentation (Osborne and Freyberg 1985). This is hardly surprising since science is often presented, in the words of a recent government report (Bennett 1986), as "a grab bag of esoteric facts and stunts."

In this paper, we describe a computer-based system, called LEGO/Logo, that offers a new approach to elementary science education. LEGO/Logo places engineering and design activities at the center of the science curriculum. Using the system, students build machines out of LEGO building pieces (including gears, motors, and sensors), connect the machines to a computer, then write computer programs to control the machines. These activities can provide a more meaningful and motivating context for learning traditional science-curriculum concepts, while also introducing elementary-school students to important engineering and design concepts that are rarely addressed in today's curricula.

The LEGO/Logo system and the activities that surround it are based on an educational philosophy that we call *constructionism* (Papert 1986). This philosophy takes as one of its central tenets the idea that students learn best when they are creating something that they believe in and care about. Like most "hands-on" approaches to science, the constructionist approach aims to make abstract ideas concrete. But constructionism goes beyond traditional hands-on science. In many hands-on lessons, students re-create *someone else's experiment.* Students are told what to measure and, in many cases, what the answer should be. Even in more open-ended experiments, students rarely have any deep sense of involvement or interest in the activity.

LEGO/Logo projects have a very different flavor. LEGO/Logo links science to a world that students are already familiar with and care about. Students *want* to build and control LEGO machines. LEGO/Logo projects feel like "real" projects, not experiments cooked up for classroom consumption. Moreover, important scientific concepts are "close to the surface" in LEGO/Logo activities. Thus, LEGO/Logo projects can act as a motivating context for learning about certain scientific principles (friction, mechanical advantage, etc.), and for learning about the scientific process itself.

Equally important, LEGO/Logo opens up new areas that are not traditionally addressed in pre-college science curricula. In particular, LEGO/Logo provides a strong foundation for the study of design and engineering, or what Herbert Simon calls "the sciences of the artificial" (Simon 1969). LEGO/Logo provides an environment in which students can explore engineering ideas like feedback and modularity. And while working on LEGO/Logo projects, students can develop a systematic approach to design and invention.

The LEGO/Logo environment is particularly powerful in that it combines design activities in both the LEGO domain and the Logo domain. Logo itself provides a good environment for learning about design ideas like modularity and abstraction (Papert 1980; Harvey 1986). With LEGO/Logo, students can experiment with the same ideas in *two* domains. Thus, they are more likely to recognize that there *are*, in fact, deeper general principles involved.

In this paper, we present a general overview of our work with LEGO/Logo. Section 2 provides a description of the LEGO/Logo system. Section 3 discusses how we have used LEGO/Logo in the classroom. Section 4 gives an extended example of a LEGO/Logo project. Section 5 offers some concluding comments and suggests directions for future study.

2. The LEGO/Logo system

In some ways, LEGO/Logo can be viewed as a throwback to the early days of the Logo programming language. Early experiments with Logo in the 1960's used a "floor turtle," a simple mechanical robot connected to the computer by a long "umbilical cord." Students used Logo commands (such as `forward`, `back`, `left`, and `right`) to move the floor turtle around the room.

With the advent of video display terminals, the Logo community shifted its focus to the "screen turtle." Screen turtles are much faster and more accurate than floor turtles, and thus allow users to create more complex graphics. Logo is currently used in more than one third of all elementary schools in the United States, typically with an emphasis on turtle graphics.

LEGO/Logo brings Logo back to three dimensions — but with several important differences. First of all, LEGO/Logo involves *two* types of building:

building LEGO structures and building Logo programs. Second, students are not restricted to turtles; they can build and program a wide variety of different types of machines. Students in our LEGO/Logo classes have built and programmed everything from roller coasters to robots, from conveyor belts to pop-up toasters.

The LEGO/Logo system includes an enhanced set of LEGO building materials. In addition to the familiar LEGO building blocks, there is an assortment of gears, wheels, motors, lights, and sensors. The computer communicates with LEGO devices through a custom-designed interface box, which connects to a parallel card in the computer. Information flows through the interface box in both directions: students can send commands to LEGO motors and lights, and receive status information from LEGO sensors.

Imagine, for example, a LEGO car with a touch sensor on the front. A student can write a program that checks whether the touch sensor is being "pressed" — that is, whether the car has bumped into a wall. The program might make the car reverse direction whenever the car hits a wall.

