LOGO MEETS KIDPIX:
A PROGRAMMING LANGUAGE FOR CHILDREN

A THESIS
SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE
OF
MASTER OF SCIENCE IN COMPUTER SCIENCE
IN THE
UNIVERSITY OF CANTERBURY
by

Michael John Hardie

University of Canterbury
1994

To the sparrow that woke me this morning.
May it rest in peace.

Abstract

The use of computers has changed as a result of the shift in emphasis from centralised computer systems to the personal computer. Users are no longer dependent on someone else to supply their information needs, as they can retrieve the information themselves. Rather than making programming a redundant skill, this has meant that more people require programming skills, albeit at a lower level.

The personal computer can now be found in many homes, but its use is mainly for the playing of games, rather than using it to learn the fundamentals of programming—sequencing, selection, and iteration. In order to get children interested in learning the concepts of programming, a skill that can be used in later life, an environment needs to be created in which not only are these skills learnt, but the user has fun learning them.

This thesis introduces sLogo, an icon driven turtle graphics programming environment that incorporates sound and animation to provide an exciting environment in which to program. Within sLogo, direct manipulation is used to insert and delete commands, control execution, and to create procedures and repeat statements. Audio feedback is given on all mouse operations, and turtle movement. Rather than just watching the turtle move across the screen, the user gets to hear it as it zooms along and turns its corners.

We describe some of the problems that children have been observed having when using Logo, and some suggested solutions of those problem. We describe various computer environments designed for use by children are examined, such as KidPix, a drawing program that uses audio feedback, and LogoMation, a Logo-like programming language that supports the use of colour, sound, and animation. The design of sLogo is discussed, and the techniques used to correct some of the problems in Logo when used by young children. SLogo was tested by twenty seven 12- and 8-year-old children from a local primary school. Their use of sLogo, and the observations from this testing are recorded.

Contents

List of Figures

Chapter 1

Introduction

The past ten years has seen a move away from large, expensive, centralised computer centres to personal computers on information workers' desktop. . Along with the shift in the type of computer used, there has been a shift in the use of the computer. No longer does the user rely on some anonymous programmer in another section of the company to supply them with data, but the end-user is now the requester of information, which is often available at their fingertips. The retrieval of the information is usually done through tools such as natural-language query applications, spreadsheets, and databases.

The popular perception of a computer programmer is young male wearing horn-rimmed glasses, writing in a low-level language such as "Assembler", but the very act of creating a macro in a spreadsheet is a form of programming. Programming is based on the use of three basic concepts: sequencing, iteration, and selection. Sequencing is the ordering of steps or instructions so that they are executed one after another. Iteration is the repetition of a group of steps or instructions. Selection is the use of selection commands such as *if* or *while* to make decisions.

These three concepts formed the basis of Babbage's Analytical Engine in 1826, have not changed since then, and seem likely to be the basis of programming in the future. Rather than programming changing, it is the job of the programmer that is changing. The programmer is being replaced by the user gathering their own information using general purpose tools. Rather than eliminating the need for programming skills, these tools mean that more people require these skills, albeit at a lower level.

With computers available in many homes today, one might think that this would encourage the concepts of programming to be learned, but this is not the case. The computer has moved from being something that is programmed to something that resembles a household appliance with the modern computer's "plug-and-play" design. With the development of the microprocessor, the availability of sophisticated electronic devices such as home computer and hand held games machines has increased. Games machines set the standards for sound and action that home computer games had to follow. Although many homes have a computer, it is used mainly for the playing of games.

Compared to playing games, most people find programming boring. Although knowledge of the fundamentals of programming is useful, there is little incentive to learn them when more fun can be had by playing games. One way to promote interest in programming is to have a system that not only teaches the concepts of programming, but is also entertaining and engaging. The KidPix program, detailed in section 4.1, has accomplished this for drawing programs. Unlike standard programs for drawing pictures, KidPix uses sound, colour, and animation to make drawing a fun experience.

In order for more children to be drawn toward programming, the environment used to teach programming must be changed. The standard keyboard and text screen are becoming less acceptable as the children of today grow up with the mouse and graphical user interface. This project is an attempt to create that new environment. It takes the basic commands of the turtle graphics part of Logo and adds an easy to use mouse driven interface, audio feedback, and animation to provide an environment in which the user not only learns the basic concepts of programming, but also has fun doing so.

In chapter 2, *Mindstorms*, the book that introduced Logo to the world, is discussed. The chapter examines Papert's philosophies, the reason for the creation of the Logo language and turtle, and how it may help in the education of children. Also discussed are arguments about the merits and disadvantages of "Instant Logo", a modification of the language where the user presses single buttons to manipulate the turtle instead of typing a command and its arguments.

In chapter 3, we look at a "microworld" used to teach the location of points on a number plane, a "microworld" used to teach Logo, and various environments that have been created to correct problems that young children have been observed to have in learning and using the standard Logo environment. In chapter 4 various systems that have either been designed to be used by children, for example by incorporating sound and colour, or have ideas that may be used in the development of a system for children, are detailed.

Chapter 5 introduces sLogo, a new Logo environment specifically designed to be used by young children. Chapter 6 details how sLogo was created and the problems involved in doing so. SLogo was tested by children from a local primary school, the results and observations of this are written in chapter 7.

Chapter 2

Logo

The "Logo" programming language was popularised by Papert's 1980 book "Mindstorms"[28]. In this book he explains the reason for its creation and how he believes that the use of Logo can help in the education of children. He sees Logo as being a tool by which we can remove the cultural barrier that makes science and technology "alien" to most people. The barriers, he explains, can be obvious, such as deprivation and isolation, or can be subtle, such as the classification of people in to arts or math inclined. It is this latter barrier that he addresses.

Papert sees the majority of people as being "Mathophobic", that is they are afraid of math or anything seen to be math. Being classed as non-mathematical generally results in the person avoiding anything seen as being math and science. Papert sees Logo as a possible tool with which to correct this.

2.1 Mindstorms and Logo

Traditionally computers have been mainly used for computer-aided-instruction, where the computer programs the child. Papert sees Logo as being a tool by which the child programs the computer. He sees programming as nothing more than communication with a computer using a language (Logo) that both computer and human can understand.

Because learning is something that children do naturally, Papert sees no problem with teaching children to program. When problems do arise, it is because of the educational environment rather than the topic itself. Papert introduces two main ideas in Mindstorms. First, it is possible to design a computer that is easy and natural to communicate (program) with. Second, learning to communicate with a computer may change the way in which we learn other things.

The aim behind Logo is to make a system where children find it natural and enjoyable to communicate with the computer. Because the computer is a "mathematic speaking entity," Papert sees this natural use resulting in children learning mathematics as a living language. He sees this as "talking mathematics" in "Mathland".

Papert questions the assumption that children cannot learn geometry until "well into their

school years". He questions their ability to learn it even then. He suggests that the problem with teaching geometry can be likened to teaching American children French. If opinions were based on the success of teaching French in American classrooms, then the conclusion would be that it was almost impossible to teach. But this is not the case because children who live in France learn French naturally. Papert's conjecture is that what we now see as being "too formal or too mathematical" may be just as easily learned as a child learning French in France, through the use of the computer.

Papert's philosophy is based on that of the Swiss philosopher Piaget. "Piagetian learning" is learning without being taught. Piaget believes children to be natural builders as they gather a vast amount of knowledge before they enter the formal teaching environment of the schooling system. Papert asks why is that some learning takes place in a spontaneous fashion so early in life, whilst other learning does not take place without formal instruction, which is not always successful. Papert suggests that we look at children as being builders. Given the right tools and materials, learning is enhanced and without those tools learning is inhibited. Papert sees a shortage of suitable tools and materials for learning math. Combined with a culture block toward math, this results in many people have trouble learning anything that they perceive as being mathematics, whilst having no trouble with mathematical knowledge that they do not perceive as mathematics.

Because most people equate education with the classroom, most research has been in ways to improve classroom teaching. Papert believes the classroom to be an "artificial and inefficient" environment created because society has failed in teaching essential domains such as writing, grammar, and math. Papert believes that the computer can be used to teach outside the classroom without the organised instruction currently used.

Papert also sees the computer as a "carrier of cultural knowledge". Many children who have a "love and aptitude" for math have it because the seed of the math culture has been passed down to them from their parents or mentors. Papert says that the absence of "love-of-math" is caused by the "seed" or "germ" required being absent. This creates a cycle as the children without the seed cannot pass it onto their children. Papert says that the cycle need only be broken at one point for to it be broken forever, and that computers and Logo are the tools with which to break it.

When Mindstorms was written, the majority of computers available had "BASIC" built into them. Why would teachers want to use Logo when BASIC was already available. It had become the "obvious" language to teach programming. Why then is it necessary to use another language, when a perfectly acceptable one was already available? Papert equates the use of BASIC to the belief in the benefits of the "QWERTY" keyboard, in what he terms as the "QWERTY" syndrome.

The QWERTY syndrome is the belief that because the QWERTY keyboard has always been used, there must be a sound reason for its design, and people justify its use on that basis. There is a reason for its design, but its no longer sound. Because the early typewriters had a habit of jamming, the most commonly used key were placed on separate areas of the keyboard (circa 1870) [15]. This was not to facilitate typing, but to inhibit it. Slowing the typist meant that the keys were less likely to jam.

BASIC suffers from the same level of misconception. Papert does not believe the idea that BASIC is simple to learn because of its small number of commands. The real reason for the limited number of commands is that because of the primitive nature of early computers, they could not support a more sophisticated language. He says that the low number of instructions does not make it an easy language to learn, but quite the opposite. Because there are so few commands, it is complicated to create even relatively simple programs. Because teachers did not expect the children to do well in learning BASIC, they didn't realise that children have trouble learning it. They saw computers as being for those of high mathematical ability, and the use of BASIC only confirmed their beliefs.

2.2 The Logo Turtle

In the forward of "Mindstorms", Papert writes about how gears and cogs influenced his love of mathematics. He was able to assimilate mathematics with his gears and cogs. His aim has been to create something that children could use to visualise mathematics. That object is the Logo turtle.

In Euclidean mathematics, a point has a position and no other properties. Most people find this difficult to understand without formal mathematical knowledge. The Logo turtle has position plus a heading, and can be shown visually. One can say "I am here and I have this heading" or "I am here and I am facing this direction". Using these properties, children can identify with the turtle—they can "become" the turtle.

Because they can associate the turtle with there own bodies, children can use the knowledge of how they move their bodies and apply that knowledge to moving the turtle. In doing so they learn formal geometry. Control of the turtle is perform with commands from the "turtle talk" language. The basic turtle geometry commands, *forward*, *backward*, *left*, and *right*, are easy to understand and easily applicable to ones own body. For a child to write the required commands to instruct the turtle to create a square, the child needs only to imitate the drawing of a square with their own body, and observe what they did to accomplish the square (*forward left forward left* ...).

Although most versions of Logo used today have a triangular cursor to represent the turtle, the original turtles were floor based objects with wheels to enable movement. The benefit of having both , is that though the floor and screen turtles are different physically, they both respond to the same commands with the same movement, introducing the notion that two physically different objects can be "mathematically" the same, reinforcing the metaphor.

Because learning to control the turtle is like learning to speak, Papert says that it stimulates the child's "expertise and pleasure" in speaking, and because it is like being in command, it stimulates the child's "expertise and pleasure" in commanding. In essence, the child is in command, and takes pleasure from that fact, rather than the usual teaching/learning environment where it is the teacher who is in command.

Using the turtle provides the ability to identify with what is learnt. Papert illustrates this with an example of a child who did not understand the rules of grammar. Verbs, adverbs, nouns, and the like made no sense to her, when she used the computer to automatically generate sentences,

that it became clear. In order for sensible sentences to be generated, it was necessary to separate words and sentence parts into verbs, nouns, etc. As a result of this undertaking, she was able to understand the need for the different classifications. She was able to identify with what she was learning.

Papert sees Logo as being more than just a programming language, he sees it as a way of learning not only how to move the turtle in turtle geometry, but also learning how to learn. An example of this is the drawing of a circle in Logo. Rather than the teacher just showing the child how to draw the circle (*repeat 360 [fd 1 lt 1]*), the teacher should encourage the child to "play turtle". By acting out the movements required for the turtle to draw a circle, the child is introduced not to the direct answer, but to means by which other problems may be solved as well. The child is taught a heuristic procedure for solving problems.

The use of the turtle in a "Mathland" like environment enables the "transfer of knowledge from familiar setting to new contexts". Instead of "computer programming", the term "teaching the turtle a new word" should be used. The child is able to associate the familiar area of learning new words with that of creating procedures in Logo. For example to create a procedure called "square" that draws a square 100 turtle steps in width when called, the Logo code required is:

```
TO SQUARE
REPEAT 4
   FORWARD 100
   RIGHT 90
END
```

Using the concept of teaching the turtle a new word not only enables the creation of tidy programs, but also encourages modularity of function. Papert writes that the mathematician George Polya believes that rather than teaching problem-specific solutions, general methods for solving problems should be taught. One of Polya's suggestions is that the approach should be to check the problem against two basic ideas: evaluate whether the problem can be broken into smaller and more easily solved problems, and evaluate whether the problem can be related to a problem one already knows how to solve. Papert believes that turtle geometry addresses these ideas. The creation of a house can be broken into two smaller problems of drawing a square and a triangle. The drawing of a circle can be related to walking around in a circle. Even though it has been suggested that Polya's ideas be taken up by math teachers, it has not been done due to the lack of good situations in which to exercise them. Papert believes that not only is turtle geometry "rich" in situations in which the Polya's ideas can be used, but that by "playing turtle", these ideas are enhanced.

2.3 Instant Logo

In an article title "Misconceptions About Logo" [29], Papert writes the people often congratulate him for writing a "good" language for children. He retorts that this is not the case and that any language that was "good for children" would "*not* be good for children", but that children deserve

a good language and that "Logo is good for children insofar as it is good for everyone". This is like purchasing a piano suitable for a beginner. The ideal one is not a cheap upright one, but a Steinway grand as the beginner would be more inspired to use it.

He asks why something said to be good for children, is not really good for children. In particular he addresses what he called an over simplification of Logo in its form of "Instant Logo". Instant Logo is a simplified version of turtle graphics where the user, usually a child, presses a certain key to achieve an action. For example, the "f" key may be mapped to the command "Forward 15". With the "Backward", "Left", and "Right" commands similarly mapped, it is possible for a very young child to manipulate the turtle without any formal knowledge of geometry.

Papert sees this as robbing the child of the opportunity of experimenting with numbers that is Logo. Because this mapping often restricts the turn command values, the child doesn't discover what different values do to the left and right commands. Using this system the child may not discover that "Left 90" and "Right 270" produce the same result.

The reason for his opposition to the use of "Instant Logo" is entrenched in the use of Logo he has proposed. In "Mindstorms" his proposal for the use of Logo by children is a continuous and self-directed use of Logo. Children are expected to use it by themselves and discover how to manipulate the turtle themselves without an artificial agenda. He endorses the use of Logo as a tool for self development. This idea is probably acceptable in an "ideal" world, but where a teacher has thirty pupils in the class, this approach may not appropriate. Whichever view is taken there is still a place for "Instant Logo". Rosen [32] writes that even though "Instant Logo" is generally meant for very young children, older children still have a use for it.

An example of its use for older children is to have the children assign an action to every key on the keyboard. This often results in the keys being mixed up, such as "f" being used to move backward. This can lead into the discussion of "programming" [32] ethics and the need for standards acceptable to the group. She writes that the advantages of using "Instant Logo" for young children are immediately obvious. Given the minimum of instruction, five- and six-year-old children can work independently. With the use of cue cards they can construct simple shapes and pictures and have the "satisfaction of feeling in control of their Logo environment".

Rosen addresses the concern that computer activities have not been integrated into the school curriculum, and that by using "Instant Logo", children do not experience the problem-solving skills that are necessary for regular Logo programming. These two concerns are addressed through "lesson cards", used to explore various problems. Lesson cards are one-line questions that the children are assigned to do in class. Example "one-liners" used for five and six year olds are; "how mays F's high is the screen," "how many R turns gets the turtle all the way around". Other tasks include drawing various shapes of various sizes, drawing two similar letters or numbers such as "c" and "o" or "s" and "8" and explaining the differences between the two. More complicated problems are used such as drawing steps, or a tic-tac-toe, squares, and spirals.

Rosen writes that used as a complement to a free draw program where the children are free to draw what they wish, the use of "one-liners" results in three positive results. First, in discussion sessions after the use of a "one-liner", the children discuss what the "one-liner" was designed to

reinforce, such as measurement or direction. The computer is used as a more interesting way to present formal mathematics. Secondly, by discussion, the children exchange ideas on the solution and the process involved in achieving it. Thirdly, the ideas developed in the "one-liners" are transferred to the pictures that the children draw themselves. For example, using an octagon shape taught through the use of the "one-liner" cards as a sun in the child's own picture.

As the child becomes more confident with the computer, they are assigned tasks that are impossible to solve with the current Instant Logo settings and are asked to explain why it cannot be done. From there the values of the key's input are changed to allow more to be accomplished. For example the turn commands may be changed from 30° to 10° allowing the drawing of less rigid angles. After this the child can be introduced to "standard" Logo.

These two articles clearly illustrate the differences between the philosophical view and the practical uses of Logo. Papert's view is that a child should only use a full and unrestricted implementation of Logo. They should use it as they wish, developing their own skills and discovering such things as the use of repeats and procedures. With no room in the school curriculum or resources to allow a child to develop with Logo the way Papert would like, if a teacher is to use Logo in the classroom, then it must build on the concepts taught through the normal teaching methods. It would be difficult for a teacher to use the "discovery method" with a class of 30-40 pupils, and so the use of "Instant Logo" is a realistic alternative. Rather than being restrictive to the child as Papert claims, the use of "Instant Logo" for young children may be beneficial, as it is possible to learn the basics first with restricted commands and therefore with minimal confusion, and then move to a less restricting version as the child becomes more competent and comfortable with it.

2.4 Microworlds

Microworlds are subsets of reality that concentrate on a particular topic. They restrict the users environment to one in which the user is free to explore a topic without distraction from irrelevant or distracting material. The first microworld was the turtle graphics part of Logo itself. Using a turtle, it is a microworld for geometry.

Papert introduces microworlds by explaining the process of learning. There are two important steps used whenever one learns something new: first, relate it to something one already knows, and second, take what is new and make it your own. An example of these principles is what happens when one learns a new word. We often look for the familiar "root" of the word and then construct sentences of our own to practice its use.

Papert then goes on to say that these task are often impeded when the new knowledge contradicts what is already known. Those differences can be either reconciled, discarded, or kept separate. As an example of this problem, Papert uses Newton's law of motion: "a body in motion will, if left alone, continue to move forever at a constant speed and in a straight line". If one actually tries this by, for example pushing a table, then it obviously doesn't happen. As soon as one stops pushing, the table stops moving. For those with sufficient knowledge of physics, it is obvious that this would happen, because they are aware that there is another external force

interacting with the table, namely friction. If Newton's laws cannot be demonstrated, then how can the laws be illustrated? Papert suggests the use of "Microworlds".

