programming for education

on the commodore 64

a handbook for primary education

john scriven and patrick hall

First published 1984 by:
Sunshine Books (an imprint of Scot Press Ltd.)
12–13 Little Newport Street,
London WC2R 3LD

British Library Cataloguing in Publication Data

Scriven, John
Programming for education on the Commodore 64.
1. Education, Elementary——Data processing
2. Commodore 64 (Computer)
I. Title II. Hall, Patrick
372.1'67 LB1028.43

ISBN 0 946408 27 0

Cover design by Grad Graphic Design Ltd.
Illustration by Stuart Hughes.
Typeset and printed in England by Commercial Colour Press, London E7.

CONTENTS

Contents in detail

Program Notes

All Commodore computers use special characters to control various functions on the screen. For instance, pressing the [CLR/HOME] key and [SHIFT] will clear the screen. If those keys are pressed after a double quote mark, an inverse heart will appear in the listing.

Although this is often a quick way to control print colours, inverse characters and cursor control, there are usually less confusing ways of achieving the same effect. To avoid confusion in the listings, we have used these codes as infrequently as possible. The exception is the use of SHIFTed CLR/HOME to clear the screen as mentioned above.

To make the listings clearer, where this character appears, it has been substituted by [SHIFT/CLR/HOME].

Note: Please be careful when entering programs listed in this book. The quality of printing from a computer printer means that some characters can look very similar — commas, in particular, can look very like full stops.

In the program listings, ↑ appears as ^.

Introduction

The aim of this book was not originally to teach you BASIC. We had hoped to let it contain educational programs which we ourselves had found useful, and merely write about their use, as we have with the other books in this series. As we were collecting material, however, we began to realise that many people who used this book might have had little experience of computers. For those that are new to computers, we have spent more time introducing the early programs.

We hope to lead you step by step through the initial stages to simple programming skills. These are combined to produce short programs that show how the Commodore 64 can be used at home or at school to make learning more interesting, while also discovering something about the computer itself.

The 64 is a very advanced microcomputer and has some outstanding features. Unfortunately, these are not accessible from BASIC without addressing the memory directly, and this makes some aspects of the 64 more obscure than they need to be.

We will attempt to show you that these drawbacks can be overcome, and that you will be able to write interesting educational programs once you have understood a little more about how the 64 operates. If something is not clear straight away, come back to it at a later stage and with luck you will understand it.

Above all, this is a book about doing. No one ever learned about computing simply by reading a book, so do try out the examples and alter things as you find necessary.

If you have some experience with computers, you can skip some of the introductory material and dive into the meatier sections. We hope that you won't be content simply to copy out the listings, but will modify and adapt the programs and routines to suit your own needs.

All the programs have been tried and tested, and are printed from the actual listings, so they should work first time. If they don't, it's probably due to a tiny typing error that's escaped your notice. Check the listing against the book, as even a missing semicolon can stop the program. If, in spite of this, you still find something doesn't work as expected, we'd like to know about it!

Ultimately, we hope that you will continue to enjoy your computing and that this book will help you achieve your aims.

CHAPTER 1
Starting with Maths

Starting off — the computer as calculator — choices and loops — programs and randomness — Table 1: a table test — improvements — Table 2: introducing time — using graphics and sound — looking for patterns — Table Patterns: changing the display — Bingo: a game setting for a traditional idea — repetitive learning — moving graphics to add interest — Factor: a program to teach factorisation.

Ten years ago, it was unusual to find calculators in schools, let alone computers. The advent of the home computer, and computers in schools, brings with it the responsibility to make sure a) that the users are familiar with them and b) that they are used wisely.

In this chapter we shall be looking at how computers use information, and how they can display this. If you have some knowledge of computers, then you can skip over the early sections. If you are coming to the 64 as your first machine, then it should clear up some of the mysteries of how computers can make simple decisions and appear to act intelligently.

If you have used a calculator, then you know that you enter data, in the form of numbers, along with operators, like plus or minus, to tell the calculator what to do with those numbers. For instance,

2+2−1=

At each stage of the calculation, the display shows the last number entered or the result at that stage. You can use the 64 as a calculator in the same way. Instead of calculator buttons, you can communicate with the 64 via the keyboard. If you enter

2+2−1 [RETURN]

it will calculate the result. (The [RETURN] key can be seen as sending the information you have entered inside the computer.) The problem is that, whereas you see the result automatically on a calculator, the 64 has to be told to display the answer. To speak to the computer, you have to use the language that it understands. The 64 uses the language BASIC, which is a

fairly small set of words that are quite close to normal English. These words are entered via the keyboard, just as on a normal typewriter.

To make the 64 print something, you must use the PRINT statement. If you enter

PRINT 2 + 2 – 1 [RETURN]

then the result, 3, will appear on the screen.

PRINT can be used to write words on the screen in just the same way. If you enter

PRINT "YOGI BEAR" [RETURN]

then the words YOGI BEAR will appear on the screen.

This is using the computer in immediate mode, which is handy for instant calculations, but is hardly using it in the manner for which it was intended.

The most useful point about computers is that they can store the instructions given to them and follow them in the correct order.

So that they can follow the instructions in the correct order, the instructions are given line numbers. First make sure there's nothing already in the 64's memory by typing NEW [RETURN]. To perform the simple example earlier, you could use the following:

10 PRINT "TWO PLUS TWO TAKE ONE EQUALS"; [RETURN]
20 PRINT 2 + 2 – 1 [RETURN]

If you enter the lines as they are written, the program lines are stored in the computer's memory.

(*Important:* Make sure that you enter the lines exactly as they are written. Humans can understand things even when there are mistakes, but computers are unforgiving and even putting a letter l instead of a figure 1, or a letter O instead of a figure 0, will prevent program instructions from working.)

To execute the instructions, the command RUN must be entered and the [RETURN] key pressed. The computer will obey the instructions in each line in turn, that is, print the sentence in quotes in line 10 followed by the answer to the calculation in line 20. As there are no more lines, the program will end there, and pass control back to you. (From now on remember to press the [RETURN] key at the end of program lines and after commands. There won't be any more reminders!)

Programs, then, are simply lists of instructions that the computer obeys in a particular order. It doesn't matter in which order you enter the lines, the computer sorts them out for you. So if you now enter

```
5 PRINT "THIS IS MY PROGRAM"
```

it will be executed first. RUN the program to test this.

If you use the command LIST, the program you have been entering will appear on the screen in the correct order. To get rid of one particular line, enter just the line number. So, entering

```
5 [RETURN]
```

followed by LIST, will show you that line 5 has indeed been deleted.

To delete a whole program from memory, use the command NEW. This clears the complete BASIC program, so don't use it unless you're sure that's what you really want.

Try it now and check that your program really has disappeared by LISTing it.

Inputs and variables

As well as entering information in program lines, you can enter it via the keyboard during the course of a program. The 64 stores this information in the form of variables. This is just a way of saying that letters can stand for various values, like in algebra. If you say $X + 6 = 10$, then X must have the value 4. In the language BASIC, you can assign a value to a letter by simply saying $X = 4$.

Enter the following

```
X=4
```

Now, as a direct statement, enter

```
PRINT X
```

The value 4 will be printed on the screen.

X is known as a variable. If the variable is going to stand for something in particular, then it's a good idea to use a longer name, for instance, the variable to hold a year could be YEAR = 1984.

The 64 will only recognise the first two letters of the variable, so YEAR and YELLOW appear the same to the computer. If you enter

```
YEAR = 1984
```

then

```
YELLOW = 6
```

then

```
PRINT YEAR
```

the result 6 will appear, so be careful how you use variable names.

If you want to store a string of characters, ie letters, numbers, graphic characters, or anything else on the keyboard, you have to use what is called a string variable. This is just like a numeric variable except that it ends in a $.

Enter this:

```
A$ = "CBM 64"
PRINT A$
```

The string variable A$ now contains the string of characters CBM 64, so when you tell it to print out A$, that is what is printed.

Just as it is a good idea to use meaningful names for numeric variables, string variables too are easier to understand when they mean something. For example, NAME$ is more meaningful than N$.

Remember that the 64 will confuse NAME$ with NAIL$, as it only recognises the first two characters. It is also important that the variable name should not contain a BASIC keyword. This means that FORT is not allowed — it contains the BASIC word FOR. The variable BRIGHTON$ is not allowed as it contains the word TO.

As was mentioned earlier, variables are useful when you want to enter information during the course of a program.

```
10 PRINT "ENTER YOUR NAME"
20 INPUT NAME$
30 PRINT "HELLO, "NAME$
40 PRINT "ENTER YOUR AGE"
50 INPUT AGE
60 PRINT "MY! YOU DON'T LOOK  "AGE;NAME$
```

This program, when RUN, will tell you to enter your name at line 10. In line 20, it will stop program execution until it receives an input, followed by a press on the [RETURN] key. Whatever is entered in this line becomes known as NAME$, and will be printed out in line 30, after the word HELLO.

Line 40 will print the request for the age on the screen, and this will be stored as variable AGE in line 50. Note that you must enter this in figures or the computer will tell you to REDO the input. Line 60 will print out the text in that line plus the name and age that were entered. The semicolon between AGE and NAME$ is important to separate them.

To save entering RUN again and again, it is possible to send program control back to the beginning. To do this, add the program line

```
70 GOTO 10
```

This will cause the program to go into a never-ending loop. To get out of this, it is necessary to press the RESTORE key while holding down the RUN/STOP key at the same time.

If it's at all possible, avoid using GOTOs in your programs. Although in Commodore BASIC you have to use them sometimes, they can make the structure difficult to understand.

Some programs

Now that you can use variables and inputs, you can start to use your 64 to write simple programs, and also make them appear more friendly. Note that, in line 10, the INPUT statement can include words that are printed.

```
10 INPUT "WHAT IS YOUR NAME";NAME$
20 INPUT "WHAT IS THIS YEAR IN FIGURES";YEAR
30 INPUT "WHAT IS THE YEAR OF YOUR BIRTH";BIRTHYEAR
40 INPUT "HOW OLD ARE YOU";AGE
50 IF AGE = YEAR - BIRTHYEAR THEN PRINT "I'LL BELIEVE
   YOU":STOP
60 IF AGE = YEAR - BIRTHYEAR - 1 THEN PRINT "I'LL
   BELIEVE YOU":STOP
70 PRINT "I DON'T BELIEVE YOU"
```

The semicolons in INPUT statements are essential to avoid a SYNTAX ERROR message. Notice too that you don't need to put a question mark in; this is automatic when the 64 expects an input. You should understand this program down as far as lines 50 and 60. In these lines, the computer checks to see if the entered age, AGE, is possible. Line 50 is for those who have already had their birthdays this year, and line 60 is for those whose birthdays are yet to come. IF the line is true, then a message is printed out, and the program stops.

Note the use of a colon to separate two statements on the same line. If neither line 50 nor line 60 is true, the program continues to line 70, where the final message is printed out, and the program stops as there are no more lines left to execute.

The IF...THEN statement is the first statement that you have come across that gives your 64 some intelligence. At last it can make decisions according to different circumstances. You can imagine almost any human activity being governed by IF...THEN rules.

IF the alarm goes THEN get up.
IF it's raining THEN go back to bed.

These statements are very useful in a program, as they can test for correct answers to problems posed by the computer.

```
10 Question
20 Answer
30 IF Answer = right THEN PRINT "OK"
40 IF Answer = wrong THEN PRINT "SORRY"
50 Next Question
60 Next Answer
70 IF Answer = . . .
etc.
```

Loops

In the last program, if you had wanted ten questions, you would have had to repeat the same structure as in the first four lines ten times. As well as being wasteful on memory, this would take a long time to enter. Luckily, you can employ a loop to take you through the same section of program a certain number of times. The only loop that the 64 possesses is the FOR . . . TO . . . NEXT loop. Although it looks complicated, it's really quite a simple idea.

You need to set up a counter in one program line at the beginning of the loop, and to have a marker line to show the end of the loop. This example should explain it more clearly. First type NEW to get rid of the last program.

```
10 PRINT "LOOP STARTS NOW"
20 FOR N = 1 TO 10
30 PRINT "LOOP NUMBER "N
40 NEXT N
50 PRINT "LOOP FINISHED NOW"
```

When the program is RUN, line 10 simply prints a message. Line 20 means 'Set a counter up called N. Start counting from 1 until you reach 10.' Line 30 prints out the number of the loop, which is 1. Line 40 means 'Increase N by one and go back to the start of the loop.' N is therefore now 2. Program control goes back to the start of the loop in line 20; line 30 prints out LOOP NUMBER 2, N is increased once again, and so on. At the end of the tenth time through the loop, N becomes 11. When the program returns to line 20, the program says 'The loop is only supposed to go up to 10, so program

control must jump straight to the first line after the end of the loop', ie line 50, and the final message is printed out.

If you enter this short program, and RUN it, you can see this in operation. After it has stopped, enter PRINT N, and you will see that it does indeed contain the value 11.

When a program is RUN, all the numeric variables are set to zero, and the string variables emptied, so you can run the program again and again.

Random numbers

If you want the computer to ask specific maths questions, you now have all the knowledge you need. It's far more useful to have it select the questions out of a hat, however, and there is a very useful facility to let it do just that.

If you enter PRINT RND(0), then a number between zero and one will be printed. This number is chosen at random, and the value can be assigned to a variable.

```
10 FOR N = 1 TO 10
20 X = RND(0)
30 PRINT X
40 NEXT N
```

This program will print out ten random numbers between zero and ten. Numbers this small may not be much use in simple number problems, so line 20 could be changed to

```
20 X = RND(0)*100
```

Note: Computers use an asterisk (*) for multiply so that this won't be confused with the letter X. The divide sign is an oblique line (/).

Line 20 will now generate numbers between zero and 100. If you RUN this now, you will see that the numbers are so random that they have several decimal places. To lose the decimal part of the number, change line 20 to

```
20 X = INT(RND(0)*100)
```

This will generate integers, or whole numbers between zero and ninety-nine.

At this stage you know how to generate numbers at random, how to loop ten times to produce ten questions, how to input answers and how to check that the answers are correct. You can combine all these points in a program designed to test knowledge of one aspect of maths.

Table 1

```
10 REM
20 REM ***  TABLE TEST  ***
30 REM
100 SC=0
110 PRINT"[SHIFT/CLR/HOME]"
120 PRINT TAB(12)"TABLE TEST"
130 FOR N = 1 TO 10
140 A = INT(RND(0)*13)
150 B = INT(RND(0)*13)
160 PRINT TAB(4-N*.11);N;".)"A" X "B TAB(24)"=";
170 INPUT G
180 IF G = A*B THEN PRINT TAB(30)"OK"
190 IF G = A*B THEN SC=SC+1
200 IF G <>A*BTHEN PRINT TAB(30)"NO- ";A*B
210 NEXT N
220 PRINT"YOU SCORED ";SC
230 PRINT"PRESS SPACE BAR TO GO"
240 GET Z$
250 IF Z$="" THEN GOTO 240
260 GOTO 100

READY.
```

Commentary

Line 100 sets the variable SC, which holds the score, to zero.

Line 120 uses the TAB statement. This moves the print position across the screen.

Lines 140 and 150 choose two numbers between 0 and 12 at random.

Line 160 prints the question on the screen. Lines 170 to 200 accept an input and check it against the correct answer.

Line 190 increments the score by one. Lines 220 to 260 print the score and allow the user to go again.

In the past, computers have always been associated with mathematics and, even nowadays, computers tend to be used more in the maths and science departments of schools. In fact, computers are good at operating with any kind of symbols, not just numbers, and it would be a pity if their 'number-

crunching' capability were the only thing they were used for in the educational sphere.

In spite of this, maths is an area where a simple introduction can be made. This is also an area where it is easy to write programs that test recall without being truly educational. It has been shown by many educational psychologists since the war, such as Jean Piaget and Jerome Bruner, that children develop by passing through distinct learning stages. They cannot learn automatically, but need much practical experience before being able to manipulate symbols in their minds and finally achieve a stage of logical, formal thought. The computer in its present form cannot replace practical experience, but its graphic capabilities can help tremendously where the child begins to learn to use symbols or concrete images rather than the objects themselves.

We have tried to avoid as far as possible the type of program that uses a computer to do what can be just as easily achieved using traditional materials. Computers are expensive artefacts to have around without careful thought being given to their most profitable use, whether at home or at school.

You may think that we have spent some time on showing you how to write a program that we are now tearing to pieces. The traditional table test is, unfortunately, an example of a weak, drill-type program. The reason it is included here is that it demonstrates simple principles of program design and is easy to understand.

If this was the only sort of program you were ever going to write, we would consider that we had failed in our aim to introduce a more dynamic approach to using computers in education. It is true that it merely tests recall, without reinforcing correct answers, diagnosing problems or attempting to offer help. It can, however, be used with caution, provided these shortcomings are realised.

Instead of leaving you with the simple version of the program, here is a listing that attempts to go one step further. It produces the numbers at random, but starts a time at the first question, so the user has a target to aim at, in trying to better his or her previous time. Also included is a facility for the teacher or parent to check the results of each program run and to compare the times taken.

Table 2

```
10 REM ** J. SCRIVEN **
20 REM
30 REM ** 10/10/82 **
40 REM
50 REM ** TIMED TABLE TEST **
60 REM
70 REM ** INITIALIZATION **
```

```
80 REM
85 REM
90 DIM NA$(30)
100 DIM SC(30)
110 DIM TM(30)
120 NU = 1
130 FL = 0
140 REM
150 REM ** TEST LOOP **
160 REM
170 FOR NU = 1 TO 30
180 PRINT"[SHIFT/CLR/HOME]"
190 PRINT TAB(12)"TABLE TEST"
200 PRINT TAB(10)"ENTER YOUR NAME"
210 INPUT NA$(NU)
220 PRINT "OK, ";NA$(NU)
230 PRINT "PRESS SPACE BAR TO BEGIN"
240 GET Z$
250 IFZ$="" THEN 240
260 SC(NU)=0
270 TS=TI
280 PRINT"[SHIFT/CLR/HOME]"
290 PRINT TAB(12)"TABLE TEST"
300 PRINT TAB(1)NA$(NU)
310 FOR N = 1 TO 10
320 A = INT(RND(0)*13)
330 B = INT(RND(0)*13)
340 PRINT TAB(4-N*.11);N;".)"A" X "B TAB(24)"=";
350 INPUT G
360 IF G = A*B THEN PRINT TAB(30)"OK"
370 IF G = A*B THEN SC(NU)=SC(NU)+1
380 IF G <> A*B THEN PRINT TAB(30)"NO- ";A*B
390 NEXT N
400 TT=INT((TI-TS)/60)
410 PRINTNA$(NU);" SCORED ";SC(NU);
420 PRINT" OUT OF 10 IN ";TT;" SECS"
430 TM(NU)=TT
440 PRINT"PRESS SPACE BAR FOR ANOTHER GO";
450 GET Z$
460 IF Z$="" THEN GOTO 450
470 IF Z$="&" THEN GOTO 1000
480 NEXT NU
490 PRINT"CLASS COMPLETED"
500 FOR M = 1 TO 2000: NEXT M
1000 REM
1010 REM ** RESULTS ROUTINE **
1020 REM
1030 PRINT"[SHIFT/CLR/HOME]"
1040 PRINT TAB(15)"RESULTS"
```

```
1050 PRINT
1055 PRINT
1060 PRINT TAB(5)"NAME";TAB(24)"SCORE";TAB(34)
     "TIME"
1070 PRINT
1080 FOR NU = 1 TO 30
1090 PRINTNU;TAB(5)NA$(NU);TAB(24)SC(NU);TAB(34)
     TM(NU)
1100 IF NU = 15 THEN PRINT"PRESS SPACE BAR TO
     CONTINUE"
1110 IF NU = 15 THEN GET Z$
1120 IF Z$ = "" THEN 1110
1130 NEXT NU

READY.
```

Commentary

One exercise, often given to children learning tables, is to discover the patterns that the answers make when these numbers are shaded in on a grid of figures. It is usual to make the grid 10 × 10 or 12 × 12, and this idea forms the basis of the next program.

There are many ways in which the 64 could be told to draw a grid of numbers on the screen, such as printing each line separately. A neater way, that uses more of the computer's in-built functions, would be to put the print routine in a loop. If you simply entered

FOR N = 1 TO 200:PRINT N:NEXT N

the numbers would certainly appear, but they would not be formatted in neat lines with units and tens underneath each other. If you consider the screen layout on the 64, you will see that the numbers could fit in very regularly. The screen is 40 characters in width, and the largest number takes three figures (2,0 and 0). If a space is left between each column, you can see that ten numbers, each one preceded by a space, will exactly fit on each line.

HTU HTU HTU HTU HTU HTU etc.

If we could tell the 64 to PRINT AT a particular row and column position, it would be easy. Unfortunately, such a command does not exist in Commodore BASIC. Luckily for us, however, there is such a routine buried inside the operating system. Near the top of the memory lies the kernal. This is a table of routines that jump to different parts of the memory to perform particular functions. The one we are interested in is called PLOT and lies at location 65520. All you need do is load two regis-

ters with the row and column values by POKEing, and then call the kernal routine PLOT. You can do this in one line:

```
POKE 782,X:POKE 781,Y:SYS 65520:RETURN
```

and GOSUB this routine just before you print something. The new print position will be at the X and Y coordinates you have just entered. This is a very useful routine and you will find it occurring quite frequently in these programs. In the next program, it is used to move the print position across to each new column and row.

When the numbers have been printed, the user is asked to enter the table that is required. So that these numbers show up, the next routine changes the colour of just these numbers to white. Again, this is not a very complex task. The 64 stores the screen text in one set of locations, and the colour in which the text appears in another. The second set of locations starts at 55296. As the grid of numbers is spaced evenly, starting at every fourth location, the colour value can be POKEd in another FOR...NEXT loop.

How can you tell it to act only on the numbers in the particular table chosen? A useful trick is to test if there is a remainder after division by the table number. If this seems difficult to grasp, try this program:

```
10 INPUT A
20 FOR N=1 TO 100
30 IF INT(N/A)=N/A THEN PRINT A
40 NEXT N
```

This runs through the numbers from 1 to 100, and divides each one by the number you input. If the number has a remainder, the INT function will round down to the nearest whole number below, eg INT(11.5)=11. It will only print those numbers that don't have a remainder, as the INTeger of 12 is 12.

Notice that this program tests the codes of keys that are pressed in GET statements. If any key other than the numbers 1 to 9 is pressed, the program loops back to ask for the key to be pressed again.

Table Patterns

```
10 REM ***  TABLE PATTERNS  ***
20 REM
30 REM ***  A.J.S. 4/10/83  ***
40 REM
50 REM
60 REM ***  CLEAR SCREEN + TITLE ***
```

```
70 REM
80 PRINT"[SHIFT/CLR/HOME]"
90 X=0:Y=0:GOSUB540
100 PRINT"TABLE PATTERNS"
110 X=2:Y=2
120 REM
130 REM *** LOOP TO PRINT NUMBERS ***
140 REM
150 FOR N = 1 TO 200
160 IFN=10 OR N=100 THEN X=X-1
170 GOSUB 540:PRINTN
180 X = X + 4
190 IF INT(N/10)=N/10 THEN X=1
200 IF INT(N/10)=N/10 THEN Y=Y+1
210 IF INT(N/10)=N/10 AND N>99 THEN X=0
220 NEXT N
230 REM
240 REM *** SELECT TABLE ROUTINE ***
250 REM
260 POKE198,0
270 X=5:Y=23:GOSUB540
280 PRINT"*PRESS THE TABLE NUMBER*"
290 GET A$:IF A$ = "" THEN 290
300 A=VAL(A$)
310 IFA<1 THEN 260
320 X=30:Y=0:GOSUB540:PRINT"TABLE"A"   "
330 REM
340 REM ***  CHANGE NUMBER COLOURS ***
350 REM
360 COL = 55296
370 FOR N = 1 TO 200
380 IF INT(N/A) <> N/A THEN 420
390 POKE COL+81+(N-1)*4,1
394 REM
400 POKE COL+81+(N-1)*4+1,1
410 POKE COL+81+(N-1)*4+2,1
420 NEXT N
430 X=5:Y=23:GOSUB540
440 POKE198,0
450 PRINT"PRESS SPACE BAR TO CLEAR"
460 GET A$:IF A$ = "" THEN 460
470 FOR N = 55296 TO 56319
480 POKE N,14
490 NEXT N
```

```
500 GOTO 260
510 REM
520 REM *** "PRINT AT" SUBROUTINE ***
530 REM
540 POKE782,X:POKE781,Y:SYS65520:RETURN

READY.
```

Commentary

Line 80 clears the screen. Line 90 sets the X and Y coordinates to zero.

Lines 150 to 220 contain the loop that prints the table grid.

Lines 190 to 210 move the print position at the end of the line — when the value of N is exactly divisible by 10.

Lines 260 to 310 allow the number of the table to be input. Line 320 prints the table number at the top of the screen.

Variable COL is the location of colour memory. Lines 370 to 420 change the text colour by POKEing the relevant memory locations. Line 380 skips over the POKEs if the value of N is not exactly divisible by the table chosen.

Line 540 contains the 'PRINT AT' subroutine.

Lines 280 and 450 contain the same number of characters in the print statements so that they can print on top of one another.

Bingo

After the element programs in this chapter, we move on to a couple in the area of maths that extend the use of the 64's features. If you are fairly new to BASIC, you may find the ideas more difficult to grasp. Although they are explained in some detail in the program notes, you may prefer simply to type in the listings. As your skills improve in the following chapters, you can come back to these programs at a later stage.

Bingo is an attempt to move beyond the simple approach of answering questions on screen. The child is presented with a game format, and, although the questions still only test recall of previously-learned know-

ledge, there is a reward motive in completing the whole card. When this stage is reached, there is a visual display.

There is always some problem over how many of a computer's facilities need to be incorporated in every program. Just because Commodore computers support voice synthesis, high-resolution graphics and music in three-part harmony, it does not mean that a good program has to possess all of these features, or indeed any of them. A happy medium has to be reached at the planning stage. Some programs benefit from using specialist features, but usually take more time to program. In some cases, their implementation can be irritating and work contrary to the expectation of the teacher or parent.

One example is the addition of loud sound effects, which can be disturbing in a classroom if there are other children engaged in quiet activities. If this is the case, removal of the offending sound commands will prove an answer. An alternative, of course, is to reduce the volume by use of the TV volume control! Flashing displays should usually be avoided. Where sections of the screen are required to flash, this is done to draw attention to part of the display. Having had personal experience of a child suffering a *petit mal* attack due to the program loading display on a cheaper computer, we are particularly aware of this possibility. Also, overuse of this feature is likely to prove distracting in many cases.

```
10 REM BINGO - PAT HALL, JAN '84
20 REM
30 GOSUB 110 : REM INITIALISATION
40 GOSUB 330 : REM SET UP
50 GOSUB 770 : REM TEST
60 GOSUB 1150 : REM REWARD
70 GOTO 40
80 END
90 REM INITIALISATION
100 REM SET UP ARRAYS
110 DIM A( 20 ) : DIM B( 20 )
120 DIM C( 20 ) : DIM D( 20 )
130 DIM LB( 20 )
140 REM SELECT LOCATIONS FOR BOXES
150 N = 0
160 FOR I = 1144 TO 1180 STEP 9
170 FOR J = 0 TO 600 STEP 200
180 N = N + 1
190 LB( N ) = I + J
200 NEXT J
210 NEXT I
```

```
220 REM INITIALISE GAME AND SCORE
230 G = 0 : S = 0
240 REM WHITE SCREEN
250 POKE 53280, 1 : POKE 53281, 1
260 PRINT ""
270 REM SWITCH ON EXTENDED COLOUR MODE
280 POKE 53265, 91
290 RETURN
300 REM
310 REM SET UP GAME
320 REM YELLOW BACKGROUND
330 POKE 53282, 7
340 REM PUT TITLE ETC. ON SCREEN
350 PRINT CHR$( 144 )
360 X = 14 : Y = 0 : GOSUB 710
370 PRINT "<< BINGO >>"
380 PRINT CHR$( 28 )
390 X = 0 : Y = 1 : GOSUB 710
400 G = G + 1
410 PRINT "GAME "; G
420 X = 31 : GOSUB 710
430 PRINT "SCORE "; S
440 FOR I = 1 TO 20
450 REM INITIALISE FLAG
460 D( I ) = 0
470 REM CALCULATE NUMBER
480 A( I ) = INT( RND( 1 ) * 8 ) + 2
490 B( I ) = INT( RND( 1 ) * 5 ) + 5
500 C( I ) = A( I ) * B( I )
510 REM PLACE BOX ON SCREEN
520 FOR J = 0 TO 3
530 FOR K = 0 TO 80 STEP 40
540 L = LB( I ) + J + K
550 POKE L + 54272, 7 : POKE L, 96
560 NEXT K
570 NEXT J
580 REM PLACE NUMBER IN BOX
590 REM RED FOREGROUND
600 POKE LB( I ) + 54313, 2
610 POKE LB( I ) + 54314, 2
620 E = INT( C( I ) / 10 ) + 112
630 POKE LB( I ) + 41, E
640 E = E - 112
650 E = C( I ) - E * 10 + 112
```

```
660 POKE LB( I ) + 42, E
670 NEXT I
680 RETURN
690 REM
700 REM PRINT TAB ROUTINE
710 POKE 782, X : POKE 781, Y
720 SYS 65520
730 RETURN
740 REM
750 REM TEST ROUTINE
760 REM CHOOSE ONE BOX AT RANDOM
770 T = INT( RND( 1 ) * 20 ) + 1
780 REM CHECK BOX IS STILL ON SCREEN
790 IF D( T ) = 1 THEN 770
800 D( T ) = 1
810 X = 9 : Y = 22 : GOSUB 710
820 PRINT "WHAT IS" A(T) "X" B(T)
830 X = 30 : GOSUB 710
840 PRINT "          "
850 X = 30 : GOSUB 710
860 INPUT AN$ : AN = VAL( AN$ )
870 IF AN <> C( T ) THEN 830
880 REM INCREASE SCORE
890 S = S + 1
900 X = 37 : Y = 1 : GOSUB 710
910 PRINT S
920 REM SOUND EFFECT, VOICE 1
930 POKE 54296, 15 : REM VOLUME
940 POKE 54277, 136 : REM ATT / DEC
950 POKE 54278, 136 : REM SUST / REL
960 POKE 54276, 17 : REM WAVEFORM
970 POKE 54273, 76 : REM HIGH NOTE
980 POKE 54272, 252 : REM LOW NOTE
990 FOR I = 1 TO 100 : NEXT I
1000 POKE 54277, 0
1010 POKE 54278, 0
1020 POKE 54276, 0
1030 REM ERASE BOX
1040 FOR I = 0 TO 3
1050 FOR J = 0 TO 80 STEP 40
1060 L = LB( T ) + I + J
1070 POKE L + 54272, 1 : POKE L, 32
1080 NEXT J
1090 NEXT I
```

```
1100 IF INT(S/20) <> S / 20 THEN 770
1110 FOR I = 1 TO 1000 : NEXT I
1120 RETURN
1130 REM
1140 REM REWARD ROUTINE
1150 PRINT ""
1160 RESTORE
1170 FOR I = 1 TO 115
1180 READ N
1190 FOR J = 0 TO 1
1200 FOR K = 0 TO 40 STEP 40
1210 POKE N + J + K + 54272, 2
1220 POKE N + J + K, 128
1230 NEXT K
1240 NEXT J
1250 NEXT I
1260 FOR I = 1 TO 3000 : NEXT I
1270 PRINT ""
1280 RETURN
1290 REM
1300 REM DATA FOR 'BINGO'
1310 DATA 1024, 1104, 1184, 1264, 1344
1320 DATA 1424, 1504, 1584, 1664, 1744
1330 DATA 1824, 1904, 1026, 1068, 1110
1340 DATA 1190, 1270, 1350, 1388, 1466
1350 DATA 1548, 1590, 1670, 1750, 1830
1360 DATA 1868, 1906, 1034, 1114, 1194
1370 DATA 1274, 1354, 1434, 1514, 1594
1380 DATA 1674, 1754, 1834, 1914, 1038
1390 DATA 1118, 1198, 1278, 1358, 1438
1400 DATA 1518, 1598, 1678, 1758, 1838
1410 DATA 1918, 1080, 1161, 1241, 1321
1420 DATA 1401, 1481, 1561, 1641, 1721
1430 DATA 1801, 1882, 1044, 1124, 1204
1440 DATA 1284, 1364, 1444, 1524, 1604
1450 DATA 1684, 1764, 1844, 1924, 1129
1460 DATA 1209, 1289, 1369, 1449, 1529
1470 DATA 1609, 1689, 1769, 1849, 1051
1480 DATA 1133, 1213, 1612, 1614, 1693
1490 DATA 1773, 1853, 1931, 1138, 1218
1500 DATA 1298, 1378, 1458, 1538, 1618
1510 DATA 1698, 1778, 1858, 1060, 1142
1520 DATA 1222, 1302, 1382, 1462, 1542
1530 DATA 1622, 1702, 1782, 1862, 1940
```

Commentary

The program is controlled by lines 30–80. Line 70 causes the program to continue indefinitely until RUN/STOP is depressed.

