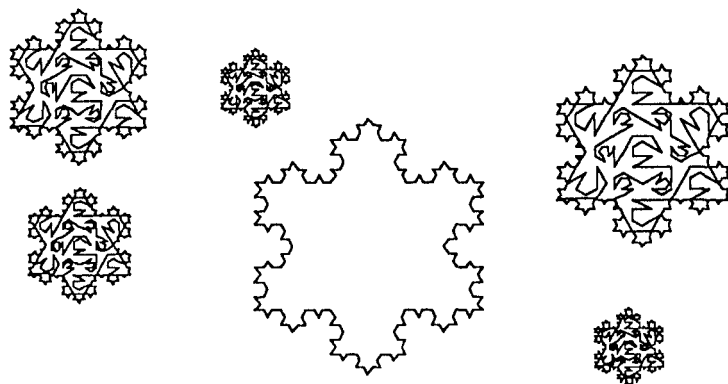


POALL

A Journal for Logo Users



Students at Work

Volume 6 Number 3, November 1991

Two fine papers this time, different, but connected in that they report work by students using Logo as a language to explore and create. We have here students establishing intimate contact with deep ideas from mathematics, history, geography and intellectual modelling, exactly what Logo was intended for. The Logo is not the most elegant in places, but elegance comes with experience, and that is what these young people are having in sympathetic environments. Special thanks to Jenny Betts and John Turner for sharing with us.

I'm taking leave next year. Time to go to sea for a while, rewrite *Thinking Logo*, paint the house, look for a writing job, etc. etc. I've been offered endless hours of part-time canoeing instruction, and I can be available for some consulting. Since I won't be in a classroom I'll depend more than ever on your contributions.

If you'll be moving for 1992, please let me know your new address. Should your copy of *POALL* be returned undelivered your name will be deleted from the mailing list.

All the best for Christmas and the New Year. May all your Turtles live long and prosper.

Peter

POTS

Papert's Vision - Students in Control of their Learning	2
The creation of databases using LogoWriter	6
Computing at Entropy House	20

Papert's Vision - Students in Control of Their Learning

John Turner, Presbyterian Ladies' College, Melbourne

(with special thanks to Audrey Yeo, Year 10 PLC student)

In his 1980 book *Mindstorms: Children, Computers and Powerful Ideas*, the founder of Logo, Seymour Papert, was extremely critical of the traditional school environment, attacking the classroom as 'artificial and inefficient' (p 8). As a post-primary teacher I have seen at first hand how set time periods for classes, and the demand to achieve units of prescribed syllabus orientated work, can work against the student using Logo as it was intended. Too often students are inhibited in their potential use of Logo beyond a limited role as a tool to complete activities that could otherwise have been completed in the conventional way with textbook, pen and paper.

To what degree, therefore, are Logo related processes transferable to conventional post-primary school communities so that all students, and teachers, can benefit from what Logo has to offer? General Logo based research in this area appears too often to rely on Logophile influences intrinsically linked to its success. I believe that if examples can be promoted applicable to traditional school demands, then, rather than waiting for revolutionary led change, reformation can be brought to each student's learning much sooner.

I am pleased to be able to provide some light to such a path. Not surprisingly the following occurred in my Year 10 Computer Studies class, and not in the Mathematics mainstream (where I also like to swim).

My students had undertaken a series of investigations looking at the potential of Logo over a wide range of activities. This had included looking at music and animation, problem solving, databases and recursion. LogoWriter on the Apple IIe was the medium.

The recursion study was generally not School Math orientated, but instead was built around McDougall and Adams' (1983) procedures for writing poetry linked to Thornburg's recursive poem 'The House that Jill Built' (Thornburg, 1986, pp 97-102).

Following these investigations the students were given options for project work; either their own choice or from a list of possibilities. One of these called for an investigation of von Koch's curve, a simple shape around which snowflake investigations could be built.

```
TO Koch :side
FD :side LT 60
FD :side RT 120
FD :side LT 60
FD :side
END
```



One student, Audrey, took up this offer. In Mathematics she had already studied snowflakes and developed formulae to calculate the length, perimeter, number of sides and area of snowflakes as part of a unit on Series.

