subroutines that are not acted upon until control is returned to the routine that called the subroutine. A good example of the use of this type of program structure is the validity test described earlier. This subroutine is called by both the TAKE and DROP routines. In each case, the subroutine makes a decision as to the validity of the object part of the command phrase. However, the flow of the program is not altered until a RETURN is effected to either the TAKE or DROP routines. Only after returning is the value of the flag, F, set by the validity test subroutine and the appropriate branch made. One criticism of this technique is that we are effectively testing the same condition twice — once to set the flag value, and again to test the value of the flag. Although this is true, the added flexibility and ease of debugging achieved by employing this technique usually outweighs the slightly longer execution time that results.

SPECIAL LOCATIONS

We are now at the point in our project where we have completed the programming of the game's skeleton; that is, the programming that allows the player to carry objects and move around in the adventure world. We can now move on to the next phase of design in which we consider the 'special' locations where objects are put to use, perils are met and where the player's ingenuity and skill are tested.

Before we look in detail at the programming of the routines for one of the special locations in the Haunted Forest, let's consider the additions to be made to the main program loop in order to detect special locations. These two lines must be inserted into the listing:

```
257 GOSUB2700:REM IS P SPECIAL ?
258 IF SF=1 THEN 300:REM NEXT INSTRUCTION
```

Line 257 calls a subroutine to see if the current location is special. If this is the case then a 'special flag', SF, is set to one. This means that when control is eventually returned to the main program loop, the part of the main loop dealing with instructions can be avoided. The subroutine that decides whether the current location is special or not is:

```
2700 REM **** IS P SPECIAL S/R ****
2705 SF=0:REM UNSET SPECIAL FLAG

2715 REM ** OTHER SPECIAL LOCATIONS **
2720 ON P GOSUB4590,4690,4790,4590
2730 RETURN
```

You will recall that, when we designed the original map for the Haunted Forest, we numbered the four special locations first (see page 766). We can, therefore, simplify the selection of the appropriate subroutine for each special location by making use of the ON. . . GOSUB command. As can be seen by the way it is used in line 2720, this command is followed by a series of line numbers, and the appropriate line number is selected according to the value of P. If P is one, for example, the command will GOSUB to the first line number from the list; if P is two, then the second line number will be used for the GOSUB call, and so on.

There are four line numbers, one for each of the special locations in Haunted Forest. If P exceeds four, then control simply passes to the following line. If each of the four subroutines that can be called from line 2720 sets an SF flag, then the fact that P was a special location can be flagged. If no routine is called, the SF flag will remain set at zero, indicating that P is just an ordinary location. The ON. . . GOSUB command is clearly an economical alternative to a series of IF. . . THEN statements testing the value of a variable and branching to different subroutines accordingly.

THE TUNNEL ENTRANCE

Two of the special locations in the Haunted Forest are the two entrances to a tunnel (locations 1 and 4). To deal with the simple scenario of the player wishing to enter the tunnel, we need to construct carefully a routine that handles the normal commands and allows the player to enter the tunnel or retreat back down the path.

```
4590 REM **** TUNNEL ENTRANCE S/R ****
4600 SF=1
4605 SN$="YOU HAVE ARRIVED AT THE MOUTH OF A LARGE
TUNNEL":GOSUB5500
4610 SN$="YOU CAN ENTER THE TUNNEL OR RETREAT
ALONG THE PATH":GOSUB5500
4620 :
4625 PRINT:INPUT"INSTRUCTIONS";IS$
4630 GOSUB2500:REM SPLIT INSTRUCTION
4635 IF F=0 THEN 4625:REM INVALID INSTRUCTION
4637 GOSUB3000:REM NORMAL INSTRUCTIONS
4640 IF MF=1 THEN RETURN:REM PLAYER RETREATS
4645 IF VF=1 THEN 4625:REM INSTRUCTION OBEYED
4650 REM ** NEW INSTRUCTIONS **
4655 IF VB$="ENTER" THEN GOSUB 4700:RETURN
4660 IF VB$="RETREAT" AND P=4 THEN MF=1:P=6:RETURN
4665 IF VB$="RETREAT" AND P=1 THEN MF=1:P=3:RETURN
4667 SN$="I DON'T UNDERSTAND":GOSUB5500:GOTO 4625
```

The routine starts by setting SF to one to indicate that a special location has been reached. After displaying a message on the screen, describing the tunnel entrance and the options open to the player, an instruction is asked for. Once again, rather than re-inventing the wheel each time we wish to analyse an instruction, we can take advantage of the modular construction of the program to call up the 'split instruction' and 'normal command' subroutines developed for use in the TAKE and DROP routines. By considering carefully the states of the various flags set by these two subroutines, we can transfer control within our new routine as required. Let's consider these flags individually.

The F flag set by the 'split instruction' routine indicates whether the instruction passed to it has a valid format. If the instruction is a one-word command not recognised by the routine, then F takes the value zero — in which case we will want to loop back to get another instruction.

The MF flag is set by the 'normal command' routine if a description of a location is required — this happens when a GO or LOOK command is issued. A RETURN to the main program loop will allow the new location to be moved to, in the former case, or the same location to be described and the special routine re-entered, in the latter case.

The VF flag is also set by the 'normal command' routine. A value of one indicates that the