Lines 100–290 form the first subroutine of the program, called by line 30 in the control module. The routine initialises various variables needed by Bingo. Five arrays are set up. The A and B arrays hold the twenty pairs of numbers presented to the child and the C array stores the answer for each multiplication. The D array consists of flags, one for each number pair, which ensure that the same question is not repeated. Finally the LB array stores the screen locations of each coloured box the numbers are printed in. The actual value of each member of the LB array is the location of the top lefthand corner of the coloured box. These values are calculated by the two nested FOR...NEXT loops between lines 160–210. Here the values that the I and J loop variables STEP through are arranged so that their sum, I + J, gives the exact location. Note the way in which the variable N is incremented through the loops to identify each box.

This first subroutine also sets the values of S (the score achieved by the child) and G (the number of 'sheets' completed) to zero. Line 250 sets the entire screen area to white and then line 280 switches on the Commodore 64's 'extended' colour mode. This allows text to be printed with a different background colour to the screen background (and is the equivalent of the command COLOUR (colour + 128) in BBC BASIC). Extended colour mode is switched on with POKE 53265, 91 and off with POKE 53265, 27. It allows three more background colours to be set up simultaneously on the screen, but requires a different character set for each.

Lines 310–680 present the bingo sheet. Use is made of the PRINT TAB routine at lines 700–730. This allows an easy way to format the display. X and Y screen coordinates have to be defined before calling the routine and this is done here, for example, at line 360. Lines 480–500 calculate the questions and answers and the boxes are drawn by lines 520–570. The J and K values in the two nested loops increment the value of LB calculated before, and this value is then POKEd to screen and colour map by line 550. Each number is placed in its box by 600–660. Extended colour mode is used and the two-digit numbers have to be separated by the simple arithmetical routine into the individual ten and unit figures. These are POKEd separately to the screen.

Lines 750–1120 present the test. The Tth box is chosen at line 770 and the flag is checked by line 790 to make sure that box and question have not already been chosen. The TAB routine is used again in presenting the question and the child's input is error-trapped. A sound effect is produced and

finally the box itself is erased by lines 1040–1090. Line 1100 tests when the entire screen is cleared.

The reward routine simply places the word 'Bingo' large upon the screen. This is done in a very similar way to the drawing of the boxes but here the location of each 'box' is not calculated but held as lines of DATA which are READ by line 1180. The nested loops between 1190–1240 draw each section of the letters.

Factor

The last program in this chapter, Factor, extends the approach to that of a video game. It does not require high-speed manual dexterity, however, simply the ability to input any factor of a number that appears on the screen. The number is successively factorised by the inputs until it reaches 1, when another number is chosen. If the factor chosen is permissible, a plane flies across the screen and drops a bomb on to the number resulting in an explosion. If the number is incorrect, the bomb misses. There is also an on-screen scoring facility.

A last point needs to be made about the type of reward offered in a program. Although an exciting display may offer positive reinforcement to the child, it is hoped that the reward will become intrinsic, ie that the child enjoys the success of getting a problem right without any external reward. When adults succeed in solving a 'whodunnit', or complete the Times crossword, the achievement alone is enough. This internalisation has taken a long time to achieve, however, and praise, even from a machine, is important!

It is also necessary to point out incorrect responses in a gentle manner, or the child feels despondent and even aggressive. If a computer has ever beaten you at chess, you will know the feeling. Although it may seem obvious to state, the difficulty of the problem has to be within the solving capacity of the child. When designing programs, the incorrect response should not result in a screen display that is more impressive than that for a correct response, or the reinforcement effect will be lost, as the child attempts to fail because the display is prettier!

```
1000 REM FACTOR - PAT HALL, JAN '84
1010 REM
1020 GOSUB 1120 : REM DEFINE SPRITES
1030 GOSUB 1610 : REM STORE PRIMES
1040 GOSUB 1700 : REM SET UP DISPLAY
1050 GOSUB 1880 : REM TEST ROUTINE
```

```
1060 GOSUB 2160 : REM FLY PLANE
1070 GOTO 1050
1080 END
1090 REM
1100 REM DEFINE SPRITES
1110 REM BEGINNING OF 6567 VIDEO CHIP
1120 VC = 53248
1130 REM PLANE, SPRITE 0
1140 REM SET POINTER FOR SPRITE 0
1150 POKE 2040, 13
1160 REM READ DATA FOR SPRITE 0
1170 FOR I = 0 TO 19
1180 POKE 832 + I, 0
1190 NEXT I
1200 FOR I = 20 TO 62
1210 READ N : POKE 832 + I, N
1220 NEXT I
1230 REM BOMB, SPRITE 1
1240 REM SET POINTER FOR SPRITE 1
1250 POKE 2041, 14
1260 REM READ DATA FOR SPRITE 1
1270 FOR I = 0 TO 39
1280 POKE 896 + I, 0
1290 NEXT I
1300 FOR I = 40 TO 62
1310 READ N : POKE 896 + I, N
1320 NEXT I
1330 REM FLASH, SPRITE 2
1340 REM SET POINTER FOR SPRITE 2
1350 POKE 2042, 15
1360 FOR I = 0 TO 39
1370 POKE 960 + I, 0
1380 NEXT I
1390 FOR I = 40 TO 62
1400 READ N : POKE 960 + I, N
1410 NEXT I
1420 REM ENLARGE SPRITES 0 AND 1
1430 REM X-DIRECTION
1440 POKE VC + 29, 5
1450 REM Y-DIRECTION
1460 POKE VC + 23, 5
1470 REM SELECT COLOUR
1480 REM BLUE PLANE
1490 POKE VC + 39, 6
```

```
1500 REM BLACK BOMB
1510 POKE VC + 40, 0
1520 REM BROWN FLASH
1530 POKE VC + 41, 9
1540 REM LOCATE FLASH ON SCREEN
1550 POKE VC + 4, 165 : REM X-COORD
1560 POKE VC + 5, 143 : REM Y-COORD
1570 RETURN
1580 REM
1590 REM STORE PRIMES
1600 REM DIMENSION PRIMES' ARRAY, PN
1610 DIM PN( 29 )
1620 REM READ PRIME NUMBERS FROM DATA
1630 FOR I = 1 TO 29
1640 READ N : PN( I ) = N
1650 NEXT I
1660 RETURN
1670 REM
1680 REM SET UP DISPLAY
1690 REM CYAN BORDER
1700 POKE 53280, 3
1710 REM CYAN BACKGROUND
1720 POKE 53281, 3
1730 REM CLEAR THE SCREEN
1740 PRINT "[SHIFT/CLR/HOME]"
1750 REM DRAW DESERT LANDSCAPE
1760 FOR I = 0 TO 280 STEP 40
1770 FOR J = 0 TO 39
1780 REM POKE COLOUR TO SCREEN
1790 POKE 55976 + I + J, 8
1800 REM POKE CHARACTER TO SCREEN
1810 POKE 1704 + I + J, 160
1820 NEXT J
1830 NEXT I
1840 RETURN
1850 REM
1860 REM TEST ROUTINE
1870 REM SELECT NUMBER
1880 TN = INT( RND( 1 ) * 70 ) + 30
1890 REM CHECK TARGET IS NOT PRIME
1900 N = TN : GOSUB 2080
1910 IF PF = 1 THEN 1880
1920 X = 19 : Y = 16 : GOSUB 2520
1930 PRINT TN
```

```
1940 X = 8 : Y = 0 : GOSUB 2520
1950 PRINT "TYPE A FACTOR OF" TN
1960 X = 30 : GOSUB 2520
1970 PRINT "          "
1980 GOSUB 2520
1990 INPUT F$
2000 IF ASC( F$) < 49 THEN 19600
2010 IF ASC( F$) > 57 THEN 1960
2020 F = VAL( F$)
2030 IF INT(TN/ F) <> TN/F THEN 1960
2040 IF F = 1 OR F = TN THEN 1960
2050 RETURN
2060 REM
2070 REM PRIME CHECK
2080 PF = 0
2090 FOR I = 1 TO 29
2100 IF N = PN( I ) THEN PF = 1
2110 NEXT I
2120 RETURN
2130 REM
2140 REM FLY PLANE
2150 REM SWITCH ON PLANE, SPRITE 0
2160 POKE VC + 21, 1
2170 REM PLANE ON RIGHT HAND SIDE
2180 REM SO SET MOST SIGNIFICANT BIT
2190 POKE VC + 16, 1
2200 FOR I = 50 TO 0 STEP -1
2210 REM ADJUST X-COORD OF PLANE
2220 POKE VC, I
2230 REM LEAVE Y-COORD CONSTANT
2240 POKE VC + 1, 80
2250 NEXT I
2260 REM CONTINUE FLIGHT ON LEFT
2270 REM TURN OFF MOST SIGNIFICANT BIT
2280 POKE VC + 16, 0
2290 REM DROP BOMB, SWITCH ON SPRITE 1
2300 POKE VC + 21, 3
2310 FOR I = 255 TO 189 STEP -1
2320 REM ADJUST X-COORD OF PLANE
2330 POKE VC, I
2340 REM LEAVE Y-COORD CONSTANT
2350 POKE VC + 1, 80
2360 REM BOMB HAS SAME X-COORD AS PLANE
2370 POKE VC + 2, I
```

```
2380 REM Y-COORD OF BOMB
2390 POKE VC + 3, 355 - I
2400 NEXT I
2410 REM SWITCH BOMB OFF, FLASH ON
2420 POKE VC +21, 5
2430 REM FLY PLANE ON
2440 FOR I = 188 TO 0 STEP -1
2450 POKE VC, I : POKE VC + 1, 80
2460 NEXT I
2470 REM SWITCH OFF PLANE AND FLASH
2480 POKE VC + 21, 0
2490 RETURN
2500 REM
2510 REM PRINT TAB ROUTINE
2520 POKE 782, X : POKE 781, Y
2530 SYS 65520
2540 RETURN
2550 REM
2560 REM DATA FOR PLANE, SPRITE 0
2570 REM
2580 DATA 7, 0, 0, 15, 3, 128, 30, 4
2590 DATA 64, 60, 127, 255, 252, 255
2600 DATA 255, 255, 255, 255, 255, 127
2610 DATA 255, 254, 3, 255, 224, 0, 255
2620 DATA 224, 0, 63, 224, 0, 15, 240
2630 DATA 0, 3, 252, 0, 0, 254, 0,0, 63
2640 REM
2650 REM DATA FOR BOMB, SPRITE 1
2660 REM
2670 DATA 4, 0, 0, 6, 0, 0, 7, 0, 0
2680 DATA 24, 0, 0, 112, 0, 0, 240, 0
2690 DATA 0, 240, 0, 0, 224, 0
2700 REM
2710 REM DATA FOR FLASH, SPRITE 2
2720 REM
2730 DATA 65, 0, 0, 42, 0, 1, 0, 64, 0
2740 DATA 136, 128, 4, 65, 16, 2, 42
2750 DATA 32, 1, 0, 64, 0, 227, 128
2760 REM
2770 REM DATA FOR PRIME NUMBERS
2780 REM
2790 DATA 2, 3, 5, 7, 11, 13, 17, 19
2800 DATA 23, 29, 31, 37, 41, 43, 47
```

```
2810 DATA 53, 57, 59, 61, 67, 71, 73
2820 DATA 79, 83, 87, 89, 91, 93, 97

READY.
```

Commentary

Factor is controlled by lines 1020–1070.

The first subroutine, called by line 1020 of the control module, defines three sprites used in the program. Sprite 0 is a plane, sprite 1 is the bomb that it drops and sprite 2 is the flash of the resultant explosion when the plane successfully hits its target number. Each of the three sprites is defined in a similar way.

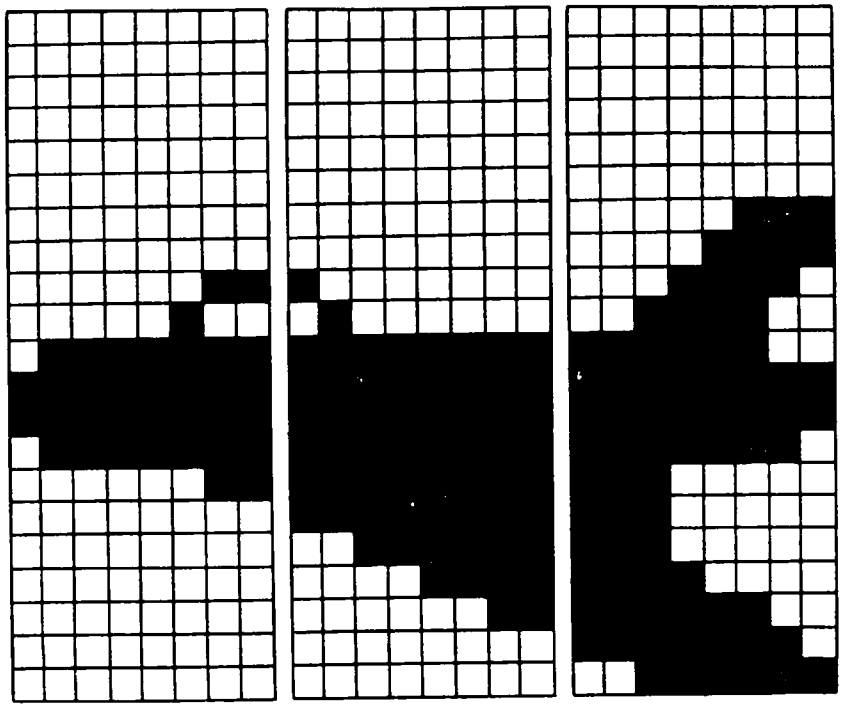
The diagram shows how the plane has been drawn on a rectangular grid of 21 rows and 24 columns. Each block of eight horizontal squares has been coded as a binary number and then stored in a special area of memory reserved by line 1150. Usually after a sprite has been designed and calculated as a set of 63 numbers, these are held inside a program as DATA. However, in the case of the plane sprite it can be seen from the diagram that the first twenty blocks of eight squares have not been filled in at all. Thus the numbers corresponding to these blocks are each zero.

It is much easier to place these values into memory with the simple FOR ...NEXT loop between lines 1170–1190 than to store them as DATA. The remaining numbers for the sprite are, however, READ from the DATA lines. The values for sprite 0 are held at locations starting at 832. The two other sprites are stored at locations starting at 896 and 960. Each time, the 'sprite pointer' is set to indicate where in memory the sprite data is to be found.

The rest of the routine then attends to other sprite parameters. These are all done in terms of the number VC defined at line 1120. This is the starting location of the video chip, which handles all the sprite manipulation. POKE VC + 29, for example, is the location which controls the size of a sprite in the horizontal, x, direction. POKEing the sprite number to VC + 29 will cause it to be stretched sideways. In this program, both sprite 0 and sprite 2 need to be expanded and so the value 5 is POKEd. This is the sum of 1 (sprite 0) and 4 (sprite 2). In a similar way, the colour and location of sprites on the screen are all controlled with reference to this important location VC.

In the operation of the program, it is necessary to be able to identify prime numbers. Primes are held, again as lines of DATA, and READ into the PN array by the routine between lines 1590–1660.

Diagram A



The screen display is built up by the routine between lines 1680–1840. As in the program Bingo, the screen is controlled by directly addressing the screen and colour maps. For example, the bottom section of the screen is turned into a desert by the nested loops between lines 1760–1830.

The test routine from lines 1860–2120 present a target number which the child has to attempt to bomb. This can only be done by typing in a correct factor. The input routine is error-trapped to prevent the number itself, or 1, or non-numeric values, from being accepted. The target number itself is carefully tested by the separate subroutine to make sure that a prime number is not set. This is achieved by using the array of prime numbers set up earlier in the program.

The routine from 2140–2500 flies the plane over the desert and bombs the number if a correct factor has been typed. The REMs present in this routine indicate each stage of the flight. Note how the location VC + 16 has been set at line 2190 to allow the flight across the righthand side of the screen.

CHAPTER 2
Small is Beautiful...

Short programs and efficiency — Missing: find the numbers — ordering numbers — Series: maths as an introduction to strategy — simplicity in program structure — Number: a program to find a hidden number — using graphics to brighten up simple ideas — Firework: a program to test general numerical skills — the problems involved in learning fractions — Fraction: a program to help with numerators and denominators.

Although it has been pointed out that exciting displays do not turn a mediocre program into a good one, there is no doubt that some programs benefit from interesting graphics.

At the beginning of this chapter, we shall show you some short programs in which the fast calculating power of your 64 is used, rather than its graphic ability. Towards the end, some of these ideas are put into a more advanced program that includes graphics.

Firstly though, an indication of simple ways in which the computer can do tasks faster and more efficiently than humans.

A question of format

In the last chapter, there was a table test program that used the computer to generate the questions. The same idea can be used with any sort of maths problem. One difficulty that many children in the early primary years experience is that of coping with problems posed in an unfamiliar way. If a sum says

$15 - ? = 7$

then it is likely to pose more problems than one that says

$15 - 7 = ?$

The following program chooses 10 problems at random using this format.

Missing

```
1 REM *** MISSING NUMBERS ***
20 REM
30 REM ***  A.J.S. 1/7/83  ***
40 REM
50 S=0:PRINT"[SHIFT/CLR/HOME]":PRINT
60 FOR N = 1 TO 10
70 A = INT(RND(0)*100)
80 B = INT(RND(0)*100)
90 IF B>A THEN 70
100 C = A - B
110 PRINT N;".)";A;" - ? =";B
120 PRINT
130 INPUT "MISSING NUMBER IS";GUESS
140 IF GUESS = C THEN PRINT "OK":S=S+1
150 IF GUESS <>C THEN PRINT "NO";C
160 FOR COUNT=1 TO 500:NEXT COUNT
170 PRINT "[SHIFT/CLR/HOME]"
180 NEXT N
190 PRINT"YOUR SCORE WAS"S"OUT OF 10"

READY.
```

Line 50 sets the score to zero and clears the screen. Line 60 starts the loop to ask 10 problems.

Lines 70 and 80 choose random whole numbers between 0 and 99, and line 90 checks that B is not greater than A.

Line 100 calculates the answer. Line 110 sets out the question line. The semicolons in this line are not essential, as they are in an INPUT line, eg line 130.

Line 120, a simple PRINT statement, is an easy way of missing a line.

Line 130 requires an input, GUESS. Lines 140 and 150 check GUESS and compare it with the correct answer, C. If it is the same, the score is increased by one.

Line 160 shows a simple way of introducing a pause. The empty FOR...NEXT loop counts up from 1 to 500, and then allows the program to continue.

Line 170 clears the screen. Line 190 prints the score, S.

Everything in order

Sequences and series of numbers sometimes take a long time before the idea is grasped. The next program shows how the computer can generate a series of numbers in a particular range. Although not an elaborate program, it will generate sequences and mark the responses all day without getting tired.

The particular sequence chosen is one that increases by the same amount each time, eg

4 8 12 16 20 24 etc.

It is quite easy to change the program so that it chooses numbers where the increment alters by a set amount, eg

1 2 4 7 11 15 etc.

or one that generates squares such as

1 4 9 16 25 36 etc.

Series

```
10 REM *** SERIES GUESSER ***
20 REM
30 REM *** A.J.S. 5/7/83  ***
40 REM
50 PRINT"[SHIFT/CLR/HOME]":PRINT
60 PRINT "SERIES"
65 PRINT:S = 0
70 FOR N = 1 TO 10
75 PRINT "QUESTION"N
80 A=INT(RND(0)*26)
90 INCR =INT(RND(0)*11)
100 PRINT A;
110 FOR X = 1 TO 4
120 PRINT A + (INCR * X);
```

```
130 NEXT X
140 PRINT
150 B = A + (INCR * X)
160 INPUT "WHAT IS THE NEXT NUMBER";G
170 IF G = B THEN PRINT"OK":S=S+1
180 IF G <>B THEN PRINT"NO"B
190 NEXT N
200 PRINT "YOU SCORED"S"OUT OF 10"

READY.
```

Commentary

Line 50 clears the screen. Line 65 sets the score to 0 at the start.

Line 70 begins the loop to generate 10 questions. Line 80 chooses A, the start of the series, at random. Line 90 chooses the size of the step or increment between each number, INCR. Line 100 prints the first number in the sequence.

Lines 110 to 130 contain a loop to generate the next four numbers in the series. Each number is bigger by the amount INCR.

Line 150 relies on the fact that the FOR...NEXT loop counters contain a value one higher than the last count of the loop, ie 5, to set B equal to the next number in the sequence. Line 160 asks for an input, G.

Lines 170 and 180 compare G with the correct answer, B, and print OK or NO, as well as increasing the score if necessary.

Line 200 prints the final score.

Logical guessing

There is a game that you can play using the blackboard, a piece of paper, or just your memory. It is the familiar one of 'I am thinking of a number — ask me any question about it you like, but I can only reply using yes or no.' Faced with this problem for the first time, the child may be inclined to guess. If he or she realises that the number can be anything under (say) 1000, it is not long before it is appreciated that the task is difficult, and that some sort of logical strategy must be employed.

In a classroom situation, the class usually divides up into 'guessers, 'hypothesisers' and 'pseudo-hypothesisers'. Guessers do just that. Hypothesisers try to reduce the number of possible choices by asking such questions as 'is it an even number' or 'is it less then 500'. Pseudo-hypothesisers are more subtle than the guessers. If the response 'no' is received to the question 'is the number less than 500', then the sort of question they will ask is 'is it more than 500'. It sometimes seems a big hurdle to jump before this type of child realises that a 'no' response to a question provides as much information as a 'yes'.

This sort of game is a useful activity as it can improve mental skills including memory and visualisation of a sort of internal number line. It also helps develop logical questioning and inquiry strategies. You can probably see that the most logical strategy to follow is to divide the answer approximately by two.

A typical series of questions may go as follows:

Number is 242.

Is it larger than 500? No.
Is it larger than 250? No.
Is it larger than 125? Yes.
Is it larger than 190? Yes.
Is it larger than 230? Yes.
Is it larger than 240? Yes.
Is it larger than 245? No.
Is it larger than 243? No.
Is it larger than 241? Yes.
Is it 242? Yes.

Using this binary chopping method, it is always possible to reach the number without guessing in less than ten attempts — two raised to the power of ten is, of course, 1024. Most older children will adopt this method, or one similar, until they get fairly close and then guess. It is not envisaged that they will actually be calculating the odds on a correct guess, although at this level it does resemble gambling!

A computer can choose the number, select correct responses and give the total number of attempts at the end. In this way, a teacher or parent can be relieved of the task of administering this game to children, and the program allows an individual or group approach. For the child who finds it difficult to acquire a good strategy, it can provide as much practice as is necessary.

There are no interesting displays, no exciting sounds, and the program has been made as simple as possible to show how even short programs can have their point.

Number

```
10 REM ** NUMBER GUESSER **
20 REM ** A.J.S.  1/4/82 **
30 REM
40 POKE53281,1:REM BACKGROUND WHITE
50 PRINT"[SHIFT/CLR/HOME]"
60 SC=0
70 BO=0
80 PRINT"I WILL THINK OF A NUMBER. ";
90 PRINT"YOU MUST TRY"
100 PRINT"TO GUESS IT IN AS FEW GOES ";
110 PRINT"AS YOU CAN."
120 PRINT"CHOOSE A TOP LIMIT ";
130 INPUT"FOR YOUR NUMBER";TP
140 N=INT(RND(1)*TP)
150 PRINT"THE NUMBER LIES BETWEEN ";
160 PRINTBO;" AND ";TP
170 SC=SC+1
180 PRINT"ENTER GUESS NO.";SC;" ";
190 INPUT GS
200 IF GS=N THEN GOTO300
210 IF GS>N THEN TP=GS
220 IF GS<N THEN BO=GS
230 GOTO150
240 REM
300 PRINT"YOU GOT IT IN ";SC;" GOES"
310 PRINT"PRESS SPACE BAR TO GO AGAIN"
320 GET Z$
330 IF Z$<>" " THEN 320
340 GOTO50

READY.
```

Commentary

Line 40 sets the background colour to white. Up to this point, you have probably been putting up with the light blue background that appears when you switch on. POKEing this location in memory with a number from zero to 16 sets the background colour.

Line 60 initialises the score, SC. Line 70 sets the bottom limit for the number to zero.

Lines 80 to 120 contain the instructions.

Line 130 allows the user to set the upper limit for the number. Line 140 chooses the number to be guessed.

Lines 180 and 190 allow the input of a guess.

Lines 200 to 220 check the guess, GS, against the chosen number, N. If the guess is larger than the number, then the guess becomes the new upper limit; and if the guess is smaller than the number, the guess becomes the new bottom limit.

Although it is not so easy to see, there is a loop between lines 150 and 230. This loop is negotiated each time a number is guessed, until the correct answer is given. This causes an exit in line 200. 64 BASIC only supports a FOR...NEXT loop, and this entails knowing how many times you will pass through it, which in this program, of course, you don't.

Lines 300 and 310 print the number of guesses, SC.

Line 320 contains the words GET Z$. This waits until a key is pressed, and then checks the input in line 330. If the key is anything except the space bar, the program repeats line 320.

Line 340 sends the program back to the start.

Firework

The next program, Firework, is also based on a simple idea — offering examples of the four rules (add, subtract, multiply and divide). The incentive to do well here is the impressive display after five correct answers.

Each of the graphic routines is held in a module, and may be adapted to your own use or copied directly into a different program. Full notes on how the effects are achieved are to be found at the end of the program.

```
1000 REM FIREWORK - PAT HALL, JAN '84
1010 REM
1020 GOSUB 1110 : REM DEFINE SPRITES
1030 GOSUB 1520 : REM SET UP DISPLAY
1040 GOSUB 1730 : REM TEST ROUTINE
1050 GOSUB 2040 : REM LAUNCH ROCKET
1060 GOTO 1030
1070 END
```

```
1080 REM
1090 REM DEFINE SPRITES
1100 REM BEGINNING OF 6567 VIDEO CHIP
1110 VC = 53248
1120 REM ROCKET, SPRITE 0
1130 REM SET POINTER FOR SPRITE 0
1140 POKE 2040, 13
1150 REM READ DATA FOR SPRITE 0
1160 FOR I = 0 TO 8
1170 READ N : POKE 832 + I, N
1180 NEXT I
1190 FOR I = 9 TO 51 STEP 3
1200 POKE 832 + I, 0
1210 POKE 833 + I, 127
1220 POKE 834 + I, 0
1230 NEXT I
1240 FOR I = 54 TO 62
1250 READ N : POKE 832 + I, N
1260 NEXT I
1270 REM EXHAUST, SPRITE 1
1280 REM SET POINTER FOR SPRITE 1
1290 POKE 2041, 14
1300 REM READ DATA FOR SPRITE 1
1310 FOR I = 0 TO 62
1320 READ N : POKE 896 + I, N
1330 NEXT I
1340 REM MOON, SPRITE 2
1350 REM SET POINTER FOR SPRITE 2
1360 POKE 2042, 15
1370 REM READ DATA FOR SPRITE 2
1380 FOR I = 0 TO 62
1390 READ N : POKE 960 + I, N
1400 NEXT I
1410 REM ENLARGE SPRITES IN X DIRECTION
1420 POKE VC + 29, 5
1430 REM ENLARGE SPRITES IN Y DIRECTION
1440 POKE VC + 23, 5
1450 REM SELECT COLOUR
1460 POKE VC + 39, 1
1470 POKE VC + 40, 2
1480 POKE VC + 41, 7
1490 RETURN
1500 REM
1510 REM BLUE SKY
```

```
1520 POKE 53280, 6 : POKE 53281, 6
1530 PRINT "[SHIFT/CLR/HOME]"
1540 REM DRAW GRASS
1550 C = 5 : REM SELECT GREEN
1560 FOR I = 0 TO 160 STEP 40
1570 FOR J = 0 TO 39
1580 L = 1824 + I + J
1590 GOSUB 2900
1600 NEXT J
1610 NEXT I
1620 REM PLACE ROCKET ON GRASS
1630 REM HORIZONTAL POSITION
1640 POKE VC, 165
1650 REM VERTICAL POSITION
1660 POKE VC + 1, 190
1670 REM SWITCH ON ROCKET, SPRITE 0
1680 POKE VC + 21, 1
1690 PRINT CHR$( 5 )
1700 RETURN
1710 REM
1720 REM TEST ROUTINE
1730 S = 0
1740 P = 10 : REM SET PAUSE
1750 X = 14 : Y = 0 : GOSUB 2700
1760 PRINT "<< FIREWORK >>"
1770 A = INT( RND( 1 ) * 96 ) + 5
1780 B = INT( RND( 1 ) * A )
1790 REM PREVENT TOO EASY QUESTIONS
1800 IF B < 11 THEN 1770
1810 GOSUB 2940 : REM PAUSE
1820 X = 8 : Y = 2 : GOSUB 2700
1830 PRINT "WHAT IS " A "-" B
1840 X = 30 : Y = 2 : GOSUB 2700
1850 PRINT "          "
1860 GOSUB 2700
1870 INPUT C$
1880 IF ASC( C$)< 49 THEN 1840
1890 IF ASC( C$)> 57 THEN 1840
1900 C = VAL( C$)
1910 IF C = A - B THEN 1950
1920 GOSUB 2700 : PRINT " NO " A - B
1930 GOTO 1770
1940 REM INCREASE SCORE
1950 S = S + 1 : IF S < 5 THEN 1770
```

```
1960 REM CLEAR TOP OF SCREEN
1970 X = 14 : Y = 0 : GOSUB 2700
1980 PRINT "                "
1990 X = 8 : Y = 2 : GOSUB 2700
2000 PRINT "                                "
2010 RETURN
2020 REM
2030 REM LAUNCH ROCKET
2040 P = 1 : REM ADJUST PAUSE
2050 REM SWITCH ON EXHAUST, SPRITE 1
2060 POKE VC + 21, 3
2070 SD = 1 : GOSUB 2770 : REM SOUND
2080 FOR I = 190 TO 120 STEP -1
2090 REM KEEP X COORDINATE CONSTANT
2100 POKE VC, 165
2110 REM ADJUST Y COORDINATE
2120 POKE VC + 1, I
2130 REM PLACE EXHAUST
2140 IF I > 160 THEN 2230
2150 REM ENLARGE EXHAUST
2160 REM ADJUST IX X DIRECTION
2170 POKE VC + 23, 7
2180 REM ADJUST IN Y DIRECTION
2190 POKE VC + 29, 7
2200 REM X COORDINATE FOR LARGE EXHAUST
2210 POKE VC + 2, 165
2220 GOTO 2240
2230 POKE VC + 2, 177
2240 POKE VC + 3, I + 43
2250 GOSUB 2940 : REM PAUSE
2260 NEXT I
2270 REM GROUND SLIPS AWAY
2280 C = 6 : REM SELECT BLUE
2290 FOR I = 0 TO 160 STEP 40
2300 FOR J = 0 TO 39
2310 L = 1824 + I + J
2320 GOSUB 2900
2330 NEXT J
2340 NEXT I
2350 P = 5 : GOSUB 2940 : REM PAUSE
2360 REM BLACK SKY
2370 POKE 53280, 0 : POKE 53281, 0
2380 PRINT "[SHIFT/CLR/HOME]"
2390 P = 10 : GOSUB 2940 : REM PAUSE
```

```
2400 REM MOON APPEARS
2410 POKE VC + 21, 7
2420 REM MOVE MOON DOWN SCREEN
2430 P = .5 : REM ADJUST PAUSE
2440 FOR I = 8 TO 250
2450 POKE VC + 4, 30
2460 POKE VC + 5, I
2470 GOSUB 2940 : REM PAUSE
2480 NEXT I
2490 P = 5 : GOSUB 2940 : REM PAUSE
2500 SD = 0 : GOSUB 2770 : REM SOUND
2510 REM STARBURST
2520 REM SWITCH OFF ROCKET, SPRITE 0
2530 POKE VC + 21, 0
2540 FOR I = 1 TO 8
2550 L = 1524 + I : GOSUB 2980
2560 L = 1524 - I : GOSUB 2980
2570 L = 1524 + I * 40 : GOSUB 2980
2580 L = 1524 - I * 40 : GOSUB 2980
2590 L = 1524 + I + I * 40 : GOSUB 2980
2600 L = 1524 + I - I * 40 : GOSUB 2980
2610 L = 1524 - I + I * 40 : GOSUB 2980
2620 L = 1524 - I - I * 40 : GOSUB 2980
2630 NEXT I
2640 REM LIGHT UP SKY
2650 POKE 53280, 3 : POKE 53281, 3
2660 P = 50 : GOSUB 2940 : REM PAUSE
2670 RETURN
2680 REM
2690 REM PRINT TAB ROUTINE
2700 POKE 782, X : POKE 781, Y
2710 SYS 65520
2720 RETURN
2730 REM
2740 REM SOUND EFFECT
2750 REM SD = 1, SOUND ON
2760 REM SD = 0, SOUND OFF
2770 IF SD = 0 THEN 2840
2780 POKE 54296, 15
2790 POKE 54277, 129
2800 POKE 54278, 129
2810 POKE 54276, 129
2820 POKE 54273, 10 : POKE 54272, 60
```

```
2830 GOTO 2870
2840 POKE 54277, 0
2850 POKE 54278, 0
2860 POKE 54276, 0
2870 RETURN
2880 REM
2890 REM PLOT CHARACTER SPACE
2900 POKE L + 54272, C : POKE L, 160
2910 RETURN
2920 REM
2930 REM PAUSE
2940 FOR T = 1 TO P * 30 : NEXT T
2950 RETURN
2960 REM
2970 REM PLOT STAR
2980 C = INT( RND( 1 ) * 7 ) + 1
2990 POKE L + 54272, C : POKE L, 42
3000 RETURN
3010 REM
3020 REM DATA FOR ROCKET, SPRITE 0
3030 REM
3040 DATA 0, 8, 0, 0, 28, 0, 0, 62, 0
3050 DATA 0, 8, 0, 0, 28, 0, 0, 62, 0
3060 REM
3070 REM DATA FOR EXHAUST, SPRITE 1
3080 REM
3090 DATA 0, 8, 0, 0, 8, 0, 0, 24, 0
3100 DATA 0, 60, 0, 0, 118, 0, 0, 231
3110 DATA 0, 1, 227, 128, 1, 225, 192
3120 DATA 1, 225, 224, 0, 241, 224, 0
3130 DATA 249, 240, 0, 125, 240, 0, 29
3140 DATA 240, 0, 15, 224, 0, 7, 192
3150 DATA 0, 7, 128, 0, 7, 128, 0, 7, 0
3160 DATA 0, 14, 0, 0, 12, 0, 0, 16, 0
3170 REM
3180 REM DATA FOR MOON, SPRITE 2
3190 REM
3200 DATA 0, 30, 0, 0, 7, 128, 0, 3
3210 DATA 224, 0, 1, 240, 0, 0, 248, 0
3220 DATA 0, 252, 0, 0, 126, 0, 0, 126
3230 DATA 0, 0, 127, 0, 0, 127, 0, 0
3240 DATA 127, 0, 0, 127, 0, 0, 127, 0
3250 DATA 0, 126, 0, 0, 126, 0, 0, 252
```

```
3260 DATA 0, 0, 248, 0, 1, 240, 0, 3
3270 DATA 224, 0, 7, 128, 0, 30, 0

READY.
```

Commentary

The control module for Firework is formed by the lines 1020–1060. It can be seen that the program is in an overall loop and so RUN/STOP has to be used to escape from it.