For the first part of her project, Audrey identified the need to design procedures for the formulae calculations - a relatively straightforward step. She applied recursive thinking to solve the simplest problem first. What happened next is best told by drawing on Audrey's procedures and comments.

```
TO Startup
CG CT HT
PU SETPOS [-50 -80] PD
Tri 90 1
No.Sides 1
Perimeter 1
Area 1
```

(continued on next page)

```

PR (SE [This is the basic shape used in the construction of the
snowflake. It has]
:ns [sides, each side measuring one-third of a unit of Logo
length. The total perimeter is]
:perim [units and the area is] :a [square units])
PR "
PR [Press RETURN key to continue]
MAKE "waiting READLISTCC
CG
PR "
PR [To look at the snowflake of any level greater than 1 type
in SNOWFLAKE followed by the level required (e.g. SNOWFLAKE 3)
then press RETURN key]
END

```

Tri, No.Sides, Perimeter and Area are the procedures used to generate the graphic, and formulae for the number of sides, perimeter and area respectively. Tri used a subprocedure SF.

```

TO Tri :side :level
IF :level = 0 [STOP]
REPEAT 3 [SF :side :level RT 120]
END

```

(Audrey's Comment: Repeats SF to draw a complete snowflake)

```

TO SF :side :level
IF :level = 0 [FD :side STOP]
SF :side / 3 :level - 1
LT 60
SF :side / 3 :level - 1
RT 120
SF :side / 3 :level - 1
LT 60
SF :side / 3 :level - 1
END

```

(Audrey's Comment: SF breaks down :side and :level to smallest possible values then draws one side of snowflake relating to :level. For simplicity, :side was given a set value)

The setting of the :side value was also a response to the limitations of the LogoWriter screen.

```

TO No.Sides :level
MAKE "ns (3 * Power 4 :level)
END

```

```

TO Side.Length :level
MAKE "sl (1 / Power 3 :level)
END

```

```

TO Perimeter :level
MAKE "perim ((Power 4 :level) / (Power 3 (:level - 1)))
END

```

```

TO Area :level
MAKE "a (((1 / 3) * (1 - Power 4 / 9 :level))
/ (1 - (4 / 9))) + 1
END

```

These procedures were then incorporated into the general case for a Snowflake of any level.

```

TO Snowflake :level
CG CT HT
PU SETPOS [-50 -80] PD
Tri 90 :level
No.sides :level
Side.Length :level
Perimeter :level
Area :level
PR (SE [On a snowflake of]
:level [levels there are]
:ns [sides, each side being]
:s1 [units long, with a total perimeter of]
:perim [units. The overall area of this snowflake is]
:a [square units])
PR [Press RETURN key to continue]
MAKE "waiting READLISTCC
CG CT HT
PR [To Quit press ESC key, otherwise enter SNOWFLAKE and the
level required]
END

```

(Audrey's Comment: Snowflake draws the snowflake using user commands for the level, then calculates number of sides, perimeter, side length and area: and prints them out on screen) {Editor's Comment: There's a sample snowflake on the front page.}

In order for these formulae to do their job Audrey now identified that she needed a procedure to calculate indices. She became driven by the need to solve a perceived Logo mathematical shortfall. She questioned whether she should accept the challenge of creating a procedure or seek assistance from others.

Other students wanted to join in but Audrey considered (1) she had the capabilities, and (2) she wanted to make sure that she would be able to 'think and understand'. Other students added value but Audrey limited their input so as to ensure that she maintained control of the environment.

(Audrey's Comment: In order to calculate values a Power procedure was necessary. As LogoWriter did not have this I had to create one. The first attempt was usable only with the snowflake procedure)

```

TO SFPower :level
IF :level = 1 [STOP]
MAKE "sq :Sq1 * :sq
SFPower :level - 1
END

```

(Audrey's Comment: For this to work each variable had to be given a value in the procedure it was used in before it could work. So a new procedure was created which could be used to support any other procedure, instead of just a snowflake)

```

TO Power :b :i
MAKE "resi 1
MAKE "count 0
POE :b :i
OP :resi
END

```

```

TO POE :b :i
IF :i = 0 [ZeroI :b :i]
IF :b = 0 [ZeroB :b :i]
IF :i < 0 [NegI :b :i]
IF :count = :i [STOP]
MAKE "resi (:resi * :b)
MAKE "count :count + 1
POE :b :i
END

```

```

TO ZeroI :b :i
MAKE "resi 1
END

```

```

TO ZeroB :b :i
MAKE "resi 0
END

```

```

TO NegI :b :i
Neg
Change
END

```

```

TO Neg
IF :count = :i [STOP]
MAKE "resi :resi * :b
MAKE "count :count - 1
Neg
END

```

```

TO Change
MAKE "post 1 / :resi
MAKE "posti :post * 10000
MAKE "resi (INT :posti) / 10000
END

```

(Audrey's Comment: NegI uses the same recursion as POE, but when :count = :i the reciprocal is found (negative indices). The computer cannot handle such long figures, and so the result is multiplied by 10 000 to give a larger integer. Then, using the INT command the decimal part is discarded and the remainder divided by 10 000 so that the final figure has only up to 4 decimal places. The procedure can now be used to support any other procedure, providing there are no fractional indices!)

