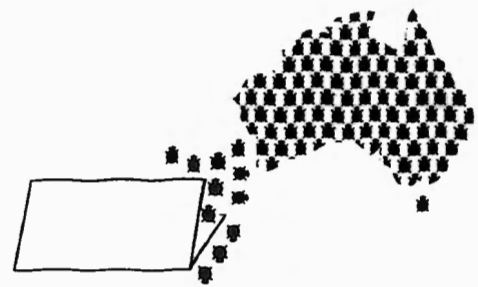


OzLogo

LogoFile



Magazine of OzLogo, a Logo Special Interest Group of the CEGV and MAV.

Vol. 2, Nos. 1 & 2

Inside ...

The Phantom Tollbooth - Integrated Unit for Year 6

SCANTOOL - Relocating Scanned Graphics

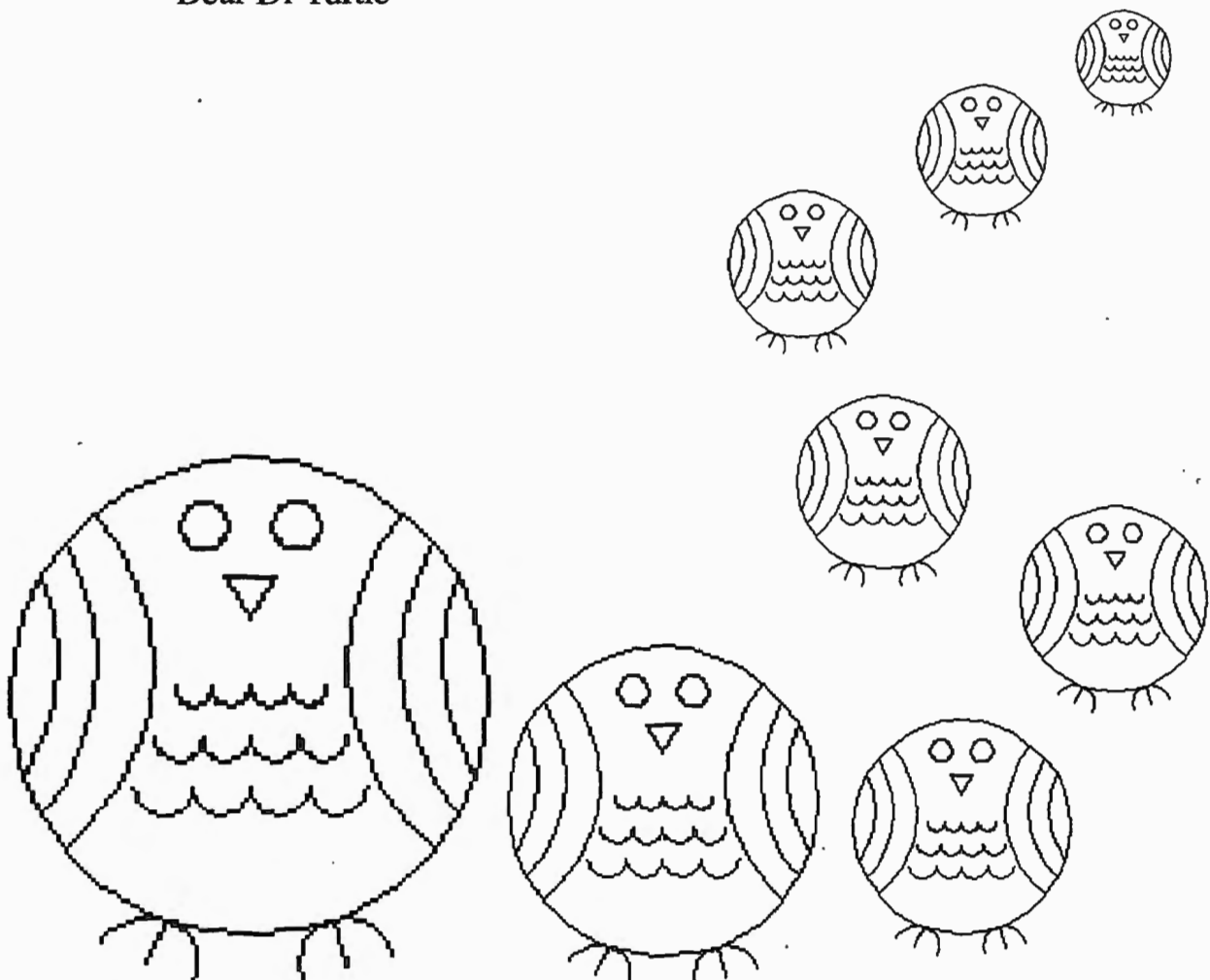
MicroWorlds - A Review

Teaching LogoWriter Programming

How to Run a Sports Day

Animals and Elements - Binary Decision Programs for Science

Dear Dr Turtle



The Creative Tool Kit Every Student Needs

MicroWorlds



Designed by the people who brought you LogoWriter

Cross-Curricula: MicroWorlds Project Builder

- A learning environment that encourages students to develop their own problem-solving strategies while creating school projects.
- A Project book that takes you through the entire process of creating cross-curricula projects, step by step.
- Drawing tool (and 140 colours) for creating backgrounds for stories.
- Infinite number of turtles that can be set to different sizes and shapes, and colours.
- Word-processing features such as multiple text boxes, different fonts, sizes, styles, and colours, plus the ability to place text over graphics, or in any direction.
- Easy to add buttons, sliders and hot spots that help start and stop animation, music and other special effects, as well as create hypermedia links.

MicroWriter's Language Arts

- A Projects book full of writing projects that illustrate how language and art can be used together to communicate ideas and emotions. Projects include Haiku Visual Poetry, Advertising, Conquain, and more.

Both packages include

- Tools to write text in any shape, style, colour or size, or in any direction.
- Drawing and other visual effects' tools such as animation, scrolling or flashing text.
- A music and sound centre that makes it easy to set words to music.
- On-disk project starters, ready-made background scenes, and project samples.
- A new and comprehensive on-line Help System. Teachers or students can even create their own project-specific information balloons.

Package Contents:

- Program disks includes on-disk sample projects
- MicroWorlds Project Book
- Teacher's Resource
- How To Book

School's Pricing

Set \$130

6 User Lab Pack \$585

Unlimited Site Licence \$1595



System Requirements:

Macintosh colour computers (LC's colour Classics or better) 4Mb, System 7 or higher Hard Drive

Available from:



Computelec
AUSTRALIA

38 Hartnett Drive Seaford, Vic 3198
Ph: (03) 786 7177 Fax: (03) 785 3599

THREE EASY WAYS TO ORDER!

By Phone

Call Toll Free 008 337 055 or
(03) 786 7177 In the
Melbourne Metro area

By Fax

Fax your order to us on
(03) 785 3599

ORDERS

By Mail

Simply complete your order details and mail with your Purchase Order, credit card details or cheque to:

Reply Paid AAA 146,
P O Box 2053 Camm Downes
Vic 3201



Editorial

This issue contains a report from the very successful 1993 International Logo Conference held in Melbourne.

The first Australian Logo conference was "Logo in Australia - Ten Years On", hosted by the Computer Education Group of Victoria and held in 1985 - ten years after Logo was first used in this country (by Sandra Wills in Tasmania). International speakers at that conference were Hal Abelson from M. I.T. and David Squires from Chelsea College in the U.K. A browse through the Proceedings reveals Logo work already well developed in primary and secondary schools, teacher education institutions, and at least one kindergarten, as well as a busy and productive research community.

Another Logo-specific conference, the International Logo and Mathematics Education Conference, was held at lake Tinaroo in Queensland in 1991.

Papers on Logo work are included of course in many other conferences. However there is great value in the focused Logo conferences as well. The common background knowledge of participants enables issues and ideas to be examined at much greater depth than otherwise, and the enthusiasm of experienced users supports novice attendees in a very productive way. We saw all of this writ large at this year's conference. We hope that further Logo conferences can be arranged to maintain this momentum.

Anne McDougall
Leon Guss

Editors

LogoFile is a magazine for interested users of Logo in education published by OzLogo.

OzLogo is a Special Interest Group of the Computing in Education Group of Victoria (CEGV) and the Mathematical Association of Victoria (MAV).

Editors:
Anne McDougall
Leon Guss

Assistant Editor:
Kirsty McDougall

Contributions to be forwarded to:

LogoFile Editor
OzLogo
Room 42
Statewide Resources Centre
217 Church Street
Richmond VIC 3121

Membership fees are:
\$20 per individual
\$30 per organisation.

Attach cheque made payable to OzLogo. Applications should be forwarded to:

Membership Secretary
OzLogo
Room 42
Statewide Resources Centre
217 Church Street
Richmond VIC 3121

Contents

Editorial.....	3
Integrated Unit of Work for Year 6	4
Four New Logos Released	7
Review of the 1993 International Logo Conference	8
SCANTOOL	10
Cover Graphic: OWL	13
Review of MicroWorlds	14
Teaching LogoWriter Programming	16
Interactive Problem Solving - Review	20
Secret Messages.....	20
DOODLE : A Macintosh LogoWriter for Toddlers.....	21
Animals and Elements : Binary Decision Programs	22
How to Run a Sports Day	26
Dr Turtle.....	27

**An Integrated Unit of
Work Using LogoWriter
for Year 6**

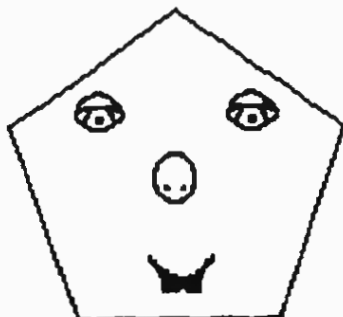
**Lyn Tritton
Wesley College
(Elsternwick Campus)
Victoria**

This paper describes an integrated unit of work around the novel *The Phantom Tollbooth* by Norman Juster. I have chosen this novel because it is one of the books suggested for the Year 5/6 Literature Course.

Chapter 14, *The Dodecahedron Leads The Way* would be appropriate for providing computer programming experiences using LogoWriter to students in Year 6. In this chapter we meet the Dodecahedron:

As they argued, a most peculiar little figure stepped nimbly from behind the sign and approached them . . .

He was constructed (for that's really the only way to describe him) of a large assortment of lines and angles connected together into one solid many-sided shape - somewhat like a cube that's had all its corners cut off and

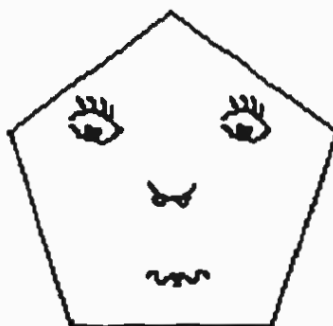


then had all its corners cut off again. Each of the edges was neatly labelled with a small letter, and each of the angles with a large one. He wore a handsome beret on top, and peering intently from one of his several surfaces was a very serious face . . .

When he reached the car, the figure doffed his cap and recited in a loud clear voice:

*'My angles are many
My sides are not few.
I'm the Dodecahedron
Who are you?'*

'What's a Dodecahedron?' inquired Milo, who was barely able to pronounce the strange word.



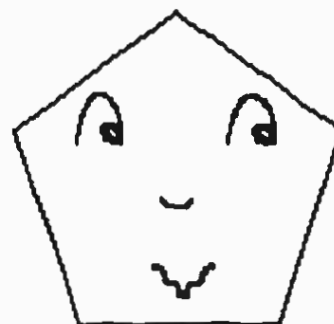
'See for yourself,' he said, turning around slowly. 'A Dodecahedron is a mathematical shape with twelve faces.'

Just as he said it, eleven other faces appeared, one on each surface, and each one wore a different expression.

Aim of the Unit

The students will create pentagonal faces and write text to tell a story using LogoWriter. The ability of

the individual students will determine how many faces they create. The more able students will probably be able to create twelve pentagonal faces with text to tell a story or a poem for each expression. Extension work would be to create faces using triangles or circles and write text, create backgrounds for the shapes, use colour, compose music to accompany the



pages and use animation. The content area is Literature and Technology based and the processes the students will use are: Mathematics/geometry, Language (reading, writing, LogoWriter, thinking, spelling), Art/graphics, Music.

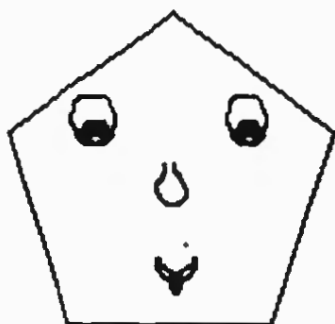
Knowledge To Be Gained

- The characteristics of a pentagon.
- How to develop an algorithm which is a set of instructions that can be followed to draw the shape.
- How to write a program in LogoWriter that will draw a pentagon.
- How to solve problems.
- How to write and illustrate a picture story book using LogoWriter.
- Abstract thinking.

Other Curriculum Areas Also To Be Developed

English:

- Reading the story *The Phantom Tollbooth* by Norton Juster.
- Writing text to accompany the changing faces of the Dodecahedron.
- Developing word lists of the following:
 - characters in the story
 - mathematical terminology - shapes, angles
 - facial expressions to be created
 - names of precious stones



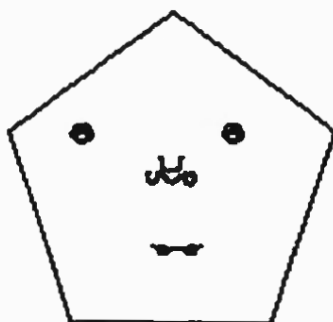
- Comprehension by understanding the Dodecahedron's character and being able to write suitable text to accompany the facial expressions.
- Using some of the words in the word list to develop a spelling list.

Social Education:

- Working in co-operative groups.

Graphics:

- Designing the layout of each page and the whole story.
- Use of colour.
- Animation for the graphics if appropriate.



Music:

- Creating musical accompaniment for the story.

Evaluation of the Unit

- The processes the students experience as they work through the unit - on-going assessment.
- The finished product/the faces created, the accompanying text, background, use of colour, animation and music. These will be assessed according to the ability of each student.
- What the students learnt as a result of the processes they experienced.
- The enjoyment and knowledge gained.
- The students will keep a Learning Journal of their experiences while creating the faces and the story. This will also include what they have learnt.

Stage 1 Tuning In

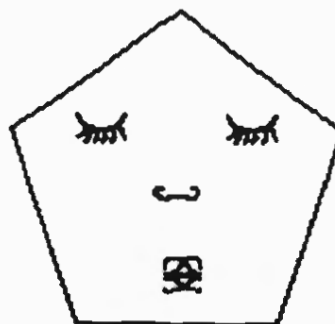
This is to get the students engaged in thinking about the topic.

1. Introduce the book to the students. Depending on the ability of the students the story could be read by the teacher to the whole class as

part of the Literature program, or the students could read the story themselves.

2. The focus chapter for this unit is *The Dodecahedron leads the way*. This chapter has been chosen because of the potential to teach geometry in a meaningful context.

3. After reading Chapter 14 the students work in pairs or groups of three or four and list the mathematical concepts introduced in this chapter, e.g. different words for distance, angles, pentagon, twelve shapes, problem solving, fractions. Make a class list of the mathematical concepts. Ask students for suggestions about what they could create using Logo-Writer. List these suggestions on a chart or on an overhead. From these suggestions work out what type of programming they could do to create a story using the twelve faces of the Dodecahedron.

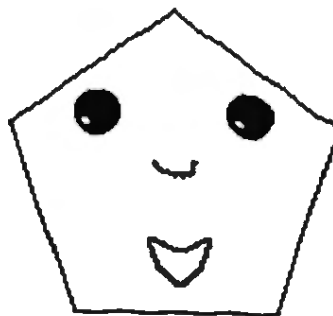


Stage 2 Finding Out

Activities that will prepare the students to write the LogoWriter program for the faces of the Dodecahedron:

1. In pairs list the characteristics of a pentagon e.g. five

equal sides, five equal angles, can be any size depending on the length of the sides, the size of the angles does not change. Check their findings with another pair and then report back to the class. A class chart is then developed for future reference.



face. This activity will illustrate the importance of clear instructions and co-operation. Each group of four will discuss which instructions were clear and which ones need to be fixed up. This is a good activity for self evaluation and appraisal.

2. Create a pentagon using straws and pipe cleaners (the pipe cleaners are used as the joiners where the angles meet and the straws used as the sides).

flip side using the repeat command.

eg.
TO PENTAGON
REPEAT 5 [FD 50 LT 72]
END

Now each student will have the skills to be able to create the Dodecahedron faces. The number of faces each student will create will depend on his or her ability and should be evaluated accordingly.

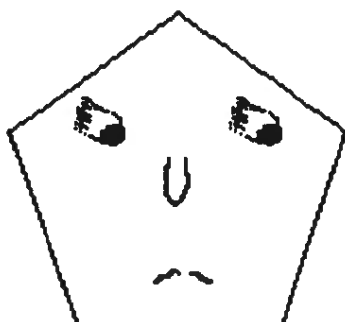
3. Create a 3D pentagon using pipe cleaners and straws, with a partner.

Write a procedure for a pentagon using variable inputs such as:

TO PENTAGON :SIZE
SETH 90
REPEAT 5 [FD :SIZE LT 72]
END

Stage 3 Sorting Out

During this stage the students will work on their own faces and stories to create a picture story book that can be published and read to others in the class or to students in the lower grades. Students who need extension work can animate the faces, use colour,



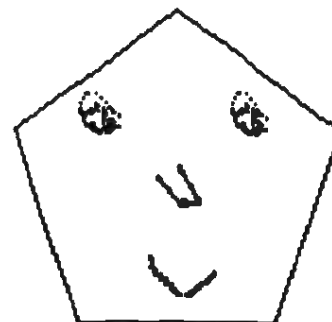
This procedure will draw the shape of the faces for the Dodecahedron.

4. Draw a pentagon on the ground using chalk. Then write an algorithm that will give a robot directions on how to walk out the shape of a pentagon. By this stage the students should have an understanding of the characteristics of a pentagon.

Next students have to investigate the shapes page to enable them to draw the expressions on the faces that they create. This activity is best done in pairs teaming a more able student with a less able student. They will work together using the Shapes page of LogoWriter to create a face on the pentagon for the Dodecahedron. Write this procedure on the flip side, then write instructions for how they created their Dodecahedron face and pass it on to another pair to see if they can duplicate the same

5. Using LogoWriter work out the commands for drawing a pentagon. Have the students do this in the Command Centre so they have the visual experience.

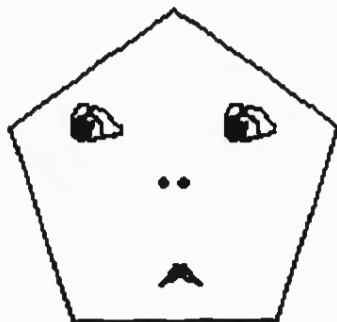
6. When they have mastered this write the procedure on the



design a background for each page, use music to depict the moods of the facial expressions, write more text for each face, or even write poems for each face or about the twelve faces.

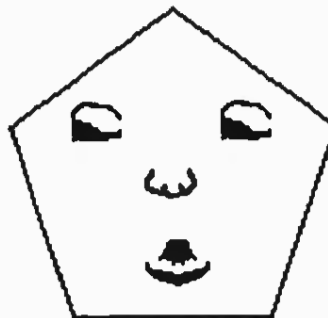
Stage 4 Sharing Stories With An Audience

The purpose of this learning experience is for the students to share the completed book with an audience. The processes that were explored while achieving this aim are also just as important as the finished product and these processes should also be assessed in terms of what the students learnt by producing their stories. Once the stories have been completed the



students could share the stories on the television monitor. Copies could also be made for the class library and the school library. After the students have watched the stories on the monitor they may want to add special features to their own story. One of the best ways of learning is to learn from your peers.

