Children, Cybernetics, and Programmable Turtles

by

Fred Martin

Massachusetts Institute of Technology

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the
degree of Master of Science in Mechanical Engineering
at the

Massachusetts Institute of Technology

August 1988

Signature of Author [illegible]
Department of Mechanical Engineering
August 12, 1988

Certified by [illegible]
Seymour Papert, Thesis Supervisor
Professor of Media Technology

Accepted by [illegible]
Woodie Flowers, Thesis Reader
Professor of Mechanical Engineering

Accepted by [illegible]
Ain Sonin
Professor of Mechanical Engineering

Children, Cybernetics, and Programmable Turtles

by

Fred Martin
Massachusetts Institute of Technology

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the
degree of Master of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology

August 1988

Abstract

A microworld to support investigations of cybernetics is developed, consisting of a miniature computer programmable in Logo, called the "Logo Brick," and mobile mechanical vehicles built of LEGO plastic parts. These "Programmable Turtles" are equipped with various sensors, including touch sensors and light sensors.

A series of experiments with these materials is performed with two groups of elementary school children, aged from twelve to fifteen years. The children design behaviors for the Turtles by writing Logo programs and building materials in the Turtles' environment.

The children encounter several powerful ideas in cybernetics: feedback, multiple levels of analysis, and object/environment boundaries. Through the course of the experiments, their understanding of the Programmable Turtles deepens and becomes more flexible.

Thesis Supervisor: Seymour Papert
Professor of Media Technology

Acknowledgements

Many people contributed to the ideas presented in this thesis. In some ways I am the scribe for ideas that were developed in on-going conversations. I would like to especially thank:

- Mitch Resnick and Steve Ocko, for developing LEGO/Logo in the first place, and always having big reservoirs of great ideas.
- Brian Silverman, for being the genius behind the technology.
- Allan Toft, for his support from the Land of LEGOs.
- Mario Bourgoin and Aaron Falbel, for providing helpful criticism of drafts of the writing.
- Edith Ackermann, for continuously challenging me.
- Seymour Papert, for making all of this possible.
- LEGO A/S Denmark, for providing the financial backing for this work.

Fred Martin
August 10, 1988

Contents

List of Figures

Chapter 1

Introduction

This thesis is about cybernetics, the study of systems or feedback networks of interacting elements. It's also about children's thinking as they build robots and observe them interacting with their environment and other robots. And it is about the relevance of this work to science education.

The foundations of cybernetics were laid two thousand years ago by the inventions of the Hellenistic period. But its principles remained in obscurity until James Watt's invention of the steam engine governor in 1784. The governor had an enormous impact on the technology of the era, catalyzed the birth of the Industrial Age, and introduced the study feedback control as an engineering technique.

In the 1940's, the work of pioneer cybernetician Norbert Wiener and others showed how the feedback could be used in a variety of areas of scientific inquiry, such as physiology and psychology. A community of scientists developed methods and styles of inquiry of the field which came to be called cybernetics. The modern methods of dynamical systems analysis and contemporary theories of mind such as the Society of Mind by Marvin Minsky have precursors in the work of the cyberneticians of the 1940's and 1950's.

The concepts of systems theory, such as feedback, seem very complex, but that is because they are explored in highly abstract and mathematical ways. The Swiss psychologist Jean Piaget argues that as a child's mind develops to become able to perform abstract types of thought, it passes through stages which support these forms of thought. I believe that the concepts of cyberneticians could have a corresponding pre-formal representation in minds of children and novice adults.

Seymour Papert argues that many ideas of mathematics are hard to learn because the tools for their appropriation do not exist in the learner's environment. This explains why people have trouble with classroom-style mathematics; the tools used to learn mathematical ideas are abstract and make it difficult for the learner to "sink his teeth" into the material. In his book *Mindstorms*, Papert describes the development of Logo, a computer programming language designed to support the "speaking" of mathematics in a way analogous to how living in France supports the speaking of French.

Papert explains how the Logo language could be situated in an environment of "math-speaking" persons and other materials that support the development of an active mathematical mind. Logo would have a central role in this environment which Papert calls "Mathland." (Papert 1980)

If this need for materials of appropriation is true of mathematics, than it makes sense to explore a miniature Mathland, or *microworld*, that supports the techniques of cyberneticians. I will review four microworlds which make these ideas accessible to children and novices:

- Papert's Logo and the Logo turtle (Papert 1980);
- Valentino Braitenberg's experiments with programmable vehicles (Braitenberg 1984);

- animal microworlds software being developed by Roy Pea and associates (Pea et al. 1988);
- a mechanical and computer-based building system, called LEGO/Logo, developed by Papert, Ocko, and Resnick (Ocko et al. 1988).

The work described in this thesis is based around an extension to the LEGO/Logo system in which the computer, rather than being a desk-sized personal computer, is miniaturized to the size of a deck of cards. It can then be incorporated into a LEGO machine, and each machine can literally carry its own program. Very small computers encourage the design of mobile, animal-like machines, which can interact with each other in unpredictable and interesting ways.

Several sorts of sensors were created for the "artificial animals" to receive data from their environment.

These materials were developed to meet three goals:

1. To build a flexible and engaging microworld that can provide children access to the powerful ideas of the cyberneticians.
2. To learn about children's intuitive ideas relating to feedback and systems thinking.
3. Through experimentation, learn how to make these ideas more readily understandable to children.

When the technological materials were ready, they were used in a series of experiments with children.

The first of those who used the materials were fifth grade children attending the Hennigan school in Jamaica Plain, Massachusetts, the site of a three-year-long Logo project called Project Headlight, led by Seymour Papert and his M.I.T. associates. The children had had intensive computer and Logo programming experience while part of Project Headlight.

Afterwards, experiments were done with a group of seventh and eighth grade students who were attending a private school in Newton, Massachusetts. The experiments with the fifth grade children took place over the course of a school year (nine months), while those done with the older children took place over a six-week period. These experiments are described in Chapter 4.

Experience gained in one experiment often guided the construction of subsequent ones; the activities performed with the older children made considerable use of knowledge gained in earlier experiments. This forms a design cycle for making the system more understandable to children. The steps in this cycle are: developing a prototype system, observing children using this system, analyzing how they appropriate or have trouble appropriating certain ideas, and then redesigning the system in order to try again.

In Chapter 5, the children's activities in each of these experiments are described. Chapter 6 contains an analysis of the children's work which brings out the following issues:

1. Development of the children's understanding of feedback, as relating to activity they have designed for their robot turtle.

2. Children's tendency to *anthropomorphize* the turtle into a living creature, which affects the style of interaction they have with it.

3. Ability of children to analyze the turtles' behaviors using several different styles or levels of explanation.

4. Emergent phenomena resulting from multi-processing (multiple robot interactions) and "environment programming" (robot behavior affected by programming the environment).

The conclusion, in Chapter 7, suggests improvements and extensions to the materials including new sensor technology and more powerful programming paradigms, proposes alternate styles of research using the materials, and makes connections between the experiments and science education.

Chapter 2

Theoretical Background

2.1 A Short History of Cybernetics

The field of cybernetics began with a single idea, that of a feedback system. Using negative feedback, a system is able to regulate one or more variables by making corrections to its performance. These corrections are based on observations of its current state.

For example, you are part of a feedback system, where you act as the controller and sensor, when you adjust the temperature of your morning shower. If the water feels too cold, you make it hotter (with either more hot water or less cold). If the water feels too hot, you make it colder. You repeat this action of testing and then correcting several times before you are satisfied.

You make these corrections based upon your preferred temperature which is your goal. If the difference between the current temperature and the goal temperature is large, then you make a large correction. If the difference is small, then you make a small correction. This guiding difference is called the *error temperature.*

This example of a feedback controlled system uses a person to sense the error and apply the correction. If we analyze this process, we can extract its key steps, and form a working definition of a feedback system. Then we can look for examples of machines doing that job.

SHOWER FEEDBACK

1. Measure water temperature.
2. Find difference between goal temperature and measured temperature.
3. Make correction so as to reduce that difference.
4. Repeat from step 1.

This feedback system is called a *negative* feedback system, because it tries to *reduce* the error. If it *increased* the error, then it would be a positive feedback system. Positive feedback systems are extremely unstable, because any error will push the system further and further away from equilibrium.

It is believed that the first machine that had a negative feedback mechanism was a water clock designed Ktesibios of Alexandria, who lived between 300 and 250 B.C. His feedback mechanism was used to increase the accuracy of a water clock.

The feedback device maintained a constant flow of water that was measured to show the passage of time. It worked by keeping a constant water level in a regulating tank, which would ensure constant water flow.

The feedback device was a *float valve*: a cork at the end of a rod. The cork floated on the water in the regulating tank, and moved the rod. It, in turn would operate a valve which controlled the amount of water *allowed to enter the tank.*

When the water level in the tank rose, the cork would rise, cause the valve to close. When the water level fell, the cork would fall, and cause the valve to open.

Throughout the following centuries, several other inventions made use of a feedback mechanism for some regulating purpose. Often the earlier machines were unknown, so the concept of feedback was rediscovered many times. Among the inventions incorporating feedback were:[1]

- A flow regulator for an oil lamp, designed by Philon of Byzantium in 250 B.C.;
- water clocks of the Islamic region, dating to the 1200's A.D.;
- heat regulated furnaces of Cornelis Drebbel of Holland, in early 1600's;
- pressure regulated boilers in the 1600 and 1700's (England);
- feedback mechanisms on windmills, designed to aim the mill blade toward the wind, and regulate the speed of the mill (1700's, England, Scotland, and Holland)
- James Watt's centrifugal governor on the steam engine (1784, England)

These machines had remained obscure in their time, so that feedback remained hidden until James Watt's governor on the steam engine.

The Watt governor greatly increased the usefulness of the steam engine and catalyzed the dawn of the Industrial Age. It also captured the attention of the engineering community, and initiated international recognition and study of feedback engineering.

Watt was believed to have invented the first feedback device, and was attributed to the patent of the steam engine governor. In fact, Watt himself

[1]This historical research is taken from Otto Mayr's book, *The Origins of Feedback Control*, which includes in-depth analyses of each of these inventions.

recognized his invention as another application of the technique that he had observed in windmill regulation, and never sought a patent for his regulator. As explained by Otto Mayr[2]:

> The governor did not strike him as new; he considered it merely an adaptation of a known invention to a new task. [...] Compared with the large but straightforward task of producing power, the regulating devices and whatever theory they involved may have appeared to the sober Watt as secondary, if not marginal.

By the 1820's, the governor had gained a firm place in the engineering textbooks, and by the late 1800's, engineering theory included dynamic control theory, and many historical and general-interest works were available to laymen.

Feedback remained exclusively in the domain of control engineering until World War II when a group of scientists, formed in 1910, explored feedback mechanism in anti-aircraft weaponry. Eventually, they came to the conclusion that feedback was important in the operation of animals.

These scientists held meetings through the 1940's and continued research and discussion to find common ground amongst the field of mathematics, physiology, psychology, and engineering. This work coalesced in Norbert Wiener's publication of *Cybernetics: Control and Communication in the Animal and the Machine* in 1948. He chose the word *cybernetic* from the same Greek root word from which *governor*, the word James Watt chose to describe his feedback invention, is derived.

The development of cybernetics as a field in its own right continued, with a regular series of conferences, a number of journals devoted to the field, and several university programs partially or wholly dedicated to cybernetics.

[2]ibid., pp. 112

In the 1960's there was some controversy over what exactly the field of cybernetics was. Michael J. Apter, a developmental biologist and cybernetician, made the following comments in his book entitled *Cybernetics and Development* (Apter 1966):

- Cybernetics has no central part but consists of many strands that overlap each other in different ways, including computer and communication engineering, mathematics and logic, biology, physiology, and psychology.
- There are genuine differences of opinion among cyberneticians themselves about what cybernetics is and what cyberneticians should be doing. As two examples along a continuum of opinions: Stanford Beer looked on cybernetics as the science of proper control within any assembly that is treated as an organic whole; while Ashby emphasized abstracting a controllable system from the flux of the real world.

Apter also pointed out criticisms of the cybernetic approach, which claimed that cybernetics consisted mostly of analogy-making and lacked empirical data.

Those criticisms became invalid as analogy-making evolved to model-building, and the technique of computer simulation, based on these models, drove research. The advent of computer simulation justified the approach of the cybernetician, who, in the words of Apter, "is doing more than merely drawing analogies between animals and machines, but is concerned with the development of conceptual systems at a level of generality which subsumes both animals and machines."

Models and computer simulations are simplified versions of the phenomena that they represent, but they can provide enough complexity to possess the properties of the actual systems of study. They may even reveal or predict phenomena that would be difficult or dangerous to observe in the actual systems.

This approach is further validated by comments of Roy Pea and associates (Pea 1988), who explain that simulation and modeling techniques combine the classical "analytic" and "synthetic" methods of science.

The analytic approach is that of observing phenomena and then seeking laws to account for them; the synthetic approach involves confirming the validity of laws by prediction and experimentation. Computer simulation combines these two approaches: in creating a model, there is an analysis of the system of study; in experimenting with a model, the scientist synthesizes observations of the model system to match with the actual system.