In writing *Power* Audrey had not only created a general procedure of considerable mathematical force, she had also in the process provided confirmation of one of Papert's fundamental beliefs - she had established a personal relationship with the computer to explore new frontiers of knowledge (1980, p 6). within a post-primary context.

Audrey commented on how much the 'thinking aspect' meant to her. The computer had become a tool to support her thinking. In her own words most of the work involved 'thinking away from the computer'. Her reflective comments also confirm the control she had established over her own thinking.

The value of this project towards the development of Audrey's mathematical understanding is also noteworthy. As well as direct mathematical application it provided her with insight into the value of planning, debugging and modular thinking. But much more, it opened her eyes to wanting to know how something worked. Previously she had been contented with just understanding its uses. Audrey likened mathematics to the computer - 'we know how to use it but we don't know how it works'.

When last seen Audrey was exploring logarithms and series approximations in order to expand her power procedure to cope with non-integer indices.

References

- McDougall A. and Adams T. (1983) 'Teaching a Computer to Write Poetry', in *Information Transfer* 3(3) p 6-14. Reprinted in Salvas A. (1984) *Computers in Education: 1984 and Beyond*, Computer Education Group of Victoria, pp 130-6
- Papert S. (1980) *Mindstorms: Children, Computers and Powerful Ideas*, Great Britain. Harvester Press.
- Thornburg D. D. (1986) *Beyond Turtle Graphics - Further Explorations of Logo* Addison-Wesley



The creation of databases using LogoWriter

Jenny Betts, Queensland Sunrise Centre, Coombabah

Introduction

Reporting the progressive steps children undertook in developing a menu for a Logo database used as a part of Social Studies, is the focus of this article. It reflects 18 months of observing the children's progression of endeavouring to develop a user friendly menu which is now used across the curriculum.

It would be true to say that the development which took place was not a "true" database as it did not contain what we associate with databases, and that is fields. These databases can be more closely associated with the form of "Choose your own path" type books. Nevertheless, even though our programming methods may seem crude to expert Logo programmers, together the children and I made some startling discoveries, especially when both the children and I were very much beginners.

The Original Plan

So often children use software that is already programmed and I almost trapped myself into believing that this was the road to take with computers in Education until the day I was introduced to Logo. Commercial software packages have their value within the class environment, however, I noticed Logo provided children with another set of experiences. My experiences with Logo had been very little, in fact I had never drawn the square that everyone seemed to be able to do, due to avoidance on my part. Shape creating was not the motivating force behind why I have chosen to use Logo, although I can now see the advantages in doing so. With the introduction of Logowriter on the market, using Logo has enabled students to create microworlds¹ by manipulating not only numbers but also text. Children have immersed themselves with programming adventure games, databases (of sorts), crosswords, word games, speed reading programs etc. and all have related to the class theme at the time. Logo has made it possible for children to be in control of learning difficult concepts while having fun.

Concentrating on the development of the programming skills is never the main focus in my class, it is developing skills and accumulating knowledge. In any event, using Logo not only allows for the development of many logical thinking processes because of the programming skills involved, but it has also motivated children, allows for children at all academic levels to achieve and they accumulate a wealth of knowledge.

¹ Papert, S. (1980) *Mindstorms: Children, Computers and Powerful Ideas* (New York: Basic Books)

Problems Faced

Because of the children's limited knowledge of programming, they faced four main problems. They were;

1. Having easy to follow instructions
2. Having a simple screen arrangement - to include instructions, text and graphics
3. Accessing Information
 - using more than one screen
 - menus
 - searching

1. Easy to follow Instructions

Preconceptions of how instructions should be written were not evident. At first the children's instructions were quite lengthy. Perhaps the lengthy instructions could have related to the limited knowledge of logo programming or limited knowledge of how computer programs worked.

(Carl)...

Type which one.

or

Press the letter that you want
to see and you can type q to quit.

- A. ARCHITECTURE
- B. CLOTHING
- C. RELIGION

Regardless, patience is the key to meaningful progression. Allowing the children to experience the lengthy instructions is very important, although at times, it is difficult not to step in and give advice. They should be left alone to create their own instructions. As we progressed, the children began to develop procedures that they shared and refined with their friends. Discussions were held on how instructions were being worded and examples were shown.

(Darren)...

This is a program about the romans and their life style. Choose one of the following names and type in the letter beside it in the command centre. To get back to the database type data

- A. ARCHITECTURE
- B. CLOTHING
- C. RELIGION

Trying to use more simple instructions seemed to mean more to the children after they had experienced the inconvenience of more involved instructions.