Many other activities could be added to those outlined. For



example, I have not elaborated on the Language Area, such as how I would develop spelling lists from the word lists that the students compiled, the discussions based on the story that develop comprehension or the processes the students would go through before they write the text for each page of their story.

Reference

Juster, N. *The Phantom Tollbooth* (1962) Lion.

FOUR New Logos Released!

Paul Nicholson
Deakin University
(Rusden Campus)
Victoria

Four new versions of Logo have been released recently. The really good news is that they are all free!

Brian Harvey of the University of California at Berkeley has produced versions of Berkeley Logo for Macintosh, IBM and Unix computers. The release includes a detailed, formatted manual and

all the exercises from his book, *Computer Science Logo Style*, as well as the source code for Logo itself.

The really nice feature of Berkeley Logo is that the file formats are identical across the three versions. This means that you could start working on a Logo program on an IBM, save your work to a floppy disk and later continue working on your file on a Macintosh with a FDHD drive and DOS Mounter or similar software installed. In a networked classroom it means that students would not have to worry about the type of computer they used to work on Logo - their files would function on both IBM and Macintosh computers. This is also true of the Unix version but most schools would not have suitable hardware to run it.

There are two variants of the IBM Logo. One runs on PC's and XT's with 640k of memory. The other runs on AT's or higher and uses extended memory to increase the size of the programs that it can execute.

The fourth Logo runs under Windows 3 on IBM computers. It has some differences from Berkeley Logo.

All the Logo versions are freely available on the SECAP computer bulletin board (03 544-1513) in the Logo files area. Modem speeds up to 9600 baud and MPN4 are supported.

**A Review of the 1993
International Logo
Conference**

**Debora Goldman
Mount Scopus
Memorial College
Victoria**

*The who's who of Logo
Came to speak, teach,
enthuse;
They left us with plenty
That we can now use.*

The highlight of 1993 for Logophiles who could be in Melbourne in July was the International Logo Conference, a conference dedicated to the use, exploration, concepts and research of Logo, and hosted by the Methodist Ladies' College, Kew.

Listening to, communicating with, and having the opportunity to meet international gurus as well as our own experts reinforced my belief that there are a lot of people who use Logo effectively.

The focus of the conference seemed to me to be to pull together people of all ability levels in Logo, to re-evaluate how, why and what to teach. Knowledge links were questioned. Should we examine the appropriation of information by students through children's eyes? I felt the need to go back to basic questions, to examine the purpose and content as well as the methods of teaching. Had I been conditioned by an adult

world into presupposing how children learn? Had I been driven into a mind set? Does Logo alone provide the stimulus? Could there be some other vehicle for learning cognitive skills? Were the experts really saying that students can learn more than we traditionally have said they could?

Some of the answers lay in how Logo can be seen to service many areas of teaching and learning. It can be seen to promote problem solving skills, generate an understanding of language rules, and reinforce, extend and highlight the thinking, learning and teaching processes.

For me the highlight of the conference was listening to Idit Harel. Her insight into "Debbie's case", that of a young non-achiever in the conventional class, and the difference made to her education, personality, and overall persona when she developed her learning skills of fractions through the use of Logo, reinforced my beliefs that Logo, used correctly, can challenge the way we teach and the way students learn. It is not enough to say that computers should be used to enhance current educational practices. Rather we need to re-evaluate what and how we are teaching.

Another insight came through the workshops for the use of Logo and language. Again Logo was shown to enhance

and challenge the way we assume students take language skills on board. I found many answers to my problems, but also found that there were many more questions raised. The challenge that was thrown to the participants was to explore new ways of using Logo, and possible ways of introducing Logo into a traditional school setting.

The legacy for me was to question the timing of teaching certain mathematical concepts, for example, when should directed numbers and the full Cartesian plane be "taught"? Concepts such as SETPOS and SETH that I once left until much later are now being introduced earlier than usual. The result is that students are exploring, devouring, and learning freely. I now make no assumptions about limiting the students' knowledge. They rise to the challenge and are keen to explore and build their own knowledge maps.

There are always complaints that can be levelled at any conference. Mine are that the time was too short and there were too many wonderful sessions from which to select!

If you would like a copy of the 1993 International Logo Conference Proceedings disk (\$20 each) please contact:

Community Education,
MLC
207 Barkers Road
Kew VIC 3101
Ph 274 6333, fax 819 2345

**1993 International Logo Conference
Keynote and International Speakers and Topics**

Paul Goldenberg - Modelling Language with Logo

Idit Harel - Constructing Constructionism

Brian Harvey - Computer Science: Logo and Its Competitors

Barry Newell - Fuzzy Teaching

Linda Polin - Why I Haven't Given Up on Logo

Gary Stager - Logo and the Next Generation: MicroWorlds

Leslie Thyberg - LogoWriter and Artful Scribbling

Dan and Molly Watt - Action Research by Logo Teachers



**From left to right: David Loader, Dan Watt, Idit Harel, Gary Stager,
Molly Watt, Brian Harvey, Leslie Thyberg, Margaret Fallshaw, Barry
Newell, Linda Polin**

SCANTOOL

Jenny Betts
John Paul College
Queensland

The Question: What possessed a 12-year-old child to create a tool such as this?

The Answer: The desire to be in control and to instruct the computer to do exactly what the child wanted it to do.

Instilling these ideas into the minds of the students I meet is a priority of mine. I believe very strongly that the computer is an extremely powerful tool; however, many users tend to let the computer dominate them in areas where it should not. Instead, the user should dominate the power that the computer is able to generate.

The tool procedures labelled SCANTOOL were written by the same student who wrote DRAWTOOL [see last issue - Eds.], approximately nine months later. Why? Because there were times when we would use a flatbed scanner to scan pictures from magazines, only to find that they were in the wrong position when they were finally placed onto the LogoWriter page. We could fiddle with the workings of the scanner itself so that after many attempts at rescanning, the picture would eventually be located in a near-enough position, and that we did. However we were still unable to control the sizing, or to position the graphic in the exact place on the page itself. Wasting time, attempting to scan and rescan pictures in order to get the right look, was a problem that had to be overcome, as students were choosing not to use the scanner simply because they did not want to waste the hours it took to get one picture right. For these reasons this child decided it would be easier to write a set of tool procedures to move the image anywhere on the LogoWriter page as well as enlarging or shrinking the graphic.

The following procedures are written in LogoWriter for IBM.

How To Use the SCANTOOL

1. Go to a new page on which you wish to work.
2. Type in the command centre GETTOOLS "SCANTOOL. The computer will get the procedures on the SCANTOOL page and treat them as tool procedures. If you are unfamiliar with tool procedures, consult your LogoWriter manual.

3. When you are ready to scan, type START in the command centre.

or

Load your graphic and type SCAN in the command centre.

The SCANTOOL Procedures

TO START

```
CC
SHOW [LOAD PICTURE AND TYPE
SCAN.]
END
```

TO START.DRAW

```
TELL 0
PU SETPOS :POS
FASTTURTLE
SETH 180 PU
MAKE "NUM 1
MAKE "DUB 0
REPEAT :DDIST1 [SETY (LAST :POS)
MAKE "NUM1 1 REPEAT :DDIST
[DRAW1] SETH 90 FD 1 SETH 180
MAKE "DUB :DUB + 1 IF :DUB = 2
[MAKE "DUB 0] MAKE "NUM :NUM +
1 IF :SIZE = "H [MAKE "NUM :NUM +
1] IF :SIZE = "D [IF :DUB = 1 [MAKE
"NUM :NUM - 1]]]
```

```
CC
END
```

TO DRAW1

```
SETC (ITEM :NUM1 THING (WORD
"LINE :NUM))
PD FD 1 PU
MAKE "NUM1 :NUM1 + 1
IF :SIZE = "H [MAKE "NUM1 :NUM1 + 1]
```

IF :SIZE = "D [MAKE "NUM1 :NUM1 - 0.5]
END

TO SCAN
CLEAR NAMES
CC PU

MAKE "DIST2 11
TYPE [MOVE THE TURTLE TO THE TOP
LEFT HAND CORNER OF THE PART
YOU WISH TO SCAN AND PRESS
ENTER. USE F & S TO CHANGE THE
DIST THAT IT TRAVELS.]

SCAN.MOVE
MAKE "POS1 POS
CC

TYPE [MOVE THE TURTLE TO THE TOP
RIGHT HAND CORNER OF THE PART
YOU WISH TO SCAN AND PRESS
ENTER. USE F & S TO CHANGE THE
DIST THAT IT TRAVELS.]

SCAN.MOVE
CC

MAKE "POS2 POS
TYPE [MOVE THE TURTLE TO THE BOT-
TOM LEFT HAND CORNER OF THE
PART YOU WISH TO SCAN AND
PRESS ENTER. USE F & S TO
CHANGE THE DIST THAT IT TRAV-
ELS.]

SCAN.MOVE
MAKE "POS3 POS
MAKE "DIST (LAST :POS1) - (LAST
:POS3)
MAKE "DIST1 (FIRST :POS1) - (FIRST
:POS2)
IF (FIRST :DIST) = "-" [MAKE "DIST MINUS
:DIST]
IF (FIRST :DIST1) = "-" [MAKE "DIST1
MINUS :DIST1]
CC
TYPE [PLEASE WAIT! SCANNING]
START.SCAN
END

TO SCAN.MOVE
NAME READCHAR "CH
IF :CH = "H [SETH 0 FD :DIST2]
IF :CH = "M [SETH 90 FD :DIST2]
IF :CH = "K [SETH -90 FD :DIST2]
IF :CH = "P [SETH 180 FD :DIST2]

IF :CH = "F [MAKE "DIST2 :DIST2 + 5]
IF :CH = "S [MAKE "DIST2 :DIST2 - 5]
IF :CH = CHAR 13 [STOP]
OUT?
SCAN.MOVE
END

TO OUT?
IF (FIRST POS) > 155 [BK :DIST2]
IF (FIRST POS) < -154 [BK :DIST2]
IF (LAST POS) > 90 [BK :DIST2]
IF (LAST POS) < -89 [BK :DIST2]
END

TO START.SCAN
ST
FASTTURTLE
PU SETPOS :POS1
SETH 180
MAKE "NUM 1
REPEAT :DIST1 [MAKE "LINE " LINE
MAKE (WORD "LINE :NUM) :LINE
SETH 90 FD 1 SETH 180 MAKE
"NUM :NUM + 1 SETY (LAST :POS1)]
CC
SHOW [WHEN READY TO START
DRAWING TYPE "DRAW"]
END

TO DRAW
CC PU
TYPE [HALF, DOUBLE OR THE SAME
SIZE? (H, D OR S)]
NAME READCHAR "SIZE
IF NOT MEMBER? :SIZE [H D S]
[DRAW STOP]
TYPE [COLOUR PINK, WHITE OR
BLUE? (P, W B)]
NAME READCHAR "COLOUR
IF NOT MEMBER? :COLOUR [P W B]
[DRAW STOP]
MAKE "DDIST :DIST
MAKE "DDIST1 :DIST1
IF :SIZE = "H [MAKE "DDIST :DIST / 2
MAKE "DDIST1 :DIST1 / 2]
IF :SIZE = "D [MAKE "DDIST :DIST * 2
MAKE "DDIST1 :DIST1 * 2]
CC
TYPE [MOVE THE TURTLE TO THE
POSITION TO START DRAWING.]

USE F & S TO CHANGE THE DIST
THAT IT TRAVELS.]

```
SCAN.MOVE  
CC  
MAKE "POS POS  
TYPE [PLEASE WAIT! DRAWING]  
START.DRAW  
END
```

```
TO LINE  
REPEAT :DIST [VARIABLE  
  COLORUNDER FD 1]  
END
```

```
TO VARIABLE :COLOUR  
MAKE "LINE (WORD :LINE :COLOUR)  
END
```

How To Make the ASCII Symbols in the Procedure SCAN.MOVE Using the CHAR Reporter

There are a few methods of using LogoWriter to program the computer to carry out instructions when an arrow key is pressed. This is the method that this child chooses to use in his work. CHAR will report the single ASCII code that represents a key, or a combination of keys, from the keyboard. For example, if you want to make the turtle respond when the arrow keys are pressed you will need to insert these codes into your procedure. Look at the procedure SCAN.MOVE.

```
TO SCAN.MOVE  
NAME READCHAR "CH  
IF :CH = "H [SETH 0 FD :DIST2]  
IF :CH = "M [SETH 90 FD :DIST2]  
IF :CH = "K [SETH -90 FD :DIST2]  
IF :CH = "P [SETH 180 FD :DIST2]  
IF :CH = "F [MAKE "DIST2 :DIST2 + 5]  
IF :CH = "S [MAKE "DIST2 :DIST2 - 5]  
IF :CH = CHAR 13 [STOP]  
OUT?  
SCAN.MOVE  
END
```

How To Use the Computer To Show the ASCII Symbol Which Represents the Key, or Combination of Keys, Being Pressed.

1. In the command centre type: SHOW READCHAR <enter>.
2. The computer pauses and waits for you to press a key, or a combination of keys. For example, press the up arrow.
3. You will see, in the command centre, a single ASCII code which is represented by " H" - This is a SPACE and the letter "H".
4. Place the cursor over the ASCII code.
5. Press "select button" (F1).
6. Highlight the whole character.
7. Press "copy button" (F3).
8. Move the cursor to the required position in the procedure.
9. Press "paste button" (F4).

NOTE:- The symbols " H", " M", " P" and " K" are not actually letters but rather symbols which represent the arrow keys of the keyboard.

H represents ↑	P represents ↓
M represents →	K represents ←

Concluding Remarks

I believe in working in a Logo Environment, armed with both the DRAWTOOL and the SCANTOOL. However, careful decisions must be made about when it is appropriate to use these tools. You should analyse the outcomes and expectations of the activity your students are undertaking, then make a decision. There are times when I specifically ask the students not to use the tools.

I find Logo brilliant because of the tools that are available for students to use. Most of these are found on the master disk. Those that are not, can be written by you, or preferably by your students.

Cover Graphic: OWL

The design on the cover of this issue of LogoFile was developed from the Logo procedure OWL, by Laura Dawes of Bateman Primary School in Western Australia. Laura's procedure was written for a BBC microcomputer. All of the lengths in the original procedure have been scaled to change the size of the owl; the lengths in the code given below produce an owl that fits conveniently on the screen of a Macintosh PowerBook.

TO OWL

HT

PU BK 50 PD LT 85

CIRCLE 10

ARC 5 10 WING 10

ARC 15 10 WING 10

ARC 10 10 CLAW 2

ARC 4 10 CLAW 2

RT 170

LARC 2 10

LT 85 BREAST

END

TO TRI :SIZE

REPEAT 3 [FD :SIZE RT 120]

END

TO FEATHERS :SIZE

LT 175

REPEAT 2 [ARC 18 :SIZE LT 180]

LT 10

REPEAT 4 [LARC 18 :SIZE RT 180]

RT 10

REPEAT 2 [ARC 18 :SIZE LT 180]

RT 175

END

TO ARC :STEPS :SIZE

REPEAT :STEPS [FD :SIZE RT 10]

END

TO LARC :STEPS :SIZE

REPEAT :STEPS [FD :SIZE LT 10]

END

TO BREAST

PU FD 36 PD

FEATHERS 6/5

PU FD 12 PD

FEATHERS 1

PU FD 12 PD

FEATHERS 4/5

PU FD 16 PD

LT 30 TRI 12 RT 30

PU FD 28 LT 90 FD 11 RT 185 PD

CIRCLE 1

PU LT 5 FD 22 RT 5 PD

CIRCLE 1

END

TO CIRCLE :SIZE

REPEAT 36 [FD :SIZE RT 10]

END

TO CLAW :SIZE

LT 5

LARC 9 :SIZE

RT 190

ARC 18 :SIZE

RT 170

LARC 9 :SIZE

LT 85

FD 6 * :SIZE

BK 6 * :SIZE

RT 90

END

TO WING :SIZE

PETAL 9 :SIZE

FD :SIZE RT 10

PETAL 7 :SIZE

FD :SIZE RT 10

PETAL 5 :SIZE

END

TO PETAL :STEPS :SIZE

REPEAT 2 [ARC :STEPS :SIZE RT (180 -
:STEPS * 10)]

END



Thanks to Paul Dench for sending us this one. Please send interesting graphics for possible use as future cover designs to the Editors at the address on page 3.

MicroWorlds - That Sounds Familiar

David Rasmussen
Victoria

Anyone who has been "Mindstormed" will have heard of microworlds, those fuzzily defined microcosms of exploration and investigation that have been a part of the Logo philosophy since the turtle was a terrapin. Well, now there is a new product from LCSi, the developers of LogoWriter, called MicroWorlds. The name is not a coincidence, I'm sure, since one of its purposes is to encourage the user to build microworlds, or Projects as they are called here. And, to encourage a novice Logo user (and, indeed, some old hands), much of the hard work of animation, music, colour, etc. has been "built-in", to enable users to get on with their project building. There are many other features that teachers and students will be familiar with, such as buttons (as in HyperCard) and drawing tools and palettes (as in Kid Pix).

In case you're about to turn the page because you think your standard of Logo may be well above this, let me assure

you that all of the programming primitives of LogoWriter are there plus many more, including many that you would have wished for in previous Logos.

There is one problem however. It is only available for the Macintosh platform. The PC version will be released sometime in July 1994. It will be a Windows version and will be well worth waiting for.

MicroWorlds has at its centre, a package called MicroWorlds Project Builder. Accompanying this are two support packages, MicroWorlds Language Arts and MicroWorlds Math which can be purchased separately. These, and other packages to be developed by LCSi, are similar in concept to the support material that accompanied LogoWriter, if you purchased the full kit. I'll discuss those in future articles.

When MicroWorlds is loaded you'll see the familiar LogoWriter page. There is the drawing and text screen above and the command centre below. But there the similarities cease. A tool palette is showing on the right of the command centre. (It can be made to hide or show

from the Gadgets menu bar as in many other Mac drawing programs). Click on the Moon face icon and the command centre is transformed into a shape table, full of editable costumes for the turtle and some costume tools (see Fig. 1). If you don't like a costume you can design your own in a shape editor that has 256 colours, as well as rotation and flipping tools.

Other tools showing in the palette are for painting, writing procedures, hatching a new turtle (the number is only limited by the memory size of your Mac), text boxes, buttons, sliders and, in the top left corner, the pointing tool. Can you identify them all? The tools for turtles, text boxes, buttons and sliders are known as Objects. Clicking on the paint tool changes the command centre menu to show a selection of familiar drawing tools and colours. Click on the button tool and you get a dialog box waiting for your instructions for the button to be typed in.

The last tool in the bottom row of the tool palette is the slider, a unique tool that allows the user to control the speed with which something happens, by pushing a horizontal slider to the right (faster) or to the left (slower).