The first routine of the program defines three sprites. First the value of VC is initialised by line 1110. Then the rocket, its exhaust and the Moon are defined in turn as sprites 0–2. The three diagrams show each of the sprites in detail. It can be seen that the major part of the rocket is a very repetitive shape and this pattern is reflected in the way that the corresponding sprite data is built up by the use of the FOR...NEXT loop from lines 1190–1230. The other two sprites are built up instead simply from the DATA stored at the end of the program. Lines 1410–1480 then select the size and colour of the sprites.

Lines 1510–1690 colour the screen blue and place grass at the bottom. The rocket is located on the grass by POKEing the two locations VC and VC+1, which control its horizontal and vertical location on the screen. Finally the rocket is made to appear by POKEing its sprite number, 1, to VC+21. This is the location which controls whether a particular sprite appears on the screen or not. Line 1690 selects white text for the subsequent test routine.

The test given to the child is simple subtraction of numbers. Line 1770 chooses A to be a number less than 100. Line 1780 then multiplies A by a random decimal number and takes the integral part. In this way, it is ensured that the number being subtracted is not larger than the one it is being subtracted from. Line 1800 then prevents the calculations from being too easy. Again extensive use is made of a PRINT TAB subroutine to allow the screen to be neatly arranged. Lines 1840–1890 stop letters being accepted as an input. The variable S keeps a record of the child's success and governs exit from the test to the reward routine at line 1950.

Lines 2030–2670 control the flight of the rocket up into the sky. Here the sprites allow easy animation. An increased sense of upward motion is attempted by removing the ground between lines 2280–2340 and by moving the Moon downwards in the loop 2440–2480. The sound of the rocket is created by the separate routine between lines 2740–2870 and controlled by the value of the flag SD. The loop from 2540–2630 uses another subroutine to place a starburst on the screen.

Diagram B

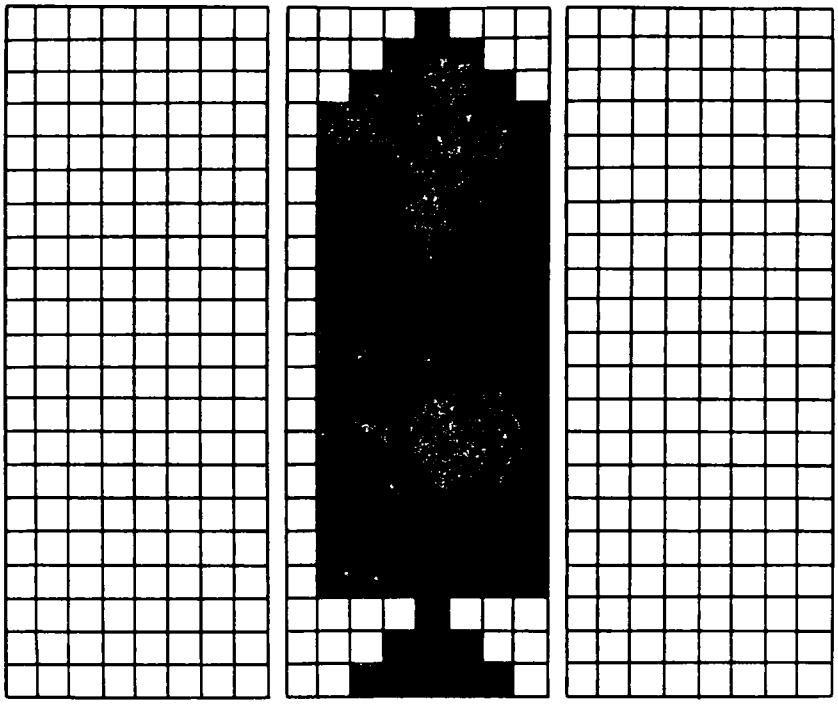


Diagram C

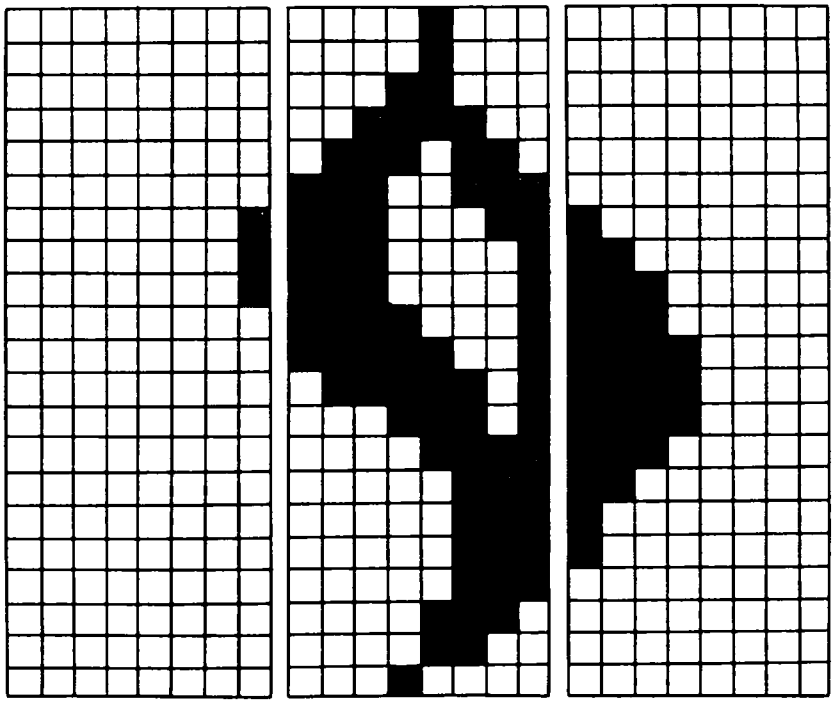
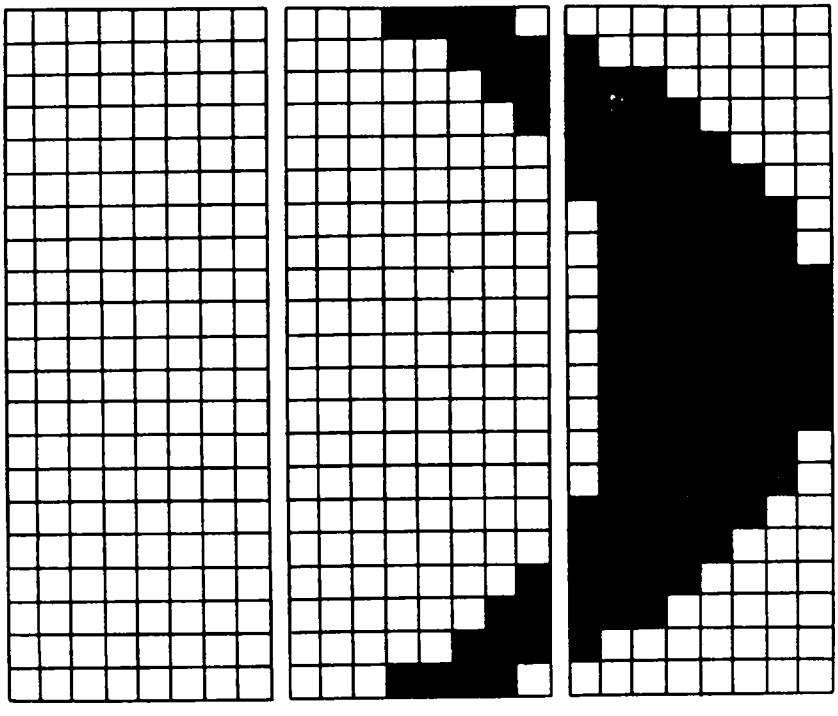


Diagram D



Fraction

There is no doubt that in the whole of maths teaching, there is no topic so off-putting to children, parents and teachers alike as that of fractions.

Children of seven and eight often have an intuitive grasp of halves and quarters, but the notation of vulgar fractions is unfamiliar and the terms numerator and denominator are awkward in themselves. Recent surveys, such as the Cockroft Report (1982), have implied that work should concentrate on decimal fractions at least until the upper end of primary schools.

This does not mean that the idea of fractions can go out of the window. Children need a lot of practical experience in dividing objects up into equal-sized parts before they understand fractional concepts. It is usually in the recognition of what a fraction stands for, and in the manipulation of its parts, that problems occur. In this chapter, we have attempted to tackle this problem in different ways.

Fraction displays a rectangle, representing the whole object; it then colours in a particular fraction and asks the child to input the numerator and denominator. These terms are used on the screen and there is no doubt about which is which.

As an aid to knowing how to express the fraction, ie whether $\frac{1}{2}$ or $\frac{2}{4}$ is required, the whole rectangle is divided up into the same number of parts as the denominator. For instance, if $\frac{2}{3}$ is required, the rectangle divides into six clearly-defined sections, and two of these are coloured in. Should the child make an incorrect input, there is a routine to show how the fraction is built up. If the child is successful, the examples become progressively more difficult, whereas those for a child with difficulties remain simple.

```
1000 REM FRACTION - PAT HALL, JAN '84
1010 REM
1020 GOSUB 1200 : REM INITIALISATION
1030 GOSUB 1170 : REM SELECT FRACTION
1040 GOSUB 1360 : REM DRAW DIAGRAM
1050 GOSUB 2010 : REM REQUEST ANSWER
1060 GOTO 1030
1070 END
1080 REM
1090 REM INITIALISATION
1100 REM SCORE AND NUMBER ATTEMPTED
1110 S = 0 : G = 0
1120 REM LEVEL OF DIFFICULTY
1130 DF = 1
1140 RETURN
1150 REM
1160 REM SELECT FRACTION
```

```
1170 D = INT( RND( 1 ) * DF ) + 2
1180 N = INT( RND( 1 ) * D )
1190 IF N < 1 THEN 1180
1200 REM SELECT WIDTH OF DIVISIONS
1210 W = 1
1220 IF D < 13 THEN W = 2
1230 IF D < 10 THEN W = 3
1240 IF D < 8 THEN W = 4
1250 IF D < 7 THEN W = 5
1260 IF D < 6 THEN W = 6
1270 IF D < 5 THEN W = 8
1280 IF D < 4 THEN W = 11
1290 IF D < 3 THEN W = 17
1300 REM LOCATE TOP LEFT OF DIAGRAM
1310 LD = 1203 - D * ( W + 1 ) / 2 - W
1320 RETURN
1330 REM
1340 REM DRAW DIAGRAM
1350 REM WHITE SCREEN
1360 POKE 53280, 1 : POKE 53281, 1
1370 PRINT "[SHIFT/CLR/HOME]"
1380 REM PRINT TITLE
1390 PRINT CHR$( 144 )
1400 X = 13 : Y = 0 : GOSUB 1950
1410 PRINT "<< FRACTION >>"
1420 PRINT CHR$( 30 )
1430 X = 12 : Y = 2 : GOSUB 1950
1440 PRINT "SCORE: " S " / " G
1450 PRINT CHR$( 144 )
1460 REM DRAW D DIVISIONS
1470 FOR I = 1 TO D
1480 REM LOCATE TOP LEFT OF DIVISION
1490 LD = LD + W + 1
1500 REM PLOT TOP AND BOTTOM OUTLINE
1510 C = 0 : REM SELECT BLACK
1520 FOR J = 0 TO W
1530 L = LD + J
1540 GOSUB 1920
1550 L = LD + 560 + J
1560 GOSUB 1920
1570 NEXT J
1580 REM SOUND EFFECT
1590 POKE 54296, 15 : POKE 54277, 129
1600 POKE 54278, 129 : POKE 54276, 17
```

```
1610 POKE 54273, 68 : POKE 54272, 149
1620 FOR T = 1 TO 90 : NEXT T
1630 POKE 54277, 0 : POKE 54278, 0
1640 POKE 54276, 0
1650 REM LEFT HAND EDGE
1660 FOR J = 40 TO 520 STEP 40
1670 L = LD + J
1680 GOSUB 1920
1690 NEXT J
1700 REM COLOUR IN DIVISION
1710 REM COLOUR NUMERATOR GREEN
1720 C = 5
1730 REM COLOUR OTHER DIVISIONS YELLOW
1740 IF I > N THEN C = 7
1750 FOR J = 40 TO 520 STEP 40
1760 FOR K = 1 TO W
1770 L = LD + J + K
1780 GOSUB 1920
1790 NEXT K
1800 NEXT J
1810 NEXT I
1820 REM FINAL RIGHT HAND EDGE
1830 C = 0 : REM SELECT BLACK
1840 LD = LD + W + 1
1850 FOR I = 0 TO 560 STEP 40
1860 L = LD + I
1870 GOSUB 1920
1880 NEXT I
1890 RETURN
1900 REM
1910 REM PLOT CHARACTER SPACE
1920 POKE L + 54272, C : POKE L, 160
1930 RETURN
1940 REM PRINT TAB ROUTINE
1950 POKE 782, X : POKE 781, Y
1960 SYS 65520
1970 RETURN
1980 REM
1990 REM REQUEST ANSWER
2000 REM INCREMENT NUMBER ATTEMPTED
2010 G = G + 1
2020 REM REQUEST ANSWER
2030 X = 8 : Y = 20 : GOSUB 1950
2040 PRINT "WHAT FRACTION IS GREEN ?"
```

```
2050 Y = 22 : GOSUB 1950
2060 PRINT "TYPE NUMERATOR:-"
2070 XA = 22 : GOSUB 2200
2080 N1 = AN
2090 X = 8 : Y = 23 : GOSUB 1950
2100 PRINT "TYPE DENOMINATOR:-"
2110 XA = 24 : GOSUB 2200
2120 D1 = AN
2130 REM CHECK ANSWER
2140 IF N1=N AND D1=D THEN GOSUB 2330
2150 IF N1<>N OR D1<>D THEN GOSUB 2500
2160 FOR T = 1 TO 2000 : NEXT T
2170 RETURN
2180 REM
2190 REM ACCEPT ANSWER
2200 X = XA
2210 GOSUB 1950
2220 PRINT "            "
2230 GOSUB 1950
2240 INPUT AN$
2250 REM REJECT NON NUMERICAL VALUES
2260 IF ASC( AN$) < 48 THEN 2200
2270 IF ASC( AN$) > 57 THEN 2200
2280 AN = VAL( AN$)
2290 RETURN
2300 REM
2310 REM REWARD ROUTINE
2320 REM GREEN SCREEN
2330 POKE 53280, 5 : POKE 53281, 5
2340 PRINT "[SHIFT/CLR/HOME]"
2350 REM INCREASE SCORE
2360 S = S + 1
2370 REM INCREASE DIFFICULTY
2380 DF = DF + 1
2390 IF DF > 17 THEN DF = 17
2400 PRINT CHR$( 5 )
2410 FOR I = 1 TO 11
2420 PRINT "WELL DONE. THAT IS THE ";
2430 PRINT "CORRECT ANSWER."
2440 PRINT
2450 NEXT I
2460 RETURN
2470 REM
2480 REM CORRECTION ROUTINE
```

```
2490 REM YELLOW SCREEN
2500 POKE 53280, 7 : POKE 53281, 7
2510 PRINT "[SHIFT/CLR/HOME]"
2520 REM DECREASE DIFFICULTY
2530 DF = DF - 1
2540 IF DF < 1 THEN DF = 1
2550 FOR I = 1 TO 11
2560 PRINT "   NO. THE CORRECT ";
2570 PRINT "ANSWER IS " N "/" D
2580 PRINT
2590 NEXT I
2600 RETURN

READY.
```

Commentary

The control module for Fraction lies between 1020–1060, which form a loop calling up initialisation, selection, drawing and testing routines. The program does not come to an end unless RUN/STOP is pressed.

Lines 1090–1140 are the initialisation routine. The variables S and G are set equal to zero at line 1110. They record the score the child has achieved and the number of fractions attempted. The variable DF is initialised by line 1130. DF controls how hard each fraction is. The smaller the value of DF, the smaller the numerator and denominator of the fraction drawn on the screen. The value of DF is continuously monitored during the execution of the program, so that when the child types in a correct answer the level of difficulty increases. If an answer typed in is incorrect, the value of DF is decreased so that the fractions become easier. It is not quite this simple, as a random factor is introduced as well, but DF does determine the overall trend towards harder or easier examples.

The fraction is generated between lines 1160–1320. The denominator, D, is selected at line 1170. DF enters into the expression as the number that the random fraction RND(1) is multiplied by. Line 1180 relates the size of the numerator, N, to that of D. Obviously, the way in which the two numbers are generated ensures that the numerator is smaller than the denominator, as is necessary.

Lines 1210–1290 relate the value of the variable W to that of D. W is used in the following routine when the diagram of the fraction is drawn on the screen. Here, however, it is adjusted to give the largest possible diagram for the fraction chosen. Finally, this routine calculates a value for the

variable LD in terms of W and D. LD allows the subsequent calculation of the top lefthand corner of each division of the fraction.

Lines 1340–1890 form the routine which places the fractional diagram on the screen. It is built up quite naturally in a series of blocks or divisions corresponding in number to the size of the denominator.

After the screen has been cleared by line 1370 and the title and other headings placed by lines 1380–1440, a lengthy FOR...NEXT loop, from lines 1470–1810, constructs each division of the diagram in turn. At the beginning of this loop, a new value is first calculated for LD for the current division. The J loop between lines 1520–1570 draws the top and bottom of the division in black. A subroutine at lines 1910–1930 is used by this loop, and also for all successive plotting on the screen. This is followed by a brief sound effect to help draw attention to the process of building up the diagram and hence to aid the child's counting. The sound is generated by the various POKEs between lines 1590–1640. After this, the lefthand edge of the division is plotted by the loop between 1650–1690.

So far, only three sides of the division have been drawn. The enclosed area is coloured in by the two nested FOR...NEXT loops from 1750–1800. The colour the area is plotted in is chosen at lines 1710–1740. The numerator of the fraction appears in green and the remaining divisions in yellow. Line 1810 ends the I FOR...NEXT loop.

After the D divisions are drawn, there is still the extreme righthand edge of the diagram to be filled in. The loop between lines 1850–1880 does this. Note how the same subroutine has been repeatedly called during the whole of the construction of the diagram.

Lines 1990–2160 make extensive use of the PRINT TAB routine between 1940–1970 to format the presentation of the question on the screen. Another subroutine at lines 2190–2290 checks that acceptable values are typed in. Lines 2140 and 2150 then select either reward or correction routines at the end of the program. It is here that the value of DF is adjusted to correspond to the degree of success or failure the child is experiencing. Success will lead fairly quickly to harder examples, whereas constant failure would lead to the child using the program being shown a diagram of $\frac{1}{2}$ over and over again, together with the correction routine. Although it is quite simple to arrange for programs to be self-adjusting to the child's level of ability, nevertheless such an arrangement allows far more effective learning programs to be constructed.

CHAPTER 3
Words, Words, Words

Handling words on a computer — sorting things — Alphasort: a program to sort words into alphabetical order — numbers in strings — Simon: a game to improve memory — diversity of subjects in primary schools — problems in spelling — different approaches — context clues — Hangman: a multi-purpose spelling program — research skills — Anagram: clues from word order.

So far in the book, the majority of programs have involved some sort of maths skills. The 64 contains many facilities that make the handling of words much easier. As has been mentioned before, computers are just as happy manipulating letters as figures. Numbers are stored as numeric variables, such as X or COUNT, and strings of characters as string variables, such as A$ or BOOK$. Each character in the 64 has a code number that refers to it, so 65 stands for A, and 90 stands for Z. You can check this out yourself by entering this short program.

```
10 FOR N = 65 TO 255
20 PRINT CHR$(N)
30 NEXT N
```

The function CHR$ converts the Commodore code to its character equivalent, so you will see the alphabet printed out if you RUN the program. (This will be in upper case if you are in the normal mode after switching on, or in lower case if you have changed modes by pressing SHIFT and the Commodore key [C =] at the same time.)

The function that does the reverse of CHR$ is ASC. ASC is short for ASCII, the initials of the American Standard Code for Information Interchange. Although the 64 has its own codes for graphics symbols, it follows ASCII codes for letters and numbers. This program shows you how ASC works:

```
10 PRINT "PRESS A KEY"
20 POKE 198,0
30 GET Z$:IF Z$ = "" THEN 30
```

```
40 C = ASC(Z$)
50 PRINT C
60 GOTO 10
```

Commentary

Line 20 clears the keyboard buffer (a temporary memory store for keyboard input).

Line 30 reads the keyboard and repeats until a key is pressed. Note that there is nothing between the two quote marks. This is called an empty string.

Line 40 assigns the ASCII code of the key pressed to the variable C.

Lines 50 and 60 print the code and send the program back to the start.

If you run this program, you will discover many useful things. Apart from pressing letter keys, try pressing number keys, the arrowed cursor keys, and the function keys. You will find that all the keys generate codes except [RESTORE], [CTRL], [SHIFT] and [SHIFT LOCK], [C=] and [RUN/STOP] on their own. Many keys, such as A, have three codes, depending on whether you are pressing [SHIFT] or [C=] at the same time.

The key A normally has a code of 65, but it produces 193 with [SHIFT] and 176 if [C=] is also held down. This is useful in programs, as you can check which keys are being pressed at any time.

If you have a program that requires a word or phrase to be input into a string, the length of the string can be checked by using the function LEN. If you enter directly

```
X$ = "HELLO"
```

then

```
PRINT LEN(X$)
```

the result 5 will appear, as HELLO has five characters.

Other string functions are RIGHT$, LEFT$, and MID$. RIGHT$ returns the righthand portion of a string, LEFT$ the lefthand portion, and MID$ can extract a portion from anywhere in the middle. To see how they operate, try this:

```
10 A$ = "ABCDEFGHIJ"
20 PRINT RIGHT$(A$,4)
```

```
30 PRINT LEFT$(A$,4)
40 PRINT MID$(A$,2,6)
```

If you RUN this, it will print GHIJ in line 20, as these letters are the last four letters on the right of the string. ABCD will be printed out in line 30, as they are the first four letters on the left of the string. Line 40 will print out BCDEFG, as these are the six letters starting from the second character in the string.

This program will show you how the 64 can search through strings using these functions:

```
10 PRINT "ENTER A WORD"
20 INPUT A$
30 IF LEN(A$)< 3 THEN PRINT "TOO SHORT":GOTO 10
40 PRINT A$" IS "LEN(A$)" CHARACTERS LONG"
50 PRINT "IT STARTS WITH "LEFT$(A$,1)
60 PRINT "AND ENDS WITH "RIGHT$(A$,1)
70 PRINT "AND CONTAINS "MID$(A$,2,LEN(A$)-2);
80 PRINT "IN THE MIDDLE"
```

Because the 64 gives all the keyboard characters numbers, it is possible to sort words into alphabetical order according to their ASCII codes. This program prints out a list of the words in the order in which they were input, then sorts them and prints them in their new order. Although 20 words can be sorted in about four seconds, 100 take about a minute and a half, as the number of individual comparisons grows. Enter line 245 S = S + 1, and PRINT S at the end, if you want to see how many comparisons were made between words.

Alphasort

```
100 REM *** ALPHASORT  ***
110 REM
120 REM *** AJS 5/8/83 ***
130 REM
140 REM
150 INPUT "NUMBER OF WORDS";N
160 DIM A$(N+1)
170 FOR X = 0 TO N-1
180 INPUT A$(X)
190 NEXT X
```

```
200 FOR X = 0 TO N-1
210 PRINT A$(X)
220 NEXT X
230 FOR K = 0 TO N-1
240 FOR L = K+1 TO N
250 IF A$(L) >= A$(K) THEN 290
260 U$ = A$(L)
270 A$(L) = A$(K)
280 A$(K) = U$
290 NEXT L
300 NEXT K
310 FOR X = 0 TO N
320 PRINT A$(X)
330 NEXT

READY.
```

Commentary

There are one or two new ideas in this program. A sorting routine needs to hold the words, not in one variable, but in an array. This has to be DIMensioned so that some space in memory is reserved for the words that will be stored in it.

Line 160 reserves space according to the number of words to be sorted. Lines 160 to 220 print out the words to be sorted.

Lines 230 to 300 contain the sorting routine. There are two loops, K and L, and the ASCII codes of the words are compared in line 250. If a word has a higher code than the one after it, they are swopped in the list. The loop compares all the words against each other until they are in alphabetical order.

Lines 310 to 330 print out the new list.

Strings and things

As has been mentioned previously, string variables can contain any keyboard character, not just letters. Although you can redefine characters to be any shape you like and use them in your programs, all Commodore computers contain graphics characters that can be produced by the keyboard. Your 64 contains two character sets, but only one can be used at any one time. When it is switched on, the 64 is in upper case mode. As you press keys, upper case characters are produced. Pressing the [SHIFT] key and a letter key gives you the graphic character printed to the right on the front of

the key. Pressing the [C =] key and a letter key gives you the graphic character printed to the left on the front of the key.

If you hold down the [SHIFT] and [C =] keys at the same time, you can switch between upper and lower case modes. Lower case mode is a little like using a typewriter: pressing a letter key gives you the lower case version, and pressing [SHIFT] with the letter gives the upper case version of that letter. Pressing the [C =] key and a letter key still gives you the graphic character printed to the left on the front of the key.

You can switch to upper case mode in a program by including PRINT CHR$(142) in a line. To switch to lower case mode, use PRINT CHR$(14).

The graphics characters can be used to draw pictures on the screen, by using quotes to surround the graphics, just as if you were printing words.

Numbers in strings

You have probably seen that there are two ways to print 123 on the screen. You can enter

```
PRINT 123
```

or

```
PRINT "123"
```

Although you see the same figures appearing on the screen, it is important to realise the difference that the 64 sees. 123 is seen as a number, with the value 123. "123" is seen merely as a string of characters. To see the difference, try this as a direct command:

```
PRINT 123 + 123
```

You should get 246 as the answer, because the computer understands these to be numbers. If you enter:

```
PRINT "123" + "123"
```

you will get 123123, because the 64 treats them as strings.

Although they are completely different, numbers can be turned into strings and vice versa. To turn 123 into a string, you can use the STR$ function, eg

```
A$ = STR$(123)
```

The reverse of this is achieved by using the VAL function. Even though strings can't normally be considered to be numbers, they do have a value.

```
PRINT VAL("123") + 1
```

will print 124. If the first character in the string is not a + or a − or a figure, the value is zero.

Simon

In this program, the fact that strings can be concatenated (added together), is used as the basis for a memory game. A string contains the STR$ version of numbers chosen at random from one to eight, and the object is to remember their order. The length of the string increases by one each time a correct answer is entered.

The game of Simon is usually played with colours and sounds as extra memory aids. The 64 stores the code of the colour in which text is printed at location 646 in the memory. If this is POKEd with numbers from 0 to 15, the sixteen available colours can be selected.

The number keys on the 64 have colours printed on them, BLK, WHT, etc. The numbers on the keys are one higher than the code to be POKEd to 646, hence the line that subtracts one from the value of the string.