2. Layout of screen

Instructions

As each screen of information was shown, it was difficult to read because the instructions and the information were cramped.

Instructions were often attached to the bottom of the print. This meant that their position depended on how much information was on the screen. Instructions therefore, were never found in the same place. Each time a new screen was given, you had to search for the instructions.

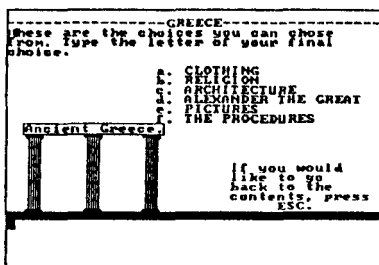


Figure 1 Instructions are not clearly noticeable

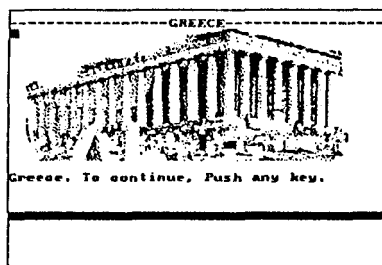


Figure 2 Different place and different instructions

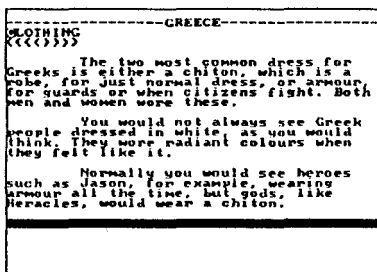


Figure 3 No instructions at all. In fact you wait.

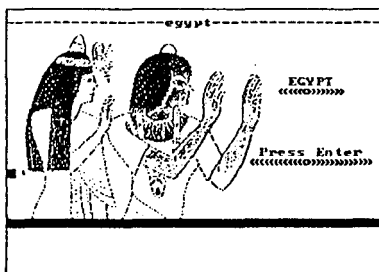


Figure 4 Clear instructions but once again not consistent with the rest of the program.

Not all these pages came directly after one another. Notice how the user faces several different methods of accessing information.

Having the instructions and the information together also meant that less information was able to be placed on the screen at any one time. It became apparent that it might be useful to display instructions in a manner where the user would Always know exactly where to look for them and the screen would not be overcrowded with information and instructions. Using the command centre seemed a great idea and it meant only a slight change to programming. Not only did the children learn about layout of a page but they now had two more commands

to work with - TYPE and SHOW. These have also been used quite extensively.

Original Version	Changes
<pre>To continue Pr [Press any key to continue.] name readlist "any.key end</pre>	<pre>To continue Show [Press any key to continue.] name readlist "any.key end</pre>

3. Accessing Information

(1) Using more than one screen

(a) Manual

Children were writing more than one screen of information on certain areas and found themselves having to deal with how the user could get access to all the information. One method was:-

Print all the text on the screen and then tell the user to;

- press <control>-U to move the cursor to the text centre
- use the arrow the keys to move through the text
- press <control>-D to move the cursor down to the command centre.

How we showed all the text and all these instructions was very difficult but manageable however, the problem with this method was that there was no way to return to the main menu. This was not too successful because the instructions were too long and it was too much for the user to remember.

Encouraging the children to find a way that didn't require the user to know how to use Logowriter was my role.

(b) Waiting

Another popular method was using the wait command. Some children programmed the computer to show one screen at a time, using the wait command between screens.

```
Eg  To egyptian
      pr [information]
      top
      wait 100
      nextscreen
      wait 100
      menu
      end
```

NB: "Information" is the data which you want displayed.

Catering for the slower reader was popular. They found this annoying when they were testing their programs as they had to wait quite a while before the next screen appeared.

(c) Scrolling

Another method was to display all of the text, go to the top of the page and program the computer to scroll through the text.

```
to egyptian
  pr [Information]
  top
  repeat 34 [wait 10 cd]
  menu
end
```

NB: "Information" is the data which you want displayed. The number of repeat depends on the number of lines in the text.

(d) Controlling

Being in control of the computer rather than letting it control you has been a priority instilled in the children. When it came to programming, it was still a matter of making sure the user was in control. I suggested that they should try and program so that the user can continue when they were ready rather than having to wait until the computer is ready. Two methods were used.