A microworld for learning Newton's laws was described by Papert by extending the turtle to obey Newton's laws. He calls this extended turtle a "dynaturtle". The use of a microworld to demonstrate Newton's laws means that it is not necessary for a pupil to have a deep mathematical understanding of the physics involved. Instead, the pupil can command the turtle to action and see the results visually, rather than the abstract mathematical results. Rather than having to wait for pupils to be able to master the mathematics required for the theoretical knowledge that goes with Newton's laws, Papert suggests that using a microworld to introduce younger pupils to Newton's laws may motivate them, and help them understand the mathematics by providing "an intuitively well understood context for their use".

Microworlds' main advantage for teaching is the ability to limit the "universe", that is, a microworld can be restricted to only the topic being studied. In section 3.5, a microworld to explore number planes is described. This is done by having the user enter coordinate positions at which to draw predefined objects such as trucks and clouds. The positioning of the objects is the only thing that the user can do, as the microworld has been designed to do only that. A common criticism of the use of computers, is the amount of irrelevant knowledge required to do even the simplest of things. The use of microworlds helps to alleviate this problem by restricting the user to the problem at hand. By creating a microworld it is possible to introduce people to complex topics without the need for previous mathematical knowledge such as in the Newton's laws example. It also removes the pure physical problems associated with experimentation. In this microworld, using the computer removes the problems usually associated with locating positions and drawing pictures by hand.

Chapter 3

Logo Environments

The typical Logo environments available today is the edit-run-edit environment. Their existence is still largely due to the origins of Logo rather than any inherent benefits for learning Logo. They still use an editor based system for entering the program, followed a separate operation to execute the program.

This chapter reviews various papers about additions to the standard Logo environment, with particular attention to those that mention specific problems encountered by young children in using the standard Logo environment.

Also reviewed are two microworlds. The first, Logoworld, is a microworld for used for teaching Logo itself, and the second, Number Plane, is used for teaching coordinate positioning.

3.1 Designing Logo-like Environments for Young Children

Geva and Cohen introduce a modified Logo environment aimed at young children [14]. Some features have been added to the normal Logo environment, whilst others have been removed. This has been done to correct various problems that the authors perceive young children as having trouble with such as the wrap around screen, recognising the values required for turn and movement commands, and the construction of repeat statements and procedures.

The wrapping around of the screen is a problem for young children because they do not easily understand the way in which the turtle is wrapped. A long line drawn in a diagonal direction appears on the opposite side "in what may seem to the child as an arbitrary position". The authors record that even after six months exposure to Logo, some grade 2 children still could not predict where the turtle would reappear. The authors recommended the introduction of a "NOWRAP" mode in which the screen becomes a *fenced* area preventing the turtle from moving off it. In some systems where such a fence exists, an error is given such as "turtle out of bounds," but does not move the turtle at all. Here it is recommended that the turtle be moved to the fence and a message "turtle hit the wall" be given.

The authors believe that there is a problem with the length of turtle steps in that they are

"too small and intangible". Young children may not even be able to imagine the size of a single step and so tend to use arbitrary values with the forward and backward commands. The authors' recommendation is that the turtle steps movement should be larger and more visible. To determine distances and sizes on the screen, rulers with these units should be displayed. Such a ruler could also be used as the "fence" mentioned earlier.

Three main problems with the turn commands have been recorded by the authors: noticeable movement, turn discrimination, and knowledge of values to input.

In most systems the turn command does not produce any noticeable movement because the movement occurs instantaneously. This means that if turn and movement commands occur together, some children fail to distinguish between them. This is compounded by the turtle's graphic design—the triangle shape usually used makes it difficult to distinguish direction.

The concepts of left and right are ego-related and as such children have no problem determining direction when the turtle is facing north (straight up), but develop trouble when it is facing down, sideways, or diagonally. Playing turtle would fix this, but most primary school age children rely instead on the hit-and-miss strategy of trying left and right until the correct direction is found. Those without formal knowledge of angles are puzzled when they try to understand the meaning of numerical inputs and even after hours of experience in Logo still use arbitrary values, except for memorised inputs such as ninety degrees, the "magic number to make corners," or 45 degrees for creating diagonals. Many children consider the size of an angle in terms of the length of its sides rather than the invisible angle of rotation. This lack of understanding is compounded when the angle of rotation exceeds 360^o. For an angle of 400^o, the turtle only moves 40^o, the remaining 360^o is not shown in any movement. The authors recommend that fixed values for angles be used in the first instance with user defined angles being introduced once the child understands the concept of angles. The turning of the turtle should be made obvious by slowing the turtle's rotation speed.

The repeat command requires that not only must the child contend with the repeat syntax itself, they must also pre-plan the commands to be executed along with the number of times to execute them. The number of repeats may cause confusion because the repeat value is required before the commands themselves. The authors recommend an interactive version of repeat be used during the learning phase, where the child is prompted to "tell the turtle" the sequence of commands, then the number of repetitions.

Without instruction few children learn to systematically pre-plan, program, and debug Logo procedures. The procedure generally used is "post facto" in which after the trial and error approach has accomplished the desired result, the commands are transcribed to paper from where they are re–entered as a procedure using the "to <procedure name>" syntax. The authors recommend an automatic recording facility that records the commands issued by the child while working on the task. This is then used to turn the commands into a procedure.

In order for young children to be able to use Logo successfully, the basic problems listed above should be corrected. As young children's vision is the prominent sense, the changes recommended in this article are most likely to be of some benefit to them.

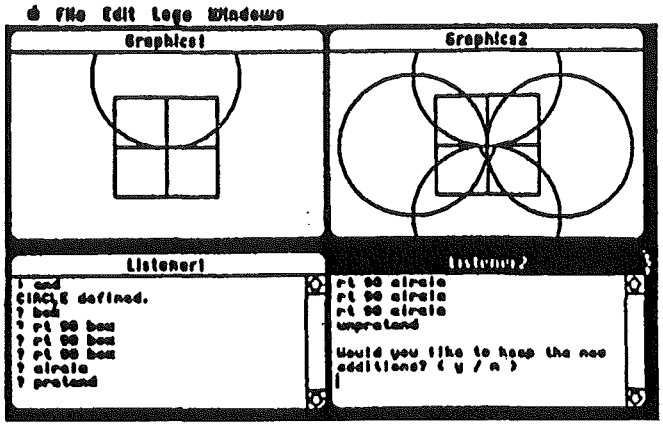


Figure 3.1: The Pretend Extension

3.2 Toward a Student Workstation

Heller has written an extension to the standard Logo environment called the "pretend" mode [16], which allows the user to try out new commands without the fear of those commands ruining any work already done.

Soon after children begin to use Logo, they begin to draw a specific object. Their ability to do so soon falls down as they try to relate their design to the properties of the turtle.

When the turtle points north (straight up), both the child's perception of left and right and the turtles left and right coincide. When the turtle is directed south (or down), the left and right directions of the child and turtle are opposite. As such, the behaviour of the turtle is not what the child expects as they issue turn commands to the turtle. When a child sees an error and does not understand the cause, the child will will erase the work completed so far, and start again.

As a way to address the problem, Heller has extended the standard Logo environment to include a new command—PRETEND. When the user enters the command, a new window opens with a copy of the work done so far. This is shown in figure 3.1. In this new window, the child has the ability to try out further commands without worrying that an incorrectly issued command might "corrupt" the work already done. To return to the original window, all that is required is the issuing of the UNPRETEND command. Upon doing so, a prompt asks whether the new work should be retained. If the child wishes to incorporate the new work with their existing work they answer "yes". If "no" is entered, the new work is discarded. The child may enter and leave the PRETEND environment without restriction.

Most of the current editing environments for Logo use some form of text editor, making the modification of a program difficult. This makes the writing of programs less enjoyable that it need be. The addition of the pretend mode means that users are more likely to try out new ideas as the ability to remove commands if they don't work, has been made easier.

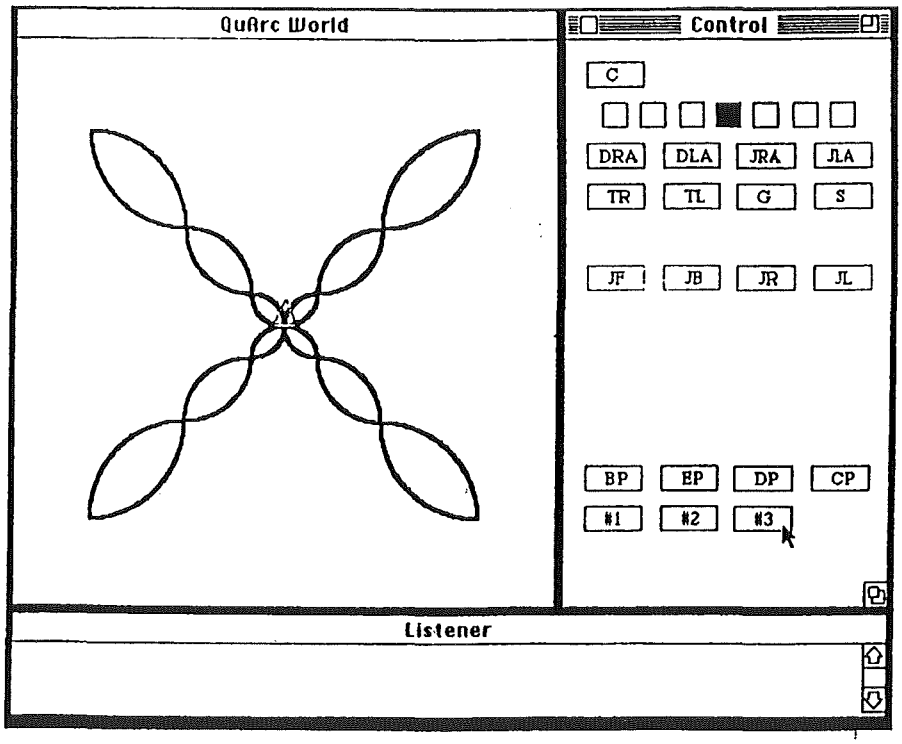


Figure 3.2: The QuarterArc World

3.3 Children, Chunking, and Computing

Graci et al have developed a microworld to introduce point-and-click, bottom-up, and structured programming methodologies to users, without the user having to have knowledge of the syntax of the language [26].

The use of the word chunking comes from "the phenomenon of encapsulating thoughts, actions, and so forth, at a particular level for subsequent use"[26]. Within the area of computer programming "chunking" is the use of procedures, functions. or objects in object–oriented programming. In normal thought processing, the "chunking" of thoughts and ideas is done naturally, but with programming, the creation of procedures requires one to "think ahead".

The use of this microworld is to "blur" the distinction between the "chunking" in thought processes and that of the creation of procedures. The aim of this is that one might capture in code the structures developed though normal thought processes.

The microworld presented in this paper is called a "QuarterArc World" where all the commands manipulate quarter arcs (see figure 3.2).

The environment consists of three windows — the "turtle" output, the control panel where commands are entered, and the listener which outputs the text of the commands entered when required.

The turtle output window is much like all other turtle output windows from Logo programs.

The turtle is represented by a small triangle which moves leaving a "trail" as it does so.

The control window consists of various buttons that are clicked on to manipulate the turtle. The window is easily modifiable and as such the buttons change to reflect the desired "world". The QuArc world has buttons for drawing left and right arcs, for jumping to the left or right without leaving a trail, and for colour changes.

To write a program one clicks the BP (begin program) button. From there one clicks on the required buttons until the desired shape is complete. At this point one clicks the EP (end program) button. Once the EP button is pressed a new button appears with the title "#1". This is now a new procedure. Clicking on this button will reactivate all the commands pressed between the BP and EP buttons presses. Perhaps the button would be better called the Begin Procedure button.

Clicking on the DP (display program) button, lists the procedures defined as Logo source code in the Listener window. Rather than being converted to a standard Logo syntax, the output is in the "QuarterArc" syntax, being made up commands such as "JR", "DRA", a reflection on the buttons pressed to enter the program.

With even the simplest of personal computers today using the mouse as the prominent method of interaction, the QuarterArc world is ahead of the majority of Logo programs in its use of the mouse.

The idea of using a mouse driven system to write logo–like programs is exciting, but where this implementation may fall down is in the design of its commands. Using two letter labels may be confusing to most who use it as the labels present few clues as to the function of the button. This also applies to the "procedures" created—"#1" does not enlighten a user to its function. After a few procedures have been created, the user may be hard pressed to remember which procedure did what.

3.4 Logoworld : A Learning Environment for the Logo Language

Usually microworlds are used to introduce the user to specialist ideas. Logoworld[12] has been designed to introduce the Logo language itself. To make the learning process more personal for the user, they are guided through the task by Turty the turtle who addresses them personally.

There are four stages to this microworld

- Basic Logo commands
- Procedures and recursion
- Processing of words and sentences
- Advanced commands eg Output, If, and Make

The basic Logo commands (FD BK LF RT) are illustrated with Turty then suggesting a square be constructed. After this the repeat command is introduced to shorten programs. The goal

here is to develop a sense of direction and distance evaluation skills. To practice grouping and classification skills, rearranging different size circles into size order is performed.

The use of procedures is introduced by means of teaching Turty new words with the user being encouraged to "build up a little library of programs". Having developed a square and a triangle shape the child is asked to build a house. If it does not work then the child is not blamed, but encouraged to make another attempt. The use of recursion is introduced with various problems being given. The problem illustrated in the article is one where a circle is developed. In the procedure Circle one call each is made to Forward and Right and then to the Circle procedure itself.

The basic commands Print, First, and Last are introduced. Several exercises to develop logical skills are introduced after which the child is encouraged to build sentences from lists of words.

More advanced commands such as Make (assignment) and Output (value return) are introduced with simple examples building up to more complex problems. The command If is introduced and is used to stop the infinite recursion that exists in the Circle procedure created in the recursion section.

Toward the end of the process, the user is asked to perform complex tasks that evaluate how well the user has learned Logo, such as exploring a maze and helping Turty find his home.

Logoworld incorporates the basic ideals for teaching children; personal direction and forgiveness of errors. Though these techniques are usually used when bringing up a child, they are seldom used in teaching children Logo. In Logoworld a special effort as been made to incorporate these techniques in the program.

3.5 Number Plane Microworld

This section examines a microworld developed to explore number planes based on the experiments made by Thompson & Wang [35].

In this experiment two groups of 20 twelve–year–old pupils were each taught about Cartesian geometry by a conventional method. One group then used a Logo microworld to explore specific techniques surrounding the topic, whilst the other continued with normal mathematical tuition.

Tests measuring knowledge of mathematical concepts and the transfer of knowledge of Cartesian coordinates to real life situations were given to both groups. The scores of those who took part in the microworld experiments were found to be statistically significantly higher than could have occurred by chance.

The microworld developed by McMillan [23] builds on the one by Thompson & Wang, but has been translated to a lower level of schooling. The microworld allows one to generate objects that can be located on a two-dimensional plane using two reference points. Such knowledge can then be applied to tasks such as those involving the location of objects on a map and positioning the turtle in turtle graphics with the *setpos* command.

Upon starting the microworld, the user is presented with a 9x6 plane with the turtle positioned in the centre. The turtle's position is given in terms of its xy coordinates, and the user is asked

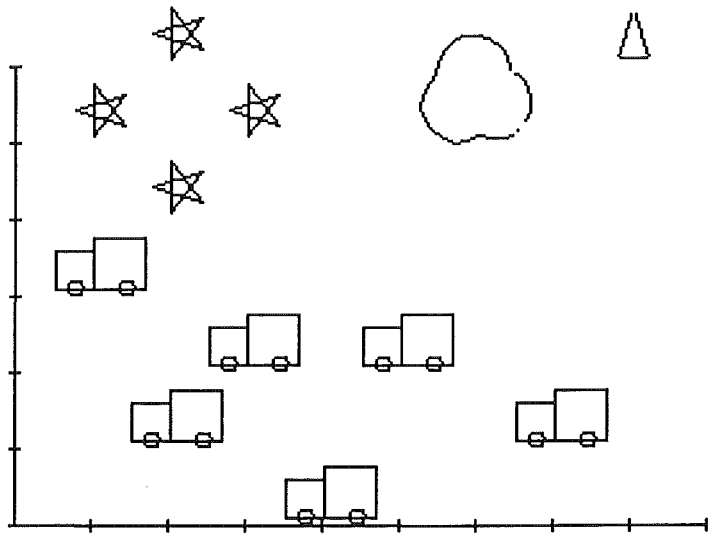


Figure 3.3: Number Plane microworld

to enter two numbers to reposition the turtle. Upon entering the values, the turtle is repositioned and the user is prompted to confirm that the new position is correct. If the position is incorrect the user is again prompted for a new position.

Once the new position is established, the user is asked to choose a picture to draw, either a star, truck, or cloud. Due to the structure of the code, adding new shapes would be a relatively simple task. The desired shape is then drawn. The user is asked whether the picture is what was wanted. Should it not be then the picture is erased. The program then returns to the beginning prompting for a new turtle position.

Though limited in its use, the microworld no doubt has a place in the teaching of children. McMillan concludes his article by recording that the microworld is not the "solution to understanding the concept of locating objects" on a two-dimensional plane, but that it can be a useful "aid and extension" to existing teaching methods.

As a program, the Number Plane microworld is easy to use and robust, features necessary for children's programs.

Chapter 4

Electronic Environments for Children

In this chapter we look at six systems designed to appeal to children and extract key ideas from them. These are:

- KidPix—A painting program that uses sound for feedback,
- Karel—A language used to teach Pascal programming,
- Boxer—A new programming environment based around the use of boxes,
- LogoMation—A Logo-like programming language with the addition of animation and sound,
- Roo & Robby—Programming with animated instruction, and
- Playground—An object-orientated programming system.

4.1 KidPix

KidPix is not a programming language, but rather a painting program. It has been included as it demonstrates how to make things interesting for children. Rather than with ordinary painting programs where output just appears on the screen, KidPix provides an audio feedback with every action taken.

Like normal painting programs, KidPix has various menus items to choose from plus a tool bar down the left of the screen. This can be seen in figure 4.1. That is where the similarity ends.

In KidPix ease of use is considered essential. Rather than an almost unintelligible number of menu options available in most systems, KidPix has only a few, and these are generally not used. These cover the basic file open, close, save, print, and editing options plus a "goodies" menu. Everything else that a child might need is right there on the screen.