Contemporary theories of mind, such as the "Society of Mind" theory (Minsky 1985) and connectionism (McClelland et al. 1986) incorporate mechanisms more sophisticated than simple negative feedback in their models. Minsky's theory is based upon hierarchical societies of interacting managers and agents, and the connectionists emphasize emergent properties within a certain level of interconnection.

These models extend negative feedback by placing each element in a web of like elements. This is similar to the technique of the cybernetician, who might study the fashion in which the body's control systems (nervous, homeostatic, circulation) interact.

2.2 Educational Initiatives

Cybernetic ideas seem complex, but often it's because the tools used to explore them are abstract and highly mathematical. Perhaps that is why they aren't treated in the traditional educational curriculum before the college level. This section will describe previous work which sought to make these concepts more accessible to children and technical novices.

Dr. Grey Walter was a British cybernetician, neurophysiologist, and pioneer in electroencephalography. In the 1950's he constructed a series of machines he called mechanical "tortoises" because of their characteristic shell-like protective cover.

They incorporated touch sensor and light sensor elements and some fairly simple electronic circuitry. Yet, they exhibited a variety of complex behaviors, including positive and negative tropism[3] for light, exploratory movement, and self-recognition (when the tortoise"saw" its reflection in a mirror, it would do a sort of dance). Some of these behaviors were specifically designed into the machines (such as the tropisms) and others were unanticipated, emerging from the interactions of the turtle with its environment (such as the self-recognition dance).

Dr. Walter did not report any experiments with children's opinions of his turtles, but did mention that their animal-like behavior would occasionally frighten visitors to his lab.

Seymour Papert added a *floor turtle* to early versions of Logo. It was a small mobile device which could be controlled by commands from the computer. He intended the turtle to be a *body-syntonic* object for children; they could use knowledge about their own bodies in understanding how the turtle moved. In this way, the turtle would become an *object-to-think-with.*

In his 1984 book, *Vehicles: Experiments in Synthetic Psychology*, Valentino Braitenberg described a series of vehicles that were hypothetical self-operating machines, built out of connections of simple electronic elements. They would display a range of behaviors, progressing from simple attraction and repulsion to complex "emotions" and "intelligence."

[3]an attraction or repulsion

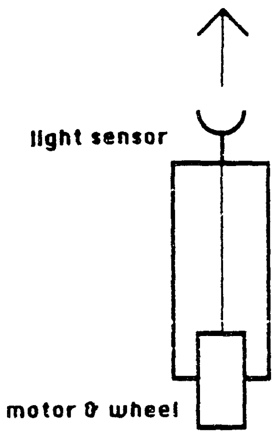


Figure 2.1: Braitenberg Vehicle No. 1

Braitenberg's vehicle No. 1 is shown in Figure 2.1. This vehicle consists of a light sensor connected (via a wire) to a motor. The more light received by the sensor, the faster the motor turns.

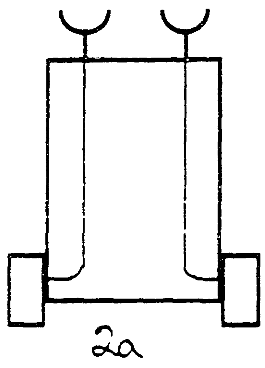
Vehicles No. 2a and 2b make use of two of these light-sensor-and-motor circuits, to form a vehicle that has the ability to turn (Figure 2.2). As indicated in Figure 2.3, these vehicles display either a positive or negative tropism to light.

Let's examine vehicle 2b. If a light is positioned to its left (as in the figure), then its *left* light sensor will be excited to a greater degree than its right sensor, and its *right* wheel will turn faster than its left wheel. This will cause the vehicle to turn *towards* the light; the sensor-motor interaction will continue to drive it toward the light.

In a complementary fashion, vehicle 2a turns away from the light.

Propcsed as a series of thought experiments, at least two projects have taken inspiration from Braitenberg's work. One is software designed for the Macintosh that allows some experimentation with early vehicles. The software is not very flexible and allows only rudimentary experimentation.

A much more elaborate and powerful Braitenberg microworld is presently being developed by Roy Pea, Michael Eisenberg, and Franklyn Turbak (Pea



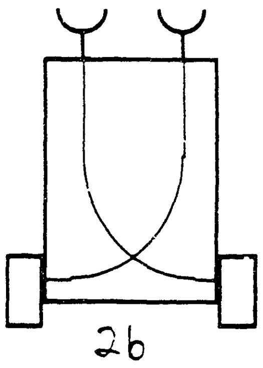


Figure 2.2: Braitenberg Vehicles No's. 2a and 2b

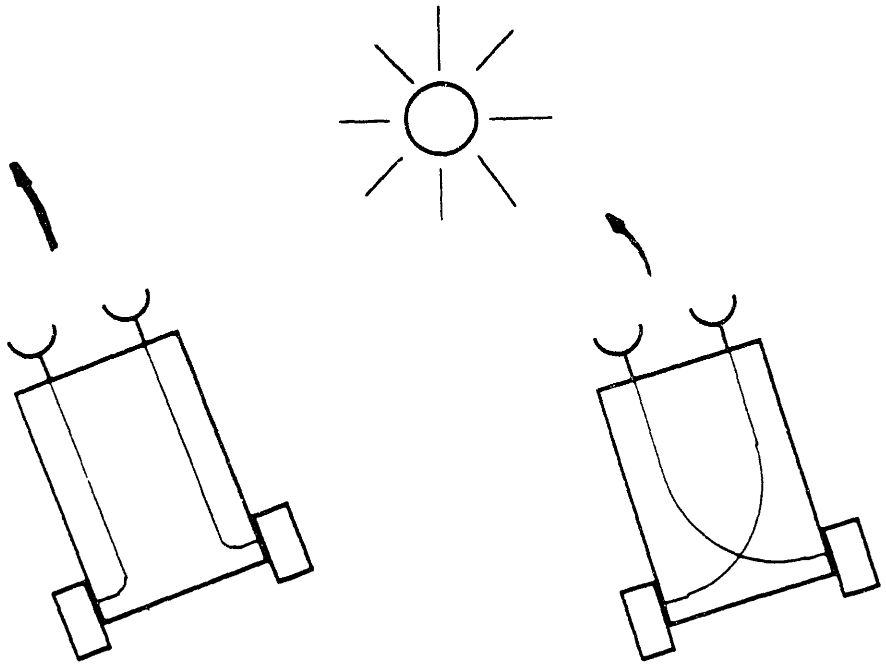


Figure 2.3: Braitenberg Vehicles 2a and 2b Encountering a Light

et al., 1988). Entitled "Creatures of Habit," it is a universe of interacting programmable "creatures" whose individual behaviors are guided by simple rules that may model naive psychology, physical laws, chemical affinities, and other domains. Using the system, students may create or revise creature rules and explore resulting emergent behaviors within the "artificial ecosystems."

A system called LEGO/Logo takes this "artificial animal environment" off the computer screen and puts it into three dimensions. Designed in a collaborative effort between LEGO A/S, a company based in Denmark, and Seymour Papert, Stephen Ocko, and Mitchel Resnick of M.I.T., LEGO/Logo is a building set consisting of LEGO plastic building materials — beams, blocks, gears, pulleys, wheels, and motors — and a computer interface involving sensors and Logo programming.

The materials are part of a learning environment for children, wherein they build motorized machines and then hook them up to the computer, thus enabling them to write programs to animate their constructions. The LEGO building parts are extremely versatile, as evidenced by the variety of machines children have constructed, including cars, dinosaurs, and roller-coasters.

Experience has showed that these materials can tap a wealth of creativity in children. Many children who do not possess verbal or mathematical skills (and those who do) are able to use a form of "spatial intelligence" and knowledge of physical materials in LEGO/Logo activity. Often collaboration occurs between children who combine LEGO skills and Logo skills to accomplish a project (Ocko et al. 1988).

The LEGO/Logo system incorporates two types of sensors that can be built into LEGO machines: a touch sensor and a light sensor. Children often will build a car, and then mount a touch sensor on its bumper. Then they can write a Logo program to cause the car to change direction when the touch sensor is triggered, as when the car hits a wall.

The LEGO/Logo system includes sensors, motors, and control (Logo programming), which are exactly the elements needed for explorations of cybernetics. However, each LEGO machine must be hooked up with wires to its computer, and it is difficult to write programs for more than one machine at a time. These limitations constrain the extent to which machines may interact with each other and their environment. And it exactly these interactions which are of most interest to the cybernetician.

Chapter 3

Programmable Bricks

3.1 Logo Bricks

In the original version of LEGO/Logo, a machine must be hooked up via long wires to a host computer, which is a desktop-sized personal computer. The limitations of this system led to the idea of a "computer-in-a-LEGO-brick" that could be built into the LEGO machine itself.

Hence, the development of the *Logo Brick*: a computer inside of a LEGO brick. One programs it in Logo, by hooking it up to a "large" computer that has a keyboard and screen. The Logo Brick can directly control up to four motors, and receive information from several sensors.

Figure 3.1 shows a picture of the Brick and its insides. The Brick measures 9 cm long by $5\frac{1}{2}$ cm wide by $3\frac{1}{2}$ cm high — about the size of a deck of cards. It is powered by a rechargeable battery pack of the same dimensions.

There are several implications for projects that are built using the Logo Brick, rather than using original LEGO/Logo. Machines can be designed for mobility without having to worry about tangled trains of wires. This encourages more "animal-like" inventions, instead of stationary machines.

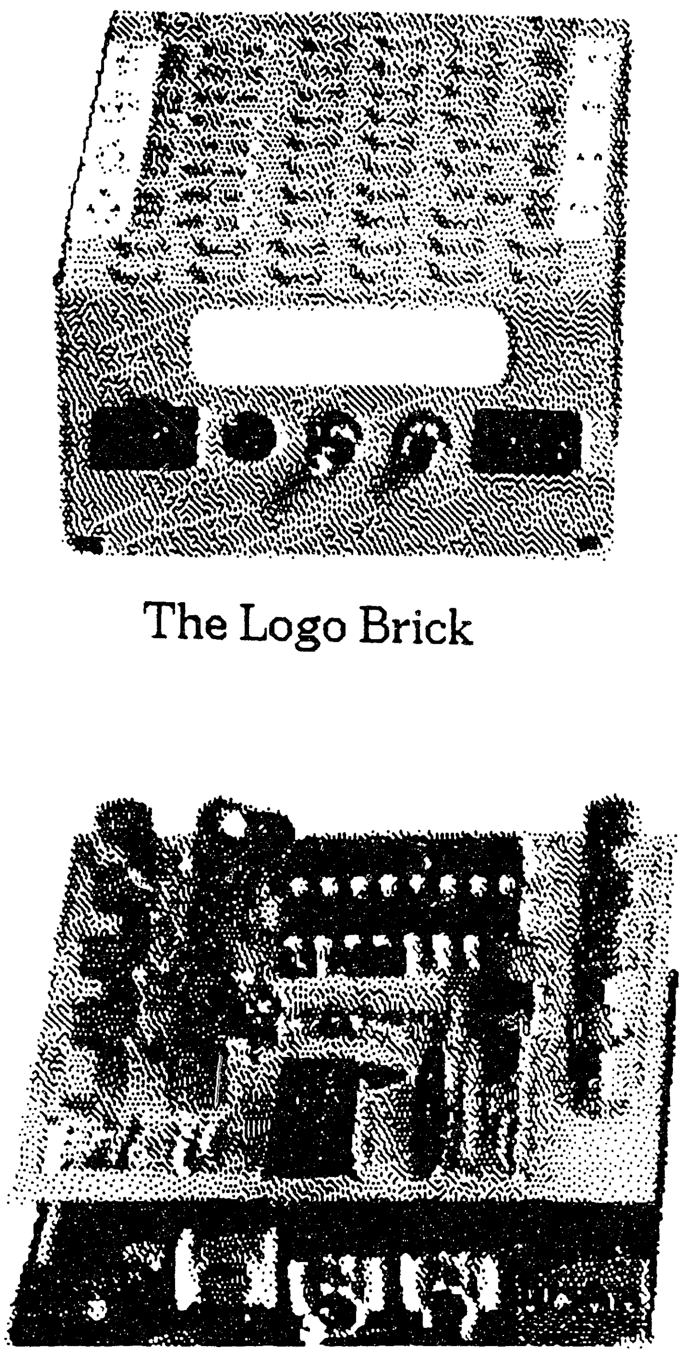


Figure 3.1: The Logo Brick

Because each machine can carry its own program aboard the Logo Brick, multi-machine projects are possible without cumbersome programming techniques. Projects involving more than one machine often reveal unpredictable emergent phenomena from interactions between the machines, which are interesting to study.

Appendix A includes a history of the development of the Logo Brick and technical information on its use.

3.2 Sensor Devices

New sensors were developed for use with the Logo Brick. I will mention these devices briefly here, and explain their operation in more detail in the context of experiments.

- *LEGO Touch Sensor.* Responds to contact pressure. Same as in original LEGO/Logo.
- *LEGO Light Sensor.* Useful for certain tasks, but not used in any of the experiments with the Logo Brick.
- *Sound Sensor.* Responds to "beep" generated by small beeper unit. Allows signalling from one machine to another.
- *Directional Light Sensor.* Indicates direction of brightest incoming light.
- *Threshold Light Sensor.* Responds when incoming light surpasses an adjustable threshold.
- *Analog-to-Digital Light Sensor.* Reports a numeric value corresponding to the amount of light it is receiving. None of the experiments made use of this sensor.

