

# MAKING THE MOST OF YOUR ZX80

Tim Hartnell





**Making the most  
of  
your ZX80**

**by**

**Tim Hartnell**

© Tim Hartnell 1980

This book was set in Oracle by  
The Drawing Room and printed by  
Westcrete Ltd of Ashford, Middx.

ISBN 0 907442 00 5

Published by Computer Publications  
Unit 3, 33 Woodthorpe Road,  
Ashford, Middlesex TW15 2RP

All Rights Reserved. This book, or parts of it, may not be reproduced in any form, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publisher.

# Contents

	Page
Introduction	1
New ROM/Old ROM	3
Random numbers	4
Simple decision makers	7
Russian Roulette	11
Pattern makers	15
Building a library	16
Review	16
For/Next	17
Flipping a coin	18
Decision maker — Mark 11	21
Number crunching	22
Writing a program — some general thoughts	25
Extra-Sensory Perception	27
Kill the Chopper	29
Days of our Lives	31
Toying with the Muse	34
Hunting on a grid	38
Slot/fruit machines	40
Lost in space	43
Arrays	44
Strings and ladders	50
Hot sauce	54
Designs on your VDU	56
First steps towards the stars	58
Doing it in your head	62
Moving graphics	64
ZX80 Active display	68
Let the games begin:	70
Did He who made the Lamb make the UFO	71
Laser roulette	72
High noon at the Old Intergalactic	73

Time warped to space	75
Turning the tables	76
A ching in his armour	78
Nine lives	79
Gnashing of teeth	80
Let the longer games begin	81
Lunar Landing	83
Labyrinth	86
The ZX80 as teacher	89
A degree of conversion	90
Multiplication quiz	91
Squares	93
French vocabulary	94
Life expectancy	95
Useful subroutines	97
Appendix — a few facts about the ZX80	101

# Foreword

This is a curious and interesting book.

While at first sight it appears to be a book which simply tells you how to write and develop games programs for the ZX80 — and gives the listings of over 60 programs in the process — closer inspection reveals that following through the explanation given for each game and computer function will teach you quite a bit about the ZX80, the BASIC language and micro-computers in general.

However, the book is not all games. If you're a parent or teacher, you'll find the section THE ZX80 AS TEACHER an aid in making the most effective use of the ZX80 in the classroom. Tim Hartnell has outlined some quite useful programs for school applications. After the teaching section, there are some useful subroutines, a run-down of the ZX80's functions and a detailed look at the features of the new 8K ROM.

The book is a worthwhile resource to make sure you do, indeed, "make the most of your ZX80". And you'll never feel quite the same about your computer after surviving a round of FRENZY, a game with a most peculiar slot-machine (FOUR FRUITS FEVER) or a crash-landing on the moon (LUNAR LANDER).

SHERIDAN WILLIAMS  
Micro Computer Consultant  
to 'Personal Computer World'  
Magazine.





# Introduction

Your ZX80 came in the post. You unwrapped it and were momentarily surprised (and secretly a little disappointed) at how tiny and — well — uninspiring it looked.

The manual that came with your machine proved a boon. After struggling through it, you actually had a pretty fair idea of how to write a program. If you were like me when I first worked through the manual, you thought the program that rolled a die, and the one that drew two bar charts after a seemingly interminable delay, were pretty good.

But, after reading through the book a few times, a faint, niggling thought came into your mind, which would not go away. "It's a little limited in what it can do," you thought, "and the screen fills up awfully fast. What's the point of spending the rest of my days in front of a flickering TV screen, rolling innumerable dice?"

So, you dashed to a computer shop, or wrote away to one of the companies that advertise in the computer magazines, and spent more of your hard-earned cash on books of programs.

And none of them could be fitted to your machine.

Either the memory in the ZX80 as supplied, with just 1K RAM, was too small to get more than half the program in, or the program demanded numbers that were not integers, or called for statements like DATA which your ZX80 doesn't have. Another gulp, and you began secretly to resent your ZX80. "I should have saved up for a PET," you think. The worst moment comes when one of your friends drops in and after watching you roll a die or two says: "Yes, but what can you do with it?"

That is where this book comes in. If you work through it, with your ZX80 turned on at the appropriate moments, you should learn enough about programming to write your own games programs. If you follow the advice given on "building a library" you'll also have a healthy set of neatly labelled cassettes with a choice of programs to please even your most boring friends.

The book assumes you would like to collect a number of different programs that work; that you want to know how and why they work; and — eventually — that you want to be able to create and develop your own programs. The book explains how the programs were written, developed, and made more complex and/or fun to run, from a simpler "core" program. By following through the programming, feeding the games and other programs into your ZX80 and running them, you should learn a fair amount of BASIC without even trying.

Before you read the book, I suggest you glance through it to get an overall picture of the ground it covers. The more BASIC you know, the further you can read into the book without plugging in your ZX80 and

running the programs. The book was written with the idea that you would input each program when you came to it. This is why, for example, all the HUNT THE HURKLE-type games are not together. You're most unlikely to want to input six different versions of the same program in a row. The order of programs is also dictated by the need to put the simpler ones in first.

My thanks to Colin Hughes and Trevor Sharples for checking the manuscript, of course, any errors are mine. Thank you also to members of the National ZX80 Users Club (44 - 46 Earls Court Road, London, W8 6EJ) who contributed many ideas and comments on the ZX80. And I guess I should also acknowledge Ron Condon who I first met in his incarnation as editor of the computer trade weekly Datalink. He has a lot to answer for. After all, it was Ron who first awakened my addiction to computers.

Tim Hartnell

# New ROM/Old ROM

If you've just bought your ZX80, and you have the standard, unadorned machine, ignore this section and go straight to RANDOM NUMBERS. If you're not sure whether you have a 4K or an 8K ROM, you're almost certain to have a 4K ROM which means the programs as given will run exactly as listed here. This book was written primarily for the 1K ZX80 4K ('old') ROM machine, which is the standard one advertised. Even if you have a ZX80 with the 8K ('new') ROM, or you decide to buy one in due course, the programs in this book will run with just two minor modifications.

One change needed to run the programs in this book on a new ROM ZX80 is to change any division line (such as  $LET J = 7/2$ ) into:  $LET J = INT(7/2)$ . Make these two changes and the programs will run perfectly.

Another feature of the new ROM that could cause problems is that the graphics characters are situated on different keys. The references in this book to thing like "shift S" refer to the graphics which are on the old ROM keys. To change them for the new ROM, use the following list (old position followed by the new):-

shift Q, graphic 5; shift W, graphic 6; shift E, graphic 1; shift R, graphic 2; shift T, graphic D; shift A, graphic A; shift S, graphic T; shift D, graphic 4; shift F, graphic 3; shift G, graphic S.

You'll also find you can get the inverse of the graphics without using a character string (CHR\$(X)) as you need to with the old ROM. When you're more experienced, you will probably want to use some of the new ROM's other features (especially PLOT, DRAW and PAUSE) to enhance the games in the book, but they will all run satisfactorily as they are listed, apart from the changes mentioned above.

# Random Numbers

Turn on your TV, plug in the ugly black power converter, and get the cursor (that's the white K on a black square) down in the left hand corner of the screen. Our first program uses one of the most useful programming aids for games — the random number generator.

Actually, as the manual points out, it is not a true random number generator, but it is close enough for our purposes.

To generate a number between zero and, say, 10, you just input `LET J = RND(10)`. You have to spell out the RND although the LET is just a single key stroke. If you follow up this line with the command `PRINT J`, your ZX80 will display the number (between zero and 10) which it has generated. Notice that when you write a program, each line must be numbered (with numbers between 1 and 9999). The computer executes each line from the lowest number to the highest.

Try this program first:

```
10 PRINT "NUMBER GENERATOR"
20 LET J = RND (10)
30 PRINT J ; " " ; (just leave a couple of spaces between the quote marks)
40 GOTO 20
```

When you run this, you'll find the screen fills up with numbers, between 1 and 10. The program stops (and gives you the error code 5/30) when the screen is full.

There are five things you can learn from this program:

```
10 PRINT "NUMBER GENERATOR"
```

This line starts with a line number (as I mentioned before). You can use any number you like (between 1 and 9999), but you'll find working in multiples of 10 or 20 will give you enough room to add new lines in between others if you need to later. Line numbers sort themselves automatically into the correct order. When you ran the program, the ZX80 acted on the lowest numbered line (in this case 10) and obeyed the instruction PRINT and printed what was between the quote marks. Pretty simple, basic stuff. So from this line you have learned the use of line numbers, and of the command PRINT.

The next line:

```
20 LET J = RND (10)
```

This tells the computer to assign the value generated to the integer variable J (a letter, a number of letters, or a letter followed by a combination of letters and numbers can be an integer variable). Then when, in the next line, you ask the ZX80 to PRINT J, it prints the number which has been assigned to J. Don't worry if you can't understand this, it will become clear in due course.

The next line:

```
30 PRINT J;" "
```

This line tells the computer to print the number which it has assigned to J. The semi-colon after the J, and after the second set of quote marks, ensures that the ZX80 will print the values for J one after another, instead of starting a new line for each one.

Try typing `30 PRINT J`. Using the NEWLINE, put this into the program in place of the original line 30. Run the program, and you'll notice the numbers printed in neat little columns. The comma divides the screen up into four parts, and when it reads a comma, automatically goes to the start of the next quarter of the screen. (The space between the quote marks in the original program ensured there was a space between each number). Try running the program without the space, i.e. with line `30 PRINT J`. You'll find the screen fills up with solid numbers, all running into each other. The semi-colon tells the ZX80 to go on and print the next J, without leaving a space, and without starting a new line.

The fourth line of our original program:

```
40 GOTO 20
```

The computer runs through the program in order, carries out the instructions in each line, and then stops. In this case, when it gets to the end of the program, line 40 tells it to go back to line 20. It does so, then runs through the program again, and again finds the instructions to go back to line 20. It stops only when you run out of screen space.

Now, you've learned that a letter, such as J, can be assigned a numerical value. What is known as a 'string' (a letter followed by a \$ sign, like A\$) can be assigned to a word or words (or any combination of letters, symbols and numbers, but we'll stick to words for the moment).

Clear the RAM with the instruction NEW, and input the following program:

```
10 PRINT "NUMBER          60 PRINT "AND TEN. PRESS  
   GENERATOR"           NEWLINE."  
20 PRINT "WHAT IS YOUR   70 INPUT B$  
   NAME?"               80 LET J = RND (10)  
30 INPUT A$             90 PRINT J;" "  
40 PRINT "OK, ";A$;" I AM 100 GOTO 80  
   GOING TO WORK OUT"  
50 PRINT "SOME RANDOM  
   NUMBERS BETWEEN  
   ONE"
```

Try running this program. You'll find the line 20 asks your name, and then accepts it in line 30, assigning the string A\$ to your name. The ZX80 then uses your name (A\$) in the next line, line 40.

The other thing to note about this program is the line 60 which tells you to press the NEWLINE key to continue. The string in line 70 (B\$) is just a way to tell the computer to stop and wait for NEWLINE to be pressed. This feature is a very useful one in games, and we will be using it time and time again.

You've probably cursed the fact that your ZX80 doesn't 'scroll' the contents of the screen upwards. It just shuts down when the screen is full, and gives you an error code.

The clear screen command (CLS) is very useful here. Looking at the output of the program you've just run, you have probably realised that all the lines spent asking WHAT IS YOUR NAME? and so on are wasted once the instruction has been followed. Add the line: 75 CLS to the program, and run it again.

The CLS, you will find, cleared the screen of everything that preceded it on the program, but didn't (as will become important later) 'forget' what had been assigned to the string, A\$. That is the computer still remembers your name (A\$), so long as you don't wipe the program with NEW or turn off the power, or go back into the 'command mode' (when you can see the whole program listed on the screen, numbers and all).

It is a bit butchery to just run the program until it fills the screen and jams off, giving you an error message. We can let the ZX80 count how many times it goes around the loop (that is, how many times it executes the lines 80, 90 and 100). Add the following:

```
76 LET K = 0
85 LET K = K + 1
95 IF K > 30 THEN STOP
```

Now run it, you should find the program prints 31 numbers between 0 and 10 and then stops. It does this by assigning the value of zero to the number variable in line 76, then adds one to K each time round (so the first time it is, in effect, LET K = 0 + 1, the second time LET K = 1 + 1, the third time LET K = 2 + 1, and so on) until K is bigger than 30, when the computer STOPS. This counting and terminating feature is a very useful one.

IF/THEN: In line 95 (IF K > 30 THEN STOP) we used one of the ZX80's facilities to make a decision, and act on it. This IF/THEN is a very useful command in BASIC. The form of an IF/THEN line is line number, IF something (such as IF Z = 30, or IF A = B) followed by THEN (shift 3) and another command. In ZX80 basic the other command can be anything (such as GOTO, LET S = 2, or STOP).

Delivery of the 8k ROM was delayed late in 1980 pending the development of a printer.

Let's get to our first real program.

# Simple Decision Makers

Input the following:

```
10 PRINT "DECISION          70 INPUT Q$
   MAKER"                   80 LET J = RND (3)
20 PRINT                    90 IF J = 1 THEN
30 PRINT                    PRINT "YES"
40 PRINT "THINK OF A       100 IF J = 2 THEN
   QUESTION AND"           PRINT "NO"
50 PRINT "PRESS NEWLINE   110 IF J = 3 THEN
   FOR A"                 PRINT "MAYBE"
60 PRINT "DECISION ON     120 STOP
   IT"
```

Run this program a few times. You'll see how the comma in lines 10, 90, 100 and 110 sets the words away from the left hand side of the screen, and how lines 20 and 30 (with just the command PRINT, with nothing following it) put a space between the printout of lines 10 and 40.

Now comes the creative bit. To dress up this program, you can do at least three things: (1) Let the ZX80 ask for, and use your name; (2) Use CLS to make the presentation cleaner; and (3) Allow you more than one go before the screen flips back into the command mode. Try and amend the program (by slipping things in between the numbered lines, and/or by changing some lines) until you can do these three things — before reading on. Cover up the following program until you've had a go at altering the original program.

This is just an example of how to modify the program. There are many, many other ways to achieve similar ends.

```
10 PRINT "DECISION          90 IF J = 1 THEN PRINT
   MAKER"                   "YES"
20 PRINT                    100 IF J = 2 THEN PRINT
25 PRINT "WHAT IS YOUR     "NO"
   NAME?"                   110 IF J = 3 THEN PRINT
27 INPUT A$                 "MAYBE"
30 CLS                      120 PRINT
40 PRINT "THINK OF A       130 PRINT "ANOTHER
   QUESTION, ";A$          GO, ";A$;"?"
50 PRINT "PRESS NEWLINE,  140 INPUT B$
   AND I WILL"             150 IF B$ = "YES" THEN
60 PRINT "MAKE A           GOTO 30
   DECISION ON IT."       160 CLS
70 INPUT Q$                170 PRINT "OK, ";A$;"BYE
80 LET J = RND (3)         BYE"
                           180 STOP
```

Note the way the computer is asked to check if B\$ = "YES" (in line 150). If it finds that B\$ is equal to YES, control goes back to line 30. There is no need to put in a line IF B\$ = "NO" THEN GOTO 160 because the ZX80 moves on until it comes to another instruction.

There is another refinement we could add to this program. Try and work out how to make the amendment before you read the program listing. If you decided you wanted the computer to give you two "NO" and two "YES" to every one "MAYBE", how would you amend it? Try to work this out (changes will be needed between lines 80 and 120) before you read on.

If you need five alternatives (two NO, two YES, and one MAYBE), you will need line 80 to generate five random numbers. For two of these the computer must print NO, and so on. To save writing IF J = 1 THEN PRINT "YES", and IF J = 2 THEN PRINT "YES", and IF J = 3 THEN PRINT "NO" and so on, we can use the line IF J < 3 THEN PRINT "YES".

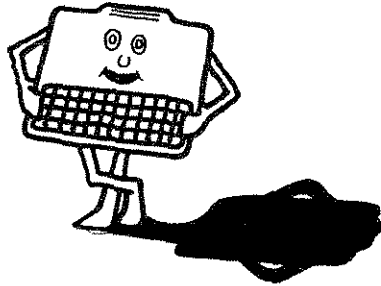
But what about the "NO" answer? If you write, for line 100, IF J < 5 THEN PRINT "NO", the computer will print a YES with a NO underneath if the number 3 or 4 is generated. You can overcome this with an AND statement, as follows:

```
80 LET J = RND*(5)           100 IF J > 2 AND J < 5 THEN
90 IF J < 3 THEN PRINT      PRINT "NO"
   "YES"                   110 IF J = 5 THEN PRINT
                           "MAYBE"
```

A little later on we'll work out a more sophisticated DECISION MAKER program, but for now let's do something a little more interesting — play Russian roulette.

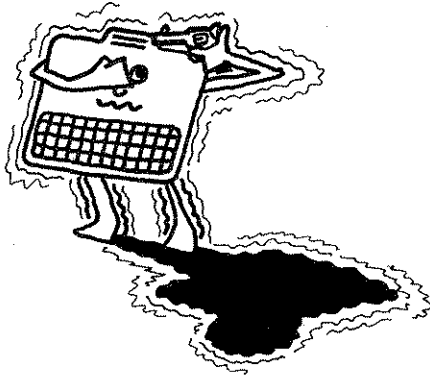


HELLO  
I'M ZEDDY



YOUR ZX80





## Russian Roulette

The principle of the game is simple. You have a pistol with six chambers, only one of which contains a bullet. You spin the chamber, pull the trigger, and...either bang, or click. Already you are thinking "Aha, the random number generator, one in six." The only major decision to make is how many shots you are going to have before ending the game. Input the following program into your ZX80, run it a few times, then come back to this book for a discussion on some of the features of it.

```
10 PRINT,"RUSSIAN
   ROULETTE"
20 PRINT,"(C) HARTNELL
   1980"
30 PRINT
40 LET J = 0
50 PRINT "PRESS NEWLINE
   TO FIRE"
60 INPUT T$
70 CLS
80 LET J = J + 1
90 LET G = RND (6)
100 IF G < 6 THEN PRINT
   "CLICK"
110 IF G = 6 THEN GOTO
   140
120 IF J = 10 THEN GOTO
   160
130 IF J < 10 THEN GOTO 50.
140 PRINT "BANG...";
150 GOTO 140
160 PRINT "YOU HAVE
   SURVIVED"
170 STOP
```

Notice this program uses the GOTO (in lines 110, 120, 130 and 150) command to either give you another 'shot', print BANG... to fill the screen, or to print YOU HAVE SURVIVED. Line 40 sets J = 0 at the beginning, and line 80 adds one to it each time you run through until (unless you've shot yourself) you've been through ten times, that is when J = 10. Lines 120 and 130 check the value of J and either send you back to 50 or advance to 160. Now, add the following line to the program and run it a few times:

```
95 PRINT,G,J
```

This added line displays the number that line 90 has generated and, to the right, the number of the run (i.e. J) that you are on. After you've run it a few times, erase line 95. Before you read on, try to amend the program (just by adding lines between the present ones) to allow the program to give you (a) the option not to play at all; and (b) the chance for subsequent games if you manage to survive.

```
30 PRINT "DO YOU WANT          34 IF A$ = "NO" THEN
   TO PLAY?"                      STOP
32 INPUT A$
```

and further down:

```
162 PRINT "DO YOU WANT        166 IF B$ = "YES" THEN
   ANOTHER GO?"                GOTO 40
164 INPUT B$
```

You may well have modified the program differently, but as long as it works, it doesn't matter exactly how you do it.

Now, the following is a very important point for the development of games' programs. The best way I've found to work out games is to set up the 'core' of the game first (in the case of RUSSIAN ROULETTE, the core would be the first version of the game). Having set up this core, and made sure it works, I then proceed to elaborate the game to make it more fun to play, while making sure I don't overload the ZX80's tiny RAM.

You may have felt that the running of the game in its present form is a little unsatisfactory, and that when line 95 was included (PRINT,G,J) it was a bit more interesting. So add the following line:

```
95 PRINT "THAT WAS SHOT ";J
```

and run the program. That certainly makes it a bit more interesting. Before reading on, I want you to try and write a new version of line 95 which will tell how many shots you've left to reach 10, rather than just telling you the number of the current shot. One way of doing it:

```
95 PRINT 10 - J;"SHOTS TO GO"
```

Now, let us have a second look at line 34. Instead of just stopping when the player decides not to play, why not let the ZX80 respond more definitely, say by printing the word CHICKEN to fill the screen. Try and work out how to do this. For a start, you'll have to change the STOP statement in line 34 into a GOTO command.

One way of doing it:

```
34 IF A$ = "NO" THEN          180 PRINT "CHICKEN...";
   GOTO 180                    181 GOTO 180
```

This version prints a most interesting pattern of the word CHICKEN. We'll use this pattern-effect to produce a new program shortly, but for now, add a few more lines to the program so that it uses the player's name. Once you've done that, and before you read my version below,

try to write in a line so the ZX80 won't print ...SHOTS TO GO on the shot which fills the screen with BANG... This is my final version of the program:

```

10 PRINT "RUSSIAN
   ROULETTE"
20 PRINT "(C) HARTNELL
   1980"
22 PRINT
24 PRINT "WHAT IS YOUR
   NAME?"
26 INPUT A$
27 PRINT
28 PRINT
30 PRINT "DO YOU WANT
   TO PLAY, "A$;"?"
32 INPUT B$
33 CLS
34 IF B$ = "NO" THEN
   GOTO 180
40 LET J = 0
50 PRINT "PRESS NEWLINE
   TO FIRE"
60 INPUT T$
70 CLS
80 LET J = J + 1
90 LET G = RND (6)
95 PRINT A$;"", ";10 -
   J;"SHOTS TO GO"
96 PRINT
97 PRINT
100 IF G < 6 THEN PRINT "
   " CLICK " (with a SHIFT
   A on each side of the
   word CLICK, within the
   quote marks)
105 PRINT
110 IF G = 6 THEN GOTO
   139 (you'll see why 139,
   rather than 140, by
   reading 139)
120 IF J = 10 THEN GOTO
   160
130 IF J < 10 THEN GOTO 50
139 CLS (this clears the screen
   for the big BANG)
140 PRINT "BANG...";
150 GOTO 140
160 PRINT "YOU HAVE
   SURVIVED, ";A$
162 PRINT "DO YOU WANT
   ANOTHER GO?"
164 INPUT C$
165 CLS
166 IF C$ = "YES" THEN
   GOTO 40
180 PRINT "CHICKEN..."; (I
   removed the STOP
   statement here so the
   following line, 190, could
   serve for a NO response
   to 162, as well as for the
   NO response to 34.)
190 GOTO 180

```

Now, before we leave Russian roulette games, we'll look at one final version that introduces you to a new command, and shows a slightly different conversational approach. SAVE your final version of RUSSIAN ROULETTE on cassette, then clear the memory, and input the following program:

```

10 PRINT "RUSSIAN
   ROULETTE (3)"
15 PRINT "(C) HARTNELL
   1980"
20 PRINT
25 PRINT "HI GUN-TOTER.
   YOU HAVE"
30 PRINT "A PISTOL WITH
   ONE BULLET IN IT."

```

```

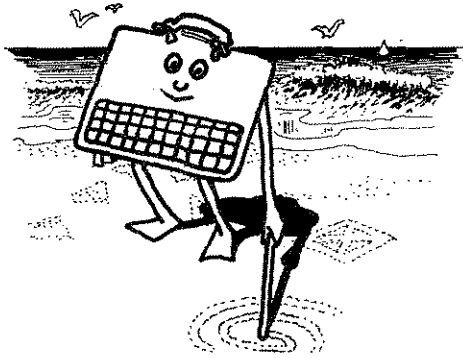
35 PRINT "YOU WILL SPIN          130 IF J < 10 THEN GOTO 70
   THE CHAMBER"                135 CLS
40 PRINT "AND FIRE 10          140 PRINT
   TIMES."                      145 PRINT
45 PRINT "ARE YOU GAME        150 PRINT "YOUVE
   TO PLAY?"                    SURVIVED."
50 INPUT A$                    155 PRINT "IF YOU WANT
55 LET J = 0                    TO RISK DEATH AGAIN"
60 PRINT                       160 PRINT "PRESS G FOR
65 IF A$ = "NO" THEN           GO...OR S FOR STOP"
   GOTO 185                      165 INPUT B$
70 PRINT "PRESS NEWLINE       170 CLS
   TO FIRE"                      180 IF B$ = "G" THEN
75 INPUT T$                    GOTO 55
80 CLS                          185 PRINT "COWARD...";
85 PRINT                       190 GOTO 185
90 PRINT                       195 LET H = RND(2)
95 PRINT                       200 IF H = 1 THEN PRINT "$
100 LET J = J + 1              CLICK $" (use a 'shift S' in
105 LET G = RND(6)             place of the dollar signs)
110 PRINT "SHOT               210 IF H = 2 THEN PRINT
   NUMBER ";J                   "EMPTY CHAMBER"
115 IF G < 6 THEN GOSUB        215 RETURN
   195                          220 PRINT "BANG...";
120 IF G = 6 THEN GOTO        225 GOTO 220
   220
125 IF J = 10 THEN GOTO
   140

```

You'll notice as you play this that an empty chamber (i.e. G 6) gives you either CLICK or EMPTY CHAMBER, which, as you can see, is generated by the lines 195 to 215. The ZX80 leaps to 195 from 115 on the command GOSUB 195.

Lines 195 through to 215 make up a "subroutine". Whenever the computer comes across a GOSUB command, it goes to the line specified, and follows on until it comes to command RETURN. The computer then returns to the line *after* the GOSUB command. The GOSUB/RETURN is a very useful feature. We'll be using it again.

You will recall that when you first worked out how to get the computer to fill the screen with the word CHICKEN that it made quite an attractive pattern. We are going to look now at a slightly more developed form of pattern-making program, which uses the FOR/NEXT loop to limit the number of times the sequence of letters or spaces is repeated. We will be looking at the FOR/NEXT loop in detail a little later.



## Pattern Makers

The core of this program is simple. Feed it into your ZX80 and after pressing RUN, input any six of the graphical symbols, then press NEWLINE.

```
10 INPUT A$
20 FOR J = 1 TO 75
30 PRINT A$;
40 NEXT J
```

You can run this program, which is surprisingly effective, using any combination of letters, numbers and symbols you like. You'll find that one or two spaces, instead of letters, enhance the pattern produced. Try a few more patterns, using spaces, letters like M and W, the \$ sign and the numbers 6 and 9.

Before reading on to see how I have done it, try to write around the core program you have to include the following features: (a) a title; (b) instructions; and (c) the chance to form another pattern without having to go back into the command mode and press RUN again. Cover up my version until you've tried your own.