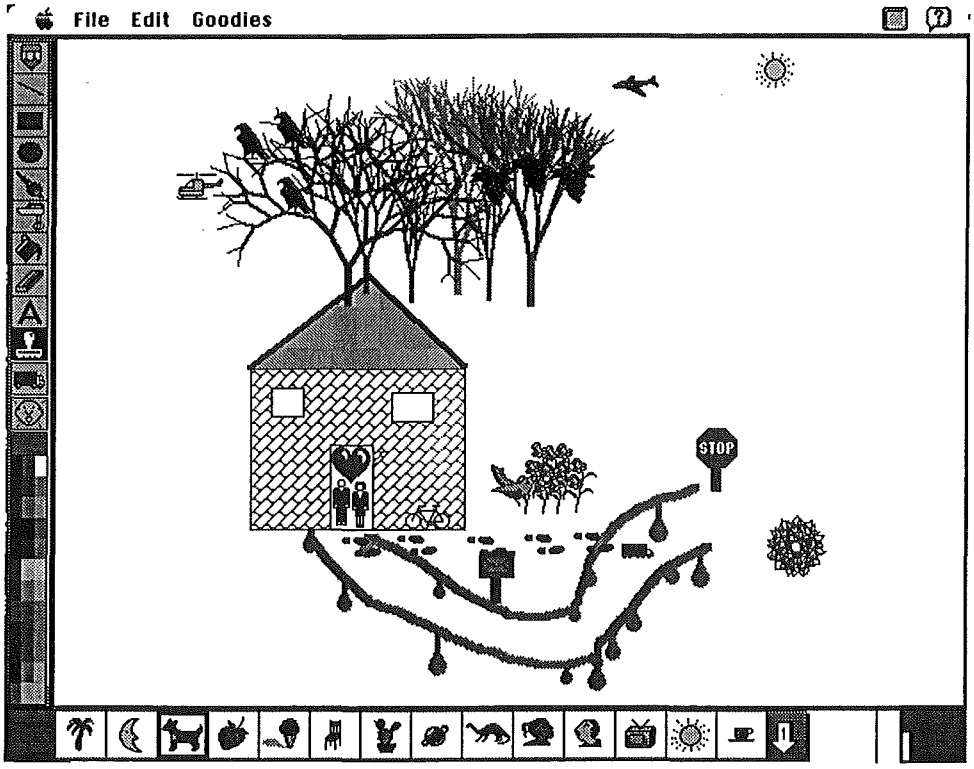


Figure 4.1: KidPix: The author's flat

Everything in KidPix has sound. Every icon selection and every movement with a tool elicits an audio response. Using the *pencil* tool results in the sound of a pencil scraping across paper being heard. The *fill* tool (paint can) has the sound of water being poured into a container. Even drawing something mundane, such as a square or a circle, has has been changed to something exciting. In KidPix, as the size of a shape increases or decreases the emitted sounds suggest that the shape is being increased or decreased with beeps that are synchronised with the actual change in size.

As well as the tool bar on the left of the screen, the bottom on the screen has another set of options. These vary with the type of tool selected from the side. If one selects the *line* tool, then the bottom options include twelve different line widths (lines and blobs) and six colours. The sixth colour is a question mark—its colour and pattern is random, producing some very interesting results. Figure 4.1 has the *stamp* tool selected with some of the stamps available being shown at the bottom of the screen.

The various tools include the usual pencil, line, circles, and squares. To add to the drawing pleasure there is a picture distortion feature with the icon of a food mixer, a paint brush option with which one can draw dot–to–dot pictures, trees, die patterns etc. The fill option, with the predictable "paint flowing from the paint can" icon, gives the standard fill patterns plus also the ever present [?] icon for random results. When one uses the *alpha* tool, the letter of the alphabet selected is spoken. The *stamp* tool gives a impressive 112 different stamps to use—enough to satisfy the needs of almost everyone. Except for the *undo* tool (also selectable from the edit menu), the last tool is the *truck* tool. This tool is used to move selected parts of the picture around hence the use of a truck as the icon.

As a painting program, KidPix has little of the necessary operations to design and maintain complex drawings. As a painting program for children, KidPix would seem to be ideal. The two main advantages of KidPix over other drawing programs must be the use of sound as positive feedback and the easy selection of tools and colours. KidPix is an exciting and easy to use program—a model for all programs designed for use by children.

4.2 Karel the Robot as a Prelude to Pascal

Rather than being a language to program in, Karel[1] is a language used to teach programming. In particular, it is used to teach programming in Pascal.

The language has no variables or data structures, but contains the structure of the Pascal language. The idea is to teach the structure of Pascal—the syntax and the use of BEGIN/END statements.

[1]Karel is named after Karel Čapek, who popularised the word "robot" in his play "R.U.R"

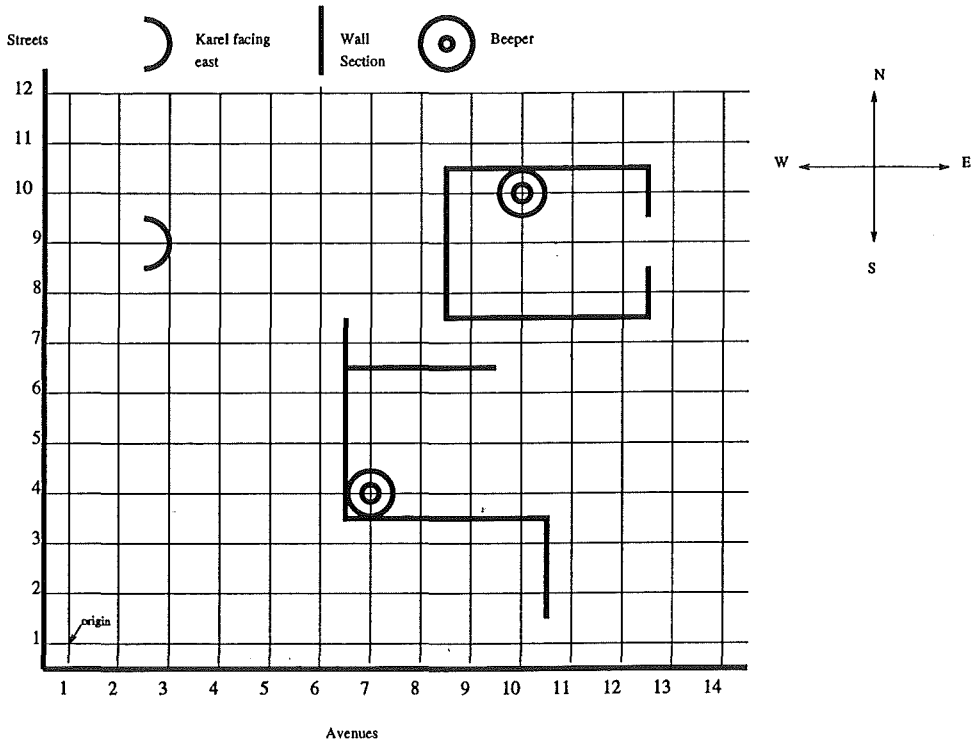


Figure 4.2: Karel's World

4.2.1 The Robot

Karel's world, an example of which is shown in figure 4.2, is a two-dimensional plane bounded by impenetrable walls to the west and south (made from "neutronium"[31]) and infinite space to the north and east. Running parallel to the south wall are horizontal avenues and parallel to the west wall are vertical streets. Karel's movement consists of moving from one corner to another. The streets and avenues are numbered from the origin.

The only other objects in Karel's world are *wall sections* and *beepers*. The wall sections, also made from neutronium and available in any length, are placed between adjacent street corners to block Karels movement. Beepers are "small plastic balls that emit a quiet beeping sound"[31]. The beepers are placed on street corners and can be placed, collected, and carried by Karel.

Being a mobile robot, Karel can move in the direction he is pointed toward (N/S/W/E), determine his direction, detect if there is a wall section half a block in front, left or right, detect if there is a beeper on the corner he currently in standing on, and pick up or put down beepers at the current corner.

4.2.2 Robot Talk

The language consists of three major areas; movement, beeper operations, and tests.

Karel can move one block forward in the current direction with the command *move*. Should a wall section be in front of Karel an *error shutoff* occurs. An error shutoff is the Karel equivalent of

```
{ karel maze demo: robot will follow the right wall until he finds a beeper }

BEGINNING-OF-PROGRAM
DEFINE-NEW-INSTRUCTION turnright AS
   ITERATE 3 TIMES
      turnleft;

BEGINNING-OF-EXECUTION
   WHILE not-next-to-a-beeper DO
      BEGIN
         IF right-is-clear
            THEN turnright
            ELSE
               WHILE front-is-blocked DO
                  turnleft;
         move
      END;
   turnoff
END-OF-EXECUTION
END-OF-PROGRAM
```

Figure 4.3: Karel—Maze Example

a runtime error, though in practice the program just terminates immediately. Within the context of the robot, this error shutoff is performed due to Karel's sense of self-preservation.

To change Karel's direction, a *turnleft* command is issued, which rotates Karel 90° to the left. Any other turns required are accomplished with multiple calls to turnleft.

The two actions performed with beepers are *pickbeeper* and *putbeeper*. When the pickbeeper command is issued, Karel picks up the beeper on the current corner and places it in his bag. Putbeeper is the opposite. Karel takes a beeper from his bag and places it on the current corner.

If pickbeeper is executed when there is no beeper on the current corner, or putbeeper is executed when there are no beepers in Karel's bag, an *error shutoff* takes place.

Karel has ten built-in tests available divided into three groups: directional, movement without collision, and beeper tests.

The robot's current direction can be tested against the four compass points with the facing-<direction> or the not-facing-<direction> tests. Testing for movement collision is made checking the front, left, and right of the robot using the x-is-clear, or with the x-is-blocked commands. Beeper tests consist of next-to-a-beeper, not-next-to-a-beeper, any-beepers-in-beeper-bag, and no-beepers-in-beeper bag.

The use of tests, beeper operations, and movement is shown in figure 4.3. In this example, Karel moves through a maze searching for a beeper.

4.2.3 The Karel Language

Because Karel is used to teach Pascal, it has the basic structure of Pascal (see figure 4.4)

Where Pascal uses procedures and functions as a means of creating new commands, Karel uses the DEFINE-NEW-INSTRUCTION command. This works in the same way as functions and

```
BEGINNING-OF-PROGRAM

DEFINE-NEW-INSTRUCTION <newname> AS
BEGIN
   <instruction>;
   ...
   <instruction>
END

BEGINNING-OF-EXECUTION
   <instruction>;
   ...;
   <instruction>
END-OF-EXECUTION
END-OF-PROGRAM
```

Figure 4.4: Karel Language Structure

```
DEFINE-NEW-INSTRUCTION <new instruction> AS
BEGIN
   <instruction>;
   ...;
   <instruction>
END


ITERATE <positive number> TIMES
   <instruction>;
```

Figure 4.5: Karel: New Instruction and Interation Structures

procedures in that it replaces many commands with just the one. In Karel, the new command is treated as a standard command requiring only its name to invoke it. The structure is shown in figure 4.5.

In order to teach the use of conditionals, Karel has as part of it's vocabulary, Pascal's IF-/THEN/ELSE structure. Given the language's limits, the only valid tests are those mentioned in section 4.2.2

Where Pascal uses the Repeat/Until format for iteration Karel has a Iterate/Times format where the user sets the iteration count to be a set value. The Times format is necessary due to the limited boolean tests available and the lack of variables in the the language.

The while loop in Karel is the same as used in Pascal (While/Do) but with the boolean tests limited as previously mentioned.

4.2.4 Using the Language

The aim of the language is to teach people new to programming how to write code that is structured and robust, without the encumbrances of variables and data structures.

Given a world such as the example give in figure 4.2, the student may be asked to write a

program to guide Karel around whilst accomplishing various tasks as required.

Karel's preoccupation with survival means that any attempt to move into areas that might cause physical harm is akin to requiring the programmer to check things such as function return values etc. The initiation of *error shutoff* if Karel is required to place a beeper when one is not available or to get a beeper when there is no beeper to get is much like the bounds checking required when using such data structures as arrays.

Learning any language can be a frustrating expérience. Presenting a version of the language with variables and data structures removed must go some way in alleviating some of the problems involved.

If there is criticism for Karel, then it must be the long–winded block headings required such as "BEGINNING–OF–PROGRAM". The length of the headings may cause problems with anyone with typing disabilities.

In [31], Pattis mentions a version of the Karel simulator available for purchase. A public domain version has been written by Jan Miksovsky, and is available by FTP (see appendix A.)

4.3 Boxer

Boxer is a new environment based on the concepts of "naive realism" and "spatial knowledge"[11].

DiSessa defines naive realism as "all computational objects will be created, presented, and manipulated in essentially the same way, and the user will be able to pretend the objects themselves are their visual representation". The spatial metaphor is used to exploit "humans'... broad collection of skills for dealing with space".

In the Boxer language almost all objects are represented as boxes. Unlike languages such as *Smalltalk* where there is a distinct difference between searching for a command and then either editing it or using it in a method, in Boxer the very act of using it *is* editing. Because the editor is the universal interface to the system, one is always talking to the system.

The boxes are two-dimensional window-like objects containing lines of instructions. The instructions can be either text such as comments or commands like input and repeat or boxes. Figure 4.7 is an example of a procedure used to draw a polygon. The use of the spatial metaphor is obvious here as the execution sequence of the commands and boxes in the "poly" procedure are represented as a list within the box. The execution sequence is the input of *Number* followed the repeat command. A box can appear on the screen in three states. Closed where it is represented as a small icon with all its information hidden, normal or big enough to show all its contents, or full screen size. In figure 4.6, the *Carpet* box is an example of a closed box and the *Fractals* box is an example of a box opened to fill the entire screen. The *poly* example in figure 4.7 is an example of a box opened to show all its information.

Within Boxer, procedures are represented as boxes (see figure 4.6). To have data local to a box such as needed for a procedure, a local library located in the upper right of the box is used. An

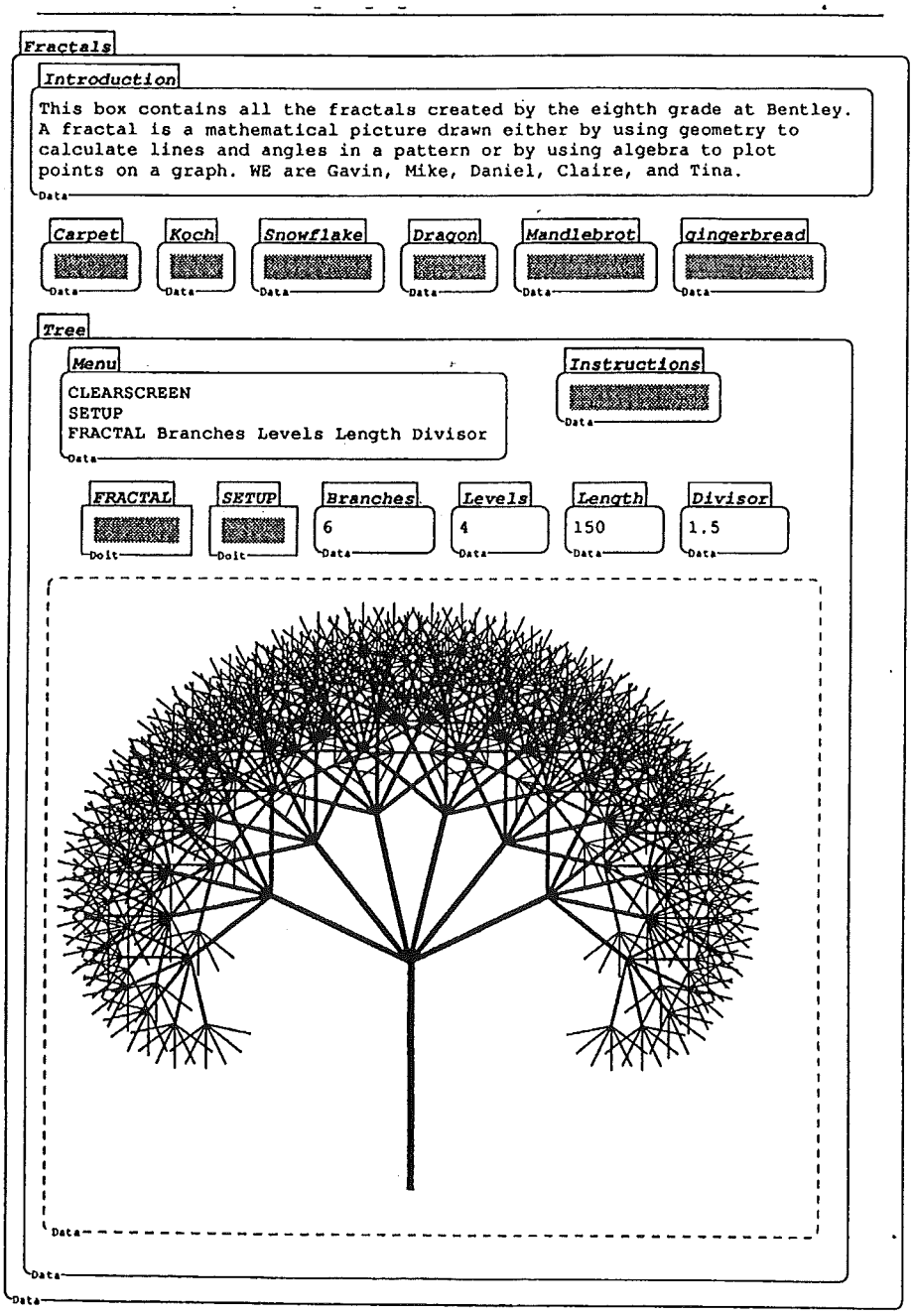


Figure 4.6: An Example Boxer Environment

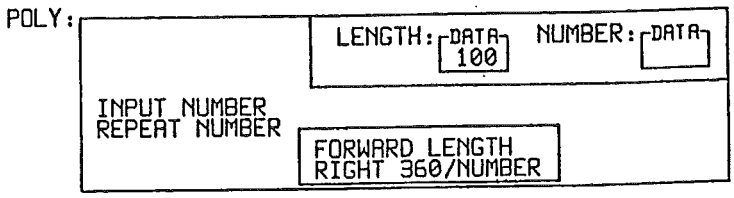


Figure 4.7: A procedure as a box

example of this can be seen in the *poly* example. The local library consists of an unnamed box with *data boxes* contained with it. Scoping is such that objects within the library can be used within the box that contains it, plus any box that is recursively contained within that box. A library has all the properties of a normal box, except that it is not executed.

Because data boxes are a distinct type in that they cannot be executed, they are always labelled. Whereas the majority of languages have different structures for different data, Boxer's text and box structure is intended to be universal. As such any box can be used as input to and output from procedures.

One of the box's implicit features is the ability to be labelled. Addressing of a box is made with the dot (.) notation. For example V.X specifies the X subpart of V. This notation allows for any number of levels to be addressed. To address a two-dimensional array the notation "RC y x *name*" can be used to access the item at row y column x of array *name*. One can also use the "Row y *name*" to address all the items in row y such as a line of text or by using the "Item n *name*" notation to address the *nth* item in the box *name*.

Boxer uses boxes and local libraries to provide an environment, something that has been neglected in computer languages. The use of a environment means that the user has available to them a particular set of actions and options with the minimum of constraints on their use. DiSessa says that any environment "must provide the ability to select and execute easily any set of built-in operations or to define a new operation".

In contrast to the normal situation where a programmer writes an application for the user to run, the Boxer environment is the programming environment thus allowing to user to write and modify their own programs when necessary. The use of boxes as an environment allows the building of large programs with a lower level of complexity. Sub-procedures for example can be tested by moving the procedure box into the current box and setting variables where necessary.

As well as the previously mentioned *data* and *doit* boxes, there are *graphics*, *sprite*, and *port* boxes.

Graphics boxes are quite simply boxes where users can draw and save graphics, with the ability to manipulate them as with any other box.

Sprite boxes are like the Logo turtle with the ability to be moved using commands such as forward and backward. Unlike the turtle, the sprite is represented as a box which contains further boxes containing information such as the "sprite's position, heading, and shape"[20].