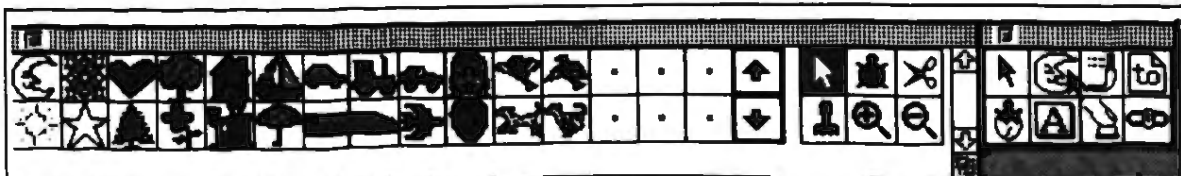


Figure 1: The Microworlds Tools Palette

(see Fig. 2). After clicking on this tool and clicking on the screen you can change the default name of the slider and set its speed range.

Next to the Tool palette (Fig. 1) in the shapes window are some tools for turtle house-keeping. The turtle tool allows you to change the costume of any turtle already

Twinkle Little Star" play in the background? Everything stopped while one turtle moved or one note played. Well now it can all happen at once without any stoppages, depending on your memory size. In the scene following called Dance, all of the people can dance at once or individually, simply by clicking on a dancer. Meanwhile the music

LogoWriter on the Mac) via a microphone can be entered and saved and "original" keyboard music can be composed (using a synthesiser of sorts) and saved for incorporation into your project.

There is more, of course, but it would be better for you to discover all of the great things you and your students can do

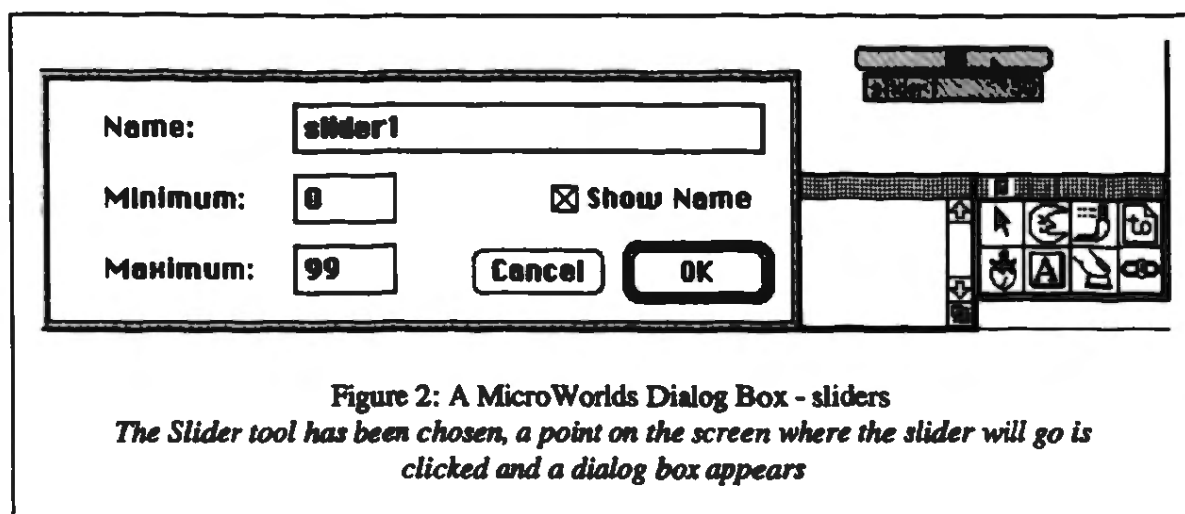


Figure 2: A MicroWorlds Dialog Box - sliders

The Slider tool has been chosen, a point on the screen where the slider will go is clicked and a dialog box appears

showing. The scissors "cut" out any object you want removed and the rubber stamp does what you'd expect. The most exciting tools here are the magnifying glasses which can magnify a shape (or shrink it), something I believe every LogoWriter user has wished for.

One of the big advantages, in my mind, of MicroWorlds over other Logos, is the concept of parallel processing. In simple terms, this means the ability to have more than one action happening in real time. Remember how you have tried to make three turtles move independently while having "Twinkle

plays endlessly when the "dance" button is clicked.

The slider bar called "speed" controls the male dancer second from the top on the right. Moving the slider right or left controls the size of the step he takes - all while the other people are dancing. All of the shapes were created in the shape table and this whole project was completed by a novice MicroWorlds user in about 2 hours.

The music fans have been well and truly catered for in MicroWorlds. There is a separate section under the Gadgets menu where both recorded sound (similar to

(such as making balloon text boxes). In my experience with both teachers and students, the excitement generated is a refreshing change from the frustrations experienced with other software when new users suddenly find themselves confronted with the familiar hill of learning before even a simple task can be performed. In keeping with LCSF's tradition, it's simple, even for Logo novices, to get some action happening, and fun too.

I wish I could say that MicroWorlds is a perfect educational tool. I can't. There are some frustrating features. Logo purists may

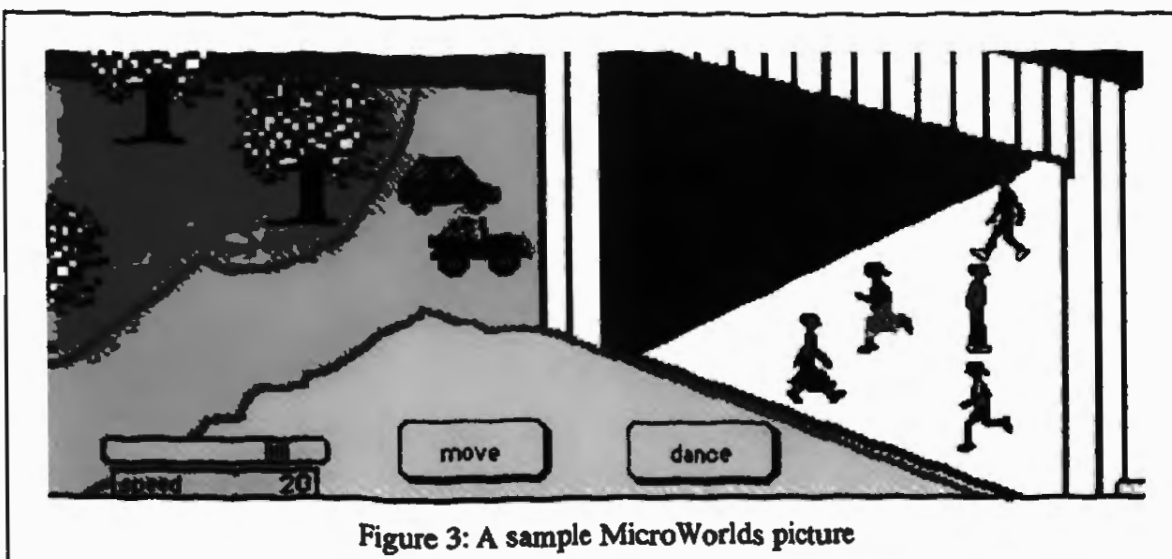


Figure 3: A sample MicroWorlds picture

gasp in horror at the automatic features that may appear to take the "thinking" out of Logo. The picture can become cluttered with buttons, sliders and text boxes (it would be nice to be able to make these invisible when not needed). And, most frustrating of all, for me, is the difficult process one has to go through to combine projects. I recently gave a group of teachers a project idea to work on individually, with the idea of joining all of their pages into one "book". Unfortunately you can only make pages within a project, and connecting projects together is very messy and tedious and not for the faint-hearted.

These grumbles aside, MicroWorlds is an exciting and colourful package, almost guaranteed to motivate even the most reluctant student. It provides all the necessary Logo programming tools as well as hypermedia capabilities. It should be included on any school's future software list. Buy it and enjoy it!

Teaching LogoWriter Programming

Jenny Callahan
Cleland
Secondary College
Victoria

As with any subject in the school curriculum, there are many different approaches that can be used in teaching computer programming. In analysing the characteristics of each of these approaches the following questions need to be considered:

- Who is in control of the activity, the teacher or the student?
- Who initiates the activities, teacher or student?
- Is the work goal oriented or exploratory?
- Is the teacher's guidance prescriptive or sought when needed?
- Is the programming being

taught from the top down or the bottom up?

In two papers, *Teaching Computer Programming* (1981) and *Approaches to Teaching Logo Programming* (1985), McDougall describes a total of five approaches to teaching programming:

1. Syntax Approach

The teacher presents statement types, a few at a time with those most easily understood or most commonly used presented first. Students are set programming exercises using these statements, a few at a time, to write programs or program segments.

In this approach the teacher is the controller and initiator of the activity in the classroom. The work could be generally described as goal oriented as students are presented with set exercises or projects to complete. This is an example of a bottom up approach to teaching computer programming.

2. Whole Program Approach

Students are presented with a complete pre-written program. The program is run, output displayed, and a discussion of the program follows. The various parts of the program are related to the outcome. Students can then be set to explore similar programs using the same commands.

In this approach the teacher initiates the activity in the classroom, but due to the more varied projects undertaken after the initial teaching period, control is given over to the learners. The work is more exploratory in nature. This is an example of programming being taught from the top down, with the teacher's guidance being sought when necessary.

3. Doodle, Design, Debug

The process is exactly as described. The learner doodles in Logo using basic commands, then begins to plan outcomes - design. The procedure or program is then modified or built on in the debug stage. Using this approach, the learner is in control of, and initiates the activity. The work is exploratory and the teacher's guidance can be sought when necessary. This differs from the whole program approach in that it is being taught from the bottom up. Learners only encounter procedures as they see a need for them. It is not

however as prescriptive as the syntax approach described earlier. In this approach, once the initial commands have been taught learners begin to take control of their own outcomes.

4. Turtle Humming

Learners use a small "phrase" and repeat and change it. The phrase is programmed by the learner using very simple commands. This is an approach where the learner has control of the situation but the activity is initiated by the teacher. The work can be classified as exploratory but is limited to exploring with the phrase. This approach is a further example of bottom up programming as learners begin with very simple commands.

5. Logo on the Run

The teacher follows the learner's inclinations. Whatever the learner wants the turtle to do, the teacher writes a program for that movement. This approach avoids students having to learn Logo syntax initially, however it could take a large amount of the teacher's time especially in initial stages. While learners are in control of what they do on the computer, and initiate their own work, they are very dependent on the teacher (at least initially) to write the programs for them. The work in this case is exploratory and the teacher's guidance sought when required. This is an approach where programming

is taught from the bottom up.

Teaching involves a constant search for the "best", most effective way to create environments in which learners develop knowledge and skills. In my experience, the answers to the following questions play a major role in determining what will be the most effective method of teaching any particular topic:

- Why am I teaching this topic?
- Who am I teaching?
- What are the desired outcomes of my teaching?
- What facilities are available to aid in teaching this subject?

Why Teach Computer Programming?

In his book *Mindstorms*, Seymour Papert states ;

In my vision, the child programs the computer and in doing so, both acquires a sense of mastery over a piece of most modern and powerful technology and establishes contact with some of the deepest ideas from science, from mathematics and from the art of intellectual model building. (Papert, 1980, p. 5)

In the years since *Mindstorms* was first published there has been a wealth of literature and research to show that children learn best when they are interested, involved and in control of the learning process. Computer programming is a powerful medium which teachers can utilise to place learners in

control of the learning process. As such, it is a valuable subject in itself, and a valuable aid to learning across a variety of subjects.

In deciding how to teach programming, the teacher should keep in mind that much of the value of learning programming is gained by placing the student in control of that learning.

To Whom Am I Teaching Programming?

As a secondary teacher I need to consider carefully the experience and abilities of all the students in my class. As a rule the group will have a diverse range of computer experience. Some may already have learned programming in primary school or elective programs in secondary school. Some may have had very negative experiences, as the Year 12 student who walked past one day, looked at the computer screen in front of me and said, "Oh, that's that Logo stuff. I never understood that." Others may be extremely skilled and have no need to be taught anything. As the majority of the research on Logo seems to have been carried out in primary classes, these conditions facing the secondary teacher have not been researched or discussed in any great depth. They are not however atypical of a secondary class in any subject. In all subjects we encounter students with a wide range of prior experi-

ences and knowledge, and we have developed skills to teach each of these students.

What Are the Desired Outcomes of Teaching Programming?

The desired outcome is an improvement in the student's learning and motivation. Whether programming is being taught purely as part of a computer studies course, or it is being taught so that students can use it for Maths, English, Science, etc., it can be a valuable tool in enhancing both learning and motivation.

What Facilities Are Available for Teaching Programming?

Most secondary schools would have at least one computer room with at least enough computers for two to a computer. At Clelland Secondary College (where I work as a Maths and PE teacher) we have two fully equipped rooms with IBM compatible computers. There are sufficient computers in each room for a computer between two students. These all have LogoWriter on them. We also have six Macintosh computers in a literacy centre, however these do not have LogoWriter. The logistics involved in organising the use of either of the computer rooms for any ongoing programme are quite prohibitive. I would have to match when my class is timetabled against

a non-computer class in a computer room and organise a weekly room swap. While this is not impossible, it is a far cry from an ideal situation.

If it were found that my class could not achieve a regular time in a computer room, this might affect the teaching approach chosen. If I were confined to a room with a blackboard, yet wanted to teach programming, perhaps a lesson on syntax would be better than no lesson at all. This could then be followed up by a lesson in a computer room at a later time.

Which Method of Teaching Programming Would Contribute to the Most Effective Learning in Students?

As with all teaching and learning environments, there is no "best" method that works all the time. As a teacher I firmly believe however that when the learner is in control of the learning environment and determining the outcomes, experience shows that he or she will learn far better than when the environment is externally controlled. If we are after understanding, if we wish students to be motivated to learn, we must choose the teaching methodology that best achieves this.

In her book *Children Designers*, Harel (1991) describes three Logo learning environments which she created as part of a research project:

1. Isolated Logo

Students learnt Logo in the school computer lab. They were taught by the computer-room coordinator, and did short programming exercises and assignments on which they were then graded.

2. Integrated Logo

Students worked in an open area near the classroom. Work was project oriented, with programs being integrated into a specific curriculum. Work was graded in a highly subjective manner taking into account the students' input, and the final projects.

3. Software Design Logo

This was an extension of the Integrated learning environment. The students worked on one project over the term of the study: to write a program to teach younger students fractions. They were designing a real product for real people.

The environments set up by Harel show examples of the syntax approach - Isolated Logo, and the whole program approach - Integrated Logo and Software Design Logo.

While the research described in *Children Designers* was not set up solely to compare outcomes from different approaches to teaching Logo, the results showed clear differences in learning outcomes. The students involved

in writing real programs performed better in post-tests than the other two groups, while those in the Integrated Logo environment had learned more in all areas than those in the Isolated Logo group.

The results of Harel's research suggest that children learned best when they were involved in a rich, meaningful, and complex task, working towards designing and programming a *real* product for *real* people.

Which approach achieves the most effective outcomes? In my discussions I have concentrated mainly on the whole program approach and the syntax approach, as I feel that the other three approaches are more appropriate for primary school students.

It is always tempting to take the approach that keeps the teacher in control of the learning environment. The uncertainty of twenty-five students undertaking twenty-five different projects can be daunting to the best of teachers at the best of times. However, the approach in which learners have control of their learning, in which learners are being taught to read and understand whole programs before writing their own, and in which learners are determining their own goals would seem to be the approach that will produce the most effective learning. As a teacher, keeping in mind constraints already mentioned, I would

ideally seek to create an environment where the students were learning programming by the whole program approach and working on projects that were meaningful to them.

References

- Harel, I. (1991) *Children Designers* Ablex Publishing Corporation.
- McDougall, A. (1981) Teaching Computer Programming. In Rogerson, A. (ed.) *Mathematics: Myths and Realities* Mathematical Association of Victoria, 370-3.
- McDougall, A. (1985) Approaches to Teaching Logo Programming. In Duncan, K. and Harris, D. (eds.) *Computers in Education* North-Holland 623-7.
- Papert, S. (1980) *Mindstorms: Children, Computers and Powerful Ideas* Harvester.

BOOK REVIEW

Debora Goldman
Mount Scopus
Memorial College
Victoria

Interactive Problem Solving Using Logo

by Heinz-Dieter Boeker, Hal Eden, Gerhard Fischer, published 1991 by Lawrence Erlbaum Associates

ISBN 0-8058-0305-X (cloth) 0-8058-0306-8 (paper)

Some weeks ago, while waiting for my children at the library, I browsed through the latest books received. There, amongst the fiction and children's references, shone a gem: **Interactive Problem Solving Using Logo**. Just by skimming the book I could see how valuable this could be, for teachers, educators and even students not very familiar with advanced Logo.

The book is divided into seven parts. The first part takes the reader through fundamentals in Logo programming: procedures, debugging, variables both local and global, and list processing. Each chapter contains clear illustrations of concepts as pictures, flow charts, or "maps". There are sample programs with outputs along the way and at the end of each chapter there are exercises.

In fact, although the topic areas change, the format is the same throughout: clear illustrations to explain a point being made, sample procedures, and exercises.

Part Two looks at mathematical topics such as prime numbers, greatest common divisor, change of base, recursion, and problem solving.

Part Three is for the computer scientist examining sorting, pattern matching, simulations, formal languages, and so on. Part Four looks at Artificial Intelligence, Part Five at Linguistics,

Six at games, and Seven at new developments in computers and education such as object-oriented programming, and extensions to programming in Logo.

The five appendices contain primitives used in the book (with explanations), utility procedures, and additional resources for extra reference.

As a resource this book is excellent. I have yet to explore its capabilities as a teaching tool. My estimation is that it is easy to read either as a whole or jumping from section to section. It is definitely a book well worth reading, if not owning.