One way (and there is an infinite number of ways you could have done it) is as follows:

```
1 PRINT "PATTERNS"
2 PRINT "(C) HARTNELL
  1980"
3 PRINT
6 PRINT "PRESS ANY
  COMBINATION OF 6"
7 PRINT "LETTERS,
  SYMBOLS AND
  SPACES..."
8 PRINT
9 PRINT "...AND I WILL
  PRINT A PATTERN"
10 INPUT A$
15 CLS
20 FOR J = 1 TO 75
30 PRINT A$;
40 NEXT J
50 PRINT " "; (24 spaces
  here)
60 PRINT "DO YOU WANT
  ANOTHER GO?"
70 INPUT B$
75 CLS
80 IF B$ = "YES" THEN
  GOTO 6
90 PRINT "OK. SEE YOU
  LATER, ARTIST"
100 STOP
```

This is quite an addictive program. The simplest combination of symbols (like three of shift W, two of shift Q and a space) produce almost three-dimensional patterns. If you set this program up for one of your friends to try, be ready to wait a long, long time before you next have a go.

---

## Building a Library

I hope you've been SAVEing the programs you create as you go along. There is little point in having to input the whole program by hand every time you want to run it.

Resist the temptation to put all your programs on one cassette, one program after another. The frustration you'll experience searching through the tape (even if you've identified each program with a voice label) to find a particular program is just not worth the trouble. Go to your computer shop, or buy by mail from one of the many companies that advertise in the personal computing magazines, and get a set of C-12 cassettes. Put just one program on each cassette, with two copies of each program on each side, just in case something happens to one of the copies and you find it difficult, or impossible, to LOAD.

Write the name of the program on the cassette, and on the cardboard insert. You'll find this makes it very easy to find the program you want, and as your library builds, it will give you quite a feeling of accomplishment to see all those programs ranked side by side. It will impress your friends as well, who will believe after visiting you, and playing with your ZX80, that you're some kind of natural computer genius (which you probably are).

---

## Review

Let's review the ground we've covered so far;

- The use of the random number generator `LET J = RND(X)`
- The output statement `PRINT`
- The assignment statement `LET`
- The use of `,` and `;` in `PRINT` statements



- The control statement IF...THEN...
- The use of strings (A\$), including the use of a string to stop a program until NEWLINE is pressed
- The statement CLS
- The symbols < and >
- The statement AND
- The control statement STOP
- The control statement FOR...TO/NEXT
- The form and use of a "counter" to limit the number of times part of a program is run through
- How to develop a game program from the core of it, to make it more elaborate and interesting
- Subroutines (with the commands GOSUB and RETURN)
- How the string, FOR...TO.../NEXT, and PRINT can be used to generate patterns.

We've come quite a long way in a short time. Make sure you understand all the preceding material before going on.

---

## For/Next

You'll remember when we wrote the first PATTERN-MAKER program, we used (in line 20) FOR J = 1 TO 75, and (in line 40) NEXT J. We will now have a look at the use of FOR/NEXT loops in some detail.

First, input the following program:

```

10 LET J = 0
20 PRINT "J = ";J
30 IF J = 10 THEN STOP
40 LET J = J + 1
50 GOTO 20

```

Run it a few times, then clear the RAM and input the following program, which uses the FOR/NEXT loop:

```

10 FOR J = 1 TO 10
20 PRINT "J = ";J
30 NEXT J

```

You can see that running this program produces exactly the same result as running the preceding one, although this second program is only three lines long, two shorter than the first program. It is not always possible or desirable to use a FOR/NEXT loop as a "counter", but because it uses less memory than the LET J = 0/LET J = J + 1 approach, it should always be investigated.

When you run a program with a FOR/NEXT loop, the computer goes from line to line after the FOR statement until it finds the NEXT command, when it reverts to the FOR line. You can place one (or more) FOR/NEXT loops inside another. Add the following lines to the current program:

```
22  FOR K = 1 TO 2
25  PRINT K
27  NEXT K
```

Run this. It is important to make sure that each loop nests neatly within each other loop. That is, if you placed the NEXT K command **after** the NEXT J, so the loops cross, you get a rather unsatisfactory result. Try it, by renumbering line 27 as line 35 (making sure you delete the "old" line 27).

Finally, here is a remarkably effective double FOR/NEXT loop program:

```
5   PRINT "YOUR NAME IN      30  FOR S = 1 TO 99
    LIGHTS"                    40  PRINT "(shift S)";
10  FOR J = 1 TO 6             50  NEXT S
20  PRINT "(put your first    60  NEXT J
    name here)";
```

This is a good program to run for your more egocentric friends.



## Flipping a Coin

The computer can be used to simulate a huge variety of things, from the growth of plants to the effects of bumps on the road on motorcycle suspension. We are going to look at a more homely example of simulation: flipping a coin.

Input the following program, and run it:

```
10 PRINT "COIN FLIP — 1"      40 IF J = 2 THEN PRINT
20 LET J = RND(2)              "TAILS"
30 IF J = 1 THEN PRINT        50 STOP
   "HEADS"
```

Once you've run this a few times, amend the program as follows:

```
10 PRINT , "COIN FLIP — 2"    50 PRINT "TO FLIP AGAIN
20 LET J = RND(2)              PRESS F"
30 IF J = 1 THEN PRINT        60 INPUT A$
   "HEADS"                    70 IF A$ = "F" THEN GOTO
40 IF J = 2 THEN PRINT        20
   "TAILS"                    80 STOP
```

You can run this until the screen is full. Although there will probably be an imbalance in the numbers of HEADS and TAILS, the longer you run the program, the more the ratio of each should approach 1:1. If we knew how many times we had flipped the coin, it would make it easier to work out how many HEADS and how many TAILS the program had printed. It is fairly simple to amend the program to do this. Try it yourself, before looking at my version.

```
10 PRINT "COIN FLIP — 3"      70 LET K = K + 1
20 LET K = 1                   80 PRINT "TO FLIP AGAIN
30 LET J = RND(2)              PRESS F"
40 PRINT "FLIP NUMBER          90 INPUT A$
   ";K;" IS ";                 100 IF A$ = "F" THEN GOTO
50 IF J = 1 THEN PRINT         30
   "HEADS"                    110 STOP
60 IF J = 2 THEN PRINT
   "TAILS"
```

Once you've run this a few times, and added up the HEADS and TAILS, and then worked out what the ratio of HEADS to TAILS is, you will probably have realised the ZX80 can do all the work — flip the coin, count the heads and tails, and then manipulate this result as you choose. Let's try a more complex program:

```
10 PRINT "COIN FLIP — 4"      90 IF J = 1 THEN LET H =
20 LET H = 0                   H + 1
30 LET T = 0                   100 IF J = 2 THEN LET T =
40 LET K = 1                    T + 1
50 LET J = RND(2)              110 PRINT "OUT OF
60 PRINT "FLIP                 ";K;" FLIPS YOU HAVE"
   NUMBER ";K;" IS ";         120 PRINT H; "HEADS
70 IF J = 1 THEN PRINT         AND ";T;" TAILS"
   "HEADS"                    130 LET K = K + 1
80 IF J = 2 THEN PRINT         140 PRINT "PRESS F TO FLIP
   "TAILS"                     AGAIN"
```

```

150 INPUT A$
160 CLS
170 IF A$ = "F" THEN GOTO
50
180 PRINT "THANKS FOR
PLAYING. BYE"
190 STOP

```

You will find that adding PRINT statements for lines 105, 106 and 107 will make the program, when running, easier to read. Now, we could use a FOR/NEXT loop to tell the ZX80 in advance how many times we wanted to flip the coin, and then we could leave it to do the work. For practice, I'd like you to produce such a program, which also asks the player at the beginning how many times he or she wishes to flip the coin. I am not going to give you a sample version of this because I think by now you should be able to write such a program easily. What I am going to give you though is another version of the program, that gives approximate percentage breakdowns of heads and tails. They are approximate because the ZX80, as you know, rounds numbers to the nearest whole integer. First write your program using the FOR/NEXT loop, SAVE it, and then have a look at my final COIN FLIP program.

```

10 PRINT "FLIP-A-COIN"
15 PRINT ":(C) HARTNELL
1980"
20 LET H = 0
30 LET T = 0
40 LET K = 1
50 LET J = RND(2)
55 PRINT
57 PRINT
60 PRINT "FLIP
NUMBER ";K;" IS ";
70 IF J = 1 THEN PRINT
"HEADS"
80 IF J = 2 THEN PRINT
"TAILS"
90 IF J = 1 THEN LET H =
H + 1
100 IF J = 2 THEN LET T =
T + 1
105 PRINT
106 PRINT
107 PRINT
110 IF K = 1 THEN GOTO
150
111 LET A = H*100/K
112 LET B = T*100/K
113 IF A + B < 100 THEN
LET A = A + 1
120 PRINT "OUT OF ";
K;" FLIPS YOU HAVE"
130 PRINT "APPROX. ";
B;" PER CENT TAILS"
140 PRINT "AND ";A;" PER
CENT HEADS"
142 PRINT
147 PRINT
150 PRINT "PRESS F TO FLIP
AGAIN"
155 LET K = K + 1
160 INPUT A$
170 CLS
175 IF A$ = "F" THEN GOTO
50
180 IF K = 2 THEN GOTO
195
181 PRINT
182 PRINT
185 PRINT "YOU FLIPPED
THE COIN ";K-1;
" TIMES:—"
186 PRINT
187 PRINT,"HAD ";A;" PER
CENT HEADS"
188 PRINT

```

189 PRINT ,“AND ”;B;“ PER	193 PRINT
CENT TAILS”	194 PRINT
190 PRINT	195 PRINT “THANKS FOR
191 PRINT	PLAYING. BYE”
192 PRINT	200 STOP

There are a few points you can learn from examining this program. Up to line 100, there is nothing new. Line 110 (IF K = 1 THEN GOTO 150) bypasses all the percentage computation, because there is no need for it if you have only flipped once. Lines 111 and 112 multiply the numbers of heads and tails by 100, then divide by the total number of flips. Because the ZX80 rounds off number when dividing, it is best to get the number as high as possible before rounding off. If line 111 was changed to: LET A = (H/K)\*100 it would produce far less accurate results. Try it and see what happens.

Line 113 adds the two “percentages” together, and if they equal 99 (which will happen pretty often) 1 is added to the HEADS’ percentage, just to make the two percentages add up to 100. Line 180 (IF K = 2 THEN GOTO 195) is used if the player decides to stop after two flips. The information given by lines 185 to 189 is only of interest if the coin has been tossed three or more times. Line 195 is just a friendly way of ending. Many times, when you write a program, you’ll use nearly all the RAM just for the program, so you will not have room for niceties like goodbyes. However, whenever you have room in the memory, it is a good idea to make the program more “user-friendly” and make the ZX80 “conversation” as natural as possible.

---

## Decision Maker — Mark 11

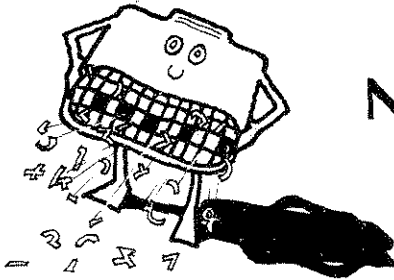
One of the first programs we developed was a simple decision maker. Let us look now at the more complex program to make decisions. This program will show how strings can be used by the ZX80 to form complete sentences, and will also demonstrate a memory-saving way of using the random number generator. Input the following program:

10 PRINT ,“DECISION	20 INPUT A\$
MAKER — 2”	22 CLS
11 PRINT ,“(C) HARTNELL	25 PRINT A\$;“, I AM HERE
1980”	TO HELP YOU”
12 PRINT	26 PRINT ,“REACH A
15 PRINT “HI, WHAT IS	DECISION”
YOUR NAME?”	27 PRINT

```

28 PRINT "JUST COMPLETE THE FOLLOWING"
29 PRINT "SENTENCE (WITHOUT USING THE"
30 PRINT "WORDS I, ME OR MINE)"
31 PRINT "COMPLETE THIS: SHOULD I..."
32 INPUT B$
33 CLS
35 LET J = RND (4)
36 GOTO 100*J (this is the memory-saving trick)
100 PRINT "IF I WERE YOU, ";A$; ", I WOULD NOT" B$
105 GOTO 600
200 PRINT "SURE, ";A$; ", ";B$
205 GOTO 600
300 PRINT "YOU MUST BE CRAZY, ";A$; ", TO THINK YOU MIGHT ";B$
305 GOTO 600
400 PRINT "GEE, ";A$; ", ITS A TOSSUP WHETHER YOU ";B$; " OR NOT"
405 GOTO 600
600 PRINT
610 PRINT
620 PRINT "ANOTHER DECISION, ";A$; "?"
630 INPUT C$
640 CLS
650 IF C$ = "YES" THEN GOTO 27
660 PRINT "THANKS FOR LETTING ME"
670 PRINT "HELP YOU, ";A$
680 STOP

```



## Number Crunching

The programs we've worked on so far have ignored the great, and fundamental ability the ZX80 has to work with numbers.

Input this program into your computer:

```

10 PRINT "MATHS DEMONSTRATION"
15 PRINT "GIVE ME A NUMBER"
20 INPUT A
25 PRINT "A SECOND NUMBER, PLEASE"
30 INPUT B

```

```

35 PRINT "AND ONE           50 PRINT A;"MULTIPLIED
   MORE"                   BY ";B;" DIVIDED BY"
40 INPUT C                  55 PRINT C;" EQUALS ";K
45 LET K = A*B/C

```

Run this through a few times, with different values for A, B and C. Notice how, in lines 50 and 55, when using PRINT, the ZX80 substitutes the values assigned to A, B and C. It prints the number rather than the letter.

Now, change lines 45, 50 and 55 as follows:

```

45 LET K = (A +           50 PRINT K
   C)/(B+C-A)           55 STOP

```

Run this a few times. Use numbers less than 100 to remove the risk of K being greater than 32767 or less than -32768 (which will stop the program, giving you error code 6).

We can use a program of this type to do practically any maths problem, no matter how complex, so long as the numbers involved do not exceed the limits given above, and decimal places are not important (the ZX80 works only on whole numbers, integers) or, if they are, a subroutine is used to generate them. We'll use the basic mathematical ability of the ZX80 to produce a rather interesting demonstration program you can run next time you have a couple of friends around.

```

10 PRINT,"NUMBER           125 INPUT D$
   JUGGLE"                 130 CLS
30 PRINT "NAME OF         135 PRINT A$,"SUBTRACT
   PLAYER?"                SEVEN"
40 INPUT A$               140 PRINT "AND TYPE IN
65 CLS                    THE RESULT"
70 PRINT A$;" THINK OF A  145 INPUT A
   NUMBER AND"             150 CLS
75 PRINT ,"DOUBLE IT"     175 LET K = (A + 2)/2
95 INPUT C$               185 PRINT "YOUR
100 CLS                   NUMBER, ";
120 PRINT A$;" ADD FIVE"  A$," WAS ";K

```

Now that did not, perhaps, seem very startling, although it is fairly impressive the first time you try it on someone. Note how the strings C\$ and D\$ were used to stop the program until you pressed NEWLINE, how CLS was used to clear the screen after each instruction, and how the final line (185) used the name string (A\$) and the assigned variable (K) to print out your answer.

Before you read on, work out a way of letting the program give you the chance to have another go. Type your version in at the end of the program, and see if it works. One way would be as follows:

```

200 PRINT "ANOTHER GO?"   215 IF Z$ = "YES" THEN
205 INPUT Z$              GOTO 65
                          230 STOP

```

Once you've run through this program a few more times, you're probably pretty bored with it. Remember that a computer can, within the limits of its memory, store a great deal more information than just one simple mathematical manipulation. This game program becomes a great deal more interesting when you interweave a second set of instructions into the first, so a second player can "think of a number" and so on, at the same time as the first person is doing so. I'll list the whole program. Apart from a word to be added to line 30, and some possible differences in the lines 200 and beyond (the lines that offered you a second run), you should be able to add the lines from the following program in between those you already have in the ZX80's memory.

```

10 PRINT "NUMBER
   JUGGLE"
20 PRINT "(C) HARTNELL
   1980"
25 PRINT
30 PRINT "NAME OF FIRST
   PLAYER?"
40 INPUT A$
45 PRINT
50 PRINT "NAME OF
   SECOND PLAYER?"
60 INPUT B$
65 CLS
70 PRINT A$;" , THINK OF A
   NUMBER AND"
75 PRINT "DOUBLE IT"
80 PRINT
85 PRINT B$;" , THINK OF A
   NUMBER AS WELL"
90 PRINT "AND ADD SIX
   TO IT"
95 INPUT C$
100 CLS
105 PRINT B$;" , I WANT YOU
   TO MULTIPLY YOUR"
110 PRINT "NUMBER BY
   THREE"
115 PRINT
120 PRINT A$;" , ADD FIVE"
125 INPUT D$
130 CLS
135 PRINT A$;" , SUBTRACT
   SEVEN"
140 PRINT "AND TYPE IN
   THE RESULT"
145 INPUT A
150 CLS
155 PRINT B$;" , I WANT
   YOUR TO SUBTRACT"
160 PRINT "FOUR THEN
   TYPE IN YOUR RESULT"
165 INPUT B
170 CLS
175 LET K = (A + 2)/2
180 LET L = ((B + 4)/3) - 6
185 PRINT "YOUR
   NUMBER, ";A$;
   " , WAS ";K
190 PRINT
195 PRINT "AND YOURS, ";
   B$;" , WAS ";L
200 PRINT "ANOTHER GO?"
205 INPUT Z$
210 CLS
215 IF Z$ = "YES" THEN
   GOTO 65
220 PRINT "OK, ";A$" AND ";
   B$;" , THANKS"
225 PRINT "FOR PLAYING.
   BYE BYE."
230 STOP

```

Try this out with a couple of friends, and watch their reaction.

I want to introduce you now to a little trick which can add a lot to



games programs (even though it will probably horrify computer purists). The ZX80 works very, very quickly. So quickly, in fact, that its responses appear to be instantaneous. Although this is fine for "serious" use of a computer, it detracts in some measure from games. You'll find that games are more interesting if the computer appears to be "thinking" between moves, rather than responding as soon as you hit NEWLINE. So, we can introduce a subroutine which will slow things down a bit.

Add the following:

```
240 FOR J = 1 TO 200          255 NEXT J
250 CLS                      260 RETURN
```

Now, go through the program, and change all the lines which have the instruction CLS to GOSUB 240. Now, run the program again. Notice how much more effective this seems, with just the right delay for the computer to "think and reach a decision". You could have other numbers in line 240. Try it as FOR J = 1 TO 500. You'll find this delay is far too long. So long that you'll get irritated. Using 300 also produces too long a delay. To me, 200 seems about right. You can use the subroutine delay in any games you like. As an exercise, try adding it to an earlier program, like RUSSIAN ROULETTE, and seeing if this adds to the playing of the game.

The general form of a "game delay":

```
Line Number   FOR X = n1 TO n2
Line Number   CLS
Line Number   NEXT X
Line Number   RETURN (if the delay is a subroutine).
```

As a general rule, you can insert a delay subroutine whenever you would normally use the CLS instruction. When you're feeding in a program that is likely to come pretty close to using all the ZX80's RAM, do not input the delay subroutine until you've got the rest of the program in. Use a CLS line instead, and if and when you find there is enough memory left for a delay subroutine, put that in.

---

## Writing a Program — Some General Thoughts

Most books on programming suggest you start by drawing up a "flowchart" — a pretty combination of circles, diamonds and slanting

rectangles — which sets out the path and operations the computer will follow to execute a program. In theory, this is fine. But in practice, especially when working with a relatively small memory such as in the ZX80, the time and trouble involved is probably not worth it.

It is, however, essential to know exactly what you want the computer to do before you start creating a new program, even if you're not quite sure how you are going to get the ZX80 to carry out the task.

Sometimes a rough sort of flowchart — just writing down the main steps the ZX80 will take and then linking these by lines and loops — will help to clarify your thinking. This is also a good way of spotting potential problems, like the danger of setting up infinite loops, or of not specifying the nature of the computer's decisions exactly.

For what it's worth, I work as follows:

Having worked out the general idea for a game (like "player thinks of an animal, computer tries to guess which animal") I then just think about the idea for a while. I might not write anything down for even a day or more. Usually, this "thinking time" allows me to work out roughly how the core of the program is going to look. Next, on paper, I write down this core, starting with line number 100. This ensures there is plenty of room at the start of the program to add a title and instructions, define constants, set up arrays and the like.

The next stage is to feed the rough program into the ZX80 and see if it does what it is meant to do. Often a far more elegant method than the first rough version is found at the keyboard, but this would probably not have been discovered if the written version was not tried first.

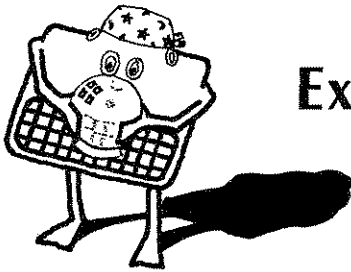
Occasionally, if I get a clear idea for a program and I cannot get to the ZX80, I will write out the entire program by hand in advance. Sometimes (like the race game LONDON TO GLASGOW), a program works almost perfectly the first time. Other programs (such as MASTERMIND) turned out to be near disasters, so the final version bears little more resemblance than title and intention to the original, hand-written version. If you're working on a fairly simple game, or are adapting from a magazine or book, it is just as well to work directly on the ZX80, but keep a notebook handy to record things like the fact that, for example, string A\$ is for the player's name.

If you're writing a program direct on the keyboard, and you want to add a subroutine which will eventually be at the end of the rest of the program, give it a very big number (like 5000). It can easily be renumbered afterwards (and the GOSUB command changed). This is simpler, and more elegant, as well as using less memory, than having to use a GOTO to jump over a subroutine which has been given too low a line number.

If there is a phrase that you'll need the computer to print at several parts of the program, such as "THE SCORE AT THIS POINT IS", assign

a string to this phrase as soon as you realise you're going to have to place the line in several different places in the program. A lot less memory is involved in the line PRINT B\$ appearing six times, than in six of PRINT "THE SCORE AT THIS POINT IS". If the information to be included several times is longer than one line, a subroutine is probably the most memory-efficient device to use. Later on this book looks at a program which makes use of a large number of strings for phrases which are used over and over again (LONDON TO GLASGOW) and when you come to the game, you'll see how efficient a process this can be.

---



## Extra-Sensory Perception

The next program we will look at uses the ZX80's "greater than" and "less than" facilities to compare two numbers, and then make a decision on the basis of whether one number is less than, or greater than, the other. This program also makes use of many of the things we've covered so far, including the IF...THEN, the counter and the random number generator.

Input the following program:

```
10 PRINT "GUESS MY          75 IF J = A THEN GOTO
   NUMBER"                    105
45 PRINT "WHAT NUMBER,      80 IF A < J THEN PRINT
   BETWEEN 1 AND 100,        "HIGHER"
   AM I THINKING OF?"      85 IF A > J THEN PRINT
50 LET J = RND (99)          "LOWER"
55 LET S = 0                 90 IF S < 5 THEN GOTO 60
60 LET S = S + 1            95 IF S = 5 THEN GOTO
65 INPUT A                  130
70 PRINT „A
```

```

105 PRINT "YOU ARE RIGHT.      145 INPUT H$
    I WAS THINKING OF ";J      150 CLS
110 GOTO 140                    155 IF H$ = "YES" THEN
130 PRINT "TIME IS UP. I      GOTO 25
    WAS THINKING OF ";J      160 STOP
140 PRINT "DO YOU WANT
    ANOTHER GO?"

```

Notice how the core of the program (lines 50 to 95) make the decision on what the ZX80 is to do next.

This program, in its present form, has only used up a part of the ZX80's memory. It is possible to dress the program up a bit, by giving the computer your name, and — as you'll see — giving it the ability to award the player a random payoff if the number is guessed correctly. Try this new form. Some of the original program remains, but certain lines have to be replaced, and others added.

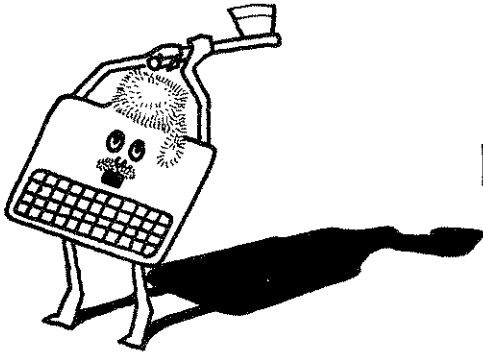
```

10 PRINT "GUESS MY          100 LET K = RND (3)
    NUMBER"                  105 PRINT "YES, I WAS
15 PRINT "(C) HARTNELL      THINKING OF ";J". I
    1980"                    HEREBY AWARD YOU
20 PRINT                    THE TITLE OF";
25 PRINT                    110 IF K = 1 THEN PRINT
30 PRINT "HI, WHAT IS      " SMART ALEC OF THE
    YOUR NAME?"              YEAR, ";A$
35 INPUT A$                  115 IF K = 2 THEN PRINT
40 CLS                        " MASTER OF ESP, ";A$
45 PRINT "WHAT NUMBER,    120 IF K = 3 THEN PRINT
    BETWEEN 1 AND 100,
    AM I THINKING
    OF, ";A$;"?"
50 LET J = RND (99)         125 GOTO 140
55 LET S = 0                130 PRINT "YOU ARE
60 LET S = S + 1            STUPID, ";A$;" I WAS"
65 INPUT A                  135 PRINT "THINKING OF
70 PRINT "A                ";J
75 IF J = A THEN GOTO      140 PRINT
    100 (note change from    145 PRINT "DO YOU WANT
    105)                     ANOTHER GO?"
80 IF A < J THEN PRINT     150 INPUT N$
    "HIGHER"                 155 CLS
85 IF A > J THEN PRINT     160 IF N$ = "YES" THEN
    "LOWER"                   GOTO 45
90 IF S < 5 THEN GOTO 60   165 PRINT "THANKS FOR
95 IF S = 5 THEN GOTO     PLAYING, ";A$
    130                       170 STOP

```

When you run this program, you'll find the lines 105 to 120 wrap

around to the following line when printing. This is not a good feature. As an exercise, try and rewrite this section so the lines print separately, so you do not run the risk of having a word cut in half, with one half on one line, and the next on the other. (Hint: you'll need two more GOTO 140 instructions).