Given that the structure of boxes is hierarchical, and that a box appears in one location only,

it is impossible to share boxes. In order for a box that is not directly accessible to be available in the local box, a *port* is used. A port appears as a rectangle in the local box and can be created and removed like any other box. The port is a *view* to the foreign box, and as such allows one to manipulate the objects within that box as if it were the local box. This contrasts with Smalltalk where the user can create a pointer to another object, but that is all it is—a pointer. Removing that pointer still leaves the object intact. In Smalltalk one is not directly addressing the object, in Boxer one is.

As Boxer has just recently become available there are few examples of its use available. It was included for evaluation due to its environment. Rather than being a modification of a language as most new systems are, diSessa has chosen to create an environment—based on the concept of boxes.

As commercial programming moves toward visually based languages such as Visual Basic and Visual C++, Boxer, with its use of "naïve realism" and "spatial knowledge," may become the research base for future programming environments.

Of particular interest to the author, is how Boxer would appeal to children, especially when compared with Logo and its traditional edit–run–edit method of use.

4.4 LogoMation

LogoMation is comprehensive Logo–like language originally written for Yuval Shavit by "Yuval's dad." It has taken the original turtle movement ideas and has added such features as sound and animation, and has connected them together with a Pascal-like language.

4.4.1 Environment

The working window for LogoMation consists of a text editing window along with various buttons (see figure 4.8). The four buttons on the main window are for running the program, beautifying the code, and for providing left and right indentation of the code. The beautify function replaces any abbreviated commands with their full word equivalents and sets the first letter of each command to upper–case. The indentation buttons automatically indents the currently highlighted commands.

LogoMation has also a comprehensive selection of menus. Along with the standard Macintosh file and edit menus, LogoMation has run, tools, and help menus.

The run menu has the run command to start the program. In addition there are options to stop execution and to execute the program in single steps.

The tools menu includes the beautify, and indentation commands plus colour, sound, and picture commands. The "Pick a Colour" selection opens a window with fifty four colours visible at any one time. Using a slider or by entering numerical values directly, the user can access scores more. Clicking "OK" returns to the program with RGB values for the selected colour written into the program at the cursor location.

Selecting a sound or picture to use in your program has been made a relatively simple task using the "Use a Sound" or "Use a Picture" commands. When the menu item is selected the

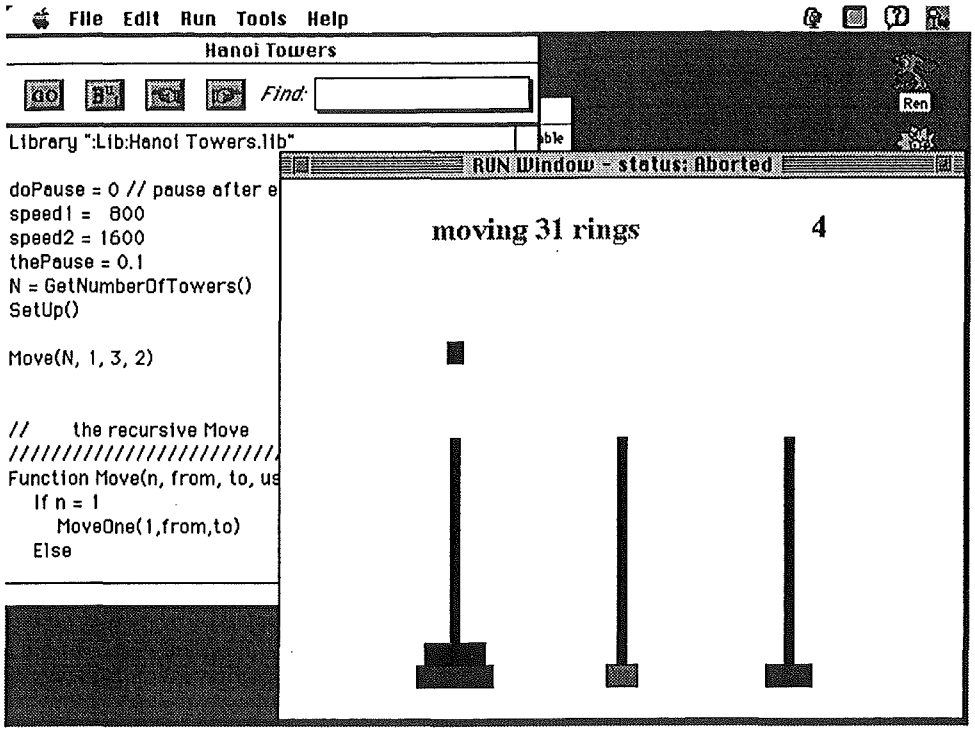


Figure 4.8: LogoMation: The Towers of Hanoi

available sounds or pictures are listed. The user need only decide which one to use. The name is placed into the code in the same way as with the colour selection. Menu options also exist for the importing and recording of new sounds.

The help menu is comprehensive, and is divided in two sublists—Commands and Built-in functions. The Command list includes help for basic drawing, advanced drawing, program control, input/output, and miscellaneous commands plus a list of all available commands. The built–in function list includes help for math, string, pen state, and miscellaneous function plus and a list of all functions available.

All help message are clear and concise. If more detailed information is required then the reference manual supplied in postscript form has all that is required.

4.4.2 Language

Upon running a program LogoMation performs a syntactical check of the code. If an error in the code is found, the offending code is highlighted and a message window is opened notifying the user that LogoMation could not understand the instruction. Like all system messages given by LogoMation the message is supportive of the user rather then being confrontational.

The language has two modes for drawing—straight and curve. When in straight mode (the default), any lines drawn are straight, whereas in curve mode, any lines drawn are arcs. To enter the curve mode, one uses the *curve* command with a parameter. This parameter is the radius of

the arc desired. Like Logo, LogoMation does not have a command primitive to draw circles. To draw a circle one would issue the curve command with the appropriate parameter and then issue a forward (or backward) command for the length of the arc required. To draw a circle, the length would of course be $2\pi r$. To return to straight mode, one issues the *straight* command.

Each LogoMation command has the format *Command [expr1 [, expr2 ...]]*, and must be placed on a separate line.

The basic commands for LogoMation include all the standard turtle graphics commands plus a few more. The sound command supports synchronous and asynchronous sound and would generally be used in connection with the sound selector from the tools menu. The colour command uses the RGB values mentioned earlier. The library command allows functions to defined and stored separately and linked in at run time.

LogoMation supports arrays, string, and floats. Like Logo, the predefinition of variable types is not required. The language has only one predefined variable—π.

As well as the four basic arithmetic operations, LogoMation supports modulo (%), power (^), and string concatenation (.). Relational operators include the basic boolean operations, with all operators obeying the usual order of precedence.

The scope of variables is dynamic and confusing. To quote the manuals, "All the variables are global at any given moment, except the local variables of a given function which shadows the global variables with the same name, if any. If function A calls function B, then B can access A's local variables unless it has local variables with those same names"—confusing enough for a graduate student, let alone a child.

Whereas most languages require program blocks to be delimited by {} as in "C" or begin/end commands as in Pascal, LogoMation defines code blocks by indentation relative to the surrounding code.

Repeat loops are defined by the following syntax

```
REPEAT count
   instruction
   ...
   instruction
```

The maximum number of repeats allowed can be set via the preferences menu option.

The If/Else structure is the same as that found in Pascal.

Functions can return values and are defined by the following syntax

```
FUNCTION name( arg1, ..., argn )
   instruction
   ...
   instruction
```

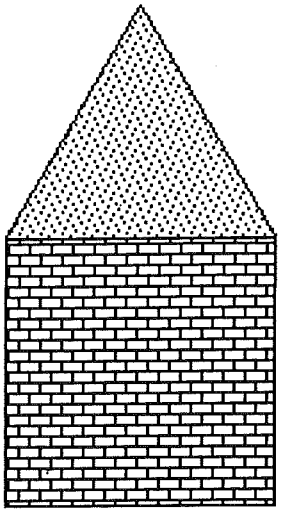


Figure 4.9: LogoMation: The rear of the author's flat

All function return a value, whether explicitly by the RETURN value command, or by an implicit NULL return value.

4.4.3 Miscellaneous

The built-in functions available include such things as pen–font size and style, memory availability, and screen height and width.

The fill pattern command allows the user to choose from 38 different fill patterns. Optional RGB parameters allow the user to choose the colour used in the fill operation.

Within LogoMation, animation is easy. One sets the pen to be a picture and then move it as desired.

```
PICTURE name
   instruction
   ...
   instruction
```

4.4.4 Comment

LogoMation is a language with a rich syntax, allowing for animation, colour, and sound. Though it took the concept of turtle graphics from Logo, the language has been greatly expanded. The syntax of its function calls are more like that of languages such as Pascal rather than Logo, making the transition from LogoMation to other languages relatively easy. This is in contrast to standard Logo whose syntax can be confusing.

```
house(100)

Function house(length)
   Fill 12      // get that brick pattern
      Repeat 5
         Forward length
         Right 90

   Left 90      // get the pen facing the right way
      Fill 10, 4729, 5167, 65535 // A dotty pattern coloured blue
         Left 120
         Forward length
         Left 120
         Forward 120
```

Figure 4.10: LogoMation: Source for the Rear of the Author's Flat

An experienced Logo user might be confused when first using LogoMation as the program does not use the generally accepted command abbreviations such as FD (forward) and BK (backward). Although annoying at first, the author soon found himself using For and Bac without problems.

For the brief period that the author used LogoMation, two problems we experienced. The first deals with drawing the house in figure 4.9. Whereas as in Logo, the user has the turtle to help identify the direction of the pen after the last command, LogoMation does not have that feature. Because of this, the error in the repeat statement in the *house* function (see figure 4.10) was not identified until after much head–scratching. The second problem encountered was creating a copy of the program. LogoMation does not have a means by which a plain text copy of the program can be created. The method by which it is saved is as a binary file. In order for the author to transfer the source of figure 4.9 to a Unix system it was necessary to print the document and save it as a postscript file. A facility for generating plain text files from would be both simple to implement and useful for the user.

4.5 Roo & Robby

Roo and Robby is a package of two components that can be used to teach programming to children, using animated output and function key input.

Roo is a Logo-like environment used to teach "basic algorithmic construction", but without data structures, whilst Robby teaches the use of variables, arrays, files, and stacks. The Roo half of the package is distributed as shareware—it is this that the author has reviewed.

Rather than the turtle used in Logo, Roo has a kangaroo that moves around the screen leaving a trail. To provide a personal touch in the use of Roo, an animated child is used to explain commands to the user or to tell the kangaroo to execute commands. This can be changed from the standard Chris (a boy) to Milly (a girl).

The environment consists of two windows, the program area and the output windows. During

operation, function key information at the top of the screen changes to reflect the current environment. For example, in program mode all valid commands and tests are assigned to the keyboards function keys. This allows programs to be quickly generated without typing.

The language of Roo consists of commands for branching, loops, and subroutines. The basic movement commands are *hop* (move without leaving a trail), *turn* (left 90°), and *step* (move and leave a trail). The only condition tests available are to check that "Roo" can move forward or not. (Roo can/can't move forward)

Both Roo and Robby and have four modes built in to the program— program, game, test, and demo.

Program mode allows one to enter commands as a program. This can be done either by typing the commands directly or by using the pre-programmed function keys. If the mouse is not inside the program sheet area, the commands are not entered into the program but are executed immediately. As the editor has what the author's call "deep syntax detection", it is impossible for an invalid program to be created.

In game mode the users is required to recreate a supplied pattern given the initial position and final result. Two prompts "Hints" and "Compare" are available. Compare allows the pattern and user program to run simultaneously. The use of Hint could not be evaluated as the documentation was unclear about it's function and the author could not initiate it.

In test mode, the user must create a simple program without any assistance from "Chris" or "Milly". Test files can be created from program file using a program option supplied.

In demo mode, the environment uses a file containing explanations of the commands and key codes to explain to users a program in a step-by-step manner.

In addition to the Roo & Robby environments a SchoolWorks package is also available. This consists of "teaching versions" or a word processor, spreadsheets, and database system. Roo & Robby and SchoolWorks are trademarks of Forware Systems, Hungary.

The concept of Roo is appealing, the use of a kangaroo to hop around the screen and an animated child to help instruction is exciting, but using the environment is not. The author found the instructions given in the various modes to be extremely confusing, how a child is expected to follow and understand them cannot be imagined. Should a decision to purchase the Robby and SchoolWorks packages need to be made, then based on the experience of using Roo, the author would not make the purchase.

4.6 Playground

Playground[2] is an object oriented environment that allows users to design objects that have their own ability to sense other objects, cause actions to be taken, do internal processing, and interact with other objects.

The environment consists of four regions, shown in figure 4.11. The top left–hand box is the "agent name list pane", which has a list of possible agent rules (the rules that the object follows) available for each object. The bottom left box holds the basic geometric shapes used in designing

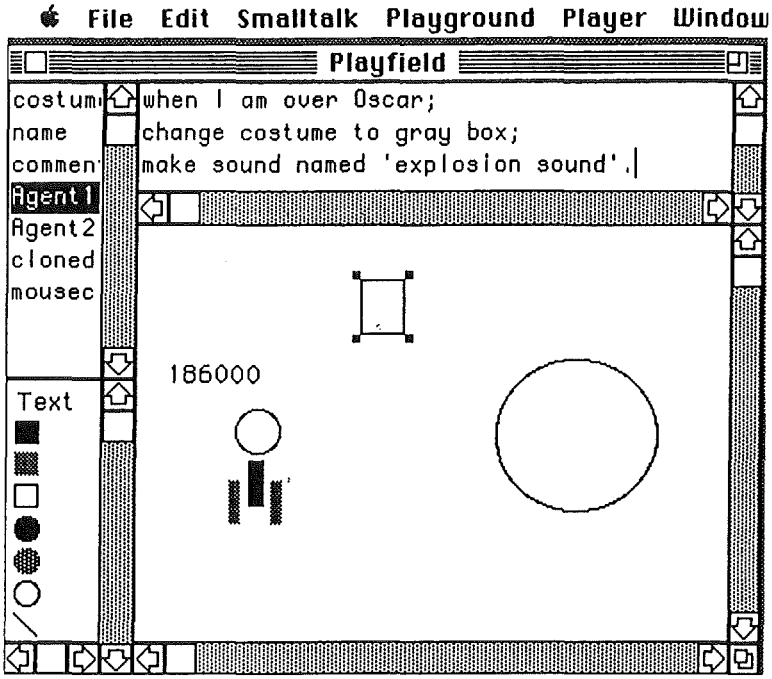


Figure 4.11: Playground: Overview

the physical appearance of the object. The two right boxes contain the agent rules themselves (caption pane) and the object playing field where the user sees the action and interaction taking place.

When a user selects an object in the playing field and an agent in the top left hand box, the rule corresponding to that object and agent is displayed in the top–right box.

Agent rules are the instructions that an object follows given a particular situation. A rule can move or change an object's appearance. In figure 4.11, the user has selected the *square* object and rule *Agent1*. This rules dictates what show happen when this object is in the same location as the object name *Oscar*, it should turn itself into a grey box and emit the sound "explosion". How an object appears on the screen is controlled with the *costume* agent. This specialist agent controls the object's text fonts, size, and colour.

Objects are created using the geometric primitives found to the left of the environment. They may be composed of circles, squares, bitmaps, and text to form a composite object.

Rule creation is accomplished by selecting the object along with an "agent" from the agent name list panel. The user can then edit the rule as required.

The authors claim that using a natural language syntax makes teaching a programming language easier. Rules are defined using an "English-like" description, which are then parsed according to grammar rules and translated into Smalltalk 80 code for compilation. Examples of the language can be seen in figure 4.12.

Playground 2, a version of Playground, was introduced to approximately 60 fourth and fifth

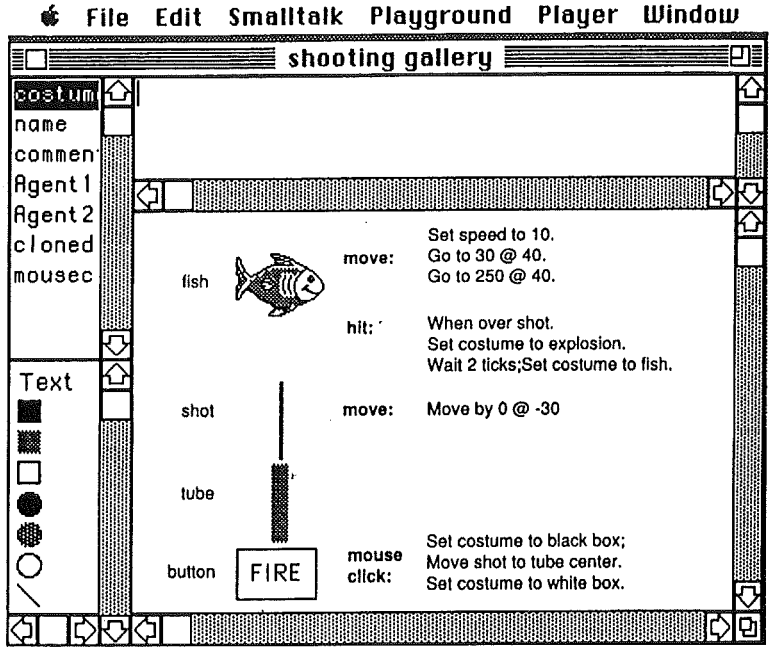


Figure 4.12: Playground: Shooting Gallery Example

grade (10 & 11 years) pupils. The pupils used Playground for one hour, twice a week, for four weeks. Six weeks were planned, but the program was interrupted by a teacher strike.

Each session consisted of a 20 minute period of demonstration followed by a period in which the children completed set assignments.

Each week the children were given specific task to complete. In the first, the children learnt how to run and quit Playground, how to create and modify the graphical objects, how to name and define objects, the coordinate system and how to move objects with simple agent rules. From there, the children learnt low to create random object motion, how change object costumes, and property assignment. Further work involved an agent rule for a "Fish" object so that it could detect food, move toward the food to consume it, detect when the fish had its fill, cloning of an object, and complete life cycles for fish and food, collecting life and deaths statistics as they go.

Due to the teacher strike mentioned, the children were not able to develop a "Shark" to consume the fish, or to add detection and movement to the fish to avoid the predator, nor to construct simulations using Playground.

The authors recorded that the children generally enjoyed using the environment, and that "most" succeeded in accomplishing the set tasks.

4.7 Summary

In this chapter the author has detailed five systems, all quite different from the other. Whereas the application of KidPix to children is immediately obvious, the same might not be said for Boxer, but all the systems have something to contribute to any system developed for children.

KidPix uses sound and ease of use actions to provide a painting program that is exciting to use. Any system developed for children must have these features so as not only to appeal to children, but also to keep them interested. Though Karel is neither an environment aimed directly at children nor a programming language specifically designed for children, its use of a robot and relatively simple programming syntax may be of some benefit in correcting these differences. Boxer's inclusion is due to its environment. It is a possible replacement for the traditional edit–run–edit method of programming used currently. Other than its use of animation to explain concepts to the user, Roo seems to have little that could be used to develop systems for children. LogoMation, being specifically designed for the program's author's son, has very useful syntax checking and a relatively simple syntax allowing a user to quickly produce programs with animation and sound.