The sound commands are a little more complicated. They are explained in more detail in the program notes.

```
10 REM ***     SIMON       ***
20 REM
30 REM ***   AJS 5/6/83    ***
40 REM
50 REM
60 REM *** INSTRUCTIONS ***
70 REM
80 PRINT CHR$(147):POKE53281,0
90 PRINT
100 PRINT"    THIS IS A GAME OF MEMORY"
110 PRINT
120 PRINT"  YOU WILL BE SHOWN A SERIES"
130 PRINT
140 PRINT"  OF FIGURES. THESE FIGURES"
150 PRINT
160 PRINT"  WILL SOON DISAPPEAR AND YOU"
```

```
170 PRINT
180 PRINT"  MUST TYPE THEM IN, IN THE"
190 PRINT
200 PRINT"  RIGHT ORDER. THE NUMBER OF"
210 PRINT
220 PRINT" FIGURES WILL INCREASE BY ONE"
230 PRINT
240 PRINT"         EACH TIME."
250 PRINT
260 PRINT"   PRESS SPACE BAR TO GO"
270 GET Z$:IF Z$="" THEN 270
280 PRINTCHR$(147):POKE53281,12
290 REM
300 REM ***  MAIN GAME LOOP  ***
310 REM
320 FOR Z= 1 TO 10
330 X=INT(RND(0)*8)+1
340 X$=STR$(X):X$=MID$(X$,2,1)
350 Y$=Y$+X$
360 GOSUB 490:POKE198,0
370 INPUT"ENTER YOUR GUESS";G$
380 IF G$ = Y$ THEN S=S+1
390 IF G$ <>Y$ THEN Z=11:GOSUB630
400 NEXT Z
410 REM
420 REM ***  SCORE ROUTINE  ***
430 REM
440 PRINT"YOU SCORED"S
450 PRINT"PRESS ANY KEY TO GO AGAIN"
460 GET Z$:IF Z$=""THEN 460
470 RUN
480 REM
490 REM *** COLOUR PRINT SUBROUTINE ***
500 REM
510 PRINT
520 FOR N = 1 TO LEN(Y$)
530 A$=MID$(Y$,N,1):V = VAL(A$)
540 POKE 646,V-1:REM COLOUR VALUE
550 PRINT A$;" ";
560 GOSUB 700
570 NEXT N
580 PRINT
590 FOR PAUSE=1 TO 1000:NEXT PAUSE
600 PRINTCHR$(147):POKE646,1
```

```
610 RETURN
620 REM
630 REM *** FAULT SUBROUTINE ***
640 REM
650 PRINT"MISTAKE!!  IT WAS "Y$
660 RETURN
700 REM
710 REM *** SOUND SUBROUTINE ***
720 REM
730 SC = 54272
740 POKE SC+24,15:REM VOLUME
750 POKE SC+4,33
760 POKE SC+5,10:POKE SC+6,4
770 V=V*2000
780 HIGH = INT(V/256):LOW = V-HIGH*256
790 POKE SC+1,HIGH:POKE SC,LOW
800 FOR PAUSE = 1 TO 400:NEXT PAUSE
810 POKESC+4,0
820 RETURN

READY.
```

Commentary

Line 80 shows another way of clearing the screen — PRINT CHR$(147). The second statement sets the background colour to black.

Lines 100 to 260 contain the instructions.

Line 270 waits until any key is pressed. Line 280 clears the screen and sets the background colour to grey.

Lines 320 to 400 form the loop which provides 10 figures.

Line 330 chooses a number at random between 1 and 8. Line 340 turns the number into a string, and the second statement strips away the first character, a space. Line 350 adds the new figure to the existing string.

Line 360 sends program control to the print routine and clears the keyboard buffer to prevent cheating!

Line 380 checks the input against the string and increments the score by one. Line 390 ends the Z loop if the answer is incorrect, and sends the program to the fault subroutine.

Lines 440 to 470 print the score and start the program running again.

Lines 520 to 570 print the separate figures in the string after their colour codes have been POKEd into memory location 646. Line 550 prints a space between each figure.

Lines 590 and 600 provide a delay of about two seconds before the string disappears.

Line 650 contains the fault subroutine.

Line 730 sets SC equal to the starting address of the 6581 SID sound chip. This makes all the subsequent addressing of locations in this chip easier to follow.

Line 740 sets the volume to maximum. Line 750 sets the total quality of voice 1 to a sawtooth waveform. Line 760 sets the envelope of voice 1 (attack/decay and sustain/release).

Line 770 sets the frequency of the note by multiplying the value of the figure in the string (V), by 2000. The SID chip is capable of producing notes of a frequency that can be stored in two bytes. The highest value that could be stored is therefore 256*256 or 65536. Line 780 calculates what should be POKEd into this two-byte frequency store to produce the required note.

Line 800 controls the length of the notes. Line 810 sets the waveform control register to zero.

Diversity of subject areas

There are many areas of English teaching in which the computer can prove a useful aid. Because of its fast sorting ability and accurate input check, it can be used to help in spelling, alphabetical ordering, punctuation and grammar in general.

It seems a little unfair to divide up the chapters in this book into the traditional subject categories of maths, English, geography, etc, as most teachers in primary schools would agree that such divisions are hard to make. A day spent in looking at a local shopping centre, for instance, might cover such areas as maths — surveys of people entering each shop; geography — the countries that provide the goods; history — how the area has changed over the years; art — drawing the shops; English — writing about the area, etc.

Such a topic-based approach is closer to the child's experience of the world, and subject divisions would be artificial. Certain basic skills still

need to be taught, however, and this book would lack any structure if the programs were not divided up in some manner.

It is with this in mind that the programs in this chapter have been included. They attempt to help the teacher or parent instil basic spelling and comprehension skills.

Hangman

Some children are lucky and have no difficulty in spelling. Others attempt a phonetic approximation, and yet others never seem to grasp the idea. Although poor spelling often goes hand-in-hand with poor reading skills, there are many good readers who reach adulthood and still suffer from difficulties with their spelling. Traditional methods of learning words by heart for a weekly test are no bad thing if the words are relevant to the child's immediate situation, eg if they come from a project or topic that is being studied. If the child uses the words in their proper context, the spelling is reinforced.

There are two approaches here — one is to show the context of the word and to offer a multiple choice of possible spellings. The other approach is to build up the word slowly, with spaces for missing letters. The game of hangman is an ideal vehicle for this approach. The program that follows builds up a picture of *le pendu*, as in the traditional game, and 10 guesses are allowed before the victim is finally throttled.

Words to be selected are held in DATA lines, so there is no problem in changing these to words relevant to a topic, or to those displaying a particular spelling point, such as 'ie' and 'ei' words. You could even use the program to help learn foreign words. The only requirement is that the total number of words should not change, as the READ statement needs 20 words. If you want a larger selection, then the number may be altered as detailed in the program notes.

```
1000 REM ***     HANGMAN    ***
1010 REM
1020 REM ***  AJS  2/8/83   ***
1030 REM
1040 POKE53281,1:POKE646,0
1050 GOSUB6000:REM*** INITIALISATION ***
1060 GOSUB7000:REM***  INSTRUCTIONS  ***
1070 GOSUB2000:REM*** GAME           ***
1080 GOTO 1070
2000 REM ***  GAME SUBROUTINE  ***
2005 REM
```

```
2010 RESTORE
2020 FOR N=1TO INT(RND(0)*20)+1
2030 READ W$
2040 NEXT
2060 PRINT""
2070 X=5:Y=15:GOSUB4000
2080 FORN=1TOLEN(W$)
2090 PRINT"-";
2100 NEXT N
2120 SC=0:Z=0
2130 X=5:Y=20:GOSUB4000
2140 G$="":PRINT"YOUR GUESS";
2145 POKE198,0
2150 INPUTG$:FLAG=1
2155 IFG$=W$ THEN GOSUB3000:RETURN
2160 LN=LEN(W$)
2170 IF LEN(G$)<>LN AND LEN(G$)>1 THEN X=15:Y=20:
     GOSUB4000
2180 IF LEN(G$)<>LN AND LEN(G$)>1 THEN PRINT"
                    "
2190 IF LEN(G$)<>LN AND LEN(G$)>1 THEN GOTO 2130
2200 IF LEN(G$)=1THENX=Z:Y=23:GOSUB4000
2210 IF LEN(G$)=1THENPRINTG$
2220 Z=Z+1
2230 FOR N=1 TO LN
2240 IFG$=MID$(W$,N,1)THEN GOSUB2500
2245 IFG$=MID$(W$,N,1)THEN FLAG=0
2250 NEXT N
2260 IF FLAG <>0 THEN SC=SC+1:GOSUB2600
2270 X=24:Y=0:GOSUB4000
2280 IFSC>0THENPRINT"LIVES LEFT ->"10-SC
2290 IF SC<10 THEN GOTO 2130
2300 Y=24:GOSUB4000
2305 REM  ***  WORD NOT GUESSED  ***
2310 FOR N = 1 TO 10:PRINT"HELP!!!!":PRINT:NEXT
2320 PRINT"THE WORD WAS "W$
2330 PRINT
2335 POKE198,0
2340 PRINT"PRESS THE SPACE BAR TO GO"
2350 GET Z$:IF Z$="" THEN 2350
2360 RETURN
2370 REM
2500 REM ***  PRINTING CORRECT LETTERS  ***
2505 REM
2510 X=4+N:Y=15:GOSUB4000
2520 PRINTG$
2590 RETURN
2595 REM
2600 REM ***  HANGING SUBROUTINE  ***
```

```
2605 REM
2610 POKE646,(SC)ANDSC>2:REM COLOURPRINT
2620 X=38:REM ***  UPRIGHT      ***
2630 IFSC=1 THEN FORY=19TO3STEP-1:GOSUB4000:
     PRINTCHR$(166):NEXT
2640 Y=2: REM ***  CROSSPIECE   ***
2650 IFSC=2 THEN FORX=29TO38:GOSUB4000:PRINTCHR$
     (166):NEXT
2660 X=29:REM ***  ROPE         ***
2670 IFSC=3 THEN FORY=3TO5:GOSUB4000:PRINTCHR$
     (180):NEXT
2720 X=27:Y=5:GOSUB4000:REM FACE *
2730 IF SC=4 THEN PRINTCHR$(100)CHR$(101);
2740 IF SC=4 THEN PRINTCHR$(102)CHR$(103)
2745 X=27:Y=6:GOSUB4000
2750 IF SC=4 THEN PRINTCHR$(104)CHR$(104);
2760 IF SC=4 THEN PRINTCHR$(106)CHR$(107)
2765 X=27:Y=7:GOSUB4000
2770 IF SC=4 THEN PRINTCHR$(108)CHR$(109);
2780 IF SC=4 THEN PRINTCHR$(110)CHR$(111)
2785 X=27:Y=8:GOSUB4000
2790 IF SC=4 THEN PRINTCHR$(112)CHR$(113);
2800 IF SC=4 THEN PRINTCHR$(114)CHR$(115)
2815 X=28:REM ***  BODY   ***
2820 IFSC=5THENFORY=9TO13:GOSUB4000:PRINTCHR$
     (166)CHR$(166):NEXT
2825 X=28:REM ***  L.ARM ***
2830 IFSC=6 THEN FORY=9TO13:X=X-1:GOSUB4000:
     PRINTCHR$(169):NEXT
2835 X=29:REM ***  R.ARM ***
2840 IFSC=7 THEN FORY=9TO13:X=X+1:GOSUB4000:
     PRINTCHR$(127):NEXT
2845 X=29:REM ***  L.LEG ***
2850 IFSC=8 THEN FORY=14TO18:X=X-1:GOSUB4000:
     PRINTCHR$(169):NEXT
2855 X=28:REM ***  R.LEG ***
2860 IFSC=9 THEN FORY=14TO18:X=X+1:GOSUB4000:
     PRINTCHR$(127):NEXT
2980 POKE646,11
2990 RETURN
3000 REM
3010 REM ***  SUCCESS ROUTINE  ***
3020 REM
3030 X=5:Y=15:GOSUB4000
3040 PRINTW$
3050 FOR PAUSE = 1 TO 2000: NEXT PAUSE
3060 X=0:Y=11:GOSUB4000
3070 PRINT"THANKS FOR SAVING ME!"
```

```
3080 PRINT
3090 PRINT"YOU GUESSED "W$
3100 PRINT
3105 POKE198,0
3110 PRINT"PRESS SPACE BAR TO GO"
3120 GET Z$:IF Z$="" THEN 3120
3130 RETURN
4000 REM
4010 REM ***  'PRINT AT' ROUTINE  ***
4020 REM
4030 POKE782,X:POKE781,Y:SYS65520:RETURN
6000 REM
6010 REM ***  MOVE CHARACTER SET  ***
6020 REM
6030 PRINTCHR$(142)
6040 POKE52,48:POKE56,48
6050 POKE56334,PEEK(56334)AND254
6060 POKE1,PEEK(1)AND251
6070 FORN=0TO1023
6080 POKE12288+N,PEEK(53248+N)
6090 NEXT N
6100 POKE1,PEEK(1)OR4
6110 POKE56334,PEEK(56334)OR1
6120 POKE53272,(PEEK(53272)AND240)+12
6130 REM
6140 FORN=1TO20:READZ$:NEXT
6150 FORN=0TO127
6160 READ Z
6170 POKE12288+68*8+N,Z
6180 NEXT N
6190 RETURN
7000 REM
7010 REM ***   INSTRUCTIONS   ***
7020 REM
7030 PRINT""
7050 PRINT"   IN THIS GAME, YOU WILL BE"
7060 PRINT
7070 PRINT"     GIVEN A ROW OF DASHES"
7080 PRINT
7090 PRINT"  EACH DASH STANDS FOR A LETTER"
7100 PRINT
7110 PRINT"  YOU HAVE TO GUESS THE LETTERS"
7120 PRINT
7130 PRINT"   IF YOU ARE RIGHT THE LETTER"
7140 PRINT
7150 PRINT"     WILL APPEAR. IF YOU ARE"
7160 PRINT
7170 PRINT"  WRONG, PART OF THE MAN APPEARS"
7180 PRINT
```

```
7190 PRINT"  AND YOU LOSE A LIFE. IF YOU"
7200 PRINT
7210 PRINT"  THINK YOU KNOW THE WORD YOU"
7220 PRINT
7230 PRINT"    MUST SPELL IT CORRECTLY TO"
7240 PRINT
7250 PRINT"       SAVE THE POOR MAN!!"
7260 PRINT
7270 PRINT"   PRESS THE SPACE BAR TO GO"
7280 GET Z$:IF Z$="" THEN 7280
7290 RETURN
9000 REM ***  DATA FOR WORDS  ***
9010 REM
9020 DATA MONDAY,TUESDAY,WEDNESDAY
9030 DATA THURSDAY,FRIDAY,SATURDAY
9040 DATA SUNDAY,JANUARY,FEBRUARY
9050 DATA MARCH,APRIL,MAY,JUNE,JULY
9060 DATA AUGUST,SEPTEMBER,OCTOBER
9070 DATA NOVEMBER,DECEMBER,YEAR
9080 REM
9090 REM  ***  DATA FOR FACE  ***
9100 REM
9110 DATA 0,0,0,0,3,3,15,15
9120 DATA 15,15,63,63,255,255,255,255
9130 DATA 240,240,252,252,255,255,255,255
9140 DATA 0,0,0,0,192,192,240,240
9150 DATA 63,63,63,63,60,60,63,63
9160 DATA 255,255,63,63,15,15,60,60
9170 DATA 255,255,252,252,240,240,60,60
9180 DATA 252,252,252,252,60,60,252,252
9190 DATA 63,63,63,63,60,60,15,15
9200 DATA 252,252,240,240,48,48,63,63
9210 DATA 63,63,15,15,12,12,252,252
9220 DATA 252,252,252,252,60,60,240,240
9230 DATA 3,3,0,0,0,0,0,0
9240 DATA 207,207,240,240,63,63,15,15
9250 DATA 243,243,15,15,252,252,240,240
9260 DATA 192,192,0,0,0,0,0,0

READY.
```

Commentary

As this is a fairly complicated program, we have inserted a lot of REMs to inform you of what is happening.

Lines 1040–1070 constitute the control module. Line 2000 begins the game subroutine.

Lines 2020–2040 choose a word, W$, at random from data. Line 2070 sets the 'print at' coordinates. Lines 2080–2100 print dashes according to the number of letters in the word. Line 2145 empties the keyboard buffer.

Line 2150 accepts the guess, G$, and sets the variable FLAG to a one. This is only changed to a zero if a correct letter is input, and is used in line 2260 to check if part of the gallows should be drawn.

Line 2155 checks to see if the whole word has been entered. If it has, the program jumps to the winning routine.

Line 2160 sets variable LN to the length of the chosen word, W$.

Lines 2170–2210 check the length of the guessed word against the chosen word. If more than one letter is input, it is ignored, unless it has the same number of letters as W$.

Line 2220 contains the counter for the number of incorrect guesses, Z.

Lines 2230–2250 check each letter of W$ to see if it contains the guessed letter, G$. If any letter is the same, subroutine 2500 prints it in the correct place. The flag is also changed to prevent part of the gallows being built.

Line 2270 and 2280 print the number of lives remaining at the top of the screen.

Line 2290 sends the program back to the start of the guess loop if less than 10 incorrect guesses have been made. On loop 10, the program continues.

Line 2500 is a short routine to print a letter in the correct place. It uses the loop counter N, to tell it where to put the letter.

Lines 2600–2990 contain the rather large drawing routine to construct the gallows and the man.

Line 2610 uses the number of incorrect guesses, SC, to control the printing colour.

Lines 2620 and 2630 use a loop to draw the upright post. CHR$(166) is a graphics character like a tiny chessboard.

Lines 2640 and 2650 use another loop to draw the crosspiece. GOSUB 4000, to the 'print at' routine, ensures that the next print position is controlled by the X value in the loop counter.

Lines 2730–2800 draw the head. This is made up from defined characters held in DATA statements. The 'print at' routine is again used to control the position of the head.

Lines 2815–2860 use the same principle to draw the rest of the body according to the value of the variable SC.

Line 2980 POKEs the printing colour back to grey. The success routine also uses the 'print at' routine to print the correct word and a thank-you message.

Line 4030 contains the short 'print at' routine where the new X and Y coordinates are POKEd into memory. Without this routine, the effects in a program like this would have been much more difficult to achieve.

Line 6000 starts the character-moving routine. Characters cannot be redefined on the 64 unless the character ROM is copied into RAM.

This is exactly the same structure as that in the program BigL, so refer to those notes for more detail.

Lines 6140–6180 redefine the 16 characters from number 68 onwards. Note that this number is not the same as the ASCII code and, when the characters are printed, 32 must be added to the number of the character before it can be used after a CHR$.

Lines 7000–7290 contain the instructions.

Lines 9000 onwards contain all the data for the program. It is divided into two sections, the first containing the words to be used, and the second the data for the redefined graphics used to draw the face.

Anagram

It is hoped that the programs that are included in this book reflect to some degree the belief that children do not come to school simply to acquire knowledge that may or may not be useful in the future. Children need to learn many skills at primary age, and, apart from the most important (social skills), thinking skills and the ability to research and store information must rank high in the list.

Although spelling is important in its own right, the child who has difficulty with spelling will be at a disadvantage when it comes to finding information from reference books. There is little point in giving a child an

encyclopedia and saying 'Find out all you can about beer production in Belgium' if the child cannot spell Belgium and does not know how to find the information in the first place. Information from a book may not be as good as firsthand experience (especially in the example in question!), but it is better than no experience at all.

The following program can be used as a spelling aid, or in any other subject area to familiarise children with new words. Anagram allows the parent or teacher to load data lines with a fresh vocabulary. The child is presented with a scrambled version of the word on the screen and has to input the word, spelt correctly. Points are gained for getting the word right. The final score is shown after 10 words have been displayed.

```
100 REM
1010 REM ***  ANAGRAM GAME ***
1020 REM
1030 REM ***  AJS  4/10/83 ***
1040 REM
1050 GOSUB 1500:REM INSTR
1060 FOR TRY = 1 TO 10
1070 GOSUB 2000:REM GET WORD
1080 GOSUB 3000:REM MIX WORD
1090 GOSUB 4000:REM GAME
1100 NEXT TRY
1110 GOSUB 5000:REM SCORE
1120 STOP
1499 REM
1500 REM ***  INSTRUCTIONS  ***
1510 REM
1520 POKE53281,1:POKE 646,0
1530 PRINTCHR$(147)
1540 PRINT
1550 PRINT"  YOU WILL BE SHOWN A NUMBER"
1560 PRINT
1570 PRINT"     OF JUMBLED UP WORDS"
1580 PRINT
1590 PRINT"  YOU CAN ASK FOR THEM TO BE"
1600 PRINT
1610 PRINT"  MIXED UP AGAIN OR YOU CAN"
1620 PRINT
1630 PRINT"  GUESS WHAT YOU THINK THE"
1640 PRINT
1650 PRINT"       WORD REALLY IS"
1660 PRINT
1670 PRINT"   AFTER TEN WORDS YOU WILL"
1680 PRINT
1690 PRINT"      BE SHOWN YOUR SCORE"
1700 PRINT
```

```
1710 PRINT"  PRESS THE SPACE BAR TO GO"
1720 GET Z$:IF Z$="" THEN 1720
1730 RETURN
1990 REM
2000 REM ***     GET WORD     ***
2010 REM
2020 RESTORE
2030 FOR N = 1 TO INT(RND(0)*21)+1
2040 READ WD$
2050 NEXT
2060 RETURN
2990 REM
3000 REM ***     MIX WORD     ***
3005 REM
3010 DUMMY$=WD$
3020 NW$=""
3030 FOR N = 1 TO LEN(DUMMY$)
3040 IF RND(0)>=.5THEN NW$=NW$+MID$(DUMMY$,N,1):
     GOTO3060
3050 NW$=MID$(DUMMY$,N,1)+NW$
3060 NEXT N
3070 DUMMY$=NW$
3080 IF NW$=WD$ THEN 3020
3090 RETURN
3990 REM
4000 REM ***     DISPLAY     ***
4010 REM
4020 PRINTCHR$(147)
4030 PRINT
4040 PRINT"YOUR JUMBLED LETTERS ARE:-"
4050 PRINT
4060 PRINT NW$
4070 X1=PEEK(782):Y1=PEEK(781)
4080 X=5:Y=22:GOSUB6000
4090 POKE198,0
4100 PRINT"M TO MIX, G TO GUESS?"
4110 GET Z$
4120 IF Z$<>"M" AND Z$<>"G" THEN 4110
4130 X=X1:Y=Y1:GOSUB 6000
4140 IF Z$="M" THEN GOSUB 3000
4150 IF Z$="M" THEN GOTO 4020
4155 X=5:Y=22:GOSUB6000
4160 INPUT"ENTER YOUR GUESS --->";G$
4170 IFG$ = WD$ THEN GOSUB4500
4190 IFG$ <>WD$ THEN GOSUB4600
4200 RETURN
4500 REM
4510 REM ***  SUCCESS ROUTINE  ***
4520 REM
```

```
4530 PRINT "YES": SC = SC+1
4540 POKE53281,5
4550 FOR PAUSE = 1 TO 2000:NEXT PAUSE
4560 POKE53281,1
4570 RETURN
4600 REM
4610 REM *** MISTAKE ROUTINE ***
4620 REM
4630 PRINT "NO, "WD$
4640 POKE53281,2
4650 FOR PAUSE = 1 TO 2000:NEXT PAUSE
4660 POKE53281,1
4670 RETURN
5000 REM
5010 REM ***  SCORE ROUTINE  ***
5020 REM
5030 PRINTCHR$(147)
5040 PRINT
5050 PRINT"YOUR SCORE WAS "SC"WORDS RIGHT"
5060 IF SC = 10 THEN PRINT "WELL DONE"
5070 IF SC>6 AND SC<10 THEN PRINT "FAIR"
5080 IF SC<7 THEN PRINT "LEARN THEM!!"
5090 RETURN
5990 REM
6000 REM ***  'PRINT AT' ROUTINE ***
6010 REM
6020 POKE782,X:POKE781,Y:SYS65520:RETURN
8990 REM
9000 REM ***    DATA STATEMENTS    ***
9010 REM
9020 REM ***  ENTER YOUR OWN WORDS ***
9030 REM
9040 REM ***  BUT MAKE SURE THERE  ***
9050 REM
9060 REM ***       ARE TWENTY      ***
9070 REM
9080 DATA AARDVARK,BABOON,CAMEL,DONKEY
9090 DATA ELEPHANT,FLAMINGO,GOOSE,HORSE
9100 DATA IBEX,JACKAL,KANGAROO,LEMUR
9110 DATA MONKEY,NEWT,OSTRICH,QUAIL
9120 DATA RABBIT,STORK,TIGER,ZEBRA

READY.
```

Commentary

Lines 1050–1120 form the control module. Lines 1060–1100 contain the main loop which selects ten words to guess.

Lines 1500–1730 contain the instructions. Line 1520 sets the background colour to white, and the printing colour to black. Line 1530 clears the screen. Line 1720 waits for a key to be pressed before the program continues.

Lines 2000–2060 get words at random from the DATA lines. If you have more than twenty words in DATA, you will have to increase the size of the loop in line 2030. RESTORE in line 2020 sets the data pointer back to the beginning of the list of words each time.

Lines 3000–3090 mix the letters of the word. The loop in lines 3030–3060 goes through the letters of the chosen word one at a time. It builds a new word by attaching a letter at random to either the front or back of the new word.

Line 3070 uses DUMMY$ to hold a copy of the new word. Line 3080 checks to ensure that the new word is not the same as the original!

Lines 4000–4200 control the display of the game. Line 4020 clears the screen. Lines 4040–4060 print the jumbled word.

Line 4070 saves the print coordinates in variables X1 and Y1. Line 4080 resets the print position to the bottom of the screen by jumping to the 'print at' subroutine. Line 4090 clears the keyboard buffer.

Lines 4100–4120 allow the input of either M or G. If M is pressed, the program jumps back to the mix subroutine. If G is pressed, the program continues.

Line 4155 sends the print coordinates for the bottom of the screen to the 'print at' subroutine. Lines 4160–4190 check the guess against the actual word and send control to the particular subroutine.

Lines 4500–4570 contain the success routine, which simply increases the score by one and temporarily makes the background green in line 4540.

Lines 4600–4670 contain the mistake routine. This turns the background colour to red.

Lines 5000–5090 clear the screen and print the score.

Line 6020 is the 'print at' subroutine, which sends the coordinates of the next print position to the operating system kernal.

Lines 9000–9120 contain all the words used. Any twenty words can be used here, but avoid those with more than 10 letters, as they will extend on to the next line.

CHAPTER 4
Making Friends with your Keyboard

Tackling entry errors — menus — making problems foolproof — moving the character set — BigL: writing large on the screen — Keyboard: learning the keys — Masking: changing binary numbers.

A computer can receive information from a variety of sources. These include cassettes, disks, joysticks, light pens, even the telephone line and, of course, the keyboard. In a program, the most common methods of entering data are via INPUT and GET statements.

You have already seen INPUT used on several occasions. Program execution halts until some information is entered by pressing the [RETURN] key. GET does not halt the program, so it's sensible to put it in a loop to wait until a key is pressed. When you press a key, the code of that key is stored in the keyboard buffer, a type of temporary memory store. To make sure that only the input you want is left in the buffer, you can clear it by POKE 198,0 just before the GET loop.

It is also quite easy to ensure that any input is within the limits you set. A program that only requires figures can use GET and check the code of keys that are pressed. If any other key is pressed, the program can loop back to request another input. This program does just that:

```
100 REM ********************************
110 REM SELECTIVE INPUT ROUTINE
120 REM ********************************
130 POKE198,0
140 GET A$:IF A$ = "" THEN 140
150 IF ASC(A$)<48 OR ASC(A$)>57THENPRINT "NOT A
    NUMBER"
155 IF ASC(A$)<48 OR ASC(A$)>57THEN GOTO 130
160 PRINT "YOU PRESSED NUMBER "A$
170 STOP

READY.
```

Line 150 checks the ASCII code to make sure it lies between 48 (0) and 57 (9).

A routine like this can also make a program easier for the user. One of the most frustrating things for an inexperienced user is for error messages to appear on the screen when they are half-way through the program. Shielding the user from the operating system is an important area of programming, for, even if a program works perfectly, it can be spoiled by lack of foresight on the part of the programmer.

Firstly, the program needs to be self-explanatory, as far as that is possible. If someone puts a cassette in and presses SHIFT and RUN/ STOP, instructions on use should be the first thing that appears after the title. If the program is easy and pleasant to use, it is usually termed 'user- friendly'. If the program has several options, it is a good idea to make it menu-driven to increase its friendliness. This means that a list of options appears on the screen, and the user can make his selection from the displayed menu, eg

Press a number to choose:

1. Instructions
2. Easy Problems
3. Harder Problems
4. Difficult Problems

If any other key apart from 1, 2, 3 or 4 is pressed, the computer should ignore it. This can be achieved by using a GET routine as explained earlier. Another advantage is that the user does not have to press the RETURN key.

In a program requesting the date of birth, INPUT can be used instead of GET, but the manner in which the data is entered should be shown by an example, then the input checked by the program.

```
1000 REM BIRTHDATE SUBROUTINE
1010 PRINT"ENTER YOUR DATE OF BIRTH"
1020 PRINT"USE THE FOLLOWING ORDER - "
1030 PRINT"DATE, MONTH, YEAR"
1040 PRINT"E.G. 4TH MARCH 1956 WOULD BE"
1050 PRINT"     4< RETURN> "
1060 PRINT"     3< RETURN> "
1070 PRINT" 1956< RETURN> "
1080 PRINT"DATE - ";
1090 INPUT DATE
1100 IF DATE < 1 OR DATE > 31 THEN 1090
1110 PRINT"MONTH - ";
1120 INPUT MONTH
```

```
1130 IF MONTH < 1 OR MONTH > 12 THEN 1120
1140 PRINT"YEAR - ";
1150 INPUT YEAR
1160 IF YEAR < 1880 OR YEAR > 1984 THEN 1150
```

The example suggests the entry format, so the user knows the correct order of input, and also that he is expected to enter figures and not words.

If impossible dates, such as −4/15/1983, are entered, they are outside the permitted range of inputs. If letters are entered, eg APRIL instead of 4, the 64 generates an error message — REDO FROM START — so the input loop is repeated once more. If the RETURN key is pressed with nothing entered, the loop is also repeated.

The way to make the routine check that only figures are entered would be to make the input lines contain string, rather than numeric, variables, and check their ASCII codes. This would be a useful exercise for you to try.

In this chapter, we are including some programs that help children find their way round the keyboard. They also introduce ideas that you may find useful in your own programs.

Keyboard encourages fast entry of all the characters on the keyboard. The instructions are presented as a subroutine and the TIME function in the machine starts when the space bar is pressed. A large version of the character appears on the screen and three attempts are allowed to find the correct key.

In the 64, the character definitions are stored at memory location 53248 onwards. Each character on the screen consists of eight rows of eight positions, like a chess board. Each row is represented by a binary number. If you imagine a black character on a white background, then ones stand for blacks, and zeros for whites, so a totally filled-in character would contain 255 in each row, as the binary code for 255 is 11111111.

On some computers, it would be easy to find the character definitions by PEEKing the locations where they are stored. The 64, however, uses a system that is more complicated, in order that more is available to the user. It reads its character set from ROM which shares the same addresses as user memory and some input/output routines. Every time the 64 needs to display a character, a pointer in another part of the memory sends it to the character ROM. If we want to read the definitions, or indeed to change the set to display our own definitions, the easiest thing is to copy the entire character set into user RAM (which can be altered). The 64 must also have the pointer changed so that it points to the new character set.

The following program copies the character set from ROM into RAM at location 12288. If this is all that is done, your BASIC program may overwrite the new set at a later stage. The 64 must be told that it cannot use this space, so the pointers to string variable space and the highest address used by BASIC have to be changed. Then the keyscan interrupts have to be

turned off and the 64 told to look at the character ROM rather than RAM. At this point, all the necessary characters in the ROM can be scanned and their definitions POKEd into their new locations in RAM.

If you were to halt the program at this point, you would find that the keyboard had 'died'. Even the RUN/STOP and RESTORE method would not retrieve your program, as the keyboard is not being scanned. With the characters in their new location, the thing to do is first to switch in the input/output routines, then to start the keyboard scan routine. Now that the character set is in user RAM, you can look at it easily, or alter it as necessary.

Once the character definitions are available, it is possible to PEEK them, and use the codes to produce large versions of them on the screen.