## Kill the Chopper

It is possible to use a very similar core to produce what appears to be a completely different program. Input the following:

```

10 PRINT "CHOPPER — (C)      95 PRINT,“(shift Q, space,
   HARTNELL 1980”           shift G, space, shift Q)”
20 PRINT                    100 PRINT,“(shift Q, three
30 PRINT                    spaces, shift Q)”
40 PRINT "A DEADLY          105 PRINT,“(shift Q five
   CHOPPER IS HIDING”      times)”
45 PRINT "BEHIND 1 OF 16    110 PRINT,“(space, shift A,
   TREES”                  space, shift A)”
50 PRINT                    115 PRINT
55 PRINT                    120 PRINT
60 PRINT "YOU HAVE ONLY    125 PRINT "GUESS THE
   6 SHOTS. OK?”           NUMBER OF THE TREE”
65 INPUT A$                 130 PRINT
70 CLS                      135 PRINT "YOU WANT TO
75 LET G = 0                SHOOT AT”
80 LET J = RND (16)         140 LET G = G + 1
85 PRINT " " 0 0”          145 INPUT M
90 PRINT,“(shift A twice,   150 CLS
   shift G once, shift A    155 IF M = J THEN GOTO
   twice)”                 220

```

```

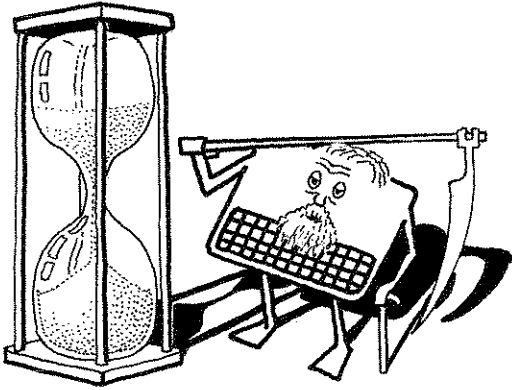
160 IF G = 6 THEN GOTO 195
165 PRINT "YOU MISSED, ";
    6 - G;" TO GO"
170 PRINT
175 PRINT
180 IF M < J THEN PRINT
    "TRY TO THE RIGHT"
185 IF M > J THEN PRINT
    "TRY TO THE LEFT"
190 IF G < 6 THEN GOTO 85
195 PRINT
    "AAAAAAAAAARGHH....."
200 PRINT
205 PRINT
210 PRINT "HE WAS BEHIND
    TREE NUMBER ";J
215 STOP
220 PRINT ,"GOT HIM"
225 PRINT "  (after the two
    zeros, space, two shift A,
    space, shift W, space,
    shift Q)"
230 PRINT"(three spaces, two
    shift A, seven spaces, shift
    D, space, shift D)"
235 PRINT
240 PRINT
245 PRINT
250 PRINT "PRESS 2 FOR
    ANOTHER GO"
255 PRINT ,"OR 4 TO
    CHICKEN OUT"
260 INPUT T
265 IF T = 2 THEN GOTO 70
270 CLS
275 PRINT ,"CHICKEN"
280 STOP

```

The idiotic appearance of the "chopper" was chosen at random. (It was named after my dog.) Make up your own name and appearance for your beastie (making sure its "dead" version looks vaguely like a deflated version of the live creature). You can also alter the value of J (the number of trees) and/or G (the number of shots you get before being killed).

This framework can also be used for, say, vultures hiding in nests, poisonous snakes in holes, or even unfriendly aliens in space, lurking behind asteroids. The "conversation" parts of the program can also be altered to what ever form you choose.

A little gimmick you can add to this (and any other program you like, assuming you have enough money for it) is to get the program to ask the player's name. You work out a subroutine which ensures the computer will only play with you (or with people fortunate enough the share your name). If any other name is fed in when the ZX80 asks who is playing, it goes straight to STOP. As an exercise, work out such a subroutine and add it to your version of CHOPPER.



## Days of Our Lives

Let's go back now to number-crunching, and have a look at a program which gives interesting (and sometimes scary or absurd) information on how long the player has to live. A little later on in this book is a program designed to give a proper look at life expectancy, but that program is a little more complicated. For now, this program — DAYS OF OUR LIVES — produces a pretty good demonstration of your ZX80, and the less mathematical of your friends will be convinced, yet again, that you are some kind of genius.

What the program does, in essence, is work out approximately how old the player is in days, and compares this with average life expectancy (67½ years for males, 74½ for females). It also assumes the player sleeps eight hours a night. Input the following program and run through it a few times. Then read the discussion that follows the program listing.

```

10 PRINT "DAYS OF OUR
   LIVES"
15 PRINT "(C) HARTNELL
   1980"
20 PRINT
25 PRINT "HI, WHAT IS
   YOUR NAME?"
30 INPUT A$
35 CLS
40 PRINT "OK, ";A$," LETS
   WORK OUT SOME"
45 PRINT "INTERESTING
   THINGS ABOUT YOU"
50 INPUT B$
55 CLS
60 PRINT "HOW OLD ARE
   YOU IN YEARS?"
65 INPUT A
70 PRINT "AND MONTHS?"
75 INPUT B
80 LET X = 365*A + 30*B
85 CLS
90 PRINT "YOU ARE ";X;
   " DAYS OLD, "A$
95 INPUT C$
100 PRINT "ARE YOU
   FEMALE?"
105 INPUT D$
110 CLS
115 IF D$ = "YES" THEN
   GOTO 130
120 LET Y = 24637 - X
125 GOTO 135
130 LET Y = 27192 - X

```

135	PRINT A\$;" , YOU HAVE ";Y;" DAYS TO LIVE — BASED ON STATISTICS"	180	INPUT G\$
140	INPUT E\$	185	CLS
145	CLS	190	PRINT "IS THERE ANYONE ELSE THERE WHO"
150	PRINT "YOU HAVE SLEPT FOR ABOUT";X/3	195	PRINT "WOULD LIKE A GO, ";A\$;"?"
155	PRINT "DAYS SO FAR"	200	INPUT H\$
160	INPUT F\$	210	IF H\$ = "YES" THEN GOTO 25
165	CLS	215	PRINT "OK BYE, ";A\$
170	PRINT "AND WILL SLEEP FOR ABOUT";Y/21	220	STOP
175	PRINT "WEEKS OF YOUR REMAINING LIFE"		

If you were thinking about what you were feeding into the ZX80, you probably have a pretty good idea of what each line and command was for, but, if not, look at it on your screen and follow through this next section, line by line with your program.

First of all, of course, the program got your name (A\$), then stopped at line 50 waiting for you to press NEWLINE in order for the program to continue (and clear the screen at line 55). This stopping of the program, as you learned earlier, does not do anything, but makes the program much more interesting to run than supplying all the information at once.

The program obtained your age in years (A) and months (B), then worked out what this was in days (line 80, which assumed there were 30 days in a month). Next the ZX80, in line 90, told you what this figure was (X). The question in line 100 — ARE YOU FEMALE? — is needed because males and females have different life expectancies. If D\$ (the answer to the line 100 question) is YES, the computer goes to line 130, where it subtracts your age in days from the life expectancy (74½ years, converted to days) for females born after 1961 (it is only half a year less for females born 1951 — 1960). If the answer is not YES, the computer moves to the next line (120) and subtracts your age in days from 24,637 days, the life expectancy for males born after 1960 (again, it is half a year less for males born 1951 — 1960).

Line 135 prints how many days after are left. Lines 150 and 155 give the number of days slept so far (your age in days, divided by three), assuming you sleep eight hours a night. Then the ZX80 takes your remaining expected days (Y), divides this figure by three (to get the days spent in sleep) and then divides this by seven to convert this figure to weeks. Actually, as you can see, the program simply divides by 21 (i.e. 3 x 7).

Lines 190 to 210 ask if someone else wants a go, and if so, goes right back to line 25, where the string A\$ is ready to be assigned to the new



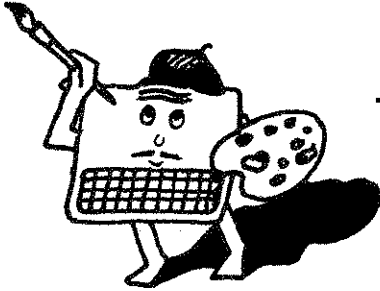
name. If there is no-one else to play the ZX80 proceeds to the end of the program.

If you feel like experimenting, there is no reason to stick with the program as I've written it. Let it ask, for example, if the player is male (changes lines 100 to 130). Instead of the somewhat unimaginative farewell (line 215) you might wish to put in some mournful line like: WELL, A\$, I GUESS YOU'D BETTER RUN OFF AND MAKE THE MOST OF THE Y/30 MONTHS YOU HAVE LEFT TO LIVE.

You could also, of course, add a delay subroutine where CLS appears in the program (lines 35, 55, 85, 110, 145, 165, 185 and 205). By all means experiment with the program, and put your own personal stamp on it. By doing this, you'll gain far more knowledge about programming than you will just by feeding in the programs in this book, line for line, and then leaving them this way.

If you'd like to work out a whole new program based on the core of this program, feed the following program into your ZX80, and then work around it as you choose.

```
10 PRINT "HOW OLD ARE          140 PRINT "HAVE
   YOU IN YEARS?"             ABOUT";Y;"DAYS TO
20 INPUT A                     LIVE,"
30 PRINT , "AND               150 PRINT "BASED ON
   MONTHS?"                   STATISTICS"
40 INPUT B                     160 PRINT "YOU HAVE
50 LET X = 365*A + 30*B       SLEPT FOR ABOUT";X/3
60 PRINT "ARE YOU            170 PRINT "DAYS SO FAR,
   FEMALE?"                   AND WILL SLEEP"
70 INPUT A$                   180 PRINT "FOR
80 CLS                         ABOUT";Y/21;"WEEKS
90 IF A$ = "YES" THEN         OF"
   GOTO 120                   190 PRINT "YOUR
100 LET Y = 24637 - X         REMAINING LIFE"
110 GOTO 130                  200 STOP
120 LET Y = 27192 - X
130 PRINT "YOU
   ARE";X;"DAYS OLD
   AND"
```



## Toying with the Muse

To make a change from working with numbers, here is a program that — after a fashion — writes poetry. Most of the poems it comes up with are pretty awful, but from time to time you'll get a real gem. And even the worst efforts of the ZX80 are not as bad as some of the worst modern poetry published today.

The heart of this program is our old friend the random number generator. It also makes use of the memory-saving device of linking the number generated to a subroutine. Input the program as it appears below, and run it *several* times. Then read the notes that follow the listing. (Note that there is a space just inside the quote marks in the "seed-phrase" lines following the command PRINT.)

```

1 PRINT , "SEASIDE          70 PRINT " SEAGULLS"
   POETRY"                75 RETURN
2 PRINT , "(C) HARTNELL    80 PRINT " WAVES";
  1980"                   85 RETURN
3 PRINT                   90 PRINT " ROCKS";
4 PRINT                   95 RETURN
5 PRINT "PRESS NEWLINE   100 PRINT " SEAWEED";
  TO WRITE A POEM"        105 RETURN
6 INPUT A$               110 PRINT " HOT";
7 CLS                   115 RETURN
10 LET J = RND (21)      120 PRINT " GRITTY";
15 IF J = 1 THEN PRINT   125 RETURN
   "...";               130 PRINT "BEATING,
20 IF J > 1 AND J < 5 THEN BEATING";
   GOSUB 1000           135 RETURN
30 IF J > 4 THEN GOSUB   140 PRINT " HARSH";
   J*10                145 RETURN
40 GOTO 10              150 PRINT " HOURS
50 PRINT " SUN"         PASSING";
55 RETURN              155 RETURN
60 PRINT " SAND"        160 PRINT " ECHOED
65 RETURN              OVER";

```

165	RETURN	200	PRINT " FAINTLY
170	PRINT " BRIGHTLY		WHISPERED";
	DREW";	205	RETURN
175	RETURN	210	PRINT " YET AGAIN";
180	PRINT " DREAMILY	215	RETURN
	EASED";	1000	PRINT
185	RETURN	1001	PRINT
190	PRINT " SHADOWED	1002	PRINT
	OVER";	1003	RETURN
195	RETURN		

When you run this, you'll be quite pleased (or at least I was when I first wrote it) to find out how often, just by chance, quite attractive poems (!) are written by the program. Note the high number of RETURN commands. You'll remember that after going to a subroutine (GOSUB), the program executes the subroutine, then at the command RETURN, goes back to the line **after** the instruction to go to the subroutine. In this program, the line after the GOSUB instruction is 40, which instructs the computer to go further back, to line 10, to generate another random number. The decisions on where to place the semicolons (;), which tell the computer to print the next words on the same line, were made partly by running the program, and just seeing how the words fell, and also by the general decision that half the nouns (such as SUN, SAND and SEAGULLS) would end a sentence, that is, would not be followed by a semicolon. I also decided that adjectives (as in lines 120 and 140) would always allow for a word to follow them, and that about half the other "seeds" (lines 150 to 210) would do the same.

By looking at the program listing you'll see the lines 15, 20 and 30 make certain decisions, based on the random number generated. Line 15 ensures that, just under 5% of the time, "... " will be printed. Line 20 prints three blank lines (GOSUB 1000) just under 15% of the time, and line 30 directs the hardworking ZX80 to print one of the "seeds" just over 80% of the time. These proportions were worked out by running the program over and over again, adjusting the lines 15, 20 and 30 until — the majority of the time — a pleasant poem layout resulted. Probably you've realised that the program runs until the screen is full (which gives a fairly good poem length). If you wanted to limit the number of lines in the poem to less than full screen length, you could place a counter (or a FOR...NEXT) at the return target for the GOSUB at line 30 (i.e. at line 40). The kind of poem the program writes depends, as you can see, almost entirely on the words placed in the PRINT lines. The best way to decide on the words to go in the PRINT statements is to pick one topic, and then make every word and phrase relate to this topic.

The original POETRY program was written when I was on holiday, and after producing about 20 SEASIDE POEMS, I decide to input some

words and phrases about the city where the program was written. The changes in the program produced some remarkably effective poems (plus a generous crop of duds). If you want to see how it works, change the following lines using the EDIT facility, making sure you put a space just inside the quote marks.

1	PRINT , " SALZBURG	130	PRINT " MEMORIES IN
	POETRY"		STONE";
50	PRINT " CHURCHES";	140	PRINT " STEADFAST";
60	PRINT " BAROQUE	150	PRINT " TRADITIONS
	TOWERS";		RELIVED";
70	PRINT " MOUNTAIN	160	PRINT " ECHOED";
	VISTAS";	170	PRINT " COPPER
80	PRINT " MUSIC BY		DOMES";
	MOZART";	180	PRINT " DREAMING"
90	PRINT " FORTRESS";	190	PRINT " SHADOWED
100	PRINT " COBBLED		OVER";
	STREETS";	200	PRINT " FAINTLY
110	PRINT " TIMELESS";		WHISPERED"
120	PRINT " BELOVED";	210	PRINT " YET AGAIN"

Try writing a program, using words and phrases based on the last place you spent a holiday in, or pick a topic like "clouds", "kittens" or "vintage cars"...and see what you, the ZX80 and the Muse can create. Here is one poem written from a set of words and phrases about London.

...TOURISTS CROWD ALWAYS MOVING  
RUSHING, PUSHING I HAVE SEEN IT  
BIG BEN CHIMES, AND RIVER THAMES  
CATHEDRAL SPIRES  
ALL BUT FORGOTTEN RUSHING, PUSHING  
PIGEONS IN TRAFALGAR SQUARE  
I HAVE SEEN IT  
AT LAST, SUN. RIVER THAMES  
TIMELESS  
I HAVE SEEN IT

Not very brilliant, I guess, but acceptable. The "seeds" for this poem are:

1	PRINT , "POEMS OF	90	PRINT " AT LAST, SUN.";
	LONDON TOWN"	100	PRINT " STREETS OF ...";
50	PRINT " BOBBIES,";	110	PRINT " TIMELESS"
60	PRINT " TOURISTS	120	PRINT " ALL BUT
	CROWD";		FORGOTTEN";
70	PRINT " PIGEONS IN	130	PRINT " MEMORIES OF
	TRAFALGAR SQUARE"		MAJESTY,"
80	PRINT " CHAOS"		

- 140 PRINT " RIVER  
THAMES";
- 150 PRINT " BIG BEN  
CHIMES, AND";
- 160 PRINT " RUSHING,  
PUSHING";
- 170 PRINT " ALWAYS  
MOVING,"

- 180 PRINT " CATHEDRAL  
SPIRES"
- 190 PRINT " GREY RAIN  
FALLS ON...";
- 200 PRINT " I HAVE SEEN  
IT"
- 210 PRINT " MANY TIMES";

Notice how, in this program, link words like AND and ON finish some lines.

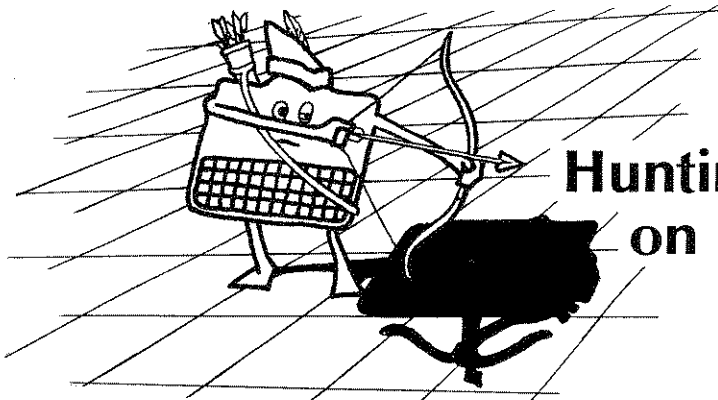
Here is another poem, from yet another set of words:

GRAVEYARDS ABOMINATIONS RELIVED  
 DARK, DARK SHRINKING WITCHES CACKLE  
 ECHOED CHILL OF ODDNESS CHANCES LOST  
 SHRINKING CALLING, CALLING, GRAVEYARDS  
 ECHOED CALLING, CALLING  
 DEATH IS NEAR, SAY WITCHES CACKLE  
 TURN AND REACH FOR  
 DARK, DARK WITCHES CACKLE  
 SKELETONS RATTLE  
 GRAVEYARDS

The "seeds" are as follows:

- 1 PRINT "BLACK  
SABBATH POETRY"
- 50 PRINT " GRAVEYARDS";
- 60 PRINT " SKELETONS  
RATTLE"
- 70 PRINT " CHILL OF  
ODDNESS";
- 80 PRINT " WITCHES  
CACKLE";
- 90 PRINT " HOPE NOW  
DEAD";
- 100 PRINT " CHANCES  
LOST";
- 110 PRINT " TIMELESS"
- 120 PRINT " DARK, DARK";

- 130 PRINT " MEMORIES OF  
PAIN,";
- 140 PRINT " SHRINKING";
- 150 PRINT " ABOMINATIONS  
RELIVED";
- 160 PRINT " ECHOED";
- 170 PRINT " CALLING,  
CALLING,";
- 180 PRINT " DEATH IS NEAR,  
SAY";
- 190 PRINT " SCREAMS ARE";
- 200 PRINT " TURN AND  
REACH FOR";
- 210 PRINT "AGAIN, AGAIN"



## Hunting on a Grid

There are many games, with names like HUNT THE HURKLE or BURIED TREASURE or DEPTH-CHARGE, which are based on trying to guess the location of one or more points on a grid. Each location is specified by quoting the co-ordinates of the point. To start our investigation of these games, input the following program, which sets up a very simple game of this type (you can leave out lines 10, 60, 70, 80 and 90 if you want to save time).

```
10 PRINT "HUNT GAME"
20 LET X = RND (10)
30 LET Y = RND (10)
40 PRINT "A ZX80 IS
   HIDDEN ON A 10 x 10
   GRID"
50 PRINT " YOU HAVE 10
   GUESSES TO FIND IT"
60 PRINT "INPUT YOUR
   GUESS, PRESSING
   NEWLINE"
70 PRINT "AFTER EACH
   DIGIT. TWO DIGITS
   ARE"
80 PRINT "NEEDED, THE
   FIRST ONE YOU INPUT"
90 PRINT "MUST BE FOR
   THE HORIZONTAL CO-
   ORDINATE"
100 FOR J = 1 TO 10
110 PRINT "FIRST NUMBER?"
120 INPUT A
130 PRINT A;
140 PRINT " SECOND
   NUMBER?"
145 INPUT B
150 PRINT B
160 IF A = X AND B = Y
   THEN GOTO 240
170 IF NOT A = X OR NOT B
   = Y THEN PRINT "TRY
   AGAIN"
180 INPUT B$
190 CLS
200 NEXT J
210 PRINT "END OF GAME"
220 PRINT "ZX80 WAS
   HIDDEN AT ";X,Y
230 STOP
240 PRINT "YOU HAVE
   FOUND IT"
250 STOP
```

Once you've played this a few times, you'll realise that you're really "shooting in the dark" when trying to guess the ZX80's location, and there is no feedback as to how close you are.

You can add the following lines to give you an idea of how you're going:

```
162 IF A = X AND NOT B =      164 IF NOT A = X AND B =
    Y THEN PRINT,A;" IS      Y THEN PRINT ,A;" IS
    RIGHT,";B;" IS NOT"      WRONG, ";B; " IS
                                RIGHT"
```

Add these lines, and run the program a few times. You can then add another line to give the player more information:

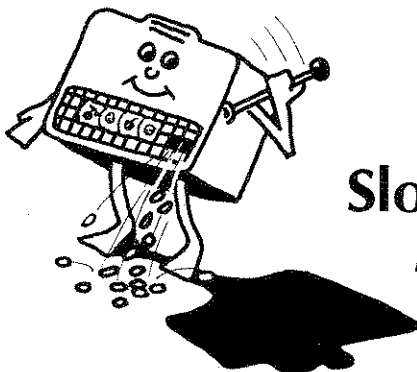
```
166 IF J = 6 THEN PRINT
    "CLUE: NUMBERS
    ADDED = ";X+Y
```

Of course, many other features can be added. Here's a version of the hunt game which allows the player to select the size of the grid, and gives clues in terms of direction.

```
10 PRINT ,"HUNT THE          152 PRINT C;" AND
    SPIDER"                  EAST/WEST?"
20 PRINT "(C) HARTNELL      155 INPUT D
    1980"                    156 CLS
25 PRINT                   160 IF C = A AND D = B
27 PRINT                   THEN GOTO 340
30 PRINT "IN THIS GAME A    165 PRINT "GUESS
    SPIDER WILL WEAVE"      NUMBER";);" WAS "
40 PRINT " A WEB AND        ;C;" ";D;
    HIDE ON IT"             " AND WAS"
45 PRINT                   170 PRINT "WRONG. TRY TO
47 PRINT                   THE" ;
50 PRINT "HEIGHT OF        180 IF C < A THEN PRINT
    WEB?";                  " NORTH " ;
60 INPUT X                 190 IF C > A THEN PRINT
70 PRINT X;" AND WIDTH?"   " SOUTH " ;
80 INPUT Y                 195 IF D < B THEN PRINT
85 CLS                     " EAST"
90 LET A = RND(X)          200 IF D > B THEN PRINT
100 LET B = RND(Y)         " WEST"
110 FOR J = 1 TO 5         210 PRINT
120 PRINT " THE SPIDER IS   220 PRINT
    HIDING ON A"           240 NEXT J
130 PRINT X;" BY ";Y;" WEB. 250 PRINT "GAME OVER.
    WHERE IS IT?"          SPIDER WAS AT ";A;" ";B
140 PRINT
142 PRINT
145 PRINT
    "NORTH/SOUTH?";
150 INPUT C                280 PRINT ,"ANOTHER
                                GAME?"
                                290 INPUT A$
                                300 CLS
                                310 IF A$ = "YES" THEN
                                    GOTO 30
```

320 PRINT "THANKS FOR  
PLAYING"  
330 STOP

340 PRINT "YOU FOUND  
THE SPIDER IN ";J;  
" TRIES"  
350 GOTO 280



## Slot/Fruit Machines

Once you've paid for your ZX80, bought some software (and this book) and possibly invested in some additional memory, your wallet may be getting a little light. Here's a program which you can use to get a little money out of your richer friends.

The random number generator (of course) can easily be used to simulate a slot machine, or fruit machine, game. A very simple program of this type, with two "fruits" could be as follows:

```

10 PRINT "SLOT MACHINE"
20 LET C = 5
30 LET A = 0
40 LET B = 0
50 FOR J = 1 TO 2
60 LET D = RND(2)
70 IF D = 2 THEN GOTO
   110
80 LET A = A + 1
90 PRINT "ORANGE"
100 GOTO 130
110 LET B = B + 1
120 PRINT "CHERRY"
130 NEXT J
140 IF A = 2 OR A = 0
   THEN LET C = C + 2
150 IF A = 1 THEN LET C =
   C - 1
160 IF C < 1 THEN GOTO 230
170 IF C > 10 THEN GOTO
   250
180 PRINT "YOU NOW HAVE
   E";C
190 PRINT "PRESS NEWLINE
   FOR NEXT GO"
200 INPUT E$
210 CLS
220 GOTO 30
230 PRINT "YOU HAVE
   BUSTED"
240 STOP

```



```

250 PRINT "YOU HAVE
      BROKEN THE BANK"
260 STOP

```

Input this program and run it a few times. It works as follows: Line 20 sets the counter for ORANGES (A) at 0, and line 30 sets the CHERRY counter (B) at 0. Your money is C, and line 20 sets it at £5 to begin the game.

Each play of the game requires two random numbers to be generated. These are counted by line 50 (and 130) and generated by line 60. If the random number is 2, program control moves to line 110 where B becomes  $B + 1$  and CHERRY is printed. Control then goes back to line 50. If the random number is 1, A becomes  $A + 1$ , and the computer prints ORANGE before going back to line 50 for the next number.

After two numbers have been generated, and the "fruits" printed out, the computer checks to see if both are the same (if they are, A will equal 2 or 0) and if so, adds two to your money (that is, lets  $C = C + 2$ ). If the fruits are not the same, the computer takes £1 off you (that is, lets  $C = C - 1$ ). Can you see how the computer knows the two fruits were different?

Next, it prints out your stake (line 160). The computer checks this amount. If it is less than £1, control goes to line 230 to print YOU HAVE BUSTED. If it is greater than 10 control moves to line 250 where the computer prints YOU HAVE BROKEN THE BANK. If neither of these conditions are true, the computer acts on the next line (190) which instructs you to press NEWLINE for the next go.

That may seem a little complicated spelt out in detail, but you should be able to follow it through on the program. Once you're sure you understand the workings of this program, write your own version with three fruits, and thus make it more like a "real" slot machine. You'll have to add some lines between others in my program to do this, and change a few lines completely. I'm not going to give you a program for a "three fruits" machine as you will benefit far more from working out your own program to simulate the working of such a machine. And it is far more fun playing a game with a program you've written yourself, than just feeding in someone else's work.