(i) Continue Tool

```
to egyptian
  pr [Information]
  top
  continue
  nextscreen
  continue
  menu
end
```

Continue is a subprocedure which halts the program until a key is pressed.

```
To continue
  show "Press any key to continue.
  name readchar "any.key
end
```

(ii) Arrow Tool

```
to arrows
  pr [information]
  top
  show "Use ↓ and ↑ to view text
  show "Press ESC to exit
  pause
end
```

Pause is a subprocedure which looks for the arrow and ESC key to be pressed.

```

to pause
  name readchar "cursor
  if :cursor = char 27 [menu stop]
  if :cursor = "H [cu]
  if :cursor = "P [cd]
  pause
end

```

To find the correct ascii characters for key type "show ascii readchar". Press enter. The computer is now waiting for you to press the key you wish to find the ascii character for. In this case press ESC and the computer will respond with the number 27.

To find the symbol for a character type "show readchar" and press "enter". The computer is now waiting for you to press the key you wish to find the symbol for. In this case press one of the arrow keys. The computer will respond to this by showing a space and the capital "H". You must remember that this is a symbol and not a letter.

(II) Menu

(a) Text Menu

Looking back it seems a minor hurdle but creating a menu was our biggest breakthrough. Ideas for these procedures were taken from students, books, the manual and visiting experts. Any new ideas were quickly seized and modified to suit our purposes.

Writing procedures that would allow the user to select from a menu was not difficult but our knowledge was very limited. We all started with the simplest methods we knew and that was

(Nathan)....

```

to startup
  rg ht ct
  pr [Do you want]
  pr []
  pr [a. Architecture]
  pr []
  pr [b. Lifestyle and Furniture]
  pr []
  pr [c. Religion]
end

to a
  gp "egypt
end

```

```
to b
gp "egypt1
end
```

```
to c
gp "egypt2
end
```

This worked quite well until we found, I should say a student found, a tool that would be extremely useful for many things other than the uses we now needed. It was:

(Warwick)....

```
To menu
  pr [Choose 1 - 3]
  pr []
  pr [1. Egypt]
  pr [2. Greece]
  pr [3. Rome]
  choose
end

to choose
name readchar "choice
if :choice = 1 [egypt]
if :choice = 2 [Greece]
if :choice = 3 [Rome]
end
```

This was a first major breakthrough. This tool been used in many of their programs and appears in many different forms. This method has allowed us to use very simple wording for instructions. For the user, the selection process is quite simple.

(b) Turtle Menu using POS

This menu was developed when the children used a commercial game where the user used the arrow keys to place an arrow next to the requested topic and then pressed enter. The menu may look like figure 5.

The procedures are:-

```
to menu
rg ct ht cc
loadtext "menu.txt
loadpic "menu.pic
setsh 1 st seth 0
pu setpos[-150 40]
choose
end
```

This procedure loads the graphics and the text. It sets the scene which is seen in figure 5. The turtle is given its shape, pointed in the right direction and placed in the position near the first item, "other.options".

```
to choose
  arrow
  middle?
  choose
end
```

The choose subprocedure makes the whole menu procedure run.

```
to arrow
  name readchar "up.down
  if :up.down = char 13 [position?]
  if :up.down = "H [fd 10]
  if :up.down = "P [bk 10]
  arrow
end
```

This procedure checks to see if the up or down arrow is pressed. If so, then the turtle will move in the correct direction ie if the up arrow key is pressed the turtle will move forward 10. Finding the symbol for a keyboard character and ascii character numbers is explained in the arrow tool.

```
to middle?
  if pos = [-150 50] [bk 10]
  if pos = [-150 -70] [fd 10]
end
```

This subprocedure checks to see that the turtle does not move above or below the top and bottom topics.

```
to position?
  if pos = [-150 40] [other.options]
  if pos = [-150 30] [animal.menu]
  if pos = [-150 20] [clothing.menu]
  if pos = [-150 10] [food.eaten]
  if pos = [-150 0] [history.menu]
  if pos = [-150 -10] [housing.menu]
  if pos = [-150 -20] [group.life]
  if pos = [-150 -30] [religion.menu]
  if pos = [-150 -40] [social.control]
  if pos = [-150 -60] [next.menu]
end
```

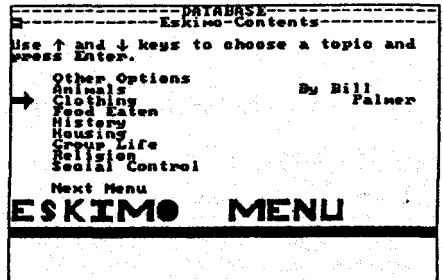


Figure 5 Sample of a Turtle Menu using POS

When the enter key is pressed, the computer will check to see the turtles position. If this is a position the turtle has been program to respond to the computer will run that subprocedure.

(c) Turtle Menu using CHARUNDER

This tool was invented by a group of 12 year old boys, Warwick Mitchell, Tim Schmidt, Colin Ruscoe and Nathan Hawkins. It is the most recent discovery and was developed when the children used a commercial database called PC Globe. It is very similar to the way the PC Globe map works.