As its programming language, LEGO/Logo uses an expanded version of Logo. Students can use any of the traditional Logo primitives and control structures (**if**, **repeat**, etc.), plus any of 20 new primitives added specially for the LEGO environment (such as **on**, **off**, and **sensor?**). The new primitives are described in the Appendix.

A typical LEGO/Logo procedure is shown at the top of the next page. The procedure waits until a LEGO touch sensor is pressed, then makes a LEGO light flash on and off five times:

```
to push-button
listento :touch-sensor
waituntil [sensor?]
talkto :light
repeat 5 [onfor 20 wait 10]
end
```

3. LEGO/Logo in the classroom

During the past two years, we tested the LEGO/Logo system with about a dozen elementary-school classes (mostly grades 3–5). Each class used the system for about ten weeks, for roughly three hours per week. With each class, we introduced LEGO/Logo through a relatively standard sequence of activities. In this section, we describe these introductory activities and explain the rationale behind them.

Students started with a simple design activity that did *not* include the computer. We set up a ramp in the classroom, and students built LEGO soapbox cars to race down the ramp. As a goal, we suggested that students try to design cars that would travel as far as possible off the end of the ramp.

Through this activity, students became familiar with basic LEGO pieces and building techniques. Equally important, the activity introduced students to some basic principles of experimentation and design. As students modified their cars to make them go further (changing wheel size, changing weight, etc.), we discussed the importance of changing just one variable at a time. We also encouraged each student to keep an "Inventor's Notebook" to document the design process.

Next, students added gears and motors to their cars and supplied power from a battery box. We suggested that students try different gear ratios to see which combinations were best for going fast on a flat surface, for climbing ramps, and for winning tugs-of-war with other cars. Through these activities, students gained some understanding of transmission systems, mechanical design, and mechanical advantage.

Finally, we added the computer. Students wrote programs to make their cars move in various patterns. Then they added sensors to their cars, and modified their programs so that the cars would, for example, reverse direction when they bumped into obstacles.

After these introductory activities, students worked on "personal projects" of their own choosing. Some continued to work on vehicles (trucks, cable cars, trains), while others moved to different types of machines (such as factory machines or household appliances). We encouraged students to view themselves as inventors. For example, we established a system of *LEGO/Logo patents*, and awarded them to students who appropriately documented their "inventions" through drawings and descriptions.

4. Sample project: A vibrating walker

It is difficult to generalize about LEGO/Logo projects. LEGO/Logo is an environment, not a constrained set of activities; students (and teachers) can use it in many different ways. Nevertheless, we attempt to give a flavor of students' experiences by describing in some detail one particular LEGO/Logo project, developed by a fourth-grade student named Nicky.

Like many students, Nicky started by building a car out of LEGO. After racing the car down a ramp several times, Nicky added a motor to the car and connected it to the computer. When he turned on the motor, the car moved forward a bit — but then the motor fell off the body of the car and began vibrating across the table.

Rather than trying to fix this bug (or, worse, giving up since his car had "failed"), Nicky became intrigued with the vibration of the motor. He began to wonder whether he might be able to use the vibrations to power a vehicle. In effect, he decided to turn the vibrations from a bug into a feature.

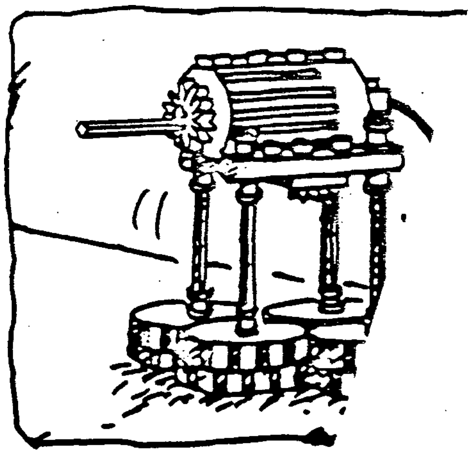

Figure 1: Nicky's [illegible] n

Nicky mounted the motor on a platform atop four "legs" (LEGO axles). After some experimentation, Nicky realized that he needed some way to amplify the motor vibrations. To do that, he drew upon his personal experiences riding his skateboard. Nicky remembered that swinging his arms gave him an extra "push" on the skateboard; he figured that a swinging arm might accentuate the vibrations of the motor as well. So Nicky connected two LEGO axles with a hinged joint to create an "arm." Then he placed a gear on the motor and inserted the arm slightly off-center in the gear. As the gear turned, the arm whipped around — and amplified the motor vibrations.