Except for Roo, all of these environments have a richness of functionality and include the ability to be extended by the user. KidPix can be extended by adding new sounds and stamps, while Karel, Boxer, LogoMation, and Playground are extended by programming them. When this functionality is integrated with an environment that supports sound, animation, and ease-of-use, a system is created that children find appealing and want to use.

Chapter 5

sLogo

SLogo is an icon driven turtle graphics programming environment, incorporating sound and graphics to provide an enhanced working environment, relative to standard Logo programs. It addresses some of the problems that young children have with the standard Logo environment. Rather than typing, the usual means by which programs are created, sLogo uses the mouse as the primary means of input.

Programs are written by selecting the commands from a "commands panel" and placing them into a "program area," with each selection resulting in a sound being emitted by the system. Rather than the usual triangular shaped turtle in "turtle graphics," SLogo uses a bitmap of a turtle for its representation, which moves across the screen in response to the commands in the program. Not only can one see the turtle move across the screen, but one can see the turtle rotates on its axis when instructed with a turn command. Both the movement and turning of the turtle has sound accompanying the actions. As a command is executed, the icon representing the command is highlighted allowing the user to monitor the execution of the program.

SLogo has buttons available for the user to create procedures and repeat statements from commands already entered into the program, and the ability to start and stop execution at will, and the ability to "print out" the program and the turtle output. The output from the print button includes not only the command icons as displayed on the screen, but also the program written in the standard Logo syntax, so the user can enter it into their own version of Logo.

5.1 The Environment

The sLogo environment shown in figure 5.1 consists of four areas: command selection (the left hand panel), program(the middle panel), control panel (the bottom right panel), and turtle screen (the top right panel).

The command selection area lists all the commands available for use. As well as the standard predefined commands, forward, backward, left, right, penup, and pendown, it also contains any user defined procedures.

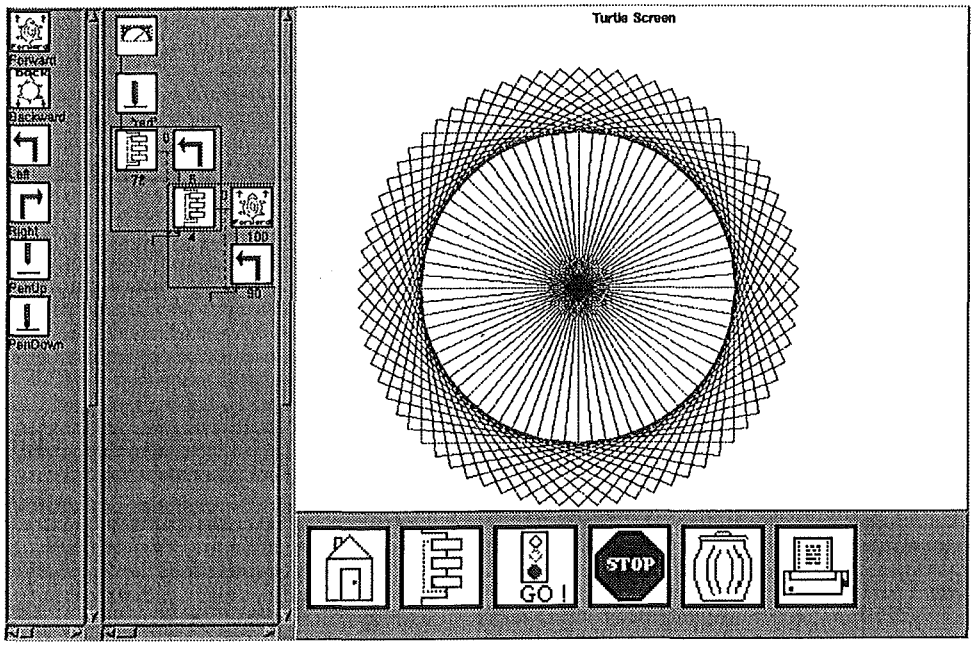


Figure 5.1: sLogo: The Environment

To select a command, the user clicks on the required command. Upon doing so, three things happen. A sound is emitted (each command has its own sound), the command is highlighted, and the cursor changes to the command image shrunk by a factor of a third. When the cursor is moved to the program area, it becomes the full size image of the command. This sequence of actions is shown in figure 5.2.

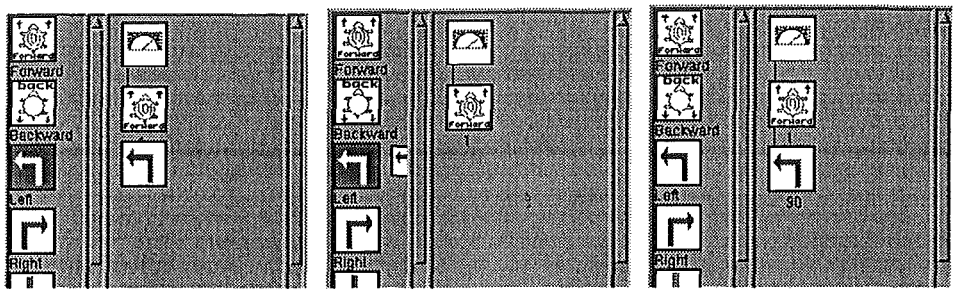
If the number of commands exceed the visible area on the screen, the scroll bars can be used to reveal the rest.

The program area is where the program is built up and where the commands used are displayed. When the user selects a command, the cursor becomes the selected command's image. The command is placed by positioning the image at the location where the user wants the command to go, and then clicking the mouse button. Depending on the command selected, and whether the "instant logo" mode is on, determines whether a dialog box is brought up or not. After completing any entries required in the dialog box, the command appears in the program.

Inserting a command between existing ones is a matter of positioning the top of the icon between the two commands already on the screen, and clicking the mouse. The screen is redrawn with the new command in the desired location.

To delete a command from the program, the user selects the command and clicks on the delete icon in the control panel. The user is then given the choice to delete the command or cancel the delete request.

When a user selects an existing command by clicking on it, the sound emitted is the same as is heard when the command was selected in the command selection area.



(a) select (b) move (c) place

Figure 5.2: sLogo: Inserting Commands

As with the command selection area, the number of items in the program area soon requires the user to use the scroll bars in order to view the entire program.

The control panel at the bottom–right of the screen consists of six button; "make procedure" (the house), "make repeat", "go" (to run the program), "stop" (to halt the program during execution), "delete" (the rubbish tin), and "print". An image of a house was used as the procedure button to imply the "hiding" nature of procedures. To examine its contents, one has only to "knock" on the door. The repeat icon is drawn to represent the iteration of the commands encapsulated within it. The dashed line represents the path that the program takes during iteration, with the solid lines representing the path that the program takes before and after the iteration takes place. The green traffic light and red stop sign were used for starting and stopping of the program because they are road signs that the children are likely to be familiar with. The delete button is an image of a rubbish tin. The use of a rubbish tin to discard something in life supports the metaphor of using the rubbish tin on the screen to delete or discard a command. The print button icon is a simple image of a printer.

When a button is clicked in the control panel, a sound is emitted and the button is briefly highlighted.

Rather than emitting a "cute" sound as commands do, the button's function is spoken. Because the "make procedure", "make repeat", and "delete" buttons require that at least one command in the program be selected, the computer emits an "oops" sound if no selection has been made.

Because of the machine dependent nature of printing documents, the print command does not explicitly print, but creates files that the user can then submit for printing. When the "print" button is pressed, the user is asked for a filename to save the output to. This name is used as the base of the filenames for the three files the print command creates. The first is a colour postscript file of the program icons in the program area, including any procedures defined broken down into their command parts. The second file is a colour postscript version of the turtle screen. The last file is an ordinary text file with the program details translated into the syntax of UCBLogo. Examples of the print command's output has been included in appendix B.

The "turtle screen" is the area that the output from the program appears. It has full colour and so makes use of the four pen colours available. Clicking the mouse in this area toggles the turtle from visible to invisible. This would be done when faster program execution is required as it minimises the amount of screen updating.

5.2 Execution

The usual Logo environment has a triangular shape as its turtle. Execution of programs in such environments consists of the "turtle " disappearing, and lines or curves appearing on the screen as each command is executed. Once completed, the turtle then reappears. This design philosophy means that the user is unable to monitor each command as it is executed and associate it with the corresponding turtle movement. Even when the commands are executed manually, that is the user types in the command and watches the screen for the turtle's action, the movement of the turtle is generally instantaneous, making the correlation difficult.

SLogo has two major differences in its execution from the usual Logo environment: each command is highlighted as its is executed; and the turtle movements are obvious.

5.2.1 Commands

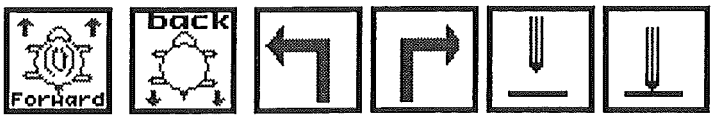
Highlighting commands during execution is achieved by reversing the icon's two primary colours. This provides a visual feedback of the execution and allows the user to relate the command to its effect on the turtle.

The "repeat" is different. As the repeat command is executed, the repeat icon itself is highlighted for the entire period of the loop's execution. The commands encapsulated by the repeat command are highlighted individually as each is executed. This allows the user to "tune" in immediately to the fact that the repeat statement is being executed. When displayed in the program area, most commands have a a value displayed slightly below the icon. For the the repeat command, this is the number of iterations through the loop. For the movement and turn commands, this value is the number of turtle steps to move or the degree of turn. The current iteration count of the repeat loop is shown just to the right of the top of the icon. As each iteration takes place, this counter is incremented.

Like the repeat command, the procedure icon is highlighted for the entire time the commands encapsulated within it are executed. If the procedure is "open", that is the encapsulated commands are visible and able to be edited, the commands within it are highlighted as they are executed.

5.2.2 The Turtle

In sLogò, the turtle cursor is a bitmap image of a turtle, that does not disappear when in motion. When the turtle is commanded to move forward or backward, the movement of the turtle is animated so the user can see the turtle move. When the turtle turns, the user can see the turtle rotate.



(a) For-ward (b) Back-ward (c) Left (d) Right (e) PenUp (f) Pen-Down

Figure 5.3: Predefined Command Icons

Allowing the user to see the turtle's movement allows the user to evaluate the choice of command used. It becomes a simple matter to determine the correct command for any desired effect.

Along with the visual feedback on the movement of the turtle, audio output is also heard. When the turtle moves, the user hears the sound of a truck. When turning the sound emitted is one of screeching tyres.

Standard Logo programs have a "wrap–around" screen on which the turtle is drawn. This causes the turtle to disappear off one side of the screen and appear again on the opposite side. In section 3.1, it was noted that young children often see the location where the turtle reappears as being arbitrary , without any real understanding of why it reappears where it does. Geva and Cohen recommended that a "NOWRAP" mode be introduced. this means the turtle stops if it hits the edge of the screen and notification is given to that effect. This recommendation has been included in the design of sLogo. When the turtle hits the edge of the screen, execution is halted and a dialog box is brought up advising the user that the turtle hit the wall. The user then clicks on the dialog box's "OK" button to continue. When sound is turned on, the user hears a car's brakes screeching as it approaches it the wall and then a loud "crash" as the turtle hits it.

5.3 SLogo Commands

There are six basic commands available; forward, backward, left, right, penup, and pendown, the icons for which are shown in figure 5.3.

The Forward and Backward commands move the turtle forward and backward the number of turtle steps assigned to the command. If in "instant logo" mode, the default value is one turtle step. If not then the user specifies the number of steps when inserting the command into the program. The icons used to represent them are a turtle with arrows depicting the direction of travel. When the command is added to the program, the number of turtle steps it will move is displayed below the icon. The actual distance travelled on the screen is dependent on the length of each turtle step which is defined using the "parameters input" dialog box explained in section 5.5.

The Left and Right commands control the turning of the turtle. The icons used are arrows that show the direction of the turn. The amount of turn is displayed in degrees under the icon in the program area.

The Penup command works as it does in Logo. After issuing this command, turtle movement

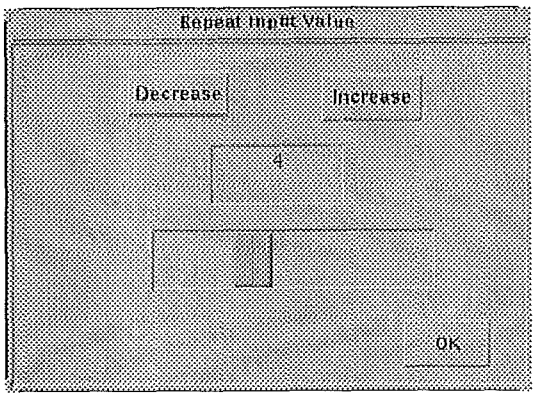


Figure 5.4: sLogo: Repeat definition

no longer leaves a trail on the screen. The Pendown command works slightly different to the command in Logo. In Logo, this command just instructs the turtle to leave a trail as it moves. When inserting the Pendown command in sLogo, the user is asked to choose the pen colour from a choice of black, green, blue and red. As can be seen in figure 5.3, the penup command has the pen slightly above the paper, whilst the pendown command touching the paper. The pendown command appears in the program with the colour of the pen and paper in the colour that the user selected when the command was inserted into the program.

5.4 Repeats and Procedures

The structure of repeat statements and procedures within sLogo has been designed to alleviate the problems mentioned in section 3.1. Within sLogo, the commands are created first and then followed by the repeat and procedure generation.

5.4.1 Repeat Statements

One of the problems with creating repeat statements within traditional Logo environments is the requirement that the number of repeats be known before starting the repeat definition. Within sLogo, the opposite is true. The user inserts into the program the commands to be repeated, and then defines how many times the commands are to be repeated.

The creation of a repeat statement has two steps; selection, and definition.

The user selects the commands required and inserts them into the program . This allows for any error in the selected commands to be corrected immediately. Once the required commands are in place, the user then selects the commands to be included in the repeat statement.

Selection of multiple commands is performed by dragging a box shape around the commands to be selected. The user clicks and holds the mouse button down to the left and top of the first command to be selected. After a short period the cursor changes from its normal pointer shape to a cross. Holding the mouse button down, the user drags the mouse down and to the right of

the other commands to be selected. A thin rectangle appears on the screen with the origin at the original mouse click, and the extent at the current location of the mouse. Once all the commands to be selected are enclosed within the rectangle, the user then releases the button. Any commands, with a few exceptions, that have at least two corners of their icon within the rectangle are selected. This is indicated by the commands appearing with their reverse icon.

Exceptions to this selection procedure are any commands contained within repeat statements already defined. Because their encapsulating repeat command icon is already enclosed, inclusion of any commands encapsulated within that repeat statement would create a nonsense program.

Once the required commands are selected, the user then clicks on the repeat button in the control panel. This brings up the "repeat input" dialog box shown in figure 5.4.

Three ways are available by which the repeat value can be entered; clicking on the increase/decrease buttons, sliding the slider, or by entering the value directly. It has been assumed that the value is most likely to be between one and ten, so the buttons and slider can only present numbers within that range.. In order to enter values greater than ten, the user must use the keyboard to enter the number directly into the centre "value entry" box.

When the user has selected the required repeat value, they need only click on the "OK" button. The selected commands are then indented, with a repeat command replacing all the commands selected. To highlight the commands contained within the repeat statement, a box is drawn around the commands and encapsulating repeat command.

One rarely writes a program correctly the first time, so there is always a need to modify the code written. The commands themselves are modified by deleting and inserting new commands as required. To change the repeat count, the user clicks and highlights the repeat icon and again clicks on the repeat button in the control panel. This brings up the "repeat input" dialog box. Once the correct value has been entered and the "OK" button pressed, the repeat command is updated with the new value.

5.4.2 Procedures

The creation of a procedure is much like that of a repeat statement. The user selects the commands to be encapsulated within the procedure in exactly the same way as with the repeat command.

Once the required commands are selected, the user clicks on the procedure button in the control panel. This bring up the "procedure input" dialog box shown in figure 5.5.

The dialog box provides for two ways of entering the new procedure's name, either by directly typing text on the keyboard, or by clicking on the appropriate buttons.

Clicking on the buttons results in the letter chosen being appended to the procedure name, and the letter chosen being spoken out loud through the computer's speaker.

If an incorrect letter is entered, the user can click on the "rubout" button to erase it. Once they are satisfied, they can click on the "OK" button.

Once the "OK" button is pressed, the commands selected are replaced with the new procedure. It appears as the "house" icon, with the procedure's name appearing slightly below the icon itself.

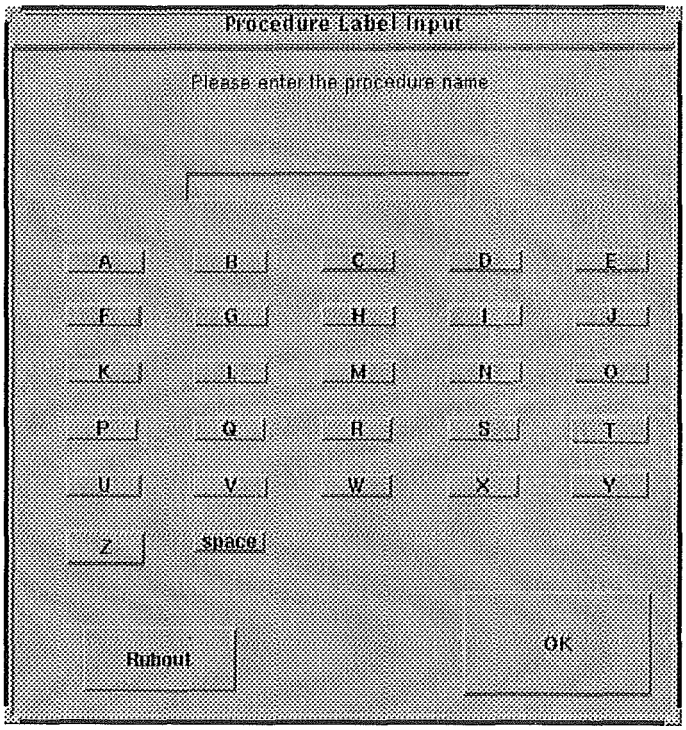


Figure 5.5: sLogo: Procedure Details Input

At the same time a new procedure is added as a command to the list of available command in the "command panel". As within the program panel, the procedure appears with the "house" icon with the procedure's name appearing below it.

To view or modify the contents of a procedure, the user clicks on the "door" in the "house" icon of the required procedure. This can be done by choosing an icon either within the program panel or command panel. The program displayed is replaced with the procedure details (see figure 5.6). The procedure can then be modified as normal. To return to the program, the user clicks inside the door of the "house" icon at the top of the screen. Any changes made are propagated though the program to any other references to the procedure.

Should the user need to modify the procedure name, selecting the procedure icon and clicking on the procedure button brings up the "procedure input" dialog box again. Once again, any changes are propagated to other references to the procedure.

5.5 Environment Modes and Parameters

In this section we examine the different modes of operation available, plus the different parameters available. Changing the parameters and modes is not a function that a user would normally do, so the ability to do that has been hidden. The parameters dialog box (figure 5.7) is invoked by the key sequence *ctrl-p* when the mouse is in the control panel area.