The way this is achieved is by storing the definitions in arrays that hold the code for each line as a binary number; so 170 is stored as its binary equivalent, 10101010. This number is now scanned. If a 1 is found, a solid block is printed; if a 0 is found, a space is printed. This results in a character sixty-four times normal area being printed on the screen. The actual program runs through the entire character set and prints the value held in each location so you can see how the character is formed.

BigL

```
1000 REM *** LARGE CHARACTER PROGRAM ***
1010 REM
1020 REM *** MOVE CHARACTER SET >RAM ***
1030 REM
1040 PRINTCHR$(142)
1050 POKE52,48
1060 POKE56,48
1070 CLR
1080 POKE56334,PEEK(56334)AND254
1090 POKE1,PEEK(1)AND251
1100 FORI=0TO511
1110 POKEI+12288,PEEK(I+53248)
1120 NEXT
1130 POKE1,PEEK(1)OR4
1140 POKE56334,PEEK(56334)OR1
1150 REM
1160 REM *** PRINT LARGE LETTERS ***
1170 REM
1180 Z=12288
1190 POKE53281,0
1200 DIMB(8):DIMC(8,8)
1220 FORX=0 TO 511 STEP 8
1230 PRINT"[SHIFT/CLR/HOME]"
```

```
1240 FORN=0TO7
1250 PRINTZ+X+N,PEEK(Z+X+N)
1260 B(N)=PEEK(Z+N+X)
1270 FORJ=0TO7
1280 C(N,J)=INT(B(N)/2^(7-J))
1290 B(N)=((B(N)/2^(7-J))-C(N,J))*2^(7-J)
1300 IFC(N,J)=1THENPOKE(1064+N*40)+15+J,81:POKE
     (55336+N*40)+15+J,0
1310 NEXTJ
1330 NEXTN
1340 NEXTX

READY.
```

Commentary

Line 1040 clears the screen. Lines 1050 and 1060 tell the 64 where to store strings and the BASIC program so that it won't overwrite the new character set.

Line 1070 frees all available memory space. Line 1080 turns off the keyboard scan. Line 1090 selects the character ROM. Lines 1100 to 1120 copy the contents of the first section of the character ROM into RAM. Line 1130 deselects the character ROM. Line 1140 turns on the keyboard scan. Line 1180 sets Z equal to the start of the location of the copied character set. Line 1190 sets the background colour to green. Line 1200 DIMensions space for the arrays necessary to hold large versions of the character definitions.

Lines 1220–1340 contain the loop to go through the entire 64 copied characters. Lines 1240–1330 contain the loop to display each line of the character.

Lines 1250 and 1260 store the decimal (normal) value of each line in a variable, B(N).

Lines 1270 to 1310 contain a routine to convert the number in B(N) to its binary equivalent. This is stored as a series of zeros and ones in array C(N,J).

It can be seen, if you run this program, that it provides an introduction to how a keyboard trainer program operates. In the next program, the characters are printed in a large version on the screen. These are presented to the

user in a random fashion, and they have to input the correct letter from the keyboard as fast as they can. If they are successful, another letter appears and the time taken is added to their total time.

Keyboard

```
100 REM ***  KEYBOARD TRAINER  ***
110 REM
120 REM ***  A.J.S.   4/11/83  ***
130 REM
140 REM ***************************
150 REM
190 GOTO 1000:REM MOVE CHARACTER SET
200 GOSUB3000:REM INSTRUCTIONS
210 GOSUB4000:REM TEST MODULE
220 GOSUB5000:REM SCORE MODULE
230 GOTO 200
240 REM
1000 REM LARGE CHARACTER PROGRAM
1010 REM
1020 REM RELOCATE CHARACTER SET IN RAM
1030 REM
1040 PRINTCHR$(147)
1050 POKE52,48
1060 POKE56,48
1070 CLR
1080 POKE56334,PEEK(56334)AND254
1090 POKE1,PEEK(1)AND251
1100 FORI=0TO511
1110 POKEI+12288,PEEK(I+53248)
1120 NEXT
1130 POKE1,PEEK(1)OR4
1140 POKE56334,PEEK(56334)OR1
1150 GOTO 200
1990 REM
2000 REM ROUTINE TO PRINT LARGE LETTERS
2010 REM
2020 Z=12288
2025 R=INT(RND(0)*26)+1
2060 PRINTCHR$(147)
2070 FORN=0TO6
2090 B(N)=PEEK(Z+N+R*8)
2100 FORJ=0TO7
2110 C(N,J)=INT(B(N)/2^(7-J))
2120 B(N)=((B(N)/2^(7-J))-C(N,J))*2^(7-J)
2130 IFC(N,J)=1THENPOKE(1064+N*40)+15+J,81
2140 IFC(N,J)=1THENPOKE(55336+N*40)+15+J,0
```

```
2150 NEXTJ
2160 NEXTN
2170 RETURN
3000 REM
3010 REM ***  INSTRUCTIONS  ***
3020 REM
3030 PRINTCHR$(147):POKE53281,5:POKE646,6
3040 PRINT
3050 PRINT"      YOU ARE ABOUT TO BE SHOWN"
3060 PRINT
3070 PRINT"     10 LETTERS AND OTHER SYMBOLS"
3080 PRINT
3090 PRINT"     THAT ARE ALL ON THE KEYBOARD"
3100 PRINT
3110 PRINT"   FIND THE CORRECT KEY AND PRESS"
3120 PRINT
3130 PRINT"       IT AS QUICKLY AS YOU CAN"
3140 FOR X = 1 TO 8: PRINT : NEXT X
3150 PRINT"     PRESS THE SPACE BAR TO GO"
3160 GET Z$:IF Z$="" THEN 3160
3170 RETURN
4000 REM
4010 REM ***  TEST MODULE  ***
4020 REM
4030 TS=TIME:SC=0
4040 FOR G= 1 TO 10
4050 GOSUB2000
4060 POKE782,5:POKE781,20:SYS65520
4070 PRINT"THE LETTER IS------->";
4080 POKE198,0
4090 GET Z$:IF Z$="" THEN 4090
4100 PRINT Z$;
4110 IF ASC(Z$)=R+64 THEN SC=SC+1
4120 IF ASC(Z$)=R+64 THEN PRINT" YES"
4130 IF ASC(Z$)<>R+64 THEN PRINT" NO, "CHR$(R+64)
4140 FOR PAUSE=1 TO 1000:NEXT
4150 NEXT G
4160 TT=INT((TIME-TS)/60)
4170 RETURN
5000 REM
5010 REM ***  SCORE MODULE  ***
5020 REM
5030 PRINTCHR$(147)
5040 PRINT
5050 PRINT"YOU SCORED "SC" CORRECT"
5060 PRINT
5070 PRINT"AND TOOK "TT" SECONDS"
5080 PRINT
5090 PRINT"PRESS SPACE BAR TO GO AGAIN"
```

```
5100 GET Z$:IF Z$="" THEN 5100
5200 RETURN

READY.
```

Commentary

Lines 190–230 contain the control module.

Line 230 loops the program through the instructions and the test for as many times as is necessary.

In order that the character set can be examined, it is copied from ROM in the same manner as that used in the last program.

Lines 1000–1150 are responsible for the moving. When you RUN the program, you will have to wait for about ten seconds while this is achieved. The subroutine is the same as the routine in BigL, so refer to that program for more detailed notes.

Note that this is not a subroutine. If GOSUB 1000 is used in line 190, and RETURN in line 1150, the program will stop with an error message in line 1150 (that there is a RETURN with no GOSUB). This is because the CLR in line 1070 resets the stack containing the RETURN addresses.

The routine from line 2000 to line 2170 is also similar to that in BigL, and is used to print a large version of the selected letter on the screen.

Line 2025 chooses this letter at random from the data.

Lines 3000–3170 contain the instructions.

Lines 4000–4170 contain the test module.

Line 4030 reads the current internal clock time into variable TS, and sets the score counter, SC, to zero.

Lines 4090 and 4100 check the keyboard for an input, and lines 4110–4130 check this input against the selected letter. If it is correct, the score is incremented and YES printed.

Line 4140 provides a short pause so that the screen can be read.

After ten attempts, line 4160 is used to calculate how many ticks have been produced by the jiffy clock, and this number is divided by 60 and rounded off to produce the number of seconds taken.

Lines 5000–5200 contain the score module which simply clears the screen (in line 5030) and prints the details of the test on the screen.

Masking

In the last program, and in one or two others, you may have found a particular feature in a line that seemed to make little sense. A typical line might be

```
POKE 56334,PEEK(56334)AND 254
```

It looks as though it means 'POKE a number with the number already there plus 254'. This is not true, however, as AND is a logical operator, not the same as +.

Logical operators are fairly easy to understand when they are used to compare numbers.

```
10 INPUT"A=";A
20 INPUT"B=";B
30 IF A=1 AND B=1 THEN PRINT "YES": GOTO 50
40 PRINT "NO"
50 STOP
```

AND TRUTH TABLE

A	B			
0	0	0	ie	FALSE
0	1	0	ie	FALSE
1	0	0	ie	FALSE
1	1	1	ie	TRUE — "YES" printed

This program would print 'YES' if both conditions were true. If either or both of them were false, it would print 'NO'.

```
10 INPUT"A=";A
20 INPUT"B=";B
30 IF A=1 OR B=1 THEN PRINT 'YES': GOTO 50
40 PRINT "NO"
50 STOP
```

OR TRUTH TABLE

A	B			
0	0	0	ie	FALSE
0	1	1	ie	TRUE
1	0	1	ie	TRUE
1	1	1	ie	TRUE

This program would print 'YES' if either of the two conditions were true and the other were false, or if both of the conditions were true. If both the conditions were false, then 'NO' would be printed.

When logical operators are used to compare numbers, which they can in the range −32768 to 32767, then they compare the way the numbers are stored in two bytes.

Bytes store numbers using binary digits, ie using only ones and zeros.

DECIMAL	BINARY
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000

If two numbers are compared using AND, the individual bits of each number are compared one by one, and the results are the same as in the truth tables above.

Suppose the two numbers are 23 AND 254. 23 is 00010111 in binary, and 254 is 11111110. Placing the numbers on top of each other and using AND, we get:

```
00010111
11111110
--------
00010110
```

You will notice, in the result line, the only bit to have altered is the last one on the right. If the two numbers to be compared are 19 OR 4, the result is:

```
00010011
00000100
--------
00010111
```

The technique of masking relies on this effect, to ensure that just one bit of a number is set to a one or a zero. All the other binary digits remain the same.

In the examples, it can be seen that, to ensure that the third bit from the right is set to one, the number should be ORed with 4 (as 100 is 4 in binary). To alter the fourth bit from the right, the number would have been ORed with 8 (as 1000 is 8 in binary).

To ensure that the last bit on the right is set to zero, the number should be ANDed with 255 – 1, ie 254. This is because the binary reverse of 1 is 254. (Zeros instead of ones, and vice versa.)

One is 00000001 and 254 is 11111110. To alter the fourth bit from the right, the number should be ORed with 255 – 8, ie 247. This is because the binary reverse of 8 is 247. 8 is 00001000 and 247 is 11110111.

Why all this complexity to achieve such a small thing, you may wonder. The reason is that, in order to use its memory carefully, several functions in the 64 are controlled by single bits in a memory location, rather than by the whole byte. There are several examples of this method being used in the graphics and sound chips. For instance, location 53269 controls whether the graphic characters called sprites are turned on. The eight bits of that location each control one sprite. In a program, you might want to turn one on and leave the others unaltered. The best way to do this is by ORing location 53269; eg to turn on the first sprite, without affecting the others, you would enter:

```
POKE 53269,PEEK(53269)OR 1
```

To turn it off without affecting the others you would use:

```
POKE 53269,PEEK(53269)AND(255 – 1)
```

CHAPTER 5
Story Time

Creativity — artificial intelligence — words at random — Story: a program to help with creative writing and adjective use — interactive use of computers — Gosh: more stories.

Can a computer be creative? To answer this question would take too long in discussing the meaning of creativity and the future prospects of artificial intelligence. At the end, there might not be any firm conclusions. What would be agreed is that, given sufficient information, computers could shuffle this information and display it in all possible ways, including some that might appear creative and original.

For parents and teachers, the problem with children's creativity is often one of encouraging the child to start writing. Once the initial paragraph has been written, the material begins to flow more easily.

The first program does not do anything that could not be achieved using pencil and paper, but it does produce a result in far less time. Various subjects and locations are held in DATA lines. If the user does not like the combination selected, the program produces more at random when a key is pressed.

The READ statement is a very useful feature of the BASIC language. Instead of entering all the data during the running of a program, it can be stored in DATA lines, usually placed at the end of the program. When data is required, the statement READ is used, followed by a numeric or string variable. When the program is RUN, a pointer moves along the DATA lines, each time they are called. During the course of the program it is possible to send this pointer back to the start of the DATA lines by using the command RESTORE.

Enter this program if you aren't sure about using READ, DATA and RESTORE.

```
10 READ A
20 READ X$
30 READ B
40 RESTORE
50 READ C,Y$,D
```

```
60 PRINT A,X$,B,C,Y$,D
70 DATA 1,HELLO,37.5
```

Some BASICs allow you to RESTORE to a particular program line, but in Commodore BASIC you have to resort to tricks if you want to use data from the middle of DATA lines. For instance, to reach the fourth piece of DATA you could have:

```
10 FOR LOOP = 1 TO 4
20 READ A
30 NEXT LOOP
40 PRINT A
50 DATA 1,2,3
60 DATA 4,5,6
```

Data can occupy several lines, and often looks neater in listings presented in this way. The 64 sees it as one big block of data. Be careful you don't attempt to READ words into numeric variables, or you will get an error message.

Story

In the following program, the child is requested to enter adjectives to describe the subject and the location. On completion, the program produces the opening paragraph of a story written around the data that has been entered.

In terms of program complexity, this does not rank very highly, but it does show the way in which apparently original text can be generated. It also requires a lot of thought on the part of the child to enter suitable descriptions that fit the mood of the opening sentence.

```
1000 REM ***      STORY      ***
1010 REM
1020 REM ***  AJS 9/10/83   ***
1025 REM
1040 GOSUB3000:REM GET WORDS
1050 GOSUB4000:REM PRINT TEXT
1060 GOTO 1040:REM LOOP PROGRAM
2020 REM
3000 REM
3010 REM ***  GET WORDS      ***
3020 REM
3030 RESTORE
3040 FORN=1 TO INT(RND(0)*5)+1
```

```
3050 READ START$
3060 NEXT
3070 RESTORE
3080 FORN=1 TO 5+INT(RND(0)*20)+1
3090 READ PERS$
3100 NEXT
3110 RESTORE
3120 FORN=1 TO 25+INT(RND(0)*20)+1
3130 READ PLACE$
3140 NEXT
3990 RETURN
4000 REM
4010 REM ***  WRITE INTRO  ***
4020 REM
4030 PRINTCHR$(147):PRINT"*YOUR STORY*":PRINT
4040 PRINT START$;" ";
4050 PRINT PERS$
4060 PRINT"WAS SEEN NEAR"
4070 PRINT PLACE$
4080 PRINT
4090 PRINT"ENTER 5 ADJECTIVES THAT BEST DESCRIBE"
4100 PRINTPERS$
4110 FORN=1 TO 5
4120 INPUT PADJ$(N)
4130 NEXT N:PRINT
4140 PRINT"ENTER 5 ADJECTIVES THAT BEST DESCRIBE"
4150 PRINTPLACE$
4160 FORN=1 TO 5
4170 INPUT LA$(N)
4180 NEXT N:PRINT
4190 PRINT"PRESS SPACE BAR TO CONTINUE"
4200 GETZ$:IFZ$="" THEN 4200
4210 PRINTCHR$(147)
4220 PRINTSTART$;" ";PERS$
4230 PRINT"WAS SEEN NEAR "PLACE$
4240 PRINT"IF YOU HAVE EVER SEEN "PERS$
4250 PRINT"YOU WILL KNOW THEY ARE "PA$(1)
4260 PRINT"AND "PA$(2)
4270 PRINT"THIS ONE WAS ALSO RATHER "PA$(3)
4280 PRINT"AND DIDN'T WANT TO ENTER "
4285 PRINTPLACE$
4290 PRINT"WHERE IT WAS "LA$(1)" AND "LA$(2)
4300 PRINT"CERTAINLY IT WAS NO PLACE FOR"
4310 PRINTPERS$
4320 PRINT"SOME CREATURES ENJOY PLACES THAT ARE"
4330 PRINTLA$(3)" AND "LA$(4)
4340 PRINT"HOWEVER, AN ADVENTURE HAS TO START"
4350 PRINT"SOMEWHERE, SO SOON THE "PA$(5)
4360 PRINT"FIGURE OF "PERS$
```

```
4370 PRINT"COULD BE SEEN ABOUT TO ENTER"
4380 PRINTPLACE$" WHICH APPEARED"
4390 PRINT"EVEN MORE "LA$(5)" THAN BEFORE"
4400 PRINT:PRINT
4410 PRINT"PRESS SPACE BAR TO GO AGAIN"
4420 GET Z$:IF Z$="" THEN 4420
4990 RETURN
8990 REM
8995 REM *** OPENINGS ***
9000 DATA ONCE UPON A TIME,A LONG TIME AGO
9010 DATA IN LONG AGES PAST,MANY YEARS AGO
9020 DATA ABOUT 200 YEARS AGO
9025 REM *** PEOPLE   ***
9030 DATA A VERY OLD MAN,A YOUNG GIRL
9040 DATA A STRANGE FIGURE,A SHAMBLING VISION
9050 DATA A BEAUTIFUL PRINCESS,A WICKED MAGICIAN
9060 DATA A FIERCE DRAGON,A DARK STRANGER
9070 DATA A HANDSOME PRINCE,A HOODED FIGURE
9080 DATA AN EVIL WITCH,A MYSTERIOUS VISITOR
9090 DATA A LOATHSOME MONSTER,A TERRIFYING SHAPE
9100 DATA A SLIMY TOAD,A MAGIC SNAKE
9110 DATA AN ENORMOUS WORM,A REPULSIVE ANIMAL
9120 DATA AN ELEPHANT,A MOUSE
9130 REM *** PLACES   ***
9140 DATA A DARK FOREST,A DEEP CAVERN
9150 DATA AN OLD CASTLE,A SHADED RAVINE
9160 DATA A DEEP HOLE IN THE GROUND,A HIDDEN MINE
9170 DATA A DESERT ISLAND,A TINY VILLAGE
9180 DATA A HAUNTED HOUSE,AN EERIE MANSION
9190 DATA A DANGEROUS MARSH,A LARGE COUNTRY HOUSE
9200 DATA A MYSTIC STONE CIRCLE,AN ARABIAN BAZAAR
9210 DATA A STEAMING JUNGLE,A DESERTED TEMPLE
9220 DATA A LOST CITY,AN EMPTY TOWN
9230 DATA A BURNING DESERT,A STEEP-SIDED CANYON

READY.
```

Commentary

Lines 1040 to 1060 contain the control module. Line 1060 loops the program back to the start for a new story.

Lines 3000 to 3990 choose words at random. Line 3040 chooses an introduction from the first five phrases held in data. Line 3080 misses the first five items of data and chooses a character at random from the next 20 items. Line 3120 misses the first 25 items of data and chooses a place at random from the last 20 items.

Lines 4000 to 4200 print the chosen introduction and request the input of suitable adjectives to describe the character and the place.

Line 4220 onwards prints a framework of text in which the adjectives are combined to make the first paragraph of a story.

Lines 9000 onwards contain the data lines. The data may be changed provided the number of items in each section remains the same.

Gosh

After the last program, the time has come for a serious program to demonstrate the potential of the computer for creating original masterpieces. Unfortunately, nothing of such high calibre was available!

We could attempt to disguise this program in educational language, to make it seem highly motivating and wonderful at increasing cognitive skills. When it comes down to it, however, it's really just a lot of fun.

Gosh is a computerised version of Consequences, and expects inputs of people, sayings, places and results. It contains some items in data, and what it lacks in educational terms should be made up for by the amusement it causes.

```
1000 REM GOSH - PAT HALL, JAN '84
1010 REM
1020 GOSUB 1150 : REM INITIALISATION
1030 GOSUB 1250 : REM MALE NAMES
1040 GOSUB 1390 : REM FEMALE NAMES
1050 GOSUB 1530 : REM PLACES
1060 GOSUB 1670 : REM HIS SPEECH
1070 GOSUB 1810 : REM HER REPLY
1080 GOSUB 1950 : REM THE CONSEQUENCE
1090 GOSUB 2090 : REM COMBINE AT RANDOM
1100 GOSUB 2430 : REM PAUSE
1110 GOTO 1090
1120 END
1130 REM
1140 REM INITIALISATION
1150 DIM M$( 15 ) : DIM F$( 15)
1160 DIM P$( 15 ) : DIM S$( 15 )
1170 DIM R$( 15 ) : DIM C$( 15 )
1180 REM WHITE SCREEN
1190 POKE 53280, 1 : POKE 53281, 1
1200 REM BLUE TEXT
1210 PRINT CHR$( 31 )
1220 RETURN
```

```
1230 REM
1240 REM MALE NAMES
1250 PRINT "[SHIFT/CLR/HOME]"
1260 X = 7 : Y = 5 : GOSUB 2470
1270 PRINT "TYPE IN FIVE BOY'S NAMES"
1280 FOR I = 1 TO 5
1290 Y = 5 + I * 2 : GOSUB 2470
1300 GOSUB 2520
1310 M$( I ) = A$
1320 NEXT I
1330 FOR I = 6 TO 15
1340 READ A$ : M$( I ) = A$
1350 NEXT I
1360 RETURN
1370 REM
1380 REM FEMALE NAMES
1390 PRINT ""
1400 Y = 5 : GOSUB 2470
1410 PRINT "TYPE IN FIVE GIRL'S NAMES"
1420 FOR I = 1 TO 5
1430 Y = 5 + I * 2 : GOSUB 2470
1440 GOSUB 2520
1450 F$( I ) = A$
1460 NEXT I
1470 FOR I = 6 TO 15
1480 READ A$ : F$( I ) = A$
1490 NEXT I
1500 RETURN
1510 REM
1520 REM PLACES
1530 PRINT "[SHIFT/CLR/HOME]"
1540 Y = 5 : GOSUB 2470
1550 PRINT "TYPE IN FIVE PLACES"
1560 FOR I = 1 TO 5
1570 Y = 5 + I * 2 : GOSUB 2470
1580 GOSUB 2520
1590 P$( I ) = A$
1600 NEXT I
1610 FOR I = 6 TO 15
1620 READ A$ : P$( I ) = A$
1630 NEXT I
1640 RETURN
1650 REM
1660 REM HIS SPEECH
```

```
1670 PRINT "[SHIFT/CLR/HOME]"
1680 Y = 5 : GOSUB 2470
1690 PRINT"TYPE IN FIVE THINGS HE SAYS"
1700 FOR I = 1 TO 5
1710 Y = 5 + I * 2 : GOSUB 2470
1720 GOSUB 2520
1730 S$( I ) = A$
1740 NEXT I
1750 FOR I = 6 TO 15
1760 READ A$ : S$( I ) = A$
1770 NEXT I
1780 RETURN
1790 REM
1800 REM HER REPLY
1810 PRINT "[SHIFT/CLR/HOME]"
1820 Y = 5 : GOSUB 2470
1830 PRINT "TYPE IN FIVE REPLIES MADE"
1840 FOR I = 1 TO 5
1850 Y = 5 + I * 2 : GOSUB 2470
1860 GOSUB 2520
1870 R$( I ) = A$
1880 NEXT I
1890 FOR I = 6 TO 15
1900 READ A$ : R$( I ) = A$
1910 NEXT I
1920 RETURN
1930 REM
1940 REM THE CONSEQUENCE
1950 PRINT "[SHIFT/CLR/HOME]"
1960 Y = 5 : GOSUB 2470
1970 PRINT "TYPE IN FIVE CONSEQUENCES"
1980 FOR I = 1 TO 5
1990 Y = 5 + I * 2 : GOSUB 2470
2000 GOSUB 2520
2010 C$( I ) = A$
2020 NEXT I
2030 FOR I = 6 TO 15
2040 READ A$ : C$( I ) = A$
2050 NEXT I
2060 RETURN
2070 REM
2080 REM COMBINE AT RANDOM
2090 PRINT "[SHIFT/CLR/HOME]"
2100 GOSUB 2390
```

```
2110 Y = 7 : GOSUB 2470
2120 PRINT M$( N )
2130 Y = 8 : GOSUB 2470
2140 PRINT "MET"
2150 GOSUB 2390
2160 Y = 9 : GOSUB 2470
2170 PRINT F$( N )
2180 GOSUB 2390
2190 Y = 10 : GOSUB 2470
2200 PRINT P$( N )
2210 Y = 11 : GOSUB 2470
2220 PRINT "HE SAID TO HER:"
2230 GOSUB 2390
2240 Y = 12 : GOSUB 2470
2250 PRINT S$( N )
2260 Y = 13 : GOSUB 2470
2270 PRINT "SHE REPLIED:"
2280 GOSUB 2390
2290 Y = 14 : GOSUB 2470
2300 PRINT R$( N )
2310 Y = 15 : GOSUB 2470
2320 PRINT "AND THE CONSEQUENCE WAS"
2330 GOSUB 2390
2340 Y = 16 : GOSUB 2470
2350 PRINT C$( N )
2360 RETURN
2370 REM
2380 REM PRODUCE RANDOM NUMBER
2390 N = INT( RND( 1 ) * 15 ) + 1
2400 RETURN
2410 REM
2420 REM PAUSE
2430 FOR I = 1 TO 6000 : NEXT I
2440 RETURN
2450 REM
2460 REM PRINT TAB ROUTINE
2470 POKE 782, X : POKE 781, Y
2480 SYS 65520
2490 RETURN
2500 REM
2510 REM CHECK INPUT FOR LENGTH
2520 GOSUB 2470
2530 PRINT "                              ";
2540 PRINT "                            "
```

```
2550 GOSUB 2470
2560 A$ = ""
2570 INPUT A$
2580 IF A$= "" THEN 2520
2590 IF LEN( A$)> 28 THEN 2520
2600 RETURN
2610 REM
2620 REM DATA FOR MALE NAMES
2630 DATA DR. WHO, FATHER CHRISTMAS
2640 DATA NEIL ARMSTRONG, A YETI
2650 DATA MICKEY MOUSE, SUPERMAN
2660 DATA SIR FRANCIS DRAKE. TARZAN
2670 DATA A HEADTEACHER, MR. SPOCK
2680 REM
2690 REM DATA FOR FEMALE NAMES
2700 DATA GOLDILOCKS, JOAN OF ARC
2710 DATA ENID BLYTON, MADAME CURIE
2720 DATA HELEN OF TROY, KATE BUSH
2730 DATA WONDER WOMAN, LOIS LANE
2740 DATA QUEEN ELIZABETH I, JANE
2750 REM
2760 REM DATA FOR PLACES
2770 DATA OUTSIDE THE ZOO
2780 DATA IN A PARALLEL UNIVERSE
2790 DATA AT HER GRANNY'S
2800 DATA ON A TELEVISION CHAT SHOW
2810 DATA IN A FLYING SAUCER
2820 DATA ON TOP OF MOUNT EVEREST
2830 DATA BEHIND A TAME GORILLA
2840 DATA ON THE BACK OF A MOTOR CYCLE
2850 DATA IN THE CREEPY FOREST
2860 DATA ON THE BEACH AT BOGNOR REGIS
2870 REM
2880 REM DATA FOR SPEECH
2890 DATA IS THIS RIGHT FOR THE BUS ?
2900 DATA YOUR HANDBAG IS ON FIRE
2910 DATA YOU REMIND ME OF GODZILLA
2920 DATA COULD YOU LEND ME FIVE QUID ?
2930 DATA THE END OF THE WORLD IS NIGH
2940 DATA I THINK IT MIGHT RAIN
2950 DATA I'VE LEFT MY TEDDY SOMEWHERE
2960 DATA YOU CLASH WITH MY TIE
2970 DATA YOUR RADIATOR IS LEAKING
2980 DATA ARE YOU FROM THE K.G.B. ?
```

```
2990 REM
3000 REM DATA FOR REPLY
3010 DATA DON'T BE SO IMPERTINENT
3020 DATA THAT IS NO CONCERN OF MINE
3030 DATA YOU'RE STANDING ON MY FOOT
3040 DATA HUH. I'VE HEARD THAT BEFORE
3050 DATA SIT DOWN. YOU LOOK ILL
3060 DATA PLEASE CARRY MY LUGGAGE
3070 DATA WOWIE! THAT'S FANTASTIC
3080 DATA PLEASE GO AWAY IMMEDIATELY
3090 DATA DANKE SEHR GUT. UND IHNEN ?
3100 DATA I KNEW YOU WOULD SAY THAT
3110 REM
3120 REM DATA FOR CONSEQUENCE
3130 DATA THE SUN WENT SUPERNOVA
3140 DATA A TIDAL WAVE ENGULFED THEM
3150 DATA IT BEGAN TO SNOW
3160 DATA THEY PLAYED HIDE AND SEEK
3170 DATA HE BLUSHED BRIGHT SCARLET
3180 DATA THEY BOTH SULKED
3190 DATA THE POLICE ASKED THEM TO GO
3200 DATA SHE PUSHED HIM OFF HIS BIKE
3210 DATA HE DROPPED HIS CRICKET BAT
3220 DATA HE JOINED THE FOREIGN LEGION

READY.
```

Commentary

The structure of Gosh is made fairly clear by the REM statements in the control module between lines 1020–1110.

Lines 1140–1220 form the initialisation routine. Six arrays are set up. M$ holds fifteen male names, F$ fifteen female names, and P$ fifteen places where the M$ and F$ array elements may happen to meet, etc. S$ is his speech, R$ her reply and C$ the consequence which happens after their exchange. The routine sets a white screen and selects blue text.

The array of male names is filled by the routine between lines 1240–1360. Ten of these names are taken from the lines of DATA, the other five being chosen by the child using the program. The child's choices are entered first into the array by the FOR...NEXT loop between lines 1280–1320. At line 1300, another subroutine, between lines 2510–2600, is called to check the length of the child's input. Gosh in operation does tend to encourage ima-

ginative reactions on the part of the user and it is important that an excessively long nickname for the younger brother is not allowed to spoil the final display. The routine also calls a subroutine between lines 2460–2490, the familiar PRINT TAB equivalent which allows the screen display to be neatly arranged.

After the child's names have been entered, the remaining ten places in the M$ array are filled by the loop between lines 1330–1350 from the names held as DATA. Obviously, here it is necessary for the person typing in the program to adjust the names used to those which will appeal most to the child. Probably regular updating of the DATA lines and reSAVEing on cassette will be quite a good idea, allowing the computer to interact more evidently with family life by making witty references to current domestic happenings. Check first what you can safely call Granny.

The remaining five arrays are filled in exactly the same way by the routines between lines 1380–2060.

Finally, the consequence is generated by the routine between lines 2080–2360. Each name, place, etc., is selected from the arrays by a random number given by the separate routine from lines 2380–2400. Again the PRINT TAB routine is used to arrange the screen neatly.

The DATA for Gosh is held between lines 2620–3220.

CHAPTER 6
Logically Speaking

Logical reasoning — problem solving — Treasure: finding pirates' gold — coordinates — using sound and sprites — Sonar: find the submarine.

Although the majority of the programs in the book so far have been concerned with teaching specific skills, many of them entail some form of logical reasoning before success can be achieved. This is the case with short programs such as Number, where, unless a logical strategy is worked out, correct answers can take a long time to be reached.

Piaget was one of the first psychologists to consider child development as proceeding through a series of stages. It is fair to say that his work was taken too literally by some educationists in the 1950s and 1960s, with the stages being seen as rigid plateaux that children reached. It was considered that no child of six or seven was capable of formal logical thought; partly because they were self-centred and therefore unable to see things from the point of view of others. More recent studies have shown that, if the questions are phrased in terms that children can understand, they are quite able on certain occasions to act logically.