Once you've written and played with the three fruits version, read on to see one way of writing a four fruits slot machine (but with pretty odd fruit). Note that in this program, different winning combinations are worth different amounts, and generate comments.

```

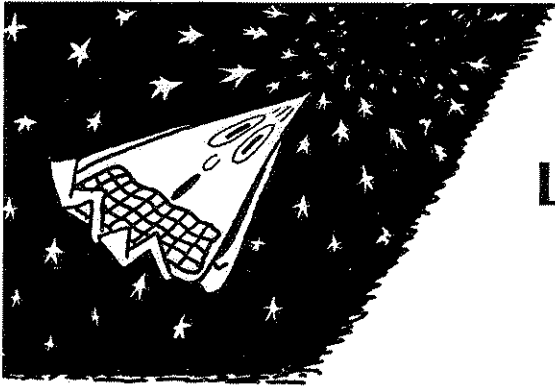
10 PRINT "SLOT MACHINE"
20 PRINT "(C) HARTNELL
   1980"
30 PRINT
40 PRINT
50 PRINT
60 PRINT
70 PRINT "STARTING
   STAKE (£<20)?"
80 INPUT M
90 CLS
100 LET C = 0

```

```

110 LET D = 0
120 LET E = 0
130 LET F = 0
140 FOR J = 1 TO 4
150 LET B = RND(4)
160 IF B = 2 THEN GOTO
    220
170 IF B = 3 THEN GOTO
    250
180 IF B = 4 THEN GOTO
    280
190 LET C = C + 1
200 PRINT "UNCLE"
210 GOTO 300
220 LET D = D + 1
230 PRINT "CLIVES"
240 GOTO 300
250 LET E = E + 1
260 PRINT "MAGIC"
270 GOTO 300
280 LET F = F + 1
290 PRINT "ZX80"
300 NEXT J
310 IF C = 1 AND D = 1
    AND E = 1 AND F = 1
    THEN GOTO 370
320 IF C = 4 OR D = 4 OR
    E = 4 OR F = 4 THEN
    GOTO 370
330 IF C = 3 OR D = 3 OR
    E = 3 OR F = 3 THEN
    GOTO 390
350 LET M = M - 2
360 GOTO 420
370 LET M = M + 10
375 PRINT "(shift A, space,
    shift A) JACKPOT (shift A,
    space, shift A)"
380 GOTO 420
390 LET M = M + 2
400 PRINT "(shift S) 3 OF A
    KIND ( shift S)"
420 IF M < 1 THEN GOTO
    490
430 IF M > 49 THEN GOTO
    540
432 PRINT
435 PRINT
437 PRINT
440 PRINT "YOU NOW
    HAVE £";M
445 PRINT
450 PRINT "PRESS NEWLINE
    FOR NEXT SPIN"
460 INPUT A$
470 CLS
480 GOTO 100
490 PRINT "END OF GAME.
    YOU ARE BROKE"
500 PRINT "ANOTHER
    GAME?"
510 INPUT B$
520 IF B$ = "YES" THEN
    GOTO 60
522 CLS
525 PRINT "OK, BYE BYE"
530 STOP
540 PRINT "YOU HAVE
    £";M;" AND HAVE"
550 PRINT "BROKEN THE
    BANK"
560 GOTO 500

```



## Lost in Space

We'll return now to the HUNTING ON A GRID idea, and work on finding objects which are located on more than two axes, that is, are hidden in three- (or even four-) dimensional space. Imagine our spider is hiding inside a cube, each side of which is X units long. Each point in the cube can be specified by giving three co-ordinates: height, width and depth. Before reading on to see how I've done it, try to work out a program in which a space capsule is lost inside a cube of space, with each side of the cube four kilometres long.

```

10 PRINT "LOST IN SPACE"
15 PRINT
20 PRINT "YOU HAVE 15
   HOURS TO FIND"
30 PRINT "A CAPSULE
   LOST IN"
40 PRINT "A 7KM CUBE OF
   SPACE"
50 LET A = RND(7)
60 LET B = RND(7)
70 LET C = RND(7)
80 FOR J = 1 TO 15
85 PRINT
90 PRINT "INPUT 3 SEARCH
   CO-ORDINATES"
100 INPUT D
110 INPUT E
120 INPUT F
125 CLS
130 PRINT "D,E,F"
140 IF A = D AND B = E
   AND C = F THEN GOTO
   320
145 PRINT
150 PRINT "(shift A) WRONG
   (shift A)"
155 PRINT
160 PRINT "HOURS OF AIR
   LEFT - ";15 - J
170 PRINT
180 PRINT "MOVE ";
190 IF A > D THEN PRINT
   "UP ";
200 IF A < D THEN PRINT
   "DOWN ";
210 IF NOT B = E THEN
   PRINT "ACROSS ";
220 IF C > F THEN PRINT
   "FORWARDS"
230 IF C < F THEN PRINT
   "BACKWARDS"
240 PRINT
250 IF J = 14 THEN PRINT
   "DANGER - DEATH
   IMMINENT"
260 NEXT J

```

270 CLS	340 PRINT "16-J"; "HOURS OF AIR
280 PRINT	A LEFT"
290 PRINT "FAILURE —	350 PRINT
ASTRONAUTS DEAD"	360 PRINT "ANOTHER
295 PRINT	MISSION?"
300 PRINT "CAPSULE WAS	370 INPUT H\$
AT ";A;" ";B;	380 CLS
" ";C	390 IF H\$ = "YES" THEN
310 GOTO 350	GOTO 20
320 PRINT	400 PRINT "FAREWELL,
330 PRINT "YOU FOUND	BRAVE CAPTAIN"
THEM WITH"	410 STOP

## Arrays

Arrays, and the use of the DIM (dimension) statement, puzzle many newcomers to BASIC. The DIM statement is used if you want to set up a list with a 'name' (the name is a letter like A, B or whatever). For example, you might want the numbers 1 to 15 in the list named A, in the form LET A(0) = 1, LET A(1) = 2 and so on to LET A(14) = 15. To tell the ZX80 you need an array to hold these *elements* (1, 2 and so on to 15) input:

```
10 DIM A(14)
```

The figure in brackets can be one less than the number of items or elements you want in the list because, in ZX80 BASIC, an array contains one more element than the number you place in the brackets; the first element is A(0), not A(1). However, you can have a much bigger array if you like, provided you don't exceed the memory. In practice, it is sufficient to put the number of elements you want in the brackets, 'forgetting' that you are actually creating an array one element bigger than you need. The number inside the brackets is known as the **subscript**, and an element of the form F(2) or K(9) is called a subscripted variable.

Here's a program to show the DIM statement:

10 DIM A(10)	40 PRINT "A(";J;")=";A(J)
20 FOR J = 0 TO 10	50 NEXT J
30 LET A(J) = 2*J	

When you run this, you will get:

```
A(0) = 0
A(1) = 2...and so on to...
A(10) = 20
```

Change line 10 to DIM A(5) and run the program again. This time you'll get just A(0) = 0...to...A(5) = 10. The program stopped at this point (giving the error code 3/30, meaning that the subscripted variable required, i.e. J = 6 to J = 10 was out of range). Change line 10 now to DIM A(20) and run the program. You'll find you get exactly the same result as having DIM (A10). As I said before, you change nothing, in practice, by having a larger array than you actually want. Now add the following lines:

```

10 DIM A(10)
60 INPUT B$
70 FOR Z = 0 TO 15
80 LET A(Z) = 3*Z
90 PRINT "A(";Z;") = ";A(Z)
100 NEXT Z

```

When you run this, you find the same A(0) = 0 through to A(10) = 20 following by the symbol asking you to input a string (or, as in this case, press NEWLINE). The ZX80 will then display:

```

A(0) = 0
A(1) = 3
...down to...A(10) = 30

```

The error code 3/80 will be displayed, because the demand made on the array by line 80 was greater than the array defined by line 10 (that is, line 70 made the next Z equal 11, and line 80 therefore wanted a subscripted variable called A(11) which, of course, it could not find because the array only had room for 11 elements).

Here's a program to show the DIM statement in use.

```

10 LET R = 1
20 PRINT "LIFE CLASH"
30 PRINT "(C) HARTNELL"
32 PRINT
35 PRINT
37 PRINT
40 PRINT "LIFE FORM
ONE?"
50 INPUT O$
60 CLS
70 PRINT "LIFE FORM
TWO?"
80 INPUT T$
90 CLS
100 PRINT "HOW MANY OF
EACH TO START (<5)?"
110 INPUT A
120 CLS
130 DIM O(A)
140 DIM T(A)
150 FOR Y = 1 TO A
160 LET O(Y) = 1
170 LET T(Y) = 1
180 NEXT Y
185 PRINT
187 PRINT
190 PRINT "
GENERATION ";R
191 LET R = R + 1
192 PRINT
193 PRINT
194 PRINT
195 FOR Y = 1 TO A
200 IF NOT O(Y) = 1 THEN
GOTO 230
210 PRINT O$
220 GOTO 260
230 LET J = RND(2)
240 IF J = 1 THEN
PRINT," ";
250 IF J = 2 THEN
PRINT,T$,T$;
260 NEXT Y
270 PRINT

```

```

280 PRINT
290 FOR Y = 1 TO A
300 IF NOT T(Y) = 1 THEN
    GOTO 330
310 PRINT ,T$;
320 GOTO 360
330 LET J = RND(2)
340 IF J = 1 THEN PRINT,
    ""';
350 IF J = 2 THEN
    PRINT,O$,O$;
360 NEXT Y
365 IF R = 4*A - 1 THEN
    PRINT ""'(shift A)FINAL
    NEXT (shift A)''
370 IF R = 4*A THEN
    GOTO 1000
375 PRINT
377 PRINT
380 PRINT "INPUT N FOR
    NEXT GENERATION"
390 INPUT N$
400 IF NOT N$ = "N" THEN
    GOTO 1000
410 CLS
420 LET M = RND(A)
430 LET F = RND(2)
440 IF F = 1 THEN LET
    O(M) = 0
450 IF F = 2 THEN LET
    T(M) = 0
460 GOTO 190
1000 PRINT
1010 PRINT
1020 PRINT
1030 PRINT "DOMINANT
    SPECIES?"
1040 STOP

```

Run this a few times (symbols are more effective than using HUMAN and ALIEN, or CAT and MOUSE for the life forms) and then read through the listing until you have worked out what each part of the program does. Now follows a very simple HUNT THE HURKLE program using the DIM statement. Input the program, run it a few times, then read through to find out how it works.

```

10 PRINT "DIM SPIDER"
20 DIM A(4)
30 DIM B(4)
50 FOR G = 1 TO 4
60 LET A(G) = 2
70 LET B(G) = 2
80 NEXT G
90 LET K = RND(4)
100 LET L = RND(4)
110 LET A(K) = 1
120 LET B(L) = 1
130 FOR D = 1 TO 5
140 PRINT "WHERE IS
    SPIDER, TRY
    NUMBER ";D
150 INPUT T
160 INPUT Y
170 IF A(T) = 1 AND B(Y) =
    1 THEN PRINT "YOU
    FOUND IT"
180 IF A(T) = 1 AND B(Y) =
    1 THEN STOP
190 NEXT D
200 PRINT "SORRY, TIME IS
    UP"
210 PRINT "SPIDER WAS
    AT";K,L
220 STOP

```

Having run this program a few times, and examined the listing, you are probably asking yourself what was achieved by using the DIM statements which could not have been achieved without them. The answer is: Nothing. However, the DIM comes into its own when you want to hide more than one object on a grid, without creating a whole

set of co-ordinates of the type LET A = RND(4), LET B = RND(4), LET C = RND(4) and so on, to put the first hidden object at A,B; the second at C,D; and so on.

Before we look at this, here are two more programs which hide a single object.

```

5 PRINT "DIMMER
   SPIDER"
10 DIM A(2)
20 DIM B(2)
30 FOR C = 1 TO 2
40 LET A(C) = RND(4)
50 NEXT C
60 FOR J = 1 TO 7
70 PRINT "WHERE IS THE
   SPIDER — ATTEMPT ";J
80 FOR C = 1 TO 2
90 INPUT B(C)
100 NEXT C
110 PRINT B(1)," ";B(2);
120 IF A(1) = B(1) AND
   A(2) = B(2) THEN GOTO
   200
130 IF A(1) = B(1) AND NOT
   A(2) = B(2) THEN GOTO
   220
140 IF NOT A(1) = B(1) AND
   A(2) = B(2) THEN GOTO
   240
150 PRINT " NO"
160 IF J = 5 THEN GOSUB
   260
170 NEXT J
180 PRINT "TIME IS UP.
   SPIDER WAS AT ";
   A(1);" ";A(2)
190 GOTO 205
200 PRINT " YOU HAVE
   FOUND IT"
205 POKE 16421, 24
210 STOP
220 PRINT ,A(1); " IS RIGHT"
230 GOTO 170
240 PRINT ,A(2); " IS RIGHT"
250 GOTO 170
260 PRINT "HINT:
   LOCATIONS ADD UP
   TO ";A(1)+A(2)
270 RETURN

```

If you like, you can change line 160 to read:

```
160 IF J = 3 OR J = 7 THEN GOSUB 260
```

This just reinforces the same hint when J = 7 as when J = 3.

If you want the 'hint' to come at random, you could change line 160 to:

```
160 IF J = 2*A(1) THEN GOSUB 260
```

or to:

```
160 IF J = 2*A(2) THEN GOSUB 260
```

Another version of this line, suggested by National ZX80 Users Club member Ian Rodgers<sup>8</sup>, is:

```
160 IF RND(7) < J THEN GOSUB 260
```

This final version is probably the best, because it means that the closer to J = 7 you get, the more likely you are to be given a hint.

The only other thing to note is line 205 (POKE 16421, 24), again suggested by Ian Rodgers. This line removes the code 9/210 at the bottom left of the screen which tells you the STOP command has been

executed. You can use this line in any program where you feel the code number will detract from the final display on the screen.

The core of the program DIMMER SPIDER can be used to produce a far more interesting game, and in this program we will introduce a new ZX80 function.

```

5 PRINT "PESKY PIKSY"          190 GOTO 305
10 RANDOMISE                   200 CLS
15 CLEAR                       205 FOR Y = 1 TO 10
20 DIM A(2)                    210 PRINT
25 DIM B(2)                    220 NEXT Y
30 FOR C = 1 TO 2              225 PRINT "YAY (shiftA)
                                CAPTURED"
40 LET A(C) = RND(7)           230 GOTO 305
50 NEXT C                      235 PRINT ,A(1); " IS RIGHT"
60 FOR J = 1 TO 11            240 GOTO 170
70 PRINT "WHERE IS PIKSY
  - ATTEMPT ";J               250 PRINT ,A(2); " IS RIGHT"
80 FOR C = 1 TO 2             255 GOTO 170
90 INPUT B(C)                 260 LET G = RND(5)
100 NEXT C                     270 IF G = 1 THEN PRINT
                                "HINT: LOCATIONS ADD
110 PRINT B(1);" ";B(2);      UP TO ";A(1) + A(2)
120 IF A(1) = B(1) AND        280 IF G = 2 THEN PRINT,
  A(2) = B(2) THEN GOTO      "HINT: DIFERENCE
  200                          BETWEEN LOCATIONS
130 IF A(1) = B(1) AND NOT    IS ";ABS(A(1) - A(2))
  A(2) = B(2) THEN GOTO      290 IF G > 2 THEN GOTO 320
  235                          300 RETURN
140 IF NOT A(1) = B(1) AND    305 POKE 16421, 24
  A(2) = B(2) THEN GOTO      310 STOP
  245                          320 PRINT "I WAS AT:"
150 PRINT "NO"                A(1);" ";A(2);" NOW"
160 IF RND(10) < J THEN       330 PRINT "I AM MOVING"
  GOSUB 260                    340 PRINT "PRESS
170 IF J = 4 OR J = 7 THEN    NEWLINE"
  CLS                          350 INPUT A$
175 NEXT J                    360 CLS
180 PRINT "TIME IS UP,       370 GOTO 10
  PIKSY WAS AT
  ";A(1);" ";A(2)

```

Have a look at line 280. The function ABS stands for "absolute". If a number is positive, the absolute value of that number is just the number. If a number is negative, the absolute value is the number multiplied by  $-1$  (i.e., the number without its negative sign). If you leave out ABS in line 280, the clue will be so specific it will almost certainly give the location away, so ABS makes for a better game. Lines 200 to 220 perform an interesting task. After clearing the screen



(line 200) the FOR/NEXT loop moves the letters to be printed (line 225) to about the middle of the screen. This makes for a more attractive display.

The next program locates a number of objects (10) on a grid, and gives a score based, of course, on how many you hit. However, if more than one object is at the same location you get more than one score. The program also awards you a "rating" at the end.

```

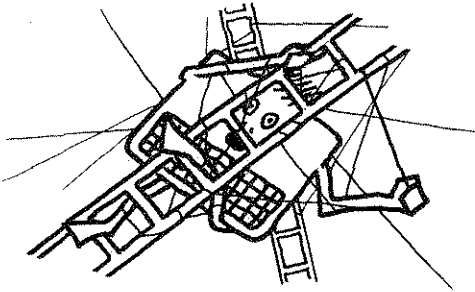
2   PRINT "ALIENS +           300 PRINT
   ASTEROIDS"                310 PRINT,
5   LET K = 0                 CHR$(128);CHR$(128);
20  DIM A(10)                 " TIME IS UP"
30  DIM B(10)                320 PRINT
40  DIM C(10)                330 PRINT " (shift S twice)
50  DIM D(10)                 YOU HIT ";K;" ALIENS
100 FOR J = 1 TO 10           (shift S twice)"
110 LET A(J) = RND(4)         340 PRINT
120 LET B(J) = RND(4)         350 PRINT "MARKSMAN
130 NEXT J                    RATING: ";K*100/
160 FOR Z = 1 TO 8           (1 + RND(3))
170 PRINT CHR$(129 +         360 PRINT
   RND(10));" SHOT ";Z;     370 PRINT "THEY ARE
   "(shift Q)";             HIDDEN AT:"
180 INPUT C(Z)               380 PRINT
190 INPUT D(Z)               390 FOR J = 1 TO 10
200 PRINT C(Z);" ";D(Z);" "; 400 PRINT A(J); " ";B(J)
230 FOR J = 1 TO 8           410 NEXT J
240 IF A(J) = C(Z) AND B(J)  420 STOP
   = D(Z) THEN LET K =      430 CLS
   K + 1                    440 FOR H = 1 TO 10
250 IF A(J) = C(Z) AND B(J)  450 PRINT
   = D(Z) THEN PRINT        460 NEXT H
   "HIT";CHR$(128);         470 PRINT " YOU GOT
   "SCORE: ";K              THEM ALL";
260 IF K = 10 THEN GOTO     480 FOR U = 1 TO 29
   430                       490 PRINT CHR$(128);
270 NEXT J                   500 NEXT U
280 NEXT Z                   510 GOTO 470
290 CLS

```

The screen display for this game when run is, unfortunately, a little cramped. However, a more elegant display would use up the available space very quickly, and generate an error code 4.

You'll notice a randomly generated graphical character before the word SHOT each time. These graphics are the inverse of the graphics available direct from the keyboard. The number of every character is given in the manual, and to print the character corresponding to a

particular number, you just input PRINT CHR\$(number of character). The black blob you get on a hit is the inverse of the space (character number 128). The same character is used at the end if you manage to destroy all the aliens. The "marksman rating" at the end adds a little interest, and is actually related to your skill in destroying the aliens. The more you killed (K) the higher your score. It is divided by (1 + RND(3)) just to make it a little more interesting. Can you see why the line does not read  $K*100/RND(4)$  or  $(K/(1 + RND(3))) * 100$ ?



## Strings and Ladders

The discipline of programming in just 1K will stand you in good stead when later you add extra memory. It is very easy to get into the habit of programming sloppily, inefficiently or inelegantly when you have memory to spare. One way of making the most of your 1K is to use strings, assigned at the start of the program, for phrases which you intend to use at various parts of the program. The following game SNAKES AND LADDERS shows these techniques in use.

```

5   LET G$ = "SCORE IS "
10  LET A$ = "SNAKE"
15  LET H$ = ":-"
20  LET B$ = "LADDER"
25  LET K$ = " WINS"
30  LET L$ = " WORTH"
40  PRINT ",A$,"S AND
    ";B$,"S"
50  PRINT
55  "PLAYER 1?"

60  INPUT C$
70  PRINT ,"PLAYER 2?"
80  INPUT D$
90  PRINT "TO START
    GAME ";
100 LET A = 0
110 LET B = 0
130 PRINT "PRESS NEWLINE"
140 INPUT E$
150 GOSUB 260

```

```

155 LET K = 0
160 GOTO 280
165 LET A = A + E
170 PRINT C$,H$,M$,L$,E
180 PRINT ,G$,A
190 PRINT
195 IF A > 19 THEN GOTO
    370
200 LET K = 1
210 GOTO 280
215 LET B = B + E
220 PRINT D$,H$,M$,L$,E
230 PRINT ,G$,B
232 PRINT
233 PRINT
235 IF B > 19 THEN GOTO
    390
240 PRINT "FOR NEXT
    MOVE ";
250 GOTO 130
260 FOR S = 1 TO RND(300)
265 CLS
270 NEXT S
275 RETURN
280 LET C = RND(5)
290 IF C < 3 THEN LET
    T = -1
300 IF C > 2 THEN LET T = 1
310 IF C < 3 THEN LET M$ =
    A$
320 IF C > 2 THEN LET M$ =
    B$
330 LET P = RND(6)
340 LET E = P*T
350 IF K = 0 THEN GOTO
    165
360 IF K = 1 THEN GOTO
    215
365 CLS
370 PRINT "(shift A)";C$,K$;
    " WITH ";A - B;
    " POINTS"
385 GOTO 410
390 CLS
395 PRINT "(shift A)";D$,K$;
    " WITH ";B - A;" POINTS"
400 PRINT "ANOTHER
    GAME?"
410 INPUT N$
420 CLS
430 IF N$ = "YES" THEN
    GOTO 90
440 PRINT "OK. BYE"
450 STOP

```

This program shows quite clearly how strings can be used to save memory (in fact, the idea was carried a little too far, just to make the technique plain). There are a few other things in the listing from which we can learn. Look at lines 290 to 320. These determine if the player will get a SNAKE (and a negative score) or a LADDER (and a positive score). If line 280 had read LET C = RND(2) there would be a pretty good chance that the game would never end, because each player's gains would approximately equal his losses, and the players would lose interest long before somebody happened to chance a win. By making the odds in favour of a ladder (a ladder, and positive score are generated about 60% of the time) the program ensures that both players' scores gradually build up.

The next program is based on exactly the same idea as SNAKES AND LADDERS but takes longer to play, and — because it has more variables — is considerably more interesting.

```

1 LET D$ = "MILES TO"
2 LET E$ = "ALL OK"
3 LET F$ = "SMASH"
4 LET G$ = "COPS"
5 LET H$ = "PUNCTURE"
6 LET J$ = "POINTS"
7 LET K$ = "OIL"

```

```

8 LET L$ = " WINS WITH"
9 LET N$ = "PETROL "
10 PRINT "ROADRACE"
20 RANDOMISE
30 PRINT "DRIVER 1?"
40 INPUT B$
45 LET A = 0
50 PRINT "AND 2?"
55 LET B = 0
60 INPUT A$
70 PRINT "PRESS
NEWLINE"
75 INPUT C$
80 FOR J = 1 TO 299
84 CLS
85 NEXT J
90 PRINT ,A$
95 LET K = 0
100 GOTO 180
105 LET A = A + Z
110 IF A > 390 THEN GOTO
270
115 IF A < 0 THEN LET A = 0
120 PRINT M$;397 - A;D$
125 PRINT
130 PRINT ,B$
135 LET K = 1
140 GOTO 180
145 LET B = B + Z
150 IF B > 390 THEN GOTO
280
155 IF B < 0 THEN LET B = 0
160 PRINT M$;397 - B;D$
170 GOTO 70
180 LET C = RND(11)
190 IF C < 6 THEN GOSUB
210
200 IF C > 5 THEN GOSUB
C*10 + 150
205 IF K = 0 THEN GOTO
105
207 IF K = 1 THEN GOTO
145
210 LET M$ = E$
212 LET Z = 27
215 RETURN
220 LET M$ = F$
222 LET Z = -19
225 RETURN
230 LET M$ = G$
232 LET Z = -7
235 RETURN
240 LET M$ = H$
242 LET Z = -12
245 RETURN
250 LET M$ = N$
252 LET Z = -5
255 RETURN
260 LET M$ = K$
262 LET Z = -23
265 RETURN
270 PRINT
A$;L$;39*(A - B) + A - B;J$
275 STOP
280 PRINT
B$;L$;39*(B - A) + B - A;J$
285 STOP

```

There are a few things to note about this program. It is meant to be a race, and although the drivers in this game actually go backwards, they are unlikely to go backwards beyond their starting point. Therefore, if at anytime the score becomes less than zero, it is reset to zero (lines 115 and 155). Also in this game, as in SNAKES AND LADDERS, there is a slightly better than even chance of getting an ALL OK (and a positive mileage). See if you can find the line in the listing that ensures this.

Comparing line 260 in SNAKES AND LADDERS with line 80 in ROADRACE is instructive. In the first program, the delay is random (and varies from a delay of practically zero when  $S = 1$  to a much longer period when  $S = 300$ ). In ROADRACE, the delay is set (purely

arbitrarily at 299). There is no reason why you can't set the delay in either program to zero (delete the FOR/NEXT loop, but leave in the CLS) or any number you like, or — if you prefer the unexpected — at a random number. Do not set the random limit too high (like, say, LET S = 1 TO RND(5000)) because you run the risk of (a) overloading the memory, depending on the program; (b) losing interest in the game if the delay is close to 5000 time and again; and (c) you may think you've just got a long delay when, in fact, your ZX80 has gone into an infinite loop for some reason (this is likely if you've either made a mistake when inputting the program, or with some ZX80's, the computer has become very hot).