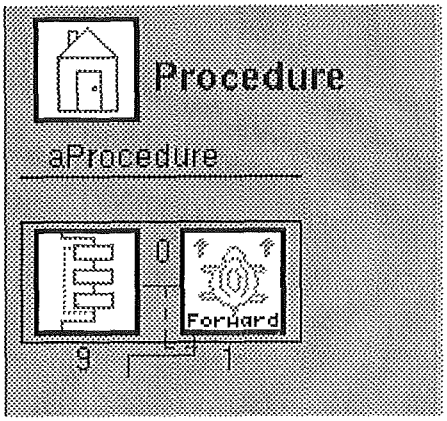


Figure 5.6: sLogo: A Procedure Detailed

The parameters available are "instant logo", "sound", "fast turtle", and "highlight commands during execution". These are individually controlled, and are either on or off. SLogo also has the ability to change the length of a turtle step.

The environment has two major modes of operation; "instant logo" and "normal". In "instant logo" mode, commands that would normally require a value assigned such as forward or left are assigned predetermined values. By default, forward and backward are set at one turtle step, the turn commands have their default value set at 90°. In "normal" model, a dialog box is opened when the user inserts the command into the program, prompting the user to enter the desired value.

Because of the machine dependency of sound output, sound is either on or off, and no volume control abilities are supplied. This can usually be controlled elsewhere on the computer.

Due to the execution speed of sLogo, it was necessary to introduce a means by which the execution speed could be increased for complex drawings as the turtle visually changing makes screen updates slow. By turning "fast turtle" on, the turtle is removed from the screen. Clicking the mouse in the "turtle screen" area also manipulates this parameter.

As with the turtle, highlighting the commands during execution slows the program considerably. By disabling the highlighting of commands and selecting the fast turtle option, the program executes much faster although slower than the speed of standard logo programs.

Different levels of user ability may require that the length of a turtle step be different. An experienced user would want each step to be one pixel, but a young child using the "instant logo" option might require that the step be 40 pixels (about 2cm on the screen) so that the movement can be easily identified. This option allows the length to be changed to any value.

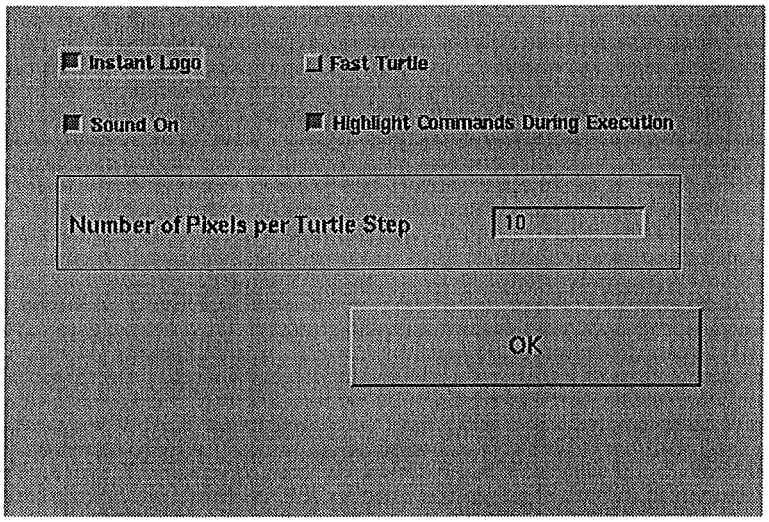


Figure 5.7: sLogo: Parameters dialog box

Chapter 6

Implementation

SLogo is written in ObjectWorks/Smalltalk v4.1, which was chosen because it supports the object oriented paradigm, allows for quick prototyping, and allows for programs to be developed on fast UNIX boxes and then run on a Macintosh computer.

When considering the language and computer type to develop sLogo in, a number of points such as availability, speed, and functionality had to be taken into consideration. Because sLogo was directed at children and might have been used by schools, it was necessary to take into account the computer equipment that schools had available. Being a mouse and icon driven program, the traditional school computers such as the Vic-20 and Apple II would not have been able to support sLogo. Only the modern machines found in schools such as Macintosh, PC, and Archimedes have the system to support the design of sLogo. The Archimedes was ruled out immediately as one was not available to use. Because the department did not have any development software for the two PCs it owned, the PC was like-wise eliminated from contention. Due to programs such as the "Computers in Schools" promotion being run, more schools have been able to acquire Macintosh computers. This then seemed to be the obvious choice.

The other main candidates for the implementation language were boxer (see section 4.3), Interviews (an object-oriented X-windows library), SUIT (an X-Windows library), and Hypercard. Boxer had the advantage of being a system that could be easily modified by the user. As well, it has Logo built in, meaning that a large proportion of the development was already done. It was not chosen as it was not available for licence at the time required and because insufficient information was available at the time to ensure that it was possible to program it as required. Interviews is a set of window creation libraries written in "$C++$". Interviews had only a limited amount of documentation available, the majority of which was in the form of example "$C++$" programs. It also required the use of the X-Windowing system, something that most schools would not have access to. Though a port to the Macintosh was in the pipeline, to date it is still is not available. Despite the ease with which Hypercard applications can be created, it does not have adequate support for sound and colour and so was not suitable for sLogo.

Command
- Forward
- Backward
- Left
- Right
- Repeat
- Procedure
  - StartCommand
- PenDown
- PenUp
- Colour

Figure 6.1: sLogo Command Inheritance Structure

6.1 Commands

An object–oriented approach was appropriate for sLogo because the main differences between commands are only in their action on the turtle and how they are displayed on the screen. The main advantage of using an object-oriented structure for the definition of commands is the ability to encapsulate functions within the object. The clearest example of this in sLogo is the method *action*. This object method is called when the system wants the command to perform its action. When it comes to actioning a command, the program does not need to know what type of Logo command it is. All commands inherit and redefine the the method action, so sLogo can safely call this method knowing that the action taken will be appropriate for the command. This feature of object-oriented programming is used extensively throughout the implementation of sLogo.

6.1.1 Command Details

Figure 6.1 shows the command definition hierarchy which is three levels deep. Because the basic Logo commands are different only in their action on the turtle, these commands inherit all the properties of the standard command defined at the top of the structure, with only their *action* method redefined for the required action. Procedure and repeat commands differ both in their action and how they display themselves on the screen, so both these command types redefine both the *action* and *display* methods.

When a command is instructed to action itself, the basic turtle movement and pen commands send only the required messages to the turtle object for action. The procedure and repeat commands take the message and forward it to all the commands they encapsulate. The repeat statement does it multiple times, based on the number of iterations required. Because these differences are only slight, it was possible to define the commands as super-sets of others.

The StartCommand is a superclass of Procedure as it inherits all the functionality of procedures, but has small differences, mainly in the displaying of the program. The StartCommand is not a user command, but an implementation specific object used to make the commands in the top level of the user's program appear to sLogo as if they were contained in a procedure. This allows for easier manipulation and control of these commands.

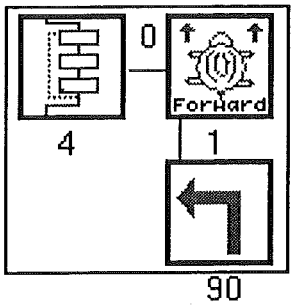


Figure 6.2: Repeat Statement Example

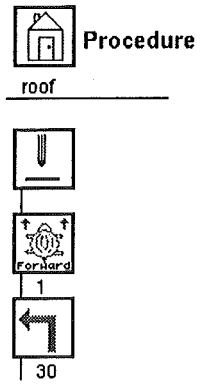


Figure 6.3: Procedure Example

The repeat command displays its encapsulated commands by indenting them one command deep, and by drawing a red line between the commands in the repeat statement, and by putting a box around the repeat icon and the commands associated with it (see figure 6.2). Red is used as the line colour to highlight the association with the red repeat icon.

Procedures are displayed be removing the program from the screen and replacing it with the procedure. As can be seen in figure 6.3, the procedure icon and the word "Procedure" are displayed at the top of the screen. Just below the icon, the procedure name is displayed. The commands within the procedure are then displayed one under another, connected by a blue line. Like the repeat command, the line's colour is the same as the command's icon colour.

Most commands is sLogo have a value assigned to them. For example the movement commands have the number of turtle steps to move by, whilst the turn commands have the number of the degrees to turn. When these commands are displayed on the screen, their values are displayed underneath the command's icon. Because the PenUp command does not have any value assigned to it, and the PenDown colour is indicated by its icon colour, neither of these commands display a value when displayed in the program.

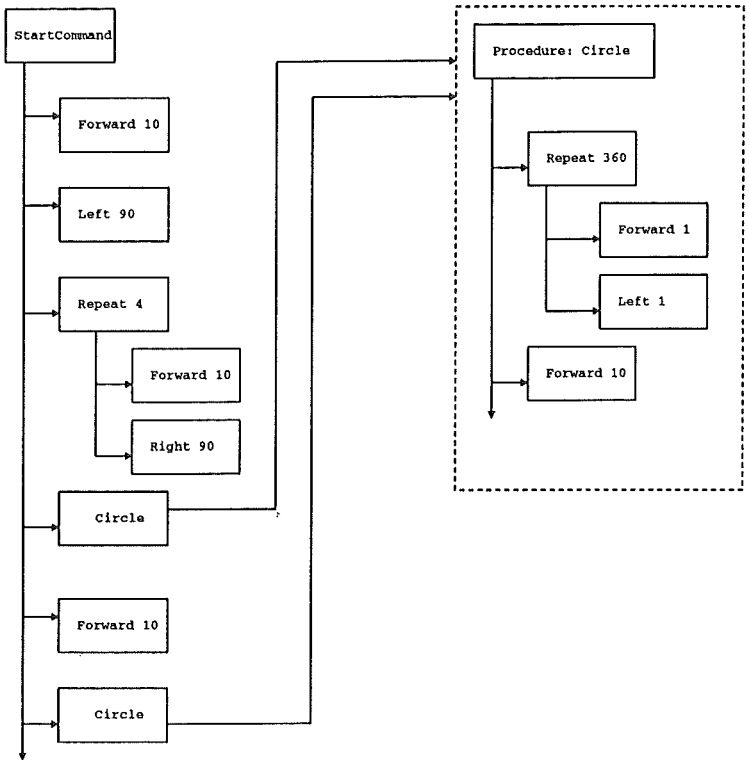


Figure 6.4: A sLogo program, internal representation

6.1.2 PenDown Command

The PenDown commands differs from the other commands in that though it is selected and inserted into the program, the actual command inserted is an instance of the "Colour" class. Issuing an insertion for the PenDown command invokes a dialog box with which the user chooses the new pen colour from a choice of four: black, red, blue, and green. After the user has chosen the required colour, a new instance of "Colour" is created with its icon colour and *value* variable set to the required colour. This is what is inserted into the program. This system was used only to simplify the implementation of different pen colours.

6.2 Internal Program Structure

In sLogo the user generated program is stored, not as a sequential list of commands, but as a pointer to the StartCommand. An example of a sLogo program and how it is stored internally is shown in figure 6.4. Being a specialisation of the *Procedure* class, it can be manipulated as an instance of the procedure class. As a procedure, it has within its *cmdList* the commands in the top level of the program. When the user presses the "Go" button, the message *action* is sent to the StartCommand. Using the method inherited from the Procedure class, the command then passes this message to each command it encapsulates. The basic Logo commands just perform

their defined action upon receipt of the *action* message, while procedures and repeats pass the message on to their encapsulated commands.

The advantage of using this data structure is that sLogo need only send one message to have all the commands respond. The main disadvantage is the time delay incurred in trying to identify a particular command given various pieces of information. For example when trying to identify which command the user has selected with the mouse.

When a command is displayed on the screen, the location of the top-left hand point of the command's icon is stored within the object. This point is used when attempting to identify on which command the user has clicked the mouse. Each command is, in turn, passed the point and asked whether that point in within the area of the screen that its icon occupies. It it is then the command selected has been found.

6.3 Command Selection and Insertion

Within the command, program, and control panel areas of sLogo, the command and button images are displayed as bitmaps rather than being widgets as would be likely if an "X" development toolkit were used, making it more difficult to determine which button or icon is selected. When widgets are used, it is the window managers job to determine which widget has been selected and pass this information to the user program. In Smalltalk, the use of widgets was not an option, and so the use of bitmaps was necessary, with the application having to establish the button or command selected itself.

Within the command and control panel areas, the commands and button are set out in a line. This allows the command or button selected and its position in its encapsulating array to be calculated buy dividing the mouse click position by the area of the screen that a command occupies. The resultant value is the index into the array in which the command is stored. In the program area, commands are not stored within one array, but are contained and pointed to by encapsulating procedure and repeat statements. In this case, the method used to establish which command has been selected involves sending the mouse-click position to each command and asking it if the position is within the area of the screen its icon occupies. If it is then the command is asked to display its colour reversed icon, giving the effect of being highlighted.

Almost all windowing systems today have the concept of "drag-and-drop", where the user selects an item on the screen, and holding down the mouse button, moves the mouse to a new location, "dragging" the selected item with it. This feature can only be implemented on "event-driven" windowing systems. Because of the number of different computers Smalltalk runs on and the different windowing systems used on these, Smalltalk has been implemented as a "polling" system. In an "event-driven" system, nothing happens until the user clicks on a window or selects a menu item. In a "polling" system, each window is repeatedly polled to check for mouse activity. The two main practical differences between the two systems is that in a "poll" based system, you cannot have "drop-and-drag", and the differences between single and double mouse-clicks cannot be calculated easily.

These differences affect sLogo in the area of command and button selection. The problem of the actual selection of a command or button has been explained above. When a user selects a command for insertion, the cursor is changed to the icon of the command selected which the user then moves to the point in the program where they wish to place the command. Once the mouse is over the location required, the user clicks the mouse button to insert it. Had "drag-and-drop" been an option, the user would have just clicked the mouse button over the command, and holding the button down, dragged the command to the required location. When the point was reached, they would just need to release the button for the command to be inserted. Having drag-and-drop would make the deletion of commands more natural as well. Currently the user selects the command and then clicks the "delete" button. With drag-and-drop the user would simply drag the command to the "rubbish bin" and drop it in. The use of a drag-and-drop metaphor to select, move, and delete commands would be better as the actions are more like those done by the user in life.

6.4 The Turtle

The turtle is implemented as a class, with an instance of it being created upon starting the sLogo program. This class includes methods for moving itself on the screen, rotation, pen action, and recording the lines that are drawn by the turtle. It also includes an array to store the collection of the twelve images it uses to display itself in its different orientations.

When a line is drawn by the turtle, a new instance of the *ScreenTurtleLine* class is created and added to the turtle's *drawList* array. This instance contains the details of the line drawn; the start and end points, and the colour of the line. The list of lines drawn is collected so that the screen can be easily redrawn or printed out. Each time the "go" button is pressed, the turtle is reinitialised to its default values and the *drawList* array is emptied.

When the turtle rotates, the user sees the rotation take place. This is accomplished by having multiple images of the turtle, each pointed in a different direction. The turtle's image list is actually a collection of twelve instances of the *TurtleImage* class. Each instance represents 30^o of the compass. Because of the information required to display just one image on the screen, a special class was created. Each instance of this class contains the image itself, a mask for that image (so we can see through the "non-turtle" parts of the image), and the position on the compass that the image represents.

To create the required images of the turtle in its various orientations, an image of the turtle pointing north was drawn first. It was intended that this would be used to create the other images required. Though tools exist for rotating bitmaps, they did not preserve the pattern of the image as well as was required when rotation was not of a 90^o angle. Another image, this time at a 30^o angle, was drawn. By reflecting and rotating this and the original image, it was possible to create the remaining images required while retaining the original pattern exactly.

The advantage of creating the turtle class was the ability to encapsulate all the turtle's functionality with the object. The objects that communicate with the turtle do not need to know how the turtle has been implemented, only the the turtle can deal with the messages sent to it. If a

floor turtle is added to sLogo, then as long as it can correctly handle or ignore the messages sent to it, no change to rest of the program is required.

6.5 Printing

As mentioned in section 5.1, the print button doesn't actually print, but generates three files, two postscript and one of plain text.

The postscript files are generated using a modified version of *postscript.st*, a Smalltalk method available by ftp (see appendix A). This method imitates a GraphicsContext on which the program and turtle areas are painted. Rather than generating an image for the screen, it generates postscript which is then saved to a file. The version available could only deal with back and white bitmaps, so it was necessary to modify it for the full colour images used by sLogo.

The plain text file is a version of the program the user generated, but in the syntax of UCBLogo. This is achieved through calling the *printLogoCommands: tabs:* method of each command. In fact, only the StartCommand is sent this message. This command then passes it on to each of the commands in its *cmdList*. Each procedure and repeat command then submits it to each command encapsulated within it, so with one call of the method, the entire program is printed.

Although Smalltalk supplies the required code to send the postscript files to a printer, this was found only to work on the development UNIX system and that the Macintosh printer did not recognise the Level-II postscript images generated by sLogo. Because the current version of sLogo is only a prototype and printing would not be required for the testing of it, the functionality required to print the output immediately was not implemented.

6.6 Sound

Sound in sLogo is implemented by using Smalltalk "primitives", small functions that a user can generate for machine specific applications. The Smalltalk distribution comes with a primitive for sound on the Macintosh, but this was found to be inadequate because it did not support synchronous sound. When using this primitive, the user experiences the turtle moving on the screen, followed later by the sound. In order to have simultaneous movement and sound, it was necessary to write a "user primitive" to support it. Smalltalk supports the use of user primitives by including the object files required to create a new Smalltalk application.

Each command, button, or turtle movement that initiates a sound makes a call to the method *playSound: aSound ofType: aType.* Within this function the computer type on which sLogo is being run and the sound parameters are checked. Because the program development was done on a UNIX system that didn't support sound, the machine type is checked to ensure that the sound primitive is called only when run on the Macintosh. If the sound parameter is set, the Macintosh emits the requested sound. On the UNIX system, the sound name is displayed on the "transcript" window.

A method for playing sounds called *playSound: sound* is supplied in the Smalltalk distribution. It calls an internally defined primitive for playing sounds through the Macintosh speaker. Using "MacBugs", it was determined that the function called the Macintosh *playSound* function with its sound channel parameter set to "NULL". With this parameter set to "NULL", the operating system's "Sound Manager" creates its own sound channel in the system heap. Though this is an easy way to implement sound in an application, it precludes the use of synchronous output. In order for synchronous sound to be used, the *playSound* function requires that it be passed a sound channel already opened by the Macintosh *SndNewChannel* function. This was achieved by writing a user primitive to open the sound channels required. User primitives are created by writing the required commands in "C" and then compiling this new code in with a object files supplied in the Smalltalk distribution. The result of this is a new version of Smalltalk.

In the user primitive written, the *SndNewChannel* function was called to open a new sound channel. This new channel was created in the application's heap. Because Smalltalk manages its heap in a undocumented manner, modifying the application heap by the channel creation during execution, caused the system to crash. It was necessary to create the new sound channel immediately upon starting the Smalltalk application and use that one channel for all instances of sLogo.

With the sound output now synchronous, it became apparent that the sLogo was requesting sounds to be played quicker than they could be played. Though the system documentation said that in this situation the requests are queued and played through in sequence, the author found that the system tended to try and play them as soon as the request was placed. This meant that multiple sounds were being played concurrently through the same sound channel. The result sounded interesting, but was not what was required. The cure to this problem was to stop any sounds currently being played before submitting the next play request. This was done in the user primitive by calling the "SndDoImmediate" function with the "quietCmd" parameter. As there may be requests still waiting to be actioned, the queue is flushed before the playing sound is killed using the "SndDoImmediate" function with the "flushCmd" parameter.