Secret Messages

Peter J Carter
Lockleys, South Australia

One of the easiest ways of enciphering messages is to substitute one letter for another, perhaps m for a. The idea has been used for centuries, often with a pair of discs with the letters around their peripheries. These procedures do the same thing with a pair of lists. Rotate the lists before you start, eg. ROTATE 14

```
TO ENCIPHER :MESSAGE
IF EMPTY? :MESSAGE [OP []]
OP SE ENCIPHERWORD FIRST
:MESSAGE ENCIPHER BF
:MESSAGE
END
```

```
TO ENCIPHERWORD :WORD
IF EMPTY? :WORD [OP ""]
OP WORD SWAP FIRST :WORD
:ALPH :SUBST ENCIPHERWORD
BF :WORD
END
```

```

TO DECIPHER :MESSAGE
IF EMPTY? :MESSAGE [OP []]
OP SE DECIPHERWORD FIRST
:MESSAGE DECIPHER BF
:MESSAGE
END

```

```

TO DECIPHERWORD :WORD
IF EMPTY? :WORD [OP "]
OP WORD SWAP FIRST :WORD
:SUBST :ALPH DECIPHERWORD
BF :WORD
END

```

```

TO SWAP :LETTER :ALPH :SUBST
IF EMPTY? :ALPH [OP :LETTER]
IF :LETTER = FIRST :ALPH [OP FIRST
:SUBST]
OP SWAP :LETTER BF :ALPH BF
:SUBST
END

```

```

TO STARTUP
MAKE "ALPH [A B C D E F G H I J K L
M N O P Q R S T U V W X Y Z]
MAKE "SUBST [A B C D E F G H I J K L
M N O P Q R S T U V W X Y Z]
END

```

```

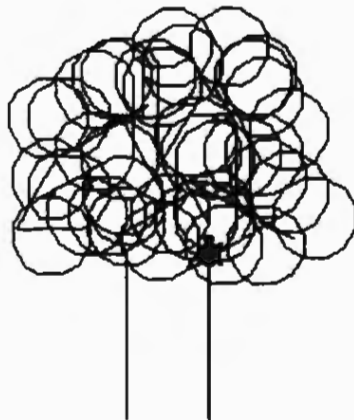
TO ROTATE :NUMBER
MAKE "SUBST ROTATEAUX :SUBST
:NUMBER
END

```

```

TO ROTATEAUX :LETTERS :NUMBER
IF 0 = :NUMBER [OP :LETTERS]
OP ROTATEAUX SE BF :LETTERS
FIRST :LETTERS :NUMBER - 1
END

```



*Graphic
produced with
DOODLE
procedures.*

DOODLE: A Macintosh Logowriter for Toddlers

Michelle Goodwin
Victoria

DOODLE is an adaptation for Macintosh LogoWriter of a program called TODDLER, a single-key Logo for the Apple II computer (from the book *Learning Logo on the Apple II* by A. McDougall, T. Adams and P. Adams - Prentice Hall, 1982)

To start the program running, type STARTDOODLE 10 (or some other number - the number sets the distance to be moved by the turtle on each command). Then single keystrokes can be used to move the turtle, change its colour, and control the screen. F, B, L and R move and turn the turtle. S, T and C make a square, a triangle and a circle respectively. Q, P and W clear graphics and control the pen and wrapping. Y is a fill command. The numbers 0 - 9 set different colours for the turtle. X stops the program.

```

TO STARTDOODLE :SPEED
CG SETBG 0 SETC 1
MAKE "ADD 0
MAKE "REM 0
MAKE "REMPEN 0
DOODLE :SPEED
END

```

```

TO GETCOMMAND :FAST
MAKE "COM READCHAR
IF :COM = "F [FD :FAST]
IF :COM = "B [BK :FAST]
IF :COM = "R [RT 30 MAKE "ADD :ADD +
30]
IF :COM = "L [LT 30 MAKE "ADD :ADD +
30]
IF :COM = "S [SQUARE :FAST + 20]
IF :COM = "T [TRI :FAST + 20]
IF :COM = "C [CIRCLE :FAST]
IF :COM = "Q [CG]
IF :COM = "P [PENS]
IF :COM = "W [WRAPS]
IF :COM = "Y [FILL]

```

```

IF :COM = "X [STOPALL]
IF :COM = 0 [SETC 0]
IF :COM = 1 [SETC 1]
IF :COM = 2 [SETC 2]
IF :COM = 3 [SETC 3]
IF :COM = 4 [SETC 4]
IF :COM = 5 [SETC 5]
IF :COM = 6 [SETC 6]
IF :COM = 7 [SETC 7]
IF :COM = 8 [SETC 8]
IF :COM = 9 [SETC 9]
END

```

```

TO PENS
MAKE "REMPEN
:REMPEN + 1

```

```

IFELSE :REMPEN = 1 [PU
[PD MAKE "REMPEN 0]
END

```

```

TO WRAPS
MAKE "REM :REM + 1
IFELSE :REM = 1
[WINDOW][WRAP MAKE
"REM 0]
END

```

```

TO CIRCLE :SIDE
REPEAT 20 [FD :SIDE RT
18]
END

```

```

TO SQUARE :LENGTH
REPEAT 4 [FD :LENGTH
RT 90]
END

```

```

TO TRI :LENGTH
REPEAT 3 [FD :LENGTH
RT 120]
END

```

```

TO DOODLE :SPEED
IF KEY? [GETCOMMAND
:SPEED]
DOODLE :SPEED
END

```

Animals and Elements: The Use of Binary Decision Programs in Teaching

Nicholas Derry
Carey Grammar School
Victoria

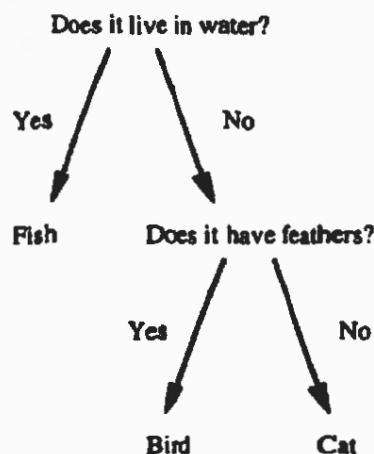
Introduction

The ability of Logowriter to deal with lists enables it to be used to work through complex binary structures. This article examines how this ability can be used as a teaching aid. A number of programs will be presented and comment made about their possible use in the classroom.

ANIMAL (Abelson, 1982)

Originally written in Logo, this program was adapted by the author to run in LogoWriter. The programs developed later draw considerably on the ideas in the ANIMAL program.

The hierarchical structure of this program employs a series of yes/no questions to decide which part of the program to use next, and hence ultimately to identify an animal the operator of the program has selected. This idea may be illustrated by the diagram below.



To run the ANIMAL program the 'knowledge' list has first to be supplied with an animal.

```

MAKE "KNOWLEDGE
"DOG
ANIMAL

```

The feature of this program is its capacity to learn. Imagine that the program has been started up with 'fish' as the only animal it knows. The program asks the user if the animal they were thinking of is a fish. If the answer is 'no', the program asks for a question that distinguishes a fish from the user's animal. In effect, the program is expanding the tree structure as shown in the example above.

ELEMENT

This is a simple adaptation of the ANIMAL program, involving the identification of an element rather than an animal. It runs in exactly the same way as ANIMAL, the only changes being the wording that comes up on the screen, and the change of one sub-procedure name to match the input.

Code for the ELEMENT program follows. To run it the 'knowledge' list has first to be supplied with an ele-

ment, then type ELEMENT. For example:

MAKE "KNOWLEDGE "OXYGEN
ELEMENT

The power of this program lies in its potential as a probe of student understanding. Students could be given a list of elements, for example, and asked to devise questions that would distinguish between pairs of them by means of yes/no questions. This would allow for a detailed analysis of understanding, particularly if the elements chosen were similar in their properties.

Of course, the potential of this program does not just lie in science. It has the potential to probe student understanding in any area where objects/concepts/facts may be distinguished by yes/no questions.

A current problem with this program in LogoWriter is that the author was unable to find a means of saving the workspace, which is where the acquired knowledge of ELEMENT is stored. This means that the questions devised by the students would have either to be written down for the teacher to see, or teachers would have to run the program once the students had 'taught' it to distinguish between the selected elements.

[We have referred this one to Dr. Turtle; see elsewhere in this issue. Eds.]

TO ELEMENT
PR [THINK OF AN ELEMENT.]
PR [I SHALL TRY TO GUESS IT BY
ASKING QUESTIONS.]
CHOOSE.BRANCH :KNOWLEDGE
PR [LET'S TRY AGAIN . . .]
END

TO QUESTION :NODE
OP FIRST :NODE
END

TO YES.BRANCH :NODE
OP FIRST (BF :NODE)
END

TO NO.BRANCH :NODE
OP LAST :NODE
END

TO CHOOSE.BRANCH :NODE
IF (WORD? :NODE) [GUESS :NODE
STOP]
MAKE "RESPONSE ASK.YES.OR.NO
(QUESTION :NODE)
IF :RESPONSE = [YES]
[CHOOSE.BRANCH (YES.BRANCH
:NODE) STOP]
CHOOSE.BRANCH (NO.BRANCH :NODE)
END

TO ASK.YES.OR.NO :QUESTION
PR :QUESTION
MAKE "INPUT RL
IF :INPUT = [YES] [OP [YES]]
IF :INPUT = [NO] [OP [NO]]
PR [PLEASE TYPE "YES" OR "NO"]
OP ASK.YES.OR.NO :QUESTION
END

TO ADD.A.OR.AN :WORD
IF (FIRST :WORD) = [A E I O U] [OP SE
"AN :WORD]
OP SE "A :WORD
END

TO GUESS :ELEMENT
MAKE "FINAL.QUESTION (SE [IS IT]
(ADD.A.OR.AN :ELEMENT) [?])
MAKE "RESPONSE ASK.YES.OR.NO
:FINAL.QUESTION
IF :RESPONSE = [YES] [PR [LOOK HOW
SMART I AM!] STOP]
GET.SMARTER :ELEMENT
END

TO GET.SMARTER :WRONG.ANSWER
PR [OH WELL, I WAS WRONG. WHAT
WAS IT?]
MAKE "RIGHT.ANSWER (LAST RL)
PR [PLEASE TYPE IN A QUESTION
WHOSE ANSWER]
PR (SE [IS YES FOR] (ADD.A.OR.AN
:RIGHT.ANSWER) [AND])
PR (SE [NO FOR] (ADD.A.OR.AN
:WRONG.ANSWER))

```

MAKE "QUESTION RL
EXTEND KNOWLEDGE :QUESTION
:RIGHT.ANSWER :WRONG.ANSWER
END

```

```

TO EXTEND KNOWLEDGE
:NEW.QUESTION :YES.ANSWER
:NO.ANSWER
MAKE "KNOWLEDGE REPACE
:KNOWLEDGE :NO.ANSWER (LIST
:NEW.QUESTION :YES.ANSWER
:NO.ANSWER)
END

```

```

TO REPACE :TREE :NODE
:REPLACEMENT
IF :TREE = :NODE [OP
:REPLACEMENT]
IF WORD? :TREE [OP :TREE]
OP (LIST QUESTION :TREE REPACE
(YES.BRANCH :TREE) :NODE
:REPLACEMENT REPACE
(NO.BRANCH :TREE) :NODE
:REPLACEMENT)
END

```

Were it possible to save the workspace, it would be relatively simple to adapt this program into a guessing game that could be programmed by the teacher for use by students, by changing the wording of the print commands already in the program. For example, in the ELEMENT procedure PR [THINK OF AN ELEMENT] would be changed to PR [I WILL THINK OF AN ELEMENT].

ANALYSE

Although much simpler than ELEMENT, this program uses the same basic structure, a series of questions to which the answer can only be yes or no. The answer selected determines the direction the program takes. The program is given in the following code.

```

TO ANALYSE
CT
PR [WE ARE GOING TO TRY AND
IDENTIFY THE CHEMICAL YOU HAVE

```

```

BEEN TESTING]
PR [ ]
PR [HAVE YOU GOT YOUR RESULTS?]
PR [ ]
PR [PLEASE TYPE 'Y' FOR YES OR 'N'
FOR NO.]
PR [ ]
MAKE "INPUT ANSWER.YES.OR.NO
IF :INPUT = [YES] [PR [GOOD]]
IF :INPUT = [NO] [OP DUNCE]
QUESTION1
END

```

```

TO ANSWER.YES.OR.NO
IF READCHAR = "Y [OP [YES]]
OP [NO]
PR [PLEASE TYPE Y OR N]
OP ANSWER.YES.OR.NO
END

```

```

TO QUESTION1
PR [PLEASE TYPE 'Y' FOR YES OR 'N'
FOR NO TO THE FOLLOWING
QUESTIONS]
PR [ ]
PR [IS A WHITE PRECIPITATE
FORMED?]
IF READCHAR = "Y [OP QUESTION2]
OP QUESTION3
END

```

```

TO QUESTION2
PR [ ]
PR [IS THE PRECIPITATE SOLUBLE IN
EXCESS AQUEOUS AMMONIA
SOLUTION?]
IF READCHAR = "Y [PR [YOUR
CHEMICAL IS: ZINC] OP AGAIN]
OP QUESTION4
END

```

```

TO QUESTION3
PR [ ]
PR [IS A BLUE PRECIPITATE
FORMED?]
IF READCHAR = "Y [PR [YOUR
CHEMICAL IS: COPPER(II)] OP
AGAIN]
OP QUESTION5
END

```



```

TO QUESTION4
PR [ ]
PR [IS THE PRECIPITATE SOLUBLE IN
  EXCESS NAOH SOLUTION?]
IF READCHAR = "Y [OP QUESTION7]
OP QUESTION6
END

```

```

TO QUESTION5
PR [ ]
PR [IS A RED/BROWN PRECIPITATE
  FORMED?]
IF READCHAR = "Y [PR [YOUR
  CHEMICAL IS: IRON(III)] OP AGAIN]
PR [YOUR CHEMICAL IS: IRON(II)]
OP AGAIN
END

```

```

TO QUESTION6
PR [ ]
PR [DOES A SOLUTION OF THIS ION
  GIVE A BRICK RED FLAME
  COLOUR?]
IF READCHAR = "Y [PR [YOUR
  CHEMICAL IS: CALCIUM] OP AGAIN]
PR [YOUR CHEMICAL IS:
  MAGNESIUM]
OP AGAIN
END

```

```

TO QUESTION7
PR [ ]
PR [IS A WHITE PRECIPITATE
  FORMED WHEN SODIUM SULFATE
  SOLUTION IS ADDED?]
IF READCHAR = "Y [PR [YOUR
  CHEMICAL IS: LEAD(II)] OP AGAIN]
PR [YOUR CHEMICAL IS: ALUMINIUM]
OP AGAIN
END

```

```

TO DUNCE
PR [WELL GO AND GET THEM!
  PRESS <ANY KEY> WHEN YOU ARE
  READY TO CONTINUE]
IF READCHAR = "RETURN
OP QUESTION1
END

```

```

TO AGAIN
PR [ ]
PR [DO YOU WANT TO IDENTIFY
  ANOTHER CHEMICAL?]
PR [TYPE Y TO CONTINUE, N TO STOP]
PR [ ]
IF READCHAR = "Y [OP QUESTION1]
PR [OK! SEE YOU NEXT TIME]
END

```

To run the program, type ANALYSE in the command centre. The program has an automatic screenclear, so care must be taken not to try to run it while on the flipside.

This program aims to provide a data base that students can use in the laboratory as they are performing experiments. Qualitative analysis of this type is used primarily for teaching practical technique. Since it is not the aim to teach a knowledge base of chemicals, it is therefore acceptable for students to use a program such as this. If they are not performing the tests accurately this database will be of no use to them anyway.

The basis of the program is clearly expandable to encompass more chemicals, or adaptable to other science topics or indeed other subjects. Plant and animal classification could easily be translated into a program such as this.

Disadvantages of the current program are that its error catching mechanisms are very basic, and the program will not respond in any way to keys other than those specified. Given its potential for use in a classroom where students delight in testing out what different keys will do to a program's operation this is clearly a disadvantage.

Reference

Abelson, H. (1982) *Apple Logo*. Byte/McGraw-Hill.

How to Run a Sports Day...

Peter J Carter
Lockleys
South Australia

I once gave a Year 12 class the task of developing a spreadsheet to calculate the results for a Sports Day. They needed some data to test their work, so I wrote a few LogoWriter procedures to generate a set of 'results' for each student:

Procedures for Sports Day spreadsheet assignment

```
TO SPORTSDAY
CT PR "RESULTS FOR SPORTS DAYI
PR " PR "IU16 BOYS 400 MI
UAGE 1 :HOUSES
PR " PR TIME :B400
PR " PR "IU16 GIRLS 400 MI
UAGE 1 :HOUSES
PR " PR TIME :G400
PR " PR "IU16 BOYS SHOT PUTI
UAGE 1 :HOUSES
PR " PR TIME :BSHOT
PR " PR "IU16 GIRLS SHOT PUTI
UAGE 1 :HOUSES
PR " PR TIME :GSHOT
PR " PR "IOPEN BOYS 100 MI
OPEN 1
PR " PR TIME :B100
PR " PR "IOPEN GIRLS 100 MI
OPEN 1
PR " PR TIME :G100
PR " PR "IOPEN BOYS HIGH JUMPI
OPEN 1
PR " PR TIME :BJUMP
PR " PR "IOPEN GIRLS HIGH JUMPI
OPEN 1
PR " PR TIME :GJUMP
END
```

```
TO UAGE :PLACE :HOUSES
IF EMPTY? :HOUSES [STOP]
MAKE "POS 1 + RANDOM COUNT
:HOUSES
PR (SE :PLACE ITEM :POS :HOUSES
CHAR 32) CB CB
UAGE :PLACE + 1 REMOVE :POS
:HOUSES
END
```

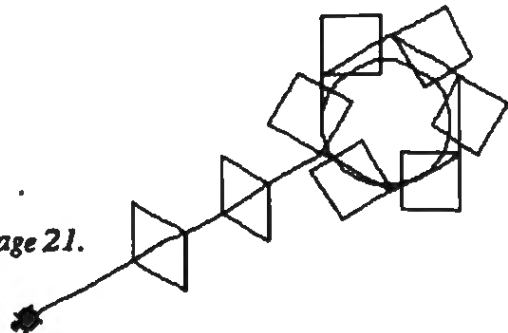
```
TO REMOVE :PLACE :LIST
IF EMPTY? :LIST [OP "]
IF 1 = :PLACE [OP BF :LIST]
OP SE FIRST :LIST
REMOVE :PLACE - 1 BF :LIST
END
```

```
TO OPEN :PLACE
IF :PLACE > 4 [STOP]
PR (SE :PLACE ITEM 1 + RANDOM 4
:HOUSES CHAR 32) CB CB
OPEN :PLACE + 1
END
```

```
TO TIME :LIST
OP ITEM 1 + RANDOM 5 :LIST
END
```

```
TO STARTUP
MAKE "HOUSES [BOOLE LOVELACE
TURING ZUSE]
MAKE "B400 [1:12 1:10 1:08 1:07 1:06]
MAKE "G400 [1:13 1:11 1:09 1:07 1:05]
MAKE "BSHOT [11.2 11.5 11.8 12.1
12.3]
MAKE "GSHOT [7.8 8.1 8.2 8.3 8.4]
MAKE "B100 [11.8 11.7 11.6 11.5 11.4]
MAKE "G100 [14.2 14.0 13.7 13.5 13.2]
MAKE "BJUMP [1.7 1.72 1.74 1.76 1.78]
MAKE "GJUMP [1.5 1.52 1.54 1.56 1.58]
END
```

Graphic produced with DOODLE procedures - see page 21.



Dear Dr. Turtle...

This section is for your Logo problems. Post your problems and Dr Turtle and a panel of "experts" will attempt to answer them.

*Dear Dr. Turtle,
Nicholas Derry, in his article about the ELEMENT program in the current issue, notes a problem as follows: "A current problem with this program in LogoWriter is that the author was unable to find a means of saving the workspace, which is where the acquired knowledge of ELEMENT is stored. This means that the questions devised by the students would have either to be written down for the teacher to see, or the teacher would have to run the program once the students had taught it to distinguish between the selected elements." This looks like one for you - what would you suggest? Eds.*

Well, at last, a genuine letter asking for some advice.

To capture some text and to be able to use it later can be done in a number of ways. One way is to create a set of tools for this purpose and then import them into your page with GETTOOLS "Pagename.

This way they are loaded "invisibly", on the Flip side and will be consistent. If you load more files than you need, they won't be seen anyway. (To see your invisible files, type SHOW TOOLLIST).

