



**P E R S O N A L**  
**COMPUTER**  
COMPUTER **NEWS** LIBRARY

**JEAN FROST**

**Instant**  
**ARCADE GAMES**  
**for the**  
**BBC MICRO**

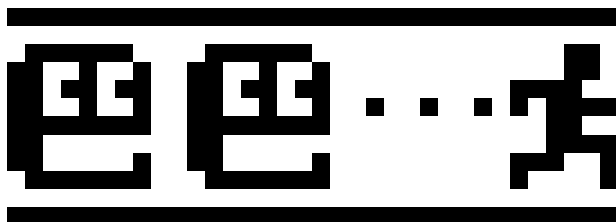


Pan/Personal Computer News  
Computer Library

**Instant Arcade Games**  
for the  
**BBC Micro**

**Jean Frost**

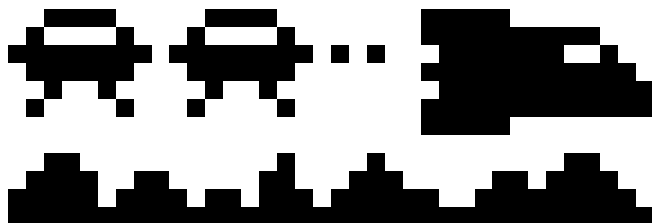
**Pan Books** London and Sydney



First published 1983 by Pan Books Ltd,  
Cavaye Place, London SW10 9PG  
in association with Personal Computer News  
9 8 7 6 5 4 3 2  
(c) Jean Frost 1983  
ISBN 0 330 28266 2

Printed in Great Britain by  
Richard Clay (The Chaucer Press) Ltd,  
Bungay, Suffolk

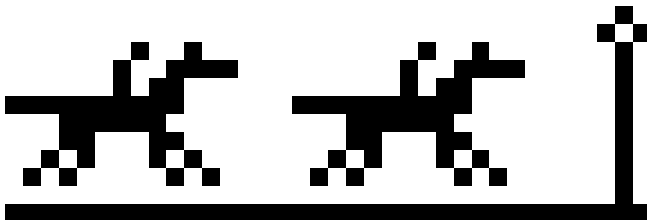
This book is sold subject to the condition that it shall not,  
by way of trade or otherwise, be lent, re-sold,  
hired out, or otherwise circulated without the publisher's prior consent  
in any form of binding or cover other than that in which it is  
published and without a similar condition including  
this condition being imposed on the subsequent purchase.



# CONTENTS

	Page
<b>Preface</b>	<b>5</b>
<b>1 BASIC and Games, Computers and Cheesecakes</b>	<b>7</b>
<b>2 Building Blocks, and Example Construction</b>	<b>17</b>
<b>3 Arcade Games, a Selection of Lego Bricks</b>	<b>39</b>
3.1 Instructions	39
3.2 Backgrounds	42
3.3 Alien Graphics	50
3.4 Player Graphics	64
3.5 Set-up Routines	80
3.6 Movement and Firing	81
3.7 Collision Detection	85
3.8 Explosions	89
3.9 Scoring	93
3.10 Fuel and Animation	95
3.11 Status Display	97
3.12 Check End of Game	99
3.13 End of Game Display	100
<b>4 Starting to Write Your Own Games</b>	<b>103</b>
<b>5 Further Explanations and Understanding BASIC</b>	<b>115</b>
<b>6 Character Graphics</b>	<b>123</b>
<b>7 Arrays and Adventures</b>	<b>132</b>
<b>8 Adventure Games, a Selection of Lego Bricks</b>	<b>136</b>
8.1 Initialisation	136
8.2 Assign Inventories	139
8.3 Instructions	139
8.4 Create the Maze	140
8.5 Describe the Room	143
8.6 Player INPUT	144
8.7 Check INPUT is Legal	145
8.8 Perform Instruction	146
8.9 PRINT Response	147
8.10 Check for End of Game	149
8.11 End of Game Message	149
8.12 Round Again	150
8.13 Food and Strength	161

8.14	Torch and Batteries	164
8.15	Trolls, Run, Fight and Teleport	166
8.16	Monsters and Magic Dust	170
8.17	Crystals and Shimmering Curtains	173
<b>9</b>	<b>Further Adventures</b>	176
<b>10</b>	<b>Some Parting Remarks</b>	185
<b>Appendix One</b>	<b>Arcade game variables</b>	188
<b>Appendix Two</b>	<b>Adventure game variables</b>	190
<b>Appendix Three</b>	<b>ASCII character set</b>	193
<b>Appendix Four</b>	<b>Decimal/Binary conversion tables</b>	196



# PREFACE

This book has been designed to help people with little or no programming knowledge to construct their own arcade and adventure games. In its simplest form this book is like a Lego-set with ready made pieces that can be put together, even by the complete novice, to form a game. For those who wish to expand their understanding of the BASIC programming language each of the building blocks or routines is explained in straight-forward terms. Further sections of the book explain the routines in greater detail. More experienced programmers may also use this book as a source of ready written and documented routines for inclusion, perhaps with alterations, in their own programs.

Many people are left bewildered when they try to make the transition from buying software to writing their own. Books on BASIC are full of obscure jargon and tend to treat a simple, highly practical subject as though it were an arcane, technical concept. The authors here have attempted to redress the balance by presenting a simple introduction to how a program is put together. A worked example of how to construct a program is given and this can be used as a step by step recipe by the readers for creating their own unique programs. This piece by piece, easy to understand approach allows the reader to become familiar with BASIC programs by 'handling' the routines and using them. In this way each small brick in the structure of programming becomes an understandable everyday object. With continued use these 'objects' will become less mysterious and will gradually become as easy to use as building blocks. In much the same way that children learn to speak English by playing with words, and even alphabet blocks, the reader will assimilate and learn the computing terminology and the BASIC language. This approach allows the learning process to be carried out at your own pace and, hopefully, through games you construct, at your leisure.

Obviously this approach also requires that this book be used in the same manner as a manual for mending cars or a recipe book: Keep it beside you as you work and refer to it for instruction at the appropriate points. Do not attempt to read

straight through the book from cover to cover without keying in and trying many of the routines listed. By the time you have looked at and used most of the example programs you should find your confidence in practical BASIC allows you to write your own programs. We hope you enjoy this book and are rewarded for your studies by a growing understanding of BASIC and many hours of fun.

Jean Frost and Jeremiah Jones  
1983

# CHAPTER 1

## **BASIC and Games, Computers and Cheesecake.**

Computer games are becoming a very popular form of entertainment. They are also quite expensive. Many computer owners would like to cut down on this expense by writing their own games and this book is here to help you do just that. How do you write a computer game, then? Well, I think we'd better start by taking a general look at computers and the problems of converting our ideas for games into computer programs.

Computers are really fast operators, and you may think an accountant is pretty nippy punching the keys on his calculator keyboard, but that's just peanuts compared to the speed at which computers work. Unfortunately computers are also dumb. They can't do anything until you tell them what you want doing. So how do we tell a computer to do something? Well, if we really wanted to talk to a computer we would have to speak entirely in numbers, because that's all they understand! Fortunately most computers are supplied with a built-in interpreter. It's just like the Prime Minister using an interpreter to tell the Russians exactly where they can put their missiles. The BASIC interpreter can translate our words into the numbers which the computer understands.

Great, so we can just stroll up to the computer, pat it on the keyboard, and say 'How much profit did I make selling cheesecake today ?' . . . Well it's still not quite that easy, we have to phrase our question very simply and explain everything to the computer in simple and exact terms. The BASIC interpreter can only translate a limited number of words and it assumes that any word it doesn't know is the name of somebody or something. Right, first we need to tell the computer the facts:

The wholesale cost of a cheesecake is 30p each.

The retail price of a cheesecake is 50p each.

We also need to tell the computer things relating to these facts:

Profit is calculated as retail price minus wholesale price. Total profit is profit per cake multiplied by the number of cheesecakes sold.

Hopefully we have now given the computer all the information a child would need to work out the answer. Of course, if we forgot to tell it something the child might say ' you forgot to tell me how many you sold' , and the computer would say something like ' No such variable' which actually means the same thing. Anyway, to tell the computer anything we must express it in BASIC terms. BASIC is like a restricted form of the English language and in BASIC the above statements would be:

```
wholesale = 30
retail = 50
sold = 40
```

(That' s what we forgot to tell it before.)

```
profit = retail - wholesale
total = profit * sold
```

(On computers the asterisk, \*, is used instead of an x for the multiplication sign because we might get the x ' mixed up' in the middle of some letters.)

If you have keyed these lines in exactly as above, pressing RETURN at the end of each line, the computer will have accepted your statements with inscrutable silence, just giving you a ' >' prompt for the next line. The computer now knows what the answer is but because we didn' t ask it to tell us, it just sits there. To get it to tell you the answer tell it to:

```
PRINT total
```

When you press RETURN the answer 800 will appear.

Let' s re-examine how we got the computer to do something. We started with a problem and then split the problem up into parts: We gave the computer the facts, we told it what steps it would need to go through to work out the answer, then we told it to tell us the answer. This sequence of instructions, like a recipe,

is called an *algorithm* and all programs are just such algorithms (methods) written out in BASIC.

If we wish to use the recipe again we would have to type it in again exactly as before. To save ourselves time we can store the instructions as a program on the computer by numbering each line or statement as we type it in:

```
1 wholesale = 30
2 retail = 50
3 sold = 40
4 profit = retail - wholesale
5 total = profit * sold
6 PRINT total
```

This time as we type in each line they will be stored in the memory as a set of instructions, and not executed immediately. This is a computer program (program comes from the word programme and, like the ones you get at the theatre, it shows the order in which things are going to be done.)

When we want the computer to show us what instructions it has got stored we use the BASIC command LIST, followed by RETURN, and the lines of the program will be displayed on the screen once again.

When we want the computer to RUN through the LIST of instructions we simply type RUN, then RETURN. The computer will execute all the instructions in the program and almost instantaneously will PRINT the answer on the screen. Well, that was quicker than working it out in your head, but what a time it took getting the question into a state where the computer could answer it. Type in exactly as shown:

```
3 INPUT "how many cheesecakes did you sell", sold
```

This will replace the old instruction number 3 and tells the computer to ask you how many cheesecakes you sold each time you RUN the program. If you type in a different number to 40 then the computer will give you a different answer. It will always be the right answer and it will be worked out far faster than you could have done it. To save the bother of pressing RUN each time we could add an extra instruction to our program:

## 7 GOTO 3

This simply tells the computer to GO TO instruction 3 and continue from there down the program. Now we have a program that will keep asking you how many you sold and telling you what profits you would make. You will have to press ESCAPE instead of entering a number of cheesecakes to get out of the program.

Our program is numbered in consecutive numbers starting at one. However this isn't usually the case with computer programs. Suppose we really had forgotten the line about the number of cheesecakes. We could have typed in a program that looked like this:

```
1 wholesae = 30
2 retail = 50
3 profit = retail - wholesale
4 total = profit * sold
5 PRINT total
```

If we typed RUN for this set of instructions the computer would say ' No such variable' . Eventually we would realise that it meant it didn't know how many cheesecakes had been sold and we would want to insert the line telling it:

```
3 sold = 40
```

We would now have to retype all the lines except the first two so that there was a gap to put the additional line in. Although the computer goes down the LIST of instructions in order it doesn't care if there are numbers that aren't used. So, we can type our program with line numbers that go up in steps of ten (or any other number). Now any other lines we've missed out can be given a number between the numbers of the lines already there.

```
10 wholesale = 30
20 retail = 50
30 profit = retail - wholesale
40 total = profit * sold
50 PRINT toal
```

Oops! We forgot that line again, but this time we can simply type:

```
25 sold = 40
```

The computer will then place it in the LIST of instructions at the appropriate position. Now we use a special instruction found on your BBC computer, the RENUMBER instruction. When we type this in (without a line number!) the computer changes the numbers of all the lines in the LIST of instructions so that they go up in tens. (It can be used to RENUMBER the program in different ways and the BBC handbook tells you how to do this.) So now our program looks like this:

```
10 wholesale = 30
20 sold = 40
30 retail = 50
40 profit = retail - wholesale
50 total = profit * sold
60 PRINT total
```

This would have worked on the version with the GOTO statement in it as well, because the computer is sensible enough to change the line number given after the GOTO instruction so that it still points to the right place.

OK, so now that we know roughly what a BASIC program is, how do we write one that plays a game? We've got to think of the right LIST of instructions. That sounds like a big job to tackle all at once so we'd better split it up in smaller sub-tasks. Let's start with an overview of the sub-tasks that have to be performed in a typical Arcade-type game.

1. Instructions. We need to tell the player how to control the movement and firing etc.
2. Background. Draw some sort of scene against which the action takes place.
3. Alien graphics. Decide what the opposition look like.
4. Player graphics. Decide what the good guys look like.
5. Set-up routines. Telling the computer the facts.

6. Movement and firing. Move the player and see if a shot is to be fired.

7. Collision detection. See if an alien has been shot or if the player has crashed.

8. Explosions. Blow up the poor unsuspecting alien who is very probably a victim of circumstances beyond his (or its) control!

9. Scoring. Award the player some points for their gleeful destructive ability.

10. Fuel and ammunition. Perhaps we might like to affect the quantities of these available to the player.

11. Status display. Show the player how they are getting on.

12. Check end of game. See if something fatal has happened to the player or if the player has won. We will need to GO TO step 6 if the game isn't over.

13. End of game display. Say goodbye to the player.

Well, that's a lot of things to do. In fact some of them look as if they need large programs just to do one sub-task. BBC BASIC has a two pairs of commands which allow us to treat small programs as though they are sub-tasks or *subroutines* of a larger program. The first pair of commands is GOSUB and RETURN. The first of these, GOSUB, is a lot like the GOTO command and makes the computer jump to the line number given, but the GOSUB command also tells the computer to remember where it came from. Once the computer has obeyed the GOSUB command, and arrived at the specified line number, it carries on down the list of instructions in the subroutine until it encounters the RETURN command. And at this point it goes back whence it came, having performed the sub-task. The second way of doing this is to use the PROCedure commands on the BBC. The set of associated commands, PROC, DEF PROC and ENDPROC, provide us with a way of defining sub-tasks as though they were just BASIC commands. Every time we say PROCfred the computer will look for the instructions which tell it how to do the PROCfred set of commands. These instructions are placed, by us, after the main part of the program, somewhere out of the way. The instructions are started by the command DEF PROCfred (which means DEFINE the PROCedure called fred), and are ended by the command ENDPROC (which marks the END of the PROCedure).

When used in this way PROCedures seem just like SUBroutines. We can, however, also DEFine PROCedures which have *parameters*, just as with mathematical functions. For instance we could say:

```
10 DEF PROCf(X)
20 Y = X*2
30 ENDPROC
```

If we now type in:

```
PROCf(2): PRINT Y
```

we should see that 4 is PRINTed out. The 2 inside the brackets was used by the PROCedure as the value of X. We will be using this type of PROCedure (with parameters) later in this book but for now we shall just stick to using them as though they were subroutines to which we can refer by name. If, after reading this book, you want to know more about PROCedures you will find a detailed explanation on page 230 of the BBC User Guide.

Using these commands we can write a control program which simply says PROCxxxx to make the computer perform each of the appropriate sub-tasks in the right order. All we will need to do then is DEFine the sub-tasks as PROCedures for it to follow and it will achieve our total task of playing a game. The following listing shows just such a control program constructed on the basis of performing the PROCedures as laid out above. This listing, and all the other listings in this book, have been produced by listing them directly from the BBC computer to the printer. This ensures that all the programs in this book are correct, so if you have any trouble check that your typing corresponds exactly with the listings given.

### *Control Program Listing*

```
10 REM *****
20 REM *
30 REM * CONTROL PROGRAM
40 REM *
50 REM *****
```

```

60 MODE 2
70 REM *****
80 REM * INSTRUCTIONS *
90 REM *****
100 PROCinstructions
110 REM *****
120 REM * BACKGROUND *
130 REM *****
140 PROCbackground
150 REM *****
160 REM * GRAPHICS (ALIEN) *
170 REM *****
180 PROCalien
190 REM *****
200 REM * GRAPHICS (SELF) *
210 REM *****
220 PROCplayer
230 REM *****
240 REM * START/RESTART *
250 REM *****
260 PROCstart
270 REM *****
280 REM * MOVE/FIRE *
290 REM *****
300 PROCmovefire
310 REM *****
320 REM * CHECK FOR HIT *
330 REM *****
340 PROCcheck
350 REM *****
360 REM * EXPLOSION *
370 REM *****
380 IF HIT THEN PROCexplode
390 REM *****
400 REM * SCORING *
410 REM *****
420 PROCscore

```

```

430 REM *****
440 REM * FUEL & ANIMATION *
450 REM *****
460 PROCfuel
470 REM *****
480 REM * STATUS DISPLAY *
490 REM *****
500 PROCstatus
510 REM *****
520 REM * END OF GAME? *
530 REM *****
540 PROCfinish
550 REM *****
560 REM * ROUND AGAIN *
570 REM *****
580 IF NOT FINISH THEN 300
590 REM *****
600 REM * GAME OVER *
610 REM *****
620 PROCgameover
630 REM *****
640 REM * START AGAIN *
650 REM *****
660 GOTO 140

```

As you can see there are a lot of lines in the program above that start with the word REM. This means that anything following the REM is just a REMark or REMinder to us humans as to what is going on. The computer ignores these lines completely, but stores them up in the LISTing so that we can understand what' s going on.

If you have only 16k memory on your BBC then you will have to change line 160 to:

```
60 MODE 5
```

You might also find that there is not enough room for the program to RUN if you have some of the options available for the

BBC fitted to your computer, such as a disc drive. If you find this to be the case you should leave out REM lines that only contain asterisks.

In the next chapter there is an example of how to fill the PROCedures for this control program and in the following chapter there is a selection of PROCedures that can be used to make up different games.

## CHAPTER 2

### Building Blocks, an Example Construction

In the previous chapter we got to grips with the fundamentals of BASIC programming. In this chapter we will not be concerning ourselves so much with understanding the programs but more with using them. We will work through an example of how to build up a game by adding subroutines to the control program. It may help you to think of a control program as being like a 'blocks and holes' type of child's toy. To use it you must place one block in each hole to get a complete program. Remember that each block must be the right shape for the hole you are putting it in, although there may be variation in the details, like colour, of the block. In the listing of the control program and in the other listings throughout this book, you will see many lines beginning with REM. These lines are *not* instructions to the computer but are merely REMarks or REMinders of what was intended when we gave the instructions. If you wish to leave these lines out to save typing time it won't make any difference to the computer or the program/game. It will mean however that you will need to take longer to find the section where some particular task is performed should you wish to examine or alter it later. Most people do not have perfect memories so these REMinders can be very important when trying to read the program.

Right then, let's make a game. Obviously the first thing we need to do is type, or LOAD from tape, the control program given in Chapter 1.

Having done this we need to decide what sort of game we wish to produce: What will be the aim of the game? Is it going to be a game of dodging ravenous spiders in the desert, or a game of blasting alien invaders out among the stars? Of course, due to the nature of the system, you could have cowboys shooting at alien flies in the middle of the ocean - whatever appeals to your sense of humour. It's entirely up to you. Whatever you decide the first things to consider are whether or not you are going to shoot at things or dodge, and which modes of movement are available to your player. LEFT, RIGHT, UP, DOWN and even HYPER-space

jumps can be selected. For our example game let' s choose to be a tank holding off alien invaders amongst the stars.

Step 1. Choose the INSTRUCTIONS. Type in the first listing from the instructions section (listing 3.1a, also given below), as this must always be used. Next we must select and add the lines that allow us to move LEFT and RIGHT (listings 3.1b and 3.1c). Finally we add the line which allows us to FIRE (listing 3.1f)

*Listing 3.1a*

```

1000 REM *****
1005 REM *
1010 REM * INSTRUCTIONS *
1015 REM *
1020 REM *****

1025 DEF PROCInstructions
1030 COLOUR 132 : COLOUR 7 : CLS
1035 LEFT=FALSE : RIGHT=FALSE : DOWN=FA
LSE : UP=FALSE
1040 FIRE=FALSE : HYPER=FALSE
1045 VDU 5
1050 MOVE 200,900 : PRINT "INSTRUCTIONS
"
1055 MOVE 200,900 : PRINT "_____
"

1090 MOVE 0,100 : PRINT "PRESS A KEY TO
START"
1095 A$=GET$
1100 ENDPROC

```

*Listing 3.1b*

```

1060 MOVE 0,800 : PRINT "USE Z TO MOVE
LEFT" : LEFT=TRUE

```

*Listing 3.1c*

```

1065 MOVE 0,700 : PRINT "USE X TO MOVE
RIGHT" : RIGHT=TRUE

```

*Listing 3.1f*

```

1080 MOVE 0,400 : PRINT "USE 'SPACE' T
O FIRE" : FIRE=TRUE

```

Step 2. Choose the BACKGROUND. We could select any listing from section 3.2 but since we want an outer space game let's choose the scene with small stars (listing 3.2f)

*Listing 3.2f*

```

1200 REM *****
1205 REM *
1210 REM * BACKGROUND
1215 REM * SMALL STARS
1220 REM *
1225 REM *****
1230 DEF PROCbackground
1235 BACK=0 : GCOL 0,128+BACK : CLG
1240 FOR I=1 TO 60
1245 GCOL 0,RND(15) : PLOT 69,RND(128
0),RND(1024)
1250 NEXT I
1255 ENDPROC

```

Step 3. Choose the ALIEN. We have already decided that we want an alien invader so we' ll use listing 3.3h.

*Listing 3.3h*

```

1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130
1315 REM * ALIEN INVADER
1320 REM *
1325 REM *****
1330 DEF PROCalien
1335 VDU 23,130, 20,28,62,127,62,28,42,
73
1340 ENDPROC

```

Step 4. Choose the PLAYER. We want a tank for our player so we' ll use listing 3.4b. It is worth noting at this point that we could have chosen identical characters for both the alien and our player (e.g. both helicopters, listings 3.3g and 3.4a), but it will look better if they are different.

*Listing 3.4b*

```

1400 REM *****
1405 REM *
1410 REM * DEFINE CHARACTER 131
1415 REM * PLAYER TANK
1420 REM *
1425 REM *****
1430 DEF PROCplayer
1435 VDU 23,131, 16,16,84,124,124,124,6
8,0
1440 ENDPROC

```

Step 5. START/RESTART routine. There is no choice about this one. We need to reset all the variables whether we use them or not, since we don't any 'No such variable' messages stopping our game.

*Listing 3.5*

```

1500 REM *****
1510 REM *
1520 REM * START/RESTART *
1530 REM *
1540 REM *****
1550 DEF PROCstart
1560 ALIENS=10
1570 SCORE=0
1580 AMMO=10
1590 FUEL=10
1600 REM *****
1610 REM * START POSITION FOR PLAYER *
1620 REM *****
1630 YP=31 : XP=640
1640 REM *****
1650 REM * CHANGE Y COORDINATE IF *
1660 REM * IT'S A DODGING GAME *
1670 REM *****
1680 IF NOT FIRE THEN YP=960
1690 MOVE XP,YP : GCOL 0,1 : VDU 131
1700 OXP=XP
1710 OYP=YP
1720 DEAD=TRUE
1730 PAST=0
1740 FINISH=FALSE
1750 ENDPROC

```

Step 6. MOVE/FIRE routine. The main routine from this section must always be typed in, so first of all key in listing 3.6a. We now need the lines which allow us to move LEFT & RIGHT (listing 3.6b). Including this routine will only allow us to move LEFT or RIGHT provided we have added the appropriate line in the INSTRUCTIONS.

*Listing 3.6a*

```

1800 REM *****
1810 REM *
1820 REM * MOVE/FIRE *
1830 REM *
1840 REM *****
1850 DEF PROCmovefire
1860 IF NOT DEAD THEN 1930
1870 YA=990 : XA=RND(1248) : OXA=XA
1880 DEAD=FALSE
1890 GCOL 0,1 : MOVE XA,YA+10 : VDU 130
1900 REM *****
1910 REM * MOVE ALIEN CHARACTER 130 *
1920 REM *****
1930 GCOL 3,1
1940 MOVE OXA,YA+10 : VDU 130 : MOVE XA
,YA : VDU 130 : OXA=XA
1950 YA=YA-10 : IF YA=0 THEN DEAD=TRUE
: PAST=PAST+1 : ALIENS=ALIENS-1
1960 IF DEAD THEN MOVE OXA,YA+10 : VDU
130
1970 XA=XA-20+RND(3)*10
1980 IF NOT FIRE AND XA>XP THEN XA=XA-1
0
1990 IF NOT FIRE AND XA<XP THEN XA=XA+1
0
2000 IF XA>1248 THEN XA=1248
2010 IF XA<0 THEN XA=0
2020 REM *****
2030 REM * MOVE PLAYER CHARACTER 131 *
```

```

2040 REM *****
2120 IF XP<0 THEN XP=0
2130 IF XP>1248 THEN XP=1248
2140 IF YP<31 THEN YP=31
2150 IF YP>991 THEN YP=991
2160 IF OXP=XP AND OYP=YP THEN 2200
2170 GCOL 3,1
2180 MOVE OXP,OYP : VDU 131
2190 MOVE XP,YP : VDU 131
2200 OXP=XP : OYP=YP
2210 IF INKEY(-99) THEN KEY=TRUE ELSE KEY=FALSE
2220 ENDPROC

```

### *Listing 3.6b*

```

2040 REM * LEFT & RIGHT *
2070 IF LEFT AND INKEY(-98) THEN XP=XP-10
2080 IF RIGHT AND INKEY(-67) THEN XP=XP+10

```

Step 7. CHECK FOR HIT routine. We must select a routine which will fire some sort of weapon so we won't use 3.7a. Let's use listing 3.7c, the LASER routine.

### *Listing 3.7c*

```

2400 REM *****
2410 REM * *
2420 REM * CHECK FOR HIT *
2430 REM * LASTER *
2440 REM * *
2450 REM *****
2460 DEF PROCcheck
2470 HIT=FALSE : IF NOT KEY THEN 2540
2480 GCOL3,11 : BX=XP+32 : BY=YP

```

```

2490 MOVE BX,BY : DRAW BX,990
2500 SOUND 0,-15,6,2
2510 T=TIME : REPEAT UNTIL TIME>T+3
2520 MOVE BX,BY : DRAW BX,990
2530 IF ABS(XA-BX+32)<32 THEN HIT=TRUE
2540 ENDPROC

```

Step 8. EXPLOSION routine. When we hit something we want to have some acknowledgement of our success. We will choose a BLIP noise (listing 3.8i) for our LASER. We must also add listing 3.8k because ours is a FIREing game and we need to wipe the alien from the screen when we hit it.

#### *Listing 3.8i*

```

2600 REM *****
2610 REM *
2620 REM * EXPLOSION
2630 REM * BLIP
2640 REM *
2650 REM *****
2660 DEF PROCexplode
2670 FOR I=1 TO 200 STEP 40
2680 SOUND 1,-15,I,2
2690 NEXT I
2750 ENDPROC

```

#### *Listing 3.8k*

```

2700 REM *****
2710 REM * ERASE ALIEN IF YOU SHOT *
2720 REM *****
2730 GCOL 3,1 : MOVE OXA,YA+10 : VDU 13
0
2740 DEAD=TRUE : ALIENS=ALIENS-1

```

Step 9. SCORE routine. When we hit something we also want to

have some increase to our SCORE to tell us what great shots we are. We will choose to give ourselves 10 points for each alien we hit (listing 3.9d).

*Listing 3.9d*

```

3000 REM *****
3010 REM *
3020 REM * SCORE ROUTINE
3030 REM * TEN POINTS
3040 REM *
3050 REM *****
3060 DEF PROCscore
3070 IF NOT FIRE OR NOT HIT THEN 3090
3080 SCORE=SCORE+10
3090 ENDPROC

```

Step 10. FUEL & AMMUNITION routine. We could decide to ignore these completely by using listing 3.10a but instead we will reduce our FUEL, or energy reserves, each time we fire our laser. So we start with listing 3.10b and add listing 3.10f to decrease the FUEL. We also want to increase the fuel when we score a hit and we' ll be generous and give ourselves a total ~~re~~FUEL for each hit (listing 3.10h)

*Listing 3.10b*

```

3200 REM *****
3210 REM *
3220 REM * FUEL & ANIMATION
3230 REM *
3240 REM *****
3250 DEF PROCfuel
3420 ENDPROC

```

*Listing 3.10f*

```

3340 REM *****
3350 REM * DECREASE FUEL

```

```

3360 REM *****
3370 IF KEY THEN FUEL=FUEL-1

```

*Listing 3.10h*

```

3380 REM *****
3390 REM * RESET FUEL *
3400 REM *****
3410 IF HIT THEN FUEL=10

```

Step 11. STATUS DISPLAY routine. We need to update the display of information on the screen. For the SCORE we will need to use listing 3.11b. If we decide to tell the player how his fuel situation is then we can add listing 3.11d, although you can leave it out if you' re feeling cruel.