Chapter 7

Evaluation

In order to test whether the sLogo environment was exciting, interesting and educational for children, 27 pupils from a local primary school were invited to use it. These were made up of 24 Standard 4 and 3 Standard 2 (approximately 10- and 8-years old respectively). It was also intended that 2 Junior 2 (6-years old) pupils would use sLogo, but due to scheduling difficulties, they were unable to take part in the testing. The children came into the department in pairs to take part in the testing. Because of illness, one of the standard two pupils scheduled to attend was unable to make it, and his scheduled partner came alone. The school saw the project as an opportunity to expose the more advanced children to something different from their usual course work. This meant that the academic ability of the pupils selected is not necessarily representative of the ability of all pupils of their age.

7.1 Introduction to sLogo

To introduce sLogo to the children, the author demonstrated the basic movement commands to the children with sLogo in "instant logo" mode. First the children were asked it they had ever heard of Logo, and if they had they were asked to describe how it worked. It was then explained that this was a version of Logo and that the object was to move the turtle around that screen. When the turtle moved, it left behind a line, meaning that simple drawings could be produced. Surprisingly, only two of the children had used Logo, with two more having heard of it enough to recognise the turtle, but no more.

A run through of the six commands available was given. Then the author selected a few commands and placed them in the program area thus moving the turtle. The sequence, *fd, lt, bk, rt*, was used to demonstrate the simple movement of the turtle and how to select and place the commands within the program. One this was done the author selected all the commands and clicked on the "trash can" to delete them. The children were then asked to "have a go". Rather than instructing the children to do anything in particular, they were left to choose for themselves what they would do. Most chose just to make the turtle move around the screen, creating various

patterns.

Because the program was limited to thirteen commands, after the first child of each pair met this limit the commands were deleted and the second child was invited to try it. Once both had "had a play," the children were asked if they wanted to create something specific. Some of the pairs had ideas on what they would like to draw. When this occurred the children were encouraged to draw it. When there was no immediate idea it was suggested that a square be constructed. Because "instant logo" mode was on, the movement was limited to one large turtle step and turns of ninety degrees, which made a square particularly simple to draw. Not only is the square an easy pattern to create, but its simple and repetitive structure provided an opening for introduction of the use of the repeat command.

When the opportunity arose, the repeat command was introduced. First the author pointed out a repetition in the commands that the child had entered, and suggested that it might be easier if the repeat command was used. Using it would mean that fewer commands need be used, and they would just be executed again and again. One of the immediate benefits to the children of using a repeat, was the ability to make available more space for new commands.

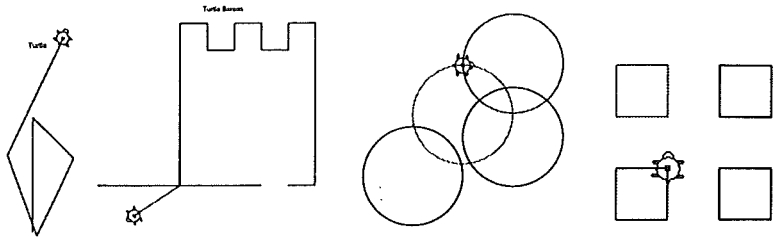
Because this was a test of the sLogo environment and not a course in Logo, there was no explicit instruction in the use of procedures except where it where seen as appropriate. For example, one of the tasks completed by the children was creating a row of squares. After creating a square using a repeat statement, they were advised to create a procedure from it so that it would become a standard command that could be used again later instead of recreating the code for the it.

Though there was little specific use of procedures in their normal role, the ability to create them was more often exploited simply to increase the size of the programs written. As has been explained, sLogo is limited in the number of commands that can be contained on the screen at any one time. When the screen became full, the procedure function was used to turn all the commands into one, allowing room for more commands to be entered. The procedure essentially provided for the "hiding" of the commands.

As well as the ability to create space for further commands, "hiding" the commands allowed for faster execution of the program. Because each command displayed on the screen is executed in one time period, all commands encapsulated within procedures are executed together within one unit of time.

All 27 children had one session of 45 minutes using sLogo. Four pairs were invited to return for a second session. They were selected by listing the children names on a sheet of papers, closing ones eyes, and stabbing the paper with a pen. The first four pairs with holes through their names were selected. One of the initial pairs found sLogo to be too slow and were excluded from returning for a second session.

Four examples of pictures created by the children are shown in figure 7.1. The first two were created during two pairs' first session and give a typical example of level of complexity of pictures drawn in the first session. As previously stated, most of the pairs just *moved* the turtle around the screen. One pair was different. After demonstrating the insertion of command in "instant turtle" mode, they wanted to draw diamonds, requiring that the "instant logo" mode be off. Unfortunately



(a) Session 1 (b) Session 1 (c) Session 2 (d) Session 2

Figure 7.1: Example of the Pictures Drawn by Children

the picture they created in that session was lost. The second picture is by a pair who decided they wanted to draw a castle. Though most of it was drawn using just simple turns and moves, the battlements at the top were created by using the repeat command to repeat the constant moves and turns.

At the second session, both the quality of the work produced and the program structure had improved. All the pupils that took part in a second session gave the impression that they were in control. Even though there was a week between their first and second sessions, they had remembered that both repeats and procedures were available and what their functions were, though a few had forgotten which button to press to get a repeat. The pair that had wanted to create diamonds in the first session wanted to create the Olympic rings in the second. They immediately realised that there were only four colours available, but decided that they would do it anyway. Because a circle in Logo is created by move one step and turning one degree for 360 times, the length of the turtle step was set to be equal to one pixel on the screen. Their efforts can be seen in the third picture in figure 7.1. Had there been sufficient time, the would have completed and correctly aligned their rings. The fourth pictures was created by *mistake.* The child created a procedure called *box*, drew the box, and, moved the turtle to a new position. This new position was not what he expected, but upon realising that the turtle was not facing in the direction expected, used this information to create the picture above. It was constructed using a procedure for each coloured box and a procedure *move* that moves the turtle to a new position and then re-orientates its direction to that before the *move* procedure was called. Within each of the procedures used to create a box, the pen colour was set, the box drawn, and the turtle orientated so that the move command would take it to the next desired position.

During the testing of sLogo it became clear that even at the age of 12, some children still suffer from an egocentric view. This was most obvious when the turtle pointed downward. When the turtle is facing this direction, the turtle's and child's left and right perspective are opposite. Even though the turn command icon's direction are still valid because they are independent of their

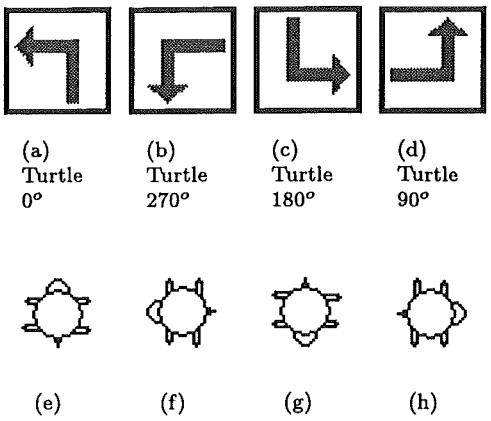


Figure 7.2: Rotating the Left command icon to suit the Turtle's direction

direction, the icon's focus is primarily for the turtle facing straight up. A way to make it easier for the child to select the correct turn command is to rotate the left, right, forward, and backward command icons in relation to the turtle's direction. Figure 7.2 shows how the Left turn icon would be positioned in relation to the turtle's changing direction.

Because the children relied principally on visual input, new commands were entered based on the position and orientation of the turtle. Currently when a command is deleted, there is no change in the position or orientation of the turtle, meaning that the visual information was out of date resulting in the next command selected being incorrect for the desired result. In order to orientate the turtle with relation to the commands in the program, it was necessary that the program be rerun from the beginning. Because this is a manual procedure (the child explicitly clicks on the "go" button") the updating takes places on the screen in real time. This method incurs delays because of the time required to update the screen, causing the children frustration while they waited for the update to complete.

There are two methods that can be used to ensure that the screen is up to date following the deletion of a command. The first is for each command to have a method that reverses the commands action. The second is to update the screen from scratch by re-running all the commands the program. The first option is more appropriate when the command deleted was the last one executed, the second is the best method when the command deleted was in the middle of the program.

In the prototype of sLogo the number of commands available of the screen was limited to thirteen. Though this was found to have some disadvantages during the evaluation of sLogo, it may be an advantage because the user is forced to use procedures. Because most children tend not to use procedures even with explicit instruction about their benefits, they may realise the benefits themselves if forced to use them by restricting the number of commands that may appear on the screen at any one time. During the evaluation of sLogo the limit of thirteen was found to

be too restrictive. A more appropriate number might be fifteen or sixteen, though this should be modifiable to suit different conditions.

When using procedures in standard Logo, all references to the procedure are to the same Logo code within the program. When using sLogo, all references are made to the same code, but visually, it is a different procedure. This could cause confusion when the procedure is modified. If the user selects one instance of the procedures icon and modifies the code within it, they may expect that other instances of the procedure would stay the same. Due to the limited nature of the testing of sLogo, the situation of editing existing procedures was not encountered because each procedure was only used in one place. This raises the issue of whether the modification should affect all instances of the procedure or if it should affect only the instance directly modified. In terms of the conventional use of procedures the former should be the case. If this is the case then a method is needed to create a new procedure from an existing one, whilst still retaining the low level of complexity that currently exists in the construction of procedures.

7.2 Input Dialogs

A feature of sLogo is the use of dialog windows to enter values for movement and turn values, repeat command values, and procedure names. In standard Logo programs, these values are entered by typing the values alongside the command. As commands in sLogo are inserted and created with icons, dialog windows are brought up on the screen whenever there is a need for a value to be entered.

7.2.1 Movement and Turn Values

When not in *instant logo* mode, the values for movement and turns are entered manually through a dialog box. This provided an opening for the children to experiment. Often the value was decided on the basis of "what will happen it we use this number?" The children were encouraged to try it to find out.

When selecting a value to move the turtle backward or forward, the children often took a guess at the correct value. Because the length of a turtle was set at 40 pixels, they were relatively easy to see on the screen so the value chosen was often correct. Only when a long line that required six or seven turtle steps to make was required, was there errors their value estimation. Some of the children equated the width of two or three of their fingers to a known turtle step length on the screen to estimate the required value. They used their initiative and usually provided the correct value.

Entering the turns values proved to be simple matter—the children had no trouble in working out the number of degrees to turn. This can be attributed to their level of education. Confusion was caused by the turtle image not moving for turns less than 30°. As explained in section 6.4, there are only 12 different turtle images, each representing a 30° area of the compass. When this occurred the children were reassured that the turtle did turn even if they couldn't see it. They

were then encouraged to draw a line which showed up the turn value selected. Some confusion was caused when a turn value of 90°, for example, was incorrectly entered as 92°. If the next lines drawn were short, the divergence from 90^o was not immediately obvious, but caused confusion when the difference showed up later.

In all, no problems with selecting and entering values was noted. The main problem was sLogo's inability to modify the value of the movement and turns command without deleting the command and inserting a new one. If the command to be modified had just been inserted, the children often inserted another command to increase the movement or inserted an opposite command to cancel the effect of the incorrect one.

7.2.2 Use of Repeat and Procedure Dialog Input

The use of a dialog to select the repeat command value enabled the children to experiment with different repeat values. When they created a square, a number of children modified the default repeat value of four to something else. Generally the value was increased to a value not a multiple of four. This was done for fun and caused some interest when the turtle kept going, often repeating what it had already done. Often the final position and direction of the turtle was different to the starting position, meaning that the next command to be executed was not made in the direction intended. Although most children reset the repeat value for squares back to four, some left it and used the effect.

There are three ways to enter values in the repeat dialog window: using the increase/decrease buttons, entering the value directly from the keyboard, and using a slider that slides left and right increasing and decreasing the value. Though the children used the scroll bars on the main window with little difficulty, few had associated the scroll bar action with that of the horizontal slider, and had to be told how to use the slider. Once that was done , they had no problems using it. Only a few chose to use the buttons. This may have been because of the buttons' position at the top of the window, or may have been a result of the lack of need for their use.

When a procedure is created, a procedure input window is opened to allow the user to give the procedure a name. The name can be entered using either keyboard directly, or clicking on the buttons with the mouse. When a button is pressed, the letter selected is spoken through the computer speaker. Some of the children had a problem finding the letter required in the window, but the second of the pair usually helped. It was hoped that the use of buttons to create procedure names might result in "sensible" procedure names being used rather than the children using their names or random words. This was not the case. The children most often used their names for procedure names. In order to make the name more meaningful for future reference, the author suggested more appropriate names such as square, tower, or circle.

7.3 The Turtle is a Turtle

Rather than using the traditional triangle to represent the turtle, sLogo uses an image of a turtle with a clearly defined head and tail. When the turtle was pointing towards the top of the screen,

the children had no difficulty in determining its direction as opposed to the problems found with the triangle representation [14].

Though the turtle image made clear the direction of the turtle, there was still evidence of the egocentricity of the children. Whilst some would orientate their bodies to that of the turtle, others would place a turn command and then wonder why it headed the wrong direction. Problems such as this were usually fixed by the children themselves through "discussions" between the two children testing the system.

Problems were experienced when, due to a bug in the program, the turtle on the screen was sometimes left facing the wrong direction after the execution of a procedure. This resulted in a forward command making the turtle go *backwards*. A turn command executed after this error occurred would correct the image. When this happened the author would apologise profusely, and advise the children the direction the turtle would move in if a forward or backward command were issued. Although most accepted this, this problem showed how important visual information is. Even after explanation of the problem, some children were still surprised when the turtle insisted on going in what seemed to be the wrong direction.

7.4 Response to Sound

When commands are selected from either the command or program windows, or a button in the control area is pressed, a sound is emitted. Not only does an audio feedback provide interest for the children, it also provides a positive response to the mouse selection. The use of sound in sLogo can be divided into three areas; turtle movement, buttons, and command selection. The sounds in the first two categories reflect their actions whilst the sounds initiated in command selection are a mix of sounds "borrowed" from various games that were available to the author.

The sounds emitted when the user clicked a button in control panel received a very positive response. Rather than being sounds extracted from games, the sounds emitted in the control panel are words spoken to indicate the button's function. The "go" and "stop" buttons evoked the most enthusiastic response. The "go" button's sound is the author saying "go turtle go" with great enthusiasm. The "stop" button's sound is the author saying "stop" with a tone of panic. All the children enjoyed these two sounds, resulting in the "stop" button being repeatedly clicked on for entertainment value. As a result of the "go" button's sound output, all the children referred to it as the "go turtle go" button.

The sounds emitted when the turtle moved were also well received. When the turtle moves forward, the sound of a truck accelerating away is made. When turning, it is the sound of a car's screeching tyres. Because the turtle screen is bounded, that is, the turtle cannot move off the edge of the screen, sLogo has the turtle "crash" into the edge rather than go off it. As the turtle approaches the wall, the "tyre" screech sound s emitted, with a loud "crash" sound being used when the turtle actually hits the wall. All the children enjoyed this feature, with some deliberately maneuvering the turtle into the wall just to have it crash.

Because the sounds heard upon command selection were arbitrarily selected they bore no resemblance to the commands themselves and as such did not support the actions of the commands. When the children selected a command for the first time there was interest in the sound, but after a few times, the interest was lost as the sound emitted was predictable. Rather than each command having its own sound, it may be better to use random sounds. By having a collection of different sounds, command selection could randomly select one of these sounds, making the act of selecting a command more interesting through not knowing what sound would be emitted when the selection is made.

7.5 Problems with Mouse Usage

A number of problems were noted in the way the mouse was used by the children. Some of these problems are attributable to the implementation of sLogo, whilst others were due to the ergonomic setup of the computer. No problems were noted that would lead the author to believe that the children had real problems that could be attributed to the use of a mouse. This is in line with the study by Crook[8] in which he studied young children and their ability to use a mouse to control a graphical user interface on a computer. The only real problem found in the Crook study was that young children (5 years), have trouble physically relocating the mouse, whilst keeping the cursor static.

When a command is to be inserted in the program area, the cursor image becomes the image of the command selected. It is inserted by aligning the new command under the previous command. In the implementation of sLogo used for testing, the location required to insert the new command was too precise. Although most of the children did not have any problems once they realised how much precision was required, a few still had problems with insertion even after two sessions with sLogo. There was no apparent correlation between this difficulty and the age of the child.

The first standard 2 child tested had problems with the speed of the mouse—in relation of the movement of the mouse on the table, the cursor moved too fast on the screen. It was thought that this might be caused by the lower level of motor control of the 8 year old against that of the 10-year-olds the system had already been tested with. This did not prove to be the case as the 8-year-olds that were tested later did not have this difficulty. A problem that affected one of the other two 8-year-olds tested was moving the mouse itself. Due to the height of the table and that of the child, the mouse was pressed down onto the table rather than guided around as is necessary. This action caused the mouse ball to be pressed into the mouse itself, resulting in no movement of the cursor on the screen.

In order to create repeat statements and procedures the user must select all the commands to be repeated or put into a procedure. As explained in section 5.4, selection of commands is accomplished by holding the mouse button down while the mouse is to the left of the top command to be selected, and then dragging the mouse (with the button still down) to the right of the last command to be selected. In doing so a rectangle is drawn around the selected commands. Once the button is released, the icon colours are reversed indicating selection. Most children had difficulty

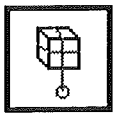


Figure 7.3: New icon proposed for procedure button

with this the first time they tried it. Though the examples of sLogo in this document have a grey background, the testing was performed on a Macintosh on which sLogo has a white background. Because the program and command areas and scroll bars were all white on the testing system, the children suffered some confusion over the instruction to place the cursor just to the left of the command's icon on the "white bit". The child either positioned the cursor on the icon itself or on the scroll bar. Using a pen to point to the required location proved a problem as it is impossible to position a pen correctly when viewing the screen from angle due to parallax.

Other problems were noticed when selecting multiple commands due to the difficulty some children had with moving the mouse whilst still holding mouse button down. The first problem noticed was the physical requirement of doing so. Some children had trouble in holding the mouse button down and moving the mouse at the same time. The second was that there was insufficient clear space on the table to move the mouse the whole distance required to selected all the commands without lifting and repositioning the mouse. This repositioning had to be made with the mouse button still depressed. The author believes both these problems were cause more by the position of the computer than any problem inherent with children using a mouse. Both the table on which the computer was placed and the chairs used by the children were "adult" sized. Thus the table was too high and the chairs too low. This did not affect the standard 4 children too much, but did affect the standard 2 children to a significant amount. Some of the children solved the height problem themselves by standing up.

The problems outlined above indicate the precautions that must be taken when designing a system for children. The insertion problem is easily fixed by making the target area of the command to be inserted (the area of the screen where the button can be clicked on and still have the correct insertion point recognised) flexible enough for the child to place it without difficulty. In this implementation of sLogo, once the mouse was clicked, the command selected was *unselected* so that if the insertion failed, the child had to reselect it. This should be modified so that the command is only unselected when the insertion was successful. Because children's motor control is still developing, such things as the precision of the mouse movement must be easily modifiable to suit each child. In this case the Macintosh has an easily modifiable control built in to the operating system. The problems with moving the mouse large distances to select the commands and holding the mouse button down whilst doing so is easily cured by proving a table and chair arrangement more suited to the height of the children.