In fact, the system vibrated so strongly that it frequently tipped over. A classmate suggested that Nicky create a more stable base by placing a LEGO tire horizontally at the bottom of each of the legs. Nicky made the revision, and his "vibrating walker" worked perfectly. When the motor turned in one direction, the walker vibrated forward and to the right. When the motor turned in the other direction, the walker vibrated forward and to the left. Nicky had succeeded in building a steerable one-motor vehicle, a non-trivial engineering feat.

Next, Nicky set out to make the walker follow a black line on the table top. He attached a LEGO optosensor (pointing down) at the front of the walker.

When the the walker passed over a black line, the sensor reported `true`. With a bit of assistance, Nicky wrote this program to make the walker follow the line:[1]

```
to follow
look-for-line
go-past-line
reverse-direction
follow
end

to look-for-line
waituntil [floor-color = "black]
end

to go-past-line
waituntil [floor-color = "white]
end

to floor-color
if sensor? [output "black]
if not sensor? [output "white]
end
```

When the **`follow`** procedure is executed, the walker veers in one direction until it "finds" the line, continues in that direction until it passes over the line, then reverses the direction of its motor and repeats the process. As a result, the walker weaves back and forth over the line, making a bit of forward progress with each cycle.

What did Nicky learn through this project? For one thing, he gained an introductory understanding of some specific engineering concepts. In building the walker, Nicky ended up with an appreciation for both the constructive uses and the destructive potential of vibration in mechanical systems. And in programming the walker to follow the line, Nicky explored basic ideas of feedback and control. Nicky used these same ideas in a later project, when he programmed a LEGO "turtle" to find its way out of a box.

Equally important, Nicky gained a sense of the *process* of design. Indeed, in building the walker, Nicky used an impressive array of *design heuristics.* Among them:

- *Take advantage of the unexpected.* When the motor fell off of his car, Nicky did not see it as a sign of failure. He saw it as an opportunity. He

[1]We have "cleaned up" Nicky's code and divided it into subprocedures in order to make the program more readable.

was on the lookout for unexpected events, and took advantage of them when they happened.

- *Use personal experience as a guide.* When Nicky needed to amplify the vibrations of the motor, he relied on knowledge of his own experiences and body movements.

- *Try using materials in new ways.* The designers of LEGO probably did not envision LEGO axles used as arms or legs. Nor did they imagine that LEGO wheels would be turned 90 degrees and used as feet. But Nicky did not feel constrained by standard usage.

- *Collaborate with others.* When the vibrations kept tipping the walker over, Nicky was uncertain how to solve the problem. So he consulted with a classmate who had a reputation for mechanical-design skills. The collaboration was a success. Such group efforts are particularly important in multi-disciplinary activities like LEGO/Logo.

In discussing LEGO/Logo projects, we encouraged students to think and talk explicitly about such design heuristics. In many cases, we believe that LEGO/Logo activities helped students develop a principled approach to design and invention.

5. Future directions

Our work with LEGO/Logo during the past two years has convinced us of the educational value of constructionist activities. A broad range of students enjoyed working on LEGO/Logo projects (to the point of wanting to continue working during lunch and after school), and their high level of motivation seemed to pay off in several ways. Not only did students appropriate new ideas about science and design, many seemed to gain a heightened sense of self-confidence in themselves as learners.

To date, however, most of our observations have been anecdotal. There is clearly a need for more fine-grained studies of students' design activities in LEGO/Logo. Future research could go in many different directions:

- *Role of the teacher.* As students worked on personal LEGO/Logo projects (after the introductory activities), we and the classroom teachers provided only a minimal structure. Clearly, though, we played an important role in helping students conceptualize their projects and make connections to general principles. Future research could look at the appropriate role for teachers in open-ended design activities like LEGO/Logo.

- *Differences among students.* We were particularly pleased that LEGO/Logo appealed to a broad range of students of both genders; its appeal was not limited to those students seen as "good at math and science." Different students, though, worked on different types of projects, and with different design styles. A study of these stylistic differences could prove interesting.