*Listing 3.11b*

```

3500 REM *****
3505 REM * *
3510 REM * STATUS DISPLAY *
3515 REM * *
3520 REM *****
3525 DEF PROCstatus
3530 VDU 4 : COLOUR 128+BACK : COLOUR 7
3535 REM *****
3540 REM * DISPLAY SCORE *
3545 REM *****
3550 PRINT TAB(5,0); "SCORE:"; SCORE

```

*Listing 3.11d*

```

3575 REM *****
3580 REM *   DISPLAY FUEL                      *
3585 REM *****
3590 PRINT TAB(15,0); "F: "; FUEL; "  "
3595 VDU 5: ENDPROC

```

Step 12. CHECK FOR END. Now we need to decide what conditions are going to indicate the end of the game. Obviously we will want to end the game at some time so we start with listing 3.12a and add at least one of the other listings from that section. For our game let' s choose to end the game if we let 3 of the opposition get PAST or if we run out of FUEL. So we have to add listings 3.12d and 3.12e.

*Listing 3.12a*

```

3600 REM *****
3610 REM *                                     *
3620 REM *   CHECKS FOR END                  *
3630 REM *                                     *
3640 REM *****
3650 DEF PROCfinish
3710 ENDPROC

```

*Listing 3.12d*

```

3680 IF PAST=3 THEN FINISH=TRUE

```

*Listing 3.12e*

```

3690 IF FUEL=0 THEN FINISH=TRUE

```

Step 13. END OF GAME DISPLAY. It it' s the end of the game we need to either stop the game or ask the player if he wants another go. Let' s choose the POLITE STOP routine (listing 3.13b) and, since we have a SCORE, we' ll also add listing 3.13c to tell us what SCORE was achieved.

*Listing 3.13b*

```

3800 REM *****
3810 REM *
3820 REM * GAME OVER
3830 REM * POLITE STOP
3840 REM *
3850 REM *****
3860 DEF PROCgameover
3870 GCOL 0,132 : GCOL 0,7 : CLG
3890 MOVE 300,500 : PRINT "ANOTHER GO ?
"
3900 IF INKEY(-69) THEN 3930
3910 IF NOT INKEY(-86) THEN 3900
3920 VDU 4,20,12 : END
3930 ENDPROC

```

*Listing 3.13c*

```

3880 MOVE 100,650 : PRINT "YOUR SCORE W
AS ";SCORE

```

And that's it! Your final listing should look like the program below. If it doesn't then make sure you've typed in all the listings with the right line numbers. Now all you have to do is press RUN and RETURN to play your game! GOOD SHOOTING!

If you have any problems or error messages then go back through your listing and check that it corresponds exactly with the one below. The error message will tell you where the problem was noticed and it would be a good idea to look there first. Since the program is split up into sections you can check through just one section at a time. Of course the error might be in one of the previous sections and only have been noticed now. So, if you can't see an error where it says, try working backwards. Remember that *anything* out of place or mistyped will alter the program and could cause it to stop.

*Example Program*

```

10 REM *****
20 REM *
30 REM * CONTROL PROGRAM
40 REM *
50 REM *****
60 MODE 2
70 REM *****
80 REM * INSTRUCTIONS
90 REM *****
100 PROCInstructions
110 REM *****
120 REM * BACKGROUND
130 REM *****
140 PROCbackground
150 REM *****
160 REM * GRAPHICS (ALIEN)
170 REM *****
180 PROCalien
190 REM *****
200 REM * GRAPHICS (SELF)
210 REM *****
220 PROCplayer
230 REM *****
240 REM * START/RESTART
250 REM *****
260 PROCstart
270 REM *****
280 REM * MOVE/FIRE
290 REM *****
300 PROCmovefire
310 REM *****
320 REM * CHECK FOR HIT
330 REM *****
340 PROCcheck
350 REM *****

```

```

360 REM * EXPLOSION *
370 REM *****
380 IF HIT THEN PROCexplode
390 REM *****
400 REM * SCORING *
410 REM *****
420 PROCscore
430 REM *****
440 REM * FUEL & ANIMATION *
450 REM *****
460 PROCfuel
470 REM *****
480 REM * STATUS DISPLAY *
490 REM *****
500 PROCstatus
510 REM *****
520 REM * END OF GAME? *
530 REM *****
540 PROCfinish
550 REM *****
560 REM * ROUND AGAIN *
570 REM *****
580 IF NOT FINISH THEN 300
590 REM *****
600 REM * GAME OVER *
610 REM *****
620 PROCgameover
630 REM *****
640 REM * START AGAIN *
650 REM *****
660 GOTO 140
1000 REM *****
1005 REM * *
1010 REM * INSTRUCTIONS *
1015 REM * *
1020 REM *****
1025 DEF PROCinstructions

```

```

1030 COLOUR 132 : COLOUR 7 : CLS
1035 LEFT=FALSE : RIGHT=FALSE : DOWN=FA
LSE : UP=FALSE
1040 FIRE=FALSE : HYPER=FALSE
1045 VDU 5
1050 MOVE 200,900 : PRINT "INSTRUCTIONS
"
1055 MOVE 200,900 : PRINT "_____
"
1060 MOVE 0,800 : PRINT "USE Z TO MOVE
LEFT" : LEFT=TRUE
1065 MOVE 0,700 : PRINT "USE X TO MOVE
RIGHT" : RIGHT=TRUE
1080 MOVE 0,400 : PRINT "USE 'SPACE' T
O FIRE" : FIRE=TRUE
1090 MOVE 0,100 : PRINT "PRESS A KEY TO
START"
1095 A$=GET$
1100 ENDPROC
1200 REM *****
1205 REM *
1210 REM * BACKGROUND *
1215 REM * SMALL STARS *
1220 REM *
1225 REM *****
1230 DEF PROCbackground
1235 BACK=0 : GCOL 0,128+BACK : CLG
1240 FOR I=1 TO 60
1245 GCOL 0,RND(15) : PLOT 69,RND(128
0),RND(1024)
1250 NEXT I
1255 ENDPROC
1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130 *
1315 REM * ALIEN INVADER *
1320 REM *

```

```

1325 REM *****
1330 DEF PROCalien
1335 VDU 23,130, 20,28,62,127,62,28,42,
73
1340 ENDPROC
1400 REM *****
1405 REM *
1410 REM * DEFINE CHARACTER 131
1415 REM * PLAYER TANK
1420 REM *
1425 REM *****
1430 DEF PROCplayer
1435 VDU 23,131, 16,16,84,124,124,124,6
8,0
1440 ENDPROC
1500 REM *****
1510 REM *
1520 REM * START/RESTART
1530 REM *
1540 REM *****
1550 DEF PROCstart
1560 ALIENS=10
1570 SCORE=0
1580 AMMO=10
1590 FUEL=10
1600 REM *****
1610 REM * START POSITION FOR PLAYER *
1620 REM *****
1630 YP=31 : XP=640
1640 REM *****
1650 REM * CHANGE Y COORDINATE IF
1660 REM * IT'S A DODGING GAME
1670 REM *****
1680 IF NOT FIRE THEN YP=960
1690 MOVE XP,YP : GCOL 0,1 : VDU 131
1700 OXP=XP
1710 OYP=YP

```

```

1720 DEAD=TRUE
1730 PAST=0
1740 FINISH=FALSE
1750 ENDPROC
1800 REM *****
1810 REM * *
1820 REM * MOVE/FIRE *
1830 REM * *
1840 REM *****
1850 DEF PROCmovefire
1860 IF NOT DEAD THEN 1930
1870 YA=990 : XA=RND(1248) : OXA=XA
1880 DEAD=FALSE
1890 GCOL 0,1 : MOVE XA,YA+10 : VDU 130
1900 REM *****
1910 REM * MOVE ALIEN CHARACTER 130 *
1920 REM *****
1930 GCOL 3,1
1940 MOVE OXA,YA+10 : VDU 130 : MOVE XA
,YA : VDU 130 : OXA=XA
1950 YA=YA-10 : IF YA=0 THEN DEAD=TRUE
: PAST=PAST+1 : ALIENS=ALIENS-1
1960 IF DEAD THEN MOVE OXA,YA+10 : VDU
130
1970 XA=XA-20+RND(3)*10
1980 IF NOT FIRE AND XA>XP THEN XA=XA-1
0
1990 IF NOT FIRE AND XA<XP THEN XA=XA+1
0
2000 IF XA>1248 THEN XA=1248
2010 IF XA<0 THEN XA=0
2020 REM *****
2030 REM * MOVE PLAYER CHARACTER 131 *
2040 REM * LEFT & RIGHT *
2060 REM *****
2070 IF LEFT AND INKEY(-98) THEN XP=XP-
10

```

```

2080 IF RIGHT AND INKEY(-67) THEN XP=XP
+10
2120 IF XP<0 THEN XP=0
2130 IF XP>1248 THEN XP=1248
2140 IF YP<31 THEN YP=31
2150 IF YP>991 THEN YP=991
2160 IF OXP=XP AND OYP=YP THEN 2200
2170 GCOL 3,1
2180 MOVE OXP,OYP : VDU 131
2190 MOVE XP,YP : VDU 131
2200 OXP=XP : OYP=YP
2210 IF INKEY(-99) THEN KEY=TRUE ELSE K
EY=FALSE
2220 ENDPROC
2400 REM *****
2410 REM *
2420 REM * CHECK FOR HIT
2430 REM * LASTER
2440 REM *
2450 REM *****
2460 DEF PROCcheck
2470 HIT=FALSE : IF NOT KEY THEN 2540
2480 GCOL3,11 : BX=XP+32 : BY=YP
2490 MOVE BX,BY : DRAW BX,990
2500 SOUND 0,-15,6,2
2510 T=TIME : REPEAT UNTIL TIME>T+3
2520 MOVE BX,BY : DRAW BX,990
2530 IF ABS(XA-BX+32)<32 THEN HIT=TRUE
2540 ENDPROC
2600 REM *****
2610 REM *
2620 REM * EXPLOSION
2630 REM * BLIP
2640 REM *
2650 REM *****
2660 DEF PROCexplode
2670 FOR I=1 TO 200 STEP 40

```

```

2680     SOUND 1,-15,1,2
2690 NEXT I
2700 REM *****
2710 REM * ERASE ALIEN IF YOU SHOT *
2720 REM *****
2730 GCOL 3,1 : MOVE OXA,YA+10 : VDU 13
0
2740 DEAD=TRUE : ALIENS=ALIENS-1
2750 ENDPROC
3000 REM *****
3010 REM * *
3020 REM * SCORE ROUTINE *
3030 REM * TEN POINTS *
3040 REM * *
3050 REM *****
3060 DEF PROCscore
3070 IF NOT FIRE OR NOT HIT THEN 3090
3080 SCORE=SCORE+10
3090 ENDPROC
3200 REM *****
3210 REM * *
3220 REM * FUEL & ANIMATION *
3230 REM * *
3240 REM *****
3250 DEF PROCfuel
3340 REM *****
3350 REM * DECREASE FUEL *
3360 REM *****
3370 IF KEY THEN FUEL=FUEL-1
3380 REM *****
3390 REM * RESET FUEL *
3400 REM *****
3410 IF HIT THEN FUEL=10
3420 ENDPROC
3500 REM *****
3505 REM * *
3510 REM * STATUS DISPLAY *

```

```

3515 REM *
3520 REM *****
3525 DEF PROCstatus
3530 VDU 4 : COLOUR 128+BACK : COLOUR 7
3535 REM *****
3540 REM * DISPLAY SCORE *
3545 REM *****
3550 PRINT TAB(5,0);"SCORE:";SCORE
3575 REM *****
3580 REM * DISPLAY FUEL *
3585 REM *****
3590 PRINT TAB(15,0);"F:";FUEL;" "
3595 VDU 5: ENDPROC
3600 REM *****
3610 REM *
3620 REM * CHECKS FOR END *
3630 REM *
3640 REM *****
3650 DEF PROCfinish
3680 IF PAST=3 THEN FINISH=TRUE
3690 IF FUEL=0 THEN FINISH=TRUE
3710 ENDPROC
3800 REM *****
3810 REM *
3820 REM * GAME OVER *
3830 REM * POLITE STOP *
3840 REM *
3850 REM *****
3860 DEF PROCgameover
3870 GCOL 0,132 : GCOL 0,7 : CLG
3880 MOVE 100,650 : PRINT "YOUR SCORE W
AS ";SCORE
3890 MOVE 300,500 : PRINT "ANOTHER GO ?
"
3900 IF INKEY(-69) THEN 3930
3910 IF NOT INKEY(-86) THEN 3900
3920 VDU 4,20,12 : END

```

3930 ENDPROC

## SAVING YOUR PROGRAM

When you are satisfied with your game you will probably want to SAVE it on cassette tape. This is done in the following way:

1. Position the tape in your cassette recorder, making sure that you have wound it past any plastic leader tape since this cannot be recorded on.
2. Decide on a name for your program - you can use up to ten letters. For the purpose of this example let' s use the name "my game".
3. Now key in, without a line number:

SAVE "my game"

and press RETURN.

4. The screen should now show the following message:

RECORD then RETURN

This means press PLAY and RECORD on your cassette recorder, and when it' s going, press the RETURN key on your BBC. If by any chance you press the key before you set your cassette recorder going, don' t panic. Just hold down the ESCAPE key and go back to step 3.

5. You should now see the following happen; the name of your program will appear followed by 00, and as the program is saved these two figures will change. Don' t get worried if they change into letters, that' s because they are in *hexadecimal* notation, as explained on page 71 of the BBC User Guide. After a period of these numbers counting up, there will appear a four digit number following the counter, and a few seconds later there will be a beep and the computer' s ' > ' prompt will reappear on the bottom of the screen. Stop the cassette recorder.

6. Now you need to check that your program has been recorded properly. Rewind the tape and key in \*CAT.

Now press PLAY on the cassette. The same series of numbers as before should appear after the name of the program as the tape

is played, but this time the computer will keep on looking for more programs until you press 'ESCAPE' , so do this after the final four digit number is displayed.

If the name of your program does not appear you will have to start again from step 1 because your program has not been SAVED properly. If you still cannot get your program to SAVE, check that the volume level (and tone level, if there is one) are set correctly on your cassette recorder. If you still have trouble recording, try removing the ear connection whilst you are recording as some cassette recorders suffer from feedback. Remember to reconnect this so that the computer can hear when you are replaying the tape to check it.

7. To LOAD the program from the tape into memory, type either of the following:

LOAD "my game"

or:

LOAD ""

The second statement will LOAD the first program it comes to on the tape. If the program will not LOAD, check you have spelt the names right, assuming you are using the first version, or that the controls of your recorder are positioned correctly.

# CHAPTER 3

## **Arcade Games, a Selection of Lego Bricks**

This chapter is made up of thirteen sub-sections. Each of these sections contains a selection of routines which can be used to fill the 'holes' in the control program. Remember to read the instructions at the start of each section which tells you whether listings are optional or not. If you don't feel too confident, why not follow the example routine and just use one different procedure. Try a different background, or a different alien, for example, and you'll see how easy it is to make changes to the game. The more routines you decide to change the more your game will differ. Remember, that is as long as you don't try to fit a square peg into a round hole you can use the routines in any combination. Some games may look a little bizarre, but at least they'll still be playable!

Following each section of listings is a brief explanation of how the routines in that section work. This includes details of what variables they change and how they relate to the other routines in the overall program.

We hope you enjoy experimenting with these routines as much as we enjoyed writing and testing them.

### Section 1: INSTRUCTIONS

In this section 3.1a always needs to be used. This performs the basic tasks of getting ready to give instructions. Add to this any or all of the following listings 3.1b-g to complete the instructions block. It doesn't matter in what order you add the lines, because the computer will place them in numerical order in its listing.

```

1000 REM *****
1005 REM *
1010 REM * INSTRUCTIONS
1015 REM *
1020 REM *****
1025 DEF PROCInstructions
1030 COLOUR 132 : COLOUR 7 : CLS
1035 LEFT=FALSE : RIGHT=FALSE : DOWN=FA
LSE : UP=FALSE
1040 FIRE=FALSE : HYPER=FALSE
1045 VDU 5
1050 MOVE 200,900 : PRINT "INSTRUCTIONS
"
1055 MOVE 200,900 : PRINT "_____
"
1090 MOVE 0,100 : PRINT "PRESS A KEY TO
START"
1095 A$=GET$
1100 ENDPROC

```

This listing gives the instructions for, and enables, LEFT movement. You would normally include the RIGHT movement listing as well but it's up to you.

### *Listing 3.1b*

```

1060 MOVE 0,800 : PRINT "USE Z TO MOVE
LEFT" : LEFT=TRUE

```

This listing gives the instructions for, and enables, RIGHT movement.

### *Listing 3.1c*

```

1065 MOVE 0,700 : PRINT "USE X TO MOVE
RIGHT" : RIGHT=TRUE

```

This listing gives the instruction for, and enables DOWNward

movement. You would normally include the UPward movement listing as well, but again, it' s up to you.

*Listing 3.1d*

```
1070 MOVE 0,600 : PRINT "USE / TO MOVE
DOWN" : DOWN=TRUE
```

This listing gives the instructions for, and enables, UPward movement.

*Listing 3.1e*

```
1075 MOVE 0,500 : PRINT "USE : TO MOVE
UP" : UP=TRUE
```

This listing gives instructions for, and enables, FIREing.

*Listing 3.1f*

```
1080 MOVE 0,400 : PRINT "USE 'SPACE' T
O FIRE" : FIRE=TRUE
```

This listing gives instructions for, and enables, HYPER-drive for HYPER-space movement.

*Listing 3.1g*

```
1085 MOVE 0,300 : PRINT "USE H FOR HYPE
RSPACE" : HYPER=TRUE
```

The main instruction routine (listing 3.1a) sets the background and foreground colour and instructs the BBC to PRINT text at the graphics cursor with the VDU 5 command. It then PRINTs up the heading INSTRUCTIONS in WHITE and underlines it. It also sets a number of variables that are used to indicate whether a particular ability is enabled. These variables (LEFT, RIGHT, DOWN, UP, FIRE and HYPER) are all set to FALSE.

The other routines, when present, will be inserted in the list of instructions at this point. Each of these will PRINT a message on

the screen telling you how to activate some action and also reset the appropriate flag variable set to TRUE. These flags are examined in other parts of the program to see whether the player is allowed to perform a particular action.

The final three lines of the main routine PRINT the message at the bottom of the screen and then wait until a key is depressed before ENDING the PROCEDURE and returning to the control program.

## Section 2: BACKGROUNDS.

This is where the fun starts: Choosing a background for your game. All the following routines set up a background scene against which your game will be played. This is fairly important, especially when it comes to dodging games. A skier switching back and forth through pines whilst dodging spiders is going to have a harder time than one on a totally black background with no stationary objects to avoid. You need to choose just one listing from this section and it can be any of the eight routines presented. Each of the listings, apart from the first (totally black), is accompanied by a black and white representation of an example scene produced by that listing. This should help you decide which one to use.

### TOTALLY BLACK

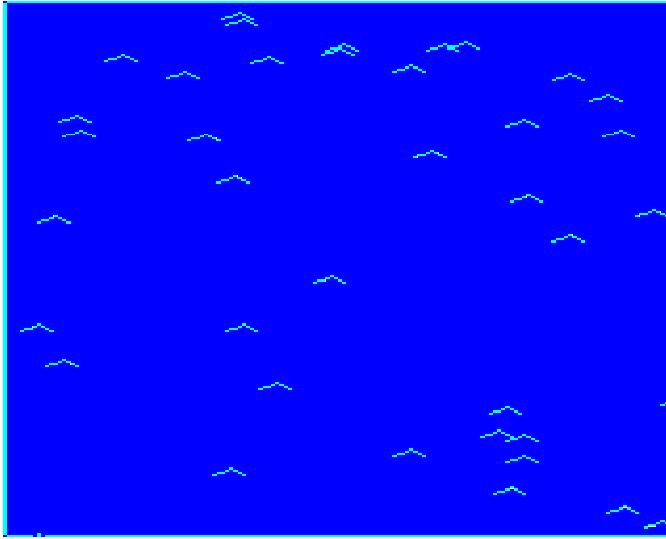
#### *Listing 3.2a*

```

1200 REM *****
1205 REM *
1210 REM * BACKGROUND
1215 REM * TOTALLY BLACK
1220 REM *
1225 REM *****
1230 DEF PROCbackground
1235 BACK=0
1240 GCOL 0,128+BACK : CLG
1245 ENDPROC

```

## SEA-SCAPE

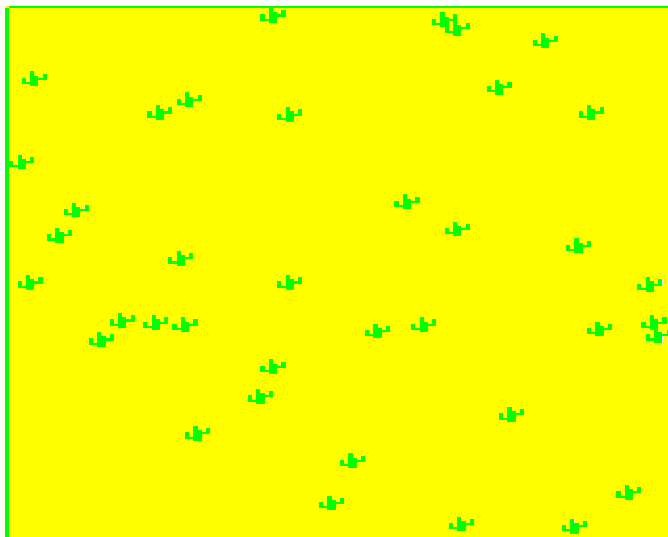
*Fig. 3.2b**Listing 3.2b*

```

1200 REM *****
1205 REM *
1210 REM * BACKGROUND
1215 REM * SEA-SCAPE
1220 REM *
1225 REM *****
1230 DEF PROCbackground
1235 BACK=4 : GCOL 0,128+BACK : CLG
1240 VDU 23,128, 0,0,0,0,8,20,98,129
1245 GCOL 0,6
1250 FOR I=1 TO 40
1255 MOVE RND(1280),RND(1024) : PRINT
CHR$(128)
1260 NEXT I
1265 ENDPROC

```

## DESERT

*Fig. 3.2c**Listing 3.2c*

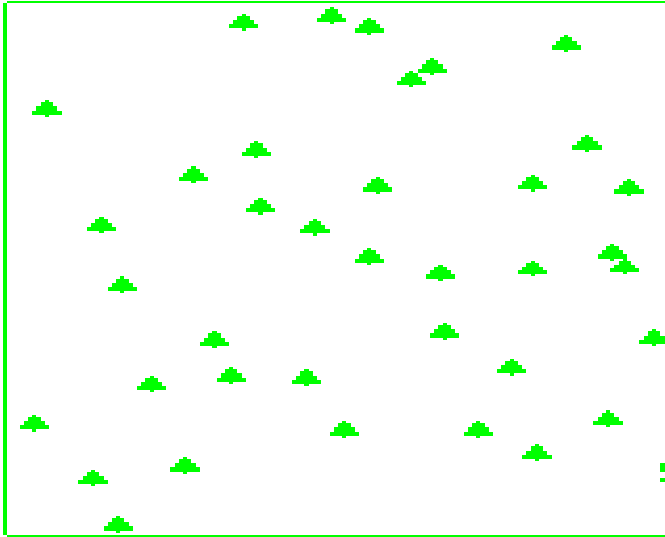
```

1200 REM *****
1205 REM *
1210 REM * BACKGROUND
1215 REM * DESERT
1220 REM *
1225 REM *****
1230 DEF PROCbackground
1235 BACK=3 : GCOL 0,128+BACK : CLG
1240 VDU 23,128, 0,16,18,26,94,88,120,24

1245 GCOL 0,2
1250 FOR I=1 TO 40
1255 MOVE RND(1280),RND(1024) : PRINT
CHR$(128)
1260 NEXT I
1265 ENDPROC

```

## SKI-SLOPE

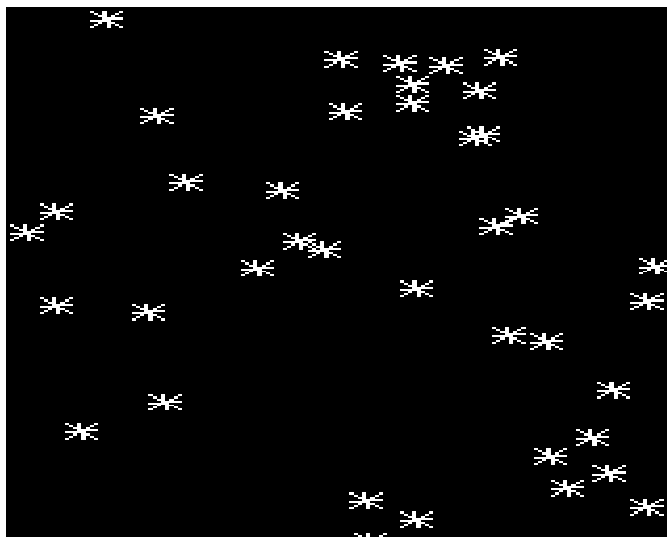
*Fig. 3.2d**Listing 3.2d*

```

1200 REM *****
1205 REM *
1210 REM * BACKGROUND
1215 REM * SKI-SLOPE
1220 REM *
1225 REM *****
1230 DEF PROCbackground
1235 BACK=7 : GCOL 0,128+BACK : CLG
1240 VDU 23,128,
16,56,56,124,124,254,254,16
1245 GCOL 0,2
1250 FOR I=1 TO 40
1255 MOVE RND(1280),RND(1024) : PRINT
CHR$(128)
1260 NEXT I
1265 ENDPROC

```

## LARGE STARS

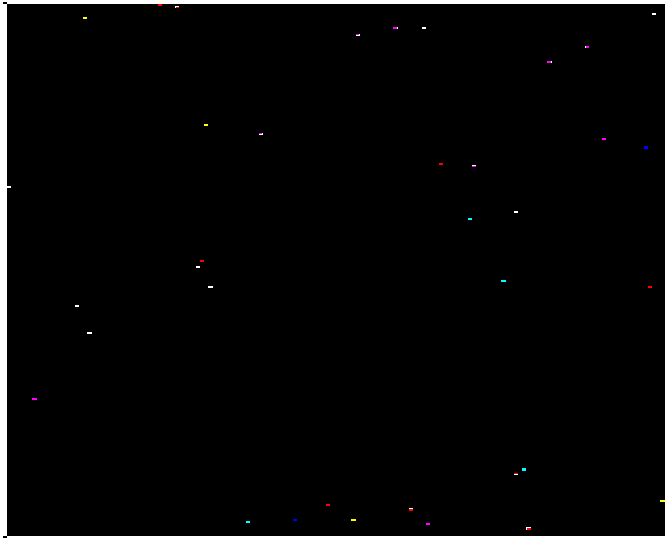
*Fig. 3.2e*

```

1200 REM *****
1205 REM *
1210 REM * BACKGROUND
1215 REM * LARGE STARS
1220 REM *
1225 REM *****
1230 DEF PROCbackground
1235 BACK=0 : GCOL 0,128+BACK : CLG
1240 VDU 23,128, 145,82,52,31,248,44,74
,137
1245 FOR I=1 TO 40
1250 MOVE RND(1280),RND(1024) : GCOL
0,RND(7) : PRINT CHR$(128)
1255 NEXT I
1260 ENDPROC