Yet again, it is clear that many adults find it easier to understand some processes when offered models or pictures rather than the ambiguous symbolic jungle of words.

Until recently, logic has been a neglected area in schools, although it is implicit in most other subjects. Using coloured attribute blocks can demonstrate logic problems to young children, and many guessing games encourage logical thinking in the players.

In this chapter, we have included two programs that involve solving particular problems. They can be used individually, but we have found that a group of three or four children using the program at the same time will be more beneficial. The reason is that the programs stimulate discussion and a pooling of ideas.

As in many subjects, there is no one right answer to solving the problems, simply several strategies that may be tried in different situations.

Treasure

The first program, Treasure, displays an island covered by a grid of squares. Under one of them is buried some treasure. X and Y coordinates are explained in the instructions and using the program reinforces basic graphical ideas: 0,0 is the origin at the bottom lefthand corner, and the X coordinate is entered first. On entering the first pair, the location is shown on the screen, and the numbers chosen and their distance from the treasure is displayed on the right of the screen. Each attempt reduces the size of the treasure, and after 20 unsuccessful attempts the true position is shown.

The way to discover the distance from the guess to the treasure is to use Pythagoras' theorem. It is easy to calculate the horizontal and vertical distances from the treasure, so these form two sides of a right-angled triangle. The distance in a straight line forms the long side or hypotenuse, and this distance is the square root of the sum of the squares of the other two sides.

distance = SQR(X^2 + Y^2)

Apart from the normal arithmetic operators, the 64 contains a selection of more advanced functions, including ^ to raise to a power (ie 3^4 is 3*3*3*3, or 81), SQR to produce square roots, ABS to produce the absolute value of numbers, and SIN, COS, and TAN for trigonometric use. In this program, it is necessary to use the absolute value for the distances to avoid negative numbers.

For older children, the program could be adapted to demonstrate three-dimensional coordinates. The third input would be the depth of the treasure — this has proved rather difficult to solve quickly, as there are more than 3000 possible locations!

```
1000 REM ***  TREASURE ISLAND  ***
1010 REM
1020 REM ***  A.J.S.  13.4.83  ***
1030 REM
1040 GOSUB3000:REM ***  INSTRUCTIONS     ***
1050 GOSUB1500:REM ***  DRAW ISLAND      ***
1060 GOSUB2000:REM ***  INITIALISATION   ***
1070 GOSUB2500:REM ***  GAME             ***
1080 GOSUB3500:REM ***  RESULTS         ***
1090 GOTO1050
1500 REM
1510 REM ***  DRAW ISLAND  ***
1520 REM
1530 POKE53281,6
1540 PRINTCHR$(147):POKE646,1
```

```
1550 PRINTTAB(6)"TREASURE  ISLAND":POKE646,7
1560 FORN=0 TO 15
1570 Y=18-N:X=3:GOSUB4000:POKE646,1:PRINTN
1580 FORP=0 TO 15
1590 X=P+6:Y=N+3:GOSUB4000
1600 POKE646,7:PRINTCHR$(166)
1610 NEXTP
1620 NEXTN
1630 X=6:Y=20:GOSUB4000
1640 POKE646,1:PRINT"0123456789012345"
1650 X=16:Y=19:GOSUB4000
1660 PRINT"111111"
1670 X=2:Y=10:GOSUB4000:PRINT"^":PRINT"  Y"
1680 X=12:Y=22:GOSUB4000:PRINT"X->"
1700 RETURN
1990 STOP
2000 REM
2010 REM ***  INITIALISATION  ***
2020 REM
2030 A = INT(RND(0)*16)
2040 B = INT(RND(0)*16)
2050 TRIES = 1
2060 WIN = 0
2070 RETURN
2500 REM
2510 REM ***  GAME              ***
2520 REM
2530 X=24:Y=6:GOSUB4000:PRINT"TRY NO. "TRIES
2540 X=24:Y=8:GOSUB4000:PRINT"ENTER THE"
2550 X=24:Y=9:GOSUB4000:PRINT"COORDINATES"
2555 X=27:Y=11:GOSUB4000:PRINT"    "
2560 X=24:Y=11:GOSUB4000:INPUT"X= ";X$
2570 IF ASC(X$)<48 OR ASC(X$)>57 THEN 2555
2580 IF VAL(X$)<0  OR VAL(X$)>15 THEN 2555
2585 X=27:Y=13:GOSUB4000:PRINT"    "
2590 X=24:Y=13:GOSUB4000:INPUT"Y= ":Y$
2600 IF ASC(Y$)<48 OR ASC(Y$)>57 THEN 2585
2610 IF VAL(Y$)<0  OR VAL(Y$)>15 THEN 2585
2620 X1 = VAL(X$)
2630 Y1 = VAL(Y$)
2640 DIST=INT(.5+SQR((ABS(A-X1)^2)+(ABS(B-Y1)^2)))
2650 X=29:Y=16:GOSUB4000:PRINT"    "
2660 X=22:Y=16:GOSUB4000:PRINT"YOU ARE"DIST
2670 X=22:Y=17:GOSUB4000:PRINT"SQUARES AWAY"
2680 X=X1+6:Y=18-Y1:GOSUB4000:POKE646,5
2690 PRINTCHR$(113):POKE646,1
2700 IF X1=A AND Y1=B THEN WIN=1
2710 IF TRIES<20 AND WIN=0 THEN TRIES = TRIES+1:
     GOTO2530
```

```
2720 RETURN
3000 REM
3010 REM ***  INSTRUCTIONS  ***
3020 REM
3030 PRINTCHR$(147)
3040 PRINT" YOU WILL SOON SEE A PLAN OF AN ISLAND"
3050 PRINT
3060 PRINT"WHERE A PIRATE HAS BURIED HIS TREASURE."
3070 PRINT
3080 PRINT"TO FIND HIS TREASURE YOU CAN ENTER THE"
3090 PRINT
3100 PRINT"COORDINATES OF WHERE YOU THINK IT IS."
3110 PRINT
3120 PRINT"YOU WILL BE TOLD HOW FAR AWAY YOU ARE"
3130 PRINT
3140 PRINT"BUT NOT THE DIRECTION. UNFORTUNATELY."
3150 PRINT
3160 PRINT"EACH GUESS COSTS YOU SOME TREASURE."
3170 PRINT
3180 PRINT"REMEMBER THAT THE FIRST COORDINATE (X)"
3190 PRINT
3200 PRINT"COUNTS ACROSS, THE SECOND (Y), COUNTS"
3210 PRINT
3220 PRINT"UPWARDS. GOOD LUCK!"
3230 PRINT:PRINT
3240 PRINT"PRESS THE SPACE BAR TO BEGIN"
3250 GET Z$:IF Z$="" THEN 3250
3260 RETURN
3500 REM
3510 REM ***     RESULTS      ***
3520 REM
3530 X=1:Y=21:GOSUB4000
3540 IF WIN=0 THEN PRINT"THE PIRATE COULDN'T WAIT"
3560 X=A+6:Y=18-B:GOSUB4000
3570 PRINT"X"
3580 X=1:Y=22:GOSUB4000
3590 IF WIN=1 THEN PRINT"WELL DONE, YOU WIN"INT
     (1000/TRIES)"DIAMONDS"
3595 PRINT" THE TREASURE WAS AT"A;B
3600 PRINT" PRESS SPACE BAR TO GO AGAIN";
3610 GETZ$:IF Z$=""THEN3610
4000 REM
4010 REM ***  'PRINT AT' ROUTINE  ***
4020 REM
4030 POKE782,X:POKE781,Y:SYS65520:RETURN

READY.
```

Commentary

Lines 1040–1090 contain the control module.

Lines 1510–1700 draw the island. Line 1530 sets the background colour to blue, and line 1540 clears the screen and sets the printing colour to yellow.

The island is drawn by two nested loops, N and P. The position of where CHR$(166) is plotted is determined by the 'print at' routine held at line 4030.

Line 1630 draws the X coordinates, while the Y coordinates are printed in line 1570 by the plotting loop.

The initialisation routine in lines 2000–2070 chooses the values of A and B, where the treasure is buried. It also sets the number of attempts, TRIES, and the win flag, WIN, to zero.

Lines 2500–2720 contain the main game loop. This makes extensive use of the 'print at' routine to ensure that the plotting of points and input does not interfere with the display. The more traditional method would have been to POKE the screen memory directly, but this does not show so clearly what is happening.

Lines 2570, 2580, 2600 and 2610 check the ASCII values of the guesses, and reject those inputs that are not numbers, and those values that would not plot on the island (less than zero, and more than 15).

Lines 2620 and 2630 assign the guesses to variables X1 and Y1.

Line 2640 calculates the distance the guesses are from the treasure. Note that the sign after the figure two is an up arrow, signifying that the value before it should be squared, eg 4 to the power of 3 could be entered as 4^3.

Lines 2680 and 2690 determine where the small green circle should be plotted on the screen to show where a hole, CHR$(113), has been dug. Line 2700 checks if the treasure has been found. If it has, the flag, WIN, is set to 1.

Line 2710 tests the number of attempts, held in variable TRIES, and if it has less than 20, and if WIN is still at zero, loops back to allow another guess. The only conditions when the program will pass this line are either when TRIES is more than 20, or if WIN has been set to 1.

Lines 3000–3620 contain the instructions.

Lines 3500–3610 contain the results subroutine. Lines 3560 and 3570 print an X where the treasure was buried. Line 3590 calculates the number of diamonds left, by dividing it between the number of attempts.

Line 4030 sets the X and Y points of the print position, by initiating a system call to 65520.

Sonar

The last program showed how an elementary idea can be developed into a game that teaches strategy. Watching children of different ages play it is an interesting experience, as different methods of solving the problem are tried. This is the sort of game where several children using the program at the same time can stimulate each other into useful discussion.

The next program is similar in concept, in that there is a grid of squares, with something hidden in it. The last program offered only distance as a clue to the position, but the next gives a compass clue as well. The program could be modified to give different clues, such as warm/cold, or the direction could be shown as an angle.

Programming points to note are the use of sprites and sound. There are notes after the program to explain these further.

Sound and graphics are two particular features of the 64, but neither of them is particularly easy to use. To do justice to either topic would take up a whole separate book. If you want to use sprites in your own programs, then try adapting ours, and seeing how you can move them round the screen. To generate them, we cannot do better than recommend a program in David Lawrence's book, also from Sunshine Books, *The Working Commodore 64*.

Now, close the hatches and prepare to dive!

```
1000 REM SONAR - PAT HALL, JAN '84
1010 REM
1020 GOSUB 1110 : REM DEFINE SPRITE
1030 GOSUB 1290 : REM SET UP DISPLAY
1040 GOSUB 1570 : REM CONTROLS
1050 GOSUB 1960 : REM SEARCH
1060 GOSUB 2540 : REM REWARD
1070 END
1080 REM
1090 REM DEFINE SPRITE
1100 REM BEGINNING OF 6567 VIDEO CHIP
1110 VC = 53248
1120 REM SONAR SIGHTS, SPRITE 0
1130 REM SET POINTER FOR SPRITE 0
1140 POKE 2040, 13
1150 REM READ DATA FOR SPRITE 0
1160 FOR I = 0 TO 62
1170 READ N : POKE 832 + I, N
1180 NEXT I
1190 REM ENLARGE SPRITE IN X DIRECTION
```

```
1200 POKE VC + 29, 1
1210 REM ENLARGE SPRITE IN Y DIRECTION
1220 POKE VC + 23, 1
1230 REM SELECT COLOUR
1240 POKE VC + 39, 0
1250 RETURN
1260 REM
1270 REM SET UP DISPLAY
1280 REM WHITE SCREEN
1290 POKE 53280, 1 : POKE 53281, 1
1300 PRINT "[SHIFT/CLR/HOME]"
1310 REM DRAW COORDINATE GRID
1320 FOR I = 0 TO 24
1330 FOR J = 0 TO 960 STEP 960
1340 POKE 55296 + I + J, 0
1350 POKE 1024 + I + J, 160
1360 NEXT J
1370 NEXT I
1380 FOR I = 0 TO 880 STEP 40
1390 FOR J = 0 TO 24 STEP 24
1400 POKE 55336 + I + J, 0
1410 POKE 1064 + I + J, 160
1420 NEXT J
1430 NEXT I
1440 FOR I = 0 TO 22
1450 FOR J = 0 TO 880 STEP 40
1460 POKE 55337 + I + J, 7
1470 POKE 1065 + I + J, 76
1480 NEXT J
1490 NEXT I
1500 REM CENTRE SIGHTS ON GRID
1510 XS = 101 : YS = 129
1520 POKE VC, XS : POKE VC + 1, YS
1530 POKE VC + 21, 1
1540 RETURN
1550 REM
1560 REM CONTROLS
1570 PRINT CHR$( 144 )
1580 X = 27 : Y = 0 : GOSUB 2450
1590 PRINT "<< SONAR >>"
1600 X = 29 : Y = 2 : GOSUB 2450
1610 PRINT"MISSION:"
1620 PRINT CHR$( 30 )
1630 X = 27 : Y = 4 : GOSUB 2450
```

```
1640 PRINT "LOCATE A LOST"
1650 Y = 5 : GOSUB 2450
1660 PRINT "RESEARCH SUB"
1670 PRINT CHR$( 144 )
1680 X = 28 : Y = 7 : GOSUB 2450
1690 PRINT "CONTROLS:"
1700 PRINT CHR$( 30 )
1710 Y = 9 : GOSUB 2450
1720 PRINT "NORTH - J"
1730 Y = 10 : GOSUB 2450
1740 PRINT "SOUTH - L"
1750 Y = 11 : GOSUB 2450
1760 PRINT "EAST  - D"
1770 Y = 12 : GOSUB 2450
1780 PRINT "WEST  - A"
1790 PRINT CHR$( 144 )
1800 Y = 14 : GOSUB 2450
1810 PRINT "DIRECTION"
1820 Y = 15 : GOSUB 2450
1830 PRINT "OF SUB:"
1840 Y = 18 : GOSUB 2450
1850 PRINT "DISTANCE:"
1860 Y = 20 : GOSUB 2450
1870 PRINT "TIME TAKEN"
1880 Y = 21 : GOSUB 2450
1890 PRINT "FOR SEARCH:"
1900 PRINT CHR$( 30 )
1910 TM = TI
1920 RETURN
1930 REM
1940 REM STEER 'SHIP'
1950 REM SET AUTO REPEAT
1960 POKE 650, 128
1970 REM LOSE SUB !
1980 XT = INT( RND( 1 ) * 120 ) + 40
1990 YT = INT( RND( 1 ) * 120 ) + 70
2000 REM PREVENT PREMATURE SUCCESS !
2010 GOSUB 2500
2020 IF S < 10 THEN 1980
2030 REM READ KEYBOARD
2040 GET K$
2050 IF K$ = "A" THEN XS = XS-1
2060 IF K$ = "D" THEN XS = XS+1
2070 IF XS < 30 THEN XS = 30
```

```
2080 IF XS > 170 THEN XS = 170
2090 IF K$ = "J" THEN YS = YS - 1
2100 IF K$ = "L" THEN YS = YS + 1
2110 IF YS < 59 THEN YS = 59
2120 IF YS > 200 THEN YS = 200
2130 REM MOVE SIGHTS
2140 POKE VC, XS
2150 POKE VC + 1, YS
2160 REM CALCULATE DIRECTION
2170 X = 28 : Y = 16 : GOSUB 2450
2180 PRINT "           "
2190 GOSUB 2450
2200 IF YS > YT+5 THEN PRINT "NORTH ";
2210 IF YS < YT-5 THEN PRINT "SOUTH ";
2220 IF XS > XT+5 THEN PRINT "WEST"
2230 IF XS < XT-5 THEN PRINT "EAST"
2240 REM CALCULATE TARGET DISTANCE
2250 GOSUB 2500
2260 REM FOUND SUB
2270 IF S < 6 THEN 2420
2280 X = 37 : Y = 18 : GOSUB 2450
2290 PRINT "   "
2300 GOSUB 2450
2310 PRINT S
2320 X = 31 : Y = 22 : GOSUB 2450
2330 PRINT INT( ( TI - TM ) / 100 )
2340 REM SOUND EFFECT
2350 POKE 54296, 15 : POKE 54277, 160
2360 POKE 54278, 160 : POKE 54276, 17
2370 POKE 54273, 200-S : POKE 54272, 9
2380 FOR T = 1 TO 50 : NEXT T
2390 POKE 54277,0 : POKE 54278, 0
2400 POKE 54276, 0
2410 GOTO 2040
2420 RETURN
2430 REM
2440 REM PRINT TAB ROUTINE
2450 POKE 782, X : POKE 781, Y
2460 SYS 65520
2470 RETURN
2480 REM
2490 REM CALCULATE TARGET DISTANCE
2500 S = INT(((XS-XT)^2+(YS-YT)^2)^.5)
2510 RETURN
```

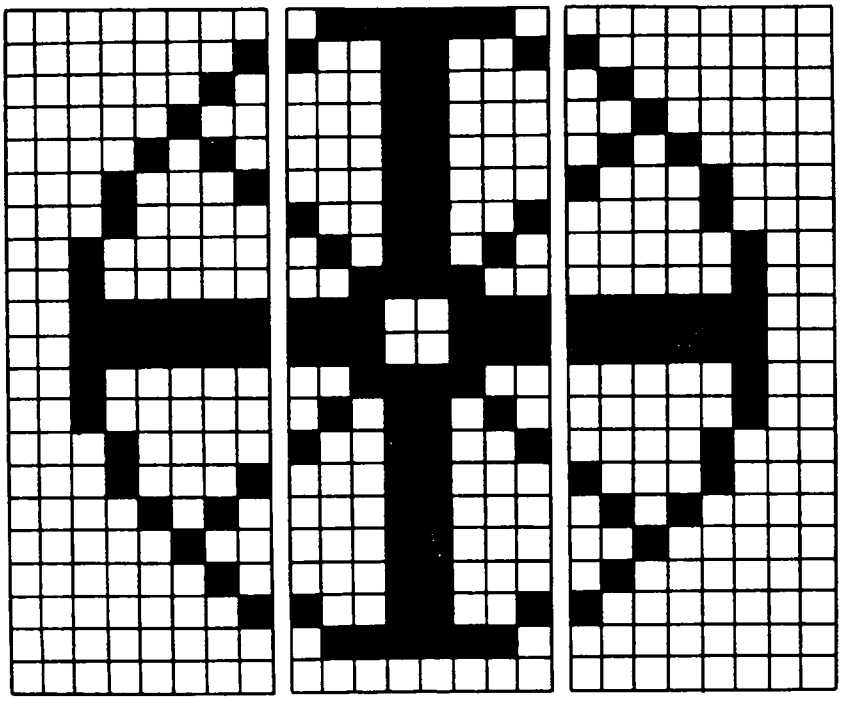
```
2520 REM
2530 REM REWARD ROUTINE
2540 POKE 53280, 3
2550 POKE 53281, 3
2560 PRINT "[SHIFT/CLR/HOME]"
2570 REM SWITCH OFF SPRITE
2580 POKE VC + 21, 0
2590 REM DRAW SEA
2600 C = 6
2610 FOR I = 0 TO 160 STEP 40
2620 FOR J = 0 TO 39
2630 L = 1824 + I + J
2640 GOSUB 2810
2650 NEXT J
2660 NEXT I
2670 REM DRAW SUBMARINE
2680 C = 7
2690 FOR L = 1794 TO 1814
2700 GOSUB 2810
2710 NEXT L
2720 FOR I = 0 TO 80 STEP 40
2730 FOR J = 0 TO 4
2740 L = 1683 + I + J
2750 GOSUB 2810
2760 NEXT J
2770 NEXT I
2780 RETURN
2790 REM
2800 REM PLOT CHARACTER SPACE
2810 POKE L + 54272, C : POKE L, 160
2820 RETURN
2830 REM
2840 REM DATA FOR SONAR SIGHTS
2850 REM
2860 DATA 0, 126, 0, 1, 153, 128, 2, 24
2870 DATA 64, 4, 24, 32, 10, 24, 80, 17
2880 DATA 24, 136, 16, 153, 8, 32, 90
2890 DATA 4, 32, 60, 4, 63, 231, 252
2900 DATA 63, 231, 252, 32, 60, 4, 32
2910 DATA 90, 4, 16, 153, 8, 17, 24
2920 DATA 136, 10, 24, 80, 4, 24, 32
2930 DATA 2, 24, 64, 1, 153, 128, 0
2940 DATA 126, 0, 0, 0, 0
READY.
```

Commentary

Sonar is controlled by lines 1020–1060. First the sprite representing the sights on the sonar screen is defined. Then line 1030 calls the routine which places the sonar screen on the computer's monitor. This is followed by a routine which fills the righthand side of the screen with a set of control instructions and prompts, including the direction to steer in, the distance of a lost submarine and the time taken so far in the search. Line 1050 calls the routine which actually allows the sonar scan to take place and finally line 1060 summons the reward routine, the submarine resurfacing after its discovery.

Lines 1090–1250 define sprite 0 as the sonar sight. Line 1110 initialises VC as the starting location of the 6567 video chip, and all subsequent sprite control is done with reference to this location. Line 1140 sets the sprite pointer for sprite 0 with POKE 2040, 13 and then the FOR...NEXT loop from 1160 to 1180 reads all the necessary DATA to define the sprite.

Diagram E



The diagram shows how the sonar sight has been constructed on an array of 21 by 24 squares. Remember that the sprite is coded as a series of horizontal blocks of eight squares. Each is regarded as a binary number. A filled square gives a 1 and an empty square a 0. For example, the fifth block of eight squares starting from the top lefthand corner, which is therefore the block of eight in the middle of the second row, gives the binary number 10011001. This creates the base 10 equivalent of (128 + 16 + 8 + 1) or 153. This number, 153, is therefore the fifth number in the DATA line 2860. Checking the values for other blocks of eight squares in the diagram will help to give confidence in developing Commodore sprite graphics, which are one of the most useful features of the machine.

One advantage of sprite graphics is the easy way in which they can be moved over a background display without ruining it. This is much harder on other machines. The feature is used in Sonar in the way the sonar sight travels happily around the screen until the submarine is located. To give added effect, the sprite is enlarged in both horizontal and vertical directions at lines 1200, 1220 by POKEing the locations VC + 29 and VC + 23 with the sprite number, which in this case, for sprite 0, is 1. Line 1240 colours the sprite black by POKEing 0 to location VC + 39, the colour location for sprite 0.

Lines 1270–1540 generate the sonar screen, over which the child has to steer the sight to locate the submarine. The two pairs of nested FOR... NEXT loops between lines 1320–1370 and 1380–1430 draw the perimeter of the screen by POKEing character 160 to locations determined by the range of the I and J variables and by the way in which they are STEPped. Colour 0 is POKEd to the corresponding locations of the colour map to give a black outline to the coordinate grid. The squares themselves are produced by POKEing character 76 to all the screen locations in the area enclosed. These values are again derived from the two loop variables, I and J. This time, colour 7 is chosen, at line 1460. Line 1520 locates the sonar sight on the screen by POKEing the values XS and YS to the locations VC and VC + 1. The initial values of XS and YS are given at line 1510 but are subsequently updated as the child types in appropriate directions. Finally, line 1530 switches on sprite 0 by POKEing VC + 21.

The instructions given to the child are produced by the routine between lines 1560–1920. All the text is printed on the right of the sonar screen. Extensive use is made of GOSUB 2450 to arrange the text neatly in the confined areas involved. Clarity is aided by switching frequently from black to green text using PRINT CHR$(144) and PRINT CHR$(30). (This has to be done before GOSUB 2450 or else the new screen location is lost.) The information on the screen is also carefully arranged to match the messages printed on the screen and continuously updated by the following

routine. Finally TM, which times the search, is initialised at line 1910 by equating it with TI, the elapsed time since the machine was switched on.

Lines 1940–2420 allow the sonar sight to be moved around the screen. Line 1960 sets the auto-repeat with POKE 650, 128. This is necessary to prevent the child constantly having to depress the control keys, A, D, J and L. The coordinates of the lost submarine are decided at lines 1980 and 1990, the following line preventing the submarine being found immediately. Here a further subroutine, lines 2490–2510, is used to calculate the submarine's distance and to jump back to line 1980 should it be too close. Lines 2040–2120 read the keyboard and adjust the sprite coordinates, XS and YS. The sprite is moved by lines 2140 and 2150.

Two pieces of information are PRINTed on the screen to aid location of the submarine. Lines 2200–2230 use a simple arithmetic test to PRINT 'North', 'East', etc., on the screen. By placing the test for North and South before the test for East and West, and by putting a semicolon after the two strings, the further combination 'North East' etc., is automatically created when necessary. Line 2250 uses GOSUB 2500 to determine the submarine's distance and line 2270 jumps out of the GOTO loop created by line 2410 if the submarine has been located. Meanwhile, a sound effect is being generated by lines 2340–2400. The pitch of the note is related to the distance of the submarine by adjusting the high note POKEd at line 2370 by subtracting S.

A simple reward routine is included at lines 2530–2780. It uses techniques already described in earlier programs to build up a picture of a submarine surfacing.

CHAPTER 7
The Real Thing

Unique role of computer in simulation — the appeal of computer graphics — simulation versus immediate experience — the realm of simulations — Lock: an excursion on the canal — Power: electricity generation — interactive simulations — elementary statistics — Coin: probability of throwing a head — Cascade: Pascal's triangle.

Finally, it is time to consider one application of computers which allows them to play a unique role, one which could not be achieved in any other way. A parent or teacher with a piece of paper, blackboard or overhead projector can, of course, quite adequately explain the addition of fractions or show how creative writing can be made more interesting by carefully chosen adjectives.

The same person would, however, find it more difficult to illustrate the operation of the human heart or the disintegration of a meteorite entering the earth's atmosphere. Yet it is precisely these sorts of demonstrations that the computer can create effortlessly on its monitor. Perhaps here it is being used to its greatest effect.

Children are already used to computer-generated images. Most nights on the television, computer graphics are seen showing the flight of a space probe past Jupiter, advertising razor blades or giving the percentage swing towards the Opposition. The undoubted appeal of such techniques, when introduced into the area of educational computing, will add extra incentive to many subject areas. Of course, there is no real substitute for firsthand experience, but, when that experience is expensive, dangerous, illegal or otherwise impossible to recreate in the comfort of your own home or classroom without eyebrows being raised, then perhaps you should consider writing a suitable program.

Four computer simulations are offered here for educational consumption. The first of these, Lock, attempts to explain something that young children usually find baffling: the problem of how canal boats go uphill. Now the enterprising teacher or parent will either organise a visit to the Grand Union or else set to work with plastic garden troughs, pieces of Meccano and a hosepipe. Probably from an educationalist's viewpoint, this remains the only real approach. However the sheer magnitude of such

an operation should bring home very clearly one point: computer simulations let you have nearly anything sitting in a cassette next to the computer.