This graphic has been loaded over the top of a text file which has been loaded in a the colour 0 so that it cannot be seen by the user. Each country or word from the menu has been assigned a character (Figure 3). As you move the turtle around the screen, the turtle checks to see which character it is under. It will then show in the command centre the appropriate country.

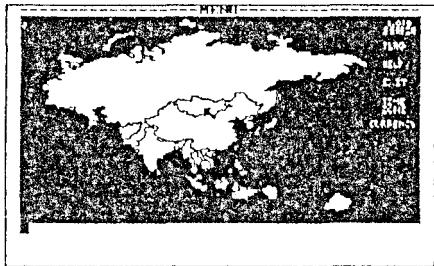


Figure 6 Map uploaded from PC Globe with modifications

If you look closely you can see how each country and menu word has been assigned a character.

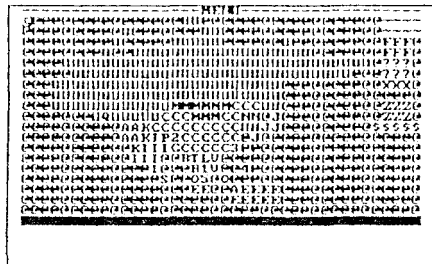


Figure 7 Text File which is loaded underneath the graphic

The procedures for this are:-

```

Line 1      to main.menu
Line 2      rg cc ct pu
Line 3      setsh 19
Line 4      settc 0
Line 5      loadtext "0.txt
Line 6      settc 0
Line 7      loadtext "asia.txt
Line 8      top delete
Line 9      loadpic "asia.pic
Line 10     make "country "China
Line 11     type :country
Line 12     make "color 1
Line 13     tell 1 pu
Line 14     setpos[95 -88]
Line 15     setsh 22 st
Line 16     tell 0 cc
Line 17     move.menu
Line 18     end

```

Loading "0.txt" in line 5, using colour 0 has to be done first so that the computer will then load the next lot of next in the same colour. Colour 0 has been chosen because it is clear on the monochrome screen. "0.txt" contains only a "return" and is deleted in line 8, after the "asia.txt" is loaded in line 7. The asia.txt is what you see in figure 3. Line 10 creates a variable, country which will change according to which character is under the turtle at the time. (see procedure "fly".) The colour is then changed back to 1 so the user can see what is written. Line 14 sets the shape of the turtle and in this case it is an arrow, and the position which is on China. We then move onto the next procedure "move.menu" which allows the user to move the turtle around the screen

```

to move.menu
  if key? [menu!]
    setc :color
    make "color :color + 1
    repeat 5 [if key?[menu!] wait 1]
    if :color = 5 [make "color 1]
    cc
    type :country
    move.menu
end

```

Warwick allows the colour of the turtle to change colour while it is not actually moving around the screen. In line 1 if a key is pressed, the computer will move onto the next procedure which is "Menu?"

```

line 1      to menu1
line 2      cc type :country
line 3      name readchar "move
line 4      if :move = char 27 [make "country "Exit]
line 5      if :move = "H [seth 0 fd 5 under]
line 6      if :move = "P [seth 180 fd 5 under]
line 7      if :move = "M [seth 90 fd 5 under]
line 8      if :move = "K [seth 270 fd 5 under]
line 9      if :move = "█ [back.menu]
line 10     if :move = char 13 [enter.menu]
line 11     if not letter? :move [stop]
line 12     make "country (word :country :move)
line 13     cc type :country
line 14     before?
line 15     end

```

This procedure checks for particular keys to be pressed.

```

Line 4      ESC key
Line 5      up arrow key
Line 6      down arrow key
Line 7      right arrow key
Line 8      left arrow key
Line 9      Backspace
Line 10     Enter

```

line 13 Depending upon the character under the turtle, a country will be shown in the command centre and then the computer will wait for another key to be pressed.

```

to letter? :letter
  op member? :letter[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

```

If the user chooses to types the choice in the command centre, the computer will output the key pressed and show it in the command centre.

```

to back.menu
  if empty? :country [stop]
  cc
  make "country bl :country
  type :country
end

