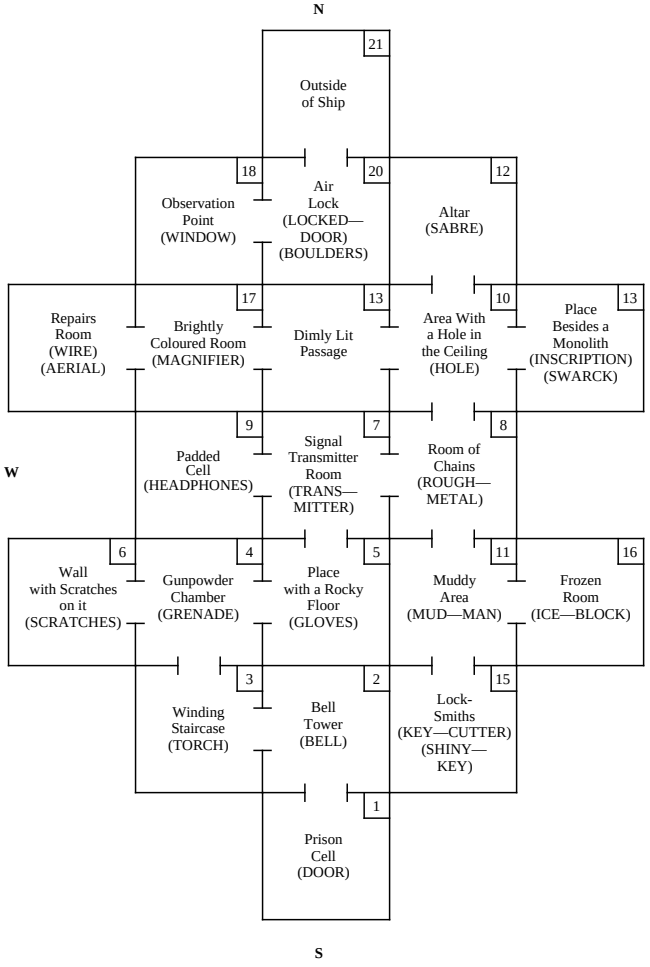
CHAPTER 2
A MODEL ADVENTURER

The Plan

The first stage in writing an adventure is the making of a plan with the layout and labelling of the rooms with a system of numerical values for the objects in each room along with the room number and name:



As can be seen from the diagram, the names of the rooms are given along with the room number for each room, and the objects associated with the individual rooms are placed in brackets inside the room square.

Although the method for numbering rooms may at first appear not to be logical, the rooms are in fact in a logical order. Room “I” is where the player starts from, and room “2” is the first room that can be entered from this room. Rooms “5” and “6” are to the right and left of room “4” — I give preference in numbering to north then south then east and then west. I build up the numbering system by looking for the room with the smallest numerical value which has an exit, or exits, to an unnumbered room, or rooms (numbering is done in the above priority). The above method works because rooms beside each other have the lowest possible difference between their numerical values — this is a key feature, especially in larger adventures, in the movement between rooms, which is dealt with later in this chapter.

If this seems a little confusing, then the step by step process for deciding the numerical values for the rooms in the given example is as follows: the only exit from room “1” is room “2”, and the only unnumbered room from room “2” is room “3”. From room “3”, the only yet unnumbered room, is room “4”. However, from room “4” there are two unnumbered rooms, the eastern being numbered first as room “5”, and the western second as room “6”. At this point, the lowest numbered room is room “5”, and the only room off this room without a number is labelled room “7”. Room “6” is a dead end, so no further rooms may be numbered from this room. Room “7” is another room which has two unnumbered rooms off it, the eastern being labelled room “8”, and the western, which is a dead end, is labelled room “9”. From room eight, there are two rooms yet to be numbered — the northern is designated room “l6”, and the southern, room “11”. The room with the lowest numerical value here, with other rooms off it wanting numbers, is room “1”, and rooms “12”, “13”, and “14” are to the north, east, and west of it — rooms “12” and “13” are both dead ends. After dealing with room “16”, the next room to be dealt with is room “11” : the rooms south and east of this room are two more dead ends, and are numbered with “15” and “l6” respectively. After this fairly complex branching, the only room left that is not a dead end, with a yet unnumbered room off it, is room “14”, and the room off this, is given the next number, which is “17”. Two rooms requiring numbers are north and west of room “I 7”, and are labelled with “18” and “19”, the latter being a dead end. Off room “18” is room “2fl” to the east, and off room “2fj” to the north is the final room, room “21”.

WORKING OUT OF THE DISPLAY

1) Display of the Room Name — Type in the following lines into your computer (if you own a computer other than the BBC Micro, then see Chapter Four for conversion notes):

```
  30    A=1
 150    CLS
 160    RESTORE
1310    DATA Prison cell, Bell tower, Winding staircase,Gunpowder
        chamber, Place with a rocky floor,Wall with scratches on it,
        Signal transmitter room, Room of chains, Padded cell, Area
        with a hole in the ceiling, Muddy area,Altar
1320    DATA Place beside a monolith, Dimly lit passage, Locksmiths,
        Frozen room, Brightly coloured room, Observation point,
        Repairs room, Air lock, Outside of ship
```

Variable "A" contains the room number for the adventure (i.e. the room in which the character is situated), and is defined as being equal to "1" in line
3a.

In line 150 the screen is cleared.

Line 160 "restores" the data pointer to the beginning of the first item of data in the program, so that the next item of data read will be that after the first time that "DATA" appears in the program.

Line 300 is a FOR/NEXT loop which counts along the first items of data until the number in "A" is reached; when this number is reached, it returns with the name of the room corresponding to it, and this name is then stored in the variable "A$".

Line 310 has the job of printing out "A$", the name of the room. "VDU 31,0,3" moves the cursor three lines down from the top of the screen, and to the first character in the line — this is necessary on the BBC Micro, since the first line or two is sometimes off the television screen. "CHR$130" is a control code which tells the computer to print in green alphanumerics — the semi-colon allows "A$" to be printed directly after the control code.

Lines 1310 and 1320 are the first two lines of data in the adventure listing, and they contain the twenty-one room names, each of which may have a corresponding value in "A".

If you are wondering why the program line numbers above are in irregular jumps, this is because there are lines omitted between the given lines which have functions other than the display of the room name. When the program at this early stage is run, the name of the first room (since "A" equals one) is printed at the top of the screen.

When you are asked to type in more lines of the program, then do not delete any existing lines of program, for the new lines will add to those

already there, and you will gradually see the program grow in both size and complexity. The complete listing of this program is given in Chapter Three for those people who may wish to type in the program all at the one time. If you do not come under this category, then it will allow you to make a reasonably quick check to see if you have missed out any of the line numbers in the sections, or even complete section which you will have to type in.

2) *A Display of the Exits from the Rooms* — Type in the following program lines:

```
 320    PRINT'CHR$131"Exits:- ";:RESTORE 1330:FOR C=1 TO
        A:READ D:NEXTC:IF D<>0 PRINT":North:";
 330    RESTORE 1340:FOR C=1 TO A:READ D:NEXTC:IF D<>0
        PRINT":South:";
 340    RESTORE 1350:FOR C=1 TO A:READ D:NEXTC:IF D<>0
        PRINT":East:";
 350    RESTORE 1360:FOR C=1 TO A:READ D:NEXTC:IF D<>0
        PRINT":West:";
1330    DATA 1,0,1,0,2,0,0,2,0,2,-3,0,0,0,-4,0,1,0,0,1,0
1340    DATA 0,-1,0,-1,0,0,-2,3,0,-2,4,-2,0,0,0,0,0,-1,0,0,-1
1350    DATA 0,0,-1,1,0,-2,1,0,-2,3,5,0,0,-4,0,0,-3,2,-2,0,0
1360    DATA 0,1,0,2,-1,0,2,-1,0,4,0,0,-3,3,0,-5,2,0,0,-2,0
```

The above routine is the first of several fairly complex routines which I use when writing an adventure.