Lock

Lock allows the child to steer a boat up and down a canal. The gates and sluices work in the correct sequence and the water level changes, too. This program uses many of the techniques used in earlier chapters, especially those involved in defining and moving sprites across the screen.

```
1000 REM LOCK - PAT HALL, JAN '84
1010 REM
1020 GOSUB 1120 : REM DEFINE SPRITES
1030 GOSUB 1360 : REM SET UP DISPLAY
1040 GOSUB 2570 : REM CHOOSE DIRECTION
1050 IF CH = 1 THEN GOSUB 2800
1060 IF CH = 2 THEN GOSUB 3420
1070 GOTO 1040
1080 END
1090 REM
1100 REM DEFINE SPRITES
1110 REM BEGINNING OF 6567 VIDEO CHIP
1120 VC = 53248
1130 REM LOWBOAT, SPRITE 0
1140 REM SET POINTER FOR SPRITE 0
1150 POKE 2040, 13
1160 REM READ DATA FOR SPRITE 0
1170 FOR I = 0 TO 62
1180 READ N : POKE 832 + I, N
1190 NEXT I
1200 REM HIGHBOAT, SPRITE 1
1210 REM SET POINTER FOR SPRITE 1
1220 POKE 2041, 14
1230 REM READ DATA FOR SPRITE 1
1240 FOR I = 0 TO 62
1250 READ N : POKE 896 + I, N
1260 NEXT I
1270 REM ENLARGE SPRITES IN X DIRECTION
1280 POKE VC + 29, 3
1290 REM SELECT COLOUR
1300 POKE VC + 39, 1
```

```
1310 POKE VC + 40, 1
1320 RETURN
1330 REM
1340 REM SET UP DISPLAY
1350 REM WHITE BORDER
1360 PRINT "[SHIFT/CLR/HOME]"
1370 POKE 53280, 1
1380 REM HILLSIDE
1390 POKE 53281, 5
1400 REM SKY
1410 C = 3 : REM SELECT CYAN
1420 FOR I = 13 TO 1 STEP -1
1430 REM LV = VERTICAL LOCATION OF BAR
1440 LV = 1549 - I * 40
1450 REM J DETERMINES LENGTH OF BAR
1460 FOR J = -5 TO I - 1
1470 L = LV + J
1480 GOSUB 2490
1490 NEXT J
1500 REM FILL IN END OF BAR
1510 POKE L + 54273, 3
1520 POKE L + 1, 105
1530 NEXT I
1540 REM LOWER RIVER
1550 C = 6 : REM SELECT BLUE
1560 FOR I = 0 TO 13
1570 FOR J = 0 TO 280 STEP 40
1580 L = 1704 + I + J
1590 GOSUB 2490
1600 NEXT J
1610 NEXT I
1620 REM UPPER RIVER
1630 FOR I = 0 TO 24
1640 FOR J = 0 TO 520 STEP 40
1650 L = 1479 + I + J
1660 GOSUB 2490
1670 NEXT J
1680 NEXT I
1690 REM SET FLAG FOR WATER LEVEL
1700 F = 1
1710 GOSUB 1850 : REM LOWER GATE
1720 GOSUB 1920 : REM UPPER GATE
1730 REM TITLE
1740 PRINT CHR$( 144 )
```

```
1750 X = 24 : Y = 1 : GOSUB 1800
1760 PRINT "<< LOCK >>"
1770 RETURN
1780 REM
1790 REM PRINT TAB ROUTINE
1800 POKE 782, X : POKE 781, Y
1810 SYS 65520
1820 RETURN
1830 REM
1840 REM CLOSE LOWER GATE
1850 C = 0 : REM SELECT BLACK
1860 FOR L = 1398 TO 1998 STEP 40
1870 GOSUB 2490
1880 NEXT L
1890 RETURN
1900 REM
1910 REM CLOSE UPPER GATE
1920 C = 0 : REM SELECT BLACK
1930 FOR L = 1409 TO 2009 STEP 40
1940 GOSUB 2490
1950 NEXT L
1960 RETURN
1970 REM
1980 REM OPEN LOWER GATE
1990 FOR L = 1398 TO 1998 STEP 40
2000 C = 5 : IF L > 1678 THEN C = 6
2010 GOSUB 2490
2020 NEXT L
2030 RETURN
2040 REM
2050 REM OPEN UPPER GATE
2060 FOR L = 1409 TO 2009 STEP 40
2070 C = 5 : IF L > 1449 THEN C = 6
2080 GOSUB 2490
2090 NEXT L
2100 RETURN
2110 REM
2120 REM DRAIN LOCK
2130 REM OPEN SLUICE IN LOWER GATE
2140 POKE 56190, 6
2150 REM LOWER WATER IN LOCK
2160 P = 10
2170 FOR I = 0 TO 200 STEP 40
2180 FOR J = 0 TO 9
```

```
2190 POKE 55751 + I + J, 5
2200 NEXT J
2210 GOSUB 2530 : REM PAUSE
2220 NEXT I
2230 P = 50 : GOSUB 2530 : REM PAUSE
2240 REM CLOSE SLUICE IN LOWER GATE
2250 POKE 56190, 0
2260 REM SET FLAG FOR EMPTY LOCK
2270 F = 0
2280 RETURN
2290 REM
2300 REM FILL LOCK
2310 REM OPEN SLUICE IN UPPER GATE
2320 POKE 56201, 6
2330 REM RAISE WATER IN LOCK
2340 P = 10
2350 FOR I = 200 TO 0 STEP -40
2360 FOR J = 0 TO 9
2370 POKE 55751 + I + J, 6
2380 NEXT J
2390 GOSUB 2530 : REM PAUSE
2400 NEXT I
2410 P = 50 : GOSUB 2530 : REM PAUSE
2420 REM CLOSE SLUICE IN UPPER GATE
2430 POKE 56201, 0
2440 REM SET FLAG FOR FULL LOCK
2450 F = 1
2460 RETURN
2470 REM
2480 REM PLOT CHARACTER SPACE
2490 POKE L + 54272, C : POKE L, 160
2500 RETURN
2510 REM
2520 REM PAUSE
2530 FOR T = 1 TO P * 10 : NEXT T
2540 RETURN
2550 REM
2560 REM CHOOSE DIRECTION
2570 PRINT CHR$( 5 )
2580 X = 23 : Y = 3 : GOSUB 1800
2590 PRINT "TYPE 1 OR 2"
2600 Y = 4 : GOSUB 1800
2610 PRINT "1 - BOAT UP"
2620 Y = 5 : GOSUB 1800
```

```
2630 PRINT "2 - BOAT DOWN"
2640 Y = 7 : GOSUB 1800
2650 PRINT "                "
2660 Y = 7 : GOSUB 1800
2670 INPUT CH$
2680 IF ASC( CH$) < 48 THEN 2640
2690 IF ASC( CH$)> 57 THEN 2640
2700 CH = VAL( CH$)
2710 IF CH <> 1 AND CH <> 2 THEN 2640
2720 FOR I = 3 TO 7
2730 Y = I : GOSUB 1800
2740 PRINT"                    "
2750 NEXT I
2760 RETURN
2770 REM
2780 REM GO UPRIVER
2790 REM SWITCH ON SPRITE 0
2800 POKE VC + 21, 1
2810 REM MOVE BOAT TO LOCK
2820 P = 1 : REM SET PAUSE DELAY
2830 FOR I = 0 TO 86
2840 XB = I : YB = 165 : GOSUB 4030
2850 NEXT I
2860 P = 50 : GOSUB 2530 : REM PAUSE
2870 REM CHECK WATER LEVEL
2880 IF F = 1 THEN GOSUB 2140 : REM DRN
2890 GOSUB 2530 : REM PAUSE
2900 REM OPEN LOWER GATE
2910 GOSUB 1990
2920 REM MOVE BOAT INTO LOCK
2930 P = 1 : REM SET PAUSE DELAY
2940 FOR I = 87 TO 162
2950 XB = I : YB = 165 : GOSUB 4030
2960 NEXT I
2970 REM CLOSE LOWER GATE
2980 GOSUB 1850
2990 P = 50 : GOSUB 2530 : REM PAUSE
3000 REM FILL LOCK
3010 REM OPEN SLUICE IN UPPER GATE
3020 POKE 56201, 6
3030 REM RAISE WATER IN LOCK
3040 P = 10
3050 FOR I = 200 TO 0 STEP -40
3060 REM RAISE BOAT
```

```
3070 POKE VC + 1, 117 + I / 5
3080 FOR J = 0 TO 9
3090 POKE 55751 + I + J, 6
3100 NEXT J
3110 GOSUB 2530 : REM PAUSE
3120 NEXT I
3130 REM CLOSE SLUICE IN UPPER GATE
3140 POKE 56201, 0
3150 REM SET FLAG FOR FULL LOCK
3160 F = 1
3170 P = 70 : GOSUB 2530 : REM PAUSE
3180 REM OPEN UPPER GATE
3190 GOSUB 2060
3200 REM MOVE BOAT OUT OF LOCK
3210 P = 1 : REM SET PAUSE DELAY
3220 FOR I = 163 TO 235
3230 XB = I : YB = 117 : GOSUB 4030
3240 NEXT I
3250 REM CLOSE UPPER GATE
3260 GOSUB 1920
3270 REM MOVE BOAT AWAY FROM LOCK
3280 FOR I = 236 TO 255
3290 XB = I : YB = 117 : GOSUB 4030
3300 NEXT I
3310 REM RHS OF SCREEN, SET MSB
3320 POKE VC + 16, 1
3330 FOR I = 0 TO 88
3340 XB = I : YB = 117 : GOSUB 4030
3350 NEXT I
3360 POKE VC + 16, 0
3370 POKE VC + 21, 0
3380 RETURN
3390 REM
3400 REM GO DOWNRIVER
3410 REM SWITCH ON SPRITE 1
3420 POKE VC + 21, 2
3430 REM MOVE BOAT TO LOCK
3440 REM RHS OF SCREEN, SET MSB
3450 POKE VC + 16, 2
3460 P = 1 : REM SET PAUSE DELAY
3470 FOR I = 88 TO 0 STEP -1
3480 XB = I : YB = 117 : GOSUB 4080
3490 NEXT I
3500 POKE VC + 16, 0
```

```
3510 FOR I = 255 TO 236 STEP -1
3520 XB = I : YB = 117 : GOSUB 4080
3530 NEXT I
3540 P = 50 : GOSUB 2530 : REM PAUSE
3550 REM CHECK WATER LEVEL
3560 IF F = 0 THEN GOSUB 2320 :REM FILL
3570 GOSUB 2530 : REM PAUSE
3580 REM OPEN UPPER GATE
3590 GOSUB 2060
3600 REM MOVE BOAT INTO LOCK
3610 P = 1 : REM SET PAUSE DELAY
3620 FOR I = 235 TO 163 STEP -1
3630 XB = I : YB = 117 : GOSUB 4080
3640 NEXT I
3650 REM CLOSE UPPER GATE
3660 GOSUB 1920
3670 P = 50 : GOSUB 2530 : REM PAUSE
3680 REM DRAIN LOCK
3690 REM OPEN SLUICE IN LOWER GATE
3700 POKE 56190, 6
3710 REM LOWER WATER IN LOCK
3720 P = 10
3730 FOR I = 0 TO 200 STEP 40
3740 REM LOWER BOAT
3750 POKE VC + 3, 117 + I / 5 + 8
3760 FOR J = 0 TO 9
3770 POKE 55751 + I + J, 5
3780 NEXT J
3790 GOSUB 2530 : REM PAUSE
3800 NEXT I
3810 REM CLOSE SLUICE IN LOWER GATE
3820 POKE 56190, 0
3830 REM SET FLAG FOR EMPTY LOCK
3840 F = 0
3850 P = 70 : GOSUB 2530 : REM PAUSE
3860 REM OPEN LOWER GATE
3870 GOSUB 1990
3880 REM MOVE BOAT OUT OF LOCK
3890 P = 1 : REM SET PAUSE DELAY
3900 FOR I = 162 TO 86 STEP -1
3910 XB = I : YB = 165 : GOSUB 4080
3920 NEXT I
3930 REM CLOSE LOWER GATE
3940 GOSUB 1850
```

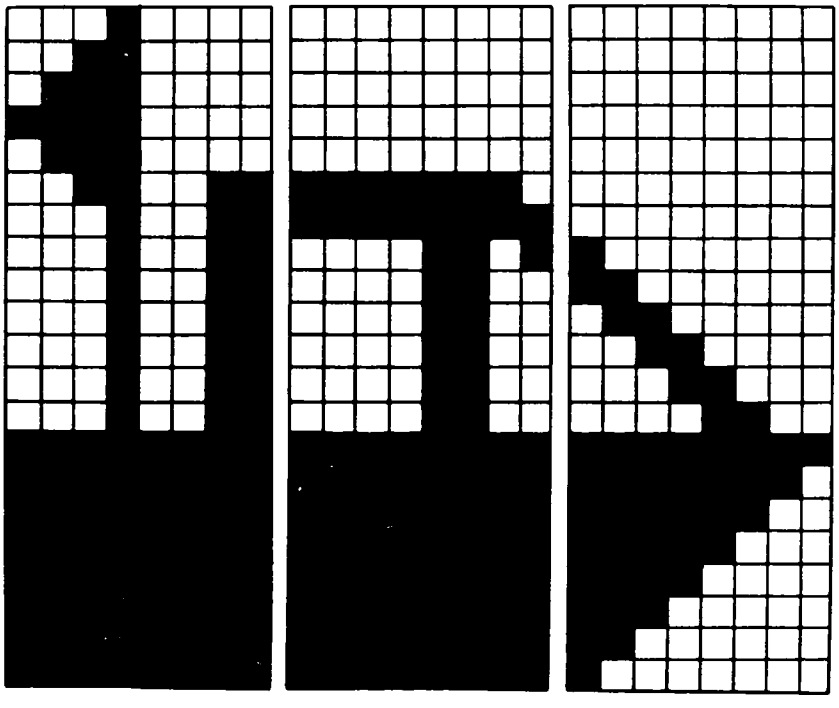
```
3950 REM MOVE BOAT AWAY FROM LOCK
3960 FOR I = 85 TO 0 STEP -1
3970 XB = I : YB = 165 : GOSUB 4080
3980 NEXT I
3990 POKE VC + 21, 0
4000 RETURN
4010 REM
4020 REM PLACE LOWBOAT
4030 POKE VC, XB : POKE VC + 1, YB
4040 GOSUB 2530 : REM PAUSE
4050 RETURN
4060 REM
4070 REM PLACE HIGHBOAT
4080 POKE VC + 2, XB : POKE VC + 3, YB
4090 GOSUB 2530 : REM PAUSE
4100 RETURN
4110 REM
4120 REM DATA FOR LOWBOAT
4130 DATA 16, 0, 0, 48, 0, 0, 112, 0, 0
4140 DATA 240, 0, 0, 112, 0, 0, 19, 254
4150 DATA 0, 19, 255, 0, 19, 13, 128
4160 DATA 19, 12, 192, 19, 12, 96, 19
4170 DATA 12, 48, 19, 12, 24, 19, 12
4180 DATA 12, 255, 255, 255, 255, 255
4190 DATA 254, 255, 255, 252, 255, 255
4200 DATA 248, 255, 255, 240, 255, 255
4210 DATA 224, 255, 255, 192, 255, 255
4220 DATA 128
4230 REM
4240 REM DATA FOR HIGHBOAT
4250 DATA 0, 0, 8, 0, 0, 12, 0, 0, 14
4260 DATA 0, 0, 15, 0, 0, 14, 0, 127
4270 DATA 204, 0, 255, 200, 1, 176, 200
4280 DATA 3, 48, 200, 6, 48, 200, 12
4290 DATA 48, 200, 24, 48, 200, 48, 48
4300 DATA 200, 255, 255, 255, 127, 255
4310 DATA 255, 63, 255, 255, 31, 255
4320 DATA 255, 15, 255, 255, 7, 255
4330 DATA 255, 3, 255, 255, 1, 255, 255

READY.
```

Commentary

The control module for Lock lies between lines 1020–1070.

The first subroutine called by the control module defines two sprites. The DATA for sprite 0 is READ by the loop between 1170–1190. This sprite represents a boat facing up-river and initially on the lower section of the river. (It is therefore referred to in the REMs as 'lowboat'.) The diagram of the 21 × 24 grid shows how lowboat is constructed. For example, the top of the flag, the first horizontal block of eight squares, is the binary number 10000 or 16 in base ten (our normal counting system). Checking line 4130 shows that the first number of DATA for sprite 0 is, indeed, 16. The DATA for sprite 1 is READ by the loop at lines 1240–1260. This defines highboat, a mirror image of lowboat, which at first appears on the upper section of the river and faces downstream. Both sprites are enlarged horizontally at line 1280 by POKEing VC + 29 with 3, the sum of the sprite numbers: 1 for sprite 0, and 2 for sprite 1. Both are coloured white at 1300 and 1310.

Diagram F



The display is set up by lines 1340–1770. It shows a green hillside and a blue river or canal in two sections, upper and lower, separated by the pair of lock gates. The display is built up with constant calls on a subroutine, lines 2480–2500, which fills in specified screen locations, L, with a chosen colour, C, as in earlier programs. For example, the hillside is constructed by lines 1420–1530. The subroutine is called by GOSUB 2490 at line 1480. The value of L is derived from the way in which the I and J loop variables are changing. This by itself would create the slope as a series of stepped bars, but the end of each bar is smoothed out by POKE L + 1, 105 at line 1520.

Similarly, the lower section of the river is constructed by the nested loops between lines 1560–1610 and the upper river by 1630–1680. The lock is superimposed on the river by drawing both lock gates in the closed position. There are four separate routines for opening and closing the lower and upper gates, between lines 1840–2100, and the two closing routines are called by GOSUB 1850 at line 1710 and GOSUB 1920 at line 1720. Note how the flag, F, is set equal to 1 at line 1700. This is because in the initial screen display the lock appears filled. F = 1 corresponds to a filled lock: F = 0 means that the lock has been drained to the lower water level. It is essential to have this representation of the status of the lock so that, when the boat is subsequently steered up and down the river, neither lock gate is ever opened to release a flood of water.

The screen display routine is followed by the PRINT TAB routine, needed for neat formatting of the screen prompts, and by the four lock gate routines already mentioned. Then follows a lock draining routine from lines 2120–2280. This simply changes the colour of location 56190 of the colour map to blue, at line 2140, to simulate a sluice opening in the lower gate. Similarly, the colour locations for the water in the lock are changed to green, by the instruction POKE 55751 + I + J, in the two nested loops from lines 2170–2220. Finally, line 2270 sets the flag F to 0. A very similar routine for filling the lock then follows at lines 2300–2460.

A short routine between lines 2560–2760 allows the child to choose whether a boat travels up or down through the lock. It is fully error-trapped and conveniently placed by calls on the PRINT TAB routine on the hillside above the lock. Two long routines then follow. Lines 2780–2280 steer the boat up-river and lines 3400–4000 steer it down-river. The two routines are identical in structure and both make many calls upon the routines already set up. Animation is achieved in two principal ways, by moving the two sprites by POKEing values to the relevant locations defined with respect to VC and by using the earlier subroutines to change colours of different parts of the screen. Both techniques can be used to produce animation in Commodore 64 programs. The two routines are

fully explained by the REMs they contain and should not be too difficult to understand.

Power

The second program, Power, continues the same theme of moving water. Here, however, the scene depicts a pumped storage electricity scheme with an upper and lower reservoir of water. Pipes connect the power station to each and a high tension line leads away off-screen.

Initially it is simply an idyllic scene taken from the Welsh countryside. The effect of rotation in the wheel is achieved by plotting three different sprites on top of each other, and turning them on and off in sequence. Electricity can actually be seen racing along the power line, either towards the station or away from it depending upon whether it is pumping water or generating electricity.

```
1000 REM POWER - PAT HALL, JAN '84
1010 REM
1020 GOSUB 1130 : REM DEFINE TURBINE
1030 GOSUB 1610 : REM EXPLANATION
1040 GOSUB 1920 : REM SET UP DISPLAY
1050 GOSUB 2610 : REM GENERATE
1060 GOSUB 2840 : REM STORE
1070 GOTO 1050
1080 END
1090 REM
1100 REM TURBINE WHEEL
1110 REM DEFINE SPRITES
1120 REM BEGINNING OF 6567 VIDEO CHIP
1130 VC = 53248
1140 REM SPRITE 0
1150 REM SET POINTER FOR SPRITE 0
1160 POKE 2040, 13
1170 REM DATA FOR SPRITE 0
1180 FOR I = 0 TO 26 STEP 3
1190 POKE 832 + I, 0 : POKE 865 + I, 0
1200 POKE 833 + I, 24 : POKE 866 + I,24
1210 POKE 834 + I, 0 : POKE 867 + I, 0
1220 NEXT I
1230 FOR I = 27 TO 32 STEP 3
1240 POKE 832 + I, 63
1250 POKE 833 + I, 255
```

```
1260 POKE 834 + I, 252
1270 NEXT I
1280 FOR I = 60 TO 62
1290 POKE 832 + I, 0
1300 NEXT I
1310 REM SPRITE 1
1320 REM SET POINTER FOR SPRITE 1
1330 POKE 2041, 14
1340 FOR I = 0 TO 62
1350 READ N : POKE 896 + I, N
1360 NEXT I
1370 REM SPRITE 2
1380 REM SET POINTER FOR SPRITE 2
1390 POKE 2042, 15
1400 REM READ DATA FOR SPRITE 2
1410 FOR I = 0 TO 62
1420 READ N : POKE 960 + I, N
1430 NEXT I
1440 REM ENLARGE SPRITES IN X DIRECTION
1450 POKE VC + 29, 7
1460 REM ENLARGE SPRITES IN Y DIRECTION
1470 POKE VC + 23, 7
1480 REM SELECT COLOUR
1490 FOR I = 39 TO 41
1500 POKE VC + I, 0
1510 NEXT I
1520 REM LOCATE SPRITES ON SCREEN
1530 FOR I = 0 TO 4 STEP 2
1540 POKE VC + I, 160
1550 POKE VC + I + 1, 161
1560 NEXT I
1570 RETURN
1580 REM
1590 REM EXPLANATION
1600 REM YELLOW SCREEN
1610 PRINT"[SHIFT/CLR/HOME]"
1620 POKE 53280, 7 : POKE 53281, 7
1630 PRINT CHR$( 144 )
1640 X = 14 : Y = 2 : GOSUB 3320
1650 PRINT "<< POWER >>"
1660 PRINT CHR$( 28 )
1670 X = 1 : Y = 6 : GOSUB 3320
1680 PRINT "SOME POWER STATIONS ";
1690 PRINT "CAN STORE ENERGY"
```

```
1700 PRINT "BY PUMPING WATER AT ";
1710 PRINT "NIGHT INTO A LAKE"
1720 PRINT "HIGH UP ON A NEARBY ";
1730 PRINT "MOUNTAIN." : PRINT
1740 PRINT "THEN DURING THE NEXT ";
1750 PRINT "DAY THE WATER IS"
1760 PRINT "ALLOWED TO RUSH BACK ";
1770 PRINT "DOWN THROUGH THE"
1780 PRINT "STATION'S GENERATORS, ";
1790 PRINT "MAKING MORE"
1800 PRINT "ELECTRICITY. IN THIS ";
1810 PRINT "WAY THE POWER"
1820 PRINT "GENERATED AT NIGHT ";
1830 PRINT "IS NOT WASTED."
1840 PRINT CHR$( 144 )
1850 X = 12 : Y = 18 : GOSUB 3320
1860 PRINT "PRESS ANY KEY"
1870 GET K$ : IF K$= "" THEN 1870
1880 PRINT CHR$( 5 )
1890 RETURN
1900 REM
1910 REM SET UP DISPLAY
1920 PRINT "[SHIFT/CLR/HOME]"
1930 REM WHITE BORDER
1940 POKE 53280, 1
1950 REM SKY
1960 POKE 53281, 3 : REM SELECT CYAN
1970 REM HILLSIDE
1980 C = 12 : REM SELECT GREY 2
1990 FOR I = 0 TO 560 STEP 40
2000 FOR J = 0 TO 24 + I / 40
2010 L = 1144 + I + J
2020 GOSUB 2560
2030 NEXT J
2040 L = L + 1
2050 POKE L + 54272, 12 : POKE L, 223
2060 NEXT I
2070 FOR I = 0 TO 240 STEP 40
2080 FOR J = 0 TO 39
2090 L = 1744 + I + J
2100 GOSUB 2560
2110 NEXT J
2120 NEXT I
2130 REM POWER STATION
```

```
2140 C = 1 : REM SELECT WHITE
2150 FOR I = 0 TO 240 STEP 40
2160 FOR J = 0 TO 9
2170 L = 1559 + I + J
2180 GOSUB 2560
2190 NEXT J
2200 NEXT I
2210 REM UPPER LAKE
2220 C = 6 : REM SELECT BLUE
2230 FOR I = 0 TO 200 STEP 40
2240 FOR J = 0 TO 9
2250 L = 1186 + I + J
2260 GOSUB 2560
2270 NEXT J
2280 NEXT I
2290 REM PIPES
2300 FOR L = 1431 TO 1671 STEP 40
2310 GOSUB 2560
2320 NEXT L
2330 FOR L = 1672 TO 1678
2340 GOSUB 2560
2350 NEXT L
2360 FOR L = 1844 TO 1964 STEP 40
2370 GOSUB 2560
2380 NEXT L
2390 FOR L = 1965 TO 1981
2400 GOSUB 2560
2410 NEXT L
2420 REM LOWER LAKE
2430 FOR L = 1932 TO 1941
2440 GOSUB 2560
2450 NEXT L
2460 REM ELECTRICITY CABLE
2470 C = 7 : REM SELECT YELLOW
2480 FOR L = 1784 TO 1798
2490 GOSUB 2560
2500 NEXT L
2510 REM SET PRINT TAB COORDINATES
2520 X = 30 : Y = 2
2530 RETURN
2540 REM
2550 REM PLOT CHARACTER SPACE
2560 POKE L + 54272, C : POKE L, 160
2570 RETURN
```

```
2580 REM
2590 REM GENERATE
2600 REM CYAN SKY
2610 POKE 53281, 3
2620 GOSUB 3320 : PRINT "DAY   "
2630 REM SELECT DIRECTION TO ROTATE
2640 D1 = 4 : D2 = 1
2650 E1 = 0 : E2 = 2
2660 FOR I = 0 TO 120 STEP 40
2670 FOR J = 0 TO 9
2680 REM DRAIN UPPER LAKE
2690 C = 12 : REM SELECT GREY 2
2700 L = 1186 + I + J
2710 GOSUB 2560
2720 REM FILL LOWER LAKE
2730 C = 6 : REM SELECT BLUE
2740 L = 1892 - I + J
2750 GOSUB 2560
2760 NEXT J
2770 REM ROTATE TURBINE
2780 GOSUB 3060
2790 NEXT I
2800 P = 10 : GOSUB 3370 : RETURN
2810 REM
2820 REM STORE
2830 REM BLACK SKY
2840 POKE 53281, 0
2850 GOSUB 3320 : PRINT "NIGHT"
2860 REM SELECT DIRECTION TO ROTATE
2870 D1 = 1 : D2 = 4
2880 E1 = 2 : E2 = 0
2890 FOR I = 0 TO 120 STEP 40
2900 REM ROTATE TURBINE
2910 GOSUB 3060
2920 FOR J = 0 TO 9
2930 REM DRAIN LOWER LAKE
2940 C = 12 : REM SELECT GREY 2
2950 L = 1772 + I + J
2960 GOSUB 2560
2970 REM FILL UPPER LAKE
2980 C = 6 : REM SELECT BLUE
2990 L = 1306 - I + J
3000 GOSUB 2560
3010 NEXT J
```

```
3020 NEXT I
3030 P = 10 : GOSUB 3370 : RETURN
3040 REM
3050 REM ROTATE TURBINE
3060 P = 1
3070 REM D1 AND D2 DETERMINE ROTATION
3080 FOR TW = 1 TO 3
3090 POKE VC + 21, D1
3100 E = E1 : GOSUB 3220
3110 POKE VC + 21, 0
3120 POKE VC + 21, 2
3130 E = 1 : GOSUB 3220
3140 POKE VC + 21, 0
3150 POKE VC + 21, D2
3160 E = E2 : GOSUB 3220
3170 NEXT TW
3180 RETURN
3190 REM
3200 REM ELECTRICITY
3210 REM E DETERMINES CURRENT DIRECTION
3220 FOR EL = 56070 TO 56056 STEP -3
3230 POKE EL - E, 2
3240 NEXT EL
3250 GOSUB 3370 : REM PAUSE
3260 FOR EL = 56070 TO 56056 STEP -3
3270 POKE EL - E, 7
3280 NEXT EL
3290 RETURN
3300 REM
3310 REM PRINT TAB ROUTINE
3320 POKE 782, X : POKE 781, Y
3330 SYS 65520
3340 RETURN
3350 REM
3360 REM PAUSE
3370 FOR T = 1 TO P * 200 : NEXT T
3380 RETURN
3390 REM
3400 REM DATA FOR TURBINE WHEEL
3410 REM
3420 REM SPRITE 1
3430 DATA 0, 128, 0, 0, 192, 0, 0, 192
3440 DATA 0, 0, 96, 0, 0, 96, 0, 0, 96
3450 DATA 0, 0, 48, 28, 0, 48, 248, 0
```

```
3460 DATA 19, 224, 0, 31, 0, 0, 248, 0
3470 DATA 7, 200, 0, 31, 12, 0, 56, 12
3480 DATA 0, 0, 6, 0, 0, 6, 0, 0, 6, 0
3490 DATA 0, 3, 0. 0, 3, 0, 0, 1, 0, 0
3500 DATA 0, 0
3510 REM
3520 REM SPRITE 2
3530 DATA 0, 1, 0, 0, 3, 0, 0, 3, 0, 0
3540 DATA 6, 0, 0, 6, 0, 0, 6, 0, 56
3550 DATA 12, 0, 31, 12, 0, 7, 200, 0
3560 DATA 0, 248, 0, 0, 31, 0, 0, 19
3570 DATA 224, 0, 48, 248, 0, 48, 28, 0
3580 DATA 96, 0, 0, 96, 0, 0, 96, 0, 0
3590 DATA 192, 0, 0, 192, 0, 0, 128, 0
3600 DATA 0, 0, 0
3610 POKE 55831 + I + J, 1

READY.
```

Commentary

Power is controlled by lines 1020–1060. The routine called by line 1020 defines the three sprites needed for the animation of the turbine wheel in the power station. The next routine called is simply a screen of text, explaining the purpose of the computer simulation and giving a little information about pumped power schemes. Then line 1040 calls the routine which sets up the display. After that, the two routines, 'Generate' and 'Store', cycle indefinitely, the first representing the generation of electricity during the day and the second the storing of energy at night by pumping water back up to the higher reservoir.

The turbine wheel is defined as a superimposition on the correct part of the display of sprites 0–2. The diagrams show how these three sprites represent different positions of the turbine as it rotates. Subsequently, by switching the three sprites on and off in the correct sequence, the wheel can be made to rotate either clockwise or anticlockwise. Line 1130 initialises VC, the starting location for the video chip. The three loops between lines 1180–1300 then define the necessary numbers for the creation of sprite 0. For example, the loop between lines 1180–1220 gives the values which produce the top and bottom rotor arms of the turbine.

The two remaining sprites are not so symmetrical and so the information needed to produce them is READ from lines of DATA by the loops between lines 1340–1360 and 1410–1430. Lines 1450 and 1470 enlarge all three sprites in both horizontal and vertical directions by POKEing

Diagram G

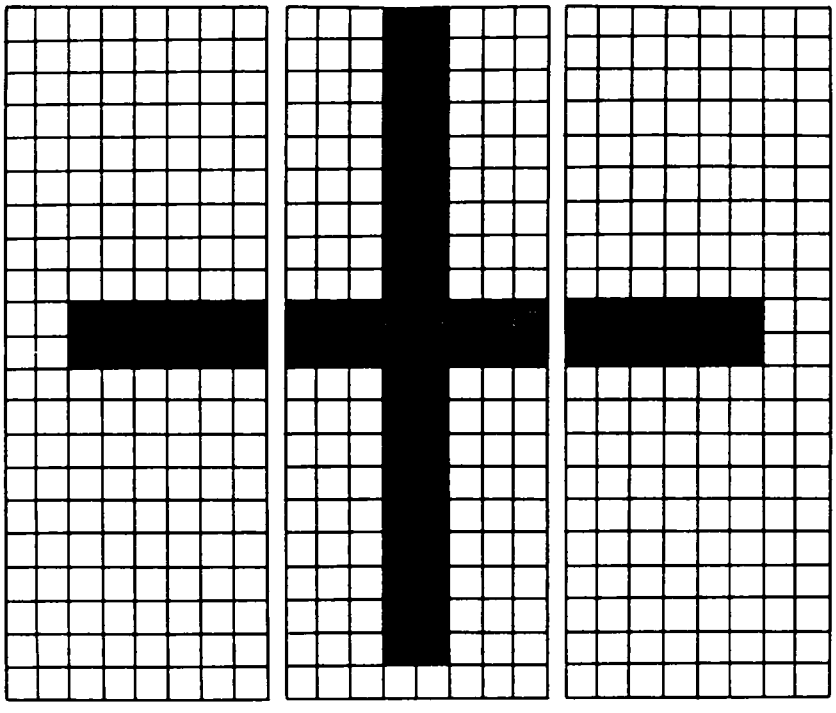


Diagram H

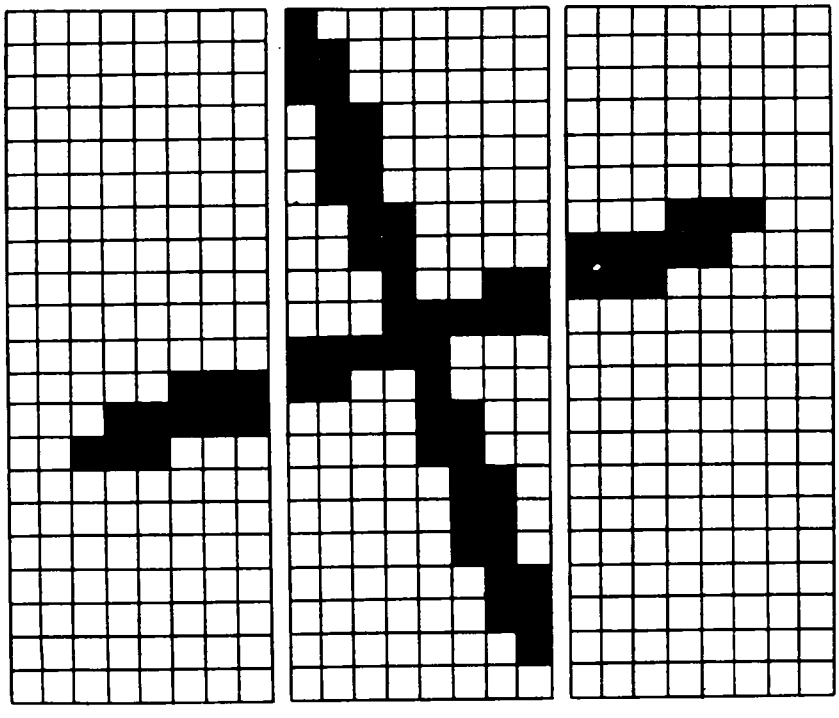
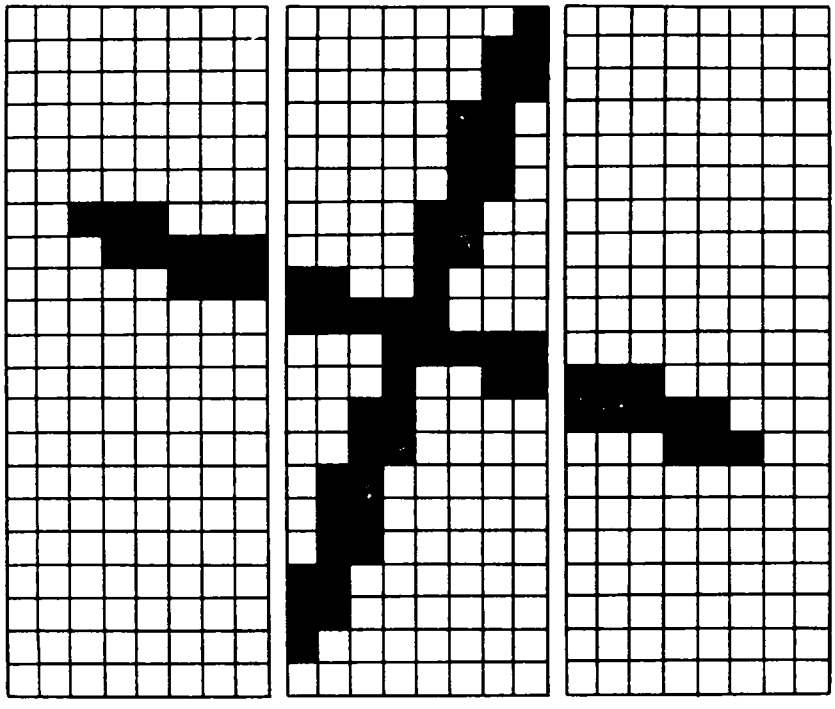


Diagram I



VC + 29 and VC + 23 by the combined sprite number of 7 (ie 1 + 2 + 4). The loop between lines 1490–1510 colours the sprites black and the loop from lines 1530–1560 places all three on the same part of the screen.