```

## SMALL STARS

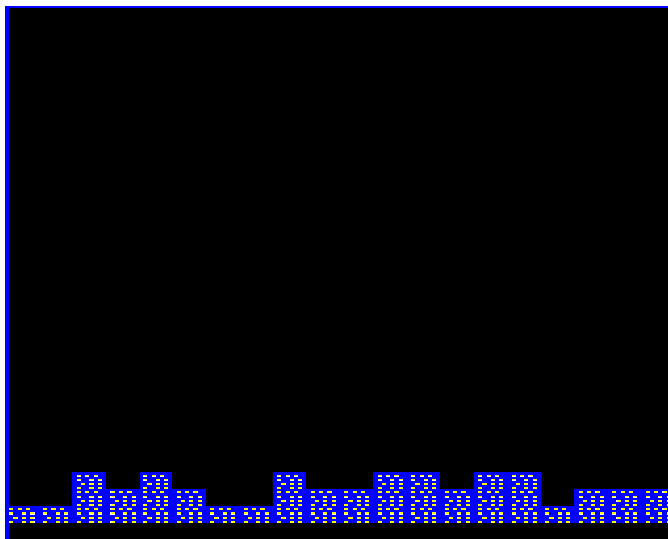
*Fig. 3.2f*

```

1200 REM *****
1205 REM *
1210 REM * BACKGROUND
1215 REM * SMALL STARS
1220 REM *
1225 REM *****
1230 DEF PROCbackground
1235 BACK=0 : GCOL 0,128+BACK : CLG
1240 FOR I=1 TO 40
1245 GCOL 0,RND(15) : PLOT 69,RND(128
0),RND(1024)
1250 NEXT I
1255 ENDPROC

```

## NIGHT SKYLINE

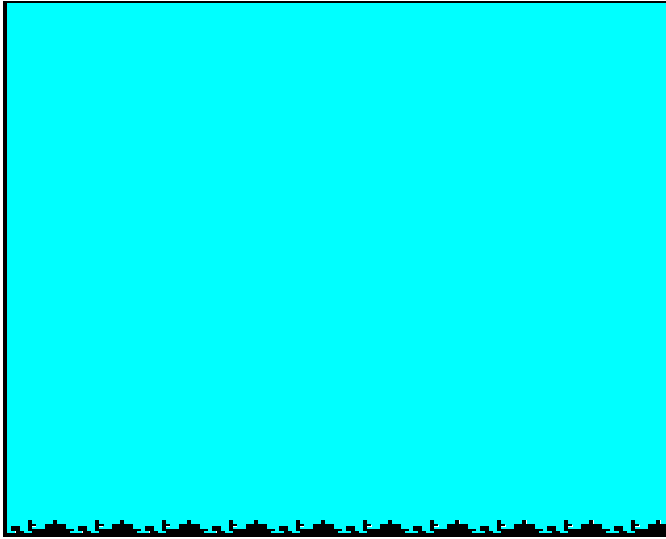
*Fig. 3.2g**Listing 3.2g*

```

1200 REM *****
1205 REM *
1210 REM * BACKGROUND
1215 REM * NIGHT SKYLINE
1220 REM *
1225 REM *****
1230 DEF PROCbackground
1235 GCOL 0,128 : GCOL 0,4 : CLG
1240 VDU 23,128, 255,171,255,181,255,21
3,255,181
1245 GCOL 0,131
1250 FOR I=0 TO 1279 STEP 64
1255     FOR J=64 TO RND(4)*32 STEP 32
1260         MOVE I,J : VDU 32,127,128
1265     NEXT J
1270 NEXT I
1275 ENDPROC

```

## CITY SKYLINE

*Listing 3.2h**Listing 3.2h*

```

1200 REM *****
1205 REM *
1210 REM * BACKGROUND
1215 REM * CITY SKYLINE
1220 REM *
1225 REM *****
1230 DEF PROCbackground
1235 BACK=6 : GCOL 0,128+BACK : CLG
1240 VDU 23,128, 2,2,3,50,179,25,255,25
5
1245 VDU 23,129, 8,8,62,62,255,255,255,
255
1250 GCOL 0,0 : MOVE 0,31
1255 FOR I=1 TO 10
1260 VDU 128,129
1265 NEXT I
1270 ENDPROC

```

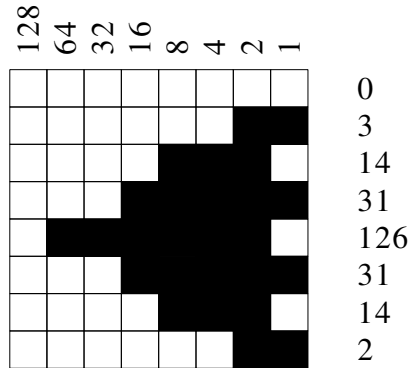
All the background routines set the background colour with GCOL and then sue CLG to clear the screen of this colour. In the first routine (listing 3.2a) this is all that is required. For the rest of the routines a user-defined character for a suitable object is specified (for further explanation of this see Chapter 6). This object is then PRINTed on the screen in various RaNDom positions (see Chapter 10). Listing 3.2f differs from this general pattern by merely PLOTting points (the 69 after the PLOT means PLOT one point) at these RaNDom positions instead of PRINTing a character each time. Listings 3.2g and 3.2h use defined characters but incorporate special methods of determining the positions where these will be placed to achieve the desired effect in each case. These routines are examined separately, and in greater detail, in Chaopter 4.

### Section 3: ALIEN GRAPHICS.

Choosing a shape for the aliens is very simple. Any of the following routines will define a character to use for the aliens. Each routine is accompanied by a display which shows the character' s shape enlarged. It is important to remember at this stage that you will want the aliens to look different from the player. You need to choose one listing from this section and it can be any one of the routines below.

## ALIEN FIGHTER

Fig. 3.3a



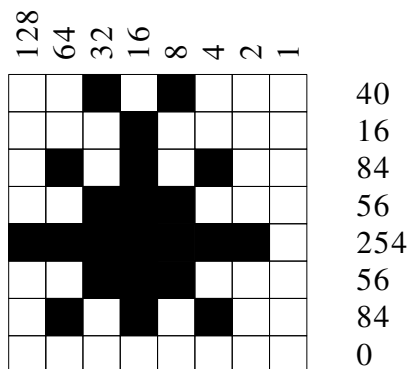
Listing 3.3a

```

1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130
1315 REM * ALIEN FIGHTER
1320 REM *
1325 REM *****
1330 DEF PROCalien
1335 VDU 23,130, 0,3,14,31,126,31,14,3
1340 ENDPROC

```

## ALIEN FROG

*Fig. 3.3b**Listing 3.3b*

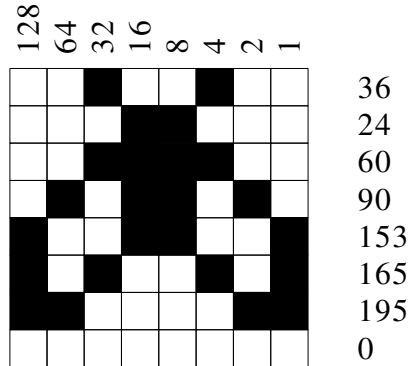
```

1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130
1315 REM * ALIEN BUG
1320 REM *
1325 REM *****
1330 DEF PROCalien
1335 VDU 23,130, 40,16,84,56,254,56,84,
0
1340 ENDPROC

```

## ALIEN FLY

Fig. 3.3c



Listing 3.3c

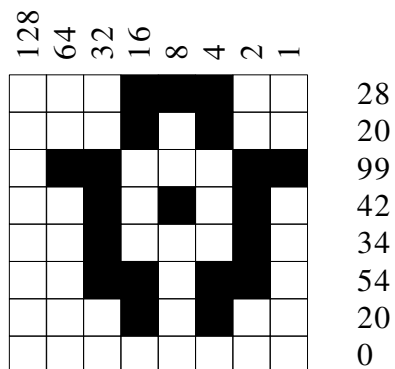
```

1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130 *
1315 REM * ALIEN FLY *
1320 REM *
1325 REM *****
1330 DEF PROCalien
1335 VDU 23,130, 36,24,60,90,153,165,19
5,0
1340 ENDPROC

```

## ALIEN DESTROYER

*Fig. 3.3d*



*Listing 3.3d*

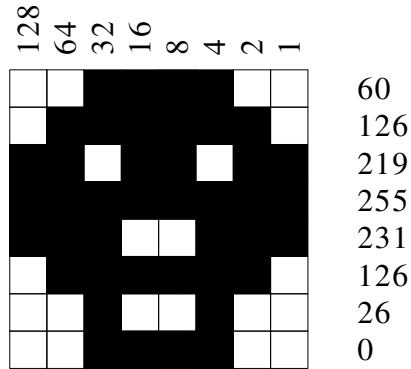
```

1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130
1315 REM * ALIEN DESTROYER
1320 REM *
1325 REM *****
1330 DEF PROCalien
1335VDU 23,130, 28,20,99,42,34,54,20,0
1340ENDPROC

```

## ALIEN SKULL

Fig. 3.3e



Listing 3.3e

```

1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130
1315 REM * ALIEN SKULL
1320 REM *
1325 REM *****
1330 DEF PROCalien
1335 VDU 23,130, 60,126,219,255,231,126
,36,60
1340 ENDPROC

```

## ALIEN INSECT

*Fig 3.3f**Listing 3.3f*

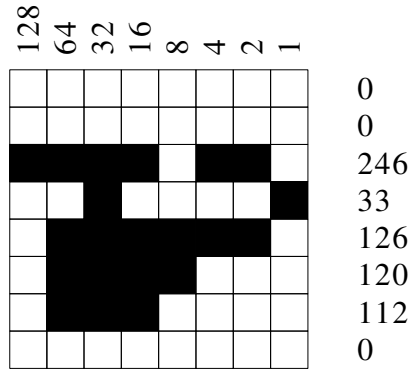
```

1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130
1315 REM * ALIEN INSECT
1320 REM *
1325 REM *****
1330 DEF PROCalien
1335 VDU 23,130, 129,66,36,24,24,60,66,
129
1340 ENDPROC

```

## ALIEN HELICOPTER

Fig. 3.3g



Listing 3.3g

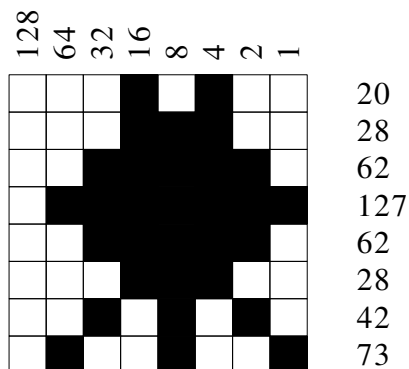
```

1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130
1315 REM * ALIEN HELICOPTER
1320 REM *
1325 REM *****
1330 DEF PROCalien
1335 VDU 23,130, 0,0,246,33,126,120,112
,0
1340 ENDPROC

```

## ALIEN INVADER

Fig. 3.3h



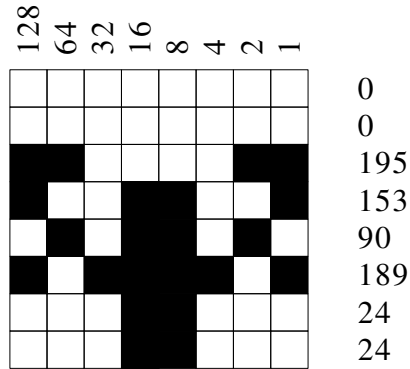
Listing 3.3h

```

1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130
1315 REM * ALIEN INVADER
1320 REM *
1325 REM *****
1330 DEF PROCalien
1335 VDU 23,130, 20,28,62,127,62,28,42,
73
1340 ENDPROC

```

## ALIEN BATTLESHIP

*Fig. 3.3i**Listing 3.3i*

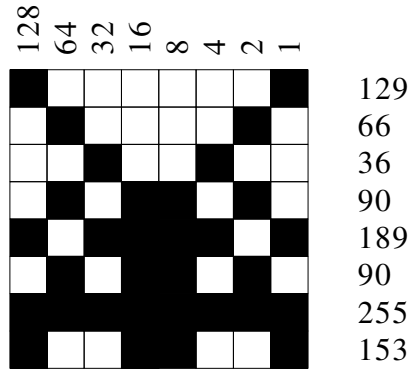
```

1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130
1315 REM * ALIEN BATTLESHIP
1320 REM *
1325 REM *****
1330 DEF PROCalien
1335 VDU 23,130, 0,0,195,153,90,189,24,
24
1340 ENDPROC

```



## ALIEN SPIDER

*Fig. 3.3k**Listing 3.3k*

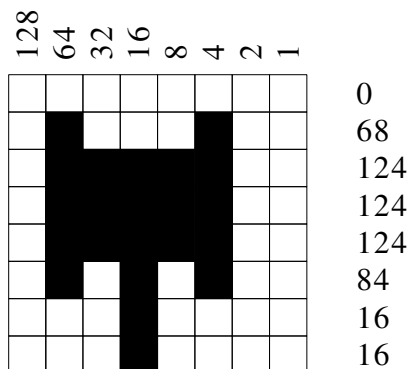
```

1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130 *
1315 REM * ALIEN SPIDER *
1320 REM *
1325 REM *****
1330 DEF PROCalien
1335 VDU 23,130, 129,66,36,90,189,90,25
5,153
1340 ENDPROC

```

## ALIEN TANK

Fig. 3.3l



Listing 3.3l

```

1300 REM *****
1305 REM *
1310 REM * DEFINE CHARACTER 130
1315 REM * ALIEN TANK
1320 REM *
1325 REM *****
1330 DEF PROCalien
1335 VDU 23,130, 0,68,124,124,124,84,16
,16
1340 ENDPROC

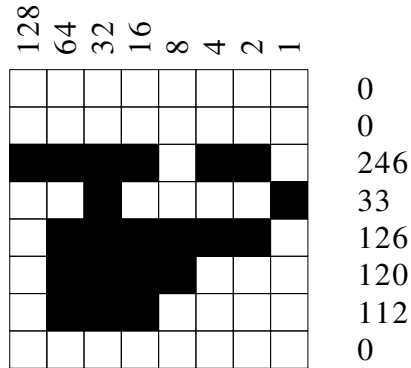
```



#### Section 4: PLAYER GRAPHICS.

This is exactly the same as choosing a shape for the aliens. Any of the following routines will define a character for use as the player. Each routine is, once again, accompanied by a display to show the character's shape enlarged. Remember that you want the player to look different from the aliens. Some shapes might be more suitable than others, depending on whether you are dodging or firing. You also need to take the background into account. Of course it is possible to have a surfer blasting away with fireballs over a city skyline but it will look a bit strange. You need to choose just one listing from this section and it can be any of the fifteen routines presented.

## PLAYER HELICOPTER

*Fig. 3.4a**Listing 3.4a*

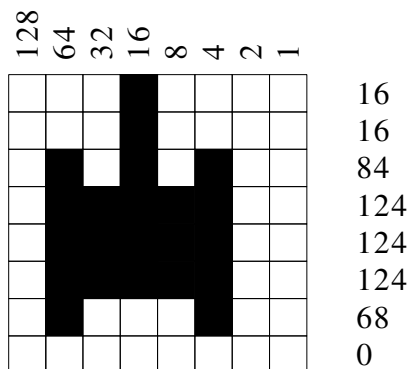
```

1400 REM *****
1405 REM *
1410 REM * DEFINE CHARACTER 131
1415 REM * PLAYER HELICOPTER
1420 REM *
1425 REM *****
1430 DEF PROCplayer
1435 VDU 23,131, 0,0,246,33,126,120,112,0
1440 ENDPROC

```

## PLAYER TANK

Fig. 3.4b



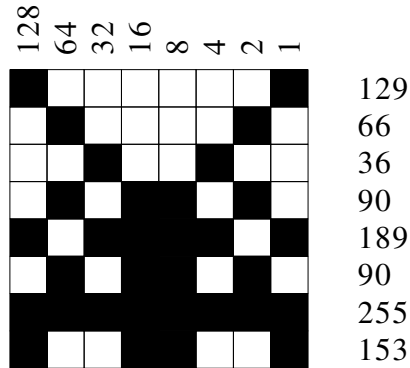
Listing 3.4b

```

1400 REM *****
1405 REM *
1410 REM * DEFINE CHARACTER 131
1415 REM * PLAYER TANK
1420 REM *
1425 REM *****
1430 DEF PROCplayer
1435 VDU 23,131,
16,16,84,124,124,124,68,0
1440 ENDPROC

```

## PLAYER SPIDER

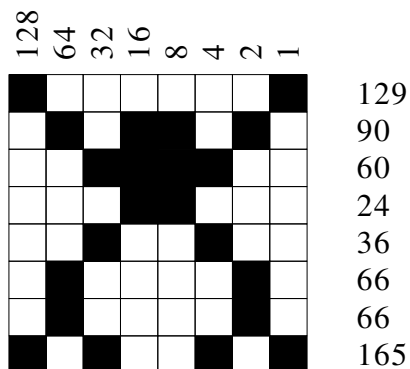
*Fig. 3.4c**Listing 3.4c*

```

1400 REM *****
1405 REM *
1410 REM * DEFINE CHARACTER 131
1415 REM * PLAYER SPIDER
1420 REM *
1425 REM *****
1430 DEF PROCplayer
1435 VDU 23,131,
129,66,36,90,189,90,255,153
1440 ENDPROC

```

## PLAYER FROG

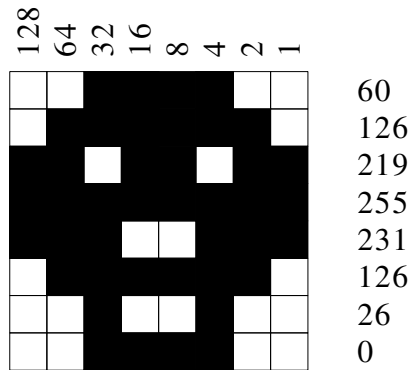
*Fig. 3.4d**Listing 3.4d*

```

1400 REM *****
1405 REM *
1410 REM * DEFINE CHARACTER 131
1415 REM * PLAYER FROG
1420 REM *
1425 REM *****
1430 DEF PROCplayer
1435 VDU 23,131,
129,90,60,24,36,66,66,165
1440 ENDPROC

```

## PLAYER SKULL

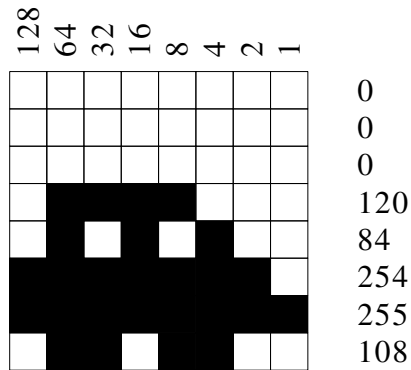
*Fig 3.4e**Listing 3.4e*

```

1400 REM *****
1405 REM *
1410 REM * DEFINE CHARACTER 131
1415 REM * PLAYER SKULL
1420 REM *
1425 REM *****
1430 DEF PROCplayer
1435 VDU 23,131, 60,126,219,255,231,126
,36,60
1440 ENDPROC

```

## PLAYER AUTOMOBILE

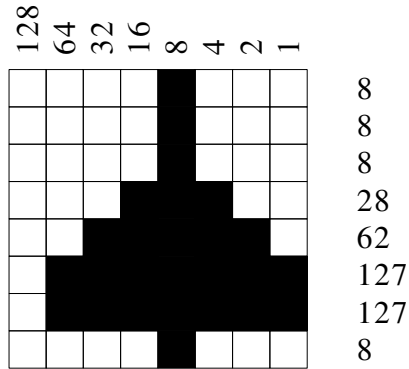
*Fig. 3.4f**Listing 3.4f*

```

1400 REM *****
1405 REM *
1410 REM * DEFINE CHARACTER 131
1415 REM * PLAYER AUTOMOBILE
1420 REM *
1425 REM *****
1430 DEF PROCplayer
1435 VDU 23,131, 0,0,0,120,84,254,255,108
1440 ENDPROC

```

## PLAYER AIRPLANE

*Fig. 3.4g**Listing 3.4g*

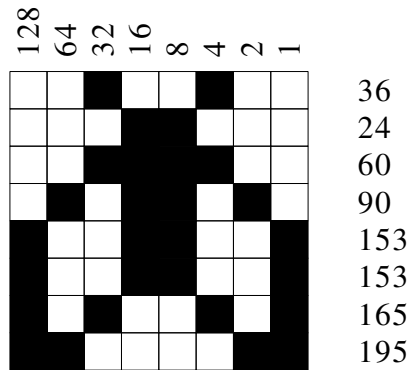
```

1400 REM *****
1405 REM *
1410 REM * DEFINE CHARACTER 131
1415 REM * PLAYER AIRPLANE
1420 REM *
1425 REM *****
1430 DEF PROCplayer
1435 VDU 23,131, 8,8,8,28,62,127,127,8
1440 ENDPROC

```

## PLAYER FLY

Fig. 3.4h



Listing 3.4h

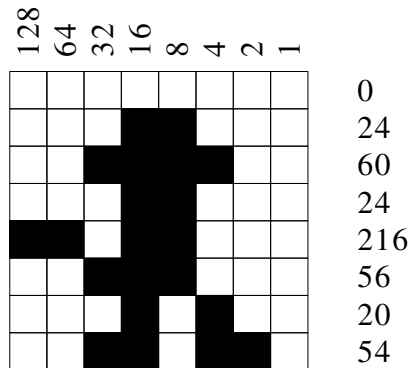
```

1400 REM *****
1405 REM *
1410 REM * DEFINE CHARACTER 131
1415 REM * PLAYER FLY
1420 REM *
1425 REM *****
1430 DEF PROCplayer
1435 VDU 23,131,
36,24,60,90,153,153,165,195
1440 ENDPROC

```



## PLAYER COWBOY LEFT

*Fig. 3.4j**Listing 3.4j*

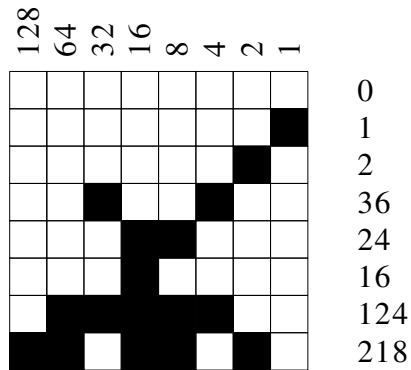
```

1400 REM *****
1405 REM *
1410 REM * DEFINE CHARACTER 131
1415 REM * PLAYER COWBOY LEFT
1420 REM *
1425 REM *****
1430 DEF PROCplayer
1435 VDU 23,131, 0,24,60,24,216,56,20,54
1440 ENDPROC

```



## PLAYER FIELD GUN

*Fig. 3.4l**Listing 3.4l*

```

1400 REM *****
1405 REM *
1410 REM * DEFINE CHARACTER 131
1415 REM * PLAYER FIELD GUN
1420 REM *
1425 REM *****

1430 DEF PROCplayer
1435 VDU 23,131, 0,1,2,36,24,16,124,218
1440 ENDPROC

```







## Section 5: START/RESTART.

This routine is *NOT* optional. We *ALWAYS* need to have this routine exactly as listed. This ensures that all the variables which might be used later on are initialised whether or not our game takes any account of them. It might be a good idea to SPOOL this routine separately onto tape so that you can use \*EXEC to merge it with your programs when required. The BBC User Guide tells you how to do this.

*Listing 3.5*

```

1500 REM *****
1510 REM *
1520 REM * START/RESTART
1530 REM *
1540 REM *****
1550 DEF PROCstart
1560 ALIENS=10
1570 SCORE=0
1580 AMMO=10
1590 FUEL=10
1600 REM *****
1610 REM * START POSITION FOR PLAYER *
1620 REM *****
1630 YP=31 : XP=640
1640 REM *****
1650 REM * CHANGE Y COORDINATE IF *
1660 REM * IT'S A DODGING GAME *
1670 REM *****
1680 IF NOT FIRE THEN YP=960
1690 MOVE XP,YP : GCOL 0,1 : VDU 131
1700 OXP=XP
1710 OYP=YP
1720 DEAD=TRUE
1730 PAST=0
1740 FINISH=FALSE
1750 ENDPROC

```

This routine essentially just initialises all the variables used in the game. For greater detail of what the variables are actually used for see Appendix One. The number of ALIENS is set to 10, your SCORE is set to zero and you are given 10 units of AMMO and FUEL. These variables must always be set to some starting value even if they' re not used by the game. This is because other instructions might refer to them and if they haven' t been assigned a value the computer will stop and tell you so. Not very helpful when you' re trying to play a game!

XP and YP are the co-ordinates for the player' s position and these must be set to some initial values. For FIREing games you start near the bottom of the screen (with YP=31) and in the middle (XP=640). If the game is not a FIREing game you start at the top of the screen (YP=990). The character for the player is now PRINTed on the screen at its initial position. We need to remember where the PLAYER was so that we can erase this character after we place the player at a new position. So, we have to copy the co-ordinates of the player' s position into OXP and OYP.

Finally we set the flags that indicate whether a new alien is required and whether the game has FINISHED. To do this we set DEAD equal to TRUE and FINISH to FALSE. We also need to reset the computer which tells us how many aliens have got PAST, so we set PAST equal to 0.

## Section 6: MOVE/FIRE.

We must have the first, main, routine from this section in our program. If you allowed LEFT & RIGHT movement then you will need to add listing 3.6b. Similarly listing 6.3c must be added for UP & DOWN movement. Don' t worry if you have only allowed, say, LEFT movement in the instructions. Just add the section for both LEFT & RIGHT and you will find that only LEFT movement is actually enabled. If you have chosen HYPER-drive as a mode of movement then you will need to add listing 3.6d. As listing 3.6a is such a large routine it might be a good idea to SPOOL this routine separately on tape so that you can MERGE it with your programs when required (details are given in the BBC User Guide).

*Listing 3.6a*

```

1800 REM *****
1810 REM *
1820 REM * MOVE/FIRE
1830 REM *
1840 REM *****
1850 DEF PROCmovefire
1860 IF NOT DEAD THEN 1930
1870 YA=990 : XA=RND(1248) : OXA=XA
1880 DEAD=FALSE
1890 GCOL 0,1 : MOVE XA,YA+10 : VDU 130
1900 REM *****
1910 REM * MOVE ALIEN CHARACTER 130 *
1920 REM *****
1930 GCOL 3,1
1940 MOVE OXA,YA+10 : VDU 130 : MOVE XA
,YA : VDU 130 : OXA=XA
1950 YA=YA-10 : IF YA=0 THEN DEAD=TRUE
: PAST=PAST+1 : ALIENS=ALIENS-1
1960 IF DEAD THEN MOVE OXA,YA+10 : VDU
130
1970 XA=XA-20+RND(3)*10
1980 IF NOT FIRE AND XA>XP THEN XA=XA-1
0
1990 IF NOT FIRE AND XA<XP THEN XA=XA+1
0
2000 IF XA>1248 THEN XA=1248
2010 IF XA<0 THEN XA=0
2020 REM *****
2030 REM * MOVE PLAYER CHARACTER 131 *
2060 REM *****
2120 IF XP<0 THEN XP=0
2130 IF XP>1248 THEN XP=1248
2140 IF YP<31 THEN YP=31
2150 IF YP>991 THEN YP=991
2160 IF OXP=XP AND OYP=YP THEN 2200

```

```

2170 GCOL 3,1
2180 MOVE OXP,OYP : VDU 131
2190 MOVE XP,YP : VDU 131
2200 OXP=XP : OYP=YP
2210 IF INKEY(-99) THEN KEY=TRUE ELSE K
EY=FALSE
2220 ENDPROC

```

This routine allows LEFT or RIGHT movement provided that the INSTRUCTIONS enable such movement.

*Listing 3.6b*

```

2040 REM * LEFT & RIGHT *
2070 IF LEFT AND INKEY(-98) THEN XP=XP-
10
2080 IF RIGHT AND INKEY(-67) THEN XP=XP
+10

```

This routine allows UP or DOWN movement provided the INSTRUCTIONS enable such movement.

*Listing 3.6c*

```

2050 REM * UP & DOWN *
2090 IF UP AND INKEY(-73) THEN YP=YP+10
2100 IF DOWN AND INKEY(-105) THEN YP=YP
-10

```

This routine allows HYPER-space movement provided the INSTRUCTIONS enable such movement.