The next game — 52 BLUFF — uses the string idea again. In this game the ZX80 deals two cards. If you think the next card to be dealt will lie between the first two, you place a bet of your choice. This game is more interesting than some computer betting games like FRUIT MACHINE because you can decide on the likelihood of a win and adjust your bet accordingly. You can even decide not to bet at all.

```

10 LET S = 30
20 LET A$ = " "
30 LET F$ = "CARD 1: "
40 LET G$ = "CARD 2: "
50 LET H$ = "CARD 3: "
60 LET A = RND(13)
70 LET B = RND(13)
80 IF B = A THEN GOTO 70
90 PRINT "“(shift Q twice)
STAKE: £";S;"(shift Q
twice)”
100 PRINT F$;A$;
110 LET Z = A
120 GOSUB 480
130 PRINT
140 PRINT "G$;A$;
150 LET Z = B
160 GOSUB 460
170 PRINT "“(shift Q)“WAGER?”
180 INPUT C
190 CLS
200 IF C = 0 THEN PRINT,
“COWARD”
210 PRINT "“(shift Q)A,B
220 LET D = RND(13)
230 IF D = A OR D = B
THEN GOTO 220
240 PRINT
250 PRINT H$;A$;
260 LET Z = D
270 GOSUB 480
280 IF A < D AND D < B
THEN GOSUB 360
290 IF A > D AND D > B
THEN GOSUB 360
300 IF D > B AND D > A
THEN GOSUB 410
310 IF D < B AND D < A
THEN GOSUB 410
320 IF S < 1 THEN GOTO 440
330 INPUT K$
340 CLS
350 GOTO 60
360 LET S = S + 2*C
370 IF S > 199 THEN PRINT
“YOU HAVE BROKEN
THE BANK”
380 IF S > 199 THEN STOP
390 IF NOT C = 0 THEN
PRINT "“(shift Q)YOU WIN £”;2*C
400 RETURN
410 LET S = S - C
420 IF NOT C = 0 THEN
PRINT "“(shift Q)YOU LOSE £”;C
430 RETURN
440 PRINT "“(shift Q)YOU
ARE BROKE”
450 STOP

```

```

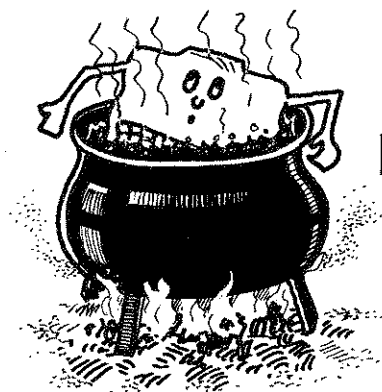
460 IF NOT Z = 1 AND NOT
    Z > 10 THEN PRINT Z
470 IF Z = 1 THEN PRINT
    "A"
480 IF Z = 11 THEN PRINT
    "JK"

```

```

490 IF Z = 12 THEN PRINT
    "QN"
500 IF Z = 13 THEN PRINT
    "KG"
510 RETURN

```



## Hot Sauce

You will recall that earlier in this book there was program in which the ZX80 thought of a number, then gave you hints to help you guess it. You probably realised that if you started with 50 as your first guess, it was pretty easy to narrow down the number by going to either 25 or 75 on the second guess. This method allowed you to get the correct number fairly easily. Here is another program which appears somewhat similar. But it is far harder to work out a system to beat it.

```

10 PRINT "HOT SAUCE"
20 PRINT "(C) HARTNELL
    1980"
30 PRINT
40 PRINT "WHATS YOUR
    NAME, PARDNER?"
50 INPUT A$
60 CLS
70 PRINT
80 PRINT
90 LET S = 0
100 PRINT "OK, ";A$;" , I AM
    THINKING OF"

```

```

110 PRINT "A NUMBER
    BETWEEN 1 AND 100"
120 PRINT "YOU HAVE 12
    GUESSES"
130 LET J = RND(99)
140 PRINT
150 LET S = S + 1
160 IF S = 13 THEN GOTO
    420
170 PRINT "WHAT NUMBER
    AM I THINKING OF?"
180 INPUT A
190 CLS

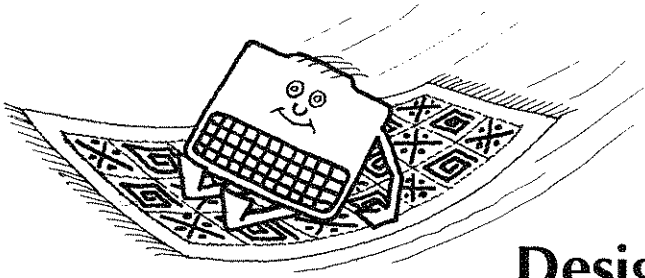
```

```

200 IF A = J THEN GOTO 360
210 IF A < J THEN GOTO 300
220 IF A - J < 5 THEN PRINT "BOILING, ";A$
230 IF A - J < 12 AND A - J > 4 THEN PRINT "HOT"
240 IF A - J < 25 AND A - J > 11 THEN PRINT "WARM"
250 IF A - J < 45 AND A - J > 24 THEN PRINT "COLD"
260 IF A - J > 44 THEN PRINT "FREEZING,
BABY"
270 PRINT
280 PRINT "NEXT GUESS?"
290 GOTO 150
300 IF J - A < 5 THEN PRINT "VERY, VERY
CLOSE"
310 IF J - A < 12 AND J - A > 4 THEN PRINT "PRETTY CLOSE"
320 IF J - A < 25 AND J - A > 11 THEN PRINT "JUST SO-SO"
330 IF J - A < 45 AND J - A > 24 THEN PRINT "PRETTY HOPELESS"
340 IF J - A > 44 THEN PRINT "PATHETIC, ";A$
350 GOTO 280
360 PRINT "YOU WERE RIGHT, ";A$
370 PRINT "I WAS THINKING OF ";J
380 PRINT "SO YOU GET ANOTHER GO"
390 LET S = 0
400 GOTO 130
410 CLS
420 PRINT "SORRY, ";A$; "YOU DIDNT GUESS IT"
430 PRINT
440 PRINT "I WAS THINK OF ";J
450 STOP

```

The term "absolute" (ABS on the keyboard) was introduced just after the listing for PESKY PIKSY. Refer back to this if you're not sure what ABS does. The HOT SAUCE program could be written quite differently from the above, by using the ABS. In the listing here, the ZX80's response to your guess is determined by whether the number you guess is higher or lower than the one it is thinking of, *and* by the difference between the numbers. If you rewrite HOT SAUCE using ABS, you'll find that only the difference between the numbers will determine the comment ("PATHETIC" or "BOILING" or whatever) the ZX80 will make. And just as HOT SAUCE is more difficult to play than GUESS MY NUMBER, HOT SAUCE with ABS is harder than without it. As an exercise, write a HOT SAUCE program using ABS. You should find it uses less memory than the above listing, so you could program the ZX80 to make much longer comments on your guesses.



## Designs on your VDU

We have used the RND function time and time again in programs. It is one of the most useful features for games. But, as I pointed out earlier, the random number generated is not really a random number at all, but is one of a very, very long list of numbers, so long that the number appears to be random. The use of the RANDOMISE function (on the J key) sets the starting point of the long, long list of numbers to a number equal to the number of times your TV screen has been scanned since the ZX80 was turned on. This next program reads the number of frames time and time again, each time a little later than before, and uses this number to generate a random number which, in turn, is used to instruct the ZX80 to print a specific "character".

You'll find in your manual a list of characters and their corresponding codes.

If you told your computer to print CHR\$(12) you would find it would print the pound sign (£). CHR\$ means the character whose code follows in brackets. A simple program to fill the screen with randomly generated characters, and their corresponding codes, is as follows:

```
10 LET K = 1 + RND(62)          30 PRINT "CODE NO. ";K;
20 IF K > 24 AND K < 28        " IS ";CHR$(K);" ";
   THEN GOTO 10                40 GOTO 10
```

(The one is added to the randomly generated number to avoid printing the character whose code is 1, a blank space, and the characters whose codes are 25, 26 and 27 are just punctuation marks.) The inverse graphics are printed by referring to their codes (numbers 130 to 139) and you can fill your screen with these by running the following little program:

```
10 LET K = RND (10)
20 PRINT CHR$(K + 129);
30 GOTO 10
```



By combining, more or less, the preceding program with the RANDOMISE function, we can develop an interesting program. Try the following:

```
10 FOR J = 1 TO 10          120 LET D = RND(22)
20 PRINT                   130 PRINT CHR$(D + 165);
30 PRINT "MAGIC CARPET"   140 NEXT H
40 NEXT J                  150 PRINT CHR$(212)
50 PRINT "IF YOU PRESS    160 PRINT
  NEWLINE I WILL"        170 FOR J = 1 TO 48
60 PRINT "CREATE A        180 RANDOMISE
  CARPET DESIGN FOR      190 LET Z = RND(11)
  YOU"                   200 PRINT CHR$(Z + 128);
70 INPUT A$              210 NEXT J
80 CLS                   220 FOR J = 1 TO 45 + G
90 LET G = RND(3)        230 LET Z = RND(10)
100 PRINT "CODE NAME      240 PRINT CHR$(Z + 1);
  FOR DESIGN:— ";        250 NEXT J
  CHR$(212);             260 GOTO 180
110 FOR H = 1 TO 4 + G
```

There are several things we can learn from this program. Lines 10 to 40 simply print the title ten times, which makes a more interesting start than just having the title printed once near the top of the screen. A similar idea is to run a number of blank PRINT statements in a FOR/NEXT loop to move the display to the centre of the screen. We will be using this idea in later programs, such as FRENZY. But for now, going back to MAGIC CARPET, line 90 sets G equal to a random number in the range 1 to 3. The value of G is used in lines 110 and 220. Lines 100 to 150 create a name for the design, by printing the characters (all inverse letters) whose codes are generated by line 120. CHR\$(212) is the inverse quote marks and these are printed at each end of the name of the design. The naming part of the program is followed by two FOR/NEXT loops, linked by a final GOTO command. The RANDOMISE is within the first FOR/NEXT loop.

Once you've run this program a few times, rewrite it by adding a further FOR/NEXT loop (with the FOR in line 165, and the NEXT in place of 260) to stop the program before it crashes because the screen is full, and to allow you to have another go without having to go back into command mode first. Set the limit of this last loop so as to get the maximum amount of "carpet" on the screen. Experiment with the size of the other two loops, and with the maximum value of G and see what this does to the final display.



## First Steps Towards the Stars

Most computer systems in the world, micro to mainframe, have at least one STAR TREK-type game in their library. The limited memory on the basic ZX80 precludes all but the simplest versions of this old favourite. However, study of the following program will teach you some of the fundamentals of star games, and will give you a core to build on when you buy extra K for your machine.

```

1  RANDOMISE
2  PRINT ,(shift A)
   TIMEWARP(shift A)
3  PRINT
4  PRINT
5  LET G = 10
6  LET H = 0
7  PRINT ,(NAME?)
8  INPUT N$
9  PRINT (TREASURE)
10 INPUT T$
11 PRINT ,(ENEMY 1?)
12 INPUT E$
13 PRINT,(ENEMY 2?)
14 INPUT F$
15 INPUT Z$
16 CLS
17 LET J = RND(2)
18 IF J = 1 THEN LET L$ =
   E$
19 IF J = 2 THEN LET L$ =
   F$
20 IF G < 1 THEN GOTO 50
21 PRINT ,(SHIELD
   THICKNESS: ;G
22 INPUT Q$
23 LET H = H + 1
24 PRINT
25 PRINT ,(TIME
   LEFT: ;17-H
26 PRINT (DANGER. ;L$;
   (AHEAD)
27 INPUT B$
28 GOSUB 55
29 IF H = 17 THEN GOTO
   52
30 LET K = RND(2)
31 IF K = 2 THEN GOTO 35
32 LET G = G - 1
33 PRINT (THE ;L$; GOT
   YOU, ;N$
34 GOTO 15
35 LET G = G + 1

```

```

36 PRINT "YOU
   DESTROYED THE ";L$, N$
37 INPUT Z$
38 CLS
39 PRINT "YOU ARE
   CLOSER TO THE ";T$
40 INPUT D$
41 CLS
42 LET M = RND(5)
43 IF M < 5 THEN GOTO 17
44 GOSUB 55
45 IF G > 0 THEN PRINT
   "CONGRATU-
   LATIONS, ";N$
46 PRINT "YOU GOT
   THE ";T$
47 PRINT "OUT OF THE
   TIMEWARP"
48 PRINT "WITH ";17 -
   H;" TIME UNITS SPARE"
49 STOP
50 PRINT "GAME OVER.
   YOU ARE DEAD"
51 STOP
52 PRINT "TIMEWARP HAS
   IMPLoded"
53 PRINT ",YOU HAVE
   FAILED"
54 STOP
55 FOR W = 1 TO RND(300)
56 CLS
57 NEXT W
58 RETURN

```

If this program followed a shorter version of the next program we will look at, and if the value of H could be used to print the "sector of the galaxy" we were in (i.e. IF H > 3 AND H < 7 THEN PRINT "YOU ARE IN SIRIUS SECTOR") we would be well on the way to creating a much better "STAR TREK". If you decide to add extra K, you could start by modifying this program. To give you room to move, multiply each line number by 10.

In this program, for the first time, we access the ZX80's frame counter. This allows us to add a time dimension to programs.

Input the following program, and run it, keeping in mind that your reactions to commands have a time limit to them. If you exceed the time, you'll get an ABORT MISSION display.

```

1 PRINT,CHR$(128);"BLAST
   OFF";CHR$(128)
2 PRINT "TO START TAKE-
   OFF ROUTINE"
3 PRINT, "PRESS
   NEWLINE"
4 INPUT K$
5 RANDOMISE
6 FOR J = 1 TO 20
7 LET H = RND(10)
8 FOR U = 1 TO
   5*RND(100)
9 CLS
10 NEXT U
11 FOR S = 1 TO 2*H
12 PRINT
13 NEXT S
14 POKE 16414,0
15 POKE 16415,0
16 GOSUB 10*RND(4) + 120
17 LET Q = PEEK (16414)
18 LET N = PEEK (16415)
19 IF J = 20 AND
   (N*256 + Q)*10 < 1500
   THEN GOTO 170
20 IF (N*256 + Q)*10 < 1500
   THEN NEXT J
21 CLS
22 PRINT CHR$(128);"(shift
   A) ABORT MISSION AT ";
   (N*256 + Q)*10
23 LET N = N + 1

```

```

24 GOTO 22
130 PRINT ,CHR$(128);"SET
    FUEL TO ";
    (H*20/3)*10*H - H
135 INPUT K
137 IF K = (H*20/3)*10*H -
    H THEN RETURN
138 GOTO 21
140 LET A$ = CHR$(H + 37)
142 PRINT "(shift D four
    times)INPUT AIR
    PRESSURE VALVE ";A$
145 INPUT B$
147 IF B$ = A$ THEN
    RETURN
148 GOTO 21
150 PRINT "SIGNAL TO
    ALIEN ";CHR$(168 + J);
    CHR$(H*19);CHR$
    (166 + 2*H);" WITH ";
    CHR$(212);CHR$
    (48 + H);CHR$(212)
155 INPUT C$
157 IF C$ = CHR$(H + 48)
    THEN RETURN
158 GOTO 21
160 PRINT "(shift S two
    times)GIVE IDENTITY
    NUMBER ";(100*H -
    H/3)+J
165 INPUT Z
167 IF Z = (100*H - H/3) + J
    THEN RETURN
168 GOTO 21
170 CLS
180 PRINT "(shift W two
    times)WE HAVE LIFT
    OFF";
190 GOTO 180

```

This program uses a number of ideas we've been introduced to recently. Line 1 uses the very useful CHR\$(128) which is the inverse space, a very solid black square. Lines 7 to 10 introduce a random delay while clearing the screen. As you run this program, you'll see that the commands appear at different points on the screen. They are placed here by the random number of blank print statements above them, generated by lines 11 to 13. The timer starts with the POKE statements (lines 14 and 15) where the count is set to zero, and the ZX80 keeps counting TV frames until you get back to the PEEK statements (lines 17 and 18) via the randomly selected (line 16) subroutine. The subroutine will only RETURN you to lines 17 and 18 if you have done what was required within the subroutine. Once you are back at the PEEK statements, the ZX80 checks the value of a manipulation of Q (line 17) and N (line 18) against the maximum time limit. You may find this game gets a little boring after you've played it a number of times. If this happens, reduce the 1500 in lines 19 and 20 to, say, 1000. Another idea, which is used in the following program — FRENZY — is to reduce the time throughout the game (by subtracting, for example, an arithmetic manipulation of the value of the counter of the master FOR/NEXT loop). You could do this by changing line 20 to read IF (N\*256 + Q) < 1500 - 25\*J THEN NEXT J, and changing line 19 to set the same restriction. You could then, perhaps, consider adding a print statement to tell you how much time you have to complete the instructions in the subroutine, and another print line

which tells you how long it took you to actually complete it. The next program, which can be made as heart-stopping as you like, uses these features.

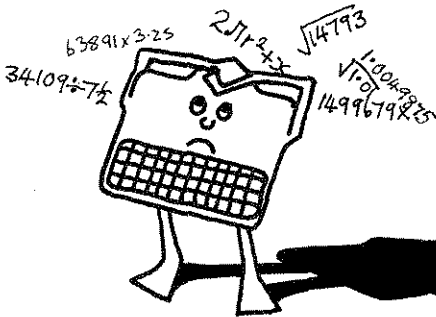
```

1  RANDOMISE
2  PRINT,CHR$(128);
   " FRENZY ";CHR$(128)
3  PRINT
4  PRINT "TO RISK SANITY,
   PRESS N/L"
5  INPUT A$
6  CLEAR
7  LET G = 5 + RND(15)
8  FOR J = 1 TO G
9  FOR D = 1 TO 50
   *(20 + RND(50))/J
10 CLS
11 NEXT D
12 IF J > 1 THEN PRINT,
   "TIME LAST GO:— ";M
13 PRINT
14 PRINT "TEST NO. ";J;
   " OUT OF ";G
15 PRINT
16 PRINT "YOU HAVE";
   CHR$(128);2*(70 - 7*J);
   CHR$(128);" NUT
   SECONDS"
17 FOR W = 1 TO RND(11)
18 PRINT
19 NEXT W
20 LET F =
   (2*G + 3*J/2)*J*J -
   RND(9)
21 POKE 16414, 0
22 POKE 16415, 0
23 PRINT "OK DUM DUM,
   COPY THIS NUMBER"
24 PRINT
25 LET X = RND(4)
26 IF X = 1 THEN PRINT ,
27 IF X = 2 THEN PRINT ,,
28 IF X = 3 THEN PRINT ,,
29 PRINT F
30 INPUT K
31 LET U = PEEK(16414)
32 LET Z = PEEK (16415)
33 LET M = 256*Z + U
34 LET K = F AND
   M < 2*(170 - 7*J) THEN
   NEXT J
35 IF NOT K = F THEN
   GOTO 49
36 IF M > 2*(170 - 7*J)
   THEN GOTO 45
37 CLS
38 PRINT "YOUR SANITY
   RATING"
39 PRINT "IS ",
   M*J/8 + RND(7)
40 PRINT
41 PRINT "COULD YOU
   STAND IT AGAIN?"
42 INPUT H$
43 IF NOT H$ = "NO"
   THEN GOTO 31
44 STOP
45 CLS
46 PRINT "YOU ARE FAR.
   TOO SLOW"
47 PRINT
48 GOTO 38
49 CLS
50 PRINT "YOU CANNOT
   EVEN COPY NUMBERS"
51 PRINT
52 GOTO 38

```

There are a few extra ideas in this program which are worth pointing out. Firstly, look at lines 25 to 28. These decide how far across the line the number F will be printed (if X = 4 the number is printed hard against the left because there is no instruction for X = 4). The second point worth noting is that the manner in which the answer to COULD

YOU STAND IT AGAIN? is treated. Any answer at all, except NO, will be interpreted as an instruction to go back for a second game. So if your smart friends input YAY, YEAH, Y, YESSIR, OK or whatever, the ZX80 will "understand" that YES (actually, NOT NO) is meant.



## Doing it in Your Head

The next two programs do not have timers, but could well be adapted to have a real time limitation if you like.

```

10 PRINT "UNICORNS AND          180 PRINT
   GRIFFINS"                    "UNICORN";CHR$(212);
20 LET G = 5 + RND(5)           "S NUMBER IS ";F
30 PRINT                          190 PRINT "WHAT DOES
40 PRINT "DEGREE OF             GRIFFIN ADD TO"
   DIFFICULTY (1 TO 5)?"        200 PRINT "MAKE
50 INPUT Q                       UNICORN";CHR$(212);
60 IF Q < 1 OR Q > 5 THEN        "S NUMBER = ";Z;"?"
   GOTO 40                       210 INPUT K
70 PRINT "PRESS NEWLINE        220 IF K + F = Z THEN
   TO START"                     NEXT J
80 INPUT A$                       230 PRINT
90 FOR J = 1 TO G                240 LET T = (F*RND(100))+F
100 FOR D = 1 TO RND(300)         250 IF J = 1 THEN LET
110 CLS                           T = 0
120 NEXT D                       260 PRINT "SCORE FOR
130 FOR W = 1 TO RND(6)           THAT ROUND WAS ";T
140 PRINT                         270 IF NOT J = G AND NOT
150 NEXT W                       K + F = Z THEN
160 LET F = 2*G + 3*J/2          GOSUB 330
170 LET Z = Q*(RND(15) +        280 IF J = G THEN GOSUB
   J*10)                          360

```

```

290 PRINT "DO YOU WANT
    TO TACKLE THE"
300 PRINT "UNICORN
    AGAIN?"
310 INPUT H$
320 IF NOT H$ = "NO"
    THEN GOTO 20
330 IF NOT K + F = Z
    PRINT "THE UNICORN
    BEAT YOU"
340 RETURN
350 PRINT
360 PRINT "YOU HAVE
    BEATEN THE UNICORN"
370 PRINT "YOUR IQ IS ";
    T*J + J
380 RETURN

```

This program introduces an idea which you can use in many games — the “degree of difficulty”. Generally, the “degree” can be used directly, to multiply or divide something, or to be added to or taken away from the limits on a FOR/NEXT loop. In other games, you might have to add lines like (if, say A was the degree): IF A = 1 THEN LET C = 200; or IF A > 7 THEN GOSUB 90. Look back at some of the earlier programs in this book, and work out ways of modifying them in the light of later things you have learned.

The “degree of difficulty” can easily be worked into the following program. However, as it becomes more and more difficult already as it proceeds, it might be better to make it a little easier before adding the option of increasing the difficulty. Line 280 is the one to modify to make the game simpler, and it is here that the “degree” factor can be added.

```

10 PRINT "ECHO
    CHAMBER"
20 LET Z = RND(8)
30 FOR G = 1 TO Z + 5
40 LET D = RND(4)
50 IF D = 1 THEN LET
    A$ = "KIDDO"
60 IF D = 2 THEN LET
    A$ = "SMART ONE"
70 IF D = 3 THEN LET
    A$ = "GENIUS"
80 IF D = 4 THEN LET
    A$ = "COMPUTER
    FREAK"
90 LET K = RND(32767)
100 PRINT
110 PRINT
120 PRINT "TRY NO. ";G;
    " OUT OF ";Z + 5
130 PRINT
140 PRINT "THE NUMBER
    YOU HAVE TO"
150 PRINT "REMEMBER, ";
    A$
160 PRINT "IS ";K
170 PRINT
180 PRINT "WHEN YOU ARE
    SURE YOU"
190 PRINT "CAN DO THIS,
    PRESS N/L"
200 INPUT B$
210 IF NOT B$ = "" THEN
    STOP
220 GOSUB 280
230 PRINT "OK, ";A$;
    " WHAT WAS"
240 PRINT "THE NUMBER?"
250 INPUT H
260 IF H = K THEN GOTO
    320
270 IF NOT H = K THEN
    GOTO 410
280 FOR J = 1 TO 300*G
290 CLS

```

```

300 NEXT J
310 RETURN
320 CLS
330 PRINT "YAY, ";A$," YOU
    GOT IT","RIGHT"
340 PRINT "WHEN READY
    FOR"
350 PRINT "NEXT ONE,
    PRESS N/L"
360 INPUT C$
370 IF NOT C$ = " " THEN
    STOP
380 CLS
390 IF G = Z + 5 THEN
    GOTO 440
400 NEXT G
410 CLS
420 PRINT "YOU BLEW
    IT, ";A$
430 STOP
440 PRINT "YOU SURE HAVE
    A PRODIGIOUS"
450 PRINT "MEMORY, ";A$
460 STOP

```

In this program, which puts a number in the range from one to 32767 (the ZX80's upper limit) on the screen, and then asks you to remember it for a time period which gets longer with each new number, note that lines 40 to 80 control what A\$ will be in each run through the master FOR/NEXT loop. A similar approach was used in SNAKES AND LADDERS and ROADRACE. Line 280, as you can easily see, determines how long the delay loop will be. By the last few shots in a round, the delay seems interminable. Have a look at the listing for line 330, noting the comma between the words IT and RIGHT. If the comma was not here, the word RIGHT would scroll around, half on one line and half on the other, when D equals 4. Can you see why?



Because the TV screen is not memory mapped, genuine moving graphics are unfortunately out of the question for the ZX80. But you can produce a sort of animated picture as this next program shows. Input it as listed, then try and work out a version that has two people running in a race, with "moving legs".

```

1 LET A$ = "(shift A)1(shift A)93
2 LET B$ = "(shift A)2(shift A)82

```



```

3 LET C$ = "O(shift G)O"
4 LET C = 0
5 LET D = 0
6 RANDOMISE
7 LET W = RND(4)
8 PRINT "GRAND PRIX"
9 FOR J = 1 TO 70
10 IF J > 1 THEN PRINT
    "AFTER LAP ";J - 1
11 PRINT
12 PRINT "POINTS: ";
    A$;"-";C;" ";B$;"-";D
13 IF C > 28 AND D > 28
    THEN GOTO 54
14 IF C > 28 OR D > 28
    THEN GOTO 44
15 PRINT
16 IF J = 1 THEN PRINT
    "TO START RACE ";
17 PRINT "PRESS NEWLINE"
18 INPUT U$
19 CLS
20 PRINT "...FINISH(space
    shift Q)"
21 LET C =
    C - 1 + RND(3)
22 GOSUB 35
23 PRINT A$
24 IF 2*(J/2) = J AND NOT
    J = RND(25) THEN LET
    N = RND(W)
25 GOSUB 35
26 PRINT C$
27 PRINT
28 PRINT
29 LET D =
    D - 1 + RND(3)
30 GOSUB 39
31 PRINT B$
32 GOSUB 39
33 PRINT C$
34 NEXT J
35 FOR T = 1 TO C
36 PRINT "(singlespace)";
37 NEXT T
38 RETURN
39 FOR Z = 1 TO D
40 PRINT "(singlespace)";
41 NEXT Z
42 RETURN
43 STOP
44 CLS
45 PRINT
46 PRINT
47 PRINT
48 PRINT "DRIVER OF ";
49 IF C > D THEN PRINT A$
50 IF D > C THEN PRINT B$
51 PRINT "WINS THE
    GRAND PRIX"
52 PRINT "WITH ";ABS
    (C - D)*1000/J;
    "(singlespace) POINTS"
53 STOP
54 CLS
55 IF C = D THEN PRINT
    A$;" DEAD
    HEAT ";B$;"(singlespace)";
56 IF NOT C = D THEN
    GOTO 44
57 GOTO 55

```

In this program the "movement" comes from the two subroutines (starting at lines 35 and 39) which print a number of blank spaces before the "car" is printed. If you wanted a person with waving arms, a random number could determine which "arm subroutine" was printed. The two people running a race program could be written by combining the idea used in GRAND PRIX for moving the cars across, with a sub-subroutine to determine the position of the figures' legs.

If you have more than 1K memory, you could write a GRAND PRIX with four cars, and include a number of hazards (such as SMASH INTO

SIDE WALL, OUT OF PETROL, or GEAR BOX BLOWS UP) which could either remove a car completely from the race, or simply take it out for a while, and then making sure it had to start again. Another variation on the above ideas is to let the cars, people or whatever move from left to right on the screen, and then move back. It should not be too hard for you to work out how this can be done.