Line 320 prints out "Exits:-" in yellow alphanumerics. The apostrophe after "PRINT" tells the computer to go onto the next line down on the screen. The semi-colon after the final inverted commas allows the possible exits to be printed directly after this on the same line. The data pointer is then restored to line 1336, which contains the data for any movement north. The FOR/NEXT loop counts along the line of data until it reaches "A", the room number, and stores the corresponding number, from the list of data, in the variable "D". If there is an exit north, then "D" will not equal zero, and so ":North:" will be printed out — the semi-colon after this is used so that if there are any other exits from the room, then they will be printed on the same line.

Lines 330 340, and 350 have a similar function as line 320 — they do not, however, print out "Exits:-" again — and instead of dealing with the direction north, the instead deal with the directions, south, east, and west.

Hence line 1340 contains the data for movement south, line 1350 holds the data for movement east, and line 1360 has the data for the direction west.

You may be wondering why the data statements hold various numbers other than zero, some of them being negative. This is so that the computer will know which room to go to, depending on whether it has been told to go north, south, east, or west from a particular room. However, at this stage of the program it is only necessary to know that if the value corresponding to a movement in one direction from a certain room is zero, then there is no exit from the room in that particular direction.

3) *A Display of the Objects in the Rooms* — Type in the following lines of program:

```
 360     PRINT' ' CHR$132"Objects:- ";
 370     H=0:RESTORE 1370
 380     FOR G=1 TO 22:READ C$:IF E(G)<>A OR H=4 NEXT G
         ELSE PRINT":";C$;":";:H=H+1:IF H<>2 NEXTG ELSE
         PRINT' CHR$132"       ";:NEXTG
1370     DATA GRENADE,ROUGH-METAL, SHINY-KEY, ICE-
         BLOCK, GLOVES, SABRE, AERIAL, TORCH,
         HEADPHONES, MAGNIFIER, LOCKED-DOOR, BELL,
         SCRATCHES, KEY-CUTTER, HOLE, TRANSMITTER,
         WINDOW, MUD-MAN, WIRE, INSCRIPTION, BOULDERS,
         SWARCK
```

However, it is necesary to enter the following lines as well at his stage so that the variables for the objects are initialised, thus preventing any errors from occurring:

```
  20     DIM E(22)
  30     A=1:T=0:W=0:RESTORE 1390:FOR B=1 TO 22:READ
         E(B):NEXTB
1390     DATA 4,8,22,16,5,12,22,3,9,17,20,1,2,6,15,10,7,18,11,19,13,22
```

I will deal with the initialisation of the variables for the objects first. The location of each object is governed by the values in the dimensioned variable of "E", which is set to be able to hold twenty-two values in line 20. "SWARCK" is a twenty-third objects, and it is in fact a magic word; it therefore cannot at any point be printed out in a room as an object, and so it does not require a location.

Line 30 is typed out again, but in addition to letting "A" equal one, there is a routine for setting the values in the dimension of "E". Firstly, the data pointer is restored to the line of data in line 1390. Next, there is a FOR/NEXT loop from one to the number of objects, which, in this case, is twenty-two. The computer then reads each of the values in the data

statement, and stores them in the corresponding values of "E". Hence E(1) will equal four, E(2) will equal eight, E(3) will equal twenty-two, and so on.

It should be obvious from the above information that most of the numbers in "E" are between one and twenty-one for them to correspond to the appropriate room numbers of the rooms that the objects are in. The reasons for an object having a value which is not between one and twenty-one are as follows: firstly, if the number equals twenty-two (one more than the number of rooms), then that particular object has not yet been brought into the game, and will be brought into it later on; similarly, if an object is to be removed from the game, its value in the dimension of "E" will be set to equal twenty-two. If the number in the dimension of "E" equals zero, then the corresponding object is being carried by the player, and if the number equals negative one, then something extra will be attributed to the object in addition to it being carried. For example, a "TORCH" will have a value of zero if it is being carried and if it is also lit, then its value will be negative one.

Now the printing of the objects in a room can be dealt with. Line 360 prints "Objects:-" in blue alphanumerics, the two apostrophes putting the cursor down two lines before printing, and the semi-colon allowing the names of the objects to be printed on the same line.

Line 370 sets the variable "H" equal to zero, and restores the data pointer to line l37 — "H" will contain the number of objects in the room, and line 1370 contains the data for the names of the objects in the adventure.

Line 380 works out if an object is in the same-room as the adventurer, and if so, this object is printed out on the screen. This routine is based around a FOR/NEXT loop from one to twenty-two, twenty-two, as previously stated, being the number of objects. Each of the twenty-two object names are read and stored in C$, and if the value in the dimension of "E" for that object is not equal to "A", the room number (the object is therefore not in the room), or if "H" equals four (there would already be four objects printed out, and so no more would be printed under this category), then the computer goes back for the next object — when the last object is reached the computer continues with the rest of the program. On the other hand, if the object is in the room and there is space to print it out, then the machine does so, incrementing the value in "H" before going back for the next object. If C$ equals "GRENADE" (assuming that it is also in the room and there is space to print it), then "PRINT" :";C$;" :"" will print out ":GRENADE:". The semi-colon allows the next object to be printed on the same line, but if there have been exactly two objects already printed out, then there will not be enough room for a third object to be printed on the same line, and as the cursor is moved onto the next line and spaces are inserted so that the third object is printed under the first object.

4) *A Display of the Constant Inventory* — The inventory is printed out in a similar manner as the objects. However, there are less lines to be typed in here since the dimension of "E" has already been assigned values, and the data for the object names need not be typed in a second time for the inventory —the lines that do have to be typed in are as follows:

```
390     PRINT' ' CHR$133"Inventory:- ";
400     F=0:RESTORE 1370
410     FOR G=1 TO 22:READ C$:IF E(G)<>0 AND E(G)<>-1 OR
        F=4 NEXTG ELSE PRINT":";C$;":";:F=F+1:IF F<>2 NEXTG
        ELSE PRINT' CHR$133"          ";:NEXTG
```

Line 390 prints "Inventory:-" in magenta alphanumerics after putting the cursor two lines down the screen — from now on I will no longer refer to the use of the apostrophe and the semi-colon, since you should by now realise their uses in the program.

Line 400 sets variable "F" equal to zero, and restores the data pointer to line 1370 — "F" will contain the number of objects being carried, and line 1370 contains the data for the object names.

Line 410, like line 386, is built around a FOR/NEXT loop which covers the numbers between one and twenty-two for each of the objects concerned. The difference between the two categories lies in that it is the objects that are being carried that have to be printed out, and not the objects in the room. Again, C$ contains the room name, and "G" contains the object number. This time, if the value in "E" is not equal to zero (the object is not being carried), and it is also not equal to negative one (the object is not one which is carried and has a special attribute), or if four objects are already printed out (if "F" equals four), then the computer goes back for the next object. The rest of this part of the program is virtually the same as for line 386 apart from that the variable "F" is concerned instead of the variable "G", magenta alphanumerics are printed instead of blue alphanumerics, and a different number of spaces are required to put the third object in its correct place.

5) *A Display of the Input Line* — This is the final aspect of the initial display. Although the computer asks for an input, the replies to whatever is inputted will be dealt with later on in this chapter. The relevant lines for the input are as follows:

```
420     VDU 31,0,13,134:PRINT"[- - - - - - - - - - - - - - - - - -
        - - - - - - - - - - - - - - - - - - - - ]"
```

```
430     VDU 31,0,17,134:PRINT"[- - - - - - - - - - - - - - - - - -
        - - - - - - - - - - - - - - - - - - ]":VDU 31,0,15,135
440     INPUT"Command? "B$
450     CLS:VDU 31,0,19,130
```

Line 420 moves the cursor to the first column and thirteen lines down from the top of the screen and prints CHR$l34, making the rest of the characters in the line become cyan in colour ("VDU" is more versatile than "PRINT", in its control of various things to do with the screen although it is very similar in function — "PRINT" is used in most cases when printing things on the screen since its function is easier to understand). A line of minus signs is then printed on the screen with side arrows at each end which point away from the centre of the screen.