*Listing 3.6d*

```

2110 IF HYPER AND INKEY(-85) THEN XP=RN
D(1248) : YP=RND(960)+32

```

The main routine is quite long and performs several different tasks, so let' s look at then in order. Firstly it checks whether a new alien is required (if the previous one is DEAD then we do need one). If we do start a new alien then we must initialise the co-ordinates for the alien. We set YA=990 and XA equal to a RaNDom number between 1 and 1248 (this is the left hand edge of the alien so we don' t want it to be right oevr the right edge of the screen). As with the player we will need to remember the position of the alien so we copy XA into OXA. We don' t need to remember the YZ co-ordinate because it' s always 10 more than the current row. Now we set the flag to show that the alien is not DEAD.

By the time we get here we have a live alien on our hands (well on the screen anyway) so we want to move it. We use GCOL 3,1 to set the overprinting mode and rePRINT the alien at its old position. This will remove it from the screen, and we then PRINT it at is new position. We must now calculate the next position for our alien. We take 10 off YA to move him down the screen. At this point we test whether he has reached the bottom of the screen (YA=0). If he has then he got PAST so we need to add 1 to the PAST counter and declare that alien DEAD. We also knock one off the number of ALIENS left. If he is dead then we PRINT him again to remove him from the screen.

Assuming he' s noDEAD we would like our alien to move from side to side as he comes down. If it' s a dodging gameNOT a FIREing game) we want the alien to move towards you so we test which side of your XP co-ordinate he is and alter his XA co-ordinate accordingly.

We also alter his XA co-ordinate by ten either side at RaNDom. Having changed the alien' s position we check that he hasn' t gone off the side of the screen and reseXA if he did.

Now we come to moving the player. It is at this stage that we will encounter some of the lines from the other listings when they have been added. If a particular sort of movement is allowed then we check to see if the appropriate key is being pressed and then alter YP or XP accordingly. If HYPER-space is being used and the H key is being held down then we choose a new position for XP and YP at RaNDom. Again we must check that the new co-ordinates for the player are not off the screen and alter them if necessary. Finally we rePRINT the player at his old position to

erase him and PRINT the player at the new position. Note that we do not do this if the player has not moved, i.e. if his new position is the same as his old position. This is to stop the player graphic flickering when it is stationary.

The final action in this section is to check the FIRE button. If this key is being held down then we set a flag, KEY, to TRUE to indicate that a shot is to be fired.

## Section 7: DETECT A HIT

This routine can have two completely different forms. This depends on whether we are playing a FIREing game or a dodging game. In the latter case we need to detect collisions between the player and other objects. These objects may be background objects or aliens. The first routine (listing 3.7a) in this section deals with the detection of this sort of collision and should be used if yours is a dodging game.

If we are playing a FIREing game then we might need to move a missile up the screen and see if this hits any ALIENS. For this type of game we can pick up any of the final four routines in this section (listings 3.7b-e). These give us various different types of projectile to choose from.

### *Listing 3.7a*

```

2400 REM *****
2410 REM *
2420 REM * CHECK FOR HIT
2430 REM * DODGING GAME
2440 REM *
2450 REM *****
2460 DEF PROCcheck
2470 HIT=FALSE
2480 GCOL 3,1 : MOVE OXP,OYP : VDU 131
2490 FOR I=OXP TO OXP+64 STEP 8
2500     FOR J=OYP TO OYP-32 STEP -4
2510         IF POINT(I,J)<> BACK THEN HIT=
TRUE
2520     NEXT J

```

```

2530     NEXT I
2540  MOVE OXP,OYP : VDU 131
2550  ENDPROC

```

*Listing 3.7b*

```

2400  REM *****
2410  REM *
2420  REM * CHECK FOR HIT
2430  REM * BULLET
2440  REM *
2450  REM *****
2460  DEF PROCcheck
2470  HIT=FALSE : IF NOT KEY THEN 2560
2480  SOUND 1,-15,0,1
2490  BX=XP+32:BY=YP
2500  GCOL 3,11:PLOT 69,BX,BY
2510  REPEAT
2520     BY=BY+10:PLOT 69,BX,BY:PLOT 69,B
X,BY-10
2530     IF ABS(XA-BX+32)<32 AND ABS(YA-B
Y)<10 THEN HIT=TRUE
2540     UNTIL HIT OR BY>990
2550  PLOT 69,BX,BY
2560  ENDPROC

```

*Listing 3.7c*

```

2400  REM *****
2410  REM *
2420  REM * CHECK FOR HIT
2430  REM * LASER
2440  REM *
2450  REM *****
2460  DEF PROCcheck
2470  HIT=FALSE : IF NOT KEY THEN 2540
2480  GCOL 3,11 : BX=XP+32 : BY=YP
2490  MOVE BX,BY : DRAW BX,990

```

```

2500 SOUND 0,-15,6,2
2510 T=TIME : REPEAT UNTIL TIME>T+3
2520 MOVE BX,BY : DRAW BX,990
2530 IF ABS(XA-BX+32)<32 THEN HIT=TRUE
2540 ENDPROC

```

*Listing 3.7d*

```

2400 REM *****
2410 REM *
2420 REM * CHECK FOR HIT
2430 REM * BOMB
2440 REM *
2450 REM *****
2460 DEF PROCcheck
2470 HIT=FALSE : IF NOT KEY THEN 2560
2480 SOUND 1,-15,0,1
2490 BX=XP:BY=YP+32
2500 GCOL 3,11:MOVE BX,BY :PRINT"."
2510 REPEAT
2520 BY=BY+10:MOVE BX,BY:PRINT"." :MOV
E BX,BY-10:PRINT"."
2530 IF ABS(XA-BX)<32 AND ABS(YA-BY)<
10 THEN HIT=TRUE
2540 UNTIL HIT OR BY>990
2550 MOVE BX,BY:PRINT"."
2560 ENDPROC

```

*Listing 3.7e*

```

2400 REM *****
2410 REM *
2420 REM * CHECK FOR HIT
2430 REM * FIREBALL
2440 REM *
2450 REM *****
2460 DEF PROCcheck
2470 HIT=FALSE : IF NOT KEY THEN 2560

```

```

2480 SOUND 1,-15,0,1
2490 BX=XP:BY=YP+32
2500 GCOL 3,11:MOVE BX,BY :PRINT "*"
2510 REPEAT
2520     BY=BY+10:MOVE BX,BY:PRINT "*":MOV
E BX,BY-10:PRINT "*"
2530     IF ABS(XA-BX)<32 AND ABS(YA-BY)<
10 THEN HIT=TRUE
2540     UNTIL HIT OR BY>990
2550 MOVE BX,BY:PRINT "*"
2560 ENDPROC

```

The first routine in this section works by temporarily erasing the player and looking at what' s left on the screen underneath him. If it' s not a space then obviously the player has hit something.

The second routine in this section first checks to see if the fire KEY was pressed. If so then a bullet is fired up the screen, starting with a bang, by using PLOT to display a point on the screen. The position of the dot is tested against the alien' s position to see if it has hit him. Otherwise it keeps going until it gets to the top of the screen. If it did hit the alien the flag HIT will be set to TRUE.

The third routine (listing 3.7c) checks whether the fire KEY was pressed and if necessary fires a laser beam up the screen by using DRAW. This is accompanied by a zap noise. The horizontal position of the laser is checked against the XA co-ordinate of the alien and if the two coincide then HIT is declared TRUE.

The final two routines both work in the same way. First a bomb or fireball is placed on the screen using PRINT. This is moved up the screen, being rePRINTed at its old position and PRINTed at its new position, until it either reaches the top of or coincides with the current co-ordinates of the alien. Again the flag HIT is set to TRUE if a HIT has been scored.

## Section 8: EXPLOSIONS

This section will be called from the control program only if a HIT has occurred. In this section we wish to make some sort of noise, and possibly give a visible signal, to indicate that this has happened. You may choose any of the first ten routines, listings 3.8a-j, as your main routine for this section. It is very important to remember, whichever one of these you choose, that listing 3.8k must also be added if you have a FIREing game. This routine will wipe out the dead alien.

### *Listing 3.8a*

```

2600 REM *****
2610 REM *
2620 REM * EXPLOSION
2630 REM * BURBLE
2640 REM *
2650 REM *****
2660 DEF PROCexplode
2670 SOUND 1,-15,30,1:SOUND 1,-15,20,1:
SOUND 1,-15,10,1
2680 SOUND 1,-15,50,1:SOUND 1,-15,100,1
: SOUND 1,-15,150,1
2750 ENDPROC

```

### *Listing 3.8b*

```

2600 REM *****
2610 REM *
2620 REM * EXPLOSION
2630 REM * TRILL
2640 REM *
2650 REM *****
2660 DEF PROCexplode
2670 SOUND 1,-15,0,1:SOUND 1,-15,50,2
2680 SOUND 1,-15,5,1:SOUND 1,-15,100,1
2750 ENDPROC

```

*Listing 3.8c*

```

2600 REM *****
2610 REM *
2620 REM * EXPLOSION
2630 REM * HIT
2640 REM *
2650 REM *****
2660 DEF PROCexplode
2670 SOUND 0,-15,6,1:SOUND 0,-13,5,1:SO
UND 0,-11,4,1
2750 ENDPROC

```

*Listing 3.8d*

```

2600 REM *****
2610 REM *
2620 REM * EXPLOSION
2630 REM * START AGAIN
2640 REM *
2650 REM *****
2660 DEF PROCexplode
2670 SOUND 1,-15,80,5:SOUND 1,-15,0,0:S
OUND 1,-15,80,5:SOUND 1,-15,68,5
2680 SOUND 1,-15,88,5:SOUND 1,-15,80,10
:SOUND 1,-15,68,15
2750 ENDPROC

```

*Listing 3.8e*

```

2600 REM *****
2610 REM *
2620 REM * EXPLOSION
2630 REM * BURBLE
2640 REM *
2650 REM *****
2660 DEF PROCexplode
2670 SOUND 1,-15,80,1:SOUND 2,-15,110,2

```

```
:SOUND 1,-15,150,2:SOUND 2,-15,200,2
2750 ENDPROC
```

### *Listing 3.8f*

```
2600 REM *****
2610 REM *
2620 REM * EXPLOSION
2630 REM * RASPBERRY
2640 REM *
2650 REM *****
2660 DEF PROCexplode
2670 SOUND 1,0,90,6:SOUND 1,0,100,6
2680 SOUND 0,-15,3,10
2750 ENDPROC
```

### *Listing 3.8g*

```
2600 REM *****
2610 REM *
2620 REM * EXPLOSION
2630 REM * FLASH
2640 REM *
2650 REM *****
2660 DEF PROCexplode
2670 FOR I=1 TO 16
2680 VDU 19,0,I,0,0,0: SOUND 0,-15,I,
1
2690 NEXT I
2750 ENDPROC
```

### *Listing 3.8h*

```
2600 REM *****
2610 REM *
2620 REM * EXPLOSION
2630 REM * BLIPS
2640 REM *
```

```

2650 REM *****
2660 DEF PROCexplode
2670 FOR I=1 TO 16
2680     SOUND RND(3),-15,RND(200),RND(3)
2690     NEXT I
2750 ENDPROC

```

*Listing 3.8i*

```

2600 REM *****
2610 REM *
2620 REM * EXPLOSION
2630 REM * BLIP
2640 REM *
2650 REM *****
2660 DEF PROCexplode
2670 FOR I=1 TO 200 STEP 40
2680     SOUND 1,-15,I,2
2690     NEXT I
2750 ENDPROC

```

*Listing 3.8j*

```

2600 REM *****
2610 REM *
2620 REM * EXPLOSION
2630 REM * SIREN
2640 REM *
2650 REM *****
2660 DEF PROCexplode
2670 FOR I=1 TO 5
2675     FOR J=150 TO 130 STEP -2
2680         SOUND 1,-15,J,1:SOUND 2,-10,J+
20,1: NEXT J
2685     FOR J=130 TO 150 STEP 2
2690         SOUND 1,-15,J,1:SOUND 2,-10,J+
20,1: NEXT J
2695     NEXT I

```

```
2750 ENDPROC
```

This routine *MUST* be added if there is FIREing in your game.

```
2700 REM *****
2710 REM * ERASE ALIEN IF YOU SHOT *
2720 REM *****
2730 GCOL 3,1 : MOVE OXA,YA+10 : VDU 13
0
2740 DEAD=TRUE : ALIENS=ALIENS-1
```

All the routines in this section except 3.8k use the SOUND command of the BBC to produce noises. The BBC does also have an ENVELOPE command for controlling sound, but this is very complicated and we won't use it in this book.

Routine 3.8k simply rePRINTS the alien to erase him when he has been blown up. It then sets the flag DEAD, to tell the move routine to start a new alien, and, finally, takes one off the number of ALIENS left.

## Section 9: SCORE ROUTINES

If you have HIT an alien then there is obviously the question of how much was SCOREd. Well, these routines are fairly simple and give you the choice of SCOREing one, five or ten points for each alien you HIT. If yours is a dodging game then you will not want to SCORE at all so use the first routine (listing 9.3a). Otherwise you can pick any of the other routines from this section.

### *Listing 3.9a*

```
3000 REM *****
3010 REM *
3020 REM * SCORE ROUTINE
3030 REM * NOT USED
3050 REM *****
3060 DEF PROCscore
```

```
3070 ENDPROC
```

### *Listing 3.9b*

```
3000 REM *****
3010 REM *
3020 REM * SCORE ROUTINE
3030 REM * ONE POINT
3050 REM *****
3060 DEF PROCscore
3070 IF NOT FIRE OR NOT HIT THEN 3090
3080 SCORE=SCORE+1
3090 ENDPROC
```

### *Listing 3.9c*

```
3000 REM *****
3010 REM *
3020 REM * SCORE ROUTINE
3030 REM * FIVE POINTS
3050 REM *****
3060 DEF PROCscore
3070 IF NOT FIRE OR NOT HIT THEN 3090
3080 SCORE=SCORE+5
3090 ENDPROC
```

### *Listing 3.9d*

```
3000 REM *****
3010 REM *
3020 REM * SCORE ROUTINE
3030 REM * TEN POINTS
3050 REM *****
3060 DEF PROCscore
3070 IF NOT FIRE OR NOT HIT THEN 3090
3080 SCORE=SCORE+10
3090 ENDPROC
```

All the routines in this section (except 3.9a) check to see if you are playing a FIREing game and whether you just scored a HIT on an alien. If you did them the required number of points are simply added to your SCORE.

## Section 10: FUEL & AMMUNITION

In some games you can fail your mission because of lack of FUEL or AMMO. If you do not wish to alter the quantities of either of these then you can select just the first routine from this section. If, however, you wish to alter either quantity then use listing 3.10b and add in the appropriate listings from the rest of this section. Note that you can only choose one out of each of the pairs of routines (listing 3.10d and 3.10e for AMMO and listings 3.10g and 3.10h for FUEL) for increasing quantities, since these occupy the same line numbers.

### *Listing 3.10a*

```

3200 REM *****
3210 REM *
3220 REM * FUEL & AMMUNITION
3230 REM * NOT USED
3240 REM *
3250 REM *****
3260 DEF PROCfuel
3270 ENDPROC

```

### *Listing 3.10b*

```

3200 REM *****
3210 REM *
3220 REM * FUEL & AMMUNITION
3230 REM *
3240 REM *****
3250 DEF PROCfuel
3260 ENDPROC

```

*Listing 3.10c*

```

3260 REM *****
3270 REM *   DECREASE AMMO                      *
3280 REM *****
3290 IF KEY THEN AMMO=AMMO-1

```

*Listing 3.10d*

```

3300 REM *****
3310 REM *   INCREASE AMMO                      *
3320 REM *****
3330 IF HIT THEN AMMO=AMMO+1

```

*Listing 3.10e*

```

3300 REM *****
3310 REM *   DOUBLE AMMO                      *
3320 REM *****
3330 IF HIT THEN AMMO=AMMO+1

```

*Listing 3.10f*

```

3340 REM *****
3350 REM *   DECREASE FUEL                      *
3360 REM *****
3370 IF KEY THEN FUEL=FUEL-1

```

*Listing 3.10g*

```

3380 REM *****
3390 REM *   INCREASE FUEL                      *
3400 REM *****
3410 IF HIT THEN FUEL=FUEL+1

```

*Listing 3.10h*

```

3380 REM *****
3390 REM *   RESET FUEL                      *

```

```

3400 REM *****
3410 IF HIT THEN FUEL=10

```

The routines for decreasing FUEL or AMMO check whether the fire KEY was pressed and take off one unit per shot. The routines for increasing FUEL or AMMO check whether you scored a HIT and if you did they either add one unit, double the number of units, or reset the value to full.

## Section 11: STATUS DISPLAY

After all the excitement so far with explosions, lasers blasting away and your SCORE rapidly rising we need to make sure the relevant information is displayed on the screen. Of course, if you have been peacefully dodging a plague of flies in the desert you won't need to display anything, but you still must put a routine into this slot. For a dodging game use listing 3.11a. For a FIREing game start with listing 3.11b, which will display the SCORE. If your game includes a varying supply of FUEL or AMMO then add the appropriate one (or both) of listings 3.11c and 3.11d.

### *Listing 3.11a*

```

3500 REM *****
3505 REM *
3510 REM * STATUS DISPLAY
3515 REM * NOT USED
3520 REM *
3525 REM *****
3530 DEF PROCstatus
3535 ENDPROC

```

### *Listing 3.11b*

```

3500 REM *****
3505 REM *
3510 REM * STATUS DISPLAY
3515 REM *

```

```

3520 REM *****
3525 DEF PROCstatus
3530 VDU 4 : COLOUR 128+BACK : COLOUR 7
-BACK
3535 REM *****
3540 REM * DISPLAY SCORE *
3545 REM *****
3550 PRINT TAB(5,0); "SCORE:"; SCORE
3595 VDU 5:ENDPROC

```

*Listing 3.11c*

```

3555 REM *****
3560 REM * DISPLAY AMMO *
3565 REM *****
3570 PRINT TAB(0,0); "A:"; AMMO; " "

```

*Listing 3.11d*

```

3575 REM *****
3580 REM * DISPLAY FUEL *
3585 REM *****
3590 PRINT TAB(15,0); "F:"; FUEL; " "

```

These routines are all very simple and merely PRINT the values in question out on the top line of the screen. The positions at which they are PRINTed are carefully arranged to ensure that they do not overlap even if all three are being used.

## Section 12: CHECK FOR END OF GAME

In this section we provide all the checks see if we have reached an end of game situation. If any of these checks are found to be true then we must set the flag FINISH to TRUE. This will signal to the control program that it is time to stop playing the game and move on to the end of game display routine. If FINISH is FALSE the control program will go back to the MOVE/FIRE subroutine. From this section we need to type in the main routine (listing 3.12a) and at least one of the remaining five routines. After all, we want to stop the game sometime!

The first of the checks (listing 3.12b) is on whether the number of ALIENS is down to 0: This could be used in a dodging game, for instance, where if you manage to dodge ten aliens then you win. For a dodging game you will also need to test for a collision with listing 3.12c.

You might decide that you were to lose if more than three aliens got PAST and over-ran your base. This could be tested for by adding listing 3.12d.

The final two listings deal with testing for running out of AMMO or FUEL, if you wish these to indicate the end of game.

### *Listing 3.12a*

```

3600 REM *****
3610 REM *
3620 REM * CHECKS FOR END *
3630 REM *
3640 REM *****
3650 DEF PROCfinish
3710 ENDPROC

```

### *Listing 3.12b*

```

3660 IF ALIENS=0 THEN FINISH=TRUE

```

### *Listing 3.12c*

```

3670 IF HIT THEN FINISH=TRUE

```

*Listing 3.12d*

```
3680 IF PAST=3 THEN FINISH=TRUE
```

*Listing 3.12e*

```
3690 IF FUEL=0 THEN FINISH=TRUE
```

*Listing 3.12f*

```
3700 IF AMMO=0 THEN FINISH=TRUE
```

These routines all test variables or flags set elsewhere in the program and, when one of the conditions is satisfied, they set the FINISH flag to TRUE.

## Section 13: END OF GAME DISPLAY

If the FINISH flag has been set for some reason then we want to stop the game. We could just stop the game. This is the simplest and most obvious method. You will then have to type RUN to play again. Listing 3.13a is just such a QUICK STOP routine. However it might be nice to end up with a polite question as to whether or not you wished to try again. Well, since it costs nothing to be polite, listing 3.13b is a POLITE STOP. If yours is a SCOREing game, then you might also like to add listing 3.13c to tell you what your score was.

*Listing 3.13a*

```

3800 REM *****
3810 REM *
3820 REM * GAME OVER
3830 REM * POLITE STOP
3840 REM *
3850 REM *****
3860 DEF PROCgameover
3870 GCOL 0,132 : GCOL 0,7 : CLG
3890 MOVE 300,500 : PRINT "ANOTHER GO ?
"

3900 IF INKEY(-69) THEN 3930
3910 IF NOT INKEY(-86) THEN 3900
3920 VDU 4,20,12 : END
3930 ENDPROC

```

*Listing 3.13c*

```

3880 MOVE 100,650 : PRINT "YOUR SCORE W
AS ";SCORE

```

The first routine in this section is fairly obvious. The second, more polite routine sets the background colour, clears the screen and then displays the question ' ANOTHER GO?' and waits for you to press either the ' Y' or ' N' key. If you press ' Y' then it RETURNS to the control program, which will start again. If you press ' N' then it resets the normal colours for the screen and ENDS. If you add in listing 3.13c then your SCORE will be PRINTed on the screen above the question.

# CHAPTER 4

## Starting to Write Your Own Games

Hopefully, you've had a lot of fun making your own games from the routines in Chapter 3. But there is no reason why you shouldn't make up your own routines to extend the selection available for a section. Perhaps you are wishing there was a different background available, or a different explosion. Well, in this chapter we will take a good look at how we could change or completely replace the routines given in Chapter 3. Let's go through the routines as listed and see how they really work.

### INSTRUCTIONS

These are fairly easy to alter. In BASIC a line can either be a command on its own or lots of commands separated by colons. These commands, plus whatever follows them up to the next colon (or the end of the line), are known as statements. There are a lot of PRINT statements in this section so let's take a look at some of them first. Basically, anything inside the quote marks gets PRINTed on the screen, so for the example game you could change line 1080 from:

```
1080 MOVE 0,800: PRINT "USE Z TO MOVE  
LEFT": LEFT=TRUE
```

to:

```
1080 MOVE 0,800: PRINT "PRESS Z TO ZOOM  
LEFT": LEFT=TRUE
```

As you can see the only differences are inside the quotes and this would simply PRINT the new message on the screen instead of the original message. Altering the position of the message on the screen is also fairly simple. The graphics screen on your BBC should be thought of as having a grid of 1280 dots across (X co-ordinate) by 1024 dots down (Y co-ordinate) on it, and if we are using the graphics cursor to position text we can specify any one

of these points to be the starting point for our PRINT statement. We simply issue a MOVE command and then PRINT the character(s). If we are using the text cursor to place text then the screen can be thought of as 32 (or, in some MODES, 25) rows of text, each 20, 40 or 80 columns wide, again depending on the MODE. We can pick any position from the number of available rows and columns at which to start our message. This is what the numbers inside the brackets after TAB mean in the PRINT command and this method of putting text on the screen is used in the scoring routines.

Let' s use the graphics cursor to position a ' GOOD LUCK!' message on the screen. We need to pick up a line number that' s not already being used. 1092 seems like a good choice. We want to PRINT the message lower than the last instruction so let' s use 300 for the Y co-ordinate to keep it well out of the way. We also need to start at an x co-ordinate, and if we choose X equal to 400, it will start the message just before the middle of the screen. Right, so our new line of BASIC is:

```
1092 MOVE 400,300:PRINT"GOOD LUCK!"
```

And it' s as simple as that.

The VDU command is exactly the same as a PRINT command but instead of putting letters inside quotes we give a list of their ASCII values (see Appendix Three). So in the above example we could have used VDU 72,79,79,68,32,76,75,67,75,33 instead of PRINT "GOOD LUCK!". This doesn' t seem particularly useful but if we are just PRINTing a single user defined character it is clearer than using PRINT. You may have noticed by now that the commands which define characters are just VDU commands that start off with a number less than 32. Characters with ASCII values less than 32 are CONTROL CHARACTERS and the rest of the numbers PRINTed after it are used as data, which is why nothing appears on the screen when these statements are used. This whole topic is dealt with in Chapter 6 so we will now move on.

If we wish to alter the colours of anything displayed on the screen we must first look at the User Guide. You will find a section which tells you what all the colours are and which numbers are used to represent them. For ease of reference a table of the colours and numbers for MODE 2 is given below.

- 0 BLACK
- 1 RED
- 2 GREEN
- 3 YELLOW
- 4 BLUE
- 5 MAGENTA
- 6 CYAN
- 7 WHITE
- 8 FLASHING BLACK & WHITE
- 9 FLASHING RED & CYAN
- 10 FLASHING GREEN & PURPLE
- 11 FLASHING YELLOW & BLUE
- 12 FLASHING BLUE & YELLOW
- 13 FLASHING PURPLE & GREEN
- 14 FLASHING CYAN & RED
- 15 FLASHING WHITE & BLACK

These numbers are understood by the BBC to represent the colours whose names are written by them. So when we say COLOUR 1 the BBC will know we mean that text COLOUR is RED and if we say GCOL 0,128+7 (see the User Guide for full explanation of why we have added 128 to the number) the BBC understands that we want the graphics background to be WHITE. To change the overall colour of the display we can then simply change the number, in accordance with the above table, which follows GCOL 0,128+ in line 1060. Don' t change the background colour so that it is the same colour as the text you are going to PRINT onto it, though, or you won' t be able to see anything!

Don' t change any of the statements which set variables to TRUE or FALSE in this section as these tell the computer important facts which it needs to know in order to play the game.

## BACKGROUNDS

Most of the backgrounds in this book involve the random distribution of shapes on a coloured background. We will see how the shapes are made up in Chapter 6 so let's not worry about that now. The background colours can be changed in exactly the same way as we did above, so there's nothing complicated here either. Line 1380 (in listings 3.2b-e) controls the number of objects, in this case 40, that we have on the screen. If you wanted to make the game a bit harder (for a dodging game) or just wanted more shapes on the screen (for a firing game) then you could change the 40 to a larger number. This would put extra waves, stars, or whatever other shape is used, on the screen.

Line 1390 puts the shapes on the screen. The first thing on the line is:

```
MOVE RND(1280),RND(1024)
```

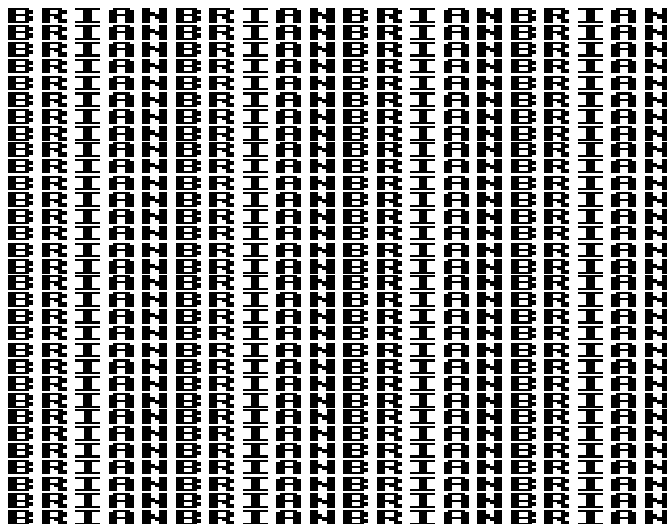
This is the line that defines the position for PRINTing. The RND(1280) chooses a number between 1 and 1280 that is used for the X co-ordinate. Similarly the RND(1024) chooses a number between 1 and 1024 that is used for the Y co-ordinate. This means that the object will be PRINTed starting from that specified graphics position.

In some cases there is a GCOL command in this line with GCOL 0,RND(15). If it is just a number after the GCOL0, then the appropriate colour from the above list is used. However with the RND(15) version, for each object that's PRINTed, a number between 1 and 15 is chosen for use as the foreground colour number.

The routines that draw skylines are slightly different from the others. In the first case you cannot have a different number of buildings across the bottom of the screen as there are only twenty columns. We also don't want the buildings printed aRaNDom positions. In listing 3.2g what we do is to use RND to determine the height of the buildings at random. So in line 1380 the X co-ordinate goes from 0 to 1280 in steps of 64 (characters are 64 points wide in MODE 2), while in line 1390 the Y position goes from 1 to RND(3), multiplied by the height of the character (32 points).