In this next game, a fly (heavily disguised as an 'X') has to land on a sugar cube (shift A). You input the co-ordinates you want the fly to follow — positive is down, followed by NEWLINE, left is negative. You have to estimate the co-ordinates you input. This program gives a pretty good imitation of moving graphics.

```

1  PRINT "MAGIC FLY"
2  RANDOMISE
3  LET V = 1
4  LET T = 0
5  LET F = RND(29)
6  LET Y = RND(31)
7  LET X = 16
8  GOSUB 37
9  LET T = T + 1
10 GOSUB 41
11 IF X < 2 THEN GOTO 13
12 GOTO 46
13 FOR K = 1 TO F
14 PRINT "(onespace)";
15 NEXT K
16 PRINT "(shift A)"
17 IF X < 2 AND Y = F
   THEN STOP
18 INPUT A
19 IF A > 5 THEN LET A =
   5
20 INPUT P
21 IF P > 5 THEN LET P = 5
22 LET B = RND(3)
23 CLS
24 IF B = 1 THEN LET M =
   X - A + 1
25 IF B = 2 THEN LET M =
   X - A - 1
26 IF B = 3 THEN LET M =
   X - A
27 LET V = V + X - M
28 LET X = M
29 IF X < 1 THEN GOTO 60
30 LET D = RND(3)
31 IF D = 1 THEN LET Q =
   P + 1
32 IF D = 2 THEN LET Q =
   P - 1
33 IF D = 3 THEN LET Q =
   P
34 LET Y = Y + Q
35 IF Y = F AND X < 2
   THEN GOTO 50
36 GOTO 8
37 FOR H = 1 TO V
38 PRINT
39 NEXT H
40 RETURN
41 FOR L = 1 TO Y
42 PRINT "(onespace)";
43 NEXT L
44 PRINT "X"
45 RETURN
46 FOR J = 0 TO X
47 PRINT
48 NEXT J
49 GOTO 13
50 PRINT "WELL DONE,
   SUGAR"
51 PRINT
   "YOU";CHR$(212);"VE
   LANDED ON THE CUBE"
52 PRINT "AND IT TOOK
   YOU ";T;" SECONDS"
53 GOTO 8
54 PRINT "YOU MISSED
   THE SUGAR"
55 PRINT "TO FLY AGAIN
   TYPE (Y)"

```

```

56 INPUT F$
57 CLS
58 IF F$ = "Y" THEN GOTO
  2
59 STOP
60 GOSUB 37
61 FOR K = 1 TO F
62 PRINT "(onespace)";
63 NEXT K
64 IF Y < F THEN PRINT
  "X(onespace shift A)"
65 IF F < Y OR F = Y THEN
  PRINT "(shift A
  onespace)X"
66 PRINT ""
67 GOTO 54

```

The last game in this section is included because it makes a very effective use of the "moving graphics" idea, and also because it seems that just as every computer in the world hankers to be Captain Kirk, so every computer operator wants to land capsules, modules, LEMs or whatever on the moon. So here's a program to do it via the ZX80. But be warned, it is fairly difficult to land safely and demands a lot of patience. Once you've mastered the art of landing, change line 2 as follows, just to make it even harder to land successfully: 2 IF H < 30 AND V < 10 AND V > -7 THEN IF ABS(Z - M) < 4 THEN GOTO 37.

```

1 GOTO 42
2 IF H < 50 AND V < 20
  AND V > -15 THEN IF
  ABS(Z - M) < 5 THEN
  GOTO 37
3 IF H > 1750 THEN GOTO
  40
4 RETURN
5 CLS
6 LET T = T + 4 +
  RND(2)
7 LET V = V + A*A*A -
  12 - RND(3)
8 LET H = H + V - 20 +
  RND(10)
9 LET F = F - (ABS(A) +
  ABS(B/5) * RND(6))
10 GOSUB 2
11 IF H < 20 OR F < 5 THEN
  GOTO 35
12 LET U = H/100
13 FOR J = 1 TO 17 - U
14 PRINT
15 NEXT J
16 LET Z = Z + B/2 + 2 -
  RND(3)
17 FOR J = 1 TO Z
18 PRINT "(1 space)";
19 NEXT J
20 PRINT "(shift F shift G
  shift D)"
21 FOR J = 1 TO U
22 PRINT
23 NEXT J
24 FOR J = 1 TO M - 1
25 PRINT "(shift W)";
26 NEXT J
27 PRINT "(shift S shift W
  shift T shift W shift S)"
28 PRINT "VEL: ";V;" XE23
  HEIGHT: ";H;" FUEL: ";F;
  " TIME: ";T
29 PRINT 13*A;" THRUST? ";
30 INPUT A
31 PRINT A;" DRIFT?"
32 INPUT B
33 LET A = A/13
34 GOTO 5
35 PRINT "CRASH (shift R
  twice space) SPEED ";
  ABS(V);"(space)";
36 GOTO 35
37 CLS

```

38	PRINT "SUCCESSFUL LANDING (shift E) RATING ";(30 - ABS(V)) *100 + V	42	LET H = 1750
39	GOTO 38	43	LET F = 827 + RND(50)
40	PRINT "YOU HAVE REACHED ESCAPE VELOCITY"	44	LET T = 0
41	GOTO 40	45	LET Z = RND(15)
		46	LET A = 1
		47	LET B = 0
		48	LET M = RND(19)
		49	LET V = 0
		50	GOTO 6

---

## ZX80 Active Display

Ron Bissell and Ken Macdonald, of Solihull, have found a way of getting moving graphics on the ZX80. They call their routine "the amazing active display", and amazing it is. Input the following program, and RUN it a few times so you can see how it works.

After you've RUN it as listed, delete lines 210 to 820 and insert your own program, with a GOSUB 900 whenever you want the ZX80 to display something and hold this display. Note that one of your lines must specify time (T). Start off with LET T = 150 and then experiment with other values. With LET T = 255 you see nothing at all, while LET T = 0 (as in line 350 of the original program) holds the display for a very long time. LET T = 250 is a very, very rapid change; so fast that your TV screen may not stabilise between changes.

There is no reason why you cannot let T change during a game. Try and write a program which prints out a line of, say, shift A, then adds 10 to the value of T, then prints out another line. You'll need to limit the number of lines of shift A's you want to print, as well as put an upper limit on the value of T. (Hint: Put LET T = T + 10 and IF T = 300 THEN LET T = 0 somewhere in the 900 subroutine). Once you've experimented with the active display, you might like to read the explanation on it given by the program authors. Note that the copyright of the active display belongs to R Bissell and K Macdonald, and like all the programs in this book, cannot be offered for sale, or used as part of a product offered for sale.

10	LET A = 17270	221E407CDE00
20	LET S = A	C823DBFE3E3832234006
30	LET M\$ =	5E10FED3FE3EEC0
	"CDE006CDC205012001D9	6192A0C40CBFCCDAD
	CDC2051803CDAD010	013EF5042BFD352318CA"
	60810FE2A1E4023	

```

40 LET H = CODE (M$)
50 IF H = 1 THEN GOTO
200
60 LET M$ = TL$(M$)
70 LET L = CODE (M$)
80 POKE A, 16*
(H - 28)+L - 28
90 LET M$ = TL$(M$)
100 LET A = A + 1
110 GOTO 40
200 LT C = 16414
210 LET R = 900
220 LET Q = 800
230 LET T$ = "THE"
240 LET U$ = "AMAZING"
250 LET W$ = "ACTIVE"
260 LET X$ = "DISPLAY"
270 LET T = 150
280 PRINT T$
290 GOSUB R
300 PRINT U$
310 GOSUB R
320 PRINT W$
330 GOSUB R
340 PRINT X$
350 LET T = 0
360 GOSUB R
370 CLS
380 LET T = 150
390 GOSUB Q
400 PRINT T$
410 GOSUB Q
420 PRINT U$
430 GOSUB Q
440 PRINT W$
450 GOSUB Q
460 PRINT X$
470 LET T = 0
480 GOSUB Q
490 CLS
500 GOTO 270
800 PRINT
810 PRINT
820 PRINT
900 POKE C,T
910 POKE C + 1, 255
920 LET X = USR (S)
930 RETURN

```

In the program listing lines 10-110 and 920 are the HEX LOADER i.e. a BASIC routine which can be loaded from cassette tape which will POKE a machine code routine into memory and pass execution to it. The machine code is held as a string of successive pairs of hexadecimal digits, M\$ in line 30. The usual method of entering machine code by POKING individual decimal values would, in this case, waste about 300 bytes of memory.

In line 10 variable A is the address to start loading and S in line 20 is the address at which execution starts in statement 920. The value of A must be somewhere between the top of the Display File and the likely range of the Stack Pointer, 17270 in this case was found to be an optimum value but the Active Display machine code routine is RELOCATABLE and may be positioned at higher addresses if more memory than the minimum 1K is available.

If this method of entering machine code is to be used for other programs which require non-contiguous areas to be loaded, then make lines 40-110 a sub-routine and use a different value of A and MA for each separate area of contiguous code. Note that machine code programs can readily be edited using the BASIC line editor.

Line 200 is the address of the System Variable used as the Frame Counter. In normal ZX80 operation this increments by 1 every time a complete TV frame is displayed, then re-cycles, but in the case of the

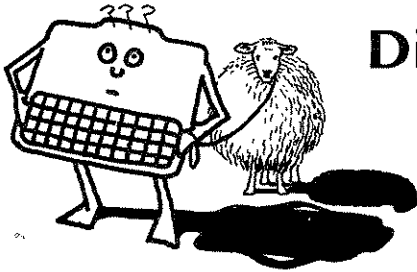
Active Display, the display routine is terminated when the count overflows back to zero and control passes back to BASIC for further processing. Lines 900 and 910 pre-set this Frame Counter for a particular display time. If the 2 byte unsigned binary number held here at locations 16414/5 is 65536 i.e. 255,255 then a return to BASIC will be made without displaying. Each unit subtracted from 65536 will cause a display persist for another 1/50th of a second so periods up to approximately 10 minutes are possible. Note that very short periods may not give the TV set time to regain synchronisation lock and result in a messy picture.

After the Frame Counter is pre-set, line 920 causes a jump to the machine code sub-routine which duplicates selected regions of the Sinclair ROM but with the added return on Frame Counter overflow. The keyboard is totally inoperative during Active Display, so to stop the program *hold* BREAK until control returns to BASIC. Remember to *SAVE* your program on tape *before* *RUN*ning, since an error in machine code will undoubtedly cause a system crash.

---

## Let the Games Begin

Now follows a series of games, all of which will fit within the 1K supplied with your machine. The games use ideas introduced in earlier sections of the book, and are here mainly for playing. You will probably learn something by studying the listings, but the main "teaching" is now behind you.



## Did He Who Made the Lamb Make the UFO

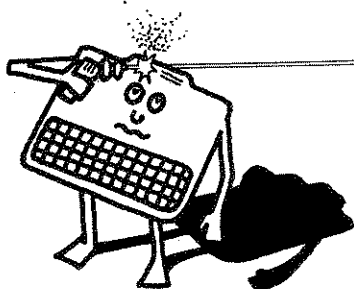
This game, which produces a remarkably attractive display, uses a subroutine to "draw" a flying saucer before each shot. Each saucer is a little different from the one that precedes it. The subroutine to generate the saucer is also used in the program LASER ROULETTE.

```
1 PRINT "SLAUGHTER"
2 LET W = 5 + RND(5)
3 FOR K = 1 TO W
4 PRINT
5 PRINT ,(shift W shift A
  shift W)
6 GOSUB 40
7 PRINT " "
8 FOR S = 1 TO 2
9 PRINT ,
10 FOR Z = 1 TO 18
11 PRINT "{shift S}";
12 NEXT Z
13 PRINT " "
14 NEXT S
15 GOSUB 40
16 PRINT "TALLY: ";K - 1;
  " OUT OF ";W
17 LET J = RND(32000)
18 POKE 16414,0
19 POKE 16415,0
20 PRINT
21 PRINT "FOR SHIP ";K;
  ", FIRE"
22 PRINT "TO VECTOR ";J;
  " IN ";500 - 10*
  (K - 1);" STARSECS"
23 INPUT M
24 LET A = PEEK (16414)
25 LET B = PEEK (16415)
26 LET Q = B*256 + A
27 IF Q > 500 - 10*K OR
  NOT M = J THEN GOTO
  50
28 GOSUB 46
29 PRINT "TIME FOR LAST
  KILL: ";Q
30 NEXT K
31 PRINT
32 PRINT ,"YOU
  SLAUGHTERED"
33 PRINT W;" SIRIUS
  SHIPS"
34 PRINT
35 PRINT ,"ANOTHER
  MISSION?"
36 INPUT B$
37 CLS
```

```

38 IF NOT B$ = "NO"
   THEN GOTO 2
39 STOP
40 PRINT ,(2 spaces);
41 LET G = RND(11)
42 FOR Z = 1 TO 14
43 PRINT CHR$(127 + G);
44 NEXT Z
45 RETURN
49 PRINT
50 CLS
51 PRINT
52 PRINT "SHIP ";K;" GOT
YOU"
53 GOTO 51

```



## Laser Roulette

```

1 PRINT "LASER
ROULETTE"
2 RANDOMISE
3 FOR K = 1 TO 10
4 PRINT
5 PRINT
6 PRINT
7 PRINT ,(shift W shift A
shift W)
8 GOSUB 47
9 PRINT " "
10 FOR S = 1 TO 2
11 PRINT ,
12 FOR Z = 1 TO 18
13 PRINT ,(shift S);
14 NEXT Z
15 PRINT " "
16 NEXT S
17 GOSUB 47
18 PRINT
19 PRINT
20 PRINT ,(PRESS N/L"
21 PRINT ,(FOR LASER"
22 PRINT ,(SHOT ";K
23 INPUT A$
24 GOSUB 53
25 LET J = RND(7)
26 IF J = 4 THEN GOTO 41
27 PRINT
28 PRINT ,(SAFE...THIS
TIME"
29 PRINT
30 IF NOT K = 1 THEN
GOSUB 57
31 NEXT K
32 PRINT
33 PRINT ,(YOU ESCAPED
FROM THE "
34 PRINT ,(REACTION
CHAMBER"
35 PRINT
36 PRINT "ANOTHER
ESCAPE?"
37 INPUT B$

```

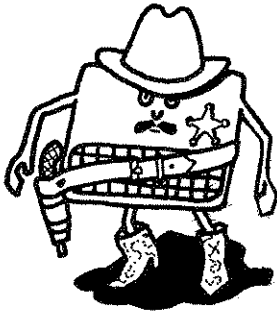


```

38 CLS
39 IF NOT B$ = "NO"
  THEN GOTO 3
40 STOP
41 FOR D = 1 TO 32
42 PRINT CHR$(128);
43 PRINT "DEAD...";
44 NEXT D
45 POKE 16421, 24
46 STOP
47 PRINT ,(2 spaces);
48 LET G = RND(11)
49 FOR Z = 1 TO 14
50 PRINT CHR$(G + 127)
51 NEXT Z
52 RETURN
53 FOR Y = 1 TO K*K*10
54 CLS
55 NEXT Y
56 RETURN
57 PRINT "YOU HAVE
  BLASTED ";K;" TENTHS"
58 PRINT "OF THE
  SAUCERS REACTOR
  WALL"
59 RETURN

```

In this game, the player is supposed to be trapped inside a flying saucer's reactor (!) with a laser pistol. If he can fire 10 shots he can get out, but every time he fires he runs the risk of blowing the saucer — and himself — up. This is an example of how a simple program (RUSSIAN ROULETTE) can be dressed up and made much more interesting.



## High Noon at the Old Intergalactic

In this game, you have to outdraw a nimble-fingered alien. The screen goes blank for a random length of time, then comes back, showing the alien (he moves across during the game). You must hit NEWLINE as fast as you can. If you do so quickly enough the alien dies, falling apart as splendidly as did the CHOPPER in an earlier

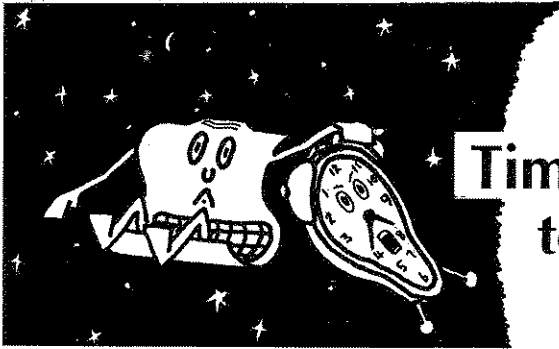
game. Once you've mastered this game, change line 38 to "IF D < 200 + RND(300) THEN GOTO 49", and you'll find you have to learn to beat the alien all over again.

```

1   GOTO 7
2   PRINT A$;"'"
3   PRINT A$;"**'"
4   PRINT A$;"(shift A)"
5   PRINT A$;"(SHIFT S, shift
   T)"
6   RETURN
7   LET T = 0
8   PRINT "YOU HAVE 6
   SHOTS"
9   PRINT "TO OUTDRAW
   THE ALIEN"
10  PRINT "BEFORE IT
   DESTROYS YOUR BASE"
11  LET A$ = "(6 spaces)"
12  FOR J = 1 TO 14
13  PRINT
14  NEXT J
15  GOSUB 2
16  PRINT "SHOT ";T + 1
17  INPUT H$
18  CLS
19  FOR C = 1 TO 14
20  PRINT
21  NEXT C
22  IF T = 1 THEN LET A$ =
   "(9 spaces)"
23  IF T = 2 THEN LET A$ =
   "(12 spaces)"
24  IF T = 3 THEN LET A$ =
   "(15 spaces)"
25  IF T = 4 THEN LET A$ =
   "(18 spaces)"
26  IF T = 5 THEN LET A$ =
   "(21 spaces)"
27  GOSUB 2
28  FOR A = 1 TO
   10*RND(200)
29  NEXT A
30  POKE 16414, 0
31  POKE 16415, 0
32  INPUT X$
33  LET T = T + 1
34  CLS
35  LET B = PEEK (16414)
36  LET C = PEEK (16415)
37  LET D = ((C*200) +
   (B*2))*20
38  IF D < 500 THEN GOTO
   49
39  IF T = 6 THEN SAVE
40  PRINT,"MISSED"
41  FOR F = 1 TO 12
42  PRINT
43  NEXT F
44  GOSUB 2
45  INPUT G$
46  CLS
47  IF NOT G$ = "NO"
   THEN GOTO 12
48  STOP
49  PRINT "YOU HIT THE
   ALIEN"
50  PRINT "AND SAVED
   YOUR BASE"
51  FOR D = 1 TO 15
52  PRINT
53  NEXT D
54  PRINT A$;"(2 spaces)*"
55  PRINT A$;"(3 spaces)*"
56  PRINT A$;"(shift G shift
   T)"
57  PRINT A$;"(2 spaces shift
   D shift R 2 spaces shift
   G)"
58  STOP

```

A novel feature of this game is line 39, which puts the computer into the SAVE mode if you lose. A far more frustrating punishment for a loss is to have to computer erase the program. Can you work out what line 39 would have to be to make the program disappear completely?



## Time Warped to Space

You will recall that, earlier in this book, a program under the imaginative name of TIME WARP was listed. The next program uses the same basic program to produce something a little more down to earth. All of the programs in this book can (and should) be developed by you in whatever direction you prefer. Only by doing this will you develop your own programming skills. Anyway, here is one way TIME WARP can be warped.

```

1   LET G = 9
2   PRINT "CAVE MASTER"
3   GOSUB 57
4   LET H = 0
5   LET E$ = "CRAZED
   WIZARD"
6   LET F$ = "WICKED
   WITCH"
7   INPUT K$
8   GOSUB 53
9   LET J = RND(2)
10  IF J = 1 THEN LET L$ =
   E$
11  IF J = 2 THEN LET L$ =
   F$
12  IF G < 1 THEN GOTO 47
13  GOSUB 57
14  PRINT ,CHR$(128 + G);
   " AURA TONE ";G
15  LET H = H + 1
16  IF H = 17 THEN GOTO
   50
17  GOSUB 57
18  PRINT "LEVEL OF
   MAGIC: ";17 - H
19  GOSUB 57
20  PRINT "HORRORS, ";L$;
   " AHEAD"
21  INPUT B$
22  GOSUB 53
23  IF H = 17 THEN GOTO
   50
24  LET K = RND(2)
25  IF K = 2 THEN GOTO 31
26  LET G = G - 2
27  GOSUB 57
28  PRINT "THE ";L$;
   " ZONKED YOU"
29  INPUT A$
30  GOTO 9
31  LET G = G + 1
32  PRINT "YOU ZAPPED
   THE ";L$
33  INPUT D$
34  GOSUB 53
35  GOSUB 57

```

```

36 PRINT "THE FOOLS          49 STOP
   GOLD IS WITHIN"         50 GOSUB 57
37 PRINT,"YOUR GRASP"     51 PRINT,"YOU TOOK TOO
39 INPUT G$                LONG"
40 LET M = RND(6)          52 GOTO 48
41 IF M < 6 THEN GOTO 9    53 FOR W = 1 TO RND
42 GOSUB 53                (400)
43 GOSUB 57                54 CLS
44 IF G > 0 THEN PRINT,   55 NEXT W
   "YOU DID IT"           56 RETURN
45 PRINT "WITH ";17 - H;  57 PRINT
   " MAGIC SPELLS LEFT"  58 PRINT
46 STOP                    59 PRINT
47 GOSUB 57                60 PRINT
48 PRINT,"DEATH COMES     61 RETURN
   TO US ALL"

```



## Turning the Tables

There are many, many programs in which the computer thinks of a number, and the human player has to guess it. There are two of them in this book. This next program, written by Trevor Sharples, turns the tables.

```

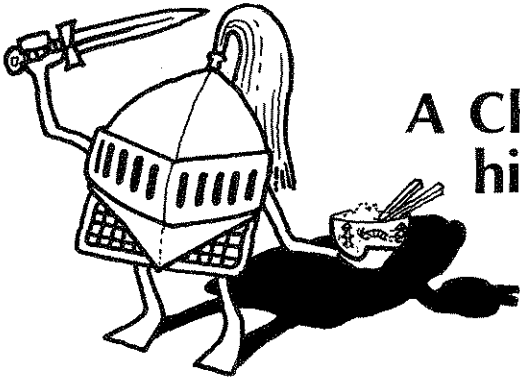
2 PRINT,"MYSTIC"          9 PRINT
4 PRINT                   10 PRINT "PRESS NEWLINE
6 PRINT "THINK OF A      TO PLAY"
   NUMBER BETWEEN"       12 INPUT A$
8 PRINT "ONE AND 100     14 GOSUB 86
   AND I WILL GUESS IT"  16 LET T = 0

```

```

18 LET Z = 100
20 LET Y = 1
22 LET X = RND(100)
24 PRINT "I GUESS ";X
26 PRINT
28 PRINT
30 PRINT,"RIGHT (R) OR
WRONG (W)?"
32 LET T = T + 1
34 INPUT B$
36 IF B$ = "R" THEN
GOTO 66
38 GOSUB 86
40 PRINT "MY GUESS
WAS ";X
42 PRINT
44 PRINT,"HIGHER (H) OR
LOWER (L)?"
46 INPUT C$
48 GOSUB 86
50 IF C$ = "L" THEN GOTO
62
52 LET Y = X
54 LET S = Z - Y
56 LET X = Y + RND(S)
58 IF X = S + Y THEN
GOTO 56
60 GOTO 24
62 LET Z = X
64 GOTO 54
66 GOSUB 86
68 PRINT "BOY, AINT I DE
SMART ONE?"
70 PRINT "I GOT IT IN
JUST ";T;" TRIES"
72 PRINT
74 PRINT "COULD YOU
FACE ANOTHER GAME,
BUD?"
76 INPUT D$
78 GOSUB 86
80 IF D$ = "YES" THEN
GOTO 4
82 PRINT "BYE BYE THEN
WET BLANKET"
84 GOTO 82
86 FOR F = 1 TO
10*RND(15)
88 CLS
90 NEXT F
92 RETURN

```



## A Ching in his Armour

This is a pretty trivial program, but if you have a copy of the "I CHING" it can be most entertaining. As you can see, a lot of the work the ZX80 does in this program is non-essential. A random number generator could do the job just as well. But it is better that the operator do more than just press NEWLINE, if only to feel that he or she has influence on what is going on.

```

1  PRINT
2  PRINT
3  PRINT
4  PRINT
5  PRINT "I CHING"
6  PRINT
7  PRINT
8  PRINT "WHAT IS YOUR
   FIRST NAME"
9  INPUT A$
10 PRINT
11 PRINT
12 PRINT "AND LAST?"
13 INPUT B$
14 LET A = CODE(A$)
15 LET B = CODE(B$)
16 CLS
17 PRINT "YOUR
   HEXAGRAM FOR TODAY
   IS:"
18 PRINT
19 PRINT
20 FOR D = 1 TO 6
21 LET C = RND(A*B)
22 IF C > A*B/2 THEN
   PRINT "(shift W shift D
   space shift W shift D)"
23 IF NOT C > A*B/2 THEN
   PRINT "(shift W shift D
   shift W shift D shift W
   shift D)"
24 PRINT
25 NEXT D
26 PRINT
27 PRINT
28 PRINT "(5 spaces)SEE
   THE ";CHR$(128);
   CHR$(174);CHR$(128);
   CHR$(168);CHR$(173);
   CHR$(174);CHR$(179);
   CHR$(172);CHR$(128);
   " FOR"
29 PRINT "AN
   INTERPRETATION"
30 PRINT
31 LET H = C/A
32 IF H = 0 THEN LET H =
   RND(B)
33 PRINT "(4 spaces)YOUR
   LUCKY NUMBER IS ";H
34 POKE 16421, 24
35 STOP

```



## Nine Lives

In HANGCAT one player inputs a word, one letter at a time, pressing NEWLINE between each letter. Then the second player tries to guess the word, pressing NEWLINE between each letter he wants to try. If the second player is wrong he loses a life (hence the title). If the guess is right, the program prints out the correct letter in its correct position in the word.

```

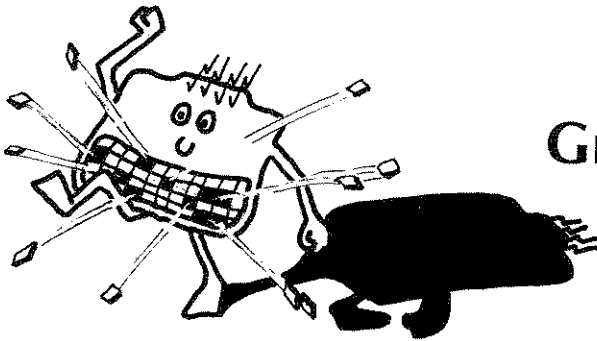
1  PRINT "HANGCAT"
2  PRINT "PLAYER 1. TYPE
   IN A WORD OF 6 (3
   spaces) LETTERS OR
   LESS"
3  PRINT
4  PRINT
5  LET G$ = "-" (shift J
   between the quote marks)
6  LET H$ = "-"
7  LET J$ = "-"
8  LET K$ = "-"
9  LET L$ = "-"
10 LET M$ = "-"
11 LET T = 0
12 INPUT A$
13 INPUT B$
14 INPUT C$
15 INPUT D$
16 INPUT E$
17 INPUT F$
18 CLS
19 PRINT "READY TO PLAY?
   YOU HAVE 9 LIVES"
20 PRINT "WHAT IS YOUR
   GUESS?"
21 INPUT X$
22 CLS
23 IF X$ = A$ THEN GOSUB
   30
24 IF X$ = B$ THEN GOSUB
   33
25 IF X$ = C$ THEN GOSUB
   36
26 IF X$ = D$ THEN
   GOSUB 39
27 IF X$ = E$ THEN GOSUB
   42
28 IF X$ = F$ THEN GOSUB
   45
29 GOTO 48
30 LET G$ = A$
31 GOSUB 70
32 RETURN
33 LET H$ = B$
34 GOSUB 70
35 RETURN
36 LET J$ = C$
37 GOSUB 70
38 RETURN
39 LET K$ = D$

```

```

40 GOSUB 70
41 RETURN
42 LET L$ = E$
43 GOSUB 70
44 RETURN
45 LET M$ = F$
46 GOSUB 70
47 RETURN
48 PRINT
49 PRINT ,G$;H$;J$;K$;L$;M$
50 PRINT
51 IF G$ = A$ AND H$ =
   B$ AND J$ = C$ AND K$
   = D$ AND L$ = E$ AND
   M$ = F$ THEN GOTO 68
52 PRINT
53 LET T = T + 1
54 IF T = 9 THEN GOTO 59
55 PRINT "YOU HAVE ";
   9 - T;"LIVES LEFT"
56 PRINT
57 PRINT
58 GOTO 20
59 PRINT "YOU'RE DEAD"
60 PRINT "THE WORD
   WAS ";A$;B$;C$;D$;E$;F$
61 PRINT ,"ANOTHER CAT?"
62 INPUT U$
63 CLS
64 IF NOT U$ = "NO"
   THEN GOTO 2
65 STOP
66 PRINT
67 PRINT
68 PRINT ,"YAY CAT"
69 GOTO 60
70 LET T = T - 1
71 RETURN

```



## Gnashing of Teeth

The following program, from the fiendish mind of Trevor, does nothing but drive you mad.

```

1 RANDOMISE
2 PRINT ,"FRUSTRATION"
3 PRINT
4 PRINT
5 LET A$ = "TYPE IN
   YOUR NAME"
6 PRINT ,"TO PLAY
   FRUSTRATION"

```



```

7 PRINT "PLEASE ";A$
8 INPUT F$
9 LET R = RND(7)
10 CLS
11 GOSUB 100
12 GOSUB (R+1)*10
13 INPUT B$
14 GOTO 9
20 PRINT "LOOK MATE.
ARE YOU GOING TO
GIVE"
21 PRINT "ME YOUR NAME
OR NOT?"
22 INPUT C$
23 IF C$ = "NO" THEN
GOTO 190
25 RETURN
30 PRINT "I SAID
PLEASE ";A$
35 RETURN
40 PRINT "I'M GETTING
FED UP WITH THIS"
41 PRINT "PLEASE ";A$
45 RETURN
50 PRINT "IF YOU WANT
TO PLAY THIS GAME"

51 PRINT "YOU'D
BETTER ";A$
55 RETURN
60 PRINT "CAN'T YOU OBEY
EASY INSTRUCTIONS?"
61 PRINT "ALL I WANT YOU
TO DO IS ";A$
65 RETURN
70 PRINT "WE CAN'T PLAY
IF YOU"
71 PRINT "DON'T ";A$
75 RETURN
80 PRINT "ARE YOU THICK
OR SOMETHING?"
81 PRINT,A$
85 RETURN
100 FOR D = 1 TO 10
110 PRINT
120 NEXT D
130 RETURN
190 CLS
200 PRINT "SOD YOU
THEN ";F$
210 STOP

```

---

## Let the longer games begin

After a while, you'll discover the 1K supplied is a bit cramping, and you'll long to stretch your wings and add a byte or two. When you do this, you'll discover that many of the good habits you've learned when you were restricted to 1K can desert you. It is very easy to set up a long and sloppy set of IF/THENS which could easily be replaced by an IF/THEN instruction to GOSUB. When you have memory to spare, it often seems too much trouble to bother cleaning up your programs. Unused subroutines clutter up the bottom ends of your programs. GOTO statements cover a multitude of situations which arose because you did not give sufficient thought to the maximum line number you would need

If you are going to take up flow-charting, now is the time to begin. If you can't be bothered with pretty triangles and things, at least discipline yourself to setting out — on paper — what your program is supposed to do, with arrows linking FOR/NEXT loops, and lines leading to the first lines of subroutines. If you can be bothered, it is worth writing out a full listing for a program once you get it working. Examine it in detail, and you're sure to find more elegant ways of achieving the same ends. Be particularly critical of each and every GOTO command which is non-conditional.