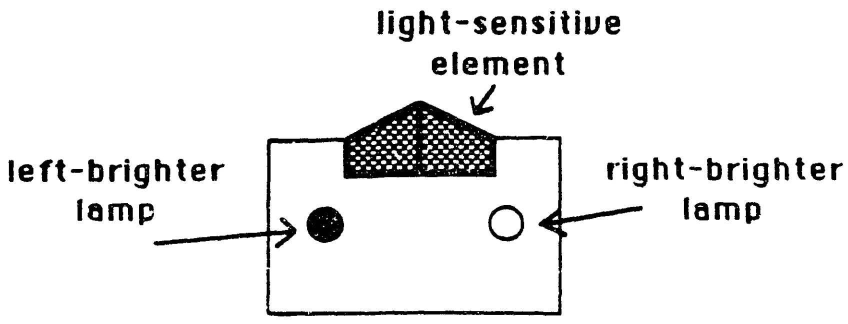


Figure 3.2: Illustration of Light Sensor

The directional light sensor played a central role in much of our experimentation.

This light sensor would indicate a *difference in brightness* of light: it would tell whether incoming light was brighter from a left-hand direction, or whether it was brighter from a right-hand direction. It would indicate this by lighting one of two small lamps, and by sending a corresponding signal to the Logo Brick (See Figure 3.2).

We used this sensor to design a behavior whereby a vehicle could *follow a light*. The vehicle would use the light sensor to correct its course repeatedly so as to point towards the brightest source of light. This behavior became the paradigm feedback process for the children to explore.

The indicator lamps were designed into the light sensor for the human experimenter's benefit when working with the device. These lamps indicate the state of the sensor, in order to help a person using the device to test what to what the device responds.

A similar feature was installed into the sound sensor. This device signalled "true" when it was "hearing" a beep, but unless the Logo Brick using the sensor is programmed to perform some specific and evident action upon reception of the sound, this state of recognition is invisible to the observer.

So I added a small lamp that would light when the sensor received the sound.

I chose to implement the sound sensor as the first wireless communications device specifically because the mode of communication, a high-pitched tone, can be directly perceived by a person. Radio or infrared communications would also have worked for a wireless communications method, and have some advantages such as increased range, but I wanted a device whose sensory modality children could directly perceive and easily understand.

3.3 Braitenberg Bricks

In addition to the fully programmable Logo Brick, a set of hardware bricks were developed that allow construction of several of the vehicles described in Braitenberg's book.

These "Braitenberg Bricks" consist of a *light sensor brick* and a *motor driver brick*. When the light sensor is connected to the motor driver, the motor will turn at a rate proportional to the amount of light being received by the light sensor: the more light shining on the sensor, the faster the motor will turn

An additional *inverter brick* can be wired in the circuit between the light sensor and motor driver. The motor will then turn according to an inverse-proportional relationship: the more light received the slower the motor will turn.

An arrangement of two of these circuits can be used to make a vehicle that is either attracted to or repelled by a light source, as described by Braitenberg in his vehicles 2a and 2b (Figure 2.2).

Chapter 4

Description of Experiments

The chapter outlines the series of experiments I led with two groups of children: fifth grade children at the James W. Hennigan school, and seventh and eighth grade children from the Fessenden school, a private school for boys in Newton, Massachusetts.

In this chapter I describe the activities that were set up for the children, explain how these activities were chosen, and indicate what I expected the children and the researchers[1] to gain from them. I wish to stress the exploratory nature of these experiments, on the part of the researchers as well as the children, and that I would classify this work as a pilot study.

Chapter 5 contains a narrative of what the children actually did in these experiments, and Chapter 6 presents an analysis of these activities.

4.1 Format of the Experiments

M.I.T. researchers worked with the Hennigan children beginning in October, 1987 through June, 1988. There were varying numbers of researchers/observers on hand at a given session; from two or three to up to ten

[1]Many people contributed regularly to these sessions, including Edith Ackermann, Mitchel Resnick, Carol Strohecker, and Allan Toft.

onlookers. Several of the sessions were videotaped.[2] The sessions ran about once a week (one hour per session) from October to December, and about once every two weeks from February to June.

The groups of Hennigan children were taken from two "advanced work" fifth grade classes. The children in these classes were considered more capable in traditional school work than children in the regular classes. We chose children from this group deliberately, expecting them to learn how to deal with complex subject matter more quickly. We met with groups of four children from each of the two classes. Each of the groups consisted of two boys and two girls.

Two of the boys in one of the groups were asked to leave the group after one month. The other group of four was split into two groups of two (two boys and two girls) beginning in March, 1988. There had been a difficulty in the larger group that not enough of the children could participate at once; there was only enough hardware to support one computer terminal for which to program the Logo Bricks. The groups of two worked out much better.

The group of older children were seventh and eighth graders from a private school in Newton, Massachusetts. They were chosen for their interest and demonstrated ability by their computer teacher. I met with six boys over a period of four weeks for two two-hour sessions per week.

4.2 Research Methodology

The research was characterized by direct involvement in the children's activities. This strongly guided approach allowed me to go further in the use of the materials within the time period of experimentation. The children's

[2]My thanks to Scott Paul for this service.

freedom was somewhat limited in this departure from earlier LEGO/Logo philosophy. Teachers using the original LEGO/Logo materials were encouraged to allow children to invent their own projects as well as to build them (Ocko et al. 1988). In these experiments, I wanted to be able to steer the children into certain types of problem-solving situations.

There was an on-going process of examination of the experimental content and style during the course of the experiments. Experiments were often planned after analyzing the previous week's activity, or were motivated by a new sensor device constructed at the M.I.T. laboratory. In one case, a project idea of the children led to the development of a sensor in order to realize that idea.

I was interested in *finding out what the problems are*; that is, conducting an in-depth survey, with new materials, to reveal what is interesting, and worth exploring further, about them.

4.3 Materials Used in the Experiments

4.3.1 Programmable Turtles

In order to focus on new activities using the Logo Brick, I felt it would be more efficient to give the children pre-constructed LEGO machines, which they could program, rather than have the children spend time on mechanical LEGO work. Therefore, I designed a vehicle for them to use in their Logo Brick activities.

I call this vehicle the *Programmable Turtle*. It is the LEGO equivalent of the Logo turtle in that it can move in a similar fashion: forward and backward movements, and left and right turns. The Programmable Turtle carries a Logo Brick and battery pack on its back. It is outfitted with two touch bumpers for sensing collisions. Figure 4.1 shows a picture of the Programmable Turtle.

A variety of sensors were used in the experiments. These sensors were plugged into the Logo Brick and then mounted physically on the Programmable Turtle. The following devices were added onto the Turtle during the course of the experiments: lights, a beeper, a light sensor (two kinds), and a sound sensor.

4.3.2 Programming Tools

To make the Programmable Turtles easier to use, I designed a set of programming tools, which I called the "Tools Words," for communicating with the Turtles.

Using the primitive operations of the LEGO/Logo language, one must give instructions explicitly to motors and sensors. For example, to make the Turtle go forward, one would have to give instructions to turn on both the left motor and the right motor. The LEGO/Logo commands to do this are:

```
TALKTO [A B] SETEVEN ON
```

The first command, TALKTO, tells the computer to send commands to the motors labeled A and B. The second command, SETEVEN, sets those motors in the "even" direction, which is wired on the turtle for forward. The final command, ON, turns the motors on.

I encapsulated all of those instructions into a single Tools Word, named GOFD (for GO ForwarD). The full set of Tools Words that I gave to the children included words for moving the turtle forward, backward, left, and

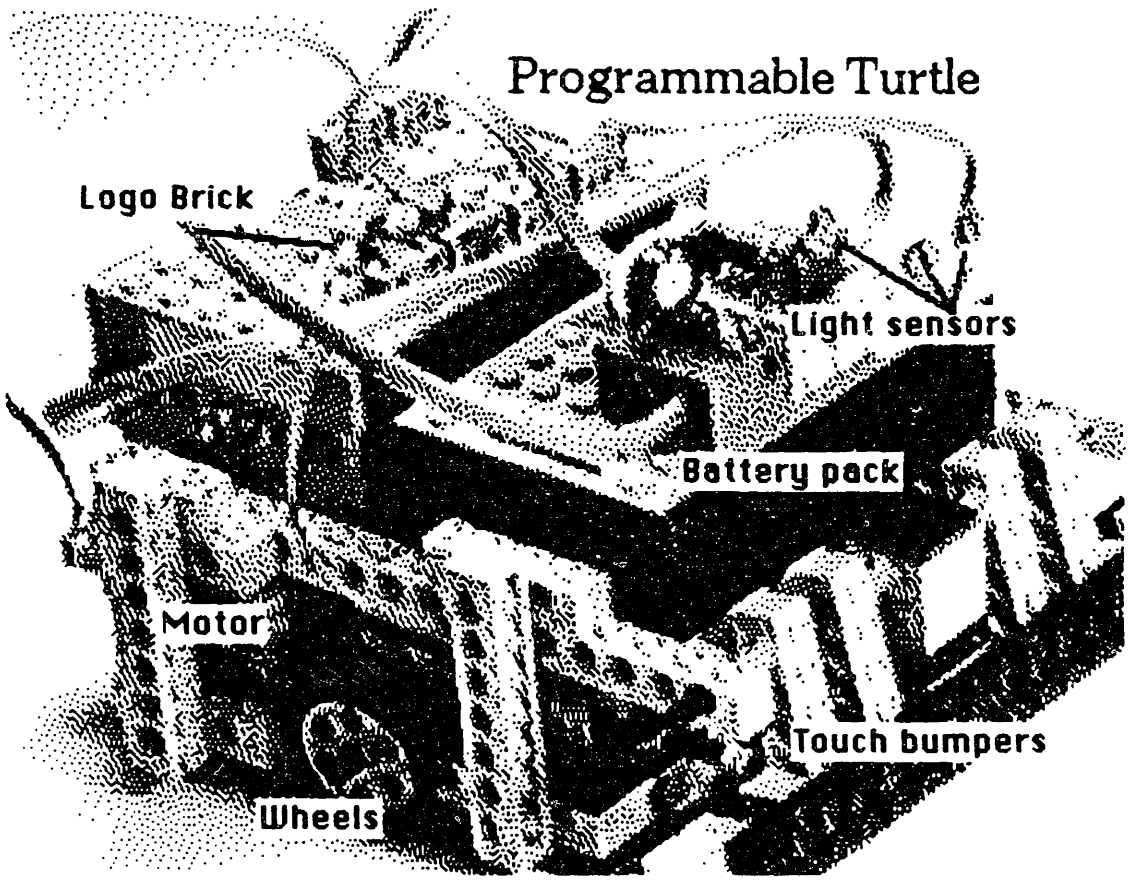


Figure 4.1: Programmable Turtle with Logo Brick and Battery Pack

right, and for inputting information from the touch sensors and light sensors.

This sort of module construction is a commonly accepted programming practice; the Logo language was specifically designed to support this sort of construction. Once you have written a tool, you can "forget" about how the work is done.

Unfortunately, I did not introduce the Tools to the Hennigan children until the springtime. Several of the earlier experiments consisted of working out low-level details; this work was unnecessary and could have been avoided by introducing the Tools earlier on. I introduced the Newton children to the Tools in their first session with the Logo Brick without causing them any confusion.

4.4 Description of Experiments: Hennigan Children

The Hennigan children were already familiar with the original LEGO/Logo materials, so we planned the first session to be a review of LEGO/Logo.

In the next session we planned to introduce the children to the Logo Brick, and teach the children how hook the Brick up to an Apple IIGS computer, how to send it commands, and how to write Logo programs for it.

We wanted to introduce the children to the Programmable Turtles using a sensor that they were already familiar with, so we chose a project using the touch sensors. The task for the children was to write a program that would make the Turtle back up and change direction after it had run into something.

We then introduced the *directional light sensor.* As mentioned earlier, this light sensor indicates direction of brightest light (either towards the left or towards the right) by lighting a small lamp and reporting a value to the Logo Brick.

We asked the children to use this sensor to make the Turtle either seek the light or avoid the light.

When this had been accomplished, we began a series of sessions to explore the behavior of the light-seeking turtle. Children were given several sorts of lights, including blinking lights, steady lights, and flashlights. We suggested experiments such as positioning two lights of equal brightness at different distances from the Turtle, or placing a bright light far away and a dimmer light nearby. The children were invited to think aloud about these problems, and then perform an experiment to see what the turtle would do. We expected these experiments would allow us to explore the children's theories about how the turtle "follows the light."

After about six weeks of sessions, we began more in-depth projects with each of the two groups of children.

In the group of four children, I encouraged them to think about a project that would use more than one Logo Brick. The previous series of experiments involved one Turtle performing a certain behavior, with a fairly fixed environment. I expected to encounter interesting issues in projects that involved interactions between two Logo Bricks.