Here are a couple of tools to get you started. You might want to adjust them slightly to suit your particular needs

```
TO GET.WORDS :LIST
CT HT TAB
PR [TYPE A WORD ... or S to STOP]
MAKE "LIST SE :LIST READWORD
IFELSE EQUAL? LAST :LIST "S
  [NAME.LIST :LIST] [GET.WORDS
  :LIST]
END
```

```
TO READWORD
OP FIRST READLIST
END
```

```
TO NAME.LIST :LIST
CT
PR []
PR [HERE IS YOUR LIST....]
PR []
PR BUTLAST :LIST
PR []
PR [WHAT WOULD YOU LIKE TO CALL
YOUR LIST?]
PR []
NAME BL :LIST READWORD
END
```

To collect a string of words such as a sentence or an equation, you could use:

```
TO GET.SENTENCE :LIST
CT HT
PR [TYPE IN YOUR SENTENCE AND
PRESS ENTER WHEN DONE]
MAKE "LIST READLIST
PR []
PR [WHAT DO YOU WANT TO CALL
YOUR SENTENCE?]
PR []
PR []
NAME :LIST READWORD
END
```

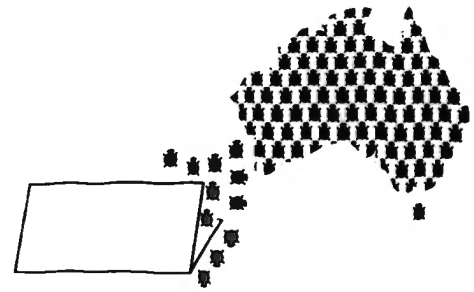
To use these tools, type GET.WORDS [] or GET.SENTENCE [] The empty brackets are necessary as input to allow the procedure to run. To use it in a procedure:

```
TO COLLECT.WORDS
CT HT
GET.WORDS [ ]
END
```

If you want to use your list somewhere else, you'll need to remember it or have the student remember it. If you want to name the list without asking the user for a name you could simply name it (say) X1 and store that name somewhere for later use.

Dr Turtle.

OzLogo LogoFile

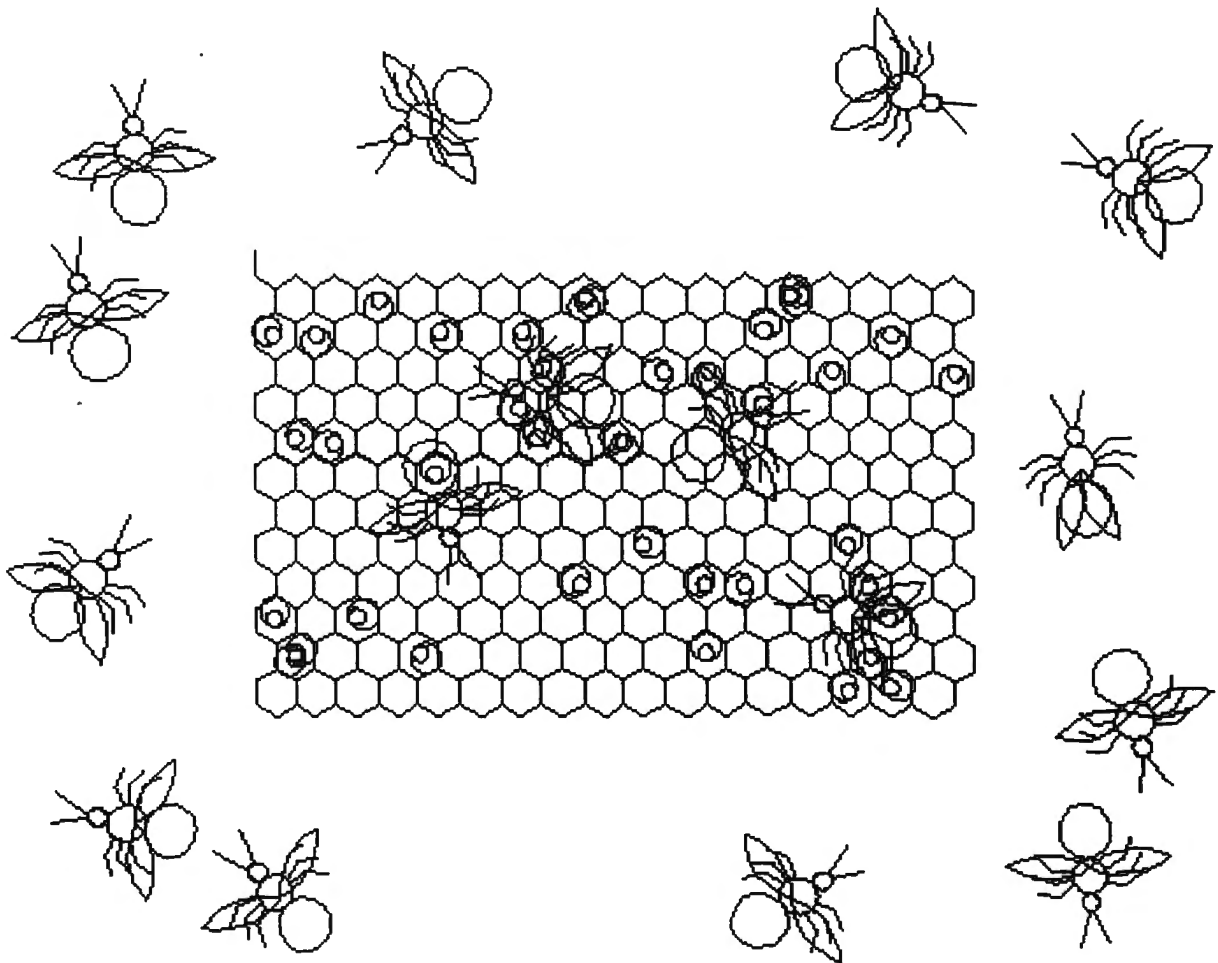


Magazine of OzLogo, a Logo Special Interest Group of the CEGV and MAV.

Vol. 2, Nos. 3 & 4

Inside ...

Review of "The Children's Machine"
Procedures for Drawing Glassware for Science
Teaching Computer Programming
Investigating Language with Logo
Drawing Flags
SPIDERQUIZ
Towns of France
Dear Dr Turtle



The Creative Tool Kit Every Student Needs

MicroWorlds



Designed by the people who brought you LogoWriter

Cross-Curricula: MicroWorlds Project Builder

- A learning environment that encourages students to develop their own problem-solving strategies while creating school projects.
- A Project book that takes you through the entire process of creating cross-curricula projects, step by step.
- Drawing tool (and 140 colours) for creating backgrounds for stories.
- Infinite number of turtles that can be set to different sizes and shapes, and colours.
- Word-processing features such as multiple text boxes, different fonts, sizes, styles, and colours, plus the ability to place text over graphics, or in any direction.
- Easy to add buttons, sliders and hot spots that help start and stop animation, music and other special effects, as well as create hypermedia links.

MicroWriter's Language Arts

- A Projects book full of writing projects that illustrate how language and art can be used together to communicate ideas and emotions. Projects include Haiku Visual Poetry, Advertising, Conquain, and more.

Both packages include

- Tools to write text in any shape, style, colour or size, or in any direction.
- Drawing and other visual effects' tools such as animation, scrolling or flashing text.
- A music and sound centre that makes it easy to set words to music.
- On-disk project starters, ready-made background scenes, and project samples.
- A new and comprehensive on-line Help System. Teachers or students can even create their own project-specific information balloons.

Package Contents:

- Program disks includes on-disk sample projects
- MicroWorlds Project Book
- Teacher's Resource
- How To Book

School's Pricing

Set \$130

6 User Lab Pack \$585

Unlimited Site Licence \$1595



System Requirements:

Macintosh colour computers (LC's colour Classics or better) 4Mb, System 7 or higher Hard Drive

Available from:



38 Hartnett Drive Seaford, Vic 3198
Ph: (03) 786 7177 Fax: (03) 785 3599

THREE EASY WAYS TO ORDER!

By Phone

Call Toll Free 008 337 055 or
(03) 786 7177 in the
Melbourne Metro area

By Fax

Fax your order to us on
(03) 785 3599

ORDERS

By Mail

Simply complete your order details and mail with your Purchase Order, credit card details or cheque to:

Reply Paid AAA 146,
P O Box 2053 Carrum Downs
Vic 3201



LogoFile is a magazine for interested users of Logo in education published by OzLogo.

OzLogo is a Special Interest Group of the Computing in Education Group of Victoria (CEGV) and the Mathematical Association of Victoria (MAV).

Editors:
Anne McDougall
Leon Guss

Assistant Editor:
Kirsty McDougall

Contributions to be forwarded to:

LogoFile Editor
OzLogo
Room 42
Statewide Resources Centre
217 Church Street
Richmond VIC 3121

Membership fees are:
\$20 per individual
\$30 per organisation.

Attach cheque made payable to OzLogo. Applications should be forwarded to:

Membership Secretary
OzLogo
Room 42
Statewide Resources Centre
217 Church Street
Richmond VIC 3121

Editorial

This double issue of LogoFile marks the end of our second year of publication. Our aim has been to provide readers with a wide variety of material, ranging from relatively scholarly articles on topics such as Logo philosophy, pedagogy and practice, to short informal notes, reports and reviews, as well as plenty of procedures to read, try out, alter and explore.

Responses we have received so far indicate that LogoFile is pleasing and useful to at least some Logo users out there. We are keen to meet your needs, so let us know your ideas for further articles and features to include.

We thank everyone who has contributed to Volumes 1 and 2 of LogoFile, and in particular our regular contributors, Jenny Betts, Peter Carter, Paul Dench, John Turner, David Williams, and of course Dr Turtle (David Rasmussen).

LogoFile Volume 3. No. 1 is being prepared now. The quality of the magazine depends upon readers and interested Logophiles sending in material to include in future issues. **AND WE ARE WAITING FOR A COVER GRAPHIC NOW!** Send us your contributions.

Anne McDougall
Leon Guss

Editors

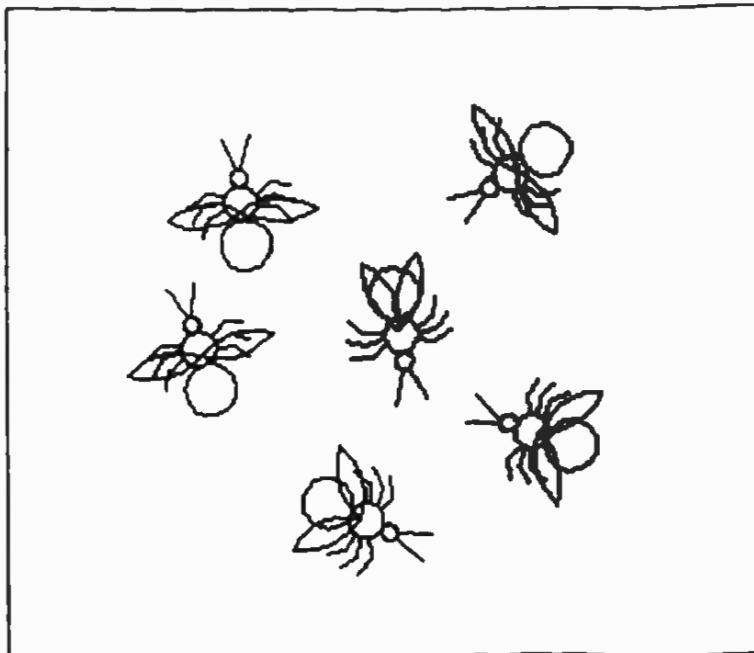
Contents

Editorial	3
Cover Graphic: BEE DANCE	4
The Children's Machine - Review	5
Drawing Glassware for Science	6
Teaching Computer Programming	7
Investigating Languages with Logo	13
Some Flags	14
SPIDERQUIZ: An interactive Game	18
Dear Dr Turtle	21
Software Development: Towns of France ...	22

**Cover Graphic:
BEE DANCE**

"Bee Dance", the graphic on the cover of this issue of LogoFile was submitted by Paul Dench. It was developed from the work of two Year 7 students from Melville Primary School in Western Australia. The procedures were written for a BBC microcomputer. All of the lengths in the original code have been scaled to change the size of the design so that it fits conveniently on the screen of a Macintosh PowerBook.

Bee Dance demonstrates sophisticated use of random variables. The grubs are randomly distributed as the cells are drawn. Note how the bees carry the honey message by their random choice of position, heading and wing angle.



```
TO BEEDANCE
HIVE
BEES
END
```

```
TO ANTENNAE
RT 180 LARC 2 0.6
RT 90 FD 14 BK 14 RT 90
RARC 4 0.6
LT 90 FD 14 BK 14 RT 90
RT 180 LARC 2 0.6
END
```

```
TO LARC :STEPS :SIZE
REPEAT :STEPS [FD
:SIZE LT 10]
END
```

```
TO RARC :STEPS :SIZE
REPEAT :STEPS [FD
:SIZE RT 10]
END
```

```
TO HIVE
WINDOW
HT
PU SETPOS [-136 -70]
SETH 60 PD
REPEAT 6 [ROWBACK]
END
```

```
TO ROWBACK
ROW FD 9 LT 180 ROW
RT 120 FD 9 LT 60 FD 9
RT 60 FD 9 RT 60
END
```

```
TO ROW
REPEAT 17 [CELL 9 LT
120]
END
```

```
TO CELL :SIZE
MAKE "EMPTY TRUE
REPEAT 8 [FD :SIZE RT
60 GRUB]
END
```

```
TO CIRCLE :SIZE
REPEAT 24 [FD :SIZE
RT 15]
END
```

```
TO GRUB
IF AND (EQUAL? 1
RANDOM 40) :EMPTY
[LT 20 RARC 18 1
CIRCLE 0.8 RARC 18 1
RT 20]
END
```

```
TO JUMP :X :Y :H
PU SETPOS SE :X :Y PD
SETH :H
END
```

```
TO BEES
HT
JUMP -60 -20 RANDOM
360 BEE
JUMP 60 30 RANDOM 360
BEE
JUMP -40 40 RANDOM
360 BEE
END
```

```
TO BEE
BODY
LEGS
ANTENNAE
WINGS
END
```

TO RLEG :SIZE
 RT 85 FD :SIZE RT 45 FD
 :SIZE BK :SIZE LT 45
 BK :SIZE LT 85
 END

TO LLEG :SIZE
 RT 85 FD :SIZE LT 45 FD
 :SIZE BK :SIZE RT 45
 BK :SIZE LT 85
 END

TO LEGS
 RARC 18 0.6
 LARC 7 1.2
 REPEAT 3 [LLEG 8 LARC
 2 1.2]
 LARC 13 1.2
 REPEAT 3 [RLEG 8 LARC
 2 1.2]
 LARC 4 1.2
 RARC 18 0.6
 END

TO BODY
 RARC 18 0.6
 LARC 18 1.2
 RARC 36 1.8
 LARC 18 1.2
 RARC 18 0.6
 END

TO WINGS
 PU RT 90 FD 13 RT 180
 FD 30 PD
 MAKE "ANGLE (20 +
 RANDOM 90)
 RT :ANGLE - 30
 WING 23/5
 LT 2 * :ANGLE
 WING 23/5
 END

TO WING :SIZE
 REPEAT 2 [RARC 7 :SIZE
 RT 110]
 END

Please send interesting
 graphics to the editors for
 future cover designs.

**Beyond "Mindstorms":
 A brief 'tour guide' to
 "The Children's
 Machine"**

**Carolyn Dowling
 Australian
 Catholic University
 Victoria**

**Papert, Seymour (1993),
 The Children's Machine:
 Rethinking School in the
 Age of the Computer,
 New York: Basic Books.
 ISBN 0-465-01830-0**

This book consolidates a number of aspects of Papert's experiences and thinking during the ten years or more since the publication of *Mindstorms*. A strongly polemical work, it addresses itself to a varied audience including learners of all ages, teachers, parents, and policy makers both in the educational and the broader political arena. Against the background of a perceived need for radical changes to take place in schooling as it is currently experienced by most learners, Papert focuses on the way in which the relationship between children and computers can affect learning, arguing that "... the powerful contribution of the new technologies in the enhancement of learning is the creation of personal media capable of supporting a wide range of intellectual styles" (Papert p. ix).

He himself describes the book as encompassing three main themes, firstly, what is currently taking place in schools, with emphases both on the role of the teacher and on strategies for change, secondly, his perspectives on how computing technology and its associated ideas and culture have evolved, and thirdly, his formulation of a new "theory of learning" (Papert p. 21).

In arguing the need for radical as opposed to piecemeal changes in our conceptions of teaching and learning, he contrasts the rigid and hierarchical nature both of current school systems and of the models of learning and knowledge which they endorse, with the values associated with 'epistemological plurals', that is, with the acceptance of a range of modes of thinking and knowing, particularly those which might be described as more personal and intuitive. These opposing attitudes are crystallised in his use of the terms 'Yearners' and 'Schoolers'. While the latter are so conditioned by the existing system that they seem unable to contemplate radical alternatives, the former constitute the 'resistance movement' who, if they do not move out of the system, undermine it from within by their adherence to an alter-

native set of values. As already mentioned, Papert sees computing technology as having the potential to support greater flexibility in thinking and learning styles. In this book he pays particular attention to the importance of a strong personal engagement with learning, and to the need to legitimise more 'concrete' forms of thinking and interaction through recognition of their intrinsic worth, as opposed to regarding them as a 'stage' to be worked through in the interests of achieving more abstract modes of thought. His substitution of 'letteracy' for 'literacy' reflects his concern regarding the institutional privileging of reading and writing as virtually the only acceptable routes to knowledge within our culture. His propositions concerning the need for both new practices and a new theory of learning include renewed advocacy of the use of the term, 'mathetics', originally suggested in *Mindstorms*, and the development of a new area of intellectual endeavour for children, 'cybernetics', through which they might be encouraged to understand and think about the world in ways less tied to traditional goals and forms of logic.

Readers who have been disconcerted by the apparent neglect of teachers in

Papert's earlier writings, particularly in *Mindstorms*, can be reassured that not only is the balance redressed in *The Children's Machine*, but a credible, characteristically personal explanation for this change of focus is made explicit.

Some curious features of this book which probably reflect both its polemical purposes and the broad nature of its intended audience include an emphasis in the Preface on the competitive strength bestowed by the ability to learn, and a strong endorsement in the concluding chapter of a decentralised model of education encompassing a range of highly individualised 'little' schools - a concept which may not sit comfortably with the ideologies of some proponents of a more uniform system of public education.

The Children's Machine is far too rich and complex a book to be summed up in a few paragraphs. In every chapter a multiplicity of observances, issues, arguments and examples create resonances both with different facets of Papert's own work over the years, and with a range of other contemporary writers and thinkers. The 'style' of the book reflects its content. Material is presented variously and often 'playfully', through formal argument,

and through a quite delightful exploratory 'tinkering' with language. Through all this, Papert's personal passion for learning shines through. *The Children's Machine* is profoundly serious fun - enjoy it!