Line 430 is the same as line 426, with the exception that the line of minus signs in this case is printed seventeen lines down from the top of the screen, instead of fourteen. The cursor is then moved to the fifteenth line, and the colour of the characters on that line are set to white.

Line 440 asks for the input of a command, and stores whatever is typed in, in the variable"B$". The two preceding lines of program cordon off (above and below) the line on the display which asks for the command.

Line 450 clears the screen and moves the cursor down to the nineteenth line, setting the character colour as green, ready for the printing out of the replies to whatever has been inputted.

WORKING OUT A ROUTINE SO THAT THE CHARACTER CAN MOVE FREELY BETWEEN ROOMS

The next stage in the adventure, now that the djsplay has been taken care of, is the movement between rooms. It is necessary for the character to be able to move in the directions implied by the "Exits" from the room, and if a direction is typed in which is not implied, then "No exit" will be printed on the screen. The following lines containing the routines for movement "North", "South", "East" and "West":

```
550     IF LEFT$(B$,1)<>"N" THEN 560 ELSE RESTORE
        1330:FOR C=1 TO A:READ D:NEXTC:IF D=0 THEN 600
        ELSE 590
560     IF LEFT$(B$,1)<>"S" THEN 570 ELSE RESTORE 1340:FOR
        C=1 TO A:READ D:NEXTC:IF D=0 THEN 600 ELSE 590
570     IF LEFT$(B$,1)<>"E" THEN 580 ELSE RESTORE 1350:FOR
        C=1 TO A:READ D:NEXTC:IF D=0 THEN 600 ELSE 590
580     IF LEFT$(B$,1)<>"W" THEN 610 ELSE RESTORE
        1360:FOR C=1 TO A:READ D:NEXTC:IF D=0 THEN 600
        ELSE 590
```

```
590       A=A+D:GOTO 160
600       PRINT"No exit!":GOTO 160
610       M=0:N=0:O=0
```

The data lines 1338 to l36{the ones regarding movement, and they have already been entered in the routine to print out the exits from a room. The main use for these lines of data will now be described.

Line 550 firstly checks to see if the inputted direction is north, and if not, it goes on to the next program line, which is line 560. However, if the direction entered is north, then the data pointer is restored to line 1330 and the value corresponding to movement north from the room "A", is read into the variable "D",in the FOR/NEXT loop. If the value in "D" equals zero, then there is no exit to the north, and so the computer jumps to line 60, where "No exit" is printed on the screen; there is then another jump which is back to line l6} to refresh the display for the room.

If there is an exit to the north then the value in "D" will not equal zero, and the computer will jump to line 596. At this line, the value in "D" is added to the value in "A", and the result stored in "A", so that the variable "A" equals the room number of the room which lies to the north of the room from which the direction was commanded. For instance, when one goes north from room "11" to room "8", the value in "D" will equal "-3" — this value of -3 is then added to "A", which equals "11", so that the final value of "A" will be "8", for "11 + (-3)=8". In short, "D" contains the number that must be added to "A" so that the new value of "A" corresponds to that of the room immediately north of that room corresponding to the previous value of "A". The computer then jumps to line 166 to refresh the screen display.

Lines 560, 570, and 580 perform the same function as line 550, except that they instead operate for south, east, and west respectively, and use the data in lines 1340, 1350, and 1360 the , which correspond to these movements.

Lines 590 and 600 have been explained with line 55}; line 610 is put in here so that a "no such line" error does not arise — if a direction is not typed in when the program is running at this stage, then the computer will jump to line 610 from line 580. The computer would become confused if it did not find this line, so it is just put in to keep the machine happy. In this line, the variables "M", "N", and "O" are defined as being equal to zero.

WORKING OUT ROUTINES FOR A RESPONSE TO THE INPUT AND CODING IN NUMERICAL VARIABLES, OF THIS INPUT

1) *Commands* — The first word of the two-word command to be entered is the actual verb, or command. The following routine works out the number of the inputted command from a line of data at line 1380, and stores the value in the variable "M" — there is also an error trap if the command entered is not recognised by the computer. The program lines are:

```
 470    IF LEFT$(B$,3)="WEA" OR LEFT$(B$,3)="EXA" OR
        LEFT$(B$,3)="SAY" THEN 610
 620    RESTORE 1380:FOR I=1 TO 16:READ C$:IF
        LEFT$(B$,3)=C$ M=I
 630    NEXT I:IF M<>0 THEN 650
 640    PRINT"I do not understand you.":GOTO 160
 650    PRINT M
1380    DATA GET,DRO,WEA,KIC,RIN,REA,CUT,EXA,KIL,LIG,
        OPE,THR,SAY,TAK,UNL,LOO
```

Line 470 checks for those commands which start with the letters "N", "S", "E", or "W", jumping to the "COMMAND" routine directly, and bypassing the movement routine if such a command is found; if it is a recognised command which starts with any of these four letters, then there is no need to bother about the movement routine. However, if it is not a registered command, then it is interpreted as a direction, for if the first letter of a command typed in is "N", "S", "E", or "W", it is read by the movement routine to be one of these four directions, and hence a line is required to check that no recognisable commands are mistaken for directions.

In line 620 the data pointer is restored to line 1380, where the command names are stored. Note that it is the first three letters of each command that are stored in this line, so that only the first three letters of each command need to be typed in when playing the game. After this, there is a FOR/NEXT loop which reads each name in the data for command names, comparing each name with the first three letters of the inputted command. If the two correspond, then "M" is given the value, as a number betwen one and sixteen, for the command — the range of possible values for the commands will vary for different adventures.

Line 630 contains the NEXT statement of the FOR/NEXT loop, and checks to see whether or not "M" equals zero, for if no command has been matched with that inputted, them "M" will still contain the value of zero, as assigned in line 610. If "M" does not equal zero, then the computer goes on to line 650 for the routine to match up the object entered, which will be dealt with after this — at the moment the value in "M" is printed out on the screen in line 650, showing that the command number has been correctly

matched, once the computer has gone through the routine; this line will be overwritten by the next routine.

If no command has been matched with that entered, then the computer will continue onto line 640 where "I do not understand you" is printed on the screen; there is then a jump to line 168 to refresh the screen format.

Line 130 contains the data for the commands, but to avoid confusion about what the commands are in full and what their functions are, then look up Appendix I. At present, the computer should be able to accept any command that it recognises, and convert it into a numerical form in the variable "M". If it does not understand the command, then it should tell you.

2) *Objects* — The second word in the two-word command is an object — this is what the verb, or command, acts upon. The object is also assigned a numerical value, depending on what it is; the value is then stored in the variable "O". There is, as usual, a method for trapping possible errors. Type in the following lines:

```
650     RESTORE 1370:D$=RIGHT$(B$,3):FOR J=1 TO 23:READ
        C$:C$=LEFT$(C$,3)
660     FOR K=4 TO 12:IF LEFT$(D$,1)<>" " AND
        C$=MID$(D$,2,3) N=1
670     IF C$=MID$(D$,2,3) O=J:K=12:J=23:GOTO 680 ELSE
        D$=RIGHT$(B$,K)
680     NEXT K:NEXT J:IF O<>0 THEN 690 ELSE
        PRINT"Pardon?":GOTO 160
690     IF N=1 PRINT"Learn to type."'  CHR$130;
700     PRINT M,O
```

In line 650, the data pointer is restored to line 137J, where the object names are stored. D$ is set equal to the first seven characters of B$, minus the first three characters. There then follows the first part of a FOR/NEXT loop which encloses a good part of the next few lines. Each object is in turn read from the list in line 1370, and the first three letters are stores in C$; this means that only the first three letters of each object need be typed in as well as for each command.