4.4.1 The Baseball Game

After discussing several ideas, we decided upon a "Baseball Game," in which a Programmable Turtle would be the base-runner, and another Logo Brick would control the outfield.

The children's plan was to have the turtle run around a baseball diamond, with lights mounted at each of the four bases. The Programmable Turtle would run a light-following program that would cause it to head toward the nearest lighted base. A Logo Brick "outfield" would sequence the lights so that the turtle would would head for each base in the proper order.

As the children were discussing the project, I conceived of a new sensor that would be very useful for the project. I then explained this idea to the children.

The project could use a *sound sensor* for signalling from the turtle to the outfield. I suggested how the sound sensor could be incorporated into the project: when the turtle hit one of the bases, it would sound a beep. The outfield Logo Brick would use the sound sensor to "hear the beep," and switch the light to the next base.

4.4.2 The Turtle Racetrack Game

In the other group of (now two) children, I brought in a "Turtle Race Track," with a plastic-tape road and styrofoam cup pylons. I gave the children the task of placing and then programming a series of lamps on the edge of the road so that a light-following Programmable Turtle would stay on the course. In this experiment, I hoped to test further the children's notions about the light-following properties of the turtle, and have the children explore interactions between the Turtle and another Logo Brick.

4.4.3 Two-Turtle Experiments

The experiments described so far took place through December of 1987. We took a break from the sessions and resumed in March of 1988. I then encouraged the children to think about projects involving two or more interacting turtles. I felt that interactions between mobile creatures would reveal even more interesting phenomena than we had seen the the previous projects.

Some of the ideas we generated were:

- A "turtle ballet" with several turtles performing dances, orchestrated by a "turtle maestro."
- Turtle hide-and-go-seek with two turtles.
- Turtle tag, with several turtles. One turtle would be "it," and would chase the other turtles. When it hit one of them, then that turtle would be "it."

We decided to work on a two-turtle version of the tag game. One turtle would chase another turtle, and when the first "caught" the second, the first would beep. The second turtle would "hear the beep," and then would begin to chase the first turtle, which would be trying to escape. The turtles would carry lights and use a light-following procedure in order to find each other.

I suggested to the children that they could use the "brightness sensor" (the threshold light sensor) in this project. This sensor could be used to help the chasing turtle know if it had hit the other turtle, rather than some other obstacle. After it had collided with something, the turtle could check if the brightness sensor were on, in which case it would be likely to have hit the other turtle, which would be emitting light.

4.4.4 Braitenberg Experiments

In my last session with the Hennigan children, I brought them a turtle equipped with the Braitenberg apparatus described earlier. I wired the turtle for two different behaviors – follow and avoid – and asked the children to experiment with the turtle and describe its behaviors. I encouraged them to make comparisons with the Logo Brick turtles.

I expected the experiment with the Braitenberg turtle to test the robustness of the children's theories about the light-following process, and to show if they could generalize from their experience with the Logo Brick turtles.

4.5 Description of Experiments: Newton Children

These sessions were led by me, with assistance from Philip Ivanier. They took place in a computer classroom at the Newton school, with a group of six boys in the seventh and eighth grades.

4.5.1 LogoWriter Screen Robots

This activity was intended as an introduction to robots and robotic sensors, prior to work with the Logo Brick and Programmable Turtles.

I suggested that the Logo screen turtle could be thought of as a robot. It had several sensors with which it could explore its environment (the Logo graphics screen). We would focus on the use of `COLORUNDER`, a LogoWriter primitive which reports the value of the color currently underneath the turtle. The turtle could navigate around the screen, and treat one color (black) as "free space," while another color (white) would indicate an obstacle to be avoided.

The next stage of the activity consisted of developing movement strategies for the turtle. Once the children could teach the turtle to sense white space as an obstacle, they could try to teach it to maneuver around those obstacles.

The final stage of the project was a competition. I wrote software to pit the children's turtles against each other in a race. Several turtles played the game at once; the first turtle to make it across the screen, successfully navigating around obstacles, would be the winner.

This screen robot project served several purposes:

1. It gave the children a "Logo refresher."

2. It was an introduction to the ideas of robots and sensors, in a domain (software and screen graphics) in which the students were already familiar.

3. It allowed the children to engage in problem-solving activity as they developed strategies for their turtles.

4.5.2 Programmable Turtles

These children had no prior experience with LEGO/Logo. I planned the first session to be a thorough introduction to the Logo Brick, covering the following topics:

- LEGO/Logo primitives of motor control and sensor input;

- the Logo Brick and the Apple IIGS user interface;

- the Programmable Turtle as a vehicle for Logo Brick projects;
- the Tools Words for ease of interaction with the Turtle.

As with the Hennigan children, I chose a touch sensor project for the boys' first programming project for the Programmable Turtles. I thought that touch sensors would be the easiest for them to understand.

The following session, I planned light sensor projects using the directional light sensor. I gave the boys the challenge of writing programs to make the turtles follow the light.

In the final session, I introduced the sound sensor and beeper, and encouraged the children to write programs to make two turtles interact.

The Braitenberg materials were not tested with the Newton school children.

Chapter 5

Narrative of Experiments

In this chapter I tell the story of the experiments that were done with the children.

5.1 Experiments with Hennigan Children

The Hennigan children began in two groups of four children each[1]:

- Patti, Jeanne, Larry, and Sam
- Elizabeth, Claudia, Bill, and Matt

After one month of work, Bill and Matt left the series of experiments. In the spring, at the beginning of the two-turtle projects, I split the group of four into two groups: Patti and Jeanne, and Larry and Sam. I was not able to meet with the other two girls in the spring because of scheduling difficulties.

5.1.1 Introductory Work

Our first sessions were with original LEGO/Logo. We set up a few machines for the children to experiment with: a LEGO car and a LEGO ferris wheel.

[1]These are not their real names.

The children were given the LEGO/Logo primitives and asked to do some simple tasks, such as making the ferris wheel spin a certain number of times, or making the car rebound when it hit a wall. They reviewed many of the same LEGO/Logo primitives that they would use with the Logo Brick: `TALKTO`, `ON`, `OFF`, `ONFOR`, `RD`, `SENSOR`.

5.1.2 Logo Brick: Motors, Touch Sensor, and Light Sensor

In these sessions the children were introduced to the Logo Brick. They were told that it was a full computer and that to communicate with it, they would connect it to a "big" computer — the Apple IIGS. We opened one up for them so they could see the circuitry inside.

With the Logo Brick mounted on the back of a turtle, the children were shown how the connect it to the Apple IIGS computer. They learned the syntax of the interface while sending motor commands to the turtle, getting it to move in different ways. The first challenge was how to get the turtle to turn, which was solved by either turning on only one motor at a time, or turn both motors on, but in opposite directions.

I asked the children what the turtle could do, using the touch sensors mounted on the front of the turtle. They responded with the obvious thing: Make it turn when it hits something. However, the task of specifying this in Logo was not so easy.

With considerable help, the children came up with a procedure that would make the turtle go backwards for a few seconds after it hit something. Then it would go forwards again.

The children immediately fell in love with the "little animal" that they had just designed. They crawled around the floor chasing it and pressing its bumpers to watch it go backwards. Then they forced it into retreat with their enthusiasm, making it go backwards repeatedly so it backed up underneath a cabinet! Finally they let it get away.

Next we began a series of sessions making use of the directional light sensor. The children began to explore the behavior of the light sensor, using a flashlight I gave to them. They saw that when the flashlight was aimed from the left-hand side, the left-mounted lamp would light, and when the flashlight was aimed from the right-hand side, the right-mounted lamp would light. Also, they saw that if they shined the flashlight head on, they could get both lamps to light. Continuing, they found that by using their hands to *block* the overhead room lighting, the "other-side" lamp would light, indicating that the other side was brighter, which was the case.

I asked the children if they could think of any uses for the light sensor. Patti, remembering how the turtle would crash into things and then back away from them, said, "It [the turtle] might crash into it [a light], and then when it senses the light it heads away." In the other group, I gave the suggestion of using the sensor to follow a light.

I helped the children towards a procedural description of how to follow or avoid the light, by asking: "Suppose the light's here, coming in from the left (I pointed the light from the left, to light the left lamp), what would we want the turtle to do? And then, if the light is coming in from the right, what would we want the turtle to do?"

Soon the children saw that when the light was coming from the left side, we would want the turtle to move towards the left, and when the light was coming in from the right, we would want it to move to the right. The final step was to see that these actions would have to be *repeated*, so that the turtle would continually check whether it should take a step to the left or a step to the right. The resulting program we made was essentially this:[2]

```
TO FOLLOW
  IF LEFT.BRIGHTER? [STEP.LEFT]
  IF RIGHT.BRIGHTER? [STEP.RIGHT]
  FOLLOW
END
```

The children began to explore the behavior of the turtle using the flashlight, their hands (blocking the light sensor), and a few hand-held lamps. With four children sticking objects into the turtle's "nose" there was total chaos, and no one could tell what was going on. So we told the children to use only one lamp, and see if they could tell what the turtle was doing.

Soon they found that it was avoiding the light; that it was turning in the opposite direction that the light was held. They changed the program so that it would turn on the other motor depending on the sensor, and tried again.

Over the next several sessions, we asked the children to consider various modifications to the turtle's environment, while keeping the turtle's program the same. What would happen if there were two lights on the table, both of equal brightness? Which one would the turtle follow? What if one were closer than the other, or if one were brighter than the other? The children were invited to think aloud about these problems, and then run the experiment to see what the turtle would do.

One of the children, Bill, saw that once the turtle got closer to one of the two equal-brightness lamps, it would then follow that one. But Matt

[2]I have modified the copy of the program for readability but the content is the same. In our first versions of this program, the sensor words LEFT.BRIGHTER? and RIGHT.BRIGHTER?, and the motor words STEP.LEFT and STEP.RIGHT were done by specific commands to numbered sensor and motor ports.

explained that the turtle would follow the "real light" while ignoring the "fake one." He was more mystified as to how the turtle chose which light it would follow, attributing it more to a random selection.

When asked how to make the turtle turn in a circle, Elizabeth invented an interesting way of modifying the turtle, using hardware rather than software. She attached one of the small battery-lights, which she had been holding in her hand, onto the turtle's "ear" — adjacent to the light sensor. With the light constantly shining on the light sensor, the turtle kept turning towards that side, and made a circle.

Then she made the light flash, so that when it flashed on, the turtle would turn in that direction, and when it flashed off, it would follow the room light. The turtle manifested a very random wandering effect.

5.1.3 The Baseball Game Project

Over the course of several sessions, the children constructed a playing-field out of cardboard, paper, and tape, with styrofoam cups for the bases. They mounted two LEGO light bricks (small flashlight bulbs encased in plastic) on each base. Meanwhile, back at M.I.T., an undergraduate student and I worked on developing the sound sensor and its matching beeper.

Figure 5.1 shows a sketch of the children's creation.

As the children were building their playing field, they spent time discussing the tasks to be performed by the base-running turtle and the outfield computer. This was a challenging problem, and I suggested that the children role-play the two parts in order to help them understand how the two Logo Bricks would interact.

In the role-play they seemed to use knowledge about how *they* would give messages to each other, rather than considering how the Logo Bricks could

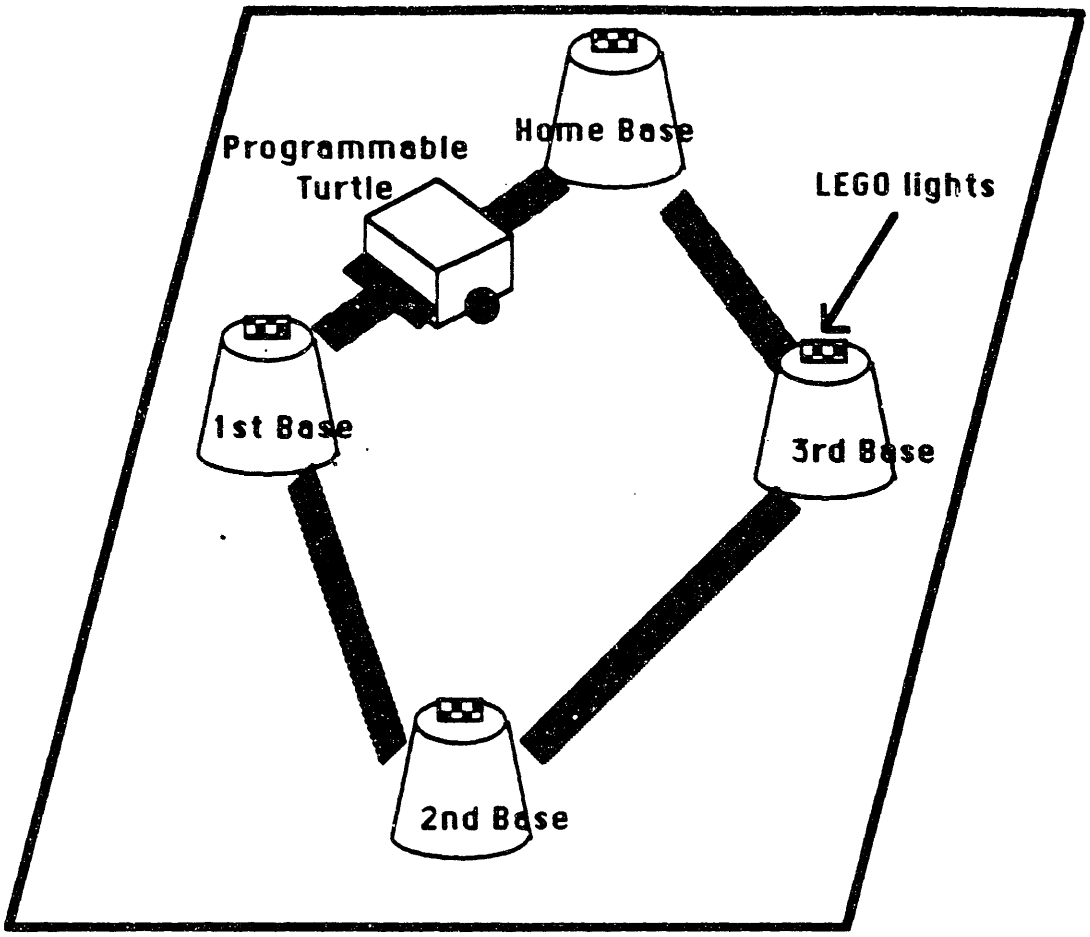


Figure 5.1: Sketch of the Baseball Game Board

receive and transmit information. So it became difficult for the children to formalize their strategies into computer programs.