- *Transfer.* Can students transfer the skills used in LEGO/Logo to other activities? In particular, does LEGO/Logo experience affect students' approach, competence, or self-confidence in other design activities?

- *New programming paradigms.* In programming LEGO/Logo machines, students sometimes want to program more than one device at a time (for example, making one LEGO "animal" chase another). Such concurrent control is not possible (or, at least, it is very difficult) using traditional programming languages. To address this problem, we have developed and tested an extension of Logo that includes concurrent-programming constructs (Resnick 1988). We believe further research in this direction would be worthwhile.

- *Different ages.* Most of our work has focused on elementary-school students. In the future, we plan to work with older students on more complex projects. LEGO/Logo could, for example, serve as the base for a high-school course on artificial intelligence and robotics, or as a useful tool within a high-school physics course.

- *New sensors and actuators.* In most of our work, students were limited to two types of sensors (touch sensors and optosensors) and two types of actuators (motors and lights). Adding new sensors and actuators would expand the range of possible projects.

Acknowledgments

Our research on LEGO/Logo is supported by grants from LEGO A/S and the National Science Foundation.

LEGO Systems Inc. now markets a version of the LEGO/Logo system under the product name *LEGO TC logo*. Logo Computer Systems Inc. implemented the software for this commercial version.

References

Bennett, William (1986). *First Lessons: A Report on Elementary Education in America.* U.S. Department of Education.

Driver, R., et al. (1985). *Children's Ideas in Science.* Open University Press. Milton Keynes, England.

Harvey, B. (1986). *Computer Science Logo Style.* MIT Press. Cambridge.

Osborne, R., and Freyberg, P. (1985). *Learning in Science: The Implications of Children's Science.* Heinemann Publishers. London.

Papert, S. (1980). *Mindstorms.* Basic Books. New York.

Papert, S. (1986). *Constructionism: A New Opportunity for Elementary Science Education.* A Proposal to the National Science Foundation.

Resnick, M. (1988). *MultiLogo: A Study of Children and Concurrent Programming.* Master's thesis, Laboratory for Computer Science, MIT.

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

West, L., and Pines, A.L. (1985). *Cognitive Structure and Conceptual Change.* Academic Press. New York.

Appendix: LEGO/Logo primitives

1. Output primitives

• `talkto` *port*

Tells the computer which ports it should "talk to" when it sends commands to the interface box.

• `on`

Turns on motors and lights.

• `off`

Turns off motors and lights.

• `alloff`

Turns off *all* motors and lights connected to *all* ports.

• `onfor` *ticks*

Turns on lights and motors for the indicated *ticks* of time. There are ten "ticks" in one second.

• `rd`

Reverses the direction of motors.

• `seteven`

Sets the direction of motors to the "even" direction. (Each motor port has two directions: the "even" direction and the "odd" direction.)

• `setodd`

Sets the direction of motors to the "odd" direction.

• `flash` *ticks-1 ticks-2*

Tells lights to begin flashing. The lights will turn on for *ticks-1* time, off for *ticks-2* time, over and over. There are ten ticks in a second.

- `setpower` *level*

Sets the power level for motors and lights. The power level affects the speed of motors and the brightness of lights.

2. Input primitives

- `listento` *port*

Tells the computer to "listen to" the indicated port for receiving information from sensors.

- `sensor?`

Reports the state of the sensor. Each sensor has two possible states: `true` and `false`.

- **`counter`**

Reports the value of the sensor counter. The sensor counter automatically increases by one every time the sensor switches from **`false`** to **`true`**. This primitive is particularly useful for measuring the rotation of the LEGO counting wheel (a disk with alternating black and white slices).

- **`resetc`**

Resets the value of the sensor counter to zero.

- **`pulsewidth`**

Reports the duration of the sensor's most recent **`true`** pulse. That is, it reports the length of time between the most recent **`false-to-true`** and **`true-to-false`** transitions. This primitive is very useful in measuring differential quantities, such as the velocity of a car.

3. Control-structure primitives

- **`waituntil`** *list*

Waits until the value of *list* is **`true`**. For example, **`waituntil [sensor?]`** waits until **`sensor?`** reports a value of **`true`**.