A routine is not required to find the object name in the entered string, for the command may have contained more than three letters, and so the computer has to look for where the object name starts; this is done with a FOR/NEXT loop in the lines 660 to 686. The computer initially checks to see if a space has been missed out by accident between the command and the object, and then checks to see if this object is valid. "D$" initially contains the four characters to the right of the second character in "B$", and

a check is made to see if the first of the four characters is not a space. If this is so, and the next three characters are the first three characters of an object name, then a space has been missed out, and "N" is set equal to one. If the object name has not been found, then the fourth character onwards is taken of "B$" and so on, until "K" equals ten, which is a sui ly high number so that the length in characters of a command will not mask what the object is. When "K" equals ten, the computer jumps back for the next object and continues with it.

If the object name matches up with that typed in, then "O" is set equal to the object number, depending on the value of "J" in the FOR/NEXT loop. If the computer comes out of this loop without having recognised your object, then "O" will equal zero, and "Pardon?" will be printed out before there is a jump back to line 168. If an object has been matched up, then "K" is set equal to twelve, and "J" is set equal to twenty-three, so that the computer will not have to continue through the loops unnecessarily.

If "N" equals one — a space was missed out between the command and the objet — then in line 690, "Learn to type" will be printed, although the computer will still recognise your command and object.

In line 700, the values in "M" and in "O" are printed after the routines, since both commands and objects have been gone through, and the inputted command and object have been recognised. This allows one at this stage to check that these words have been correctly recognised. This line will be overwritten by another routine later on.

THE FILLING FOR THE ADVENTURE

Now that values have been assigned to variables for the inputted command and object, it is necessary for these values to be enacted upon. In doing so, problems are formulated for the adventure player to solve by inputting the relevant words. An adventure should be filled with problems to which there are logical solutions — each room should have at least one function apart from being simply a passage, although in some cases no functions suitably coincide with the room name, and so it just remains a "link" room. This is probably the hardest part of writing adventures, since there are often many cross references which must be dealt with, and so it is necessary to gradually build up the actual adventure from the skeleton adventure, to which it has so far been built up to.

There now follows a number of program lines which contain the replies which are most frequently used when replying to commands which have been entered — each of these lines contains a PRINT statement with the relative information insid uotation marks, and this is followed by "GOTO 160" which makes the eomputer jump back to refresh the display:

```
720     PRINT"I cannot do that.":GOTO 160
730     PRINT"O.K.":GOTO 160
740     PRINT"I am carrying too much.":GOTO 160
750     PRINT"I do not see it here.":GOTO 160
760     PRINT"I am not carrying it.":GOTO 160
770     PRINT"I do not see a place to put it.":GOTO 160
780     PRINT"I do not have them.":GOTO 160
790     PRINT"I do not see them here.":GOTO 160
```

When the above lines are typed in no apparent difference can be seen since the routines which use these lines have not yet been dealt with. The first of these routines is the ability to pick up objects, or "GET" them. Type in the following lines:

```
700     ON M GOTO 800
800     IF O>10 THEN 720
810     IF F=4 THEN 740
830     IF E(O)<>A THEN 750
840     E(O)=0:GOTO 730
```

In line 700, if "M" equals one, then the computer will jump to line 800 — "GET" is the first command. However, if "M" has any other value, then the computer will break out of the program with an "ON range" error, since it has interpreted the entered command and has found nowhere to go with it; it then becomes confused and returns to the command mode. This means that while the program is running and all the commands have not yet been dealt with, the only commands that can safely be entered are those which have already been covered — the computer would be unable to provide a reply to such commands which cause these errors at this stage anyway. Line 700 will be updated every time another command is dealt with.

Line 800 checks to see whether an object is moveable or not. It is wise to have all moveable objects below a certain value, and those which are not moveable should be above that value to help separate them from each other; although this may work in theory, in practise it is hard to have such a fine line separating them, for at one point an unmoveable object may be dealt with, and at another point, one which can be moved may be dealt with, and so there is a tendency for an intermingling of the objects as the object vocabulary is dealt with. An example of an unmoveable object is a "DOOR", and an example of a moveable object is a "TORCH".

Line 810 sees if four objects are already being carried, for the variable "F" contains the number of objects which are being carried by the player. If the player is unable to carry any more, then the computer will make a jump to line 740 to print out the relevant information depending on what the situation is.

In line 830, if the value in the dimension of "E" for the variable "O" does not equal the room number — the object is not in the same room as the player — then a jump will be made to line 756 to print out a reply to this. An object may only be picked up if the conditions are met to enable it to be picked up.

The value in the dimension "E" for the variable "O" is set equal to zero line 840, for since the conditions have been met, the player is allowed to carry the object he/she wanted to pick up. There is then a jump to line 730 for the affirmative "O.K." to be printed on the screen.

The next command to be analysed is "DROP", which allows the player to drop an object, which has been picked up, in a room — not necessarily the room in which it was picked up. The lines that need to be typed in for this command are:

```
700     ON M GOTO 800,850
850     IF E(O)<>0 AND E(O)<>-1 THEN 760
860     IF H=4 THEN 770
870     E(O)=A:GOTO 730
```

If the command inputted is "DROP", then "M" will contain the value of two, and so the computer will go to line 856 from line 786, since 856 is the second number in the "ON M GOTO" statement.

Line 850 checks to make sure that the object is being carried. Remember that an object may be carried under normal circumstances, or else it may have a special attribute, like a lit "TORCH", for example; hence it could have its value in "E" equal to zero or negative one. An object must be carried before it may be dropped, and so if it is not being carried then the computer jumps to line 700 to say so.

If there are already four items in the room then "H" will equal four, and one would not be able to drop anything else in that room until something is picked up, and if this is the case, a jump is made in line 860 to line 770. "I do not see a place to put it" would then be printed before the screen format is updated.

In line 870 the value in "E" of "O" is set equal to the room number so that if the player moves to another room, the object dropped would remain in the room in which it was dropped. The computer ends up by going to line 730 for a response in the affirmative.

Now that the more general commands of picking things up and dropping them down again, have been dealt with, the more specialised commands can now be covered — I will deal with them in the order in which they appear

in the program. The first of these commands is "WEAR" which allows the adventurer to wear any objects that are suitable for wearing; in this adventure two objects which may be worn are the "GLOVES" and the "HEADPHONES". The following lines add this command to the commands "GET" and "DROP", and the commands of movement:

```
700     ON M GOTO 800,850,880
880     IF O<>5 AND O<>9 THEN 720
890     IF E(O)=-1 PRINT"I am already wearing them.":GOTO 160
900     IF E(O)<>0 THEN 780
910     E(O)=-1:GOTO 730
```

The line number for the third command is added to line 7 so that when "M" equals three — i.e. the command is "WEAR" — the computer jumps to line 880 to check the context at that stage of the adventure and decide what should appear on the screen as a result.

Line 880 selects which objects are allowed with the command "WEAR"; remember that "O" contains the object number. Only objects five and nine — the "GLOVES" and the "HEADPHONES" — will be accepted. If any other object is typed in then the computer will jump to line 720, rejecting this object.

Line 890 checks to see if the object is already being worn, for if the command had been typed in previously for the "GLOVES" or the "HEADPHONES" they would already have the special attribute of being worn, and the value corresponding to them in "E" would equal negative one. This line is inserted to provide a conformation that the object is being worn.

If the object is not being carried at all, then this is taken care of in line 900, for if this is the case the "E" of "O" would not equal zero, since the possible value of negative one for carrying an object has already been dealt with. It therefore follows from this that it is only possible to wear an object if it is firstly in one's possession.

Once the parameters have been met, the value in "E" of "O" can equal negative one giving the special attribute to the object. There is then a jump for a reply in the affirmative.