The key to helping the children write Logo procedures was to use sub-procedures. They first made a main procedure, which read like a script. Later, they wrote the code for the sub-procedures, filling in the details of the script.

5.1.4 The Turtle Racetrack Game

Figure 5.2 shows a drawing of the racetrack board.

This project had two phases. In the first phase, I gave the children five lights and a switch panel to turn them on. The children positioned some of the lights along the edge of the track, and used the switch panel to flash the lights in sequence. It was an easy task to guide the turtle along the path.

In the second phase of the project, the children plugged the lights into a Logo Brick. They then wrote a Logo program to turn the lights on in sequence. But, they paid little attention to the timing.

After discussion with the researchers, the children ran an experiment where they timed the length of time the turtle took to navigate the course from lamp to lamp. Then they converted those measurements into `ONFOR` statements in Logo. The project was eventually a success.

5.1.5 Two-Turtle Experiments

These experiments just got going near the end of the school year. I worked with two groups of the fifth grade children with this project, developing strategies for the "chasing turtle."

Top View

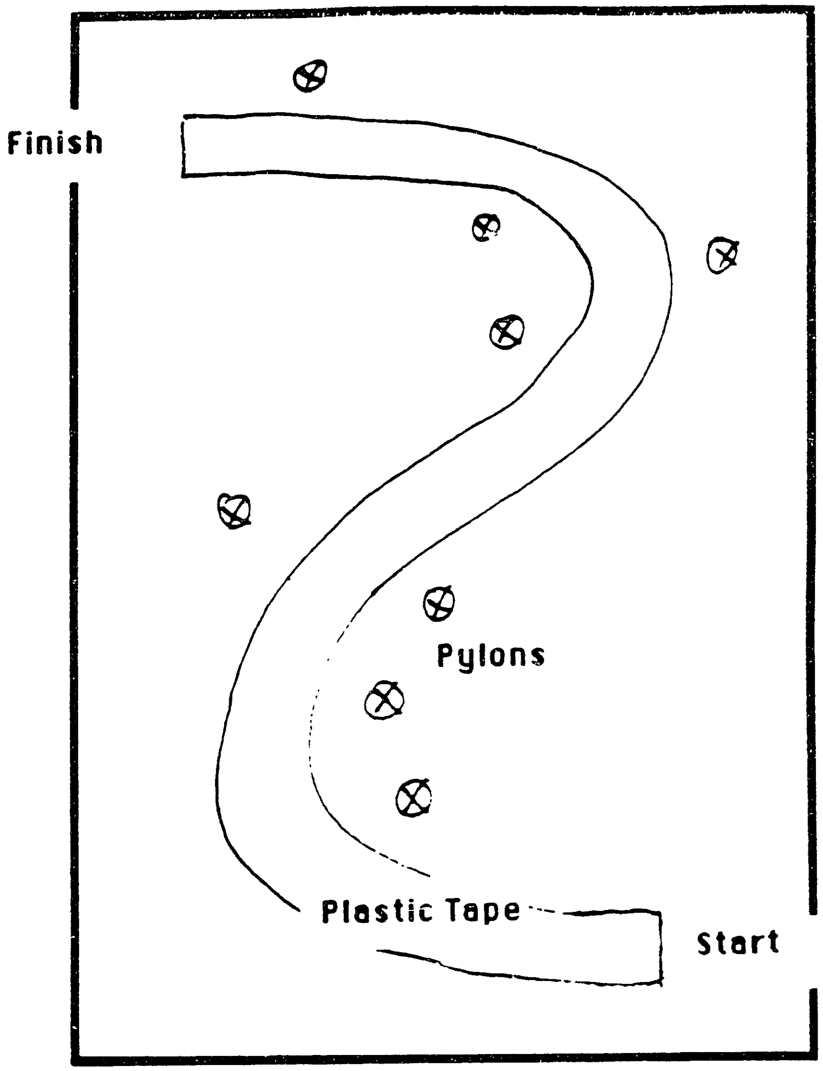


Figure 5.2: Sketch of Turtle Racetrack

As we were discussing how the two turtles would interact, Patti observed that the two of them could use exactly the same program, except that one would be doing the chase part while the other would be doing the escape part.

We had gotten to the stage of being able to "recognize" the other turtle, using the threshold light sensor, when the school year ended.

5.1.6 Braitenberg Experiments

I explained each of the components to the children — the light sensors, the motor control bricks — and how to wire them together (we did not use the inverter bricks). I then showed them how to wire both of a turtle's motors to the same light sensor, making a vehicle like Braitenberg's vehicle no. 1 (Figure 2.1). The turtle would move straight forward, with its speed dependent upon the amount of light that the children delivered to it using a flashlight.

After they discovered the basic relationship of more light to more speed, I wired the turtle for either a "Braitenberg follow" or "Braitenberg avoid" scheme. I asked the children if they could figure out what the turtle would do.

At first it seemed like the turtle was following the light, no matter how it was wired, because any light shined on the sensor would cause the turtle to advance. I asked them to check if it were following the light, and Patti made a comparison to the follow scheme performed by the Programmable Brick turtle. She blocked off one of the turtle's light sensors ("no light") and shined the flashlight directly into the other light sensor. Then she checked if the turtle were turning towards the flashlight or away from it. In this case, the turtle turned towards the light, so she concluded, correctly, that it was following the light.

I asked the children if they saw any other similarities between this turtle and the Logo Brick turtles they had used. Sam commented that "Yes, they are similar, because they follow the light in the same way. But they are different because the Logo turtle you have to program, and this turtle you just wire up."

5.2 Experiments with Newton School Children

5.2.1 LogoWriter Screen Robots

Most of the children had a considerable background in computers, including Pascal, Logo, and word-processing. However, I surmised that most of their experience was of the cut-and-dried programming class variety.

I started the boys off with a review of Logo and and introduction to the special features of LogoWriter. Each of the six boys was given his own LogoWriter disk, so that he could continue work after our sessions.

In the first session we covered the basic turtle movements, and reviewed how to draw geometric objects using `REPEAT` statements. Then I showed them how to use `FILL` to color in their figures. They also asked for the set color primitive (`SETC`), and they learned how to make their own shapes using the shape editor.

I then led a discussion in which we talked about *robots* and *robotic sensors*. I held up one of the LEGO turtles, and pointed out its touch sensors. We named some other devices that had sensors: electric doors that used a weight sensor, door-bells that used a light sensor. Then I told them that the Logo turtle also had a "sensor": the COLORUNDER primitive.

I suggested to the boys that the turtle could use COLORUNDER to navigate around the screen. Suppose *black*, the background color, were considered free territory, but *white* objects were obstacles. When the turtle encountered a COLORUNDER equal to white, then it would know that it had hit an obstacle.

We set off to make procedures that would draw a smattering of white geometric obstacles on the screen. When all of the boys had accomplished this, I gave them a Logo procedure that would cause the turtle to advance slowly, until it hit a white object. Then the turtle would stop.

We called the procedure HUNT:

```
TO HUNT
  FD 2
  IF COLORUNDER = :WHITE [STOP]
  HUNT
END
```

The procedure begins by advancing the turtle two steps. Then it checks if the color under the turtle is equal to the white color. If it is, the procedure exits and the turtle stops moving. Otherwise, the procedure "runs itself" again, causing the process to repeat.

The boys were terribly excited as they each got this working. Sure enough, the turtle would creep across the screen, and as soon as it hit something – presto! – it would stop. It was quite a thrill.

Their assignment between sessions was to continue this project. They were to write procedures to draw a few obstacle patches on the screen. Then they would write a HUNT procedure for the turtle. They said, "This is too easy," and it was, since that was what they had just done in class, so I asked them to think about how to make the turtle *find* the obstacles, not just wander around.

When I came back in two days, the children had been working furiously on their turtle programs. They all had procedures that would cause the turtle to *turn* when it found an obstacle, not simply stop. I pointed out to them how they had each invented different *strategies* for dealing with the obstacles. One child's turtle would turn left ninety degrees when it hit an obstacle, one would turn right one hundred degrees, and another would make an about-face, turning one hundred and eighty degrees.

Now I gave them a new task: Make a turtle that can advance from the left side to the right side of the screen, running around obstacles that it may encounter. I posed it as a race: pretend there were a "finish line" on the right-hand edge of the screen, and the first turtle to get there would win.

The children set to work. Within a half-hour, most of them had invented algorithms that would successfully steer their turtles around square-shaped and circle-shaped obstacles. They all invented variations of the same theme: Go forward, until your turtle hits something. Then move backwards and upwards, and then try to go forward again. (In this case, "forwards" means left-to-right, i.e., towards the finish line.)

At that point, I gave them a new kind of obstacle to think about, one that was not so friendly as a circle or a square. It was a trap: once the turtle got in, it would have to go diagonally backwards to get out (see Figure 5.3).

This obstacle provided a difficult challenge for all of the children. None of them were able to solve it completely; the best solutions were only able

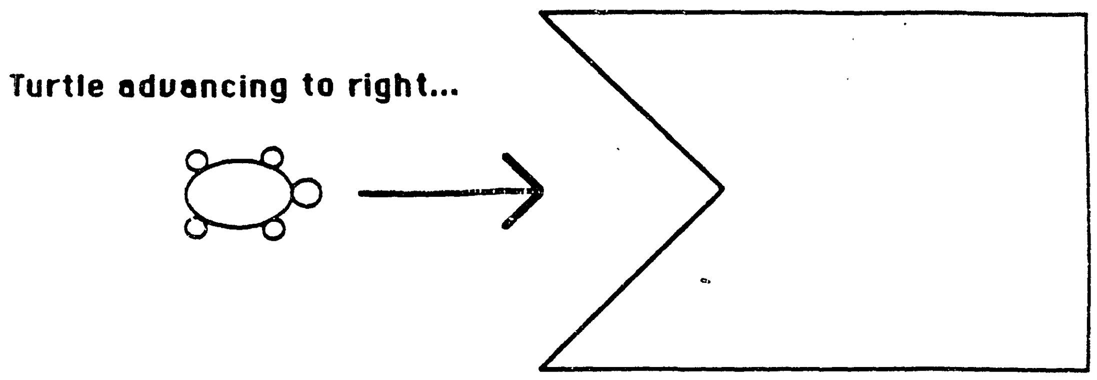


Figure 5.3: Trapping Obstacle

to retreat at some diagonal angle to the forward advance. If the wedge of the trap were at an angle greater than the maximum retreat angle, then the turtle would not be able to retreat successfully, and would pass through the interior of the obstacle (see Figure 5.4).

At the next session, I brought in a master race program that would allow the children's turtles to compete against each other. The program drew a race-track with a start line, three obstacles, and a finish line. It loaded the children's procedures as "tools," and assigned each of the children's procedures to one of the LogoWriter turtles.

Up to four procedures could compete at the same time. The master race program called each of the children's procedures sequentially. The children's procedures were allowed to advance their turtle up to four turtle steps per turn. After each round, the master program checked for turtles that might have crossed the finish line or run into an obstacle.

The first turtle to cross the finish line would win! Turtles that strayed

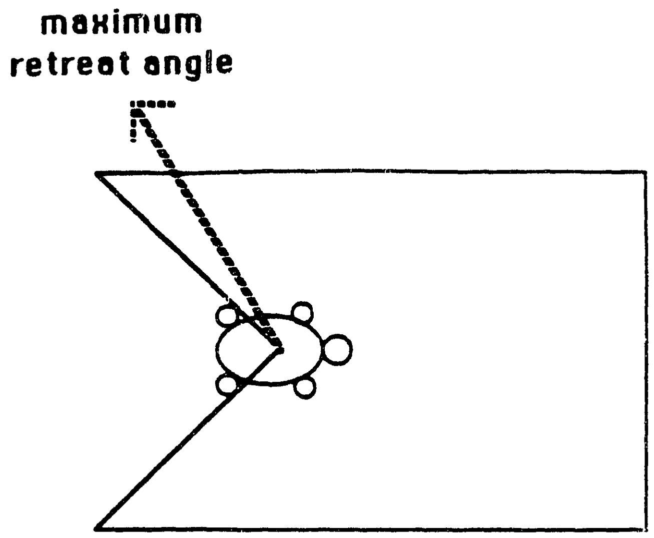


Figure 5.4: Wedge and Retreat Angle

deep in the middle of obstacles were eliminated from the competition. Figure 5.5 shows a picture of the race game screen.