All the programs given so far in this book can act as starter ideas for much bigger and better programs when you get extra memory. For example, the LUNAR LANDING program is a grown up version of the 1K LUNAR LANDER and LABYRINTH is a much-expanded version of TIMEWARP and CAVE-MASTER.

The best thing you can add to a program with added memory is the element of surprise. If you can include situations which do not occur every time a game is played, you'll ensure the game will remain interesting for a much longer time than would be the case if every situation is triggered every time a game is run.

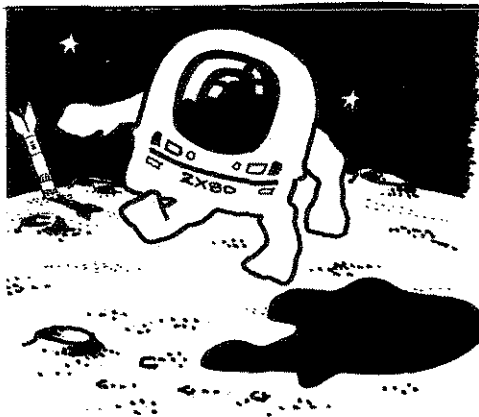
As you know, many of the games in this book run far too fast to be good games without the use of a "delay subroutine". However, as you get into longer games (and I mean ones much longer than those listed in this section) you'll find that slow-running programs and response-times can be boring, especially if you have a graphical element in your program, which "moyes" in some way from go to go. You'll find that programs run much faster if you place often-used subroutines at the TOP of the listing, rather than after the main body of program as we have tended to do. The program runs faster because the computer must search through the whole listing, from the lowest number, when it comes across a GOTO or GOSUB command. If the GOSUB is near the top, it does not have to go through, say, the instructions every time it searches for a subroutine.

You can make the first line of the program GOTO some line number to jump over the subroutines. Variables which must be assigned at the start of a game can actually be listed right at the end of the program, ending with a GOTO leading back to the start of the game program proper. You can have a line like "DO YOU WANT INSTRUCTIONS?" near the start of a program, and if the answer is "YES" the computer can GOTO the end of the program where the instructions are. LABYRINTH, if you wanted it to run more quickly, could have the instructions at the very end and the delay/spacing subroutine (GOSUB 1000) near the start. Doing this would, of course, somewhat defeat the purpose of having a delay subroutine. However, when you add a 16K pack to your ZX80, you'll have very, very long programs, and you will not always want to wait while the computer searches a vast listing for the subroutine.

Another way of improving programs, and making them interesting to players for a longer time, is to use a feature you've seen in some programs, the "degree of difficulty". Make sure that this feature really does increase the difficulty of the game. Ensure that, even at the highest level of play, the final score (or successful landing, or obliterated aliens or whatever) is attainable.

You can add interest to games by awarding points, or scores, or ratings or whatever, that are genuinely related to the speed, skill or whatever the player demonstrated. A further twist is to award a "rank" (like "star fleet captain", "novice" or "incompetent fool") to the player, depending on how well he or she did. Points and ranks ensure that a player remains interested in a game for a longer time, as the player will try to beat his or her previous best score or ranking.

Keeping these features in mind, have a look at the next two games, input them and run them, and then try to improve on them.



## Lunar Landing

```

10 LET M = 0
20 LET T = 0
30 LET S = 0
40 LET H = 5000
50 PRINT "LUNAR
LANDING"
60 LET F = 5000/RND(3)
70 LET Q = -17
80 LET B = 1
90 RANDOMISE
100 PRINT
110 PRINT
120 GOTO 300
130 PRINT "(shift Q twice) +
IS TOWARDS LUNA (shift
Q twice)"
140 INPUT Z
150 IF Z < -50 OR Z > 50
THEN GOTO 410

```

```

160 PRINT "FOR HOW MANY
    SECONDS?"
170 INPUT E
180 CLS
190 LET T = T + E
200 LET S = S + 10 +
    3*E((Z + 1)/B)
210 LET F = F -
    3*E*ABS(Z*RND(3))
220 IF F 500 THEN PRINT
    "(shift Q three times)
    FUEL LOW (shift Q three
    times)"
230 LET H = H - E*S
240 IF H < 20 AND H > -10
    AND S < 12 THEN GOTO
    460
250 IF H < -10 THEN GOTO
    430
260 IF F < 0 THEN GOTO 430
270 LET X = RND(10)
280 IF X = 5 AND NOT M =
    2 THEN GOSUB 790
290 PRINT ""
300 PRINT CHR$(128 + M);
    " HEIGHT ABOVE
    SURFACE: ";H
310 IF NOT Q = -17 THEN
    LET Q = Q - RND(16)
320 IF Q < 0 AND Q > -17
    THEN GOTO 430
330 IF NOT Q = -17 THEN
    PRINT "(shift S six times)
    OXYGEN LEFT: ";Q
340 PRINT CHR$(128 + M);
    " VELOCITY: ";S
350 IF NOT B = 1 THEN
    PRINT CHR$(128 +
    M);"(shift Q three times)
    WARNING - THRUST
    ERRATIC"
360 PRINT CHR$(128 + M);
    " FUEL LEFT: ";F
370 PRINT CHR$(128 + M);
    " FLIGHT TIME: ";T
380 GOSUB 510
390 PRINT
400 GOSUB 550
410 PRINT "THRUST (-50 TO
    +50)?"
420 GOTO 130
430 CLS
440 PRINT "CRASH. HIT
    SURFACE AT ";ABS(S);
450 GOTO 440
460 CLS
470 PRINT "SUCCESSFUL
    LANDING"
480 PRINT
490 PRINT "FINAL
    VELOCITY: ";ABS(S);
500 GOTO 490
510 FOR A = 1 TO 64
520 PRINT CHR$(128 + M);
530 NEXT A
540 RETURN
550 PRINT
560 PRINT " "(space shift F
    shift D)"
570 FOR Y = 1 TO 2
580 PRINT " "(space)";
    CHR$(128 + M);
    CHR$(128 + M)
590 NEXT Y
600 PRINT " "(shift A shift G
    shift G shift A)"
610 PRINT " "(shift A shift G
    shift G shift A)"
620 PRINT
630 FOR G = 1 TO 2
640 LET K = RND(10)
650 IF K = 1 THEN LET W$
    = " : : "
660 IF K = 2 THEN LET W$
    = " : : "
670 IF K = 3 THEN LET W$
    = " : : "
680 IF K = 4 THEN LET W$
    = " : : "
690 IF K = 5 THEN LET W$
    = " : : "

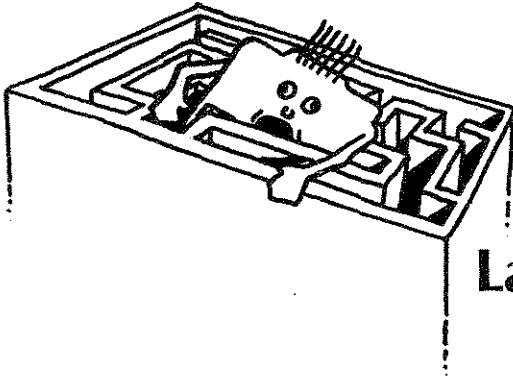
```

```

700 IF K = 6 THEN LET W$
    = "...."
710 IF K = 7 THEN LET W$
    = "...."
720 IF K = 8 THEN LET W$
    = "...."
730 IF K = 9 THEN LET W$
    = " :'"
740 IF K = 10 THEN LET W$
    = " .."
750 PRINT „W$
760 NEXT G
770 PRINT
780 RETURN
790 CLS
800 LET M = M + 1
810 RANDOMISE
820 FOR V = 1 TO 7
830 PRINT
840 NEXT V
850 LET U = RND(32000)
860 FOR V = 1 TO 6
870 PRINT "HOUSTON, WE
    HAVE A PROBLEM..."
880 PRINT ,CHR$
    (127 + RND(11));
    " DANGER ";CHR$
    (127 + RND(11))
890 NEXT V
900 PRINT
910 PRINT "MALFUNCTION.
    USE COMPUTER"
920 PRINT "ACCESS
    CODE ";U;" FOR
    DETAILS"
930 INPUT V
940 CLS
950 IF NOT U = V THEN
    GOTO 430
960 LET V = RND(2)
970 IF V = 1 THEN GOSUB
    1030
980 IF V = 2 THEN GOSUB
    1070
990 PRINT "N/L TO RETURN
    TO FLIGHT"
1000 INPUT V$
1010 CLS
1020 RETURN
1030 LET Q = 100 + RND(19)
1040 PRINT "OXYGEN METER
    UNRELIABLE"
1050 PRINT "(shift A 23 times)"
1060 RETURN
1070 LET B = B + RND(3)
1080 PRINT "THRUST
    CONTROL ERRATIC"
1090 PRINT "(shift W 22
    times)"
1010 RETURN

```

---



## Labyrinth

```

10 GOSUB 1000
20 PRINT "LABYRINTH"
30 PRINT "(shift G nine
   times)"
40 LET X = 0
50 LET S = 30
60 LET W = 1
70 PRINT "YOU ARE AT
   THE START OF A"
80 PRINT "LABYRINTH OF
   MANY TWISTING,"
90 PRINT "TURNING
   TUNNELS. YOU HAVE"
100 PRINT "A SACK
   HOLDING 30 PIECES OF"
110 PRINT "SILVER. YOU
   MUST GET TO"
120 PRINT "THE END OF
   THE LABYRINTH WITH
   AT"
130 PRINT "LEAST 20 TO PAY
   THE MINOTAUR"
140 PRINT
150 PRINT "PRESS
   NEWLINE"
160 INPUT A$
170 RANDOMISE
180 IF NOT A$ = "" THEN
   STOP
190 GOSUB 1000
200 IF W < 1 THEN LET
   W = 1
210 PRINT "(shift Q seven
   times) THIS IS
   MAZE/TUNNEL"
220 PRINT
230 IF W = 10 THEN GOTO
   1070
240 PRINT "NUMBER ";W;
   " OF THE LABYRINTH"
250 PRINT
260 PRINT "(10 IS THE END)"
270 PRINT
280 LET X = X + 1
290 PRINT "(shift S) THIS IS
   CHALLENGE NUMBER ";X
300 IF S < 1 THEN LET S = 5
310 PRINT
320 PRINT "YOU HAVE ";S;
   " SILVER PIECES"
330 LET K = K + RND(5)
340 PRINT
350 PRINT "FACING YOU
   NOW ARE ";K;" DOORS"
360 PRINT "WHICH ONE
   WILL YOU TRY?"
370 INPUT A
380 GOSUB 1000
390 IF RND(10) = 5 THEN
   GOSUB 690

```

```

400 IF NOT A = K THEN
    GOSUB 420
410 IF A = K THEN GOSUB
    690
420 LET K = RND(4)
430 IF K = 1 THEN LET E$ =
    "RODENTING RAT"
440 IF K = 2 THEN LET E$ =
    "WART-FACED
    WOGGLE"
450 IF K = 3 THEN LET E$ =
    "ELLIPSOID OCTOPUS"
460 IF K = 4 THEN LET E$ =
    "WACKED-OUT
    WIZARD"
470 PRINT "FOOL, YOUVE
    WALKED IN ON"
480 LET E = RND(4)
490 IF E = 1 THEN LET F$ =
    "FLAMING BRAND"
500 IF E = 2 THEN LET F$ =
    "SHINING SWORD"
510 IF E = 3 THEN LET F$ =
    "POISONED NEEDLE"
520 IF E = 4 THEN LET F$ =
    "GOSUB-MACHINE-GUN"
530 PRINT "{four
    spaces}";E$;" ARMED"
540 PRINT "{three
    spaces}WITH A ";F$
550 PRINT
560 PRINT "WHICH WEAPON
    DO YOU CHOOSE?"
570 PRINT
580 PRINT "A FLOATING
    POINT ROM (1),"
590 PRINT
600 PRINT "A FOR/NEXT
    LOOP (2),"
610 PRINT
620 PRINT "OR A POKED
    ADDRESS (3)?"
630 INPUT B
640 LET C = RND(3)
650 GOSUB 1000
660 IF B = C THEN GOSUB
    1170
670 IF NOT B = C THEN
    GOSUB 1240
680 GOTO 140
690 LET K = RND(4)
700 IF K = 1 THEN GOSUB
    760
710 IF K = 2 THEN GOSUB
    810
720 IF K = 3 THEN GOSUB
    850
730 IF K = 4 THEN GOSUB
    900
740 GOTO 140
750 PRINT
760 PRINT "YOUVE FALLEN
    THROUGH"
770 PRINT " A
    TRAPDOOR...."
780 LET W = W - 1
790 LET S = S - RND(2)
800 RETURN
810 PRINT "A WALL OF
    FLAME ENGULFS YOU"
820 LET W = W - 1
830 LET S = S - RND(2)
840 RETURN
850 PRINT "THE LOVELY
    PRINCESS SEMOLINA"
860 PRINT "SOOTHES YOUR
    FEVERED BROW"
870 LET S = S + RND(5)
880 LET W = W + RND(3)
890 RETURN
900 PRINT "JOY OH JOY. A
    HOARD OF"
910 PRINT "SILVER. CHOOSE
    UP TO 5 PIECES"
920 PRINT "BUT BE
    WARNED. THE MORE"
930 PRINT "YOU TAKE, THE
    MORE IT WILL"
940 PRINT "COST YOU. HOW
    MANY?"
950 INPUT D
960 LET S = S + D
970 LET W = W - D/2

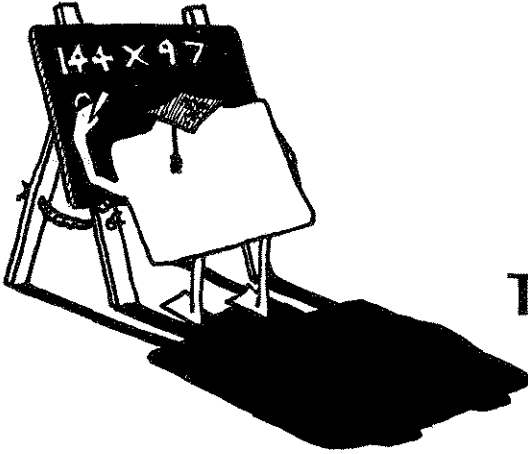
```

```

980 RETURN
990 CLS
1000 FOR O = 1 TO
      20*RND(50)
1010 NEXT O
1020 FOR I = 1 TO 5
1030 PRINT
1040 NEXT I
1050 RETURN
1060 IF NOT W = 10 THEN
      RETURN
1070 PRINT "YOU ARE AT
      THE END"
1080 PRINT "DO YOU HAVE
      ENOUGH SILVER?"
1090 PRINT "PRESS N/L TO
      FIND OUT"
1100 INPUT C$
1110 IF S < 20 THEN PRINT
      "THE MINOTAUR HAS
      EATEN YOU"
1120 IF S < 20 THEN GOTO
      1110
1130 IF S > 19 THEN PRINT
      "YES, YOU HAVE ";S;
      " SILVER"
1140 IF S > 19 THEN PRINT
      "PIECES. YOU HAVE
      WON",
1150 IF S > 19 THEN GOTO
      1130
1160 STOP
1170 PRINT "YOU BEAT
      THE ";E$
1180 LET S = S + RND(3)
1190 PRINT "AND HAVE ";S;
      " SILVER PIECES"
1200 LET W = W + RND(4)
1210 PRINT
1220 PRINT "YOU ARE
      APPROACHING
      SECTOR ";W
1230 RETURN
1240 PRINT "THE ";E$;
      " BEAT YOU,"
1250 LET S = S - RND(4)
1260 PRINT "LEFT YOU
      WITH ";S;" SILVER"
1270 LET W = W - 1
1280 IF W < 1 THEN LET
      W = 1
1290 PRINT "AND SENT YOU
      BACK TO ";W
1300 RETURN

```





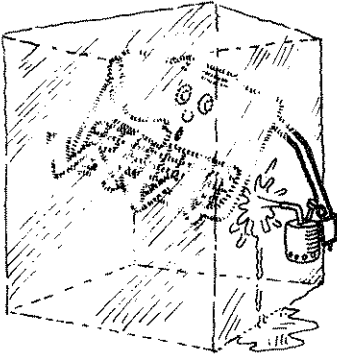
## The ZX80 as Teacher

The ZX80 is an effective game-player. Writing and running programs on the computer enhances programming skills. However, the ZX80 can also be used in a direct role as a teaching aid. Its main use is in the field of quizzes. While the ZX80 can only select from a list of non-numerical questions, the computer can easily be programmed to create its own numerical questions.

Another use in teaching is for the ZX80 to create lists and tables. We will look at this use first.

```
1  PRINT
2  PRINT
3  PRINT,
   "MULTIPLICATION
   TABLES"
4  PRINT
5  PRINT
6  PRINT
7  PRINT "WHICH TIMES
   TABLE WOULD YOU"
8  PRINT , "LIKE ME TO
   PRINT?"
9  INPUT A
10 CLS
11 FOR J = 1 TO 12
12 PRINT ,,J;" X ";A;
   " = ";A*J
13 NEXT J
14 PRINT
15 PRINT
16 PRINT "DO YOU WANT
   ANOTHER GO?"
17 INPUT A$
18 CLS
19 IF NOT A$ = "NO"
   THEN GOTO 4
20 FOR S = 1 TO 5
21 PRINT
22 NEXT S
23 PRINT , "OK. BYE FOR
   NOW"
24 STOP
```

A far more useful table is produced by the following program (and the display is a little more imaginative). Again, this program is given to suggest ideas for your own programs.



## A Degree of Conversion

```

10  RANDOMISE
20  PRINT "(5 spaces)A
    DEGREE OF
    CONVERSION"
30  LET K = RND(9)
40  FOR S = 1 TO 32
50  PRINT CHR$(K + 129);
60  NEXT S
70  PRINT
80  PRINT
90  PRINT
100 PRINT
110 PRINT "WHAT IS THE
    LOWEST TEMP. (F) YOU"
120 PRINT "WANT TO
    CONVERT?"
130 INPUT A
140 PRINT ",AND HIGHEST?"
150 INPUT B
160 IF B < A THEN LET
    E = A
170 IF A < B THEN LET
    F = A
180 IF B < A THEN LET
    F = B
190 IF A < B THEN LET
    E = B
200 PRINT "IN WHAT
    DEGREE STEPS?"
210 INPUT Z
220 CLS
230 PRINT ",F", "C", "K"
240 PRINT " ";
250 LET H = RND(9)
260 FOR U = 1 TO 21
270 PRINT CHR$(H + 129);
280 NEXT U
290 PRINT
300 FOR D = F TO E + Z
310 LET C = 5*(D - 32)/9
320 LET K = C + 273
330 PRINT ",D,C,K
340 LET D = D + Z - 1
350 NEXT D
360 STOP

```

In all programs, and especially in educational ones, you have to try and anticipate any mistakes (deliberate or otherwise) users will make when running a program. The lines 160 to 190 cover the possibility that a user will input the higher temperature first, rather than the lower one which was requested.

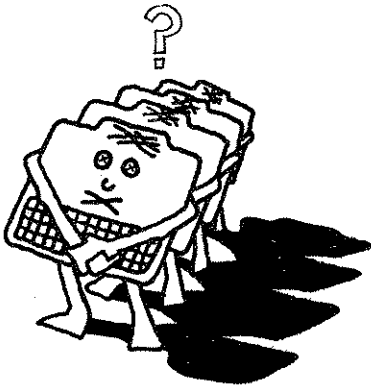
There is no reason why educational programs should not have displays that are as attractive as games programs, so lines 30 to 60 and lines 250 to 280 select, in effect, a random underline. It is a good idea to incorporate such features whenever you have sufficient memory.

Line 300 tells the ZX80 to print the value above the maximum requested, in order to ensure that the required temperature is covered.

This is quite an interesting program to run, especially if you select an enormous range of temperatures, and select an unexpected increment (like 117 degrees). Of course, the ZX80 deals as easily with these cases as it does with the more predictable 0 degrees to 100 degrees in steps of 10 degrees, but it seems a little surprising to first-time users that it does so.

As an exercise, modify the program so it will not print any value but 0 if the temperature drops below absolute zero (0 Kelvin - 273 C).

We will look now at a numerical quiz, in which the ZX80 creates the questions, checks their correctness, and then gives the user a score.



## Multiplication Quiz

- |   |                              |   |   |
|---|------------------------------|---|---|
| 1 | LET H = 0                    | 5 | PRINT "DEGREE OF DIFFICULTY (1 TO 10)?" |
| 2 | GOSUB 46                     | 6 | INPUT A                                 |
| 3 | PRINT, "MULTIPLICATION QUIZ" | 7 | IF A < 0 OR A > 10 THEN GOTO 5          |
| 4 | GOSUB 46                     | 8 | LET A = A/2                             |

```

9   IF A = 0 THEN LET
    A = 1
10  GOSUB 46
11  PRINT "HOW MANY
    QUESTIONS?"
12  INPUT B
13  IF B < 1 THEN GOTO 11
14  CLS
15  FOR G = 1 TO B
16  LET C = A*RND(10)
17  LET D = A*RND(10)
18  LET E = C*D
19  GOSUB 46
20  PRINT "QUESTION
    NUMBER ";G
21  GOSUB 46
22  PRINT "WHAT IS ";C;
    " TIMES ";D;"?";
23  INPUT F
24  PRINT "(5 spaces)";F
25  GOSUB 46
26  IF F = E THEN GOTO 42
27  PRINT "INCORRECT. THE
    ANSWER IS ";E
28  GOSUB 46
29  PRINT "YOUR SCORE
    IS ";H
30  PRINT ,(single
    space)RIGHT OUT OF ";G
31  PRINT "PRESS
    NEWLINE";
32  IF NOT G = B PRINT
    "(single space) TO
    CONTINUE"
33  INPUT A$
34  CLS
35  NEXT G
36  FOR K = 1 TO 3
37  GOSUB 46
38  NEXT K
39  PRINT "END OF QUIZ.
    YOUR SCORE IS"
40  PRINT ,H*100/G;" PER
    CENT"
41  STOP
42  LET H = H + 1
43  GOSUB 46
44  PRINT "CORRECT. THE
    ANSWER IS ";E
45  GOTO 28
46  FOR S = 1 TO 3
47  PRINT
48  NEXT S
49  RETURN

```

To make this an addition, subtraction or division quiz, simply change the arithmetic operation specified in line 18, and the word (TIMES in the above program) in line 22.

There are a number of things which can be learned from this program. The most obvious is the use of the subroutine (GOSUB 46) which spaces the question and answer across the screen, enhancing readability without wasting memory with endless blank PRINT statements. Line 24 prints the user's answer, so it can be compared with the correct answer. It can be quite frustrating if the answer just disappears when NEWLINE is pressed. Lines 7 and 13 ask the questions in lines 5 and 11 again if an answer in the required range is not given the first time. There is no reason why you cannot put in an additional line which says something like ONLY ANSWERS BETWEEN 1 AND 10 ARE ACCEPTABLE if you have sufficient memory. Lines 8 and 9 can be omitted if you want very difficult quizzes to be available. The zero to 10 option in line 5 is to give the user the impression that he or she has greater influence on the program than in fact is the case. As an exercise, try to rewrite lines 8 and 9 so only one line of program is

given. It was put in the two-line form so it would be clear what was happening, but it can easily be written more efficiently.

At the end of all questions in a set, but the last, lines 31 and 32 cause PRESS NEWLINE TO CONTINUE to appear at the bottom of the display. Line 32 ensures that after the final question only the words PRESS NEWLINE to appear.



The following table, which prints out numbers and their squares, uses a counter to stop execution when the screen is full. You can use this idea if the program output is sequential and is likely to cause the program to crash by demanding more screen space than you have.