The fourth command on the list is "KICK" which forms the first real problem to be met by the adventurer, for in the first room, a prison cell, the only exit is through a door, and the problem lies in how to go through it. After fumbling about with various verbs, the adventurer may finally come across "KICK" — what else would you do to a door once you have become really frustrated with it? The following lines should enable you to try kicking the door yourself:

```
540     IF LEFT$(B$,1)="N" AND E(12)=A PRINT"The door is in the
        way.":GOTO 160
700     ON M GOTO 800,850,880,920
920     IF O<>12 THEN 720
930     IF E(O)<>A THEN 750
940     E(O)=22:PRINT" The hinges were weak and the door has "
        CHR$130"collapsed into a pile of dust.":GOTO 160
```

Line 540 is positioned before the movement routines so that no movement north can be made until the "DOOR" has been removed. The first character of the input is taken, and if it equals "N" for north along with "E" of twelve, the number corresponding to the object "DOOR", equalling the room number that the adventurer is in, then "The door is in the way" will be printed out. The only room that the door can be found in, is the first room, and so if it is there then "E(l2) = A", where "A" has to equal one. There is then a jump to refresh the display, and the provision for movement is not allowed.

The line number corresponding to "M" equalling four is line 920, and in this line, any other object than "DOOR" will be rejected, since no other object has the value twelve.

In line 930 the computer makes sure that the door has not already been removed, for if "E" of twelve does not equal one — from the above "O" must equal twelve, and "A" must be one — then the "DOOR" is nowhere in sight and need not be bothered with.

Once the conditions have been satisfied, the "DOOR" is put out of action by setting its "E" value to be twenty-two, the number one greater than the number of room in the adventure. An expIana,tion is then given for the disappearance of the door before the computer returns to the main program for the next command and object.

The next command I will deal with is "READ", for I intend to leave out "RING" until later as it concerns a possible death for the adventurer, and all the deaths will be dealt with later on. The purpose of the command "READ" is to read information which is available for reading by the adventurer. In the model adventure it is possible to read "SCRATCHES" and also an "INSCRIPTION". However, when reading the latter object it is necessary to have a "MAGNIFIER" in your possession, for otherwise "The writing is too small too read" will be printed out. Anyway, type in the following lines of programme:

```
700     ON M GOTO 800,850,880,920,720,980
980     IF O<>14 AND O<>21 THEN 720
```

```
 990     IF A=6 THEN 1020 ELSE IF A<>13 THEN 790
1000     IF E(10)<>0 PRINT"The writing is too small to read.":GOTO
         160
1010     PRINT"The magic word is '  swarck'  .":GOTO 160
1020     PRINT" A transmitted signal will allow a door"CHR$130"from
         the '  air lock'    to be opened.":GOTO 160
```

Since the sixth command is dealt with before the fifth, the line number corresponding to "M" equalling five is line 720 — at this line, "I cannot do that" is printed out — and this will be changed when "RING" is finally brought into the vocabulary. If "M" equals six then the computer will jump to line 980

In line 980 all other objects apart from "SCRATCHES" and "INSCRIPTION" will be rejected since no other object can be read. "SCRATCHES" will have its value in "O" as fourteen, and "INSCRIPTION" will have this value equal to twenty-one.

The next line, line 990, checks to make sure that the adventurer is in a position to read the objects and therefore be in the same room as the objects. If "A" equals six then the computer jumps to line l{2 fl to deal with what the "SCRATCHES" say, and any other value than thirteen, which is the room number of the "INSCRIPTION" will be discarded.

If the "MAGNIFIER" is not in the possession of the adventurer, then "E" of ten will not equal zero, and hence the "INSCRIPTION" will not be read. However, if' this is carried, then the "magic word" will be divulged — what has to be worked out when playing the game is what this word should be used for, but that will be covered later.

The seventh command is "CUT", and in this adventure, this allows a piece of "ROUGH-METAL" to be "CUT" into a "SHINY-KEY" provided that a "KEY-CUTTER" is in the room. However, in other adventures, "CUT" may have different uses, but this is the first time that I have used, or seen use of, this particular use of the command "CUT". When writing an adventure you may find considerable pleasure in thinking up plausible and original uses of commands. I only have this one use of this command in this adventure, and this is detailed below:

```
 700     ON M GOTO 800,850,880,920,720,980,1030
1030     IF O<>2 THEN 720
1040     IF E(O)<>0 THEN 740
1050     IF A<>15 PRINT"I see no place where it can be cut.":GOTO
         160
1060     E(2)=22:E(3)=0:PRINT" The piece of metal has been cut into
         a"CHR$130"key.":GOTO 160
```

Line 700 is again repeated, but this time it has an extra jump, on the condition that "M" equals seven, to line 1030. In line 1030, if the object is not the "ROUGH-METAL" — "O" would not equal two — then the action would be rejected.

The computer then checks to see if the "ROUGH-METAL" is being carried, for if it is not being carried then it cannot be cut. If "E" of "O" does not equal zero then this object is not being carried.

Next, the machine makes sure that the player is in the same room as the "KEY-CUTTER", for if there is nowhere in sight where a key can be cut, then there is no way that it can be done. Therefore, if this is the case, then the action is dismissed.

In line 1060, once the conditions have been met, the "ROUGH-METAL" is removed from the game and the "SHINY-KEY" is brought into the possession of the player — "E" of two is set equal to twenty-two, and "E" of three is set equal to zero. The appropriate result is then printed on the screen before the computer returns for the next command.

The next and eighth command is "EXAMINE" which allows the adventurer to see something in greater detail. In this adventure, if a "WINDOW" is examined, then the reply that a spacecraft can be seen outside and is ready to take off, will be printed out. This is quite a versatile command which may reveal objects that were otherwise not visible to the player, but this use of it is not detailed in this adventure. Type in the following lines:

```
 700    ON M GOTO 800,850,880,920,720,980,1030,1070
1070    IF O<>16 AND O<>18 THEN 720
1080    IF A=10 THEN 1100 ELSE IF A<>18 THEN 750
1090    PRINT" A space ship can be seen outside. It is"CHR$130"ready
        to take off.":GOTO 160
```

Line 700 adds the line number 1070 to the list of GOTO lines, and this corresponds to "M" equalling eight. If any other object than the "WINDOW" is inputted, then it will be rejected in line 1070. If the window is not in sight — the adventurer is not in the appropriate room — then the action would be rejected this time in line 1080. Finally, in line 1090, the result to the command would be printed out about the space ship.

Command number nine is "KILL" which may be useful if any nasty creatures are in your way. Although you may not always be allowed to kill things, a "MUD-MAN" in this adventure may be killed, but to do so it is imperative that the "SABRE" is carried, for otherwise it will kill you, but

that will be dealt with later on. The necessary lines are:

```
 700     ON M GOTO 800,850,880,920,720,980,1030,1070,1110
1110     IF O<>19 THEN 720
1120     IF E(O)<>A THEN 750
1130     PRINT"You have killed the mud-man.":E(O)=22:GOTO 160
```

Line 700 adds the ninth possible line number to the list, thus catering for the possibility of the value in "M" being nine. Line 1110 makes sure that no other object than the "MUD-MAN" is allowed, and it also certifies that the "SABRE" is being carried. There is a check to make sure that the adventurer is in the same room as the "MUD-MAN" in line ll20, and in line 1130 the "MUD-MAN" is removed from the scene by letting "E" of "O" equal twenty-two. The result is then printed on the screen before a jump to refresh the display.

The next command is "LIGHT" which allows the adventurer to light, for example, a "TORCH" or a "LAMP" — a "TORCH" is used in this adventure — and so enable the player to move about without stumbling into anything in the dark. In the "Dimly lit passage" adventurers must carry a lit "TORCH" or else meet their fate; type in the lines below to allow this implement to be lit:

```
 700     ON M GOTO 800,850,880,920,950,980,1030,1070,1110,1140
1140     IF O<>8 THEN 720
1150     IF E(O)=-1 PRINT"It is already lit.":GOTO 160
1160     IF E(O)<>0 THEN 760
1170     E(O)=-1:PRINT"It is now lit.":GOTO 160
```

Line 700 adds line 1140 as the tenth line number for the computer to jump to, since "LIGHT" is the tenth command. The machine makes sure that no other object than the "TORCH" is accepted in line 1140, and in 1l50, a check is made to see whether or not the "TORCH" has already been lit or not, and if it has not, then the computer will go on to the next line. If the "TORCH" is not being carried then it cannot be lit, and so this would be rejected in line 1160. In line 1170 "E" of "O" is set equal to negative one to show that it has the special attribute of being lit. The appropriate reply is made before returning for the next command.