Notice that there is a gap at the bottom of the screen where an exact number of names could not be fitted in. This would not have happened if the name of the player was a length that divided exactly into the number of places on the screen, such as:



## PLAYER AND ALIEN GRAPHICS

These will be dealt with separately in Chapter 6.

## SET UP AND RE-START

Although this is a small routine, and there are no alternative routines, there are quite a few major changes you can make.

You could change the number of aliens you have to fight - either reducing the number to make the game easier to beat, or increasing the number to make the game harder. At present it is set to 10, but if you wanted to have 15 aliens then you could change the line which sets the number of aliens to:

ALIENS=15

You could also change your starting level of ammunition to make the game harder or easier in much the same way as you can alter the number of aliens. This is done in the following lines.

Supposing you wished to make the game a lot easier by having 100 bullets or bombs at the start then you could change the line to:

AMMO=100

The amount of fuel can also be changed. Increasing the number will make the game easier (your fuel will last longer) and decreasing the number will make the game harder (you will run out of fuel sooner). Rememebr that you might also want to change the routines in the section where fuel and ammunition are altered.

## MOVE AND FIRE

There is very little in this section that can be changed unless you know exactly what you are doing. However it is worth looking at how we move the characters around. The characters are all PRINTed on the background using the GCOL3 option, so that they are PRINTed on top of what is already there. In this way, they don' t wipe out any stars, waves, etc., and when they are removed the background will not have changed. This method is also used in the hit detection routines when moving bullets or bombs about. The section that looks to see if keys are pressed uses INKEY with a negative parameter to check whether any key is being held down. In this way the computer can tell whether, say, ' Z' is being held down even if you are pressing other keys, like the space bar, at the same time. This is very useful in games and a table of all the negative values used to test for specific keys can be found in the INKEY section of the User Guide.

## DETECT AND HIT

These routines use simple animation techniques to move a

missile from the player up the screen. When you are more experienced at writing programs you could rewrite these to move the missile in any of four directions from the player, for example, instead of just restricting firing to up the screen. These are similar to previous routines, and are also fairly simple, so we won't bother to look at these routines in detail.

## EXPLOSIONS

You can really have a lot of fun in this section, creating different noises to go with your game. Once you have grasped the methods of producing music you could even add a signature tune to your game! The SOUND command on the BBC has quite a range of length and pitch, going from 0.05 to over 12 seconds and from a low, almost inaudible, note through to a very high pitched whistle. The volume can also be altered from almost silent to positively annoying. The SOUND command has four parameters following it. The first of these is the channel to be used for the sound, and using this we can play notes on all three normal channels (1, 2 or 3) at the same time and make chords! But hang on, does that imply there's another channel which isn't normal? Well, yes, channel 0 is the 'white noise' channel. This means that any notes played through this channel produce hissing, scratchy noises, and this has been used in several places in our listings to make bangs and zaps.

Having decided which channel to use we can now choose a volume in the range 0 to -15 (the volume of noise increases as the value gets more negative, positive values here are for use only with the ENVELOPE command). Next we choose the note we are going to play and this is specified by a number between 0 and 255. Finally we can choose the length of the note, between very short (0) and very long (254). This gives the length of the note in 1/20ths of a second. If we use length 255 then the note will keep on playing till we press ESCAPE.

The pitch numbers go up in steps of a quarter of a semi-tone (of which there are twelve to an octave). This means that the note played with pitch 49 will be one octave above that with pitch 1. In this way it's quite easy to write routines that play musical tunes and there is one routine, listing 3.8d, that uses SOUND in this

way. For the most part, though, we will be wanting to make explosion type noises and we can do this best by using fast repetitions of very short notes.

## SCORING

There are one or two alterations that can be made to these routines, mainly concerned with changing the way scoring is carried out. The amount your score is incremented can be changed quite easily, and this has already been done in the routines listed. If you wanted to increase your score by 20 points for every alien shot down then you could simply change the line which adds to your score to make it:

$$\text{SCORE}=\text{SCORE}+20$$

A more complex routine would give a higher score the earlier you managed to hit the alien, so you would need a routine that converts a high Y co-ordinate into a high score number. As there are 990 points before the alien gets to you, if you take the number of points up the screen of the alien' s position, divided by 30, this will give you a score between 0 and 33 and you will score higher for hitting the alien earlier. The variable that stores the alien' s co-ordinate is YA so that the new statement would look like this:

$$\text{SCORE}=\text{SCORE}+(\text{YA DIV } 30)$$

This can be changed to other ratios, but we' ll leave that for you to play with.

## FUEL AND AMMUNITION

The routines in this section can be altered to increase or decrease the amounts by which the fuel and bullets are used up. The value for ammunition is stored in the variable AMMO and the amount of fuel is stored in the variable FUEL. Another variable used in these routines is HIT, which as we have seen already is used to

signify whether or not you managed to hit an alien (it is FALSE if you didn't, and TRUE if you did). This variable is used in dodging games to indicate that you've crashed.

## REDISPLAY THE SCREEN

There are not many things you can alter in this section but you could make a couple of changes if you wanted to make the game faster and a little harder. If you leave out the lines that PRINT your fuel and ammunition then you won't know what quantities you have left. That will make the game much harder to play, because you won't know if you're just about to run out of fuel or bullets. Because the BBC computer now has two less jobs to do, the program runs somewhat faster. It's like someone taking away some of your typing - you work much faster.

A more complicated change would be to put in a high score variable, so that you could see whether or not you were improving. This is more complex because you also have to put extra lines in other routines before you can PRINT it on the screen.

First you must decide what to call the variable in which you want to keep the high score value. If you look at Appendix One at the back of this book, you will see a list of the variables and their uses. Choose one that is not in the list. It is best to choose something meaningful, such as HISCORE. The next step is to initialise the variable. This means you have to set it to a definite value, otherwise the computer won't know where to start counting from. Some computers assume the variables start from zero unless otherwise specified, but the BBC has no such facility and so we must tell it to start at an initial number. To do this, we have to go back to the beginning of the program and the obvious place is in the INSTRUCTIONS routine. You can either set the high score to zero or to a high value to give yourself something to aim at. Use a statement like:

```
HISCORE=0
```

Don't forget to keep track of which extra line numbers you use or you could find yourself overwriting something you wanted to

keep. It' s best to look at the listing on the computer before adding any new lines.

In the END OF GAME routine we have to write a line which compares the high score with the score you have just achieved. If the latter is the greater it must alter the high score to be equal to your score. This means we will have to use an IF statement and we can write the statement out in English as:

IF (MY SCORE) IS GREATER THAN (HIGH SCORE)  
THEN MAKE (HIGH SCORE) EQUAL (MY SCORE).

This is very easy to convert to the BASIC language and is equivalent to:

IF SCORE>HIScore THEN LET HIScore=SCORE

Which goes to show how much like a shortened form of English the BASIC language is!

Having done this we must return to altering the routine which updates the screen display. We need an extra line in here to PRINT the high score on the screen and we could put one in that looked like this:

PRINT TAB(3,1);"HIGH SCORE: ";HIScore

You can of course put the high score anywhere you like but try to ensure that it doesn' t interfere with the rest of the game. Look again at the section on changing the INSTRUCTIONS if you' re not sure how to go about it.

## CHECK FOR END/END OF GAME

These routines would be quite awkward to change but perhaps when you have some more experience you could devise a system which tells you why the game has ended. You would need to examine the different variables to see why the game ended, or you could set one variable according to the reason why the game:

- 1 - The aliens overran your base

- 2 - You shot down all the aliens
- 3 - You ran out of fuel
- 4 - You ran out of bullets

The variable that flags the end of the game is FINISH so you could change the value that is assigned to it in each line of the CHECK routine from TRUE to a numebr (on the BBC, FALSE is equivalent to 0 so anything else is assumed to be TRUE but remember that unless TRUE is actually -1 then NOT the number won' t give you FALSE). In the END OF GAME routine you could PRINT a different message, dependent on the value of FINISH, that would tell the player why he had won or lost the game. These messages could be slotted in between lines already used and would look like this:

```
IF FINISH=1 THEN MOVE 0,400 : PRINT  
"YOU WERE OVERRUN BY ALIENS"  
IF FINISH=2 THEN MOVE 0,400 : PRINT  
"YOU SHOT DOWN ALL THE ALIENS"  
IF FINISH=3 THEN MOVE 0,400 : PRINT  
"YOU RAN OUT OF FUEL"  
IF FINISH=4 THEN MOVE 0,400 : PRINT  
"YOU RAN OUT OF AMMO"
```

Of course you could still add these lines even if you were using the impolite ending.

That concludes this chapter on altering the routines in the arcade games. Remember to take everything one step at a time, testing as you go, so that you don' t have to look through reams of alterations to find one mistake. It is also a good idea to keep saving your program after making alterations just in case the power goes off for any reason. Whether it' s because grandma fell over the plug or because the blackout for World War Three has come, you are going to be very annoyed if several hours' worth of unsaved typing suddenly changes into a blank screen.

# CHAPTER 5

## Further Explanations and Understanding BASIC

Some of the BASIC used in creating the routines has already been explained in the previous chapter, but from now on we will be using and referring to more advanced techniques for programming. If you have been working through the book and want to learn more about the technicalities of BASIC then carry on with this chapter. In this chapter we will examine the way BASIC actually works.

### MORE PRINT ITEMS

In previous routines you may have noticed that various parts of the PRINT line are separated by semi-colons (;), these are, logically enough, called *print separators*, but they are not just there to break up the statement, they have a definite purpose. Try the following short routine:

```
10 FOR X=1 TO 9
20 PRINT X;
30 NEXT X
```

and notice the result - it should look like this:

```
123456789
```

Now try the routine without the semi-colon at the end of the PRINT statement:

```
10 FOR X=1 TO 9
20 PRINT X
30 NEXT X
```

The result is different - it looks like this:

```
1
2
3
4
5
6
7
8
9
```

Try it yet again in the following form:

```
10 FOR X=1 TO 9
20 PRINT X,
30 NEXT X
```

Once more the result is different:

```
1      2      3      4
5      6      7      8
9
```

From the above, we have noticed several things which can be summarised as follows:

1. A semi-colon causes the PRINT items to be PRINTed on after another, with no spaces between them.
2. A comma causes the PRINT items to be PRINTed on the same line but spaced out by the field width of 10 characters.
3. The absence of either a semi-colon or a comma causes an automatic line feed which means that each PRINT item is PRINTed on a separate line.

An apostrophe (') can also be used as a PRINT separator, and it is used to produced a line feed, so that:

```
10 PRINT "HELLO" ' "HOW ARE YOU?"
```

has the same effect as:

```
10 PRINT "HELLO"
20 PRINT "HOW ARE YOU?"
```

and looks like this:

```
HELLO
HOW ARE YOU?
```

We have already used PRINT TAB with two parameters and seen how you can use this to PRINT anywhere on the screen, but there is another form with only one parameter which PRINTs at a specific column on the screen. Note you cannot backspace on a line. If you typed:

```
10 PRINT TAB(20);"My name is";TAB(18);"Fred"
```

you would get:

```
My name is
Fred
```

## PRINTING VARIABLES

So far, we have only seen how to PRINT items enclosed in quotes (") but we can also PRINT variables. We've already met numeric variables, these are the ones used to store numbers, such as the score and screen positions. They can be PRINTed in just the same way as letters enclosed in quotes:

```
10 PRINT TAB(0,0);SCORE
```

and if, for instance, the number 2200 is stored in SCORE, then 2200 will be PRINTed in the top left-hand corner of the screen. You can mix the two sorts of item in one PRINT statement:

```
5 LET SCORE=2200
10 PRINT TAB(0,0);"SCORE=";SCORE
```

and this will PRINT the following in the top left-hand corner of the screen:

```
SCORE=2200
```

## PROGRAM CONTROL

Programs are controlled by asking questions and then directing the flow to different parts of the program depending on the answer. You ask a question in BASIC by saying IF (CONDITION) THEN (ACTION). CONDITION can be many things and you can use the following symbols (or operands) to make the test.

```
=   Equals
>   Greater than
<   Less than
<=  Less than or equal to
>=  Greater than or equal to
<>  Not equal to
```

Here are a few examples:

```
20 IF A>B THEN GOTO 40
30 PRINT A
40 PRINT B
```

If A holds a bigger number than B, the program will jump to line 40 and only the value of B will be PRINTed, but if A holds a number equal to or less than B then both the values of A and B will be PRINTed, because the program will carry on with line 30, instead of jumping to line 40.

```
20 IF A$<>B$ THEN STOP
30 PRINT A$
40 PRINT B$
```

In this case, we are testing to see if two string variables contain the same letters. If they don' t, the program will stop, and if they are identical, the program will PRINT out the contents of A\$ and B\$.

```
20 IF A-B=5 THEN GOSUB 1000
30 GOSUB 2000
```

Line 20 checks to see if the result of subtracting the value of B from the value of A is equal to 5, then the program goes to the subroutine at 1000 if it is or goes to the subroutine at 2000 if it isn' t. The contents of A and B are not affected by this.

In this case we could have used the more advanced form of IF statement which has the following format. IF (CONDITION) THEN (ACTION) ELSE (ANOTHER ACTION). Using this we could combine the above two statements into one line which said:

```
20 IF A-B=5 THEN GOSUB 1000 ELSE GOSUB 2000
```

Of course the tricky bit is that these two pieces of BASIC will not behave in the same way!!! The use of the THEN/ELSE construction has given us the equivalent of:

```
20 IF A-B THEN GOSUB 1000:GOTO 40
30 GOSUB 2000
```

Which is quite a different kettle of fish! So be very careful when using these statements. ACTION can also be many things and we have already used three different actions in the previous examples. Here are some more:

```
20 IF Y<>0 THEN Y=0
```

This resets Y to zero if it is some other number.

```
30 IF M$="Y" THEN SOUND 1,-15,100,1
```

This will produce a sound if the string variable M\$ contains the letter Y.

```
40 IF (X+Y)/2<Z THEN CLS
```

Line 40 will clear the screen (CLS) if the result of adding the values in X and Y together and then dividing by two is less than the value of Z. Calculations in brackets are always performed first; without the brackets multiply and divide will be done first and the value of Y would first be divided by 2, and then added to the value of X. This can be seen from the following example:

$$(3+7)/2 = 10/2 = 5$$

whereas:

$$3+7/2 = 3+3.5 = 6.5$$

Two or more conditions can be tested at one time, using AND, OR, EOR and NOT.

## AND

This is very similar to its meaning in English, and when it is used, the specified action will only happen when all the conditions are true. Thus:

```
20 IF X=Y AND M$="YES" THEN STOP
```

or even:

```
30 IF H=67 AND X$<>"NO" AND A-B=Z+W AND  
G$=K$ THEN CLS:GOTO 20
```

In the above line, the screen will only be cleared and the program then jump to line 20, when all four conditions are met.

## OR

This is really the opposite of AND, in that the action will be performed if either or both of the conditions is TRUE, but not if they are both FALSE. For example:

```
20 IF X=0 OR B=1 THEN SOUND 1,-15,0,1
```

The note will sound if X holds the value zero, if B holds the value 1, or if both are true.

## EOR

This is an awkward one and is best avoided unless you really know what you are up to. This is like an either/or construction in English and will give TRUE if either one or the other of the things in question is TRUE. But it gives FALSE if they are both TRUE (and also if both are FALSE).

## NOT

NOT is very useful. NOT is used on computers in exactly the same way as it is in English so ' NOT TRUE means FALSE and ' NOT 0' means -1 (since 0 is FALSE and -1 is TRUE). We can use this to see if flags are set:

```
1510 IF NOT DEAD THEN GOTO 1525
```

## FOR/NEXT

This is a statement we have used a lot, but it has not been explained before. The FOR statement always goes with a NEXT statement and it is a way of counting and performing an action a specified number of times, i.e.

```
10 FOR B=0 TO 7
.
.
.
.
100 NEXT B
```

will carry out the instructions in between eight times - the count starts from zero and goes up to, and including, seven in steps of one. It can be made to go up or down in other steps, for instance:

```

10 FOR K=10 TO 20 STEP 2
.
.
.
50 NEXT K

```

will mean that K will hold the following values as the program goes round the loop:

10,12,14,16,18,20

and the FOR . . . NEXT loop:

```

10 FOR T=100 TO 10 STEP -10
.
.
.
50 NEXT T

```

will give the following values of T:

100,90,80,70,60,50,40,30,20,10

This is one of the most frequently used constructions in BASIC so it might be worth playing with it for a while to make sure you really understand what' s going on. Right, that' s enough BASIC for now. In the next chapter we will deal with all the fun and frolics of how to define characters like the aliens and the players.



Above is a blank grid and next to it we' ve drawn a new alien so that you can see how to turn him into a set of numbers. If you look again, you can see the row of numbers across the top of the grid. Each square in the grid has a value, and that is given by the number above it, so to get the number for each line, you start at one end, and if the square is blank you move to the next, and if the square is blacked in you add it to your total. The first line is hence:

$$16 + 8 = 24$$

and the rest of the lines are as follows:

$$32 + 16 + 8 + 4 = 60$$

$$64 + 16 + 8 + 2 = 90$$

$$32 + 4 = 36$$

$$16 + 8 = 24$$

$$32 + 4 = 36$$

$$64 + 2 = 66$$

$$128 + 1 = 129$$

This is just like binary arithmetic, which isn' t too surprising. Binary is a number system based on zeroes and ones and it is the only number system that your micro-computer understands directly. It appears to understand decimal numbers but that is only because there is a program inside your micro-computer that converts everything - even the words - into binary numbers.

Now you know enough to be able to design your own graphics characters, so you can have different aliens or players. You can even design new shapes for use in the background. What we really need to know, though, is how the routine actually works. So let' s go back over the program in greater detail.

The REM statement we have already dealt with, and we know that REM is short for REMinder or REMark.

The VDU statement contains a list of numbers which are the decimal equivalent of the pattern of binary numbers. Binary arithmetic is used beacuse it is easy for a computer processor to recognise one of two states, on or off, making it possible to represent only the two numbers, 1 and 0. Memory is divided into bytes - there are a maximum of 65536 of these in your computer

and each byte is further subdivided into 8 bits. Each bit (short for binary digit) can have the values 0 or 1 and as you move along the byte towards the left, each bit is worth double that of the one before it - hence the sequence of numbers at the top of our graphics grid:

128,64,32,16,8,4,2,1

If each bit was set to 1, the number held in that byte would be 255 decimal, which is represented by 11111111 in binary. These bits are copied from memory in a special area (at address &C00) and stored there for future reference. When you PRINT a character the data is copied to the appropriate screen position and a point of light appears where each 1 bit is and a dark point where each 0 bit is. This means that when you PRINT out your character now you will get a little monster or whatever shape you designed.

Following this text is a utility program (a utility program is one that helps you to design and create other programs) to help you make up your own characters and change them around without using yards and yards of paper and wearing out lots of pencils. (After all, what's a computer for if not to make life easier?) The instructions for using it are as follows:

Your position in the grid is shown by an asterisk. To move it around, use the cursor keys (the arrow keys).

When you reach a square you want to change - either from black to white or white to black - press C and it will change.

After you are satisfied with the design, press S and the program will ask you in which character you wish to save your design. If you wished your design to replace character 128 then you would simply enter 128 followed by RETURN, and the character at the top of the screen would be replaced by your new character and you would be able to see what it looked like at proper size.

This is a longish program so be careful when you key it in.

*Character Generator Listing*

```

10 ON ERROR GOTO 670
20 MODE 4: DIM D 9
30 VDU 23,1,0;0;0;0;0;
40 *FX4,1
50 REPEAT
60   PROCsetup
70   REPEAT
80     PROCcursor
90     IF INKEY(-122) AND X>0 THEN X=
X-1
100     IF INKEY(-26) AND X<7 THEN X=X
+1
110     IF INKEY(-42) AND Y<8 THEN Y=Y
+1
120     IF INKEY(-58) AND Y>1 THEN Y=Y
-1
130     PROCcursor
140     IF INKEY(-83) THEN PROCchange(
X,Y)
150     IF INKEY(-35) THEN PROCedit
160     FOR I%=1 TO 8: PRINT TAB(5,I%+
10);D?I%;"  ": NEXT
170     UNTIL INKEY$(0)="S"
180     PROCinput:VDU 23,N
190     FOR I%=1 TO 8: VDU D?I% : NEXT
200 UNTIL FALSE
210 DEF PROCsetup
220 FOR I=0 TO 8:D?I=0:NEXT
230 CLS : X=7:Y=1:PROCcursor
240 PRINT TAB(5,1);STRING$(23,"*")
250 PRINT SPC(5);"* CHARACTER GENERATO
R * "
260 PRINT SPC(5);STRING$(23,"*") '
270 FOR I%=0 TO 8:GCOL 0,1
280     MOVE 320+32*I%,416:DRAW 320+I%*3

```

```

2,672
290     MOVE 320,32*I%+416:DRAW 576,I%*3
2+416
300 NEXT
310 PRINT TAB(20,11);"C TO CHANGE SQUA
RE"
320 PRINT TAB(20,12);"E TO EDIT CHARAC
TER"
330 PRINT TAB(20,13);"S TO SAVE CHARAC
TER"
340 ENDPROC
350 DEF PROCinput
360 *FX15,1
370 REPEAT
380     PRINT TAB(0,25);"WHICH CHARACTER
";
390     INPUT N: VDU 11
400 UNTIL N>31 AND N<256
410 PRINTSPC(17)
420 ENDPROC
430 DEF PROCedit
440 PROCinput
450 ?D=N
460 X%=D:Y%=D DIV 256:A%=10:CALL &FFF1
470 FOR I%=1 TO 8
480     P=D?I%
490     FOR K%=7 TO 0 STEP -1
500         IF P>=2^K% THEN P=P-2^K%:PROCF
ill(I%,K%)
510 NEXT :NEXT
520 ENDPROC
530 DEF PROCcursor
540 GCOL 3,1: MOVE (7-X)*32+320,(9-Y)*
32+416
550 VDU 5,42,4
560 ENDPROC
570 DEF PROCfill(Y,X)

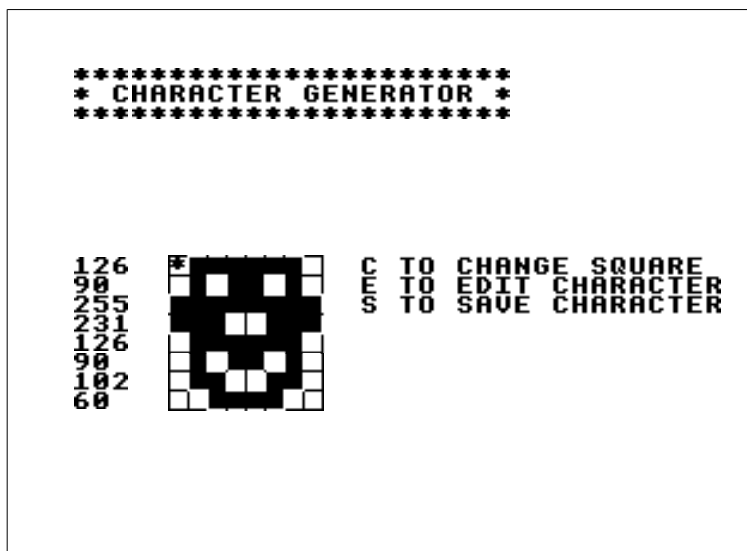
```

```

580 X%=(7-X)*32+320:Y%=(9-Y)*32+416
590 MOVE X%,Y%: MOVE X%+32,Y%
600 PLOT 85,X%,Y%-32:PLOT 85,X%+32,Y%-
32
610 MOVE X%,Y%-32:DRAW X%+32,Y%
620 ENDPROC
630 DEF PROCchange(X,Y)
640 PROCfill(Y,X)
650 D?Y=D?Y EOR 2^X
660 ENDPROC
670 *FX4,0
680 REPORT:PRINT" ";ERL:END

```

### Example Display





If you wanted to define this bomb as character 132 you could add the statement:

```
VDU 23,132,24,60,60,60,60,24,60,126
```

into one of the defining routines.

This would mean that every time we printed character 132 we would get a bomb. Now all we need to do is go through changing all PRINT statements in the bomb or fireball routine so that they are either VDU 132 statements or PRINT CHR\$(132); statements (the two are exactly equivalent). In this way you should be able to design many different types of missile for yourself.

The next program is a very short one to enable you to examine the user-defined graphics and see them in giant form with the decimal numbers used to create the character alongside each representation. To use it simply key it in and run it. It would be a useful program to SAVE for future use.

### *Character Display Listing*

```

10 MODE 4:DIM D 9
20 REPEAT
30   CLS
40   PRINT TAB(10,1);STRING$(20,"*")
50   PRINT SPC(10);"* GRAPHICS DISPLA
Y * "
60   PRINT SPC(10);STRING$(20,"*") '
70   REPEAT
80     PRINT"WHICH CHARACTER ";
90     INPUT N:VDU 11
100    UNTIL N>31 AND N<256
110    PRINT SPC(17)' ' : ?D=N
120    X%=D:Y%=D DIV 256:A%=10:CALL &FF
F1
130    FOR I%=1 TO 8: COLOUR 128
140      P=D?I%: PRINT P;"          ";
150      FOR K%=7 TO 0 STEP -1
```

```
160          IF P>=2^K% THEN P=P-2^K%:C=1
ELSE C=0
170          COLOUR 128+C:PRINT " ";
180          NEXT :PRINT
190          NEXT :COLOUR128
200          PRINT'"PRESS SPACE TO CONTINUE"
210          A$=GET$
220 UNTIL FALSE
230 END
```

# CHAPTER 7

## Arrays and Adventures

By now you should be quite familiar with the idea of a variable. We have been using variables to store scores, number of aliens and many other things. These variables are called simple variables - they contain one number only. If you want to store another number you either use another variable or overwrite what was in the first one. For example, going back to our cheesecake selling for a moment, suppose you wanted to keep track of your profit on cheesecake over a week. One way would be to have seven different variables, one for each day of the week. Then your program would be seven times as long, as it would be doing the same calculations on seven different variables. At the end of your program you would probably have a routine to print out the profit for all seven days that looked something like:

```
100 PRINT P1
110 PRINT P2
120 PRINT P3
130 PRINT P4
140 PRINT P5
150 PRINT P6
160 PRINT P7
```

Well, it would work, but there is a better way - using arrays.

An array is a collection of variables with the same name. But unlike simple variables, you cannot set them equal to anything until you have told the computer that you are going to use them. This is done with a DIM statement, and this is called DIMensioning the array, i.e. we are going to tell the computer what the size of the array is.

In our example we are going to need an array with seven elements. DIM P(7) then will create an array of seven elements which we can refer to as P(1),P(2) . . . P(7). Now the really clever bit about arrays is that the number inside the brackets (called the subscript) can be another variable! So we could store our cheesecake profits for each day in P(1) to P(7) and PRINT them

out using a routine which looked like this:

```
100 FOR N=1 TO 7
110 PRINT P(N)
120 NEXT N
```

This is much shorter than the previous method and is much easier to use. Similarly the program for calculating the profits could be done inside a loop instead of being written out seven times.

The array we have been dealing with so far has one dimension, with seven elements. We have used this DIMENSION for the days of the week. However, you can have arrays with as many dimensions as you like. Let's say we wanted to find out the most popular line of cheesecake (so that we can make more of the popular flavours and hence make more profit). We would need to set up a two DIMENSIONAL array. One dimension would have seven elements as before and the other dimension would have as many elements as we have flavours. Let's keep it simple and say that we have strawberry, chocolate, blackcurrant and plain - four flavours in all. We would want an array with four elements across for each of the seven elements down. So our DIM statement would be DIM P(7,4). The first day's profit for strawberry cheesecake would be stored in P(1,1), for chocolate it would be P(1,2), blackcurrant P(1,3) and plain P(1,4). The second day's profit would go into P(2,1), P(2,2), P(2,3) and P(2,4), and so on for the other days. To PRINT these out at the end of the program, you would use a routine like this:

```
100 FOR DAY=1 TO 7
110 FOR FLAVOUR=1 TO 4
120 PRINT P(DAY,FLAVOUR);" ";
130 NEXT FLAVOUR
140 PRINT
150 NEXT DAY
```

This would give us a neat table - flavours across and days down, with 28 entries in all. Can you imagine using 28 variables and 28 PRINT statements to do the same job?