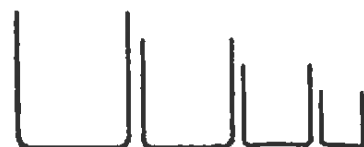
Some LogoWriter
Procedures for Drawing
Glassware for Science
Experiments

Nicholas Derry
Carey Grammar School
Victoria

```
TO ARCR :SIZE
REPEAT 10 [FD :SIZE *
0.00174 RT 1]
END
```

```
TO ARCL :SIZE
REPEAT 10 [FD :SIZE *
0.00174 LT 1]
END
```

```
TO BEAKER :SIZE
LT 90
REPEAT 9 [ARCR :SIZE]
FD :SIZE
PU RT 90 FD :SIZE * 0.9
RT 90 PD
FD :SIZE
REPEAT 9 [ARCR :SIZE]
FD :SIZE * 0.7
SETH 0
END
```



```

TO CONICALFLASK :SIZE
LT 90
REPEAT 12 [ARCR :SIZE]
FD :SIZE
REPEAT 3 [ARCL :SIZE]
FD :SIZE * 0.5
PU RT 90 FD :SIZE/3 RT
90 PD
FD :SIZE * 0.5
REPEAT 3 [ARCL :SIZE]
FD :SIZE
REPEAT 12 [ARCR :SIZE]
FD :SIZE * 1.25
SETH 0
END

```



```

TO FILTERFUNNEL :SIZE
FD :SIZE
LT 40 FD :SIZE
PU RT 130 FD :SIZE * 1.5
RT 130 PD
FD :SIZE
LT 40 FD :SIZE
SETH 0
END

```



```

TO TESTTUBE :SIZE
FD :SIZE
PU
RT 90 FD :SIZE/5 RT 90
PD
FD :SIZE
REPEAT 180 [FD :SIZE *
0.00174 RT 1]
END

```

```

TO ROUNDBOTTOMFLASK
:SIZE
LT 90
REPEAT 140 [FD :SIZE *
0.0174 RT 1]
REPEAT 50 [FD :SIZE *
0.0174 LT 1]
FD :SIZE
PU RT 90 FD :SIZE * 0.8
RT 90 PD
FD :SIZE
REPEAT 50 [FD :SIZE *
0.0174 LT 1]
REPEAT 140 [FD :SIZE *
0.0174 RT 1]
FD :SIZE * 0.25
SETH 0
END

```



Teaching Computer Programming

Dale Lauton
Victoria

Introduction

Kurland sums up the situation many computer programming teachers find themselves in as follows:

We know far too little about what to expect students will learn in a reasonable period of time, how they learn, what conceptual difficulties they encounter, what forms of cognitive support they may require to guide them over these difficulties, or whether individual differences in learning styles are reflected

in programming and need to be guided differently in instruction. (Kurland, 1986, p.240)

What then constitutes effective teaching of programming? To answer this, teachers will have to engage in investigative practices themselves. I shall highlight some approaches that could be useful; although they may not be applicable to every student, they do contain ideas on how to teach programming. I shall focus, in some depth, on how programming can be introduced to primary and secondary students. The introduction to computer programming will essentially involve non-structured programming. Later I shall focus on how to teach programming to students who may be interested in pursuing careers in computer programming. This will involve a more structured type of programming. Although I have distinguished between the two groups (structured and non-structured programming), I hasten to add that there is certainly an overlap of ideas and teachers should not hesitate to select ideas from any group to improve their teaching approach.

Introducing Programming

Van Merriënboer identified three prevalent instructional approaches to teaching introductory programming courses, two of which, the Expert approach and the Reading approach, emphasise

that students start out with fairly complex code which they are to read, understand or interpret. They look at the syntax rules only after having studied the program logic and flow of control. The third strategy, the Spiral approach, presents syntactic and semantic knowledge simultaneously in incremental steps, and therefore programming skills are not utilised in this method until quite late in the course (Van Marrienboer, 1987, p. 253-58).

Whichever approach is used, teachers should bear in mind that students come to programming with different motivation, backgrounds, interests, and biases. Programming can and should be taught with this variety in mind. The subsequent suggestions are based on the premise that the teacher will teach the concepts in an order that makes the most sense to beginners, and that is most intrinsically interesting to beginners. The goal is not to weed out those who are not destined to become computer scientists; it is to involve and educate as best we can the greatest number and variety of students.

Start With Graphics

Many textbooks start with RAM, ROM and CPU and maybe even with binary arithmetic, and then (in BASIC, for example), PRINT, LET, DATA, READ, with lots of tax and payroll examples. Students wonder what

this has to do with anything they can use or do. Much, much later the textbook gets to graphics, inputs, branching and string manipulations, with which students can do more interesting things. This type of textbook could easily influence a teacher, not trained in ways to teach programming, to adopt a similar approach in introducing programming to students.

The inventors of Logo designed the language to have a graphics entry point, "a motivating and understandable way to learn some powerful programming ideas" (Papert, 1980). If children are encouraged to design their own graphics projects within a framework laid down by the teacher, they will probably spend more time perfecting it than they would on other assignments. Graphics is an excellent arena in which to introduce programming concepts such as variables, loops and branching.

Give Some Time To Music

Far too many programming texts are devoted almost entirely to calculations using numeric variables. According to McGrath (1990), "this is a surefire way to turn off many girls, minorities, and humanists. (That's a large group to lose all at once!)" There is no reason that variables, loops, procedures or recursion can only be taught through using mathematical concepts. These programming ideas are equally important in other

media as well, and the use of concepts in more than one medium should give a boost to the students' understanding (Dickson, 1985). For the majority of children and teenagers, music plays an important role in their culture and it is for this reason I have focussed on how to use music to introduce programming concepts.

The tune for the Mexican Hat Dance comprises the notes C F C F C F C F E F G E E E E E C E F E D F C. The command

TONE <frequency> <duration>
produces a musical note in the frequency specified by the first parameter and for the duration specified by the second parameter. The command can be demonstrated by varying the parameters and letting students discover the differences in sound with different parameter values. The teacher can give the students a table with the frequencies of the musical notes A, B, C,... etc. and then challenge the students to play the Mexican Hat Dance by executing successive tone commands. Very soon they realise that they are repeating a lot of work. For example, the note E is used ten times. The teacher can then introduce the concept of a procedure and a procedure can be written for every note.

eg.
TO E
TONE 262 10
END

Because all the durations are the same, the song will sound very robot-like. Students will quickly realise that they must alter the notes' duration. They will now be ready for the teacher to introduce the concept of a variable.

```
TO E :DUR
TONE 262 :DUR
END
```

The Mexican Hat Dance now becomes a sequence of calls to procedures playing the notes C 5 F 10 C 5 F 10 C 5 F 15 etc. Soon the students will realise that to play the piece again, it is necessary to write the calls all over again. A solution to this problem can turn out to be to write calls inside a procedure say, MEXICANHATDANCE. To play the piece again simply type in MEXICANHATDANCE.

As more elaborate pieces of music are attempted, students will realise that it is quite laborious to write a procedure for every note appearing in the tune. The teacher can introduce the concept of a list of numbers as well as the functions FIRST and BUTFIRST. The students can do several exercises to make sure that they understand these three new concepts. Next, the teacher can introduce recursion but let the procedures that use it end with an error comment, i.e. when the list is exhausted. The MAKE command could also be introduced at this time. The students are now

ready to play a tune with only one procedure and two lists, one with frequencies (FREQ) and the other with durations (DUR).

```
eg.
TO PLAY :FREQ :DUR
TONE FIRST :FREQ
FIRST :DUR
PLAY BUTFIRST :FREQ
BUTFIRST :DUR
END
```

The execution ends with an error because of the empty list. In order to avoid the error the teacher can introduce the IF command. The new version is listed below:

```
TO PLAY :FREQ :DUR
IF :FREQ = [ ] [STOP]
TONE FIRST :FREQ
FIRST :DUR
PLAY BUTFIRST :FREQ
BUTFIRST :DUR
END
```

So one can go on teaching various computer programming concepts through music.

Accept Both Top-Down and Bottom-Up Programming

One factor that often prevents teachers from allowing bottom-up programming, even among beginners, is perceived pressure from the university. University professors often claim that pre-university teaching of programming is so sloppy that it makes teaching university computer science more difficult. In fact, beginning college computer science students who have taken previous comput-

ing courses tend to do better than those who have not (McGee et al., 1987). Furthermore, almost all college programming students start with an introductory computer science course, even if they've already had programming in high school. So unless teachers are teaching a special Advanced Placement course, it is not their job to prepare them for a beginning course in computer science, one they won't even take unless they understand and enjoy the course.

Let Students Work Together On Projects

Programming is a very social process in the professional setting, with programmers working in teams to complete systems of programs. In a school atmosphere, a teacher's concern with individual accountability and evaluation often overshadows this social aspect. MacGregor (1988) found out that students spent more time planning their programs when their work was to be critiqued by their fellow students. Research by Webb (1986) revealed that students were more successful with their own program generation if they participated in group discussion during the planning phase.

Girls, especially, have a difficult time getting involved with computers because it is such a solitary activity. When it becomes a social activity, girls enjoy it more. What this points to is that teachers

should occasionally allow group planning of projects.

Debugging

In the early stage Harvey (1991) believes that debugging should not be a large part of a student's experience. It is true that debugging is an excellent mental exercise, but real beginners tend to make uninteresting mistakes. Spending 15 minutes struggling with an unnoticed punctuation error could result in frustration, so sometimes it is better to debug a beginner's program and encourage the student to get on with the big idea that the bug interrupted. Only if the bug seems to indicate a serious conceptual misunderstanding, then only should a teacher perhaps explain the conceptual error to the student or class.

Use Motivating, Interactive Assignments Like Games

Allow students to program their own version of a game, like Guess My Number, Hangperson, or MasterMind (some are tougher than others). It is sometimes better to allow students to begin working on assignments on the computer and when they encounter problems, the teacher will have a good reason to show them that sometimes (especially with more complicated programs), it is better to plan on paper in advance. This leads us on to structured programming.

Structured Programming

So what is structured programming? It is defined as a systematic way of designing, building, validating and documenting computer programs which leads to the production of efficient and reliable software. I would recommend that this type of programming be pursued by those interested in following computer programming careers.

Ingram (1988) designed an instructional model for teaching structured programming (top-down design). In top-down programming, a problem is divided into major subtasks. Each subtask is further divided into smaller subtasks and each successive subdivision is known as stepwise refinement. Research by Ingram showed students were more successful when they followed a four step plan:

- describing the problem.
- dividing the problem into modules.
- writing algorithms for each module.
- translating the algorithm into code.

Ingram reported that students found difficulty in breaking the problem into smaller modules. This was because the practice examples were too small or trivial to be easily subdivided. Teachers should be aware of this and examples should lend themselves to easy subdivision.

Writing an Algorithm for the Program/Module

Students have to understand algorithms and to develop algorithms by themselves. Algorithms are "abstract representations of program logic, and many students are not capable of thinking at the level of abstraction required". (Oliver, 1992). So how is it possible to teach students how to design algorithms?

Typical Problem:

A company pays its employees an allowance for private use of their motor vehicles. Employees are paid 35c per km travelled if their car engine size is less than 2000cc and 45c per km if their car has an engine of size 2000cc or more. Plan a program to calculate an employee's allowance for car use.

It is not difficult to imagine the discussion that would accompany the teaching of this algorithm in a typical classroom. Some teachers would discuss the inputs required, then the processing, and then finally the output. At the end of the lesson, students would be expected to describe the steps and to trace the algorithm with specific data, for example, 500km travelled in a 2500cc car.

Does this form of instruction lead students to an understanding of the algorithm? Oliver (1992), from evidence of observing teachers discussing algorithms, suggests that

Question	Algorithmic Step
1. What is the required endpoint?	Allowance
2. How is allowance calculated?	$\text{Allowance} = \text{Rate} * \text{Kilometres}$
3. How is rate determined?	When Size < 2000, rate = 0.35 otherwise rate = 0.45
4. What information is needed?	Read kilometres, Size

Figure 1

this instruction is frequently not effective.

The alternative method involves showing the algorithm to students in reverse fashion, with the development starting from the final step in the solution and proceeding back to the starting position. As each new step is determined, the focus turns to those which precede it. At all times there is focus on unravelling the solution and a direction to follow. The method is a form of top-down design that goes from the bottom-up. To illustrate this method, I will look at the preceding example.

We can start the algorithm by discussing the required output of the program, the ALLOWANCE to be paid. The student writes down Allowance. All that remains is how the Allowance might be calculated.

The ALLOWANCE is calculated by multiplying the kilometres travelled by the appropriate rate,
 $\text{ALLOWANCE} = \text{KILOMETRES} * \text{RATE}$.
 At this stage in the development of the algorithm, we need to include steps that will

provide values for kilometres and rate. Kilometres is a value that can be entered directly while rate is determined by the size of the engine. Guided by this information, we can determine the remainder of the algorithm. See Figure 1 (above).

When describing an algorithm, it is possible to gauge the level of understanding achieved by students by having them suggest the next step. In the research project, it was noticeable that with conventional teaching only the more able students in the class would participate in discussions. When this alternative approach to algorithm development was used, the number of students who participated in the discussion rose considerably. An unexpected outcome of this teaching strategy was the help it provided students in creating their own algorithms.

Once the student had the algorithm planned, it was then time to write the code. Linn found that feedback had a direct effect on student performance. Feedback consisted of teachers writing comments on completed assignments; returning assign-

ments promptly; explaining to small groups or individuals how to improve programs; providing a solution to the assignments; and describing different ways in which the assignment could have been solved.

Research by Ross (1989) revealed that low ability students benefited from unlimited access time, while medium and high ability students performed better in a restricted access environment.

Debugging

Before a program could be considered complete, all the syntax and logical errors had to be removed in the debugging process. Pascal students spent as much as 47% of their lab time on debugging related activities (Pintrich, 1987)

Research by Linn revealed that experienced programmers could identify bugs in a matter of seconds just from looking at the output. For example, if a numerical value was grossly inaccurate, the most likely problem would be an uninitialised variable; if the output was close to being correct, a loop process could have been off by one. If

teachers could anticipate students' errors, debugging time could be cut down significantly. To clear up misconceptions and encourage debugging skills, teachers should require students to read programs and predict the output, and closely track the changing status of variables during execution.

New programming environments may have changed the cognitive skills required for debugging. The new editors have good trace routines, windows displaying both the code and the corresponding output simultaneously on the screen, and step-by-step execution that displays the changing values of the variables as the program executes.

Samurcay (1985) studied students' use of their existing mathematical models in understanding programming concepts. In many cases, the model is acceptable, but there were differences of which the student needed to be made aware explicitly. In mathematics, for example, the equal symbol meant equality, but as part of a programming assignment statement, it displays an asymmetric relationship. A variable has a unique value in maths that remains static throughout the problem, while in programming, a variable often has a dynamic changing value. In the accumulation process, $n = n + 1$, for example, the same variable designated both a previous and present value in programming.

Conclusion

I have highlighted some of the approaches that could be used in teaching programming both to introductory and advanced courses. Because teaching is personality based, different teachers will adopt different approaches, but the main goal always should be to facilitate the students' learning and understanding. This will entail responding to individual students and small groups in such a way as to build on their existing knowledge base and enable them to extend their ideas by using the appropriate programming techniques.

References

- Dickson, W. P. (1985) Thought-provoking Software: Juxtaposing Symbol Systems, *Educational Researcher*. 14 (5), 30-38.
- Harvey, B. (1991) Symbolic Programming vs. the A. P. Curriculum, *The Computing Teacher*. Feb, 27-30.
- Ingram, A. L. (1988) Instructional Design for Heuristic-Based Problem Solving, *Educational Communication and Technology Journal*. 36 (4), 211-30.
- Kurland, D. et al. (1986) A study of the Development of Programming Ability and Thinking Skills in High School students, *Journal of Educational Computing Research*. 2 (4), 429-458.
- Linn, M. C. (1988) Autonomous Classroom Environments for Learning, *Progress Report and Annotated Bibliography*. May, 1-13.
- MacGregor, S. (1988) The Structured Walk-Through, *The Computing Teacher*. 15 (9), 7-10. —
- McGee, L. et al. (1987) The Influence of Basic on Performance in Introductory Computer Science Courses Using Pascal. *SIGCSE Bulletin*. 19 (3), 29-37.
- McGrath, D. (1990) Eight Ways To Get Beginners Involved in Programming, *The Computing Teacher*. Sept, 19-20.
- Oliver, R. (1992) A Backward Approach to teaching Algorithms, *The Computing Teacher*. Nov, 17-18.
- Papert, S. (1980) *Mindstorms: Children, Computers and Powerful Ideas*. New York: Basic Books.
- Pintrich, P. R. (1987) Students' Programming Behaviour in a Pascal Course, *Journal of Research in Science Teaching*. 24 (5), 451-66.
- Ross, M. S. (1989) Computer Access and Flowcharting as Variables in Learning Computer Programming, *Report: Annual meeting of the association for Educational Communications and Technology*. Feb, 1-10.

Samurcay, R. (1985) *Learning Programming: An Analysis of Looping Strategies Used by Beginning Students, For the Learning of Mathematics*. 5 (1), 37-43.

Van Merriënboer, J. G. (1987) *Instructional Strategies and Tactics for the Design of Introductory Computer Programming Courses in High School, Instructional Science*. 16 (3), 251-85.

Webb, N. M. et al. (1986) *Problem Solving Strategies and Group Processes in Small Groups Learning Computer Programming, American Educational Research Journal*. 23 (2), 243-61.

An Investigation into Language with Logo

Debora Goldman
Mount Scopus
Memorial College
Victoria

During the July Logo conference one of the workshops that raised a number of questions about learning was the one run by Paul Goldenberg. These questions were directed towards the way in which rules for the formation of language are learnt by children. My last contact with language rules was of the form of skill and drill, when students were presented with a list of words to learn.

Paul challenged us to play, develop and explore for

ourselves some basic language rules. Initially he took us through neat sequential developmental steps that showed how the topic could be introduced with any class, even those not fully Logo literate.

The following program was developed during this workshop. Its aim was to attach prefixes to words to negate the word, eg. "literate" to "illiterate". After running this program through there are obvious words that do not fit the rules. The program tested whether there was a need to "double" the first letter, for words beginning with "r" and "l", and for the attachment of "im" and "in" to others. The aim is for the student to test as many words as required until an appreciation for the rules and its exceptions are understood.

The program that starts the procedures is REDO. It runs the program 10 times and prints out the result from CHECK using the results from FBB and LBB as the variables for CHECK. The CHECK procedure tests whether the last letter of the prefix matches the first letter of the word. If both are either an "l" or an "r" the appropriate prefix is attached (double the letter). Further tests attach the prefix "im" to words beginning with "p", and the "ir" prefix to words beginning with "s" and "c".

```
TO REDO
REPEAT 10 [PRINT
CHECK FBB LBB]
END
```

```
TO FBB
OP PICK [IL IM IR IN]
END
```

```
TO LBB
OP PICK [LOGICAL
POSSIBLE PLAUSIBLE
RELEVANT
COMPETENT SECURE
REGULAR
RESPONSIBLE
RATIONAL LOVABLE
LIKEABLE LITERATE
PARTICULAR PARTIAL
PLANNED POPULAR
SENSITIVE SATIABLE
SATURATED CAPABLE
COOPERATIVE]
END
```

```
TO PICK :OBJ
OP ITEM (1+RANDOM
COUNT :OBJ) :OBJ
END
```

```
TO CHECK :PRE :END
IF AND ("r= FIRST :END)
("r= LAST :PRE) [OP
(WORD :PRE :END)]
IF AND ("l= FIRST :END)
("l= LAST :PRE) [OP
(WORD :PRE :END)]
IF AND ("s= FIRST :END)
("n= LAST :PRE) [OP
(WORD :PRE :END)]
IF AND ("c= FIRST :END)
("n= LAST :PRE) [OP
(WORD :PRE :END)]
CHECK FBB LBB
END
```


A run delivers ten words similar to the following sample run:

IRRATIONAL
INCAPABLE
ILLITERATE
INCOMPETENT
ILLOVABLE
IRREGULAR
ILLOGICAL
INCOOPERATIVE
INSATIABLE
IRRESPONSIBLE

The only problem is that sometimes there may be a word or two that repeats on screen. This "bug" has yet to be corrected. Perhaps you can finish off this program and send in your solution. Hopefully this will be of use to you in some format. If this seems too difficult or you would like a different slant I suggest getting hold of Paul Goldenberg's book: E.P. Goldenberg and W. Feurzeig, *Exploring Language with Logo*, The MIT Press, Cambridge, MA, 1987.

Some Flags

Peter J Carter
Lockleys, South Australia

For ten years I've been using flags as Logo programming exercises. Although many flags use a few simple shapes, there is often a lot of thought needed to find the best way to draw those shapes, even the order in which to draw them.