It was a hectic session; children kept bringing their procedures up to me for loading into the computer that was running the race program. But all of them got a chance to watch their turtle in competition. I finished the session by showing them how the race program worked. When I came back next time one of the students had fixed a bug in it.[3]

5.2.2 Programmable Turtles

We began the next session with Logo Bricks mounted on turtles. In a classroom-style discussion, I introduced them to the LEGO/Logo primitives: `TALKTO`, `ON`, `OFF`, `ONFOR`, `RD`, `SETEVEN`, `SETODD`, and `SENSOR`. I also illustrated an if-then control structure using a sensor primitive.

[3]Two of obstacles that my race program would place on the screen were frequently drawn so that they slightly overlapped. One of the boys modified the code to ensure that this would not happen.

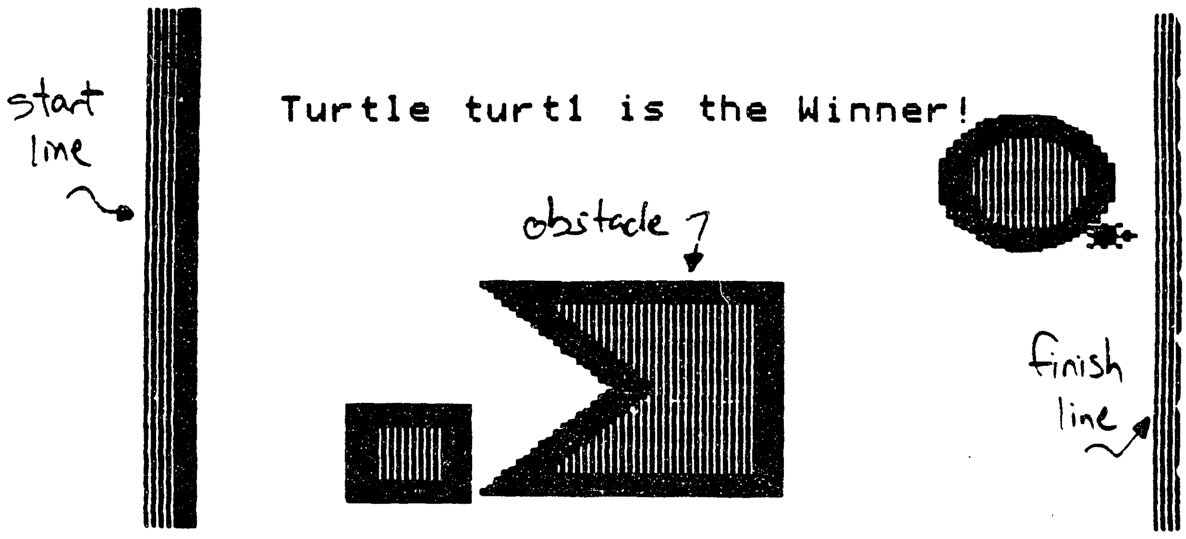


Figure 5.5: Turtle Race Game Screen Display

With one child at the keyboard and the other children watching, we hooked up a Logo Brick to the Apple IIGS, downloaded Logo, and sent the Brick some commands. Their first task was simply to make the turtle move.

After most of them had gotten a chance to type a command to the Brick, I introduced the Tools Words. The boys quickly adopted the use of the new "primitives" for moving the turtle around.

As with the Hennigan children, I started the Newton boys off with a touch sensor project: Make the turtle turn away from obstacles that it might hit. The boys were primed from their LogoWriter robot work and easily wrote a procedure to make the turtle turn back away and turn when it collided with something.

At the next session I introduced them to the directional light sensor (Figure 3.2). I gave them the same challenge I had given the younger children: Can you use this sensor to make the turtle follow a light?

I surmised that the Newton children had a more facile command of the Logo language and had more of a basis to solve this sort of problem that the Hennigan children had had. For this reason I abstained from helping them solve the challenge of writing a procedure to make the turtle follow the light.

The children invented a several strategies before converging on the same one that the Hennigan children used. Their early strategies performed rather poorly, and that encouraged them to find better ones. When they did discover the "check-left, turn left, check-right, turn right" approach, it also performed badly. But, the children saw it had promise, and modified the size of the turtle steps to fine-tune the algorithm into good performance.

I suggested to the boys to have the two turtles chase each other, by carrying lights and using the light sensor to find the other turtle. They were thrilled with the idea and set to work. Two boys worked on one turtle while I helped a third boy with the other turtle (there were only three children present at this session). By the end of the hour, we had turtles that avoided obstacles while hunting down light sources. The turtles performed quite well, banging into each other, backing away and turning, and then finding each other again.

In the final session, I introduced the sound sensor and beeper. The children successfully implemented a communications scheme between two turtles. One turtle would try to find the other turtle, but it would occasionally beep along the way. The other turtle used a light sensor to *avoid* the first turtle. Also, when it "heard" the beep, it went into hiding, turning off its lights and sitting still for a short while. Then, it would "wake up" and continue trying to get away.

Braitenberg turtle experiments were not performed with the older group of children due to lack of time.

Chapter 6

Analysis of Experiments

In this chapter I present an analysis of the experiments described in the preceding two chapters.

I begin with one very special quality I have observed in working with the Programmable Turtles: the human participant's propensity to anthropomorphize the turtles into living creatures. This inclination helped to motivate the children's excitement and involvement in the experiments.

6.1 Turtle Anthropomorphism

I will tell a story to impress upon you how easily we are seduced by small animal-like robots. It is recounted from a scene in the film *Victim of the Brain* by Piet Hoenderdos.

> A woman, who is a technical professional, is taking a male humanist on a tour of a robotics factory. The man is speaking of his reverence for living things, and how he could never imagine treating a machine with the care and respect that he does with animals. The woman takes him into a lab room; on the shelves are a series of small devices, platforms on wheels with tortoise-like shells. She takes one down, flips a switch mounted on its underside, and places it on the floor.

> The device flashes on two red "eye" lights, makes some cute beeping noises, and dances around the floor a bit before coming to rest underneath a table. The man is quite charmed.
>
> The woman reaches on the lab bench and produces a large steel hammer. She approaches the man, and extends to him the hammer. "Now kill it," she says invitingly.
>
> A look of horror crosses the man's face. Kill it? Why should he want to harm an innocent creature...

The children treat their Logo Brick turtles in a similar way as does the man in the film, not wanting to "hurt" the mobile robot.

The human is like the parent of an artificial animal. One of the MIT researchers commented, when the turtle's link to the keyboard-and-screen computer was detached: "It is like cutting the umbilical cord."

After the children have written a program that causes the turtle to back up when it hits something, they began to chase the turtle around on the floor. They forced the turtle to retreat underneath a cabinet, in the same way that an overly-excited child will chase away a house-pet.

This anthropomorphism of the turtle into an animate creature underlies many of the children's interactions with it. Because many children have an affinity for animals, they have strong emotional ties with the "animals" that they create. This leads the children to interactions with the turtles that are meaningful to them and interesting to observers.

Also, the children gain an inclination to talk about the turtle's "behavior," and use psychological descriptions when explaining it. These descriptions allow the children to explain something in a way that is meaningful to them. Using terms they apply to themselves, anthropomorphism becomes a useful style of inquiry.

6.2 Multiple Levels of Interpretation

Using psychological theories to explain how the turtle works are but one of many styles of inquiry. Low-level, mechanistic theories can also be used. Knowing how and when to apply different levels of analysis is a key skill in problem-solving.

The use of the term "level" to express these different avenues of approach is not wholly satisfactory, because it can imply that one level subsumes the previous, which is often not the case. Choice of a certain level depends on the context of the problem and the style of the problem-solver.

The cybernetic approach hinges on distinguishing different levels of analysis. It is therefore interesting to examine the children's use of different levels of explanation in the experiments.

An example is the case of how Matt described the turtle as following the "real light" and ignoring the "fake light." This description gave him a way of explaining a response that he did not have another way of doing.

Later on in the experiments, the children developed more sophisticated behavioral metaphors for explaining the turtle, and also became more fluent in moving between different behavioral metaphors and mechanistic explanation.

When Patti was pondering whether the turtle would sound a beep when its touch sensor were pressed, she spanned several levels within a few seconds of thought:

> "It depends on whether the machine wants to tell ... if *we* want the machine to tell us ... if we *tell* the machine to tell us."

First she viewed the machine psychologically, giving it autonomy and intention. Then she shifted to the intention to that of the programmer.

Finally she concluded that the programmer had control, and could tell the machine what to do.

This playful shifting between levels helped her to come to a satisfying explanation of the turtle's behavior. All of these explanations helped her come to a fuller understanding of the object she was studying.

I think several different factors contributed to the children's ability to use these different levels:

1. The children were encouraged by the M.I.T. researchers to use different styles of explanation to express their ideas.

2. The children became more comfortable with the "lowest level" of Logo primitives: how a Logo command related to a specific physical or symbolic action.

3. The children gained a sense that the turtle had an *internal state*. They realized that it *knew* something (it had an internal state which changed), and that its state could affect its behavior. This suggests the use of psychological metaphors to describe the turtle's "state of being."

In general, "shifting amongst levels" is a very powerful way to appropriate and relate to ideas. Often the different views afforded by different levels allow the learner to connect ideas in novel ways.

6.3 Feedback and Following the Light

The feedback process is a central theme of cybernetics because it is a powerful example of a *system* that involves properties of emergence and organization.

When the children were first guided to write the "follow-the-light" program, they were required to think about the problem mechanistically. That is, they had to break the problem down into two cases, determined by possible states of the light sensor. For each of these two states (left brighter or right brighter), they had to make a rule for the action of the turtle (step left or step right).

The younger children needed help from the researchers to generate this algorithm. They needed reminders or clues to help them re-create the algorithm in sessions after the one in which it was first done.

I think it was difficult for them because the rules did not seem to describe a *process*. When they observed the turtle in operation, the children used process words to describe it. When they said, "It follows the light," they were using a theory of process.

In several of the experiments, the children were asked to predict and explain which light the turtle would follow when presented with multiple lights. Matt's explanation that "The turtle follows the real light and ignores the fake ones," captures the discriminatory behavior of the feedback process, but uses a kind of magic to explain the choice. One of the other children observed that the turtle would wander when it was far from the lights, and then "lock in" when it got close to one.

At first, the children found these two styles of description – the procedural, behaviorist one, and the mechanistic, rule-based one – very disparate. They tended toward the process description – "it follows the light" – and used the rule-based description only when queried by researchers, or when they had to write a Logo program.

Over time the children gained more confidence in their understanding of the mechanistic description. They understood the iterative nature of the feedback process; that the repeated application of the two sensor-to-motor rules would lead to the response displayed by the turtle. The Tools Words helped to make the rules clear to them.

When the children were testing the Braitenberg turtle, they used a mechanical test to see if it would follow the light, and determined that it worked in a similar fashion to the Programmable Turtle. Patti concluded that the Braitenberg turtle was similar to the Programmable Turtle because it used the same sensor to motor rules.

The Newton school children invented several algorithms as they tried to make the turtle follow a light. They needed little or no help from me in order to discover the simple strategy which the Hennigan children used. I think that their work with the LogoWriter Robots developed their problem-solving ability, and that they used body-syntonic thinking (they imagined themselves as the turtle) in both situations.

When I introduced the children to the sound sensor, they generalized the process of following the light by expecting the sensor to help them *follow the sound.* Unfortunately, the sound sensor was non-directional, so this was not possible, but the idea was correct. This generalization, which was made by both the younger and older children, shows a transferable understanding of the feedback idea.

6.4 Multi-Processing and Environment Programming

Single turtles often gave the appearance of multi-processing machines. For example, one of the Newton boys' projects involved two turtles that would chase each other (using the directional light sensor), rebound off of obstacles using the touch sensors, and halt if a tone were detected by the sound sensor.

The Newton boys were using a programming technique, called *time-slicing*, which allowed Logo, a serial language, to appear to do several things at once. The boys built a program loop that would check for several types of sensory input: light, touch, and sound. If any of those inputs triggered the respective sensors, then the boys' program would branch to the proper action, perform it, and then re-enter the loop again. It was as if there were a *demon*[1] attached to each sensor that would fire when that sensor were triggered.

True multi-processing involves more than one independent processor, or at least a programming language that supports multiple interpreters. Examples of the children's projects that used multi-processing are the Baseball Game and the Turtle Race Game.

The Baseball Game project was considerably more complex of the two. The children were very confused as to *who knew what*, that is, which Logo Brick was responsible to know what about the state of the game. For example, the Outfield Brick was responsible to know which base to turn on, but it seemed to the children that the Turtle, who signalled to the Outfield when it hit a base, should know which base were on.