The routine from lines 1590–1890 gives an explanation of the program on the screen. Line 1870 governs the further execution of the program and is linked to the screen prompt at 1860.

The screen display is built up by the routine between lines 1910–2530. GOSUB 2560 is used frequently to call the subroutine between lines 2550–2570 to colour in specified screen locations. Lines 1980–2120 produce a grey hillside and lines 2140–2200 a white power station, which coincides with the position of the turbine wheel's sprites on the screen. The upper reservoir is drawn by lines 2220–2280 and the lower by lines 2430–2450. Other FOR...NEXT loops with suitable ranges plot the pipes connecting the power station to the reservoirs and also, in yellow rather than blue, the electricity 'cable'.

The routine between lines 2590–2800 simulates the generation of electricity during the day. Line 2610 gives a pale blue sky to represent day. (Cyan rather than blue is used to allow contrast with the water.) Lines 2640 and 2650 define values for the variables D1, D2, E1 and E2 needed for the routines which animate the turbine and the electricity in the cable. Then the loop between lines 2670–2760 changes the colour of one row of character spaces in the upper reservoir to grey and one row in the lower reservoir to blue. This, repeated by the loop between lines 2660–2790, gives the effect of water draining through the turbine. Line 2770 calls the turbine routine, lines 3050–3180.

As explained above, the turbine is made to rotate by switching on and off the three sprites in the correct order. This is done in the TW loop from lines 3080–3170. Lines 3090, 3120 and 3150 switch on the three sprites in turn by POKEing VC + 21, and lines 3110 and 3140 switch them off again by POKEing VC + 21, 0 after a suitable short pause. The sprites switched on at lines 3090 and 3150 are determined by D1 and D2 and this allows the direction of rotation to be specified. The pause between different positions of the wheel is produced by the routine between lines 3310–3290, which also animates the flow of electricity in the cable. The electricity is produced by the FOR...NEXT loop between lines 3220–3240. Here, the colour locations for the cable are altered in a way, determined by the value of E (either E1 or E2), which gives the illusion of current flowing either into or out of the power station.

Lines 2820–3030 form an identically structured routine in which water and electricity both flow in the opposite direction to simulate the energy being stored at night in the upper reservoir.

Coin

The final two simulations are in elementary statistics. Coin allows the child to drop up to 25 coins and see what fraction is heads. An animated hand and coin are shown together with a block graph of results which is continuously updated. The limiting value of the fraction is clearly shown by the graph plotted.

There would be little point in using this simulation unless children already had experience of coin-tossing experiments. However, once they had a grasp of the basic concepts involved, the program could be used to great effect. Its endless repetition, crunching of difficult numbers and display of the approaching limit of 0.5, would all help towards an appreciation of probability as a real mathematical concept.

```
1000 REM COIN - PAT HALL, JAN '84
1010 REM
1020 GOSUB 1120 : REM DEFINE SPRITES
1030 GOSUB 1520 : REM SET UP DISPLAY
1040 FOR TH = 1 TO 25
1050 GOSUB 1890 : REM DROP COIN
1060 GOSUB 2030 : REM RESULT
1070 NEXT TH
1080 END
1090 REM
1100 REM DEFINE SPRITES
1110 REM BEGINNING OF 6567 VIDEO CHIP
1120 VC = 53248
1130 REM COIN, SPRITE 0
1140 REM SET POINTER FOR SPRITE 0
1150 POKE 2040, 13
1160 REM READ DATA FOR SPRITE 0
1170 FOR I = 0 TO 62
1180 READ N : POKE 832 + I, N
1190 NEXT I
1200 REM HAND, SPRITE 1
1210 REM SET POINTER FOR SPRITE 1
1220 POKE 2041, 14
1230 REM READ DATA FOR SPRITE 1
1240 FOR I = 0 TO 20
1250 READ N : POKE 896 + I, N
1260 NEXT I
```

```
1270 FOR I = 21 TO 44
1280 POKE 896 + I, 255
1290 NEXT I
1300 FOR I = 45 TO 62
1310 READ N : POKE 896 + I, N
1320 NEXT I
1330 REM HAND, SPRITE 2
1340 REM SET POINTER FOR SPRITE 2
1350 POKE 2042, 15
1360 REM READ DATA FOR SPRITE 2
1370 FOR I = 0 TO 62
1380 READ N : POKE 960 + I, N
1390 NEXT I
1400 REM ENLARGE SPRITES IN X DIRECTION
1410 POKE VC + 29, 6
1420 REM ENLARGE SPRITES IN Y DIRECTION
1430 POKE VC + 23, 6
1440 REM SELECT COLOUR
1450 POKE VC + 39, 0
1460 POKE VC + 40, 8
1470 POKE VC + 41, 8
1480 RETURN
1490 REM
1500 REM SET UP DISPLAY
1510 REM CYAN SCREEN
1520 POKE 53280, 3 : POKE 53281, 3
1530 PRINT "[SHIFT/CLR/HOME]" : PRINT CHR$( 144 )
1540 REM LOCATE HAND ON SCREEN
1550 POKE VC + 2, 30 : POKE VC + 3, 55
1560 POKE VC + 4, 78 : POKE VC + 5, 55
1570 REM DRAW GRAPH
1580 REM AXES
1590 FOR I = 1118 TO 1918 STEP 40
1600 POKE I + 54272, 1 : POKE I, 160
1610 NEXT I
1620 FOR I = 1919 TO 1943
1630 POKE I + 54272, 1 : POKE I, 160
1640 NEXT I
1650 REM COORDINATE SQUARES
1660 FOR I = 0 TO 760 STEP 40
1670 FOR J = 0 TO 24
1680 POKE 55391 + I + J, 1
1690 POKE 1119 + I + J, 80
1700 NEXT J
1710 NEXT I
1720 REM PRINT TITLE
1730 X = 13 : Y = 2 : GOSUB 2280
1740 PRINT "1"
1750 Y = 22 : GOSUB 2280
```

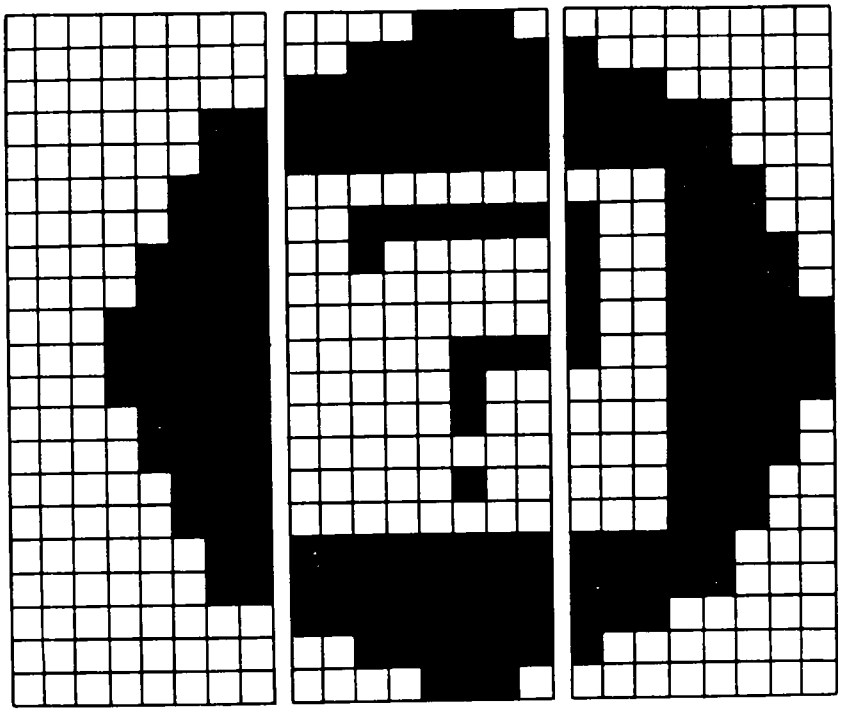
```
1760 PRINT "0"
1770 X = 19 : Y = 0 : GOSUB 2280
1780 PRINT "FRACTION OF HEADS"
1790 REM SET PRINT TAB COORDINATES
1800 X = 5 : Y = 22
1810 REM RANDOMISE
1820 N = RND( -TI )
1830 REM INITIALISE H AND N
1840 H = 0 : N = 0
1850 RETURN
1860 REM
1870 REM DROP COIN
1880 REM SHOW HAND
1890 POKE VC + 21, 6
1900 GOSUB 2330 : REM PAUSE
1910 REM SHOW COIN
1920 POKE VC + 21, 7
1930 REM DROP COIN
1940 FOR I = 100 TO 225
1950 POKE VC, 70 : POKE VC + 1, I
1960 NEXT I
1970 GOSUB 2330 : REM PAUSE
1980 POKE VC + 21, 0
1990 RETURN
2000 REM
2010 REM RESULT
2020 REM DECIDE RESULT
2030 F = 0
2040 IF RND( 1 ) > .5 THEN F = 1
2050 REM SHOW RESULT OF THROW
2060 GOSUB 2280
2070 IF F = 0 THEN PRINT "TAILS"
2080 IF F = 1 THEN PRINT "HEADS"
2090 REM DRAW GRAPH
2100 REM INCREMENT THROW
2110 N = N + 1
2120 REM INCREMENT HEADS
2130 H = H + F
2140 REM CALCULATE HEIGHT OF BAR
2150 HT = INT( 20 * H / N ) * 40
2160 IF H = 0 THEN 2220
2170 REM PLOT BAR ON GRAPH
2180 FOR I = 1878 TO 1918 - HT STEP -40
2190 POKE I + N + 54272, 0
2200 POKE I + N, 160
2210 NEXT I
2220 GOSUB 2330 : REM PAUSE
2230 REM ERASE RESULT
2240 GOSUB 2280 : PRINT "      "
```

```
2250 RETURN
2260 REM
2270 REM PRINT TAB ROUTINE
2280 POKE 782, X : POKE 781, Y
2290 SYS 65520
2300 RETURN
2310 REM
2320 REM PAUSE
2330 FOR T = 1 TO 1000 : NEXT T
2340 RETURN
2350 REM
2360 REM
2370 REM DATA FOR COIN, SPRITE 0
2380 REM
2390 DATA 0, 14, 0, 0, 63, 128, 0, 255
2400 DATA 224, 3, 255, 248, 3, 255, 248
2410 DATA 7, 0, 28, 7, 63, 156, 15, 32
2420 DATA 158, 15, 0, 158, 31, 0, 159
2430 DATA 31, 7, 159, 31, 4, 31, 15, 4
2440 DATA 30, 15, 0, 30, 7, 4, 28, 7, 0
2450 DATA 28, 3, 255, 248, 3, 255, 248
2460 DATA 0, 255, 224, 0, 63, 128, 0
2470 DATA 14, 0
2480 REM
2490 REM DATA FOR HAND, SPRITE 1
2500 REM
2510 DATA 0, 0, 0, 0, 0, 0, 0, 0, 1, 0
2520 DATA 0, 3, 0, 0, 3, 255, 224, 7
2530 DATA 255, 255, 31, 255, 255, 207
2540 DATA 255, 252, 15, 0, 0, 7, 0, 0
2550 DATA 1, 0, 0, 0, 0, 0, 0
2560 REM
2570 REM DATA FOR HAND, SPRITE 2
2580 REM
2590 DATA 31, 192, 0, 127, 192, 0, 255
2600 DATA 0, 248, 240, 15, 252, 224, 63
2610 DATA 248, 255, 255, 0, 255, 240, 0
2620 DATA 255, 255, 254, 255, 255, 255
2630 DATA 255, 255, 254, 255, 252, 0
2640 DATA 255, 240, 0, 255, 255, 252
2650 DATA 255, 255, 254, 255, 255, 252
2660 DATA 255, 224, 0, 255, 255, 240
2670 DATA 255, 255, 248, 255, 255, 240
2680 DATA 127, 255, 240, 63, 128, 0

READY.
```

Commentary

Coin is controlled by lines 1020–1070. Line 1020 calls the routine which defines three sprites, one for the coin being tossed and the other two to produce the hand which tosses it. Line 1030 sets up the screen display and finally the FOR...NEXT loop from 1040 to 1070 tosses twenty-five coins in turn, and draws a block graph of the result.

Diagram J



Lines 1100–1480 define the three sprites. Sprite 0 is the coin shown in the diagram. There is a question mark in the middle of the coin to indicate that as it falls the outcome of the result is in doubt. Sprites 1 and 2 define a hand and these are both enlarged by POKE VC + 29, 6 and POKE VC + 23, 6 at lines 1410 and 1430. The coin is left unenlarged for contrast.

The screen display is set up by lines 1500–1850. A cyan screen is created by 1520. The two FOR...NEXT loops from 1590–1610 and 1620–1710 add the coordinate squares by POKEing character 80 to the screen. Lines 1730–1780 calibrate and title the graph. At line 1840, the variables H and N are initialised: H records the total number of heads thrown at any stage of the program's execution and N is the total number of throws.

Lines 1870–1990 animate the fall of the coin. Line 1890 switches on the hand. After a pause, line 1920 switches on the coin. Then the FOR... NEXT loop between lines 1940–1960 moves the coin down the screen.

The routine between lines 2010–2250 decides and records the result of the throw. Line 2040 uses RND(1) to select heads or tails. F is a flag indicating the result: F = 0 is a tail and F = 1 a head. This result is recorded on the screen by lines 2070 and 2080. Line 2110 then increments the number of throws and 2130 the total number of heads. The current fraction of heads is therefore H/N. Line 2150 then decides how tall a bar has to be added to the graph to represent this result. The bar is plotted by the loop between lines 2180–2210.

Cascade

Probability is also the subject of Cascade. This program sets up an array of pegs and allows marbles to fall down, the totals for each column appearing beneath the bottom row. This arrangement shows the different paths possible for the balls to travel, and is a practical demonstration of Pascal's triangle.

The computer simulation has, however, a number of advantages over the usual pegboard and nails approach. Children can easily select how many rows they want for the array, up to seven. Additionally, the program menu allows the selection of an alternative mode in which no marbles appear, but instead each possible path through the pegs is plotted individually.

```
1000 REM CASCADE - PAT HALL, JAN '83
1010 REM
1020 GOSUB 1110 : REM INITIALISATION
1030 GOSUB 1180 : REM EXPLANATION
1040 REM CHOOSE CASCADE OR PATHS
1050 IF CH = 1 THEN GOSUB 1550
1060 IF CH = 2 THEN GOSUB 1860
1070 END
1080 REM
1090 REM INITIALISATION
1100 REM GREEN SCREEN
1110 POKE 53280, 5 : POKE 53281, 5
1120 PRINT"[SHIFT/CLR/HOME]"
1130 REM SET RANDOMNESS
```

```
1140 N = RND( -TI )
1150 RETURN
1160 REM
1170 REM EXPLANATION
1180 PRINT CHR$( 5 )
1190 X = 13 : Y = 2 : GOSUB 2030
1200 PRINT "<< CASCADE >>"
1210 PRINT CHR$( 144 )
1220 PRINT
1230 PRINT " THIS PROGRAM ";
1240 PRINT "DEMONSTRATES PASCAL'S"
1250 PRINT "TRIANGLE IN STATISTICS. ";
1260 PRINT "YOU CAN WATCH"
1270 PRINT "MARBLES FALLING DOWN ";
1280 PRINT "A TRIANGLE MADE"
1290 PRINT "OF PEGS AND SEE THE ";
1300 PRINT "WAY THEY COLLECT"
1310 PRINT "AT THE BOTTOM."
1320 PRINT : PRINT CHR$( 5 )
1330 PRINT " PLEASE CHOOSE WHETHER ";
1340 PRINT "YOU WANT TO SEE"
1350 PRINT "MARBLES FALLING DOWN ";
1360 PRINT "THE TRIANGLE OR"
1370 PRINT "TO LOOK AT ALL THEIR ";
1380 PRINT "POSSIBLE PATHS."
1390 PRINT : PRINT CHR$( 144 )
1400 PRINT " TYPE 1 TO SEE THE ";
1410 PRINT "MARBLES FALLING." : PRINT
1420 PRINT " TYPE 2 TO LOOK AT ALL ";
1430 PRINT "THEIR PATHS." : PRINT
1440 X = 1 : Y = 21 : GOSUB 2030
1450 PRINT "                "
1460 GOSUB 2030
1470 INPUT CH$
1480 IF ASC( CH$) < 48 THEN 1440
1490 IF ASC(CH$) > 57 THEN 1440
1500 CH = VAL( CH$)
1510 IF CH <> 1 AND CH <> 2 THEN 1440
1520 RETURN
1530 REM
1540 REM CASCADE
1550 PRINT "[SHIFT/CLR/HOME]"
1560 X = 7 : Y = 6 : GOSUB 2030
1570 PRINT "TYPE NUMBER OF ROWS 2 - 7"
```

```
1580 Y = 8 : GOSUB 2030
1590 PRINT "                "
1600 GOSUB 2030
1610 INPUT R$
1620 IF ASC( R$ ) < 49 THEN 1580
1630 IF ASC( R$ ) > 57 THEN 1580
1640 R = VAL( R$)
1650 IF INT( R ) <> R THEN 1580
1660 IF R < 2 OR R > 7 THEN 1580
1670 Y = 10 : GOSUB 2030
1680 PRINT "TYPE NUMBER OF MARBLES"
1690 Y = 12 : GOSUB 2030
1700 PRINT "                "
1710 GOSUB 2030
1720 INPUT M$
1730 IF ASC( M$ ) < 49 THEN 1690
1740 IF ASC( M$ ) > 57 THEN 1690
1750 M = VAL( M$ )
1760 IF INT( M ) <> M OR M< 1 THEN 1690
1770 PRINT CHR$( 5 )
1780 GOSUB 2080 : REM PEGS
1790 REM DROP M MARBLES
1800 FOR DR = 1 TO M
1810 GOSUB 2200
1820 NEXT DR
1830 RETURN
1840 REM
1850 REM PATHS
1860 PRINT "[SHIFT/CLR/HOME]"
1870 X = 7 : Y = 6 : GOSUB 2030
1880 PRINT "TYPE NUMBER OF ROWS 3 - 6"
1890 Y = 8 : GOSUB 2030
1900 PRINT "                "
1910 GOSUB 2030
1920 INPUT R$
1930 IF ASC( R$ ) < 49 THEN 1890
1940 IF ASC( R$ ) > 57 THEN 1890
1950 R = VAL( R$)
1960 IF INT( R ) <> R THEN 1890
1970 IF R < 3 OR R > 6 THEN 1890
1980 GOSUB 2080 : REM PEGS
1990 GOSUB 2490 : REM PATHS
2000 RETURN
2010 REM
```

```
2020 REM PRINT TAB ROUTINE
2030 POKE 782, X : POKE 781, Y
2040 SYS 65520
2050 RETURN
2060 REM
2070 REM PEGS
2080 PRINT "[SHIFT/CLR/HOME]"
2090 S = 1044 + INT( 13.5 - R*1.5 )*40
2100 FOR I = 0 TO R - 1
2110 LV = S + I * 118
2120 FOR J = 0 TO I
2130 LH = LV + J * 4
2140 POKE LH + 54272, 0 : POKE LH, 90
2150 NEXT J
2160 NEXT I
2170 RETURN
2180 REM
2190 REM DROP
2200 LB = S - 40 : GOSUB 2340
2210 A = 0
2220 FOR I = 1 TO R - 1
2230 D = -1
2240 IF RND(1) > .5 THEN D=1 : A=A+1
2250 FOR J = 1 TO 2
2260 LB = LB + 40 + D : GOSUB 2340
2270 NEXT J
2280 LB = LB + 40 : GOSUB 2340
2290 NEXT I
2300 GOSUB 2400
2310 RETURN
2320 REM
2330 REM PLACE BALL ON SCREEN
2340 POKE LB + 54272, 1 : POKE LB, 81
2350 FOR T = 1 TO 150 : NEXT T
2360 POKE LB + 54272, 5
2370 RETURN
2380 REM
2390 REM PRINT TOTAL FOR COLUMN
2400 TC( A ) = TC( A ) + 1
2410 X = 21.5 - R * 2 + A * 4
2420 Y = 12.5 + R * 1.5
2430 GOSUB 2030
2440 PRINT TC( A )
2450 RETURN
```

```
2460 REM
2470 REM DETERMINE ALL PATHS
2480 REM CALCULATE BINARY NUMBERS
2490 N = 2 ^ ( R - 1 ) - 1
2500 DIM E( N ) : DIM P$( N )
2510 FOR I = 0 TO N
2520 B = I
2530 C = 0 : D = 0.1
2540 D = D * 10
2550 B = B / 2
2560 IF INT( B )<>B THEN C = C + D
2570 IF INT(B) <> B THEN E(I) = E(I)+1
2580 B = INT( B )
2590 IF B <> 0 THEN 2540
2600 REM IDENTIFY PATH AS BINARY STRING
2610 P$( I ) = STR$( C )
2620 IF LEN( P$(I))>= R THEN 2640
2630 P$(I) = "0" + P$(I) : GOTO 2620
2640 NEXT I
2650 REM SORT INTO ORDER
2660 SW = 0
2670 FOR I = 1 TO N - 1
2680 IF E( I ) <= E( I + 1 ) THEN 2730
2690 SW = 1
2700 B=E(I) : E(I) = E(I+1) : E(I+1)=B
2710 B$ = P$( I ) : P$( I ) = P$( I+1 )
2720 P$( I + 1 ) = B$
2730 NEXT I
2740 IF SW = 1 THEN 2660
2750 REM PLOT EACH PATH IN TURN
2760 FOR I = 0 TO N
2770 A = 0 : C = 1
2780 LP = S - 40
2790 POKE LP + 54272, C : POKE LP, 34
2800 FOR J = 1 TO R + 1
2810 IF MID$(P$(I),J,1)<>"0"THEN2830
2820 D = -1 : GOSUB 2930
2830 IF MID$(P$(I),J,1)<>"1"THEN2850
2840 D = 1 : A = A + 1 : GOSUB 2930
2850 NEXT J
2860 IF C = 1 THEN GOSUB 2400
2870 FOR T = 1 TO 500 : NEXT T
2880 IF C = 1 THEN C = 5 : GOTO 2780
2890 NEXT I
```

```
2900 RETURN
2910 REM
2920 REM MARK PATH
2930 FOR K = 1 TO 2
2940 LP = LP + 40 + D
2950 POKE LP + 54272, C : POKE LP, 34
2960 NEXT K
2970 LP = LP + 40
2980 POKE LP + 54272, C : POKE LP, 34
2990 RETURN

READY.
```

Commentary

Cascade is controlled by lines 1020–1060. Line 1020 calls an initialisation routine which gives a green screen and sets the randomness. Line 1030 calls the explanation routine, and then the value returned for the variable CH selects either the 'Cascade' or 'Paths' routine. The cascade routine shows a series of marbles falling down the screen, being deflected by a triangular array of pegs and collecting at the bottom in columns reflecting the distribution of Pascal's triangle. The paths routine works systematically through the entire diagram, showing all the possible routes a given marble could follow. The cascade routine gives an experimental set of numbers at the bottom of the pegs, whereas numbers produced by the paths routine actually are the values found in Pascal's triangle.

Lines 1090–1150 form the initialisation routine.

Lines 1170–1520 place explanatory text on the screen. At the end of the routine, a value for CH is obtained which selects which of the two possible simulations of Pascal's triangle is then executed by the computer.

The cascade routine is between lines 1540–1830. At the beginning of the routine, lines 1560–1760 request the number of rows of pegs, R, and the number of marbles to be dropped, M. Line 1780 then calls the routine which draws the array of pegs on the screen with GOSUB 2080. Finally, the loop between lines 1800–1820 uses GOSUB 2200 to drop M marbles down the screen.

The paths routine follows, between lines 1850–2000. Lines 1870–1970 request the number of rows for the array of pegs. GOSUB 2080 places the pegs on the screen and GOSUB 2490 plots all the possible paths.

The pegs are placed on the screen by lines 2070–2170. R is the number of rows. Since the vertical spacing of the pegs leaves two rows of the screen free between each row with pegs, this means that the total height of a triangular array of R pegs is R*3 – 2. As there are 25 rows altogether on the screen, the top margin above the array will be 25 – (R*3 – 2) divided by 2. This simplifies to 13.5 – R*1.5. Hence the location of the top peg of the array must be 1044 + INT(13.5 – R*1.5)*40, as calculated at line 2090. The location of the first peg of each successive row, LV, is found by incrementing this value by 118. This corresponds to three rows down and two squares back, ie 40*3 – 2. This expression appears in the I loop at line 2110. Finally, the horizontal position of each peg in any row, LH, is obtained by incrementing the value of LV by four for each peg. This is done by the J loop at line 2130. Line 2140 actually places the peg on the screen by POKE LH, 90.

The marble is dropped down the screen by lines 2190–2310. LB is each location calculated for the falling marble. GOSUB 2340 calls a separate routine which then places the ball on the screen and erases it after a short pause. The initial value of LB must be S – 40, at line 2200, because the ball begins immediately above the top peg. The I loop which follows now uses RND(1) to decide whether the ball falls to the left or to the right. D is a flag indicating the direction: D = – 1 represents a marble bouncing to the left, and D = 1 is a marble going to the right. Line 2260 adjusts the position of the marble accordingly: LB = LB + 40 + D. If the marble is deflected to the right, the variable A is also incremented. This is to allow the final column arrived at to be identified. The subroutine called by line 2300 increments the value of the array variable TC(A) and places this total on the screen. It records the total number of marbles which have fallen into that column. Lines 2410 and 2420 make sure that each column's total is printed in the correct place.

All the possible paths are shown on the screen by the routine between lines 2470–2990. The algorithm used to do this first converts all the decimal numbers, up to the number of possible paths, into binary notation. This is done by lines 2490–2590. Thus the eleventh path would become 1011. This binary number is then used as a route map through the pegs. The number is scanned digit by digit and the 1 or 0 used to give the direction to turn. Lines 2610–2630 turn each of the binary numbers into a string of equal length and then the routine between lines 2660–2740 sorts all these routes into order, so that they are shown subsequently on the screen starting at the left and working across gradually to the righthand side of the pegs. The path is plotted through the pegs by lines 2760–2890. Line 2810 uses MID$ to detect a deflection to the left and line 2830 similarly discovers deflections to the right. The path is erased after a pause by the colour change at 2880.

Other titles from Sunshine

SPECTRUM BOOKS

Spectrum Adventures
A guide to playing and writing adventures
Tony Bridge & Roy Carnell
ISBN 0 946408 07 6
£5.95

ZX Spectrum Astronomy
Maurice Gavin
ISBN 0 946408 24 6
£6.95

Spectrum Machine Code Applications
David Laine
ISBN 0 946408 17 3
£6.95

The Working Spectrum
David Lawrence
ISBN 0 946408 00 9
£5.95

Master your ZX Microdrive
Andrew Pennell
ISBN 0 946408 19 X
£6.95

COMMODORE 64 BOOKS

Graphic Art for the Commodore 64
Boris Allan
ISBN 0 946408 15 7
£5.95

DIY Robotics and Sensors on the Commodore Computer
John Billingsley
ISBN 0 946408 30 0
£6.95

Artificial Intelligence on the Commodore 64
Keith & Stephen Brain
ISBN 0 946408 29 7
£6.95

Commodore 64 Adventures
Mike Grace
ISBN 0 946408 11 4
£5.95

Business Applications for the Commodore 64
James Hall
ISBN 0 946408 12 2
£5.95

Mathematics on the Commodore 64
Czes Kosniowski
ISBN 0 946408 14 9
£5.95

Advanced Programming Techniques on the Commodore 64
David Lawrence
ISBN 0 946408 23 8
£5.95

The Working Commodore 64
David Lawrence £5.95
ISBN 0 946408 02 5
Commodore 64 Machine Code Master
David Lawrence & Mark England £6.95
ISBN 0 946408 05 X

ELECTRON BOOKS

Graphic Art for the Electron Computer
Boris Allan £5.95
ISBN 0 946408 20 3
Programming for Education on the Electron Computer
John Scriven & Patrick Hall £5.95
ISBN 0 946408 21 1

BBC COMPUTER BOOKS

Functional Forth for the BBC Computer
Boris Allan £5.95
ISBN 0 946408 04 1
Graphic Art for the BBC Computer
Boris Allan £5.95
ISBN 0 946408 08 4
DIY Robotics and Sensors for the BBC Computer
John Billingsley £6.95
ISBN 0 946408 13 0
Essential Maths on the BBC and Electron Computer
Czes Kosniowski £5.95
ISBN 0 946408 34 3
Programming for Education on the BBC Computer
John Scriven & Patrick Hall £5.95
ISBN 0 946408 10 6
Making Music on the BBC Computer
Ian Waugh £5.95
ISBN 0 946408 26 2

DRAGON BOOKS

Advanced Sound & Graphics for the Dragon
Keith & Steven Brain £5.95
ISBN 0 946408 06 8
Dragon 32 Games Master
Keith & Steven Brain £5.95
ISBN 0 946408 03 3

The Working Dragon
David Lawrence
ISBN 0 946408 01 7
£5.95

The Dragon Trainer
A handbook for beginners
Brian Lloyd
ISBN 0 946408 09 2
£5.95

ATARI BOOKS

Atari Adventures
Tony Bridge
ISBN 0 946408 18 1
£5.95

Writing Strategy Games on your Atari Computer
John White
ISBN 0 946408 22 X
£5.95

Sunshine also publishes

POPULAR COMPUTING WEEKLY

The first weekly magazine for home computer users. Each copy contains Top 10 charts of the best-selling software and books and up-to-the-minute details of the latest games. Other features in the magazine include regular hardware and software reviews, programming hints, computer swap, adventure corner and pages of listings for the Spectrum, Dragon, BBC, VIC 20 and 64, ZX 81 and other popular micros. Only 35p a week, a year's subscription costs £19.95 (£9.98 for six months) in the UK and £37.40 (£18.70 for six months) overseas.

DRAGON USER

The monthly magazine for all users of Dragon microcomputers. Each issue contains reviews of software and peripherals, programming advice for beginners and advanced users, program listings, a technical advisory service and all the latest news related to the Dragon. A year's subscription (12 issues) costs £10 in the UK and £16 overseas.

MICRO ADVENTURER

The monthly magazine for everyone interested in Adventure games, war gaming and simulation/role-playing games. Includes reviews of all the latest software, lists of all the software available and programming advice. A year's subscription (12 issues) costs £10 in the UK and £16 overseas.

COMMODORE HORIZONS

The monthly magazine for all users of Commodore computers. Each issue contains reviews of software and peripherals, programming advice for beginners and advanced users, program listings, a technical advisory service and all the latest news. A year's subscription costs £10 in the UK and £16 overseas.

For further information contact:
Sunshine
12–13 Little Newport Street
London WC2R 3LD
01-437 4343

Printed in England by Commercial Colour Press, London E7.

If you have recently bought a Commodore 64 and are looking for useful learning programs then this is the book for you.

Teachers and parents will find it provides a source of programs in a variety of subjects. Aimed primarily at younger children these programs show how the Commodore 64 can be used as a learning machine as well as teaching different programming skills.

Using a modular approach the programs are developed in stages and can be freely adapted to suit a variety of educational needs. Although a knowledge of BASIC is not essential using this book will encourage you to develop your programming style in novel and interesting ways.

Both authors have helped to introduce computers to schools in Hampshire. Patrick Hall is Deputy Head Teacher of a junior school in Hampshire. He has run in-service courses for teachers in computing. John Scriven has taught in both primary and secondary schools for several years as well as lecturing in computing. He is a regular contributor to Popular Computing Weekly.

ISBN 0 946408 27 0

£5.95 net