## STRING VARIABLES AND STRING ARRAYS

Variables and arrays which contain numbers are called 'numeric' variables or arrays. This is not just a pointless piece of jargon; it distinguishes them from another, quite different, type of variable - a *string variable* - and another type of array - a *string array*.

We have briefly used string variables in our arcade game to PRINT a name as the background, but we have not really looked at them very closely.

Whereas a numeric variable holds a number, a string variable holds a string of characters. String arrays are simply arrays in which every element is a string. This is the one major difference between the two types and it is a very important difference. Strings are recognised by the computer when they are enclosed in quote marks. "FRED" is recognised by the computer as a string of four characters but FRED will be taken to be the name of a numeric variable. String variables have \$ added to the end of their names so we can tell the two types apart and it is very important to remember that you cannot mix them up.

If, for example you tried to say FRED="fred" the computer would reply "Type mismatch" error which is exactly what it is, since FRED is the name of a numeric variable and "fred" is a string. You will have to type the statement with a \$ added to the end of FRED before the computer will accept it. Obviously you will also get an error if you try to say FRED\$=10. Once again you will have mixed up the two different types of variable.

Digits are characters, though, so why can't we put them into string variables? Well, we can, and there are two functions which allow us to change variables from one sort to another. We could say A\$=STR\$(10). STR\$ is a function that gives the equivalent string of characters to the number in brackets. So in this case we could simply have said A\$="10". The advantage of using the function comes when you use another variable as the parameter inside brackets. You could say A\$=STR\$(SCORE), for example. This would give you the string of characters as A\$ that would have been PRINTed had you said PRINT SCORE. This sort of thing comes in very handy if you want to format numeric output. In the above example of displaying a table of profits, we could use it along with the function for measuring the length of the string

(LEN) to get all the numbers lined up. The program would look something like this:-

```
100 FOR DAY=0 TO 6
110 FOR FLAVOUR=0 TO 3
120 A$=STR$(P(DAY,FLAVOUR))
130 PRINT TAB(FLAVOUR*10-LEN(A$+10
140 NEXT FLAVOUR
150 NEXT DAY
```

Arrays and strings are used a lot in adventure games and this is what we' ll take a look at next. In the adventure game given in the next chapter we will see multi-DIMensional arrays used to keep track of things like the player' s position and the contents of rooms. The program in the next chapter builds up into a game in much the same way that the sections of Chapter 3 built up into an arcade game. Although there are some differences the process is sufficiently similar not to need a full example program.

## CHAPTER 8

### **Adventure Games, a Selection of Lego Bricks**

Just as we did in our arcade game we will begin by considering the basic building blocks we will need for our adventure game. The tasks which we will have to perform are as follows:

1. Initialisation.
2. Assign inventories.
3. Instructions.
4. Create the maze.
5. Describe the player's situation.
6. Player INPUT.
7. Check INPUT is legal.
8. Perform Instruction.
9. Print a response.
10. Check for end of game.
11. End of game message.
12. Round again.

To begin with we will write the simplest (well almost!) version of the game. We have chosen a scenario of a maze of dungeons. The player must search for a Crown of Emeralds left there aeons ago by the king of a long forgotten race of Troglodytes. The only "feature" of the game is that some of the passages from cave to cave have doors that are locked. To open these the player must find and take a key, of which there are several lying about in the maze. To prevent things from becoming too easy we will not allow any key to be used more than once - they will always either break or get stuck in the lock.

Step 1. Initialisation. OK! Let's start on the game. As before we will need a control program which follows the sequence set out above. Again we will simply use a PROC command where most of the blocks are needed. The REMs are there once again to REMind us what we are doing.

The control program is shown in listing 8.1. Type it in exactly as shown. The control program will be needed no matter which

version of the game we build so it might be a good idea to SAVE it on cassette so that you can use it again later.

### *Listing 8.1*

```

10 REM *****
20 REM * TREASURE TRAIL *
30 REM *****
40 REM *****
50 REM * INITIALISE BBC *
60 REM *****
70 MODE 7
80 DIM VERB$(9,2),NOUN$(11),OBJECT$(3
)
90 DIM ROOM(36,6),ADJECT$(5,2),P(5)
100 FOOD=FALSE : BATTERIES=FALSE : MN=
FALSE : KEY=0 : CRYSTAL=0 : MD=FALSE
110 WON=FALSE : LOST=FALSE : K=0
120 REM *****
130 REM * ASSIGN INVENTORIES *
140 REM *****
150 PROCinvent
160 REM *****
170 REM * INSTRUCTIONS *
180 REM *****
190 PROCinstructions
200 REM *****
210 REM * CREATE MAZE *
220 REM *****
230 PROCmaze : NR=TRUE
240 REM *****
250 REM * DESCRIBE ROOM *
260 REM *****
270 PROCdescribe
280 REM *****
290 REM * PLAYERS INPUT *
300 REM *****
310 PROCinput

```

```

320 REM *****
330 REM *   LEGAL INPUT?               *
340 REM *****
350 PROClegal : IF QUIT THEN STOP
360 REM *****
370 REM *   PEFORM INSTRUCTIONS       *
380 REM *****
390 IF LEGAL THEN PROCperform
400 REM *****
410 REM *   COMPUTERS RESPONSE       *
420 REM *****
430 PROCresponse
440 REM *****
450 REM *   CHECK END OF GAME         *
460 REM *****
470 IF NOT LOST AND NOT WON THEN 270
480 REM *****
490 REM *   END OF GAME               *
500 REM *****
510 IF WON THEN PROCwon ELSE PROClost
520 REM *****
530 REM *   PLAY AGAIN                *
540 REM *****
550 INPUT "DO YOU WANT ANOTHER GAME";U
$
560 IF LEFT$(U$,1) <> "N" THEN CLEAR: GO
TO 10
570 END

```

The lines numbered less than 100 simply initialise the computer. In other words they do all the things which only have to be done once. Remember that we have to DIMension all the arrays at the start of the program before we can use them. If we accidentally re-DIMension an array it will produce an error and stop the program, so it' s best to do all the DIMs at the start of the program.

Step 2. Assign Inventories. Now all we have to do is to write the procedures which are called from the main program. The first of these is given here.

*Listing 8.2*

```

1200 REM *****
1205 REM * ASSIGN INVENTORIES *
1210 REM *****
1215 DEF PROCinvent
1220 FOR N=1 TO 3
1225 READ VERB$(N,1), VERB$(N,2)
1230 NEXT N
1235 FOR N=1 TO 7
1240 READ NOUN$(N)
1245 NEXT N
1265 FOR N=1 TO 5
1270 READ ADJECT$(N,1), ADJECT$(N,2)
1275 NEXT N
1280 DATA GO,1234,TAKE,67,OPEN,5
1285 DATA NORTH,SOUTH,EAST,WEST,DOOR,KEY,EMERALD
1295 DATA DAMP,MISERABLE,COLD,DARK,SCARY,OPPRESSIVE,SMALL,GLOOMY,LARGE,DRAUGHTY
1300 ENDPROC

```

Type this in with the control program. This routine assigns the inventories. The inventories are arrays containing all the words (verbs and nouns) which the program needs to be able to understand.

Step 3. Give Instructions. Instructions are needed to tell the player what to do in the game and what the aim is. However it is essential not to give too much information here - it is supposed to be a game of exploration and adventure after all! Combine the lines of listing 8.3 with your program to date.

*Listing 8.3*

```

1000 REM *****
1010 REM * INSTRUCTIONS *
1020 REM *****
1030 DEF PROCInstructions
1040 FOR I=0 TO 1 : PRINT TAB(4,I);CHR$
141;"T R E A S U R E   T R A I L" : NEXT
I
1050 PRINT" YOU ARE IN THE DUNGEONS OF
GORM AND"'" YOU ARE SEARCHING FOR A"
1060 PRINT CHR$130;CHR$157;CHR$136;CHR$
132;" CROWN OF EMERALDS"
1070 PRINT" THE COMPUTER UNDERSTANDS TH
E FOLLOWING VERBS:-" '
1080 FOR N=1 TO 3
1090 PRINT TAB(12);VERB$(N,1): NEXT N
1180 PRINT: INPUT"PRESS 'RETURN' TO CON
TINUE" U$
1190 ENDPROC

```

Step 4. Create the Maze. It is essential to make absolutely certain that you could win. This routine creates the maze and places the target, the Crown of Emeralds and the player at RaNDom positions. There will always be a possible path from one to the other but the rest of the maze is decided at random. Listing 8.4 gives the simplest version of the maze creation routine.

```

1400 REM *****
1405 REM * CREATE MAZE *
1410 REM *****
1415 DEF PROCmaze
1420 CLS : PRINT TAB(13,10);CHR$ 136;"P
LEASE WAIT"
1425 FOR I=1 TO 36
1430 FOR J=1 TO 6
1435 ROOM(I,J)=-1
1440 NEXT
1445 NEXT

```

```

1460 CROWN=RND(36)
1465 REPEAT : PLAYER=RND(36) : UNTIL PL
AYER<>CROWN
1470 X=PLAYER
1475 R=RND(4)
1480 IF R=3 THEN PROCputgoody ELSE IF R
=4 THEN PROCputbaddy
1485 L$="1": IF KEY>0 THEN L$=L$+"2"
1495 L=LEN(L$): R=RND(L): P=VAL(MID$(L$
,R,1))
1500 IF P=2 THEN KEY=KEY-1
1510 Y=CROWN-X
1515 IF Y>5 THEN ROOM(X,4)=P: X=X+6 : R
OOM(X,3)=P: GOTO 1545
1520 IF Y<-5 THEN ROOM(X,3)=P : X=X-6 :
ROOM(X,4)=P:GOTO 1545
1525 IF Y>0 AND X/6<> X DIV 6 THEN ROOM
(X,5)=P:X=X+1:ROOM(X,6)=P:GOTO 1545
1530 IF Y>0 THEN ROOM(X,4)=P:X=X+6:ROOM
(X,3)=P:GOTO 1545
1535 IF Y<0 AND (X-1)/6<>(X-1) DIV 6 TH
EN ROOM(X,6)=P:X=X-1:ROOM(X,5)=P:GOTO 15
45
1540 IF Y<0 THEN ROOM(X,3)=P:X=X-6:ROOM(
X,4)=P
1545 IF X<>CROWN THEN 1475
1550 PROCrestofmaze
1555 ROOM(CROWN,1)=2
1565 CLS : ENDPROC
1570 REM *****
1575 REM * DEPOSIT OBJECT *
1580 REM *****
1585 DEF PROCputgoody
1590 L$="0"
1595 IF RND(1)>.5 THEN L$=L$+"1"
1615 L=LEN(L$):IF L>1 THEN R=RND(L) ELS
E R=1

```

```

1620 P=VAL (MID$ (L$, R, 1))
1625 IF P=1 THEN KEY=KEY+1
1640 ROOM (X, 1)=P
1645 ENDPROC
1650 REM *****
1655 REM * DEPOSIT HAZARD *
1660 REM *****
1665 DEF PROCputbaddy
1670 L$="0"
1675 IF RND (1)>.5 THEN L$=L$+"1"
1685 L=LEN (L$):IF L>1 THEN R=RND (L) ELSE
E R=1
1690 P=VAL (MID$ (L$, R, 1))
1700 ROOM (X, 2)=P
1705 ENDPROC
1710 REM *****
1715 REM * FINISH MAZE *
1720 REM *****
1725 DEF PROCrestofmaze
1730 B$="0101010100":H$="0000":L$="1224
"
1735 FOR N=1 TO 36
1740 FOR M=3 TO 6
1745 IF ROOM (N, M)=-1 THEN R=RND (4):
P=VAL (MID$ (L$, R, 1)):ROOM (N, M)=P:PROCcomple
ment
1750 NEXT M
1755 IF N<7 THEN ROOM (N, 3)=4
1760 IF N>30 THEN ROOM (N, 4)=4
1765 IF N/6=N DIV 6 THEN ROOM (N, 5)=4
1770 IF (N-1)/6=(N-1) DIV 6 THEN ROOM
(N, 6)=4
1775 IF ROOM (N, 1)=-1 THEN R=RND (10):R
OOM (N, 1)=VAL (MID$ (B$, R, 1))
1780 IF ROOM (N, 2)=-1 THEN R=RND (4):RO
OM (N, 2)=VAL (MID$ (H$, R, 1))
1785 NEXT N : MD=0

```

```

1790 ENDPROC
1795 REM *****
1800 REM * COMPLEMENTARY DOORS *
1805 REM *****
1810 DEF PROCcplement
1815 IF M=3 AND N>6 THEN ROOM(N-6,4)=P:
ENDPROC
1820 IF M=4 AND N<31 THEN ROOM(N+6,3)=P:
ENDPROC
1825 IF M=5 AND N/6<>N DIV 6 THEN ROOM(
N+1,6)=P:ENDPROC
1830 IF M=6 AND (N-1)/6<>(N-1) DIV 6 TH
EN ROOM(N-1),5)=P:ENDPROC
1835 ENDPROC

```

Step 5. Describe Situation. Describe the situation and surroundings to the player. At this point we embark on our adventure to the player. At this point we embark on our adventure, and it is to this point that the program will return after our latest attempt at derring-do has met with total failure. When we enter each cave the computer describes it and the objects therein are detailed. It is this section which must produce the excessive verbiage that is a pre-requisite of adventure games. This routine produces a lot of pleonastic purple prose with much repetition and even tautology. Much of the information is, of course, apocryphal. This task is dealt with by listing 8.5 which is given below. Once again we must key in the whole lot to add this function to our program.

#### *Listing 8.5*

```

2000 REM *****
2010 REM * DESCRIBE ROOM *
2020 REM *****
2030 DEF PROCdescribe
2050 IF NOT NR THEN 2100
2060 RK=RND(4)*6-5
2070 N=RND(5):M=RND(5)
2080 PRINT "YOU ENTER A ";ADJECT$(N,1);

```

```

", "; ADJECT$(M, 2);
2090 IF RND(1) > .5 THEN PRINT " CAVE." ELSE PRINT " PASSAGE."
2100 NR=FALSE :PRINT"YOU SEE: "; :L=ROOM(PLAYER,1) : M=ROOM(PLAYER,2)
2110 IF L=0 AND M=0 THEN PRINT "NOTHING"
"
2120 IF L=1 THEN PRINT TAB(9);MID$("LARGE RUSTY GOLDENWOODEN",RK,6);" KEY"
2130 IF L=2 THEN PRINT TAB(9);"THE CROWN OF EMERALDS"
2200 FOR M=3 TO 6
2210     L=ROOM(PLAYER,M)
2220     IF L=1 OR L=2 THEN PRINT "THERE IS A DOOR TO THE ";NOUN$(M-2)
2240 NEXT M
2250 ENDPROC

```

Step 6. Player' INPUT. We must allow the player to make his life and death decisions, carefully weigh the pros and cons and finally stumble blindly on. In other words, what do you want to do now? This routine will accept the commands of the player as typed in at the keyboard. We must add this section to our rapidly growing program (bear in mind that computer adventures are always fairly large programs so you might want to save your programs a few times along the way as an insurance policy against power failure). Listing 8.6 below contains the instructions to accept INPUT.

### *Listing 8.6*

```

2500 REM *****
2501 REM *  PLAYER INPUT  *
2502 REM *****

2505 DEF PROCinput
2510 VDU 131,157,132
2520 INPUT"WHAT NEXT ? "U$ :U$=U$+" "
2530 IF U$=" " THEN VDU 11:GOTO2510

```

```
2540 ENDPROC
```

Step 7. Check INPUT is Legal. This section checks that what you typed actually made sense, e.g. that it uses a legal verb like GO or TAKE, and in a legal manner, such as GO NORTH (GO BANANAS is not legal input). Type this section in from the next listing.

*Listing 8.7*

```
3000 REM *****
3010 REM *   LEGAL INPUT?   *
3020 REM *****
3030 DEF PROC legal
3040 V=0:RF=0
3050 LEGAL=FALSE:QUIT=FALSE
3060 IF INSTR(U$,"QUIT") THEN QUIT=TRUE
: ENDPROC
3070 N=0:REPEAT: N=N+1 : UNTIL MID$(U$,
N,1)=" "
3080 S$=LEFT$(U$,N) : Z$=MID$(U$,N)
3090 REPEAT: V=V+1

3100 UNTIL INSTR(S$,VERB$(V,1)) OR V=4
3110 IF V=4 THEN RF=1 : ENDPROC
3130 L$=VERB$(V,2)
3140 X=0:REPEAT : X=X+1
3150 N=EVAL("&" + MID$(L$,X,1))
3160 N1=INSTR(Z$,NOUN$(N))
3170 UNTIL N1 OR X=LEN(L$)
3180 IF N1 THEN LEGAL=TRUE ELSE RF=2
3190 ENDPROC
3200 DEF FN go(room,dir)
3210 IF dir=1 AND room>6 THEN =room-6
3220 IF dir=2 AND room<31 THEN =room+6
3230 IF dir=3 AND room/6<> room DIV 6 T
HEN =room+1
3240 IF dir=4 AND (room-1)/6<>(room-1)D
IV 6 THEN =room-1
```

```

3250 DEF FNopposite(dir)
3260 IF dir=2 OR dir=4 THEN =dir-1 ELSE
      =dir+1

```

Step 8. Perform Instruction. Well, we' ll try to! Just because GO NORTH seems like a reasonable idea doesn' t mean we can necessarily do it. There might be a blank wall in the way. There are so many possibilities that the routine is made up of many smaller routines which deal with individual words. This sort of further subdivision of tasks is a common feature in structured programs and makes both reading and writing them a lot easier.

### *Listing 6.8*

```

4000 REM *****
4010 REM * PERFORM INSTRUCTION *
4020 REM *****
4030 DEF PROCperform : RF=0: OK=FALSE
4040 GOSUB (4000+100*V)
4050 IF OK THEN RF=15
4060 ENDPROC
4070 ENDPROC
4100 REM *****
4110 REM * GO *
4120 REM *****
4140 DIR=N
4150 IF ROOM(PLAYER,N+2)=1 THEN OK=TRUE
:PLAYER=FNgo(PLAYER,DIR):NR=TRUE:RETURN
4160 IF ROOM(PLAYER,N+2)=2 THEN RF=4 EL
SE RF=3
4170 RETURN
4200 REM *****
4210 REM * TAKE *
4220 REM *****
4240 IF ROOM(PLAYER,1)<>N-5 THEN RF=5 :
RETURN
4250 OK=TRUE
4260 I=N-5: IF I>3 THEN I=I-1

```

```

4270 P(I)=P(I)+1
4280 ROOM(PLAYER,1)=0
4290 RETURN
4300 REM *****
4305 REM * OPEN *
4310 REM *****
4320 IF P(1)<1 THEN RF=6 : RETURN
4325 LD=FALSE : DIR=0
4330 FOR I=1 TO 4: IF INSTR(Z$,NOUN$(I)
) THEN DIR=I
4335 IF ROOM(PLAYER,I+2)=2 THEN LD=TR
UE
4340 NEXT I
4345 IF DIR=0 THEN RF=13: RETURN
4350 IF NOT LD THEN RF=7: RETURN
4355 IF ROOM(PLAYER,DIR+2)<>2 THEN RF=1
5:RETURN
4360 ROOM(PLAYER,DIR+2)=1 : OK=TRUE:T=F
Ngo(PLAYER,DIR):W=FNopposite(DIR)
4365 ROOM(T,W+2)=1
4370 P(1)=P(1)-1:K=-1:OK=TRUE
4375 RETURN

```

Step 9. PRINT a Response. By now we have figured out what has just happened with regard to the player's commands. We need to tell the player what he has (or more likely what he has not) achieved. Which response is given depends upon the RESPONSE flag set by the previous routines. The choice of response is dealt with by listing 8.9 so add this to your program.

## Listing 8.9

```

5000 REM *****
5010 REM * RESPONSE SECTION *
5020 REM *****
5030 DEF PROCresponse
5040 GOSUB (5200+RF*10)
5050 IF P(2) THEN WON=TRUE
5060 IF LOST THEN ENDPROC
5170 ENDPROC
5200 PRINT"WHAT?!":RETURN
5210 PRINT"I DO NOT KNOW THE VERB ";S$:
RETURN
5220 PRINT"YOU CANNOT ";S$;" THE ";Z$:R
ETURN
5230 PRINT"YOU CANNOT GO ";NOUN$(DIR):R
ETURN
5240 PRINT"THE DOOR IS LOCKED":RETURN
5250 PRINT"I SEE NO ";NOUN$(N);" HERE":
RETURN
5260 PRINT"BUT YOU DO NOT HAVE A KEY":R
ETURN
5270 PRINT"THERE IS NO LOCKED DOOR TO O
PEN HERE":RETURN
5280 RETURN
5290 PRINT"YOU DO NOT HAVE THE ";NOUN$(
N):RETURN
5350 PROCok:RETURN
5500 DEF PROCok
5510 PRINT"YOU ";U$
5520 IF K THEN PRINT"BUT THE KEY ";:R=R
ND(2)
5530 IF K AND R=1 THEN PRINT"BREAKS"
5540 IF K AND R=2 THEN PRINT"GETS STUCK
"' "YOU MUST LEAVE IT BEHIND"
5550 K=0
5580 ENDPROC

```

Step 10. Check for End of Game. If the game is neither won nor lost we have to loop back to the description routine. This task has been incorporated into the control program, so there is no need to have a separate LISTing for this routine.

Step 11. End of Game Message. There are two routines in this section of which only one will be called, depending on whether the game has been won or lost. We will need to enter both, though, to guard against the possibility of success. So type in both listings below (listing 8.11a and 8.11b)

*Listing 8.11a*

```

6000 REM *****
6010 REM * YOU LOST *
6020 REM *****
6030 DEF PROClost
6040 FOR I=1 TO 2
6050     PRINT CHR$141;CHR$129;"YOU HAVE
LOST THE GAME - BAD LUCK!!!"
6060 NEXT
6070 ENDPROC

```

*Listing 8.11b*

```

6100 REM *****
6110 REM * YOU WON *
6120 REM *****
6130 DEF PROCwon
6140 FOR I=1 TO 2
6150     VDU 130,157,132,141
6160     PRINT"CONGRATULATIONS - YOU HAVE
RECOVERED ";
6170     VDU 10,130,157,132,141
6180     PRIN"  THE FABULOUS CROWN OF EME
RALDS"
6190     VDU 10,130,157,132,141
6200     PRINT"                WELL DONE"

```

```

6210     IF I=1 THEN VDU 11,11,11,11
6220 NEXT
6230 FOR I=1 TO 255
6240     SOUND 1,-15,I,1:SOUND 2,-15,256-
I,1
6250 NEXT
6260 ENDPROC

```

Step 12. Round Again. Here we ask whether the player would like to chance his arm, and probably his neck, again. If he does want to play again then we re-start the game. This has again been incorporated into the control program. If you don't want to play again then the computer won't sulk, it will just respond with its usual unemotional >. So that means we have finally got the whole program typed in. It should look like the listing below (listing 8.12).

### *Listing 8.12*

```

10 REM *****
20 REM * TREASURE TRAIL *
30 REM *****
40 REM *****
50 REM * INITIALISE BBC *
60 REM *****
70 MODE 7
80 DIM VERB$(9,2),NOUN$(11),OBJECT$(3
)
90 DIM ROOM(36,6),ADJECT$(5,2),P(5)
100 FOOD=FALSE : BATTERIES=FALSE : MN=
FALSE : KEY=0 : CRYSTAL=0 : MD=FALSE
110 WON=FALSE : LOST=FALSE : K=0
120 REM *****
130 REM * ASSIGN INVENTORIES *
140 REM *****
150 PROCinvent
160 REM *****

```

```

170 REM * INSTRUCTIONS *
180 REM *****
190 PROCinstructions
200 REM *****
210 REM * CREATE MAZE *
220 REM *****
230 PROCmaze : NR=TRUE
240 REM *****
250 REM * DESCRIBE ROOM *
260 REM *****
270 PROCdescribe
280 REM *****
290 REM * PLAYERS INPUT *
300 REM *****
310 PROCinput
320 REM *****
330 REM * LEGAL INPUT? *
340 REM *****
350 PROClegal : IF QUIT THEN STOP
360 REM *****
370 REM * PERFORM INSTRUCTIONS *
380 REM *****
390 IF LEGAL THEN PROCperform
400 REM *****
410 REM * COMPUTERS RESPONSE *
420 REM *****
430 PROCresponse
440 REM *****
450 REM * CHECK END OF GAME *
460 REM *****
470 IF NOT LOST AND NOT WON THEN 270
480 REM *****
490 REM * END OF GAME *
500 REM *****
510 IF WON THEN PROCwon ELSE PROClost
520 REM *****
530 REM * PLAY AGAIN *

```

```

540 REM *****
550 INPUT "DO YOU WANT ANOTHER GAME";U
$
560 IF LEFT$(U$,1)<>"N" THEN CLEAR: GO
TO10
570 END
1000 REM *****
1010 REM * INSTRUCTIONS *
1020 REM *****
1030 DEF PROCInstructions
1040 FOR I=0 TO 1 : PRINT TAB(4,I);CHR$
141;"T R E A S U R E   T R A I L" : NEXT
I
1050 PRINT" YOU ARE IN THE DUNGEONS OF
GORM AND'"      YOU ARE SEARCHING FOR A"
1060 PRINT CHR$130;CHR$157;CHR$136;CHR$
132;"      CROWN OF EMERALDS"
1070 PRINT" THE COMPUTER UNDERSTANDS TH
E FOLLOWING  VERBS:-"'
1080 FOR N=1 TO 3
1090 PRINT TAB(12);VERB$(N,1): NEXT N
1180 PRINT: INPUT"PRESS 'RETURN' TO CON
TINUE" U$
1190 ENDPROC
1200 REM *****
1205 REM * ASSIGN INVENTORIES *
1210 REM *****
1215 DEF PROCinvent
1220 FOR N=1 TO 3
1225   READ VERB$(N,1),VERB$(N,2)
1230 NEXT N
1235 FOR N=1 TO 7
1240   READ NOUN$(N)
1245 NEXT N
1265 FOR N=1 TO 5
1270   READ ADJECT$(N,1),ADJECT$(N,2)
1275 NEXT N

```

```

1280 DATA GO,1234,TAKE,67,OPEN,5
1285 DATA NORTH,SOUTH,EAST,WEST,DOOR,KEY,EMERALD
1295 DATA DAMP,MISERABLE,COLD,DARK,SCARY,OPPRESSIVE,SMALL,GLOOMY,LARGE,DRAUGHTY
1300 ENDPROC
1400 REM *****
1405 REM * CREATE MAZE *
1410 REM *****
1415 DEF PROCmaze
1420 CLS : PRINT TAB(13,10);CHR$(136);"PLEASE WAIT"
1425 FOR I=1 TO 36
1430     FOR J=1 TO 6
1435         ROOM(I,J)=-1
1440     NEXT
1445 NEXT
1460 CROWN=RND(36)
1465 REPEAT : PLAYER=RND(36) : UNTIL PLAYER<>CROWN
1470 X=PLAYER
1475 R=RND(4)
1480 IF R=3 THEN PROCputgoody ELSE IF R=4 THEN PROCputbaddy
1485 L$="1": IF KEY>0 THEN L$=L$+"2"
1495 L=LEN(L$): R=RND(L): P=VAL(MID$(L$,R,1))
1500 IF P=2 THEN KEY=KEY-1
1510 Y=CROWN-X
1515 IF Y>5 THEN ROOM(X,4)=P: X=X+6 : ROOM(X,3)=P: GOTO 1545
1520 IF Y<-5 THEN ROOM(X,3)=P : X=X-6 : ROOM(X,4)=P:GOTO 1545
1525 IF Y>0 AND X/6<> X DIV 6 THEN ROOM(X,5)=P:X=X+1:ROOM(X,6)=P:GOTO 1545
1530 IF Y>0 THEN ROOM(X,4)=P:X=X+6:ROOM(X,3)=P:GOTO 1545