The eleventh command is "OPEN", which allows a "LOCKED-DOOR", or a "BOX", or some similar object to be opened, but here, the only object that may be opened is the "LOCKED-DOOR". Since there is already a "DOOR", the prefix "LOCKED-" is put in front of the word "DOOR" to form another object by the name of "LOCKED-DOOR". The program lines are:

```
700     ON M GOTO 800,850,880,920,720,980,1030,1070,
        1110,1140,1180
1180    IF O<>11 THEN 720
1190    IF E(O)<>A THEN 750
1200    IF E(3)<>0 PRINT"I have no key.":GOTO 160
1210    E(O)=22:E(22)=20:PRINT"The door came away in your
        hands,but"' CHR$130"the exit is now blocked by
        boulders"' CHR$130"which had been behind the door.":GOTO
        160
```

The line number corresponding to "M" equalling eleven is added to the list of line numbers in line 76. This line number, line 1180, makes sure that no other object than the "LOCKED-DOOR" is allowed. Line 1190 checks to see that the player is in the same room as the "LOCKED-DOOR" and in a position to open it, provided that he/she is carrying the "SHINYKEY" (this is checked for in line 1200). Now that the conditions have been met, the "LOCKED-DOOR" is removed from the scheme, but the "BOULDERS" are brought into the action.

The next commands is "THROW" which allows an object to be thrown, the object in this adventure being a "GRENADE". However, in other adventures, a popular use of this command is the "THROW" a "ROPE" — this "ROPE" would then attach itself to a sui le point high up and become climbeable; another way of manipulating a "ROPE" in this fashion would be to "TIE" it to a sui le point, but instead of being able to climb up, one would be able to climb down. Now return to the typing:

```
700     ON M GOTO 800,850,880,920,720,980,1030,1070,
        1110,1140,1180,1220
1220    IF O<>1 OR E(22)<>A THEN 720
1230    IF E(O)<>0 THEN 760
1250    E(1)=22:E(22)=22:PRINT" You have cleared a passage through
        the"CHR$130"boulders.":GOTO 160
```

Line 700 addes the line number for the twelfth command, "THROW", to the list of line numbers. In this line to which the computer may jump, a check is made to see that it is the "GRENADE" that is being thrown, and that the player is beside the "BOULDERS" and therefore in a position to clear a passage through them. The next line, line 1236, makes sure that the "GRENADE" is being carried, for if it is not, then it cannot be thrown. Line 125J removes the "GRENADE" and the "BOULDERS" from he scene before printing out the ap ropriate result.

"SAY" is a command which allows a player to say a magic word which could perform an action which would otherwise be impossible for the player to make. One use of saying such a word could be to reveal a hidden exit from a room, but the use for it in this adventure is to move the player from the outside of the space ship into it and make it take off, provided that the player is outside the spacecraft to begin with. The following lines allow this command to be entered while the program is running:

```
 700     ON M GOTO 800,850,880,920,720,980,1030,1070,
         1110,1140,1180,1220,1260
1270     IF A<>21 PRINT"Nothing happens.":GOTO 160
1280     PRINT" You have materialised inside your
         ship"CHR$130"which has immediately taken off."
1290     END
```

Line 700 adds, as usual, another line number to the list — the line that it adds is line 126} for the thirteenth command. This line, which the computer may jump to, checks to see if the right object, which is the magic word "SW ARCK", has been entered. The only other condition that has to be met is that the player is in the right room, and this is dealt with in the next line. The result is printed out and the program ends in line 1290; this will be changed later on as there will be a provision for a score for playing the adventure.

The last three commands on the list are "TAKE", "UNLOCK", and "LOOK", which act as alternatives to the commands "GET", "OPEN", and "EXAMINE" — it is useful in an adventure to have several commands which perform the same task since the player would then have a choice of commands, and would be able to choose the commands which he/she prefers to use. For example, one may prefer using the command "GET" to the command "TAKE". The only line that needs to be typed in is an update of line 700:

```
 700     ON M GOTO 800,850,880,920,720,980,1030,1070,
         1110,1140,1180,1220,1260,800,1180,1070
```

Note that the last three line numbers correspond with those for the commands "GET", "OPEN", and "EXAMINE", since "TAKE", "UNLOCK", and "LOOK" are alternatives to these commands.

The next feature of the adventure is the provision of various deaths which may be encountered by the adventurer. The first death that I will deal

with is the one which concerns the command "RING" which has been missed out until now. The player is tempted to "RING" a "BELL" which is in one of the rooms — what else would one do with a bell? This is, however, a red herring, and the noise from the "BELL" aggravates the inhabitants of the maze, and they kill the player who then has to start again. The following lines detail the routine:

```
700     ON M GOTO 800,850,880,920,950,980,1030,1070,
        1110,1140,1180,1220,1260,800,1180,1070
710     VDU 23;11,0;0;0;0,31,6,23:PRINT"Press space to start
        again":IF INKEY$(50)=" " VDU 23;11,255;0;0;0:GOTO 30
        ELSE VDU 31,6,23:PRINT"                    ":IF
        INKEY$(50)=" " VDU 23;11,255;0;0;0:GOTO 30 ELSE 710
950     IF O<>13 THEN 720
960     IF A<>2 THEN 750
970     PRINT"You have woken the dead who do not
        like"CHR$130"you too much.":GOTO 710
```

Line 700 is repeated for the last time with the line number corresponding to "M" equalling five and the command being "RING". In line 950, where - the machine would jump to, any other object than the "BELL" would be rejected. In the next line the computer then checks to see if the player is in the same room as the "BELL". Line 970 the prints out the reason for the player's death, and then jumps to line 710 which is the main part of the death routine.

The first part of line 710 with the assortment of numbers after the VDU statement removes the flashing cursor from the screen, and moves this now invisible cursor to the position on the screen twenty-three lines down and six lines along. "Press space to start again" is then printed out at this position — the computer waits for the space bar to be pressed, and if it is pressed within the time limit then the flashing cursor will be restored, and the machine will jump back to line 38 to start the game again. If the time limit ends, however, and the space bar has not been pressed, then the message is blanked over, and another time limit is set. The net effect of this is to have the message flashing On and off on the screen until the appropriate condition has been met, and then there will be a jump to the aforementioned line number.

The next few line numbers concern other deaths which are concerned with What commands and objects are typed in. Deaths concerning other situations will be dealt with after all the routines pertaining to the commands have been attended to. The line numbers for the remainig deaths directly related to the comnands are as follows:

```
1070    IF O<>16 AND O<>18 THEN 720
1080    IF A=10 THEN 1100 ELSE IF A<>18 THEN 750
1100    PRINT" Something large has fallen through the"CHR$130"hole
        and flattened you.":GOTO 710
1240    IF E(9)<>-1 PRINT" The noise from the explosion has
        burst"CHR$130"your ear drums.The shock of this
        has'"  CHR$130"killed you.":GOTO 710
```

The first of these deaths occurs if one examines a "HOLE" in the ceiling of one of the rooms, for something large would then fall out of the hole and flatten the player. Line 1070 makes sure that no other object than the "HOLE" or the "WINDOW" is examined, and if it is the "HOLE" that is examined, then the appropriate message will be printed out before there is a jump to line 710 for the player to start again.