Synchronization between the Outfield and the Turtle was a problem. The Turtle would "think" that it had completed four bases and would stop, but the Outfield had just lit up second base! This surprised the children; they had not anticipated it. This out-of-phase phenomena was picked up by Patti, when she was exploring the turtle tag game: she proposed that both turtles run the same program, out of phase, so that one would chase while the other followed.

[1]A demon is a process that continually runs, checking for a certain condition to be true. If that condition becomes true, the demon activates and executes a corresponding action.

Another phenomenon involves interactions between one processor and a dynamic environment. I shall call this phenomenon *environment programming.*

We encountered an example of this in the early follow-the-light experiments. I had asked the children, "How could you make the turtle go in a circle?" while the turtle was running a follow-the-light program. Elizabeth picked up one of the hand-held lamps and clipped it on the turtle's right side, forcing the light sensor to stay in the "right brighter" state. The turtle started going in a circle.

Then she switched the lamp so that it would blink. When it blinked on, the light sensor would be in the "right brighter" state. When it blinked off, the light sensor responded to the room light to determine its state. The turtle started to wander in a very erratic fashion that was not anticipated by either the children or the researchers. It was a wonderful case of a "hardware program" and a change in the environment having an unexpected emergent effect.

In the Turtle Race game, there were two processors, but they did not interact in an especially interesting way. The interaction that existed was uni-directional: the Light-Timer's actions affected the Turtle, but the converse was not true. It more a case of programming the turtle's environment: the Light-Timer set up a situation for the Turtle to respond to.

Both of these types of interactions, multi-processing and environment programming, convey the importance of thinking about objects in relation to their environment. A turtle's behavior is determined by its program and features of its environment. Either or both of these can be modified to create different behaviors in the turtle. Also, unexpected phenomena may result from interaction between turtles and their environment.

Chapter 7

Conclusion

This chapter discusses future directions for technology development, suggests different styles of experimentation using the Logo Brick, and recommends an approach for incorporating cybernetic study into the classroom.

7.1 Future Directions

7.1.1 Sensor and Actuator Development

In the original version of LEGO/Logo, two sensors were available: a touch sensor and a light sensor. Actuators, or output devices consisted of motors and lights.

New sensors were developed for the Logo Brick, such as the sound sensor and directional light sensor. The beeper for the sound sensor is an example of a new actuator.

Several more devices are presently in development. These are:

- *color directional light sensor.* This sensor will be especially sensitive to a certain color of light, for example, red. It will enable a vehicle to track red lights, while ignoring green ones. Several pairs of light/sensor combinations using different colors will be built.

- *ultrasonic range sensor.* Like the device used in Polaroid auto-focusing cameras, this device uses ultrasound waves to measure distance. When hooked up to a Logo Brick, it will measure the distance from the sensor to the nearest large obstacle. Project examples using this sensor are a maze-running vehicle and vehicles that run around without hitting objects.

- *linear stereoscopic vision sensor.* This ambitious project is being undertaken by Professor Alex Pentland and his students of the M.I.T. Media Laboratory. Using a pair of linear CCD arrays, the sensor will provide a one-dimensional view of the world. Software will be written to perform pattern recognition and stereo matching. Objects in the sensor's viewing span could be identified and distances to them calculated.

- *speech output brick.* This device will allow Logo Bricks to "talk" by transforming English sentences into recognizable speech.

7.1.2 Inter-Brick Communications

The sound sensor and beeper were the first attempt at communications devices between Bricks, and were used in the children's Baseball Game. They provided rudimentary communications at best, but communications nonetheless.

The sound sensor/beeper system was good because it used a signal audible to human observers. However, the range of the sensor was limited to about six feet, and the communications channel was low-bandwidth (it was limited to a very low density of information).

As Bricks can sense more of their world, there will be more for them to "talk about"; that is, there will be more data available to each Brick for sharing with other Bricks. For this reason, we have begun to develop communications devices that have both a long range and a high bandwidth capacity.

Underway is an AM radio transceiver that allows two Bricks to communicate bidirectionally. We expect the device will provide good range and bandwidth; however, if this is not the case, we will consider other transmission media, such as FM radio, infra-red light, and ultrasonic.

Alan Blount, an undergraduate working at the LEGO/Logo Lab, has written Logo procedures that transmit and receive character strings. A procedure called `send` will transmit a Logo word as a series of flashes and pauses, and a procedure called `receive` will decode the information. The `receive` procedure must be running before the `send` begins, which means that a Brick must "hang" while it is waiting for information.

This preliminary version of communications software is for "proof-of-concept" purposes, because concurrent programming issues must be explored before more sophisticated software is written. For example, should a Brick have to wait for incoming information or should it get interrupted when the data arrived?

Mitchel Resnick has studied similar questions in his Master's thesis on *MultiLogo*, a concurrent Logo programming language he designed (Resnick 1988). In this language, separate agents, each with its own (virtual) Logo interpreter, communicated using two different levels of requests: a gentle `ask` and an insistent `demand`. The "ask" primitive's request was done at the receiver's convenience, while the "demand" request interrupted the recipient to get an immediate reply. There also were primitives that instructed the message initiator to wait until it received a reply.

Communications issues become more interesting when one starts to think about *multiple* Logo Bricks in communication. For example, a Brick may want to broadcast a message to all of the surrounding Bricks, or it may want to send a message to one specific Brick. Bricks must take turns talking in some orderly fashion or there will be mass confusion.

It would be a good idea to give each brick a unique name, so that each Brick may prefix its message with it, and name the recipients of the message.

In the best of worlds, all communication styles should be supported with appropriate Logo commands. Metaphors to explain to users (children programmers) the communication methods should be developed to make it easier for them to understand how the communication works.

7.1.3 Programming Metaphors

The Logo Brick is presently programmed in Logo, a serially-executed programming language. Logo is a very explicit and powerful language for organizing and expressing programming ideas.

However, some of the projects we have done with Programmable Turtles have suggested the value of an agent-based concurrent programming language. A good example is the turtle strategies constructed by the Newton boys (described in Section 5.2.2). Their programs branched to a different behaviors dependent upon which sensory input was active. They had invented an agent-like approach within Logo.

Given the processing capability of the Logo Brick, it may not be possible to implement a full concurrent language (such as MultiLogo) for it. However, a language with simpler forms of concurrency may be possible.

For example, a `receive` primitive could be a demon dedicated to receiving incoming messages. When the information would begin to arrive, it would automatically interrupt Logo in order to receive it. When the transmission were complete, the agent would signal to Logo by (for example) running some specially named procedure, or changing the state of a global variable.

Concurrency could be extended by tying sensor inputs to named procedures. For example, a procedure called `SENSOR1` could be run whenever the device attached to the first sensor port changed state.

A wholly different tack can be taken towards the issues of concurrent languages through the use of iconic programming.

Iconic languages are not new. With the popularity of the Macintosh user interface, it is clear that the visual paradigm appeals to a wide range of people. It is certainly appropriate for novice users; there is general agreement that the Macintosh is far easier to learn than a traditional MS-DOS-like machine.

However, there are drawbacks. Text-based languages are more succinct than iconic ones. Certain abstractions, such as sub-procedures and naming, are more difficult in a visual metaphor.

Even within an iconic language, there are trade-offs. An iconic language called *Hookup* (Levitt 1986) is very good at handling parallel signal flows, but not at specifying sequentiality. Other languages might be good at ordering things, but not able to handle concurrency.

Despite these complications, there are special reasons why a visual language would make sense for the Logo Brick. Children's programs are often not very complex; a simple arrangement of icons would suffice for many of the tasks that children like to do. In these cases, using a visual representation would make programming more easy and more fun.

Children often think that sensor stimulation drives the action of a Programmable Turtle; it appears to them that a turtle changes direction *because* they hit its touch sensor. It would be nice to have a programming language that makes explicit their intuitive ideas about such causality. An icon-based language that used wires to connect from sensor to motor would allow for this kind of expression.

7.2 Research Experimentation

The experiments described in this thesis were conducted in a strongly directed fashion. Several other strategies for using the materials have yet to be explored. Choice of these strategies depends on the intentions of researchers leading the experiments.

Because I wanted an in-depth experience with the Logo Brick and supporting materials, I conducted experiments with small numbers of children. Because I wanted to structure the sort of problems the children would encounter, I guided the experiments.

Another research style could place the Logo Brick back into the undifferentiated bin of LEGO parts. Then children could build what they want, and use the Brick when they want. This would not have been possible in my experiments because of the fragility of the Brick, but now it is approaching the stage where this would be possible.

Children are certainly more involved with projects when they are their own constructions. Part of the success of the Baseball Game project was due to the fact that the children built the game-board themselves. Conversely, the Turtle Racetrack game was less successful, partly because I built the racetrack.

More time would be needed for the experiments if children were given more freedom in choosing and constructing projects. It would be interesting to see what children would create given LEGO parts and the Logo Brick without other direction.

Another set of experiments could emphasize children's analytical abilities rather than constructive skills. The researchers could present children with fully built and *programmed* machines and ask them to experiment with the machines and try to explain what sort of program could direct their behavior.

These experiments could progress from using a single Programmable Turtle to a small society of turtles. This would be a natural way to expose emergent phenomena. Interactions that were not observable with only one or a few turtles would emerge when more of them were present.

Both competitive activities, such as the LogoWriter screen turtle game, and cooperative activities pique children's interest. Their game-like aspects make them fun and engaging. As children learn the capabilities of Logo Bricks and Programmable Turtles, I would expect that by consulting them, one would generate a wide range of such amusing and challenging projects. These projects would involve interactions between the creatures in the game, and would be likely to contain high-quality conceptual content.

I would like to see all of these avenues of further study explored, in order to learn more about children's own ways of thinking, and to understand how to best engage them in learning activity.

7.3 Pedagogy

In the popular view, "scientific literacy" means being able to recognize the vocabulary of science, and memorize a collection of definitions and facts. The traditional classroom approach based on this definition has not had much success; in a recent New York Times article, an expert is quoted who claims that only 2.7% of Americans are capable of scientific reasoning.[1]

A strategy of involving the learner in the *process of science* seems more promising. The Technical Education Research Centers (TERC) in Cambridge, Massachusetts have designed a series of experiments for grade-school children based around a *microcomputer-based laboratory* (MBL). Children run experiments using the computer to take data, often from sensors attached directly to the computer. Then they chart and analyze the data. Children use the computer as a tool for experimentation, and are involved in a process of scientific inquiry. Tests have shown the system to be successful when used as intended.[2]

Roy Pea's "Creatures of Habit" has a similar goal. Pea explains that by involving the learner in scientific exploration, the Creatures of Habit system can "encourage a wide range of reasoning and learning central to scientific methodology — pattern observation, hypothesis formation, experimentation, data collection and analysis, and deduction." (Pea et al. 1988) Science projects based on the Logo Brick and Programmable Turtles could take advantage of many properties of the materials that lend themselves well to "inquiry oriented science."

[1]"About Education," *The New York Times*, August 3, 1988.

[2]"Real Science Education," "Emerging MBL Research," and "Coping with Inquiry," in *Hands On!*, a publication of Technical Education Research Centers, vol. 10, no. 1.

Most classroom science is flawed in that it promotes one preferred style — mechanistic and technical — over all others. This alienates many children whose personal styles do not match this one. If children are allowed to use their own styles and approaches to gain understanding, they are likely to feel more capable and be more successful.

Different styles can be used when doing "science" with the Logo Brick materials. The children in my study made use of different styles of thinking in their interpretations of Programmable Turtle behavior. Their use of these different levels of explanation allowed them to gain access to understanding that would have been difficult otherwise.

Also, it makes sense to use different levels of analysis in explaining activity or solving a problem. This is one of the main messages of the cyberneticians. For example, in analyzing an ant colony, one must study the individual ant, how ants in the colony relate to each other, and how the ant colony relates to its surrounding environment. Clearly, one level of analysis will not do; many different models must be used to gain an understanding of the workings of the colony.

These models are often based on the idea of a system, and systems of interacting systems. Other examples of this sort of modeling can be found in animal physiology, environmental analysis, chemistry, social sciences — practically all avenues of scientific inquiry.

Projects using Programmable Turtles provide a concrete model for this principle of "systems thinking," that is, understanding phenomena in terms of systems and interacting systems. In this way, they can help a learner to develop scientific ways of thinking.

Appendix A

The Logo Brick

A.1 History of the Brick

Work on the Logo Brick began in the fall of 1986. The initial design team were (in alphabetical order) Fred Martin, Steve Ocko, Seymour Papert, Mitch Resnick, Brian Silverman, and Allan Toft (on loan from LEGO).

Our first decision was whether to build *fine-grained*, low-level bricks or *coarse-grained*, more powerful bricks. We opted for the coarse-grained approach to allow programmability in software rather than in hardware. Later, we tried the fine-grained approach and designed the set of Braitenberg bricks.

To expedite the design process, we decided to use processor chip that is used in the Apple II. A version of LEGO/Logo, written by Brian Silverman and other members of Logo Computer Systems, Inc. (LCSI), already existed for the Apple II. Brian began the work of converting that code to a version that would run on the Brick, using a Lisp development system that LCSI had built for the Apple IIGS.

In the spring of 1987, we built our first breadboard prototype of the Brick. The first mobile version of the Brick was mounted on a large platform car built by Allan. It carried its batteries behind it on a cart.