I used to insist that flags fill the entire Logo screen, but the Mac LogoWriter screen is too long and narrow for many flags, so I use the correct flag proportions.

Most encyclopedias have a section on flags. The Flag Society of Australia (POB 142, Collins Street, Melbourne, Victoria 3000) publishes a chart, *Flags of Non-Independent Peoples*, (\$13, I think) with many flags you won't find elsewhere. I used it as the reference

for the Lapland flag.

This is obviously not the place to debate a new flag for Australia, but here's one way to experiment with a few ideas. You might try stamping Turtles instead of drawing stars for an Australian Logo flag.

AusFlag 91, the Latest in AusFlag's Series of Possible New Australian Flags.

The odd way of drawing the stars in the field colour and then FILLing came about because at the time I wrote this I didn't understand how colour 0 behaved. These days I'd use colour 249.

```
TO AUSFLAG91
FIELD
CRUX
PU HOME PD FILL
END
```

```
TO STAR7 :SIZE
REPEAT 7 [FD :SIZE RT 141.4 FD :SIZE
LT 90]
END
```

```
TO STAR5 :SIZE
REPEAT 5 [FD :SIZE RT 144 FD :SIZE
LT 72]
END
```



```

TO FIELD
CG HT SETC 4
PU SETPOS [-248 -110] PD
REPEAT 2 [FD 220 RT 90 FD 302 RT 90]
END

```

```

TO CRUX
PU SETPOS [-107 -70] SETH 19 PD
STAR7 10
PU SETPOS [-107 90] PD
STAR7 10
PU SETPOS [-167 10] PD
STAR7 10
PU SETPOS [-47 35] PD
STAR7 10
PU SETPOS [-70 -10] SETH 90 PD
STAR5 8
END

```

Christmas Flag

A Christmas flag designed by Tony Burton, of FlagGraphics in Sydney for Christmas 1991, and described in Crux Australis, Vol 7/4, No 32, December 1991

```

TO CHRISTMAS
FIELD
STAR 90
FILLSTAR 90
END

```

```

TO FIELD
HT
PU SETC 236 PD FILL
SETC 25
END

```

```

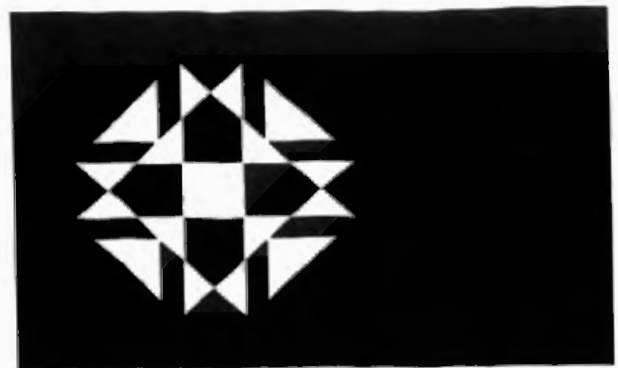
TO STAR :SIZE
PU SETPOS [-40 30] SETH 315 PD
REPEAT 4 [FD :SIZE * .7071 LT 135
FD :SIZE LT 90
FD :SIZE LT 135
FD :SIZE * .7071 RT 45
FD :SIZE / 2.47 RT 135
FD :SIZE * 1.4142 / 2.47
RT 135 FD SIZE / 2.47
PU FD :SIZE * 1.4142 / 2.2
LT 45 PD]
END

```

```

TO FILLSTAR :SIZE
PU
REPEAT 4 [RT 90 FD 5 PD FILL
PU BK 10 PD FILL
PU FD 5 LT 135
FD :SIZE * 1.4142 / 2.2
LT 45]
PU FD :SIZE / 2.1875
REPEAT 4 [LT 45
BK 5 PD FILL
PU FD 10 PD FILL
PU BK 5 LT 45
FD :SIZE / 1.0932]
PU LT 135 FD :SIZE * .707
PD FILL
END

```



The Flag of Lapland (Sapmi)

Developed from the Flags of Non-Independent Peoples chart by Clive Jackson.

```

TO LAPLAND
RG HT SETBG 8
BLOCKS
CIRCLE
END

```

```

TO BLOCKS
PU SETPOS [-165 -110] PD SETC 5
PD BLOCK 220 94
PU SETPOS [-71 -110] PD SETC 234
PD BLOCK 220 27
PU SETPOS [-44 -110] PD SETC 15
PD BLOCK 220 27
PU SETPOS [-17 -110] PD SETC 4
PD BLOCK 220 172
END

```

TO CIRCLE
 PU SETPOS [-44 57] SETH 90 PD
 SETC 5 ARCR 57 180
 SETC 4 ARCR 57 180
 PU SETPOS [-44 62] SETH 90 PD
 SETC 5 ARCR 62 180
 SETC 4 ARCR 62 180
 PU SETPOS [-57 58] PD FILL
 PU SETPOS [-104 0] PD FILL
 PU SETPOS [-57 -58] PD FILL
 SETC 5
 PU SETPOS [-31 58] PD FILL
 PU SETPOS [14 0] PD FILL
 PU SETPOS [-31 -58] PD FILL
 END

TO BLOCK :LENGTH :WIDTH
 REPEAT 2 [FD :LENGTH RT 90 FD
 :WIDTH RT 90]
 PU RT 45 FD 5 PD FILL
 PU BK 5 LT 45
 END

TO ARCR :RADIUS :ANGLE
 LOCAL [STEP REM]
 MAKE "STEP 2 * :RADIUS * PI / 36
 MAKE "REM REMAINDER :ANGLE 10
 REPEAT :ANGLE / 10 [RT 5 FD :STEP
 RT 5]
 IF :REM > 0 [FD :STEP * :REM / 10 RT
 :REM]
 END



The Namibian Flag

TO NAMIBIA
 OUTLINE
 RED
 GREEN
 BLUE
 SUN
 END

TO OUTLINE
 CG HT SETC 1
 PU SETPOS [-159 -95] PD
 REPEAT 2 [FD 190 RT 90 FD 320 RT 90]
 END

TO RED
 SETC 5
 PU SETPOS [-159 -95] PD
 SETPOS [-159 -77]
 SETPOS [112 95]
 SETPOS [160 95]
 SETPOS [160 77]
 SETPOS [-112 -95]
 SETPOS [-159 -95]
 PU HOME PD FILL
 END

TO GREEN
 SETC 235
 PU SETPOS [-98 -95] PD
 SETPOS [160 70]
 SETPOS [160 -95]
 SETPOS [-98 -95]
 PU SETPOS [100 0]
 PD FILL
 END

TO BLUE
 SETC 212
 PU SETPOS [98 95] PD
 SETPOS [-159 -70]
 SETPOS [-159 95]
 SETPOS [98 95]
 PU SETPOS [-100 60]
 PD FILL
 END

```

TO SUN
SETC 25 SETH 15
REPEAT 12 [POINT]
PU SETPOS [-110 40] SETH 0
PD CIRCLER 15
PU SETPOS [-95 40]
PD FILL
END

```

```

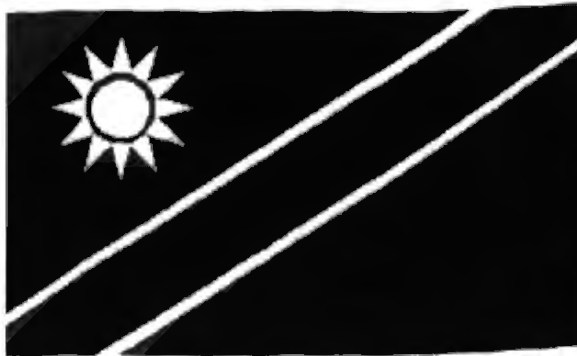
TO POINT
FD 20 RT 150
FD 20 RT 105
FD 10.4 BK 10.4
RT 60 PU FD 4 PD FILL
BK 4 RT 75
END

```

```

TO CIRCLER :RADIUS
LOCAL "STEP
MAKE "STEP 2 * :RADIUS * PI / 36
REPEAT 36 [RT 5 FD :STEP RT 5]
END

```



The Flag of the Northern Territory

```

TO NT
SETFIELD
CRUX
ROSE
END

```

```

TO SETFIELD
RG HT
SETBG 0
SETC 12
PU SETPOS [-53 -95] PD
REPEAT 2 [FD 190 RT 90 FD 213 RT 90]
PU SETPOS [120 0] PD FILL
END

```

The stars on this Southern Cross are different from those on the Australian flag: 8, 7, 6 and 5 points. (The new National Australia Bank Flags of the Nations brochure has printed it wrongly.)

```

TO CRUX
SETC 1
PU SETPOS [-106 -50] SETH 22.5 PD
STAR 6 8
PU SETPOS [-132 0] SETH 22.5 PD
STAR 6 7
PU SETPOS [-106 50] SETH 22.5 PD
STAR 6 7
PU SETPOS [-79 10] SETH 22.5 PD
STAR 6 6
PU SETPOS [-93 -10] SETH 22.5 PD
STAR 6 5
END

```

```

TO STAR :SIZE :POINTS
REPEAT :POINTS [FD :SIZE RT 135 FD
:SIZE LT 135 - (360 / :POINTS)]
PU RT 90 FD :SIZE / 2 PD FILL
END

```

The flower is a Sturt's Desert Rose. Two FILLs are needed in the petals because the outline crosses the boundary of the centre circle.

```

TO ROSE
SETC 0
PU SETPOS [28 0] SETH 0 PD
ARCR 25 360
PU SETPOS [53 0] PD FILL
SETC 1
REPEAT 7 [PETAL RT 360 / 7]
END

```

```

TO PETAL
PU FD 18 LT 90 PD
ARCR 8 35
ARCR 20 60
ARCR 5 60
ARCR 24 50
ARCR 5 60
ARCR 20 60
ARCR 8 35
PU RT 90 FD 5 PD FILL

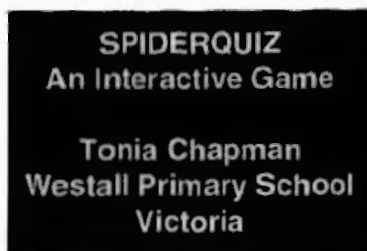
```



```
PU FD 5 PD FILL
PU BK 28
END
```

ARCR is taken from the tools page:

```
TO ARCR :RADIUS
:DEGREES
MAKE "STEPS (2 *
:RADIUS * 3.1416 / 36)
MAKE "REM REMAINDER
:DEGREES 10
REPEAT :DEGREES / 10
[RT 5 FD :STEPS RT 5]
IF :REM > 0 [FD :STEPS *
:REM / 10 RT :REM]
END
```



SPIDERQUIZ was designed to accompany part of a Grade 3 & 4 ESL Science/Language program. It is linked with a Grade 4 unit of study "Creepy Crawlies", part of a "Scientists in Schools" program. Some of the children had a particular fascination for spiders, and there had been a few residential spiders in the classroom (including the odd plastic one landing on the teacher).

The game is interactive in nature, as long as the children

playing can read simple English. The graphics appear in relation to the subject matter according to different questions in the quiz. The format of the game is simple, and can be used as long as the people using it attempt to key in answers and return. The quiz allows for incorrect answers and gives opportunities for further attempts. It gives opportunities to change each question slightly, allowing sometimes for a choice of two answers.

The opening titles use STAMP to announce the game. Four animated spiders in the CLIMB procedure walk in a circle before they branch off to the four corners of the screen. The spiders are made from three different shapes drawn in the shapes editor.

FLY and CATERPILLAR are similar procedures, using multiple shapes and some animation.

GROWTRIANGLE makes a randomly multicoloured spiderweb. SPIWEB includes two spiders spinning the web, and each triangle segment is drawn in a different colour. Random colour was used here, and the on screen result has a shimmering, semi-transparent quality, from the overlapping triangles.

SPIDERQUIZ can be modified and adapted, simplified or extended. Questions can be added or answers changed within the lists. It is a useful

teaching tool to see how well children are reading and using knowledge gained in class. It also provides a model of LogoWriter programming that the children could aspire to as their skills in procedure writing develop.

```
TO SPIDERQUIZ
TITLE
TITLE2
SETUP
CLIMBS
QUIZ1
QUIZ2
FLY
CATERPILLAR
QUIZ3
QUIZ4
QUIZ5
QUIZ6
SETUP
CLIMBS
QUIZ7
WOW
END
```



```
TO QUIZ1
SETBG 50
PR [HOW MANY EYES]
PR [DOES A SPIDER
HAVE?]
MAKE "ANS1 READLIST
IF OR :ANS1 = [8] :ANS1 =
[EIGHT] [PR [EXACTLY!]
STOP]
PR [ARE YOU SURE?]
PR [DO YOU WANT TO
TRY AGAIN?]
MAKE "TRY READLIST
IF :TRY = [NO] [PR [THE
ANSWER IS 8] STOP]
IF :TRY = [YES] [PR
[GOOD.] QUIZ1 STOP]
END
```

```
TO QUIZ2
SETBG 90
PR [WHAT DO SPIDERS
```

```

CATCH]
PR [IN THEIR WEBS?]
MAKE "ANS2 READLIST
IF OR :ANS2 = [FLIES] :ANS2 =
  [INSECTS] [PR [YES]]
PR [CAN YOU MAKE ANOTHER GUESS?]
MAKE "TRY READLIST
IF :TRY = [NO] [PR [OK, THANKS FOR
  TRYING.] STOP]
IF :TRY = [YES] [QUIZ2 STOP]
END

```

```

TO QUIZ3
SETBG 130
PR [CAN YOU THINK OF A SPIDER WEB
  NAME]
PR [THAT BEGINS WITH THE LETTER
  O?]
MAKE "ANS3 READLIST
IF OR :ANS3 = [ORB] :ANS3 = [ORB WEB]
  [PR [CORRECT! WATCH THE WEB
    GROW.] STOP]
PR [WOULD YOU LIKE ANOTHER
  CLUE?]
MAKE "TRY READLIST
IF :TRY = [YES] [PR [THE NEXT LETTER
  IS R.] QUIZ3 STOP]
IF :TRY = [NO] [PR [THE ANSWER IS ORB
  WEB. WATCH THE WEB GROW.] STOP]
END

```

```

TO QUIZ4
SETBG 32
PU SETPOS [40 10] PD
SETSH 1
SETC RANDOM (1 + 256)
SPIWEB
END

```

```

TO QUIZ5
SETBG 29
PR [HOW MANY TRIANGLES IN THE
  WEB?]
MAKE "ANS5 READLIST
IF :ANS5 = [6] [PR [THAT'S RIGHT!]
  STOP]
PR [THERE ARE 6 TRIANGLES. IT IS A
  HEXAGONAL SHAPE.]
END

```

```

TO QUIZ6
SETBG 60
PR [WHAT DO SPIDERLINGS HATCH
  FROM?]
MAKE "ANS6 READLIST
IF :ANS6 = [EGGS] [PR [YOU CAN'T BE
  TRICKED, CAN YOU?] STOP]
PR [TRY AGAIN. THE WORD HAS A
  DOUBLE LETTER IN IT.]
PR [DO YOU WANT TO HAVE
  ANOTHER GO?]
MAKE "TRY READLIST
IF :TRY = [YES] [QUIZ6 STOP]
IF :TRY = [NO] [PR [THE ANSWER IS
  EGGS.] STOP]
END

```

```

TO QUIZ7
SETBG 83
PR [WHAT IS A FAMOUS STORY
  ABOUT A SPIDER?]
PR [THERE ARE TWO CHOICES.]
PR [TRY TO SPELL THE TITLE
  CAREFULLY.]
MAKE "ANS7 READLIST
IF OR :ANS7 = [ARANEA] :ANS7 =
  [CHARLOTTE'S WEB] [PR [DID YOU
  ENJOY IT?]]
MAKE "TRY READLIST
IF :TRY = [YES] [PR [THAT'S NICE TO
  KNOW.] STOP]
IF :TRY = [NO] [PR [THAT'S OK. NOT
  EVERYONE ENJOYS SPIDER
  STORIES.] STOP]
END

```

```

TO GROWTRIANGLE :SIDE
IF :SIDE > 100 [STOP]
TRIANGLE :SIDE
GROWTRIANGLE :SIDE +5
END

```

```

TO TRIANGLE :SIDE
REPEAT 3 [FD :SIDE RT 120]
END

```

```

TO SPIWEB
REPEAT 6 [GROWTRIANGLE 1 RT 60
  SETC RANDOM (1 + 256)]
END

```



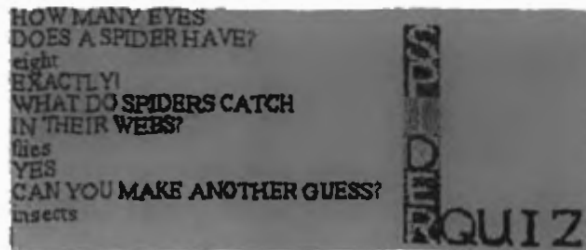
TO WOW
 SETBG 155
 TELL ALL PU
 TELL [1 2 3] HT
 TELL 0 RT 45 SETSH 31 FD 20 WAIT
 10 PD STAMP
 SETSH 32 PU FD 20 WAIT 10 PD
 STAMP
 SETSH 33 PU RT 3 FD 20 WAIT 10 PD
 STAMP
 SETSH 41 PU FD 20 WAIT 10 PD
 STAMP
 SETSH 38 RT 45 PU FD 20 WAIT 10 PD
 STAMP
 SETSH 40 RT 45 PU FD 20 WAIT 10 PD
 STAMP
 SETSH 42 PU FD 20 WAIT 10 PD
 STAMP
 SETSH 43 LT 45 PU RT 45 FD 20 WAIT
 14 PD STAMP
 SETSH 44 PU FD 20 WAIT 10 PD
 STAMP
 SETSH 45 PU FD 20 WAIT 10 PD
 STAMP
 END

TO CLIMB
 PU
 SETSH 1
 FD 5 WAIT 10
 SETSH 2
 WAIT 5
 SETSH 3
 FD 5
 WAIT 5
 SETSH 3
 FD 5 WAIT 10
 SETSH 2
 WAIT 10
 END

TO CLIMBS
 TELL [0 1 2 3] ST
 REPEAT 10 [CLIMB RT 36]
 REPEAT 9 [CLIMB]
 TELL [1 2 3] HT
 END

TO SETUP
 SETBG 28
 TELL ALL HOME PU

TELL 0 RT 45 SETC 238 TELL 1 RT 135
 SETC 239 TELL 2 LT 135 SETC 150
 TELL 3 LT 45 SETC 225
 END



TO SETSHAPES
 TELL 0 SETSH 4
 TELL [1 2 3]
 ST SETSH 5
 END

TO LINE
 TELL ALL
 CG PU
 SETPOS [0 0]
 RT 90 TELL 1 BK 12
 TELL 2 BK 24
 TELL 3 BK 36
 END

TO CRAWL
 REPEAT 10 [TELL ALL EACH [FD 10
 WAIT 10] WAIT 30]
 END

TO CATERPILLAR
 SETSHAPES LINE CRAWL
 END

TO SETSHAPES1
 TELL 0 SETSH 6
 TELL 1 ST
 SETSH 7
 END

TO LINE1
 TELL ALL
 CG PU
 SETPOS [0 0]
 RT 90 TELL 1 BK 15
 END



TO BUZZ
 REPEAT 10 [TELL ALL
 EACH [FD 10 WAIT 10
 RT 45] WAIT 30]
 END

TO FLY
 SETSHAPES 1 LINE 1
 BUZZ
 END

TO TITLE
 PU
 TELL 0 SETPOS [80 40]
 TELL [1 2 3] HT
 TELL 0 PU ST
 FD 50 SETC 10
 SETSH 31 PD STAMP PU
 BK 20 SETC 25
 SETSH 32 PD STAMP PU
 BK 20 SETC 225
 SETSH 33 PD STAMP PU
 BK 20 SETC 216
 SETSH 41 PD STAMP PU
 BK 20 SETC 151
 SETSH 38 PD STAMP PU
 BK 20 SETC 229
 SETSH 40 PD STAMP PU
 END

TO TITLE2
 RT 90 FD 20
 SETC 25
 SETSH 42 PD STAMP PU
 FD 20
 SETC 216
 SETSH 43 PD STAMP PU
 FD 20
 SETC 151
 SETSH 44 PD STAMP PU
 FD 20
 SETC 200
 SETSH 45 PD STAMP PU
 FD 20
 TELL 0 HT
 END

TO RUB
 CLEAN
 END

Dear Dr. Turtle...