```

The "Back.menu" is included because the user can also type in the command centre which country or action from the menu they wish to pursue. The boys have included a backspace so if a mistake is made when typing, the user can use the backspace to delete the mistakes.


```

to under
  if charunder = "$ [make "country "Currency]
  if charunder = "u [make "country "USSR]
  if charunder = "m [make "country "Mongolia]
  if charunder = "c [make "country "China]
  if charunder = "z [make "country "|Time Zone|]
  if charunder = "e [make "country "Indonesia]
  if charunder = "s [make "country "|Sri Lanka|]
  if charunder = "t [make "country "Thailand]
  if charunder = "l [make "Country "Laos]
  if charunder = "v [make "country "Vietnam]
  if charunder = "p [make "country "Nepal]
  if charunder = "b [make "country "Burma]
  if charunder = "a [make "country "Afghanistan]
  if charunder = "k [make "country "Pakistan]
  if charunder = "@" [make "country "]
  if charunder = "h [make "country "|South Korea|]
  if charunder = "l [make "country "Khmer]
  if charunder = "n [make "country "|North Korea|]
  if charunder = "i [make "country "India]
  if charunder = "j [make "Country "Japan]
  if charunder = "~ [make "country "|Word Search|]
  if charunder = "?" [make "country "Help]
  if charunder = "x [make "country "Exit]
  if charunder = "f [make "country "Flag]
  if charunder = "o [make "country "Malaysia]
  if charunder = "g [make "country "Bangladesh]
  if charunder = "2 [make "country "Bhutan]
  if charunder = "3 [make "Country "Taiwan]
  if charunder = "4 [make "country "Philippines]
  if charunder = "5 [make "country "Singapore]
end

```

The "menu" procedure assigns country a character. When the turtle is under a letter it recognises it will then make the variable "country" the appropriate name.

```

line 1      to enter.menu
line 2      if member? :country [Exit Flag Help |Word Search| |Time Zone|
            currency] [run (se :country ")]

line 3      if member? :country [Singapore Philippines Taiwan Bhutan
            Bangladesh Japan India |North Korea| Khmer |South Korea|
            Pakistan Afghanistan Burma Nepal Vietnam Laos Thailand |Sri
            Lanka| Malaysia Indonesia China Mongolia USSR] [fly
            start.get.info :country stop]
            end

```

The ":enter.menu" will be invoked when the enter key is pressed. The computer will check which country is the :country variable, check to see if it is a member of the list of countries available. There are two lists here. Line 2 lists the menu type actions such as Exit, Flags, Help, Word search, time zones and currency. If the character is matched to one of these it will invoke the procedures which work that part of the program. Line 3 checks to see if you have the turtle over a specific country. If it is then the computer will search for the information on that country and display it.

to fly

```

if :country = "USSR [if not charunder = "u[setpos[-15 40]]]
if :country = "Mongolia [if not charunder = "m[setpos[-15 10]]]
if :country = "China [if not charunder = "c[setpos[-15 -15]]]
if :country = "Indonesia [if not charunder = "e[setpos[20 -75]]]
if :country = "|Sri Lanka| [if not charunder = "s[setpos[-45 -55]]]
if :country = "Thailand [if not charunder = "t[setpos[-15 -45]]]
if :country = "Laos [if not charunder = "l[setpos[-13 -40]]]
if :country = "Vietnam [if not charunder = "v[setpos[-10 -37]]]
if :country = "Nepal [if not charunder = "p[setpos[-45 -26]]]
if :country = "Burma [if not charunder = "b[setpos[-25 -35]]]
if :country = "Afghanistan [if not charunder = "a[setpos[-70 -15]]]
if :country = "Pakistan [if not charunder = "k[setpos[-60 -20]]]
if :country = "|South Korea| [if not charunder = "h[setpos[26 -10]]]
if :country = "Khmer [if not charunder = "l[setpos[-9 -50]]]
if :country = "|North Korea| [if not charunder = "n[setpos[25 -5]]]
if :country = "India [if not charunder = "i[setpos[-55 -30]]]
if :country = "Japan [if not charunder = "j[setpos[45 -10]]]
if :country = "Malaysia [if not charunder = "o[setpos[4 -65]]]
if :country = "Bangladesh [if not charunder = "g[setpos[-33 -34]]]
if :country = "Bhutan [if not charunder = "2[setpos[-33 -29]]]
if :country = "Taiwan [if not charunder = "3[setpos[17 -34]]]
if :country = "Philippines [if not charunder = "4[setpos[17 -49]]]
if :country = "Singapore [if not charunder = "5[setpos[-15 -60]]]
make "pos pos
tell 1
seth towards :pos
repeat distance :pos [fd 1 if key? [stop] wait 1]
tone 1000 5
end

```

This is a little extra animation. When a country is selected, a plane flies to that country selected.

(III) Searching

TO BE DONE

Summary

Most problems encountered, such as lengthy instructions, crowded screens, difficulties with selecting topics and difficulties with accessing information, have all been overcome by improving our programming skills.