The second and last of these deaths is when the "GRENADE" is thrown at the "BOULDERS", for if the "HEADPHONES" are not worn, then the shock of the noise from the explosion will result in death; if "E" of nine does not equal negative one, then there will be no special attribute of the "HEADPHONES" being worn. The result is printed out and the player will have to start again.

Another routine concerning the commands, but not with the death of the adventurer, is when he/she wishes to pick up a block of ice. The condition for doing this is, however, the wearing of a pair of gloves. The following line should be included in the "GET" statement;

```
820     IF O=4 AND E(5)<>-1 PRINT"It is too cold to carry.":GOTO
        160
```

This line checks, first of all, to see if the object entered is the "ICEBLOCK", and then makes sure that the "GLOVES" are being worn. If the conditions are not met then this object cannot be picked up in line 840.

When the player finishes the adventure then it is a good idea for a score to be given along with the best score obtained, and so the next few lines deal with this:

```
  10    X=0
  30    A=1:W=0:RESTORE 1390:FOR B=1 TO 22:READ
        E(B):NEXTB
 190    W=W+1
1290    Y=120-W:IF Y>X X=Y
1300    VDU 31,0,16,131:PRINT"Score=";Y;"    Best
        Score=";X:GOTO 710
```

Line 10 sets the variable "X" equal to zero, for this will contain the next best score for the game, and must be at the lowest score possible at the outset of the game, and therefore at zero. In line 30, the variable "W" is set equal to zero. This variable contains the number of moves made in the adventure, and this is incremented by one every time the computer passes line 190.

The score for a particular game is contained in the variable "Y", and this is evaluated in line 1290 as the number "120" minus the number of moves made. If the present score is better than the best score, then it become the new best score in this same line. The next line prints out the relative scores before jumping to line 710 to restart the game for the player to try and better his/her score.

Now that all the routines pertaining to the commands in the main routine have been dealt with, I will now deal with another command which does not require an object. This routine is placed before the movement routines, and corresponds to the command "TRANSMIT" which allows a signal to be transmitted. This signal will open up an entrance to an "air lock", but several parameters are required before this may be done, although these will be discussed below these lines which are relevant to this routine:

```
 30    A=1:T=0:W=0:RESTORE 1390:FOR B=1 TO 22:READ
       E(B):NEXTB
250    IF A=7 AND E(7)=7 PRINTCHR$130"The transmitter is fully
       operational."
260    IF E(4)=7
       E(4)=22:T=1:E(7)=19:E(20)=22:PRINTCHR$130"The
       transmitter has cooled down,"'  CHR$130"but it does not have an
       aerial."
280     IF A=7 AND T=0 PRINTCHR$130"The transmitter is
       overheating."
510    IF LEFT$(B$,3)<>"TRA" THEN 540
520    IF A<>7 OR E(7)<>7 THEN 720
530    T=2:PRINT" An entrance has appeared into the
       '  air"CHR$130"lock'  .":GOTO 160
```

To begin with, the variable "T" is set equal to zero in line 30. This variable controls the state of operation of the transmitter. When this contains its Original value of zero, then the transmitter is in a state of overheating, and so Whenever one is in the appropriate room, then the message, as detailed in line 280, will be printed out.

However, wh the "ICE-BLOCK" is dropped in this room, then "E" of four will equal seven, and the transmitter will be cooled down, the ice will be removed from the scheme since it will have taken in the heat from the

transmitter and melted The piece of "WIRE" in room number nineteen is replaced by the "AERIAL", and the variable "T" is set equal to one — the transmitter is now working apart from it requiring the aerial as stated in line 260 of the adventure.

For the transmitter to be in a state of full operation, the "AERIAL" must be in the correct room, as checked for in line 250. The only thing now for the player to do is to transmit the signal. The entered command is checked in line 510, and if it does not correspond, then the computer goes onto the next routine. The machine then makes sure that the player is in the right room to transmit, and that the aerial is in that room. Now that all the conditions have been met, the variable "T" is given the value of two to show that the entrance has been revealed. Once this action has been made, there is then a jump back for the next command.

It is necessary for the players to be allowed to "QUIT" if they consider their relative positions to be desperate. This is quite an easy routine, for if the user simply types out the first three letters of the command "QUIT", then there is a jump to line 71fl to restart the game. This is another command that does not require an object, and only the one line of program given below need be typed in:

```
460     IF LEFT$(B$,3)="QUI" THEN 710
```

There often arise several instances in adventures when the player is not allowed to go in certain directions for particular reasons. The two instances in this adventure are contained in the next two lines:

```
490     IF (LEFT$(B$,1)="N" OR LEFT$(B$,1)="S" OR
        LEFT$(B$,1)="E") AND E(19)=A PRINT"You cannot pass the
        mud-man.":GOTO 160
500     IF T<>2 AND A=18 AND LEFT$(B$,1)="E" PRINT"You
        cannot pass into the '  air lock'  .":GOTO 160
```

If the mud-man is in the same room as the adventurer then no movement is allowed until this mud-man has been removed from the scene. The only way of removing the mud-man is by killing it with the "SABRE". If the signal has not been transmitted from the "Signal transmitter room" then "T" will not equal two. If this is the case and the player tries to go east from room eighteen then this will not be allowed until the signal has been transmitted.

There are two deaths which concern the room in which the adventurer is in and what is being carried. The first requires the "SABRE" being carried in the presence of the mud-man, or else death will result. The other has the need for carrying the lit "TORCH" in the "Dimly lit passage" or else the

player will fall down a hole in the poor light. Type in the following two fines:

```
270     IF A=11 AND E(6)<>0 AND E(19)=A PRINT"A mud-man has
        just killed you.":GOTO 710
290     IF A=14 AND E(8)<>-1 PRINTCHR$130" You have fallen into
        a hole in the"' CHR$130"dim light.":GOTO 710
```

Line 270 must have the player not carrying the "SABRE" in the appropriate room, and the mud-man must be in that room before death will result. Line 290 requires "A" equalling fourteen, and "E" of eight not equalling negative one — the "TORCH" would not have the special attribute of being lit —before the adventurer is killed.

The one form of death left to be dealt with is the running out of time. In this adventure the player is allowed 120 moves in which to complete it before the planet, on which the player is situated, blows up. Various warnings are given depending on how many moves have been made. Once 12 moves have been made, the adventurer has to start again. The lines to be typed in are as follows:

```
190     W=W+1:IF W>20 AND W<40 PRINTCHR$130"A rumbling
        sound can be heard."
200     IF W>39 AND W<60 PRINTCHR$130"The noise is becoming
        louder."
210     IF W>59 AND W<80 PRINTCHR$130"The ground is starting
        to shake."
220     IF W>79 AND W<100 PRINTCHR$130"I' d advise you to get
        out quickly."
230     IF W>99 PRINTCHR$130"The roof is caving in."
240     IF W=120 PRINTCHR$130"The planet has blown up.":GOTO
        710
```

After the incrementing of the variable "W", the computer checks for values between twenty and forty, printing out the message if the value in "W" lies within these limits. If not, then the computer may find a corresponding value and message in the next few lines, but if "W" has the value of l2fl, then there is no hope for the player and death results.