Here's a problem I'm often asked about :

How can I print out the shapes from the Shapes page and their numbers when I'm using LogoWriter for the IBM or IBM compatible?

Well, it's simple. Just get all of the shapes showing on the screen on any page and print them with PRINTSCREEN! There's one little catch. How do you get them to show themselves on the workpage?

Here's one method. It should be easily adapted for other LogoWriters or Logos if needed.

```
TO SHOWSHAPES :N
  RG CT HT
  SETSH :N - 1
  PU SETPOS [-135 65]
  MAKE "Y LAST POS
  REPEAT 5 [REPEAT 6
    [SETSH SHAPE + 1
    LABEL SHAPE PU FD 25
    PS FD 25] MOVE]
  END
```

```
TO PS
  PD STAMP PU
  END
```

```
TO MOVE
  MAKE "Y :Y - 30
  PU SETPOS SE -135 :Y
  END
```

Now to show the first 30 shapes, type
 SHOWSHAPES 1 then print them out with

PRINTSCREEN.

Then you can type
 SHOWSHAPES 31
 and print them etc., etc.

(Of course, the above procedures are not "bullet proof". You would only have to type SHOWSHAPES 99, for example, and an error will occur. I assumed you would want to use them in a normal fashion, but if you want to make them fool proof, you could easily set some error traps, to test whether :N is in an acceptable range or not, for instance.) Happy shaping.

By the way, I have a problem and maybe someone out there can help ME for a change. You see, I have some pictures that I have scanned in and I want to load them into my PC Notebook in LogoWriter. But when LCSi "built" LogoWriter, they made it recognise only one peculiar picture format. When I try to load my picture with LOADPIC I get an error message saying "Wrong file format". I have tried several file formats using a file conversion program but nothing seems to help.

If you can offer any help/advice/suggestions, please write your answer to the editor of LogoFile. I'll publish the solution in one of my columns.

Dr. Turtle

**Investigating Logo as a
Programming and
Software Development
Environment: "The
Towns of France"**

**W. Martin Boyle
Melbourne Grammar
School
Victoria**

Setting the Scene

I set myself a problem the solution to which was to be coded in Logo. I use this problem as a standard to test the functions of various programming languages and I wanted to move away from the only aspect of Logo which I had seen before - turtle graphics. I was fairly proficient in using the turtle, but had no knowledge of other aspects of the language, so I was learning as I worked on the project.

Problem Specification

Display a map of France and a list of towns in France. A town is chosen at random and an appropriate symbol is displayed at the correct position on the map. The user is invited to enter the letter corresponding to the town, and if correct is told so in an encouraging manner. If the response is incorrect the user must try again until correct. As the towns are correctly chosen they are removed until all twelve towns in the list have been displayed.

I have a full top-down design and graphics grid for the solution to the problem so my main concern was to translate those designs into something which could be coded in Logo.

First I wanted to place text at given positions on the VDU. I started by trying the PRINT command but found I had no control over the positioning as it was by default on the left. The LCSi manual introduced me to SETPOS and LABEL solving the problem.

I wanted to be able to play the National Anthem of France. The LCSi Logo reference Guide (p. 2-96) gives a wonderfully useful table of the code numbers for our musical notation. (I've spent many hours working these out for other languages!) A bit of hit-and-miss whistling and the useful note on WAIT in the manual soon gave rise to a somewhat off-key Marseillaise.

The immediacy of the graphical interface in Logo is a real delight for a junior secondary software developer, though it must be said that the IBM version (which I used) is grossly inferior in comparison to the Mac!

A book by Haigh (1986) gave me my start for designing the graphics of the map of France. An excellent section on the United States described a technique of closed curve filling which was exactly what I wanted. I decided to

display the map on the right of the screen and the town list on the left so I could use simple PRINT statements for the town list.

What versatility and what fun to be able to design your own shapes! In comparison my previous versions of this software have been dull! My first intention was that the flashing cursor representing a town would be a circle but then I designed on the shapes an Eiffel Tower as cursor.

Then a powerful idea for children developed: design a shape which gives a visual clue as to the name of the town and display that shape in the town's location. Thus the Eiffel Tower will be used for Paris, a red wine bottle for Bordeaux, a bridge for Avignon, and so on. Some quite detailed research is needed so that the town shapes are placed at the correct position on the map.

My excitement at this discovery in Logo was fuelled by realising that now as well as gaining knowledge about the geography of France we had the bonus of knowledge of the culture of the country. This will become a powerful idea in my development of the software for next year's Grade Seven.

Now for some interesting programming! I want a procedure which will:
1. check to see if all the towns have been displayed; thus I need a suitable data structure

to hold the towns;

2. create a number at random in the range 1 to 12, corresponding to each town;

3. check if the town has already been removed from the town data structure;

4. if it has been removed keep generating a random number until I get one which is in the structure;

5. now that we're using the number get rid of it from the town structure so it won't be used again;

6. call the appropriate town procedure, depending on the number;

7. assign the correct answer to the town; this will be a letter of the alphabet which the user must press.

My solutions to these problems were as follows:

1. Harvey (1988) and McDougall (1982) put me on to the LIST data structure. So I will need to initialise a list which I call "ALLNOS" containing the numbers 0 to 11. I will need to ask questions of this list, for example if it empty? The IF structure coupled with the handy EMPTY? offered the promise of success.

2. Random number generation turned out to be easy: RANDOM 12 gave me a number in the range 0 to 11 and when 0 is generated I convert it to 12 to get my town list 1 - 12.

3. However, thinking ahead, if a town has been selected already it will have been

removed from the list and I would want to generate another random number before going on - so if the procedure GENRAND does not come up with the goods it has to call itself to try again. I had finally arrived at recursion. Harvey, McDougall, and Haigh had much to say on recursion, in fact Harvey so much that I thought of over-kill!

4. I found that I could check for membership of a list in IBM Logo with MEMBER? and if the number wasn't there I would call GENRAND recursively.

5. The first attempt I made to solve the removal problem made me realise that this was complex and so I decided to shelve the problem. I would get the rest of the software working, accepting repeats of the same town, before returning to the problem of stripping out towns once they had been used.

6 & 7. Now a whole series of assignments were necessary to call procedures to display a town's shape and to assign correct answers. Some false starts made it important to sort out assigning with the "=" operator and inspecting the value of variables with the "=" operator.

Now I needed to give the user a simple instruction on what to do, and then get his or her response.

The first problem was, as

expected, easy to implement but the second came close to undoing the entire project. Were I to start again the entire top-down design of the structure of the software would be different - an almost inevitable position in computing when starting to develop the solution to a complex problem from a knowledge base of zero!

My instructions were put in place with SETPOS and LABEL.

The READCHAR keyword was what I would need to get the user input. There were to be two possibilities. If the entry was correct the user would be congratulated and the procedure which called the USERTRY procedure would be called to start the cycle over again from the beginning. So the procedure which called the procedure would in turn be called by the procedure it called! I'm very deep into Hofstadter (1979) here with a fair measure of Lewis Carroll and a splash of Russell's Paradox - can I come out the other side!

The second case is when the users give the wrong response. Then we give them another go by calling the USERTRY procedure recursively. IFELSE will solve the technical problem of choice.

Now to put it all together.... I will drive the software package by a procedure called MAIN which will call each of

the other necessary procedures until all of the towns have been correctly identified.

At this stage I carried out testing, both myself and with Year 7 students, and all worked well except that we were not stripping out towns. So now I can postpone no longer returning to the list processing problem!

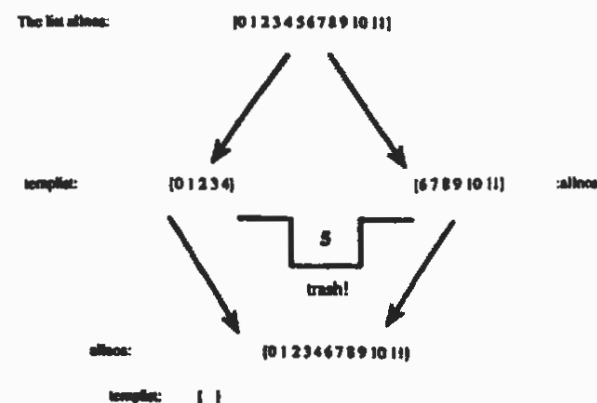
This session was actually spread over quite a number of days. The purpose of the strip procedure is to search through the town list until it comes to the one we're after; then it needs to return a new list with that one removed. In this way, as we cycle through the software, the list is gradually reduced until we've identified all of the towns.

It took quite a few attempts using FIRST, BUTFIRST, SENTENCE, IFELSE and a temporary list before we had a working model.

STRIP is the heart of the software and the most complex and compact of the procedures. It is a tail recursive procedure which searches through the list using FIRST until it finds the number we're after. Meanwhile, numbers already looked at are stored in a temporary list.

When found, the number is discarded from our now depleted list, which is then restored to the position it should be in by the addition of the members of the temporary list.

So, for example, we're after 5...



... and we're ready to go again with the new list ALLNOS!

```

TO FROGBAK6
CT
CG
HT
CLEAR NAMES
CLEAR TOOLS
MAKE "ALLNOS [0 1 2 3 4 5 6 7 8 9 10
11]
MAKE "TEMPLIST []
TITLES
MAIN
END

```

```

TO MAIN
FROGMAP
TOWNLIST
INSTRUCT
GENRAND
USERTRY
END

```

```

TO TITLES
SETPOS [-70 30]
LABEL [TOWNS IN FRANCE]
TONE 294 2 WAIT 1
TONE 294 1 WAIT 1
TONE 392 8 WAIT 1
TONE 392 8 WAIT 1
TONE 440 8 WAIT 1
TONE 440 8 WAIT 1
TONE 587 12 WAIT 1
TONE 494 3 WAIT 1
TONE 392 8 WAIT 1
HOME
WAIT 50
CG
END

```

```

TO FROGMAP
SETPOS [64 80]
PD
SETPOS [76 72] SETPOS [84 64]
SETPOS [96 56] SETPOS [108 46]
SETPOS [120 40] SETPOS [136 34]
SETPOS [144 32] SETPOS [140 16]
SETPOS [131 4] SETPOS [115 -4]
SETPOS [101 -14] SETPOS [102 -22]
SETPOS [112 -16] SETPOS [116 -28]
SETPOS [111 -44] SETPOS [112 -56]
SETPOS [121 -64] SETPOS [109 -71]

```

```

SETPOS [95 -77] SETPOS [87 -84]
SETPOS [77 -75] SETPOS [62 -69]
SETPOS [44 -73] SETPOS [36 -81]
SETPOS [34 -85] SETPOS [22 -83]
SETPOS [10 -86] SETPOS [-2 -88]
SETPOS [-6 -81] SETPOS [-20 -85]
SETPOS [-34 -77] SETPOS [-45 -69]
SETPOS [-51 -61] SETPOS [-44 -49]
SETPOS [-40 -37] SETPOS [-34 -26]
SETPOS [-32 -16] SETPOS [-32 4]
SETPOS [-38 18] SETPOS [-43 26]
SETPOS [-54 29] SETPOS [-54 35]
SETPOS [-54 39] SETPOS [-48 41]
SETPOS [-34 45] SETPOS [-22 37]
SETPOS [-14 39] SETPOS [-10 56]
SETPOS [0 59] SETPOS [9 49]
SETPOS [21 53] SETPOS [31 59]
SETPOS [41 64] SETPOS [47 76]
SETPOS [64 80]

```

PU

```

SETPOS [0 0]
PD SETC 1 PU
END

```

TO TOWNLIST

```

PRINT "
PRINT "
PRINT "
PRINT "
PRINT "A...BREST
PRINT "B...BORDEAUX
PRINT "C...AVIGNON
PRINT "D...NICE
PRINT "
PRINT "E...PARIS
PRINT "F...LYON
PRINT "G...LILLE
PRINT "H...TOULOUSE
PRINT "
PRINT "I...STRASBOURG
PRINT "J...MARSEILLE
PRINT "K...NANTES
PRINT "L...GRENOBLE
END

```

TO GENRAND

```

IF EMPTY? "ALLNOS [STOP]
MAKE "TOWN RANDOM 12
IF NOT MEMBER? :TOWN :ALLNOS
[GENRAND]

```

```

IF MEMBER? :TOWN :ALLNOS [STRIP]
IF :TOWN = 0 [MAKE "TOWN 12]
IF :TOWN = 1 [PARIS]
IF :TOWN = 2 [LILLE]
IF :TOWN = 3 [STRASBOURG]
IF :TOWN = 4 [BREST]
IF :TOWN = 5 [NANTES]
IF :TOWN = 6 [BORDEAUX]
IF :TOWN = 7 [LYON]
IF :TOWN = 8 [AVIGNON]
IF :TOWN = 9 [GRENOBLE]
IF :TOWN = 10 [NICE]
IF :TOWN = 11 [MARSEILLE]
IF :TOWN = 12 [TOULOUSE]
IF :TOWN = 1 [MAKE "ANS "E]
IF :TOWN = 2 [MAKE "ANS "G]
IF :TOWN = 3 [MAKE "ANS "I]
IF :TOWN = 4 [MAKE "ANS "A]
IF :TOWN = 5 [MAKE "ANS "K]
IF :TOWN = 6 [MAKE "ANS "B]
IF :TOWN = 7 [MAKE "ANS "F]
IF :TOWN = 8 [MAKE "ANS "C]
IF :TOWN = 9 [MAKE "ANS "L]
IF :TOWN = 10 [MAKE "ANS "D]
IF :TOWN = 11 [MAKE "ANS "J]
IF :TOWN = 12 [MAKE "ANS "H]
END

```

TO STRIP

```

IFELSE :TOWN = FIRST :ALLNOS [MAKE
"ALLNOS SE BUTFIRST :ALLNOS
:TEMPLIST MAKE "TEMPLIST [] STOP]
[MAKE "TEMPLIST SE :TEMPLIST FIRST
:ALLNOS MAKE "ALLNOS BUTFIRST
:ALLNOS STRIP]
END

```

TO INSTRUCT

```

SETPOS [-140 90]
LABEL [ENTER THE LETTER OF THE
FLASHING TOWN]
END

```

TO USERTRY

```

SETPOS [0 0] LABEL [ ]
SETPOS [-23 20] LABEL [ ]
IFELSE READCHAR = :ANS [SETPOS [0
0] LABEL "EXCELLENT!!! WAIT 20
LABEL "EXCELLENT!!! MAIN] [SETPOS
[-23 20] LABEL [HAVE ANOTHER GO!!!]

```

WAIT 20 LABEL [HAVE ANOTHER GO!!!]
USERTRY]
END

TO PARIS
SETSH 1
PU
REPEAT 3 [ST SETPOS [60 28] WAIT 10
HT HOME WAIT 10]
HT
END



TO LILLE
SETSH 2
PU
REPEAT 3 [ST SETPOS [76 62] WAIT 10
HT HOME WAIT 10]
HT
END



TO STRASBOURG
SETSH 3
PU
REPEAT 3 [ST SETPOS [130 20] WAIT 10
HT HOME WAIT 10]
HT
END



TO BREST
SETSH 4
PU
REPEAT 3 [ST SETPOS [-50 35] WAIT 10
HT HOME WAIT 10]
HT
END



TO NANTES
SETSH 5
PU
REPEAT 3 [ST SETPOS [-25 6] WAIT 10
HT HOME WAIT 10]
HT
END



TO BORDEAUX
SETSH 6
PU
REPEAT 3 [ST SETPOS [-25 -30] WAIT 10
HT HOME WAIT 10]
HT
END



TO LYON
SETSH 7
PU
REPEAT 3 [ST SETPOS [75 -25] WAIT
10 HT HOME WAIT 10]
HT
END



TO AVIGNON
SETSH 8
PU
REPEAT 3 [ST SETPOS [60 -40] WAIT
10 HT HOME WAIT 10]
HT
END



TO GRENOBLE
SETSH 9
PU
REPEAT 3 [ST SETPOS [100 -35] WAIT
10 HT HOME WAIT 10]
HT
END



TO NICE
SETSH 10
PU
REPEAT 3 [ST SETPOS [114 -60] WAIT
10 HT HOME WAIT 10]
HT
END



TO MARSEILLE
SETSH 11
PU
REPEAT 3 [ST SETPOS [75 -70] WAIT
10 HT HOME WAIT 10]
HT
END

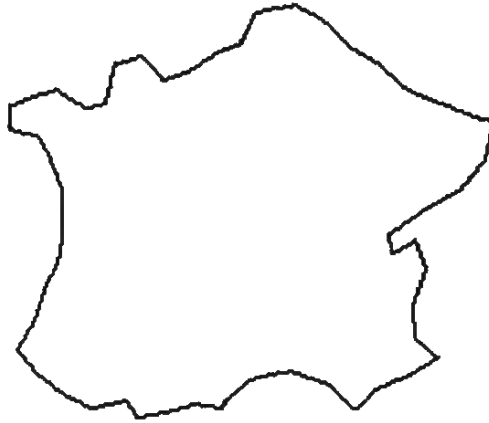


TO TOULOUSE
SETSH 12
PU
REPEAT 3 [ST SETPOS [-12 -68] WAIT
10 HT HOME WAIT 10]
HT
END



ENTER THE LETTER OF THE FLASHING TOWN

- A...BREST
- B...BORDEAUX
- C...AVIGNON
- D...NICE
- E...PARIS
- F...LYON
- G...LILLE
- H...TOULOUSE
- I...STRASBOURG
- J...MARSEILLE
- K...NANTES
- L...GRENOBLE



ENTER THE LETTER OF THE FLASHING TOWN

- A...BREST
- B...BORDEAUX
- C...AVIGNON
- D...NICE
- E...PARIS
- F...LYON
- G...LILLE
- H...TOULOUSE
- I...STRASBOURG
- J...MARSEILLE
- K...NANTES
- L...GRENOBLE



I will need to redesign the structure of the "Towns in France" software now that I am more confident in the language. In particular I will localise data structures and pass them between procedures, and re-examine the recursion which slows the program down noticeably as the town data structure becomes depleted.

It's a great pity, but the IBM version is poor compared with the Macintosh and pathetic when measured against MicroWorlds Project Maker.

The obvious geometric and general mathematical advantages of Logo appeal; and I think that the most significant insight for me from the whole exercise is that now I can see how to take these powerful mathematical tools and incorporate them into game style applications which in turn can be linked into database applications.