```

```

1535 IF Y<0 AND (X-1)/6<>(X-1) DIV 6 TH
EN ROOM(X,6)=P:X=X-1:ROOM(X,5)=P:GOTO 15
45
1540 IF Y<0 THEN ROOM(X,3)=P:X=X-6:ROOM(
X,4)=P
1545 IF X<>CROWN THEN 1475
1550 PROCrestofmaze
1555 ROOM(CROWN,1)=2
1565 CLS : ENDPROC
1570 REM *****
1575 REM * DEPOSIT OBJECT *
1580 REM *****
1585 DEF PROCputgoody
1590 L$="0"
1595 IF RND(1)>.5 THEN L$=L$+"1"
1615 L=LEN(L$):IF L>1 THEN R=RND(L) ELS
E R=1
1620 P=VAL(MID$(L$,R,1))
1625 IF P=1 THEN KEY=KEY+1
1640 ROOM(X,1)=P
1645 ENDPROC
1650 REM *****
1655 REM * DEPOSIT HAZARD *
1660 REM *****
1665 DEF PROCputbaddy
1670 L$="0"
1675 IF RND(1)>.5 THEN L$=L$+"1"
1685 L=LEN(L$):IF L>1 THEN R=RND(L) ELS
E R=1
1690 P=VAL(MID$(L$,R,1))
1700 ROOM(X,2)=P
1705 ENDPROC
1710 REM *****
1715 REM * FINISH MAZE *
1720 REM *****
1725 DEF PROCrestofmaze
1730 B$="0101010100":H$="0000":L$="1224

```

```

"
1735 FOR N=1 TO 36
1740   FOR M=3 TO 6
1745     IF ROOM(N,M)=-1 THEN R=RND(4):
P=VAL(MID$(L$,R,1)):ROOM(N,M)=P:PROCcomplement
1750   NEXT M
1755   IF N<7 THEN ROOM(N,3)=4
1760   IF N>30 THEN ROOM(N,4)=4
1765   IF N/6=N DIV 6 THEN ROOM(N,5)=4
1770   IF (N-1)/6=(N-1) DIV 6 THEN ROOM
(N,6)=4
1775   IF ROOM(N,1)=-1 THEN R=RND(10):R
OOM(N,1)=VAL(MID$(B$,R,1))
1780   IF ROOM(N,2)=-1 THEN R=RND(4):RO
OM(N,2)=VAL(MID$(H$,R,1))
1785 NEXT N : MD=0
1790 ENDPROC
1795 REM *****
1800 REM * COMPLEMENTARY DOORS *
1805 REM *****
1810 DEF PROCcomplement
1815 IF M=3 AND N>6 THEN ROOM(N-6,4)=P:
ENDPROC
1820 IF M=4 AND N<31 THEN ROOM(N+6,3)=P
:ENDPROC
1825 IF M=5 AND N/6<>N DIV 6 THEN ROOM(
N+1,6)=P:ENDPROC
1830 IF M=6 AND (N-1)/6<>(N-1) DIV 6 TH
EN ROOM(N-1,5)=P:ENDPROC
1835 ENDPROC
2000 REM *****
2010 REM * DESCRIBE ROOM *
2020 REM *****
2030 DEF PROCdescribe
2050 IF NOT NR THEN 2100
2060 RK=RND(4)*6-5

```

```

2070 N=RND(5):M=RND(5)
2080 PRINT "YOU ENTER A ";ADJECT$(N,1);
", ";ADJECT$(M,2);
2090 IF RND(1)>.5 THEN PRINT " CAVE." ELSE PRINT " PASSAGE."
2100 NR=FALSE :PRINT"YOU SEE: "; :L=ROOM(PLAYER,1): M=ROOM(PLAYER,2)
2110 IF L=0 AND M=0 THEN PRINT "NOTHING
"
2120 IF L=1 THEN PRINT TAB(9);MID$("LARGE RUSTY GOLDENWOODEN",RK,6);" KEY"
2130 IF L=2 THEN PRINT TAB(9);"THE CROWN OF EMERALDS"
2200 FOR M=3 TO 6
2210     L=ROOM(PLAYER,M)
2220     IF L=1 OR L=2 THEN PRINT "THERE IS A DOOR TO THE ";NOUN$(M-2)
2230 NEXT M
2240 ENDPROC
2500 REM *****
2501 REM * PLAYER INPUT *
2502 REM *****
2505 DEF PROCinput
2510 VDU 131,157,132
2520 INPUT"WHAT NEXT ? "U$:U$=U$+" "
2530 IF U$=" " THEN VDU 11:GOTO2510
2540 ENDPROC
3000 REM *****
3010 REM * LEGAL INPUT? *
3020 REM *****
3030 DEF PROClegal
3040 V=0:RF=0
3050 LEGAL=FALSE:QUIT=FALSE
3060 IF INSTR(U$,"QUIT") THEN QUIT=TRUE
: ENDPROC
3070 N=0:REPEAT: N=N+1 : UNTIL MID$(U$,N,1)=" "

```

```

3080 S$=LEFT$(U$,N) : Z$=MID$(U$,N)
3090 REPEAT: V=V+1
3100 UNTIL INSTR(S$,VERB$(V,1)) OR V=4
3110 IF V=4 THEN RF=1 : ENDPROC
3130 L$=VERB$(V,2)
3140 X=0:REPEAT : X=X+1
3150 N=EVAL("&" + MID$(L$,X,1))
3160 N1=INSTR(Z$,NOUN$(N))
3170 UNTIL N1 OR X=LEN(L$)
3180 IF N1 THEN LEGAL=TRUE ELSE RF=2
3190 ENDPROC
3200 DEF FNgo(room,dir)
3210 IF dir=1 AND room>6 THEN =room-6
3220 IF dir=2 AND room<31 THEN =room+6
3230 IF dir=3 AND room/6<> room DIV 6 T
HEN =room+1
3240 IF dir=4 AND (room-1)/6<>(room-1)D
IV 6 THEN =room-1
3250 DEF FNopposite(dir)
3260 IF dir=2 OR dir=4 THEN =dir-1 ELSE
=dir+1
4000 REM *****
4010 REM * PERFORM INSTRUCTION *
4020 REM *****
4030 DEF PROCperform : RF=0: OK=FALSE
4040 GOSUB (4000+100*V)
4050 IF OK THEN RF=15
4060 ENDPROC
4070 ENDPROC
4100 REM *****
4110 REM * GO *
4120 REM *****
4140 DIR=N
4150 IF ROOM(PLAYER,N+2)=1 THEN OK=TRUE
:PLAYER=FNgo(PLAYER,DIR):NR=TRUE:RETURN
4160 IF ROOM(PLAYER,N+2)=2 THEN RF=4 EL
SE RF=3

```

```

4170 RETURN
4200 REM *****
4210 REM * TAKE *
4220 REM *****
4240 IF ROOM(PLAYER,1)<>N-5 THEN RF=5 :
RETURN
4250 OK=TRUE
4260 I=N-5: IF I>3 THEN I=I-1
4270 P(I)=P(I)+1
4280 ROOM(PLAYER,1)=0
4290 RETURN
4300 REM *****
4305 REM * OPEN *
4310 REM *****
4320 IF P(1)<1 THEN RF=6 : RETURN
4325 LD=FALSE : DIR=0
4330 FOR I=1 TO 4: IF INSTR(Z$,NOUN$(I)
) THEN DIR=I
4335 IF ROOM(PLAYER,I+2)=2 THEN LD=TR
UE
4340 NEXT I
4345 IF DIR=0 THEN RF=13: RETURN
4350 IF NOT LD THEN RF=7: RETURN
4355 IF ROOM(PLAYER,DIR+2)<>2 THEN RF=1
5:RETURN
4360 ROOM(PLAYER,DIR+2)=1 : OK=TRUE:T=F
Ngo(PLAYER,DIR):W=FNopposite(DIR)
4365 ROOM(T,W+2)=1
4370 P(1)=P(1)-1:K=-1:OK=TRUE
4375 RETURN
5000 REM *****
5010 REM * RESPONSE SECTION *
5020 REM *****
5030 DEF PROCresponse
5040 GOSUB (5200+RF*10)
5050 IF P(2) THEN WON=TRUE
5060 IF LOST THEN ENDPROC

```

```

5170 ENDPROC
5200 PRINT"WHAT?!!":RETURN
5210 PRINT"I DO NOT KNOW THE VERB ";S$:
RETURN
5220 PRINT"YOU CANNOT ";S$;" THE ";Z$:R
ETURN
5230 PRINT"YOU CANNOT GO ";NOUN$(DIR):R
ETURN
5240 PRINT"THE DOOR IS LOCKED":RETURN
5250 PRINT"I SEE NO ";NOUN$(N);" HERE":
RETURN
5260 PRINT"BUT YOU DO NOT HAVE A KEY":R

ETURN
5270 PRINT"THERE IS NO LOCKED DOOR TO O
PEN HERE":RETURN
5280 RETURN
5290 PRINT"YOU DO NOT HAVE THE ";NOUN$(
N):RETURN
5330 PRINT"WHICH DIRECTION?":RETURN
5350 PROCok:RETURN
5500 DEF PROCok
5510 PRINT"YOU ";U$
5520 IF K THEN PRINT"BUT THE KEY ";:R=R
ND(2)
5530 IF K AND R=1 THEN PRINT"BREAKS"
5540 IF K AND R=2 THEN PRINT"GETS STUCK
""YOU MUST LEAVE IT BEHIND"
5550 K=0
5560 ENDPROC
6000 REM *****
6010 REM * YOU LOST *
6020 REM *****
6030 DEF PROClost
6040 FOR I=1 TO 2
6050 PRINT CHR$141;CHR$129;"YOU HAVE
LOST THE GAME - BAD LUCK!!!"

```

```

6060 NEXT
6070 ENDPROC
6100 REM *****
6110 REM * YOU WON *
6120 REM *****
6130 DEF PROCwon
6140 FOR I=1 TO 2
6150     VDU 130,157,132,141
6160     PRINT"CONGRATULATIONS - YOU HAVE
RECOVERED ";
6170     VDU 10,130,157,132,141
6180     PRIN"    THE FABULOUS CROWN OF EME
RALDS"
6190     VDU 10,130,157,132,141
6200     PRINT"                WELL DONE"
6210     IF I=1 THEN VDU 11,11,11,11
6220 NEXT
6230 FOR I=1 TO 255
6240     SOUND 1,-15,I,1:SOUND 2,-15,256-
I,1
6250 NEXT
6260 ENDPROC

```

We are now ready to play our game, but hang on . . . wouldn't it be a good idea to SAVE your game before anything nasty happens!!! When you've done that RUN the program and see if you can find the Crown of Emeralds. GOOD LUCK!

Now that we have our basic (no pun intended) game, we can think about adding various features to it. The features which can be added are limited only by your imagination (and the BBC's memory of course). In the following sections we will see how to add five sets of additional features and by the end of this section you should have a reasonable idea of how the system works. Chapter 9 goes into the details of planning behind the various routines and by the end of that chapter you should be able to make your own alterations.

The program is designed so that features can be stuck on to it like Lego Bricks - in a similar way to that in which we could add the ability to move up and down (for example) in our arcade

game. There is a difference, however, due to the very nature of the game. Whereas in the arcade game, the routine for (say) moving the alien didn't give a hoot whether you could move up and down or not, if the adventure game alterations need to be made to almost all sections when a new feature is added. This is like the suggested alteration to allow a high score to be kept in the arcade game. In that case we had to go through the program altering quite a few of the routines to allow for the new features. In just the same way we are going to have to go methodically through our adventure routines adding lines to each procedure.

## FOOD AND STRENGTH

The first feature which we add we will call FOOD & STRENGTH. The first thing to do to it is figure out how this feature is related to the game scenario.

We conjure up an evil force which pervades the chill caverns in which we are adventuring. The constant forces of cold and evil sap your physical and spiritual strength. To combat this we will have food which restores your fitness, body and soul. If however you do not find any food you will weaken and die!

The first section we have to add is given in listing 8.2a and this is adding to the inventory of objects in the dungeon. After all, if we are going to put food in the maze, the computer is going to need to know about it. So, with your original game in memory, add in the following lines.

### *Listing 8.2a*

```
1220 FOR N=1 TO 8
1235 FOR N=1 TO 8
1285 DATA NORTH,SOUTH,EAST,WEST,DOOR,KEY,EMERALD,FOOD
```

The next thing we need to do is to tell the player about the presence of the food and the evil spirit. We add these lines to the instruction routine (listing 8.3a)

*Listing 8.3a*

```

1080 FOR N=1 TO 8
1100 PRINT: INPUT"PRESS 'RETURN' TO CON
TINUE" U$
1110 CLS
1130 PRINT" THERE IS AN EVIL FORCE IN T
HE DUNGEONS WHICH DRAINS YOUR STRENGTH,
BUT IF YOU "
1140 PRINT" ARE LUCKY ENOUGH TO FIND TH
E MAGIC FOOD YOU MAY SURVIVE" : FOOD=TRU
E
1150 PRINT" ONLY THE LIGHT FROM YOUR
1170 PRINT" GOOD LUCK!! "

```

We now need to set the strength of the player and also change the "create maze" routine so that it distributes some food around the place as well as other things. Add the couple of lines given in listing 8.4a to take care of this.

*Listing 8.4a*

```

1450 STRENGTH=1000
1610 IF RND(1)>.7 THEN L$=L$+"3"
1730 B$="0310100310":H$="0000":L$="1234"

```

The next thing to add is a line to make the computer tell us if there is actually some food in the room when we enter it. This is done by the line in listing 8.5a

*Listing 8.5a*

```

2140 IF L=3 THEN PRINT TAB(9); "SOME FOO
D "

```

Of course at some stage we are going to want to EAT FOOD, so the computer must recognise this as a legal command. Add these lines to the legality checking procedures (listing 8.7a)

```

3100 UNTIL INSTR(S$,VERB$(V,1)) OR V=5
3110 IF V=5 THEN RF=1 : ENDPROC

```

Similarly, we must add a routine to deal with an EAT command. Listing 8.8a has the lines to alter this section to call a new routine which deals with EATing.

#### *Listing 8.8a*

```

4400 REM *****
4410 REM * EAST *
4420 REM *****
4430 IF MN THEN RETURN
4440 Y=N-5: IF Y>3 THEN Y=Y-1
4450 IF P(Y)<1 THEN RF=9: RETURN
4460 IF N<>8 THEN RF=10: RETURN
4470 OK=TRUE: P(Y)=P(Y)-1: STRENGTH=100
0
4480 RETURN

```

Finally to make life more fun add listing 8.9a to the response section. This will deal with PRINTing messages for the various eating habits possible and also add lines to warn the player of impending doom.

#### *Listing 8.9a*

```

5070 IF NOT FOOD THEN 5130
5080 STRENGTH=STRENGTH-30:IF STRENGTH>5
00 THEN 5130
5090 IF STRENGTH>400 THEN PRINT"YOUR ST
RENGTH IS FADING": GOTO 5130
5095 IF STRENGTH>300 THEN PRINT"YOU ARE
GETTING VERY WEAK":GOTO 5130
5100 IF STRENGTH>200 THEN PRINT"YOUR ST
RENGTH IS RAPIDLY DIMINISHING - YOU CAN
NOT GO ON MUCH LONGER":GOTO 5130
5105 IF STRENGTH>100 THEN PRINT"IF YOU

```

```

DON'T EAT SOON YOU'VE HAD IT!!!":GOTO 51
30
  5110 IF STRENGTH>0 THEN PRINT"YOU ARE O
N YOUR LAST LEGS MATE!!!!":GOTO 5130
  5115 PRINT"YOU HAVE DIED OF EXHAUSTION"
:LOST=TRUE:ENDPROC
  5130 ENDPROC
  5300 PRINT"YOU EAT THE ";NOUN$(N); " AND
YOU CHOKE"'"TO DEATH!!!":LOST=TRUE:RETU
RN

```

That completes our first modification. RUN the program and convince yourself that no disasters have occurred.

## TORCH AND BATTERIES

Our next modification deals with TORCH & BATTERIES supplies.

You have a torch, and it's just as well, because the resident goblins don't like the light. What they would like, though, is to eat you. In fact they think that you're the best thing since sliced bread and preferably between slices as a snack for them. If your torch goes out then . . . The problem is that your batteries only last for 15 minutes, so unless you can find some more within that time, you will be eaten.

Well, that should put the pressure on a bit. Once again we will need to make modifications to several routines and as before the first place to add things is in the inventory. Add listing 8.2b

### *Listing 8.2b*

```

1455 T%=0:TIME=0
1730 B$="0310100340":H$="0000":L$="1224
"

```

We need to tell the player too (unless you are feeling heartless) so add the lines of listing 8.3b to the instructions routine.

*Listing 8.3b*

```

1150 PRINT "    ONLY THE LIGHT FROM YOUR
TORCH STOPS  THE GOBLINS FROM EATING YOU
, AND YOUR "
1160 PRINT " BATTERIES ONLY LAST 15 MINU
TES!" : BATTERIES=TRUE

```

Now we need to let the maze creation routine know there are extra objects to be distributed and we must also set the TIME variable to allow us to keep track of the number of turns you' ve had. Add listing 8.4b

*Listing 8.4b*

```

1455 T%=0:TIME=0
1730 B$="0310100340":H$="0000":L$="1224
"

```

Add the following to the room description routine.

*Listing 8.5b*

```

2150 IF L=4 THEN PRINT TAB(9); "SOME BAT
TERIES "

```

Listing 8.8b will save your b(e)acon if you pick up some batteries.

*Listing 8.8b*

```

4250 OK=TRUE: IF N=9 THEN TIME=0: GOTO
4280

```

Now give the computer a few words to say to tell you what a mess you' re making of the game.

*Listing 8.9b*

```

5130 IF NOT BATTERIES THEN ENDPROC
5140 T=TIME/6000
5150 IF T>15 THEN PRINT"YOUR TORCH HAS
GONE OUT, THE GOBLINS EATYOU AND PICK TH
EIR TEETH WITH YOUR BONES":LOST=TRUE: EN
PROC
5160 IF T>9 THEN PRINT"YOUR TORCH WILL
ONLY LAST ";INT(15-T); " MORE MINUTES"
5170 ENDPROC

```

And that' s it! Try the game again and see if you survive.

## TROLLS, RUN, FIGHT AND TELEPORT

If you have not met the goblins yet you might be feeling lonely wandering around the maze, so how about introducing some trolls to keep you company!

Trolls are lurking in the caves, and you never know when you are about to stumble upon one of these tin, rubbery and loathsome creatures. When you do you will have to decide whether to run or fight - if you try to do anything else he will attack you anyway. Fortunately you can usually beat him in a fight (although this might not be the case if your strength is low). If you find yourself cornered you have one other option - teleport - but heaven knows where you will end up if you use it, and it weakens you considerably to do so.

This modification has brought in the three new verbs RUN, FIGHT and TELEPORT and one new noun TROLL. Also it seems that tangling with a troll will affect a player' s strength (as will teleporting) and that the presence of a troll will prevent him from picking up objects, opening doors etc. We begin to see how closely interrelated things become in adventure games as soon as they have more than one or two features. Implementing this feature follows the same pattern as the previous modifications.

First of all we add the new words in listing 8.2c.

*Listing 8.2c*

```

1220 FOR N=1 TO 7
1255 READ OBJECT$(1)
1280 DATA GO,1234,TAKE,6789,OPEN,5,EAT,
6789,RUN,,FIGHT,,TELEPORT,
1290 DATA TROLL

```

Next we give the player a cryptic hint of what is to come with the addition of listing 8.3c to the instructions.

*Listing 8.3c*

```

1080 FOR N=1 TO 7
1120 PRINT " THERE ARE HIDDEN HAZARDS IN
THE ' ' " DUNGEONS WHICH YOU WILL HAVE TO "
' " OVERCOME!! "

```

Again we alter the maze routine to deposit a few trolls on its way to creating a maze.

*Listing 8.4c*

```

1675 IF RND(1)>.5 THEN L$=L$+"1 "
1730 B$="0310100340":H$="0100":L$="1224
"

```

Next we make certain that the computer is actually going to tell you when you bump into a troll. Add listing 8.5c

*Listing 8.5c*

```

2040 MN=FALSE
2180 IF M=1 THEN PRINT TAB(9); "A SMELLY
TROLL": MN=TRUE

```

As before we have to alter the routine to recognise valid commands so that it will accept these words. This is done by listing 8.7c.

*Listing 8.7c*

```

3100 UNTIL INSTR(S$,VERB$(V,1)) OR V=8
3110 IF V=8 THEN RF=1 : ENDPROC
3120 IF V>4 AND V<8 THEN LEGAL=TRUE : E
NDPROC

```

You have probably noticed by now that the routine to perform verbs has one smaller procedure for each verb. The following listing contains procedures for each of RUN, FIGHT and TELEPORT. Type in all of the listing given below.

*Listing 8.8c*

```

4060 IF MN AND RF<>11 AND RF<16 THEN RF
=16
4130 IF MN THEN RETURN
4230 IF MN THEN RETURN
4315 IF MN THEN RETURN
4430 IF MN THEN RETURN
4500 REM *****
4510 REM * RUN *
4520 REM *****
4530 DIR=0
4540 IF ROOM(PLAYER,2)=0 THEN RF=0: RET
URN
4550 FOR D=3 TO 6 : IF ROOM(PLAYER,D)=1
THEN DIR=D-2
4560 NEXT D

```

```

4570 IF DIR=0 THEN RF=19: RETURN
4580 PLAYER=FNgo (PLAYER, DIR): RF=17
4590 RETURN
4600 REM *****
4610 REM * FIGHT *
4620 REM *****
4630 IF NOT MN THEN RF=0: RETURN
4640 ROOM (PLAYER, 2)=0: STRENGTH=STRENGTH
-RND (100)
4650 RF=18: RETURN
4700 REM *****
4710 REM * TELEPORT *
4720 REM *****
4730 NR=TRUE: PLAYER=RND (36)
4740 STRENGTH=STRENGTH-200
4750 RF=15: RETURN

```

Finally we add listing 8.9c to the response section and we' re finished.

### *Listing 8.9c*

```

5360 PROCattack: RETURN
5370 PRINT "YOU RUN BLINDLY": RETURN
5380 PRINT "YOU FIGHT THE BEAST AND SMASH IN HIS SKULL. HE VANISHES IN A PUFF OF GREASY SMOKE.": RETURN
5390 PRINT "YOU CANNOT RUN - THE ENTRANCES ARE ALL CLOSED": RETURN
5600 DEF PROCattack
5610 PRINT "YOU TRY TO "; U$' " BUT YOU ARE ATTACKED BY THE "
5620 R=RND (5)
5630 IF ROOM (PLAYER, 2)=1 THEN PRINT MID $ ("OBNOXIOUS STINKING DISGUSTING FLATULENT LOATHSOME ", R*10-9, 10); " TROLL"
5640 IF ROOM (PLAYER, 2)=2 THEN PRINT MID $ ("GHASTLY HORRIBLE BIBULOUS HIDEOUS SADIS

```

```

TIC",R*8-7,8);" MONSTER"
5650 ROOM(PLAYER,2)=0:MN=FALSE
5660 STRENGTH=STRENGTH-(100+RND(100))
5670 IF STRENGTH<1 THEN PRINT"YOU ARE T
O WEAK TO KILL HIM, HE RENDS YOU TO MI
NCEMEAT"
5680 IF STRENGTH>0 THEN PRINT"YOU MASH
HIM TO A SMEAR - HIS BLOOD SOAKS IN
TO THE DUST"
5690 ENDPROC

```

RUN the program again and look out for the trolls!

## MONSTERS AND MAGIC DUST

Well, things are starting to hot up a bit now, aren' t they? Now for another action packed feature.

Monsters are found in the dungeons. These have the same effect on you as trolls but nastier. There are also little piles of magic dust in the labyrinths. If you throw this at the monsters then they will vanish.

This introduces another verb, THROW, and two more nouns, MONSTER and DUST, so the first thing to do is add these to the inventory section.

### *Listing 8.2d*

```

1220 FOR N=1 TO 8
1235 FOR N=1 TO 10
1250 FOR N=1 TO 2
1255 READ OBJECT$(N)
1260 NEXT N
1280 DATA GO,1234,TAKE,6789A,OPEN,5,EAT
,6789A,RUN,,FIGHT,,TELEPORT,,THROW,6789A
1285 DATA NORTH,SOUTH,EAST,WEST,DOOR,KE
Y,EMERALD,FOOD,BATTERIES,MAGIC DUST

```

```
1290 DATA TROLL, MONSTER
```

Add listing 8.3d to tell the player he can THROW things.

*Listing 8.3d*

```
1080FORN N=1 TO 8
```

As usual we have to change the maze creation routine so that it puts the new objects into the maze. We will also make sure that there is always a monster guarding the Crown of Emeralds. To do this we add the next listing.

*Listing 8.4d*

```
1560 ROOM(CROWN, 2)=2
1605 IF RND(1)>.6 THEN L$=L$+"5"
1635 IF P=5 THEN MD=MD+1
1680 IF MD>0 AND RND(1)>.8 THEN L$=L$+"
2"
1695 IF P=2 THEN MD=MD-1
1730 B$="0310105340":H$="0120":L$="1224
"
```

Now we need to add listing 8.5d to make sure the player is fully informed when he enters a room.

*Listing 8.5d*

```
2160 IF L=5 THEN PRINT TAB(9); "SOME MAG
IC DUST"
2190 IF M=2 THEN PRINT TAB(9); "A HIDEOU
S MONSTER": MN=TRUE
```

Now we must extend the legal input check procedure to check that you have thrown something which is throwable (e.g. not a door). Add listing 8.7d to the program.

## Listing 8.7d

```

3100 UNTIL INSTR(S$,VERB$(V,1)) OR V=9
3110 IF V=9 THEN RF=1 : ENDPROC

```

With a new verb we must obviously have a new action procedure to perform the required task. The listing for THROW is given in listing 8.8d.

*Listing 8.8d*

```

4800 REM *****
4805 REM *   THROW   *
4810 REM *****
4815 Y=N-5: IF Y<>3 THEN Y=Y-1
4820 IF P(Y)<1 THEN RF=9: RETURN
4825 P(Y)=P(Y)-1
4830 REM LOOK FOR TARGET
4835 M=1
4840 IF INSTR(Z$,OBJECT$(M)) THEN 4850
ELSE M=M+1
4845 IF M<4 THEN 4840 ELSE M=0
4850 O=M
4855 P(Y)=P(Y)-1
4860 MD=(N>9):IF NOT MD THEN 4870
4865 IF O<3 THEN ROOM(PLAYER,2)=0: OK=TRUE:MN=FALSE:RETURN
4870 RF=13:IF O<3 THEN RF=12 : RETURN
4900 RETURN

```

Finally for this section we add listing 8.9d to allow the BBC to make the appropriate responses.

*Listing 8.9d*

```

5640 IF ROOM(PLAYER,2)=2 THEN PRINT MID
$( "GHASTLY HORRIBLEBIBULOUSHIDEOUS SADIS
TIC",R*8-7,8); " MONSTER"

```

OK, try the game again, and try throwing objects other than magic dust. Also, try throwing things at objects other than monsters.

## CRYSTALS AND SHIMMERING CURTAINS

Our final feature is the introduction of shimmering curtains of impassable energy.

Some of the doorways in the cave are blocked by curtains of shimmering energy. These can be neutralised by magic dust, but you may need to save that for the monsters. Instead, try to find a crystal, which will act as a key and totally remove the curtain, leaving an open doorway.

No prizes by now for guessing that the first thing we do is add the new words to the inventory. This is done in listing 8.2e

### *Listing 8.2e*

```

1235 FOR N=1 TO 11
1250 FOR N=1 TO 3
1280 DATA GO,1234,TAKE,6789AB,OPEN,5,EA
T,6789AB,RUN,,FIGHT,,TELEPORT,,THROW,678
9AB
1285 DATA NORTH,SOUTH,EAST,WEST,DOOR,KE
Y,EMERALD,FOOD,BATTERIES,MAGIC,DUST,CRYSTAL
1290 DATA TROLL,MONSTER,SHIMMERING CURTAIN

```

I will leave it up to you to decide what you' re going to add in the instructions and we will move straight on to the "create maze" routine. In this section we must make the computer deposit articles around the dungeon, which we do by adding listing 8.4e.

*Listing 8.4e*

```

1490 IF CRYSTAL>0 THEN L$=L$+"3"
1505 IF P=3 THEN CRYSTAL=CRYSTAL-1
1600 IF RND(1)>.6 THEN L$=L$+"6"
1730 B$="0310165340":H$="0120":L$="1234
"

```

Enter listing 8.5e to describe these new occurrences when we meet them.