7.6 Recognition of Icons

In designing a system such as sLogo, attention must be given to the design of the icons used in the system. Not only must the icon represent its function, they must be simple in their design.

None of the children had any problem in understanding the six icons used to represent the basic turtle commands as their function was clear from the simple graphics used. The repeat statement icon was created with a loop to represent the iterative nature of repeats. There was little feedback to the design of this icon though one child did ask what the "railway tracks" button was for. The main benefit of the repeat and procedure icons was that they where the same as the icons that appeared in the program. After having used the delete button a number of times, one child asked "why's that a cookie jar?"

As explained in chapter 5, the *house* icon was used to give the impression of hiding, with only a knock on the door required to see its contents. Though it causes no confusion with the children, the icon is too abstract in its relation to procedures. Also the icon may be confused with the Logo *home* command, which when executed, moves the turtle to its home position on the screen. An alternative proposal is to use an icon displaying a box with a piece of string hanging from it. A representation of this is shown in figure 7.3. It uses the box to represent encapsulation, with the loop at the end of the string providing a means by which to open it and examine its contents. Because of the difficulty in explaining the use of procedures and its representation as a box, it may be better if the use of procedures was referred to as *boxing*. In *boxing*, the commands selected are put into a box, which can then be used again when required. If one wishes to examine or modify its contents, then a simple tug or click in the loop at the end of the string will open it. This term also ties in with some of the children calling the movement by which a box is dragged around the commands to be selected as being *boxing*.

7.7 Summary

The features available in sLogo are limited. The lack of variables and recursion means that the fractal patterns usually associated with turtle graphics are absent. Even with these limitations the children were still able to create simple pictures. As most of the children had not experienced Logo previously, almost all enjoyed using it and wanted to use it again. Of the 27 children who used the system, only two did not want to use it again. Both of these were boys who play spacey type games extensively. Though sLogo has sound and colour, it lacks the action of most video games, so this response was not unexpected.

SLogo supports only the sequencing and iteration, two of the three concepts of programming. As such it is not a fully implemented programming language and so is limited in its use. Where sLogo excels is in its ability to excite children. Using sound and graphics it provokes an interest, an interest that can be developed and transferred to a full version of Logo.

There is no doubt that the children enjoyed using sLogo. For almost all of them it was their first introduction to not only Logo, but any program of this type. Although most use a computer at home, this is mainly for playing games and writing with a word processor. Very few programs today required the user to do any other than press a few button. SLogo requires that the user

think about what they are trying to do and then figure out how to do it.

Chapter 8

Conclusion

In chapter 3, a number of papers were examined. Two dealt with problems that young children had with the usual Logo environment. Another documented an environment in which a mouse was used to control an environment designed for teaching structured programming without the need for knowledge of the language's syntax. The last two papers each introduced a "Microworld," one to teach the Logo language, the second to help in the teaching of location identification on a number plane.

These papers suggested solutions to various problems that children are perceived as having when using Logo. The first two papers are particularly relevant to this project because they detail specific problems that young children have with the usual Logo environments, and suggest possible solution for the problems. The use of a "wrap-around" screen confuses some children who see the point where the turtle emerges as being arbitrarily . Also, the small length of the "standard" turtle step has been identified as being difficult to imagine by young children. When turtles turn, the movement is usually instantaneous. When a turn is combined with a movement of the turtle, it is often impossible to differentiate one from the other. By slowing down the action made by the turtle, it becomes possible to separate the turn and movement actions. Because young children lack formal mathematical knowledge, the angles that they use to turn the turtle are often selected arbitrarily. In order for children to manipulate the turtle without the mathematical knowledge this manipulation requires, the use of fixed turns has been suggested. Once these children have the mathematical knowledge required, they can then use an environment that allows them to select the angle of the turn. Also, the triangle-shaped turtle often used in Logo environments makes it difficult to establish the turtle's direction, so the use of turtle with a clearly defined head has been recommended.

When creating repeat statements and procedures in Logo, it is necessary to know in advance what commands are to be included and in the case of repeats, how many iterations are to be made. The creation of repeat statements and procedures can be made easier by providing a means by which commands already entered into the program are transferred in the new procedure or repeat statement. Because young children are egocentric in their view of life, the turn commands selected

when the turtle is facing downward often reflect the child's left and right directions rather than the turtles. Although some will "play" turtle to establish the correct direction to turn the turtle, most will enter different turn commands until the correct direction is found. When the movement of the turtle is not what the child planned, and the reason for the incorrect movement is not immediately obvious, the child will often delete all the work completed and start again. Heller[16] created a version of Logo to help cure this problem by providing a means by which new code could be tested and then possibly discarded without affecting any work already done.

By creating an environment that addresses these problems, not only do young children have an environment suited to their abilities, but they also would have one in which they can accomplish something without their lack of knowledge and skill interfering with their enjoyment.

The best system, for use by children supports sound, colour, and animation to make them look and sound interesting. They also allow the children to use them without restriction, giving them the feeling that they are in control of it. KidPix uses sound, colour, animation, and ease of use to provide an environment that is both easy and fun to use, and that allows the user to create endless images. Though Karel wasn't designed for children, its language has a small vocabulary with which to perform simple tasks. This is a feature that may be beneficial to young children as it can minimise confusion. LogoMation, a Logo derivative, provides a friendly interface in which to program, and a language that supports sound, colour, and animation.

SLogo was designed to provide not only a means by which young children could use turtle graphics, but also to provide an environment in which they would enjoy doing so. Rather than just extending or modifying existing Logo environments, as is commonly done, sLogo was designed to be totally different from the usual Logo environment with the use of a mouse-driven interface.

It was observed that children using sLogo had problems with separating the concrete view of their world from the abstract view of the turtle's on the screen. For example, the turtle used in sLogo is a bitmap of a turtle with a clearly defined head and tail, resulting in no confusion as to the direction the turtle is pointed. When the turtle faced downward, the children showed obvious signs of egocentricity. Most were unable to identify the correct command to turn the turtle in the direction desired. Even though the turn command's icons were bent arrows and valid even when the turtle was facing downward, they were not clear enough in their directional information to help the children. It was suggested in chapter 7 that rotating the commands to suit the turtle's direction may help in correcting this flaw.

Rather than the user having to type in commands, they select their commands from a list on the left of the screen. Using icons to represent the command means that not only does user not have to remember what commands are available, but the function of each command is clear though the command's icon. The "Instant Logo" mode in sLogo allows young children without any formal knowledge of math to guide the turtle around the screen. Because the input values of the movement and turn commands are fixed, it is not necessary for them to try and guess the correct value to use. This mode can be turned off, allowing someone with more knowledge of math to use input values of their choice. The structure of the program generated in sLogo allows the user to easily create procedures and repeat statements from commands already entered into the

program. This allows the user to experiment with commands to be included in the procedure or repeat statement before the procedure or repeat are created.

The use of a mouse to control the environment and write Logo programs was successful. Although problems were encountered with its use, these were caused either by bugs in the application itself or the physical arrangement of the computer. Where the children did have problems using the mouse, for example with the insertion or selection of commands, the problems were caused either by limitations in the sLogo implementation or by the table and chair used being unsuitable for use by the children, rather than being caused by the children's inability to control a mouse.

SLogo is an engaging environment in which to learn programming. Not only did the children enjoy using it, they learnt a little about programming as well. There is be no doubt that the children found using sLogo fun. Their positive reactions to the sounds emitted by the turtle and buttons indicated that the use of sound was a major factor in the children enjoying their time with sLogo and wanting to spend more time on it. Despite the problems encountered during the insertion of commands, the use of the mouse to drive sLogo was successful and it was used successfully for the construction of programs and the selection and deletion of commands. By allowing commands to be entered into the program and then transferred to either repeat statements or procedure, both repeats and procedures were easily created. This feature allowed the children to create the desired effect before formalising their ideas in procedures and repeat statements. The use of the arrows for the turn commands did not have the effect that was hoped for in overcoming the egocentric views of the children, although the children had no difficulty in identifying which of the movement commands was appropriate for their needs.

It was hoped that the use of a dialog box to make procedure name entry easy would encourage the children to use descriptive names for procedures, but this did not happen. Providing the means by which children can easily enter procedure names does not necessarily promote the use of "sensible" names for procedure. When a procedure was created and the children entered a name for it, the name tended to be either the child's or their partner's name. Even though each time a procedure was created they were encouraged to use more descriptive names, such as using *square* when the procedure drew a square, they continued to use personal names. It may be that this problem is to do with their age and maturity, and that at their age they do not have the association skills required to invent or recall appropriate names.

Despite the problems experienced in using sLogo, the concept of using a direct manipulation interface to an audio-responsive programming environment was successful. Not only did the children learning a little about programming, they enjoyed doing so and wanted to come back for more.

Acknowledgements

I am indebted to many people for their support and ideas in the design of this project. In particular, I wish to thank Tim Bell, who not only read and re-read innumerable drafts of this report, but also provided the enthusiasm with which to carry on through the project's darkest hours. I would also like to thank Andy Cockburn for his ideas in creating a usable environment, Bruce McKenzie and Paddy Krishnan for their assistance in taming LaTeX, and Nigel, David, and Hugh for putting up with me for so long.

Bibliography

[1] R Baecker. A programmer's interface; a visually enhanced and animated programming environment. In *Hawaii International Conference on System Sciences, 23rd, Kailua Kona Hawaii*, 1990.

[2] Jay Fenton; Kent Beck. Playground: An object oriented simulation system with agent rules for children of all ages. In *OOPSLA '89 Proceedings*, 1989.

[3] Tim Bell. Making computer science into child's play. 1993.

[4] Gwenda Bensemann. Capturing the interest of young children in computer science. Honours report, Dept of Computer Science, University of Canterbury, 1993.

[5] Judith A Kull; Joyce Carter. Wrapping in the first grade classroom. *The Computing Teacher*, 17(4):12–13,52, Dec/Jan 89/90.

[6] Judith A Kull; Bert Cohen. Pre-logo games. *The Computing Teacher*, 17(1):39–43, Aug/Sept 89.

[7] Peter Cope and Malcolm Simmons. Children's exploration of rotation and angle in limited logo microworlds. *Computers and Education*, 16(2):133–141, 1991.

[8] Charles Crook. Young children's skill in using a mouse to control a graphical computer interface. *Computers and Education*, 19(3):199–207, 1992.

[9] Yuval Shavit's Dad. *LogoMation User's Manual*, 93.

[10] Yuval Shavit's Dad. *LogoMation's Reference Manual*, 93.

[11] Andrea A diSessa. A principled design for an integrated computational environment. In *Human Computer Interaction*, volume 1, pages 1–47. Lawrence Erlabaum Associates, Inc, 1985.

[12] Enrico Fischetti and Antonio Gisoffi. Logoworld: A learning environment for the logo language. *Computer Education*, 69:16–19, November 1992.

[13] Max K Frazier. Logo and angle estimation skills. *Journal of Computers in Mathematics and Science Technology*, pages 22–28, 1889.

[14] Rina Cohen; Esther Geva. Designing logo-like environments for young children: the interaction between theory and practice. *Journal of Educational Computing Research*, 5(3):349–377, 89.

[15] David Gries. *The Science of Programming.* Springer-Verlag New York Inc, 1985.

[16] Rachelle S Heller. Towards a student workstation: Extensions to the logo environment. *Journal of Educational Computing Research*, 7(1):77–88, 1991.

[17] Garret Lange J. Allen Watson. Logo mastery and spatial problem-solving by young children. *Journal of Educational Computing Research*, 8(4):521–540, 92.

[18] C D Maddux; D L Johnson. *Logo: Methods and Curriculum for Teachers.* Haworth Press Inc, 1988.

[19] J E Tuovin; D M Hill; R W Kay. Logo: A vehicle for development and application of knowledge and thinking strategies. *Computer Education*, 67:19–23, February 1991.

[20] Don Ploger; Ed Lay. The structure of programs and molecules. *Journal of Educational Computing Research*, 8(3):347–364, 1992.

[21] Bernadette Martin and J. Dixon Hearne. Transfer of learning and computer programming. *Educational Technology*, 30(1):41–44, January 1990.

[22] L Borghi; A De Ambrosis; C I Massara. Logo programming and experiments to study motion in primary school. *Computers and Education*, 17(3):203–211, 1991.

[23] Bruce McMillan. Teaching with logo microworlds. *Computers in NZ Schools*, 1(1):49–54, 1989.

[24] Lai Kwok-Wing; Bruce McMillan, editor. *Learning With Computers; Issues and Applications in New Zealand Schools.* The Dunmore Press, 1992.

[25] D H Clements; J S Meredith. Turtle math. *Logo Update*, 2(3):9–10, 1994.

[26] C Graci; R Odendahl; J Narayan. Children, chunking, and computing. *Journal of Computing in Childhood Education*, 3(4):247–258, 1992.

[27] Margaret L Neiss. Logo learning tools and motion geometry. *Journal of Computers in Mathematics and Science Teaching*, Fall:17–24, 88.

[28] Seymour Papert. *Mindstorms; Children, Computers, and Powerful Ideas.* Harvester Press Ltd, Great Britain, 1980.

[29] Seymour Papert. Misconceptions about logo. *Byte*, 10(11):229–232, November 1984.

[30] Seymour Papert. Different vision of logo. *Computers in the Schools*, 2(2/3):3–8, 1985.

[31] Richard E Pattis. *Karel the Robot: A Gentle Introduction to the Art of Programming.* John Wiley and Sons, Inc, 1981.

[32] M B Rosen. Using single-stroke logo programs with young children. *Logo in the Schools*, 2(2/3):255–261, 1985.

[33] Elliot Soloway. How the nintendo generation learns. *Communications of the ACM*, 34(9):23–26, September 1991.

[34] Elliot Soloway. Quick, where do the computers go. *Communications of the ACM*, 34(2):29–33, February 1991.

[35] A. D. Thompson; H. C. Wang. Effects of a logo microworld on student ability to transfer a concept. *Journal of Educational Computing Research*, 4(3):335–347, 1988.

[36] Daniel H Watt. The computer as microworld and microworld maker. *SIGCUE Outlook*, Fall:81–89, 1988.

Appendix A

FTP Information

Title	File Name	FTP Address
Karel the Robot	Karel.tar.Z	ftp.germany.eu.net
UCBLogo	ucblogo.tar.Z	gatekeeper.dec.com
LogoMation	logomation.cpt.hqx	plaza.aarnet.edu.au
Postscript GraphicsContext	postscript.st	mushroom.cs.man.ac.uk
Roo & Robby	roo10.zip	wuarchive.wustl.edu

Appendix B

Example sLogo Output

```
ClearScreen
repeat 4 [ Setpencolor blue square  roof PenUp ...
Left 30 Forward 1 Left 90 Forward 3 Left 90 ]

to square
repeat 4 [ Forward 1 Left 90  ]
end

to roof
PenUp
Forward 1
Left 30
Setpencolor  red
Forward 1
Left 120
Forward 1
end
```

Figure B.1: sLogo UCBLogo output

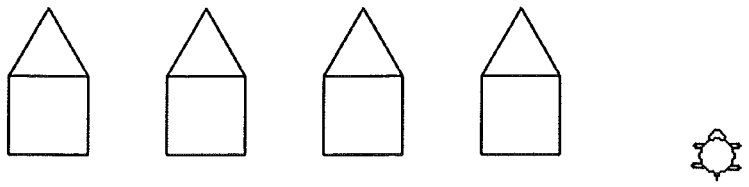


Figure B.2: sLogo turtle screen output

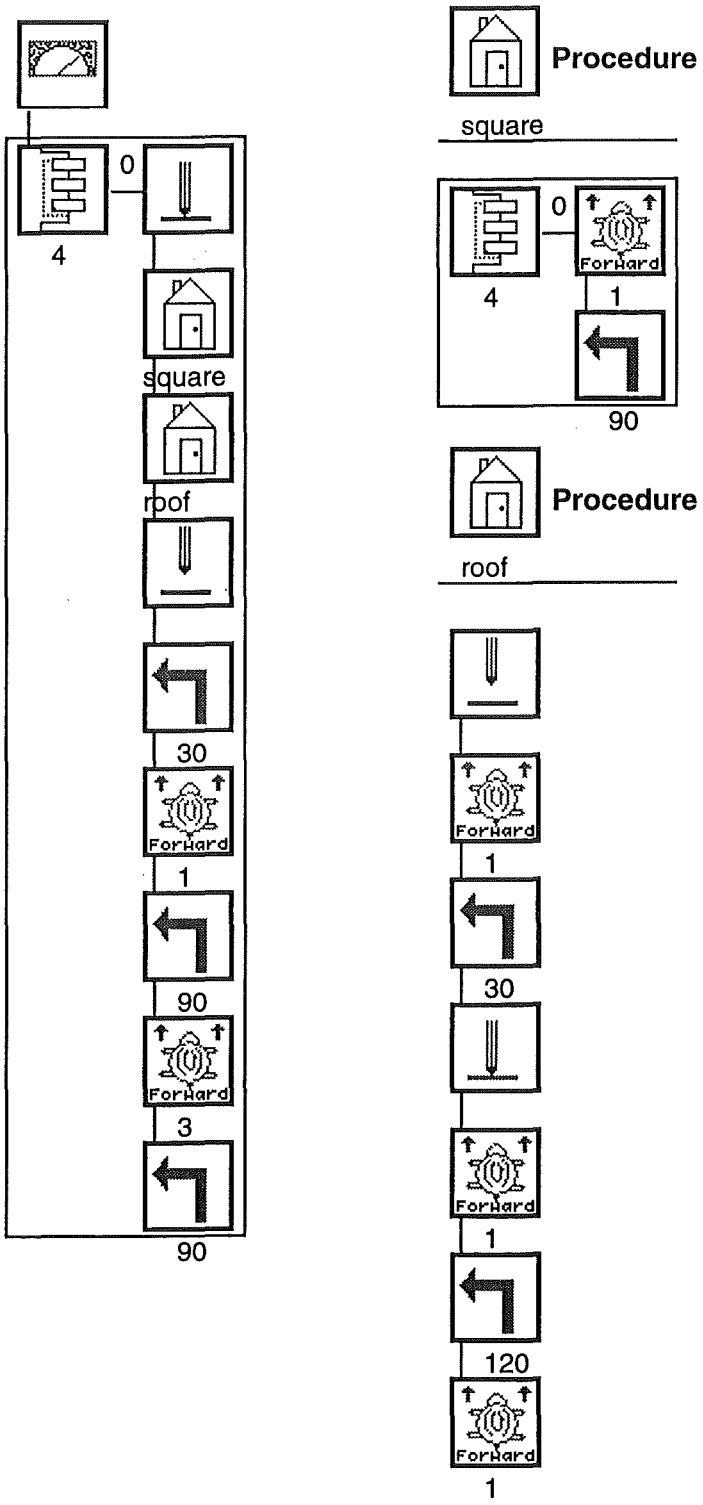


Figure B.3: sLogo program output