TIDYING UP THE PROGRAM

1) *Full Instructions* — When writing programs, full instructions are necessary in most cases, since anyone loading a certain program would be able to work out the basis of the program without too much confusion. The adventure is no exception, although most of the instructions are taken up with a storyline of the situation in which the adventurer is in before starting the game. The following lines contain the instructions for this adventure:

```
 40    CLS:PRINT' ' ' CHR$129"Do you want the instructions(Y or N)
       ?";:Z$=GET$:IF Z$="N" THEN 150 ELSE IF Z$="Y" THEN
       50 ELSE 40
 50    CLS:PRINT' ' ' CHR$130" You have been captured by creatures
       on"CHR$130"an uncharted planet."
 60    TIME=0:REPEAT UNTIL TIME>400
 70    PRINT' ' CHR$131" Unfortunately the planet happens to
       be"CHR$131"unstable,and has been evacuated."
 80    TIME=0:REPEAT UNTIL TIME>400
 90    PRINT' ' CHR$132"You therefore have to escape before
       the"CHR$132"planet blows up with you on it."
       100 TIME=0:REPEAT UNTIL TIME>400
110    PRINT' ' CHR$133" The computer has a fairly large
       number"CHR$133"of commands,so therefore if one
       command"CHR$133"does not work then try another."
120    TIME=0:REPEAT UNTIL TIME>500
130    PRINT' ' CHR$134"The first three letters of each
       command"CHR$134"and object need be typed
       in,although,if"CHR$134"desired,the full word may be entered."
140    TIME=0:REPEAT UNTIL TIME>500
```

Line 40 gives the player the option of receiving the instructions or not: a letter is inputted and stored in the variable "Z$" — if this letter is "Y" then the computer goes on to print the instructions, and if it is "N" then the instructions are not printed out. However, if neither of these letters is inputted, then the question is asked again, and the process gone through until an appropriate reply is obtained.

Lines 50, 70 and 90 describe the situation that the player is in at the beginning of the adventure — this just provides a story for the player to continue. Lines 110 and 130 provide a short description on how to operate the game, although those people who have played adventures before will have the basic idea on how to play, for the same fundamental principles are transferred across most true adventures.

Lines 60, 80, 100, 120 and 140 are delay loops which give the player sufficient time in which to read each message before the next message is displayed. The size of the number at the end of each line will vary the amount of time before the computer prints out the next message. An alternative to the REPEAT/UNTIL loop could be the FOR/NEXT loop in the form; FOR Z = 1 TO TIME:NEXT Z, where "TIME" is a number corresponding to the duration of time required for the player to read the message.

2) *Lower Case Graphics* — As you may have noticed, lower case text is freely mixed with upper case text. A lot of micros are unable to support this, and so if the one you possess does not, then just type in upper case — lower case is just more pleasing for the eye to read; it is not a terribly great disadvantage in not having this.

3) Colour — For those people with colour micros other than the BBC Micro, you may be interested in what the colours are corresponding to the CHR$ codes that I have used, and so they are as follows:

CHR$ 129 —red alphanumerics
CHR$ 130 —green alphanumerics
CHR$ 131 —yellow alphanumerics
CHR$ 132 —blue alphanumerics
CHR$ 133 —magenta alphanumerics
CHR$ 134 —cyan alphanumerics
CHR$ 135 —white alphanumerics

If your machine does not have colour, then just miss out these CHR$ codes from the PRINT statements.

4) *Sound* — Sound is another that may be included in an adventure, but also only if your micro has this feature. Some people may find the sound aggravating, and so it may be missed out if desired. The purpose of the sound is to produce a changing tone which increases in volume and pitch as the player uses up more and more moves. The program lines for the BBC Micro are given below, and so using the information below these lines, along with the syntax for emitting sound on your own micro, the tone should be easily adapted for it:

```
 30    A=1:T=0:W=0:RESTORE 1390:FOR B=1 TO 22:READ
       E(B):NEXTB:SOUND 1,0,1,1
170    ENVELOPE1,1,-1,1,-1,0,15,30,0,0,0,0,W,0
180    SOUND1,1,W*2,1
```

Line 30 has a "SOUND" command which has the volume set to zero, so that any lingering sounds from a previous game are cancelled for the start of a fresh game. Remember that the variable "W" contains the number of moves already made: line 170 defines the wavering tone that is to be produced, and "W" controls the volume of this tone. The next line, line 180, is the line that actually produces the sound, and "W", in this case, controls the frequency. Th• means that with every increment of "W", the pitch and volume of the sound increase. For those that understand the attack decay, sustain, and release of notes, there is a release value of zero in the envelope so that the tone is cotinuous.

VARIABLES

To ease the understanding of the program, a list of all the variables and their relative usage are given below:

1) *Numerical Variables*

(i) A —Number of the room in which the adventurer is situated.

(ii) B — Variable of a FOR/NEXT loop which is in one instance used in the reading of values for the positions of objects into the dimension of "E", and in another instance for the reading of the room name of the room in which the player is in.

(iii) C — Variable of a FOR/NEXT loop for reading if certain directions are to be printed on the screen and for seeing if the player can go in particular directions.

(iv) D — The variable that decides if there is a certain exit from a room, and this depends on whether the value in it is, or is not, equal to zero; it will contain the value which must be added to "A" to go in the chosen direction.

(v) F —The number of objects that are being carried by the player.

(vi) G — FOR/NEXT loop variable which counts for each object to see if it should be printed on the screen display format under either "Objects" or "Inventory".

(vii) H —The number of objects that are lying in a room.

(viii) I — FOR/NEXT loop variable used to help determine the number of the command entered by the player.

(ix) J — FOR/NEXT loop variable which aids the determination of the number of the object entered after the command.

(x) K — FOR/NEXT loop variable which allows for the input of the full number of letters in any command chosen by the adventurer.

(xi) M —Variable in which the command number is stored.

(xii) N — Depending on whether this variable contains the value zero or one, a space will either have been entered or missed out between the command and the object by the player.

(xiii) O —Variable in which the object number is stored.

(xiv) T —The value in this variable determines the status of the transmitter.

(xv) W — The number of moves made by the adventurer are stored in this variable.

(xvi) X —This contains the value of the best score.

(xvii) Y —The current score is in this variable.

(xviii) TIME —The variable for the computer's internal clock.

2) *Dimensioned Variables* — The objects corresponding to the following possible dimensions of "E" are given alongside each variable below:

E(1) —GRENADE
E(2) —ROUGH-METAL
E(3) —SHINY-KEY
E(4) —ICE-BLOCK
E(5) —GLOVES
E(6) —SABRE
E(7) —AERIAL
E(8) —TORCH
E(9) —HEADPHONES
E(10) —MAGNIFIER
E(11) —LOCKED-DOOR
E(12) —DOOR
E(13) —BELL
E(14) —SCRATCHES
E(15) —KEY-CUTTER
E(16) —HOLE
E(17) —TRANSMITTER
E(18) —WINDOW
E(19) —MUD-MAN
E(20) —WIRE
E(21) —INSCRIPTION
E(22) —BOULDERS

Remember that there is a full list of the commands and the objects in the first two Appendices at the end of the book along with detailed descriptions according to their relative functions.

3) *String Variables*

(i) A$ — The variable in which the room name is stored in. This name is displayed on the screen and updated every time the adventurer moves to another room.

(ii) B$ — A string which is inputted by the player and contains, or should contain, a command and an object, unless, however, a command which does not require an object is inputted.

(iii) C$ — The variable into which is read the names of the objects to be printed under "Objects" and "Inventory". This is also used in the determination of the numerical values for the commands and objects entered, for the first three letters of each command is stored in "C$", and then conpared with the first three letters of the inputted command —the same applies for the objects.

(iv) D$ — This is used to help search for the name of the object entered in

"B$", for the three letters starting from one character are compared with those in "C$" ("D$" is what the characters from "B$" are stored in). Another three letters are then taken starting from the next character, and so on, until either the object name has been found, or the computer runs out of all possible characters to compare. The first three letters of the next command would then be stored in "C$".

(v) Z$ — This is the input for whether or not the player wishes to receive the instructions at the start of the game, with the acceple replies being "Y" and "N".

THE PURPOSES CORRESPONDING TO EACH LINE OF DATA

1) *Lines 1310-1320* — In these lines, the data for the names of all the rooms in the adventure are stored.

2) *Line 1330* — the data for movement "NORTH".

3) *Line 1340* — the data for movement "SOUTH".

4) *Line 1350* — the data for movement "EAST".

5) *Line 1360* — the data for movement "WEST".

6) *Line 1370* — the names of all the objects used in the adventure are contained in this line.

7) *Line 1380* — this line contains the first three letters of the names of each command.

8) *Line 1390* — the data for the relative positions of each object at the beginning of the game.