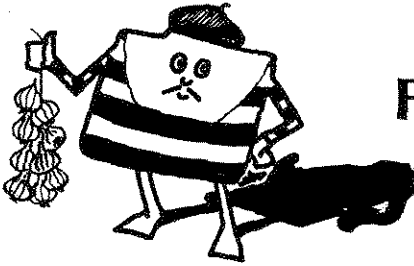
```
1 LET K = 0
2 GOSUB 22
3 PRINT "SQUARES"
4 GOSUB 22
5 PRINT "LOWEST
NUMBER?"
6 INPUT A
7 GOSUB 22
8 PRINT "HIGHEST
NUMBER?"
9 INPUT B
10 GOSUB 22
11 PRINT "PRESS NEWLINE
FOR TABLE"
12 INPUT U$
13 CLS
14 FOR X = A TO B
15 LET K = K + 1
16 IF K = 20 THEN GOSUB
26
17 PRINT,X;" SQUARED
IS ";X*X
18 NEXT X
19 PRINT
20 PRINT "END OF TABLE"
21 STOP
22 FOR C = 1 TO 3
23 PRINT
24 NEXT C
25 RETURN
26 PRINT "PRESS NEWLINE"
27 LET K = 0
```

```

28 INPUT A$
29 CLS
30 RETURN

```

I'm sure you will now be able to work out an endless stream of numerical quizzes for yourself, your children or your class to tackle. Non-numerical quizzes are very useful, but they require much more work in programming. Whereas the ZX80 can create its own numerical questions, each non-numerical question must be specified, and each answer included in full in the program.



## French Vocabulary

Here is a sample quiz which can be adapted for any subject.

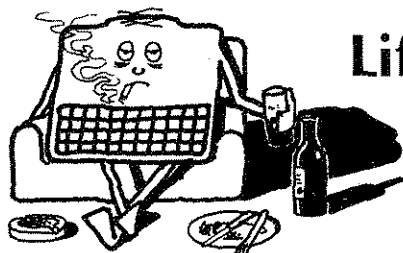
```

10 RANDOMISE
20 LET A$ = "GIVE THE
  FRENCH FOR"
30 LET B$ = "CORRECT"
40 LET C$ = "INCORRECT.
  ANSWER IS "
50 PRINT "FRENCH VOCAB
  1"
60 LET A = 0
70 PRINT "HOW MANY
  QUESTIONS?"
80 INPUT N
90 FOR J = 1 TO N
100 CLS
110 LET R = RND(10)
120 GOSUB 10*R + 240
130 PRINT A$,E$
140 INPUT G$
150 IF G$ = F$ THEN LET
  A = A + 1
160 IF G$ = F$ THEN PRINT
  B$
165 IF NOT G$ = F$ THEN
  PRINT C$,F$
170 PRINT "PRESS NEWLINE"
175 INPUT H$
180 CLS
185 NEXT J
190 PRINT "OUT OF ";N;
  " QUESTIONS YOU"
200 PRINT "HAD ";A;
  " RIGHT, ";A*100/N;
  " PERCENT "
210 PRINT "ANOTHER GO?"
220 INPUT D$
230 IF D$ = "YES" THEN
  GOTO 60
240 STOP
250 LET E$ = "NEXT"
255 LET F$ = "PROCHAIN"

```

257	RETURN	307	RETURN
260	LET E\$ = "YOUNG"	310	LET E\$ = "HOT"
265	LET F\$ = "JEUNE"	315	LET F\$ = "CHAUD"
267	RETURN	317	RETURN
270	LET E\$ = "HERE"	320	LET E\$ = "FULL"
275	LET F\$ = "ICI"	325	LET F\$ = "PLEIN"
277	RETURN	327	RETURN
280	LET E\$ = "BETTER"	330	LET E\$ = "SUGAR"
285	LET F\$ = "MEILLEUR"	335	LET F\$ = "LE SUCRE"
287	RETURN	337	RETURN
290	LET E\$ = "MORE"	340	LET E\$ =
295	LET F\$ = "PLUS"		"EVERYWHERE"
297	RETURN	345	LET F\$ = "PARTOUT"
300	LET E\$ = "INSIDE"	347	RETURN
305	LET F\$ = "DEDANS"		

Note that there is no mechanism in this program for preventing the same question being asked more than once in a run. The limit set on the random number in line 110 (that is, the number in brackets after RND) is equal to the number of different questions in the program. You can get ten programs of this length onto one side of a C-12 cassette (just), so — with programs similar to the one listed above — a 200-word vocabulary can be tested with the programs stored on a single cassette. If your questions need more than a one word reply, you will find (of course) that you can fit far fewer questions into a single program before running out of RAM.



## Life Expectancy

There was a light-hearted life expectancy program earlier in the book. The next program, based loosely on actuarial tables, is closer to

a "serious" life expectancy program. However, it has had to be simplified, and therefore to some degree rendered less accurate, to fit within 1K. It is not really an "educational" program (although it could be used within a classroom as a demonstration of the "real" use of computers) but this seemed the best place in the book to put it.

```

1   LET C = 0
2   PRINT "LIVES"
3   LET A = 71
4   PRINT "AGE (YRS)?"
5   INPUT B
6   GOSUB 54
7   PRINT "MARRIED?"
8   INPUT A$
9   GOSUB 54
10  IF A$ = "YES" THEN LET
    A = 76
11  PRINT "HAVE YOU BEEN
    RICH OR POOR"
12  PRINT "MOST OF YOUR
    LIFE?"
13  INPUT C$
14  GOSUB 54
15  IF C$ = "YES" THEN LET
    A = A - 3
16  PRINT "ARE YOU
    OVERWEIGHT?"
17  INPUT D$
18  GOSUB 54
19  IF D$ = "NO" OR B < 40
    THEN GOTO 24
20  IF D$ = "YES" THEN
    PRINT,
    "BY HOW MANY
    POUNDS?"
21  INPUT P
22  LET C = C + P
23  GOSUB 54
24  PRINT "EXERCISE?
    NEVER. SOMETIMES.
    OFTEN"
25  INPUT E$
26  GOSUB 54
27  IF E$ = "SOMETIMES"
    THEN LET A = A + 3
28  IF E$ = "OFTEN" THEN
    LET A = A + 5
29  GOSUB 5
30  PRINT "ARE YOU OFTEN
    TENSE?"
31  INPUT F$
32  GOSUB 54
33  IF F$ = "YES" THEN LET
    A = A - 3
34  IF F$ = "NO" THEN LET
    A = A + 3
35  PRINT "DRINK? LITTLE
    (0), MOD.(5)"
36  PRINT "HEAVY(10)"
37  INPUT G
38  IF B > A THEN LET A =
    B + (B - A)/2
39  GOSUB 54
40  PRINT "DO YOU
    SMOKE?"
41  INPUT H$
42  IF H$ = "YES" THEN LET
    A = A - 5
43  GOSUB 54
44  PRINT "OFTEN ILL?"
45  INPUT K$
46  IF K$ = "YES" THEN LET
    A = A - 3
47  IF K$ = "NO" THEN LET
    A = A + 3
48  GOSUB 54
49  PRINT "EST. AGE AT
    DEATH:"
50  PRINT
51  PRINT "FEMALE - ";
    A + 7 - C/5 - G
52  PRINT "MALE - ";
    A - C/5 - G
53  STOP

```



```
54 CLS
55 FOR Z = 1 TO 3
56 PRINT
```

```
57 NEXT Z
58 RETURN
```

---

## Useful Subroutines

Science of Cambridge claim the 1K RAM supplied with the standard ZX80 is equal to 4K of anybody else's RAM, because most commands and statements are stored in a single byte. Although it appears that this claim is a little ambitious, the RAM can be made to hold a pretty long program (from 30 to 100 lines) and the shortage of memory teaches ingenuity in programming. As you'll have discovered, Sinclair's dialect of BASIC has no facility for READ/DATA or for STEP and works only with whole number (integer) arithmetic. In this section of the book we will look at some useful subroutines, some of which have been introduced in the programs. I thought it would be handy to have them all together in one place. Many of the subroutines were worked out by inventive members of the Users Club and have already been printed in the club magazine INTERFACE.

Clive Davies of Cheltenham devised the following subroutine to replace the missing READ/DATA function:

```
10 LET N = 0
20 FOR J = 0 TO 4
30 LET N = (N*10 + CODE
   (Z$ - 28)
40 LET Z$ = TL$(Z$)
50 NEXT J
```

This subroutine supplies five digits of data stored as Z\$.

In Basildon, Dave Fenn developed an elegant subroutine to overcome the lack of a STEP facility in Sinclair BASIC:

Instead of:

```
10 FOR X = 0 TO 130 STEP
   10
20 PRINT X
30 NEXT X
```

Dave suggests substituting:

```
10 FOR X = 0 TO 130
20 PRINT X
30 LET X = X + 10 - 1
40 NEXT X
```

To STEP down, substitute for line 30: LET X = X - 10 - 1, and for line 10: FOR X = 130 TO 0.

The STEP-up routine is used in the temperature program A DEGREE OF CONVERSION.

An efficient way of making the most of memory in some programs is to arrange for subroutine destinations to be specified by a multiple of, or a multiple plus an arithmetic manipulation of a randomly generated number. That sounds more complicated than it is in practice. Look at the following program:

```
10 LET K = RND(4)           70 PRINT "SUBROUTINE 70"  
20 GOSUB 10*K + 50         75 RETURN  
30 PRINT                   80 PRINT "SUBROUTINE 80"  
40 PRINT "THIS IS LINE 40" 85 RETURN  
50 STOP                   90 PRINT "SUBROUTINE 90"  
60 PRINT "SUBROUTINE 60"   95 RETURN  
65 RETURN
```

As you can see, line 20 simply manipulates the random number generated in line 10 to produce a destination for the subroutine jump. The manipulation can be a simple multiple (GOSUB 5\*J), a multiple plus an addition (as in line 20) or subtraction, or a conditional expression plus a manipulation of the random number (IF J > 10 THEN GOSUB 10\*J + N).

Science of Cambridge have an optional plug-in 8K ROM which provides floating-point arithmetic, but without it, ZX80 owners have to fall back on subroutines or "tricks" to get an approximation of decimal places. There is a subroutine in the manual, but it uses a lot of precious memory. The following is not really satisfactory, but there is a way to approximate a few decimal places. One way is to multiply one of the integers you're dealing with by, say, 100 before you start division and then mentally add the decimal point yourself, two digits from the right when you get the result.

For example, to divide 12 by 7 (if you'd ever bother to use your ZX80 to do such a thing) you would normally input:

```
10 LET J = 12/7  
20 PRINT J
```

This would give the answer 1 which is pretty useless. To get two "decimal places" you could input the following program:

```
10 INPUT A           40 PRINT J;" WITH  
20 INPUT B           DECIMAL POINT TWO  
30 LET J = 100*A/B  DIGITS FROM RIGHT" (A  
subroutine can easily be  
devised to get the ZX80  
to put the decimal point  
in, but it seems scarcely  
worthwhile.)
```

This program would give you, if you let A equal 12 and B equal 7, a value for J of 171 ("1.71") which is a lot better than 1 for an answer (and not too far away from the real answer of 1.71428...). Multiply A by

1000 and you get three "decimal places", i.e. 1.714, but this would limit you to numbers less than 32, to avoid exceeding the ZX80's upper integer limit.

The subroutine is useful if you want the ZX80 to give the result of a computation as an approximate percentage (as in the French vocabulary program, and the one which flipped a coin). For example, if you wanted the ZX80 to express 7/12 as a percentage, you could not use the line: PRINT "THE ANSWER IS ";(7/12)\*100;" PER CENT". This would give you the result 0. However, if you change the order of the computation to read: PRINT "THE ANSWER IS ";7\*100/12;" PER CENT" you would get an answer which would be within 1 per cent of the correct value.

In the program DIMMER SPIDER we introduced a line which removed the numerical message at the bottom of the screen which indicates a STOP command has been executed. This line, suggested by Ian Rodgers of Wimbledon, is: POKE 16421, 24 and is placed in a program just before the STOP command.

Colin Hughes of Luton has written a subroutine to renumber lines of a program listing in steps of 10. The subroutine was slightly modified by John Bloxham of Stratford and Brian Cross from Liphook (John and Brian both suggested swapping PEEK addresses in line 9989).

```

9985 REM RENUMBER          9994 IF PEEK(N)*256 + PEEK
9986 LET L = 10           (N + 1) = 9985 THEN
9987 LET S = 10           STOP
9988 LET B = 16424        9995 POKE N, L/256
9989 LET E = PEEK         9996 POKE N + 1,
    (16393)*256 + PEEK    L - (L/256)*256.
    (16392)              9997 LET L = L + S
9990 LET F = 0           9998 IF PEEK(N) = 118 THEN
9991 FOR N = B TO E      LET F = 0
9992 IF F THEN GOTO 9998 9999 NEXT N
9993 LET F = -1

```

As John Bloxham points out, the manual indicates that the ms byte of the line number comes first, but that appears to relate to the actual program storage area in the RAM, whereas 16392/3 relate to the area marked VARS.

Joe Fitzpatrick, a users club member from Dublin suggests a subroutine which stops a program crashing when the screen is full. He included it in a program which displays each address and the character stored in it. His program essentially consists of printing the characters whose values are the variable A in the expression A = PEEK(B) where B is a number in the range 16424 to 17424. The subroutine to temporarily halt the program when the screen is full is:

```

30 IF PEEK (16421) < 3      ....
    THEN GOSUB 100         100 PRINT

```

```

110 PRINT "PRESS          130 CLS
    NEWLINE TO          140 IF A$ = "" THEN
    CONTINUE"           RETURN
120 INPUT A$           150 STOP

```

The use of the quote marks in line 140 ensures that the ZX80 will RETURN only if just NEWLINE is pressed.

Pressing any other key before NEWLINE will cause the program to STOP.

Paul Jobling, from Rugeley, developed two useful subroutines which he sent to the users club. The first imitates the action of LEFT\$ in other versions of BASIC:

```

1000 LET X$ =
    CHR$(CODE(X$))
1010 RETURN

```

Paul points out that this is useful for input which can be YES/NO or other decisions. It prevents confusion if a player inputs Y for YES, instead of YES, which it will also accept. GOSUB 1000, if you have assigned the last variable input X\$ will make X\$ become the first letter of X\$. That is, YES becomes Y, as does YUP, Y and YESSIR.

The second subroutine contributed by Paul allows you to specify the last letter in a word, a function carried out in some BASICS by RIGHT\$:

```

990 LET A = 0           1040 GOTO 1010
1000 LET X$ = Z$       1050 FOR F = 1 TO A - 1
1010 IF CODE Z$ = 1 THEN 1060 LET X$ = TL$(X$)
    GOTO 1050          1070 NEXT F
1020 LET A = A + 1     1080 RETURN
1030 LET Z$ = TL$(Z$)

```

You will recall that a delay was built into many games. The delay subroutine was of the form:

```

1000 FOR X = n1 TO n2      1010 CLS
    (where n1 and n2 are    1020 NEXT X
    any numbers)          1030 RETURN

```

In fact, it is not necessary for the CLS to be within the loop. It can just as easily be after the NEXT line and before the RETURN line or even at the start of the subroutine. It is just a matter of personal choice where you put CLS. The ZX80 will delay for a period related to the difference between  $n_1$  and  $n_2$  whether it has anything to do within the loop or not. If you want the delay to be variable, line 1000 can be of the form: FOR X = 1 TO A\*RND(B) where A and B are either set at the beginning of the game, or vary depending on other variables during the course of a run. For example, A can rise or decrease depending on the value of the master FOR/NEXT loop which is counting the number of attempts a player is having in a game, or A can be a "degree of difficulty".

Jeremy Ruston of Kensington contributed a single line which tells you how many bytes (that is, how much memory) the current program takes up. Just input: 9999 PRINT PEEK (16392) + PEEK (16393)\*256 - 16461. If you then input RUN 9999 you'll find the screen will clear and a number will appear in the top left-hand corner of the screen, the number of bytes in the program which you currently have in the computer.

You can get your ZX80 to play music (after a fashion) by using the following program, written by club member Philip Joy of Romford.

This is just the bare bones of the music idea. You can use the concept to write music into programs so a win is rewarded with a trill of a few notes, or whatever. This program uses a machine code subroutine (the USR function).

First input the following:

```
10 POKE 17000, 237
20 POKE 17001, 65
30 POKE 17002, 201
```

RUN this program, then delete lines 10, 20 and 30 by inputting the line number. Do not press NEW.

Next, input the following program:

```
5 LET K = 17000
10 INPUT X
20 INPUT Y
30 INPUT Z
40 FOR D = 1 TO 30
50 FOR A = 1 TO X
60 RANDOMISE USR (K)
70 NEXT A
80 FOR A = 1 TO Y*D
90 RANDOMISE USR (K)
100 NEXT A
110 FOR A = 1 TO Z
120 RANDOMISE USR (K)
130 NEXT A
140 NEXT D
```

Run this inputting values for X, Y and Z of 50, 60 and 70. Then experiment with other values. You need to turn the volume of your TV to maximum to hear the "music".

---

## A Few Facts About the ZX80

The ZX80 display, which is *not* memory mapped (so moving graphics are out of the question, unfortunately), can be up to 24 lines of 32 characters each. The computer has its own modulator which produces an RF signal which can be connected via the lead provided to a TV set. The 1 volt composite video signal which feeds the modulator inside

the ZX80 can be taken out, via a wire, and fed via a buffer circuit to a monitor.

1K byte, inside the ZX80, is supplied as standard. Extra RAM must be added externally via the edge connector at the rear. The ZX80 will take a maximum of 16K bytes. The edge connection at the rear of the ZX80 has the following: 8 data, 16 address, 13 control lines from the processor, clock, chip select for internal RAM and OV, 5V stabilised and 9 to 11V unstabilised supply lines.

The data and address lines are not buffered, and there are no I/O ports. Sinclair has plans to introduce an interface to support floppy discs, printer, teletype or I/O bus. There are sockets on the rear of the computer for connection to standard cassette recorders. Leads are provided with 3.5 mm jack plugs on them. The ZX80 loads the whole of the BASIC program into the cassette. You cannot load or fetch data on its own.

Like most BASICS, "Sinclair BASIC" is not compatible with other BASICS, although many programs can be adapted to run on the ZX80. The computer's BASIC interpreter, character set, operating system and monitor are contained in a 4K byte ROM, and works in integer 2 byte arithmetic (with a lower limit of  $-32767$  and an upper limit of  $32767$ ). Arithmetic operators provided are  $-$ ,  $+$ , multiply, divide and raise to the power. With the 4K ROM as supplied, the ZX80 does not handle floating point arithmetic, log, tan and the like, or exponents. An optional plug-in 8K ROM (which fits into the 4K ROM socket) does provide these. It is only available as an extra.

The BASIC instruction set for the ZX80 is as follows:

#### **System Commands**

NEW	Clears ZX80 for new program
RUN	Runs the current program
LIST	Lists the current program
LOAD	Loads a program from tape
SAVE	Saves a program on tape

#### **Control Statements**

GOTO	Shifts control to line number specified
IF THEN	IF a condition is satisfied THEN carry out some other control statement
GOSUB	Go to subroutine specified
STOP	Pretty obvious what this one means
RETURN	This comes at the end of a subroutine, and sends control back to the line <i>after</i> the GOSUB line
FOR..TO	A counter, as in FOR X = 1 TO 10

NEXT	At the end of the counter loop (set up by FOR..TO), sends control back to the FOR...TO line (and X becomes X + 1)
CONTINUE	Allows program execution to continue without losing variable

### **Input/Output Statements**

PRINT	Allows computer to output data on TV screen
INPUT	Allows user to enter data via keyboard

### **Assignment Statement**

LET...	Assigns a value to a variable, as in LET X = 10
--------	---

### **Others**

CLEAR	Clears the stored values of variables
CLS	Clears the screen
DIM	Sets up the size of an array (one-dimensional only, no string arrays)
REM	Remark. This does not make the computer do anything, but is useful in a program to let you know what the following section does, e.g. REM COMPUTES VELOCITY. There is often not enough memory to indulge in REM statements
RANDOMISE	Sets seed of random number generator equal to number of frames of TV since switch-on
POKE	Allows user to talk to computer in binary
PEEK	Peeks at contents of an address
CHR\$	Allows user to print any character
STR\$	Allows integer number of variable to be treated as a string variable
TL\$	Gives the string minus its first character
CODE	Gives the code corresponding to the first character in a string
RND	Provides a pseudo-random number
USR	Machine code subroutine
ABS	The absolute value of a number

# The 8K ROM

The 8K ROM fits into the socket for the 4K ROM and provides a number of additional features for the ZX80. Programs are not upwardly compatible, that is, you cannot LOAD tapes recorded from an old ROM machine into a new ROM. However, they can be typed in again on a new ROM machine practically without alteration (see introduction of this book for the changes needed). The 8K ROM is supplied with an additional manual and a new overlay for the keyboard. The most important features of the 8K ROM are:

- full floating-point arithmetic to 9-digit accuracy
- logs, trigs, and their inverse functions, graph plotting facility
- pseudo-animated graphics using PAUSE n
- full set of string-handling facilities
- n dimensional arrays
- n dimensional string arrays, cassette LOAD and SAVE with named programs.

## Numbers

Stored in 5 bytes in floating point binary form giving  $9 \times 10^{-44}$  to  $1.1 \times 10^{43}$  accurate to  $9\frac{1}{2}$  decimal digit

## Variables

Numeric:	Any letter, followed by alphanumerics.
String:	A\$ — Z\$.
FOR-NEXT:	A — Z.
Numeric arrays:	A — Z.
String arrays:	A\$ — Z\$.

## Arrays

Numeric arrays:	'n' dimension, subscript range starts at 0.
String arrays: (more correctly, character arrays)	'n' dimension, subscript range starts at 0. If the last subscript is omitted it's treated as a fixed length string.

## Strings

Undimensioned strings can be any length.  
Can be concatenated (+).  
Substring eg B\$ = A\$(2 TO 4).  
Literal strings eg C\$ = "QWERTY".



## Statements available

In this list,

- v represents a variable.
- x, y, z represent numerical expressions.
- m, n represent numerical expressions that are rounded to the nearest integer.
- e represents an expression.
- f represents a string valued expression.
- s represents a statement.

Note that arbitrary expressions are allowed everywhere (except for the line number at the beginning of a statement). Thus "GOTO LN A \*\* 2" is valid.

- CLEAR** Deletes all variables, freeing the space they occupied.
- CLS** (Clear Screen) deletes all PRINT output in the display file.
- CONTINUE** Resumes execution of the last run program -- repeats the last statement if an error was detected, otherwise restarts at the next one. Note that a command (immediate execution) statement counts as a program and so destroys the re-entry data.
- DATA...** Standard, but no unquoted strings.
- DIM...** Deletes any array or string with the same name, sets up space for a new array in the usual way, and initialises its elements to 0 or " ".
- DRAW m, n** Let (u, v) be the current PLOT (q.v.) position. Draws a line as straight as possible from (u, v) to (u + m, v + n) by blacking in pixels (quarter character squares). Changes the PLOT and PRINT positions.
- FOR A TO B  
STEP C** Generally standard, but entirely dynamic in its action.
- NEXT** The effect of a NEXT statement is to look up the corresponding FOR-variable, increment its value by the STEP, check whether the limit is exceeded and if not jump to the looping line number.
- GOSUB n** Transfers control to BASIC subroutine.
- GOTO n** Jumps to line n.

IF x THEN s	If x is true (defined to mean greater in absolute value than $2^{-112}$ ) then s is executed. The standard values of true and false as yielded by relational operators are 1 and 0.
INPUT v	<i>Outputs the display file to the screen with no special INPUT prompt; the rest is standard. Cannot be used as a command (immediate execution) statement.</i>
LIST	Lists from start of program.
LIST n	Lists program starting at line n with program cursor pointing at line n.
LOAD f	<i>Looks for a program called f on tape and loads it and its variables.</i>
NEW	Default n = 0. Erases BASIC program and variables.
NEW n	n is used to alter a system variable known as RAM TOP, which is the address of a byte in RAM. The area from RAM TOP on is untouched by the BASIC system, the POKEd programs can be left there in safety.
PAUSE n	Sends the display file to the TV screen for n frames (50 frames per second) or until a key is pressed.
PLOT m, n	Sends the PLOT position (a system variable) to m, n) and blacks in that pixel. Also changes the PRINT position.
POKE m, n	Writes n in byte m in RAM.
PRINT...	Mostly standard. The display file has 22 lines of 32 characters each (2 zones of 16 characters) and when this is filled it is sent to the TV with error 5. CONTINUE carries on with the program with no loss of data.
PRINT AT m, n	Moves the PRINT position to line m, character n.
PRINT TO de	Alters the PRINT format. Here d is an optional digit between 1 and 8 (default value 8) and e is an optional letter E. From now until another such formatting item, numbers will be printed to d significant digits, and if E is present they will always be printed using scientific notation. On switch-on, the format is initialised so that numbers are printed to 8 digits and scientific notation is avoided where possible. Note that PRINT does not change the PLOT position.

RANDOMIZE	Standard.
RANDOMIZE n	If n is given, this is made the value of the seed of the random number generator.
READ v	Reads v for a data statement.
REM...	Remember, for program comments.
RESTORE	Reinitialises the data (so it can be read again).
RETURN	Return from subroutine.
RUN	RUNS the BASIC program.
RUN n	CLEAR followed by GOTO n.
SAVE f	Saves program and variables on tape and calls it f.
SCROLL	Scrolls display file up one line, losing top line and making space at bottom.
STOP	
UNDRAW m, n	These are like DRAW and
UNPLOT m, n	PLOT, but blank out pixels instead of blacking them in.

<b>Functions</b>	<b>Type of Operand</b>	<b>Result</b>
—	number	Negate
ABS	number	Absolute magnitude
ARCOS	number	In Radians
ARCSIN	number	In Radians
ARCTAN	number	In Radians
CHR\$	number	The character whose code is x.
CODE	number	The code of the first character in x (or 0 if x is empty)
COS	number	In radians
EXP	number	$e^x$
INKEY \$	number	Reads the keyboard. The result is a character representing the key pressed, otherwise the empty string.
INT	number	Integer.
LEN	string	The length of x.
LN	number	Natural log

NOT	number	Exclusive — ORs the first byte of x with 113, so that NOT 0 = 1, NOT 1 = 0. Unlike the other functions, NOT has binding power 4 (between AND and the relational operators) so that for instance NOT A = B has the same value as NOT (A = B) (and A < > B).
PEEK	number	The value of the byte in store whose address is x (3.1415927)
PI		A random
RND		number between 0 and 1.
SGN	number	Yields -1, 0, +1.
SIN	number	In Radians.
SQRT	number	Square root.
STR\$	number	The string of characters that would appear on the screen if x were PRINTed.
TAN	number	In Radians.
USR	number	Converts x to an address in store and calls that address as a machine code subroutine. On return, the result is the contents of the h1 register pair.
VAL	string	Evaluates x as a numerical expression (x must not contain the quote image character).
AND		Logical AND
OR		Logical OR

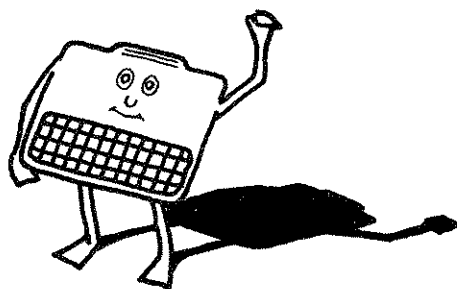
### Relational operators

=	Equal
>	Greater than
<	Less than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal

### Graphics

All characters, their reverses, and all graphics can be entered directly from the keyboard.

AU REVOIR...



... FOR NOW