*Listing 8.5e*

```

2170 IF L=6 THEN PRINT TAB(9); "A SHINY
CRYSTAL"
2230 IF L=3 THEN PRINT "THERE IS A CURT
AIN OF SHIMMERING ENERGY TO THE ";NOUN$(
M-2)

```

With all these options for THROW the procedure to check this is getting a little complicated, but don't worry about this, we will explain it all later. For the moment just add the next list to your program.

*Listing 8.8e*

```

4875 DIR=0
4880 REPEAT:DIR=DIR+1
4885 UNTIL DIR=5 OR INSTR(Z$,NOUN$(DI
R))
4890 IF DIR=5 THEN RF=131: RETURN
4895 ROOM(PLAYER,DIR+2)=1:ROOM(FNgo(PLA
YER,DIR),FNopposite(DIR)+2)=1:OK=TRUE

```

We now add just one more response to section 9 of the program and that's it.

*Listing 8.9e*

```
5340 PRINT "THERE IS NO SHIMMERING CURT  
AIN TO THE "; NOUN$(DIR) : RETURN
```

I hope you enjoy the challenge of surviving and beating this final version of our adventure game.

## CHAPTER 9

### Further Adventures

The previous chapter explained, in a general way, how the game worked by looking at each routine. In this chapter we will take a closer look at how each of the procedures work and, more importantly, at how the program works as a whole. By the end of this chapter you should be well prepared for adding features of your own to the game, and possibly even to be ready to write your own adventure games.

Before we get into the routines themselves, we shall have to look at the data, or knowledge, upon which they work. This is stored in the inventories, which represent the computer's knowledge of the world of the dungeons.

The inventories are made up of three string arrays: VERB\$, NOUN\$ and OBJECT\$. VERB\$ is a two-dimensional array of strings. If we think of it as a table of strings then in the first column we have all the verbs which the program can recognise and in the second column a string of numbers which correspond to the elements in the NOUN\$ array which can legally follow that verb. This can be seen in figure 9.1. NOUN\$ is an array of strings (a two dimensional array of characters) which has the names of all the objects that the computer recognises. The numbers in the second string of VERB\$ are the subscript numbers of the appropriate noun(s) in NOUN\$, however you might be wondering why there are letters in some of the lists of numbers. This is because we couldn't put 10 or 11 into the string without them being thought of as two numbers, so we use HEXadecimal notation of A for 10 and B for 11. If we look at the string of numbers corresponding to GO we see that it has a list of numbers, "1234". Looking at NOUN\$(1) to NOUN\$(4) we can see that these contain NORTH, SOUTH, EAST and WEST. So we could legally command the computer to GO in any of these directions. The list of names is given in full in figure 9.2 and for ease of reference we will label the elements in HEXadecimal (this notation is explained in the BBC User Guide).

*Figure 9.1*

	1	2
1	GO	1234
2	TAKE	6789AB
3	OPEN	5
4	EAT	6789AB
5	RUN	
6	FIGHT	
7	TELEPORT	
8	THROW	6789AB

*Figure 9.2*

1	NORTH
2	SOUTH
3	EAST
4	WEST
5	DOOR
6	KEY
7	EMERALDS
8	FOOD
9	BATTERIES
A	MAGIC DUST
B	CRYSTAL

These tables can be used to see what INPUTs are legal, but not necessarily feasible. We can see that the player may EAT any of the objects from 6 to B, but of course anything but FOOD will choke him.

When we look at the routine that checks the legality of a player's INPUT we can see how it performs the process of cross checking the contents of the arrays. The third array OBJECT\$ is very similar to NOUN\$, but the objects in this array can only be referred to indirectly using the THROW verb. The contents of this array are shown in figure 9.3

*Figure 9.3*

- 1 TROLL
- 2 MONSTER
- 3 SHIMMERING CURTAIN

Another "data structure" (to use the lingo) which we must look at before going on to examine the various PROCedures, is the array which stores the information about the maze itself. The game assumes a maze of 36 rooms/caves/dungeons which are arranged into a 6 x 6 block. If we think about what we need to know about each room, we find that there are six essential pieces of information.

- 1. Is there a "friendly" object?
- 2. Is there a "hostile" object?
- 3. What kind of entranceway is to the NORTH?
- 4. What kind of entranceway is to the SOUTH?
- 5. What kind of entranceway is to the EAST?
- 6. What kind of entranceway is to the WEST?

This means that we need an array of 36 by 6 numbers and you can see that we have DIMensioned ROOM(36,6). Within this array we can think of each room as having a number between 1 and 36, and to see if room number 4 (say) has a friendly object (e.g. a key) we would look at ROOM(4,1) and see if it is equal to 1. Now all we need is a convention for representing the various objects, and the convention we have chosen is as follows:

In ROOM(4,1) - Friendly objects

- 1 = KEY
- 2 = EMERALD CROWN
- 3 = FOOD
- 4 = BATTERIES
- 5 = MAGIC DUST
- 6 = CRYSTAL

In ROOM(4,2) - Hostile objects

- 1 = TROLL
- 2 = MONSTER

In ROOM(4,3) to ROOM(4,6) - Entranceways

- 1 = OPEN DOORWAY
- 2 = LOCKED DOORWAY
- 3 = SHIMMERING CURTAIN
- 4 = WALL

The array ADJECT\$ contains a series of adjectives which are then chosen at random to describe a room which the player has just entered.

The last array we use in the game is the numeric array GOT(5). This is used to store the number of items that the player has in his possession.

- GOT(1) = Number of Keys
- GOT(2) = Number of Emerald Crowns
- GOT(3) = Number of Food Packs
- GOT(4) = Number of Piles of Magic Dust
- GOT(5) = Number of Crystals

Now that we know how the information is stored we can look at how the various subroutines make use of it.

The first subroutine assigns the inventories, i.e. it reads all the verbs, etc., from the DATA statements in the routine into the arrays VERB\$, NOUN\$, OBJECT\$ and ADJECT\$. This should be easy to follow as it is very similar to the way DATA was read in the sections of our arcade game which defined graphics characters.

The next section, the instructions, is also pretty simple and self explanatory. We have already learnt about PRINT statements and FOR . . . NEXT loops so we' ll go straight on to the next section.

By contrast, the next section, creating the maze, is probably the most complicated routine in the program. Don' t turn over, though, it' s still quite easy to follow what' s going on. The routine begins by choosing a random position for the Crown of Emeralds and the player. These are stored in the variables CROWN and PLAYER. The next thing the computer does is to "walk" from the player' s position to the Crown Room, hanging various types of doors and depositing various objects on the way. This is done in

such a way as to ensure a viable path exists through the maze.

It does this by first calling at RaNDom either a routine to deposit an object or a routine to deposit a hazard. These routines make a RaNDom choice between the various types of object or sometimes deposit a -1 which stands for nothing. The routine which deposits objects keep a record of items deposited, so that, for example, the number of keys is stored in the variable KEY.

The next thing the routine does is to choose what kind of door to put up. The choice available depends on what objects have been deposited so far. For example, if KEY is equal to one or more the character "2" (for locked door) will be added to the list of possibilities (stored in L\$). After all we wouldn't want to leave you trying to get through a locked door if there wasn't a key on this side of it. When we place a locked door we must assume that the player will use up a key on the door so we knock one off the number of KEY\$ available.

How do we find the routine from the player to the crown though? Well, we do this by starting the computer's position at the player (the computer's position is stored in K). The difference between the player's position and that of the crown is stored in X. Bearing in mind that we are using 6 x 6 grid of rooms, it should be clear that if Y is bigger than six then the Crown Room must be negative and more than six then the room is to the North. Similarly we can tell whether we need to go East or West if the difference is less than six. A look at figure 9.4 should make all this clearer.

*Figure 9.4 The Maze*

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	03
31	32	33	34	35	36

Once the direction has been decided upon the computer puts the chosen door on the chosen wall, "spirits" itself through the door into the next room and - very important - puts the same type of door on the other side. (It would be a bit silly to have a door that

was locked on one side and wide open on the other.)

Once the Crown Room is reached the Crown of Emeralds is deposited and a monster is placed to guard it (if you have added this feature). The computer now goes through the rest of the maze room by room and puts up RaNDom entrances and deposits RaNDom objects wherever it finds blank walls or empty rooms. Note that we stop doors being placed which would link up to the outside world.

Well, you can see that building the maze is going to be quite a job. This is why we put up a message saying PLEASE WAIT. The computer can take a couple of seconds to do all this work and a player might think it had stopped, compared to its usual instant response, if there was no message.

Having created the maze, the command program now moves into the main loop, which will be repeated until the game is either won or lost. The first section in this loop calls the routine to describe the room the player is currently in. If the player has just entered a new room then the routine will also give a short description of the room itself. This condition is flagged by the variable NR (for new room). If it is TRUE the routine will describe the room, but if it is FALSE it will only reiterate the room' s contents.

The description of the room, the first part of the routine, is done by choosing two adjectives at random from ADJECT\$ and including these in the PRINT statement. The word CAVE or PASSAGE is tagged on to the end at random.

The next part of the routine simply describes which (if any) friendly objects are in the room and also which hostile objects are present. We then need to describe each of the possible exits in turn. To add variety to the description we give a selection of descriptions which can be applied to keys: Rusty, large, golden or wooden, and one of these is chosen at random IF ROOM(PLAYER,1)=1, indicating the presence of a key.

Once we have informed the player of his situation we give him the chance to tell the computer what he wants to do. This is very simple. You will notice however that we do not just INPUT the player' s instructions. We also want the player' s typing to stand out on the screen so we PRINT the prompt with a teletext control character at the end to make the player' s INPUT appear green on

the screen after the INPUT statement. The instruction is now in US\$ ready for processing.

The next routine checks the player' s instruction to see if it is legal. This procedure uses another function of the BBC in this checking. This function is INSTR, which searches through a string to discover whether another string is contained in it. The string to be searched is the first parameter and the string in question is the second. The function returns the position of the first character in the search string of the (smaller) string we were searching for. If the string isn' t contained then of course we get 0 returned.

This is first tried to see if the INPUT contains the word QUIT. If it does then there' s no point in checking any further, so we just set the QUIT flag and ENDPROC out of this procedure. We don' t really care where "QUIT" occurs in the INPUT so we just check whether INSTR(US,"QUIT") is greater than 0. After all the player may have typed "QUIT" or even "I !\*@@!\* QUIT", and in either case we need to do the same thing.

Of course most of the time the player will not have quit, and so the legal check routine has to flex its muscles a little more. The procedure first searches for the first word in the sentence by dropping any leading spaces and then searching for the space after the verb. The word thus found ought to be one of the verbs in VERB\$ so we look through that trying to find a match. If there' s no match we set the response flag (RF) accordingly and RETURN. Otherwise we drop the first word and search for the existence of one of the nouns from NOUN\$. This is done only for those NOUN\$ which can legally follow the verb (for RUN, FIGHT and TELEPORT we can ignore the rest of the INPUT because they are complete commands on their own). If one of the legal nouns is found then we set V to the appropriate subscript value of the VERB in VERB\$ and N to the value of the NOUN found in NOUN\$. The LEGAL flag will also be set accordingly at this point. If the verb is THROW then we also need to see if there is an object named which we can throw the noun at. We now extract the noun from the instruction and search for a match with one of the items in OBJECT\$. Note that though this has no effect on the legality of the command it would be nice to have something to aim at. If you are silly enough to type THROW MAGIC DUST then you will simply throw it away.

If, at this stage, LEGAL is FALSE then the program will carry

on to give a response and loop back. If the INPUT was legal then we want to try to perform the instruction. The fact that the INPUT was legal doesn't necessarily mean that it can be done; we might have commanded TAKE THE KEY, but we won't be able to if there is no key in sight. There are clearly many possible situations to be considered and for this reason the task of performing the various verbs has been split into separate smaller procedures, one for each of the verbs. The first part of the 'perform instruction' routine sets the OK flag to FALSE, and then uses GOSUB with a calculated line number to perform the appropriate procedure. These procedures will set the OK flag to -1 or TRUE if the instruction was carried out and no special message is required.

All the routines to action verbs check to see if there is a nasty monster in the room, because if there is then you won't be allowed to do anything apart from deal with them. Assuming there is no beastly then the procedures will perform your instruction, updating the number of possessions or your position as appropriate. The response flag will be set to indicate the required message to go with your action and strings are placed in Z\$, Z1\$ and U\$ for use in these messages by some of the routines. For example, the GO routine sets Z\$ to the direction you wanted to move in for use with the message "YOU CANNOT GO";Z\$.

It is now up to the response section of the program to either tell you what has happened or poke fun at your silly actions. In this section we also check the player's possessions and if he has the Crown of Emeralds then the WON flag is set. If either the WON or LOST flags are set then the program goes to ENDPROC.

At this point the TIME is updated and various messages regarding the durability of your torch and body will be PRINTed if your batteries are running out or strength is low.

The last three routines are only called at the end of a game. The first two simply PRINT out a message to commiserate with or congratulate the player depending on whether or not he won. After this the control program asks if you wish to play another game. If you accept then the game RUNs itself again. If you decline the opportunity the control program moves on to its final statement - END.



# CHAPTER 10

## Some Parting Remarks.

Well by now you've probably learnt quite a lot about writing and altering the sort of games we have discussed in this book. There should be nothing left to hold you back from writing your own games now. You are now in a position to write your own BASIC games and save your hard-earned pennies for only the best, top-quality, machine code, commercial games. Before we leave this introductory book behind though it is probably a good idea to have a look at one of the most important and much used BASIC functions in game writing. Random numbers have been used all through this book and we have never quite got round to looking at them in detail. They are used a lot because they can help you produce unexpected events, and that is far more interesting than having a game that does exactly the same thing all the time. Without random numbers the aliens would always appear in the same place at the top of the screen and the maze in the adventure would be totally predictable. Of course we could write so complex a program that we wouldn't know which one of the many parts of the program we were playing against but this would take up a lot of memory and, more importantly, a lot of effort from the people writing the game.

The BASIC word associated with random numbers is RND, which on its own produces a random number between -2 billion and +2 billion. However this is of little practical use, and it is more usual to use RND with a parameter to suitably modify the range of output. RND(1) gives us a random number between 0 and 1 and although it can give a value of 0 it cannot reach 1 (the highest it will ever get is about 0.999999999). Look at the range of numbers with this little program:

```
100 PRINT RND(1)
110 GOTO 100
```

You should see many different numbers between 0 and 1 with no discernable pattern. Actually if you carry on long enough the

pattern would repeat after 424967296 numbers. This is because RND is really only a pseudo-random function. The computer actually calculates the next random number from the current one by adding a large prime non-divisor of  $2^{32}$  then reducing this modulo  $2^{32}$  to give a number between 0 and 4294967296. The random number returned is either this number interpreted in two's complement notation for an integer (i.e. for RND without parameters) or this number divided by 4284967286 for RND(1), so it's all very rational really!!

To get a number in a different range, you can use a number other than 1 to get an integer in the range 1 to n (where n is the parameter in brackets after RND). We can simulate the throwing of a die in this way.

Here is a short program to do this:

```
10 REM PROGRAM TO THROW A DIE
20 PRINT RND(6)
30 GOTO 20
```

The above program will give random numbers in the range 1 to 6. Of course if we wanted to simulate two dice being thrown we would have to produce numbers in the range 2 to 12 but would work it out by calculating two numbers in the range 1 to 6 and adding them together, not by producing a new function call along the lines of INT(RND(11)+1).

Well, that's enough about random numbers. If you have spent enough time playing with (and altering) the games in this book you will be feeling ready to use BASIC in the development of your own games and adventures. Who knows, you may even decide to start writing programs to keep track of your bank balance and such like. Whatever you decide to do remember to keep clearly in mind the overall job. Always split your task up into smaller sub-tasks and never try to solve all the details of one sub-task before you even have an idea what the rest of the tasks are going to be. If you approach programming, or even writing a book, in this way, you will find that you can easily get through even the longest of jobs. This sort of approach is known as structured programming and although BASIC is not intrinsically a structured language we can still approach the building of a program in an orderly fashion. It would almost certainly be worth your while getting a book on

structured BASIC programming to help you expand your knowledge of working BASIC. Remember to go for one that has plenty of programming examples and if possible one with exercises that you can work through. If you got on well using the BASIC blocks in this book then maybe you should consider a career in programming!

# APPENDIX ONE

## Arcade Game Variables

ALIENS	Number of aliens left to fight or dodge - starts off at 10.
AMMO	Amount of ammunition left. This starts off at 10.
BX	The horizontal position of the bomb or bullet.
BY	The vertical position of the bomb or bullet.
DEAD	A flag to indicate whether the current alien is still alive or not and whether a new one must be started.
DOWN	Flag used to signify if downward movement is allowed or not.
FINISH	This is the end-of-game flag. If FINISH is FALSE then the game is still in progress. If FINISH is TRUE then it's all over.
FIRE	Flag used to signify if firing is enabled.
FUEL	Amount of fuel left. This starts off at 10.
HIT	Flag used to indicate a collision. Either an alien has been shot or you have crashed.
HYPER	Flag used to signify if hyper-space movement is allowed or not.
I	Variable used in loops.
J	Variable used in loops.
KEY	Flag to indicate whether or not the fire button has been pressed.
LEFT	Flag used to indicate whether left movement is allowed or not.
PAST	Counter to see how many aliens have reached the bottom of the screen and got past your defences.
RIGHT	Flag used to indicate whether right movement is allowed or not.
SCORE	This holds your score and is updated by the scoring routine.
UP	Flag used to indicate whether upward movement is allowed or not.

X	Variable used in reading data for defining characters.
XA	Horizontal position of the alien.
OXA	The old horizontal position of the alien.
YA	Vertical position of the alien. (The old vertical position is not stored since it is always 10 more than the current position.)
XP	Horizontal position of the player.
OXp	The old horizontal position of the player.
YP	Vertical position of the player.
OYP	The old vertical position of the player.

## APPENDIX TWO

### Adventure Game Variables

VERBS(8,2,8)	Contains the verbs and a string of subscripts of the nouns to which it can be copied.
NOUN\$(11,10)	Contains the nouns referring to each of the items in the game.
OBJECTS\$(3,20)	Contains the names of the objects which can be the target of the THROW command.
ROOM(36,6)	Contains the data for the contents of each of the rooms in the maze.
ADJECT\$(5,2,10)	Contains the adjectives used in describing the rooms of the maze.
P(5)	Counts the number of each object in player's possession.
FOOD	Flag to see if food is available.
BATTERIES	Flag for batteries.
MN	Flag for monsters.
KEY	Counter for number of keys deposited.
CRYSTAL	Counter for number of crystals deposited.
MD	Magic dust flag.
NR	New room flag.
QUIT	Flag for quitting programs.
LEGAL	Flag to indicate legal INPUT from player.
LOST	Flag to indicate player has lost game.
WON	Flag to indicate player has won game.
U\$	User's input.
N	Miscellaneous loops and the Noun found in the user's INPUT.
V	The subscript of the verb found in the user's INPUT.
STRENGTH	Strength of player.
T%	Time used up for batteries.
CROWN	Position of crown in maze.

PLAYER	Player' s position in maze.
X	Computer' s position in maze (used when constructing maze).
Y	Difference between co-ordinates of computer and crown.
R	Used for general random numbers.
L\$	List of possible objects for deposition.
P	Object about to be Placed.
B\$	List of friendly objects for deposition.
H\$	List of Horrible monsters for deposition.
Z\$,Z1\$	String used in response section.
RF	Response Flag indicates which response is to be made.
DIR	Direction indicator.
T	Temporary position variable.
OK	Flag for successful (or otherwise) performance of instruction

## RESPONSES

1	WHAT?!
2	I DO NOT KNOW THE VERB . . .
3	YOU CANNOT GO . . . THE . . .
4	YOU CANNOT GO . . .
5	THE DOOR IS LOCKED.
6	I SEE NO . . . HERE.
7	BUT YOU DO NOT HAVE A KEY.
8	THERE IS NO LOCKED DOOR TO OPEN HERE.
9	Not used. (Left free for further expansion).
10	YOU DO NOT HAVE THE . . .
11	YOU EAT THE . . . AND CHOKE TO DEATH!!
12	YOU THROW . . . AT THE . . . HE EATS IT AND LAUGHS. HA HA HA.
13	YOU THROW . . . AT THE . . . , I HOPE IT MAKES YOU FEEL BETTER.

14 PLEASE GIVE A DIRECTION.  
15 THERE IS NO SHIMMERING CURTAIN  
TO THE . . .  
16 OK, YOU . . .  
17 YOU TRY TO . . . BUT YOU ARE  
ATTACKED BY THE . . .  
18 YOU RUN.  
19 YOU FIGHT THE BEAST, YOU STAVE IN  
HIS SKULL, HE CRUMBLES TO DUST.  
20 YOU CANNOT RUN - THE ENTRANCES  
ARE CLOSED.

# APPENDIX THREE

## ASCII Character Set

<i>Code</i>	<i>Character</i>
0	NOTHING
1	NEXT CHARACTER TO PRINTER
2	PRINTER ON
3	PRINTER OFF
4	SEPARATE CURSORS
5	JOIN CURSORS
6	ENABLE VDU
7	BEEP
8	CURSOR BACK
9	CURSOR FORWARD
10	CURSOR DOWN
11	CURSOR UP
12	CLEAR TEXT
13	CARRIAGE RETURN
14	PAGED MODE ON
15	PAGED MODE OFF
16	CLEAR GRAPHICS
17	DEFINE TEXT COLOUR
18	DEFINE GRAPHICS COLOUR
19	DEFINE LOGICAL COLOURS
20	DEFAULT LOGICAL COLOURS
21	DISABLE VDU
22	SELECT MODE
23	REDEFINE CHARACTER
24	DEFINE GRAPHICS AREA
25	PLOT TO GRAPHICS AREA
26	RESTORE DEFAULT WINDOWS
27	ESCAPE
28	DEFINE TEXT AREA
29	DEFINE GRAPHICS ORIGIN

<i>Code</i>	<i>Character</i>	<i>Code</i>	<i>Character</i>
30	CURSOR HOME	67	C
31	MOVE TEXT CURSOR	68	D
32	SPACE	69	E
33	!	70	F
34	"	71	G
35	#	72	H
36	\$	73	I
37	%	74	J
38	&	75	K
39	'	76	L
40	(	77	M
41	)	78	N
42	*	79	O
43	+	80	P
44	,	81	Q
45	- (minus sign)	82	R
46	.	83	S
47	/	84	T
48	0	85	U
49	1	86	V
50	2	87	W
51	3	88	X
52	4	89	Y
53	5	90	Z
54	6	91	[
55	7	92	\
56	8	93	]
57	9	94	↑
58	:	95	—
59	;	96	£
60	<	97	a
61	=	98	b
62	>	99	c
63	?	100	d
64	@	101	e
65	A	102	f
66	B	103	g

<i>Code</i>	<i>Character</i>
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z
123	{
124	
125	}
126	~
127	DELETE

# APPENDIX FOUR

## Decimal/Binary Conversion Tables

0	00000000	31	00011111
1	00000001	32	00100000
2	00000010	33	00100001
3	00000011	34	00100010
4	00000100	35	00100011
5	00000101	36	00100100
6	00000110	37	00100101
7	00000111	38	00100110
8	00001000	39	00100111
9	00001001	40	00101000
10	00001010	41	00101001
11	00001011	42	00101010
12	00001100	43	00101011
13	00001101	44	00101100
14	00001110	45	00101101
15	00001111	46	00101110
16	00010000	47	00101111
17	00010001	48	00110000
18	00010010	49	00110001
19	00010011	50	00110010
20	00010100	51	00110011
21	00010101	52	00110100
22	00010110	53	00110101
23	00010111	54	00110110
24	00011000	55	00110111
25	00011001	56	00111000
26	00011010	57	00111001
27	00011011	58	00111010
28	00011100	59	00111011
29	00011101	60	00111100
30	00011110	61	00111101

62	00111110	100	01100100
63	00111111	101	01100101
64	01000000	102	01100110
65	01000001	103	01100111
66	01000010	104	01101000
67	01000011	105	01101001
68	01000100	106	01101010
69	01000101	107	01101011
70	01000110	108	01101100
71	01000111	109	01101101
72	01001000	110	01101110
73	01001001	111	01101111
74	01001010	112	01110000
75	01001011	113	01110001
76	01001100	114	01110010
77	01001101	115	01110011
78	01001110	116	01110100
79	01001111	117	01110101
80	01010000	118	01110110
81	01010001	119	01110111
82	01010010	120	01111000
83	01010011	121	01111001
84	01010100	122	01111010
85	01010101	123	01111011
86	01010110	124	01111100
87	01010111	125	01111101
88	01011000	126	01111110
89	01011001	127	01111111
90	01011010	128	10000000
91	01011011	129	10000001
92	01011100	130	10000010
93	01011101	131	10000011
94	01011110	132	10000100
95	01011111	133	10000101
96	01100000	134	10000110
97	01100001	135	10000111
98	01100010	136	10001000
99	01100011	137	10001001

138	10001010	176	10110000
139	10001011	177	10110001
140	10001100	178	10110010
141	10001101	179	10110011
142	10001110	180	10110100
143	10001111	181	10110101
144	10010000	182	10110110
145	10010001	183	10110111
146	10010010	184	10111000
147	10010011	185	10111001
148	10010100	186	10111010
149	10010101	187	10111011
150	10010110	188	10111100
151	10010111	189	10111101
152	10011000	190	10111110
153	10011001	191	10111111
154	10011010	192	11000000
155	10011011	193	11000001
156	10011100	194	11000010
157	10011101	195	11000011
158	10011110	196	11000100
159	10011111	197	11000101
160	10100000	198	11000110
161	10100001	199	11000111
162	10100010	200	11001000
163	10100011	201	11001001
164	10100100	202	11001010
165	10100101	203	11001011
166	10100110	204	11001100
167	10100111	205	11001101
168	10101000	206	11001110
169	10101001	207	11001111
170	10101010	208	11010000
171	10101011	209	11010001
172	10101100	210	11010010
173	10101101	211	11010011
174	10101110	212	11010100
175	10101111	213	11010101

214	11010110	252	11111100
215	11010111	253	11111101
216	11011000	254	11111110
217	11011001	255	11111111
218	11011010		
219	11011011		
220	11011100		
221	11011101		
222	11011110		
223	11011111		
224	11100000		
225	11100001		
226	11100010		
227	11100011		
228	11100100		
229	11100101		
230	11100110		
231	11100111		
232	11101000		
233	11101001		
234	11101010		
235	11101011		
236	11101100		
237	11101101		
238	11101110		
239	11101111		
240	11110000		
241	11110001		
242	11110010		
243	11110011		
244	11110100		
245	11110101		
246	11110110		
247	11110111		
248	11111000		
249	11111001		
250	11111010		
251	11111011		

Robert Erskine & Humphrey Walwyn with Paul Stanley and Michael Bews  
60 Programs for the Sinclair ZX Spectrum £5.95

Robert Erskine & Humphrey Walwyn with Paul Stanley and Michael Bews  
60 Programs for the BBC Micro £5.95

Robert Erskine & Humphrey Walwyn with Paul Stanley and Michael Bews  
60 Programs for the Dragon 32 £5.95

Robert Erskine & Humphrey Walwyn with Paul Stanley and Michael Bews  
60 Programs for the Oric 1 £5.95

Ian Adamson  
**The Companion to the Oric 1** £5.95

Geoff Wheelwright  
**The Companion to the BBC Micro** £4.95

Jean Frost  
**Instant Arcade Games for the Sinclair ZX Spectrum** £3.95

Jean Frost  
**Instant Arcade Games for the BBC Micro** £3.95

Jean Frost  
**Instant Arcade Games for the Dragon 32** £3.95

J.J. Clessa  
**Micropuzzles** £2.95

All these books are available at your local bookshop or newsagent, or can  
be ordered direct from the publisher. Indicate the number of copies  
required and fill in the form below.

Name \_\_\_\_\_  
(Block letters please)

Address \_\_\_\_\_

---

Send to Pan Books (CS Department), PO Box 40, Basingstoke, Hants  
Please enclose remittance to the value of the cover price plus:  
35p for the first book plus 15p per copy for each additional book ordered  
to a maximum charge of £1.25 to cover postage and packing.  
Applicable only in the UK.

While every effort is made to keep prices low, it is sometimes necessary  
to increase prices at short notice. Pan Books reserve the right to show  
on covers and charge new retail prices which may differ from those  
advertised in the text or elsewhere.