This version used a dedicated serial communications chip for interfacing with the Apple IIGS. Using the prototype design, we eliminated this chip by writing software to perform the serial input and output functions.

During the summer of 1987 we began work on the production version of the Brick. Philip Ivanier joined our team at that time.

We decided to build the Brick using a dual circuit board design. One board would house the complete microprocessor circuit, and the other board would contain all motor and sensor circuits.

Our next major design choice was to allocate the sensor and motor ports. The LEGO interface box, used in the original LEGO/Logo system, had ports for three motors and two sensors. Three motors seemed to be adequate for most projects, but it was sometimes desirable to have more than two sensor inputs.

We had a total of twelve I/O lines available for both sensors and motors. Each motor required *two* lines so that the motor could run forwards and backwards.

We chose to implement four motor ports and four sensor ports. We also decided that two of the four sensor ports would support the LEGO optosensor device. The other two could be used for custom sensors or touch sensors.

Reflecting on this choice, I think that it was a good one. Programmable Turtles use two motor ports for their left and right motors, leaving two more for lights and other devices. Their touch bumpers take up either one or two sensor ports (the left and right bumpers can be joined together to make a single front bumper), leaving two free sensor ports for other devices. When we build the communications hardware described in Section 7.1.2, we will use up an additional motor and sensor port, leaving one of each free for other devices.

With the design complete, we proceeded with the production. We chose a LEGO brick that was originally a battery box to house the design. We hired a PC board layout firm to do the circuit layout. Because of our tight space constraints, we used some unconventional layout techniques, such a mounting one IC chip *underneath* another one.

We received the first PC boards in August 1987. Having procured the necessary parts, we built and debugged the first Logo Brick in that month. We proceeded to build a few Bricks, battery packs, and the directional light sensor, for use in our early experiments.

A.2 How to Use the Brick

This section explains how to operate the Logo Brick. It is written as a "Logo Brick User Manual."

A.2.1 Hardware Requirements

The following equipment is required to operate a Logo Brick:

1. A Logo Brick, charged battery brick, and a battery cable;
2. An Apple IIGS computer with at least 1Mbyte RAM;
3. Logo Brick system software;
4. Communications hardware consisting of a *serial line converter box*, and a printer or modem cable.

A.2.2 Setting Up

Begin by plugging the serial line box into your GS's modem port, using the printer/modem cable. If your serial converter box has a switch, select the proper cable type. If not, assume that you have the right cable.

Plug the Brick into its battery box using the battery cable. *Reset* the Logo Brick by pressing its recessed red button.

Boot the Logo Brick software. It will load into the GS, send Logo to the Brick[1], and then place you in the "Table of Contents".

If you get the message `Brick not connected`, see the section on debugging for help.

Select the Logo page you want to use from the table of contents using the *up & down arrow keys*, and pressing <*Return*>. Or, just press <*Return*> over `New Page` to get a blank Logo page.

A.2.3 User Interface

The screen you will see on the GS is divided to two portions: an upper portion and a lower portion. The upper portion is called the *Logo Page*; this is where you compose Logo programs to send to the Brick.

The lower portion is called the *Command Center*. All commands that you type here are Logo words that are sent to the Brick for execution. If the Brick sends a message back, it is printed here. The GS will also print its own error messages in this area.

To move the cursor up into the Logo page, type *Apple-u* (for up). To move the cursor down into the Command Center, type *Apple-d* (for down).

[1]While it is doing this, the message `Changing the Brick's mind` will be displayed.

You use the *Apple* key like a control key: you hold it down, and then type the letter for the function you want.

IMPORTANT NOTE: Make sure that the *Caps-Lock* key is NOT depressed. The *Apple* functions will not work with capital letters!

To send the Logo page to the Brick, type *Apple-b* (for Brick). You can type this when the cursor is either in the Command Center or in the Logo page area. After the page is sent to the Brick, the cursor will return in the Command Center.

The GS does not check that the Brick is there receiving the Logo page. If the Brick stops working, the GS will "hang": the cursor won't return! When this happens, type *Apple-s* (for stop) and the cursor will return, along with an error message.[2]

To initialize a Brick with Logo (after it has been reset), type *Apple-SHIFT-1* (same as *Apple-!*). This tells the GS to send a copy of Logo to the Brick.

A.2.4 Naming and Saving Logo Pages

You can name the page that you are working on in order to save it. To do this, type *Apple-n* (for name). The GS will prompt you to type in a name for that page. It must start with a letter and be less than nine characters long.

To save the page after you have named it, press the *Esc* key (for Escape). If you had given the page an invalid name, an error message will appear at this point.

[2]The error message that you get after typing *Apple-s* is arbitrary and should be ignored.

You can copy a page by selecting it at the Table of Contents, and then changing its name using *Apple-n*. When you press *Esc*, it will be saved by the new name.

To erase a page, type *Apple-e* (for erase). The GS will prompt you for the name of the page to be erased.

A.2.5 Text Editing Functions

The user interface includes several helpful keys for text editing.

Both the Command Center and the Logo page are treated as regions of text. All of the text functions work the same in both of these regions. You will be able, for example, to *copy* text from the Command Center to the Logo page.

To move around text, you simply use the arrow keys. But, when you combine the *Apple* key with the arrow keys, you get different functions:

- *Apple*-$\boxed{\uparrow}$ Move to top of the text region.
- *Apple*-$\boxed{\downarrow}$ Move to the bottom of the text region.
- *Apple*-$\boxed{\leftarrow}$ Move backwards by one screenful of text.
- *Apple*-$\boxed{\rightarrow}$ Move forwards by one screenful of text.

Four keys have been defined to cut, copy, and paste text. These are:

SELECT To begin selecting a region of text, type *Apple-1*. After this, when you move the cursor, the region from where you have selected to the current position of the cursor will be highlighted in inverse video. This region may be *cut* or *copied* using the next two function keys.

CUT To *cut* the highlighted region of text, type *Apple-2*. The text will be removed from the screen and placed into the *clipboard*, which is an invisible place in the GS's memory.

COPY If you type *Apple-3* over a highlighted text region, the region will be *copied* into the clipboard. But it will not be removed from the text buffer, like it is with Cut.

PASTE To paste the clipboard into the text buffer, type *Apple-4*. This leaves the clipboard intact; you can paste multiple copies of the same clipboard.

To move text from the Command Center to the Logo page, you would first select it using *Apple-1* and the cursor movement arrows. Then you could cut it or copy it, using *Apple-2* or *Apple-3*, respectively. Then, you would move up to the Logo page with *Apple-u*. Finally, you paste the text back with *Apple-4*.

There is an obscure bug that can be troubling. Here is a fix when it happens.

The Command Center has only a finite amount of space in its text area. When this space gets filled up, the GS won't print any more characters to the Command Center. If you are running procedures that cause the Brick to do a lot of printing to the Command Center, you are likely to fill up all of the available space.

When this happens, it will seem like the Brick has died, because the GS just stops printing. But then it will seem like the GS is dead too, because you won't be able to type any more in the Command Center!! Clear the Command Center by typing *Apple-c*.

A.2.6 Brick LEGO/Logo

Here is a non-exhaustive list of the LEGO/Logo words in Brick Logo.

Motor Stuff

`talkto` "Talks to" motor and light ports. Examples: `talkto "a`, `talkto [a 3]`, `tto 7`.

`on` Turns motors and lights on.

`off` Turns motors and lights off.

`onfor --` Turns motors and lights on for specified amount of time. Example: `onfor 50`.

`rd` Reverses Direction of motors that are currently being talked to.

`seteven`, `setodd` Sets motor ports for even or odd directions, respectively.

`flash -- --` Flashes light ports. Does not work with motor ports. Give Flash two inputs: first is flash-on-time, second is flash-off-time. Example: `flash 10 20`.

`alloff` or `ao` Turns off all motors and lights, resets ports to "untalked to" state, and resets port polarity to "seteven" position.

Sensor Stuff

`sensor --` Returns value of sensor, as `"true` or `"false`. Example: `pr sensor 3` will print the value of sensor number three.

`level --` Returns the length of a pulse read from the sensor port, as a value from 0 to 255. Example: `pr level 3`.

<u>Logo Control Stuff</u>

`repeat` *num* `[` *<Logo expression>* `]`
  Repeats Logo expression by the number of times specified. Example:
  `repeat 10 [onfor 30 rd]`

`if` *condition* `[` *<Logo expr-if-true>* `]`
  Executes Logo expression if condition evaluates to true. Example:
  `if sensor 3 = "false [talkto "a rd onfor 50]`

`ifelse` *condition* `[` *<Logo expr-for-true>* `]` `[` *<Logo expr-for-false>* `]`
  Executes first Logo expression if condition evaluates to true, or second Logo expression otherwise. Example:
  `ifelse (level 2) > 100 [talkto 3 on] [talkto 3 off]`

`wait` -- Waits for specified amount of time. Example: `wait 30`

A.2.7 Operating the Brick

The Logo Brick has three switches on it. If you look at the end panel of a Brick, you will see one recessed red switch and two other switches.

The recessed red switch is the *Brick Reset Switch.* You must press this switch after first connecting the Brick to its battery. If you are having trouble with your Brick, it is often helpful to press this reset switch and try downloading Logo ("changing the Brick's mind") over again.

Pressing the red switch causes the Brick to lose all of its Logo procedures.

The other two switches cause the Brick to start and halt the running of its Logo procedures.

The middle switch is the *Start Switch.* To make this switch work, you must give the Brick a Logo procedure that has the name **start**. Then, when you press this switch, the Brick begins running that procedure.[3]

For example, suppose you have a procedure called `follow.the.light` that you would like to run when you press the Start Switch. You can make the procedure `start` that will run your procedure:

```
to follow.the.light
  ---
  -- code for follow proc. --
  ---
end

to start
  follow.the.light
end
```

Then, when you press the Start Switch, your turtle will begin to follow the light! (Hopefully...)

The third switch, in the right-hand position when you are looking at the Brick's end panel, is the *Stop Switch.* When you press this switch, the Brick interrupts the procedure or command that it was running, and executes an `alloff` command.

A.2.8 Debugging

If the message "Brick not connected" is displayed, then you have one of the following problems:

1. You have not reset your Brick. Press the recessed red button, and try to send Logo again.

[3]Older Bricks may have a toggle switch in the middle position instead of a pushbutton switch. This is because of an earlier "pause" function that had been implemented. If your Brick has the toggle switch, you should leave the toggle in the right-hand position, and flick it to the left and back to the right to perform the start function.

2. There is a communications problem between the GS and the Brick. Here are some possibilities:

   (a) Make sure that you have plugged the serial converter box into the *modem* port of the GS.

   (b) Your serial converter may not have power. Make sure its batteries/AC adapter are plugged in correctly.

   (c) You may have the wrong type of cable. Try switching the toggle switch on the serial converter box to its other position, or try the opposite kind of cable (the opposite of a printer cable is a modem cable and vice-versa.)

3. Your Brick's battery box is dead. Recharge it.

A.3 Technical Specifications

Figure A.1 presents the technical specifications of the Logo Brick.

Feature	*Specification*
microprocessor	65C802
system clock speed	1 MHz
RAM	32K byte
ROM	2K byte
communications rate	9600 baud, TTL-level standard serial
power supply	logic: 7.2v (1.2v NiCad × 6) motor & sensor: 6v (1.2v NiCad × 5)
motor ports	4 ports × 2 output lines per port 4.5v (nom.), 1A (max.)
sensor ports	2 LEGO optosensor 2 TTL-level input, internal pull-up resistor
switches	system reset (red, recessed) "Stop" (black, pushbutton) "Start" (black, pushbutton)

Figure A.1: Logo Brick Technical Specifications

References

Apter, Michael J. (1966). *Cybernetics and Development.* Pergamon Press, London.

Berg, David & Smith, Kenwyn (1985). *Exploring Clinical Methods for Social Research.* Sage Publications, Beverly Hills.

Braitenberg, Valentino (1984). *Vehicles: Experiments in Synthetic Psychology.* The MIT Press, Cambridge, Massachusetts.

Levitt, David (1986). "Hookup," Vivarium Technical Memo, MIT Media Laboratory.

McClelland, James L, Rumelhart, David E., and the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* The MIT Press, Cambridge, Massachusetts.

Mayr, Otto (1970). *The Origins of Feedback Control.* The MIT Press, Cambridge, Massachusetts.

Minsky, Marvin. (1986). *The Society of Mind.* Simon and Schuster, New York.

Ocko, Stephen, Papert, Seymour, & Resnick, Mitchel (1988). "LEGO, Logo, and Science," *Technology and Learning*, vol. 2, no. 1.

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

Pea, Roy, Eisenberg, Michael, & Turbak, Franklin (1988). "Creatures of Habit: A Computational System to Enhance and Illuminate the Development of Scientific Thinking." Unpublished paper.

Resnick, Mitchel (1988). "MultiLogo: A Study of Children and Concurrent Programming." Master's Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Walter, W. Grey (1953). *The Living Brain.* W. W. Norton & Company, New York.

Wiener, Norbert (1948). *Cybernetics: Control and Communication in the Animal and the Machine.* The Technology Press, New York and Paris.