You can give children 101 tools with which to work before you start programming but they will not know the true usefulness of them until they have the need to use something similar. They then begin to see the need to enhance their ideas. The key is to be patient and introduce each tool as it is needed or discovered by the children. It seemed that, when the children were given a few tools with which to work, they seemed to work harder. We were always searching for solutions to problems, and I think this is what made programming databases exciting.

Developing programming skills is not the main focus, nor is it the developing of a database. It is the actual research skills and the knowledge gained as the children develop the other two areas.

We have come a long way from the first attempts of programming. Although we still have not reached a "true" database form, the most important element is that the children have gained a great deal of knowledge, developed academic skills (such as researching, note taking and communicative writing skills) acquired social skills and developed thinking processes, all of which are objectives of the current Queensland Social Studies Syllabus.



The Screen as Data

The fact that the screen can be used to store information, rather than use assignment (ie. MAKE) is one of LogoWriter's neat features, and used to advantage by Warwick Mitchell.

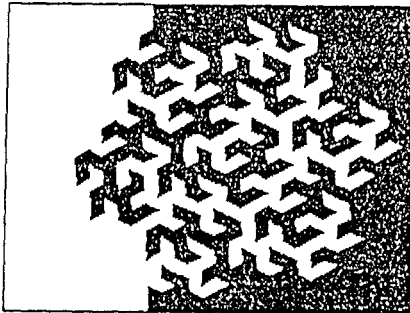
There are a couple of other ways of setting it up, for instance, the invisible text can be part of a page, along with the graphics and procedures, instead of being loaded separately. To make such a page, put the text in place visibly, then `top select bottom settc 0` will hide it before the page is saved. Be aware that `getpage` will be slower than separately loading graphics and text.

For loading transparent text separately we can avoid the extra file by simply printing the blank line:

```

ct                ; make sure screen is clear
pr "              ; print a blank line
settc 0           ; set transparent text colour
loadtext "whatever ; load the text
top delete        ; delete the carriage return at the top
  
```

There's one catch: this doesn't work on the Mac.



A Gosper FlowSnake
(see next page.)



Computing at Entropy



Many thanks to Pam Gibbons who kindly sent a C-64 LogoWriter disk to Oakbank Area School. Next problem is the difference between the graphics of North American and Australian C-64s. Anyone know the fix?

POALL has managed to score a minor victory over the bureacrazy. The bank account 'has been approved and certified as exempt in accordance with exemption 4 Second Schedule Stamp Duties Act, 1923.'

Try these:

- ' (a) In which year was the first spreadsheet developed?
- (b) What was the name of this first spreadsheet?
- (c) By whom was this first spreadsheet developed?
- (d) For which brand of computer was this first spreadsheet designed?

From a trivia game? No, from an information technology workbook. Profound. Anyone remember the TI-99/4A?

Spent a week last term writing a sample program for the SACE Computing Studies book, and found it an irritating and frustrating business trying to formalise what I want to be spontaneous and open-ended. I only hope the formality doesn't extend to Stage-1 etc.

SACE is already distorting the language. Compare these dictionary definitions...

approve: confirm, sanction, pronounce or consider satisfactory or good, agree to

endorse: confirm, declare one's approval of

...with SACE parlance:

endorse: 'I've seen this [SACE Stage 1 Program] but haven't a clue what it's about, be it on your own head.'

This year's *Scientific American* special, the September issue, is entitled 'Communications, Computers and Networks.' The latest ideas by Larry Tesler, Vinton Cerf, Nicholas Negroponte, *et al.* There's a vintage piece, 'Computers, Networks and Education' by Alan Kay. Buy a copy.

Did you see the program 'Signs of Life' in the *Discovery* series on ABC-TV on November 14th? If you didn't, track down someone who taped it. Artificial life, cellular automata (Life), emergent behaviour and that sort of thing. Featured were W. Danny Hillis (designer of the Connection Machine) and Bill Gosper (the FlowSnake fractal). Try it all for yourself with The Phantom Fishtank (Apple or MS-DOS) by one-time fellow student of Hillis, Brian Silverman.

The Bug Stops Here

The SACE *Developing a Stage 1 Computing Teaching Program* book contains at least one error in a Logo procedure, and there's another that can be cleaned up. On page 136 there's a line missing from Lead. It should be:

```
TO Lead :speed
IF KEY? [Command]
FD :speed
Lead :speed
```

(If you'd like a copy of the procedures from pages 135..142 of the book, in Mac, ProDOS or MS-DOS format, send a blank disk to POALL with SAE.)

At the bottom of page 56 you can shorten the line in Questions by changing it to: RUN ITEM :choice {[No_Turn] [No_Turn] [Stalls] [uns_stalls]

[Spark_Plugs] [Finish STOP]}

Questions

END