the rest of the sentence. This task can be easily achieved by scanning through the sentence one character at a time, using MID$, until a space is found. The part of the sentence that lies to the left of the space is the verb, and can be assigned to the variable VB$. The part of the sentence to the right can be assigned to a second variable, NN$. This subroutine is used in Haunted Forest to split the instruction assigned to the variable IS$:

```
2500 REM **** SPLIT COMMAND S/R ****
2510 IF IS$="LIST" OR IS$="END" THEN VB$=IS$:F=1:RETURN
2515 IF IS$="LOOK" THEN VB$=IS$:F=1:RETURN
2520 F=0
2530 LS=LEN(IS$)
2540 FOR C=1 TO LS
2550 A$=MID$(IS$,C,1)
2560 IF A$<>" " THEN 2590
2570 VB$=LEFT$(IS$,C-1):F=1
2580 NN$=RIGHT$(IS$,LS-C):C=LS
2590 NEXT C
2600 :
2610 IF F=1 THEN RETURN
2620 PRINT:PRINT"I NEED AT LEAST TWO WORDS"
2630 RETURN
```

Before the routine attempts to split up the sentence, it first checks to make sure that the command is not one of the three possible single-word instructions — that is, LIST, LOOK or END. If it is a single-word command, then the complete instruction is assigned to VB$, and the routine is exited. If the command is not one of these, then the routine enters a FOR...NEXT loop and begins to scan for the first space. Two techniques used within this loop need special mention. Both relate to the fact that it is extremely bad programming style to perform a conditional jump out of a FOR...NEXT loop without passing through the NEXT statement. Instead, to signal the fact that some condition has been met — in this particular case, that a space has been found — a flag, F, is set to one. Secondly, when the first space has been found, it is a waste of time to continue scanning through the rest of the sentence.

The loop can be neatly terminated at this point by setting the loop counter, C, to its upper limit, LC. Consequently, when the program again reaches NEXT, it will pass on to the following instruction, rather than loop back to the FOR statement. Once the loop has been correctly terminated, then the status of the flag, F, can be tested. A flag value of one indicates that the sentence consists of more than one word, and all that remains to do at this stage is to return to the main loop. If the flag is not one, then the command has only one word and is not one of the single-word commands tested for earlier. In this case, a message stating that two words are required is printed before returning for another command.

NORMAL COMMANDS

For the main part of the program, the player will simply move from location to location and pick up or drop objects that may be found. Therefore, for the majority of locations, the commands GO, TAKE, DROP, LIST, LOOK, END — and their variants — are sufficient to allow the player to do this. Only in unusual circumstances will the player wish to use other more specialised commands. For example, there is little point in using the command KILL if there is nothing present to kill. We can, however, devise a program structure where, on the majority of occasions, only the six commands associated with movement and objects are tested for. When the player enters a new location, the program can test to see if it is one that has been designated 'special' in some way. If this is the case, then any new command requirements can be dealt with by a specific command subroutine for that particular location. Therefore, the main calling loop to our program should do the following:

1) Describe the location and list the exits.
2) Determine whether the location is 'special'.
3) Ask for a command and, if the location is not special, scan the list of normal commands.

There must also be a facility in the main loop to distinguish between a command that causes a move to a new location and one that does not. In the first case, the loop needs to go back to the beginning of the loop to describe the new location and decide whether or not it is special. In the second case, it is necessary only to loop back to ask for a new command. The simplest way to implement this is to use a 'move flag', MF, which is normally set to zero. If a command involves movement then this flag is set to one. The status of MF can be tested at the end of the main loop and the appropriate jump made. Add the following lines to Haunted Forest:

```
270 GOSUB2500:REM SPLIT INSTRUCTION
275 IF F=0 THEN 260:REM INVALID INSTRUCTION
280 GOSUB3000:REM NORMAL COMMANDS
290 IF VF=0 THENPRINT:PRINT"I DONT UNDERSTAND"
300 IF MF=1 THEN 240:REM NEW LOCATION
310 IF MF=0 THEN 260:REM NEW INSTRUCTION

3000 REM **** NORMAL COMMANDS S/R ****
3010 VF=0:REM VERB FLAG
3020 IF VB$="GO" OR VB$="MOVE" THENVF=1:GOSUB3500
3030 IF VB$="TAKE" OR VB$="PICK"THEN VF=1:GOSUB3700
3040 IF VB$="DROP" OR VB$="PUT"THEN VF=1:GOSUB3900
3050 IF VB$="LIST" OR VB$="INVENTORY"THEN VF=1:GOS
     UB4100
3055 IF VB$="LOOK" THEN VF=1:MF=1:RETURN
3060 IF VB$="END" OR VB$="FINISH" THEN VF=1:GOSUB4
     170
3070 RETURN
```

In the first routine, another flag, VF, is used to indicate whether or not the verb has been understood and obeyed. Only when the verb has been isolated is VF set to one. We can insert a failsafe 'I don't understand' statement in the main loop by testing the status of VF. If VF remains zero then the verb in the command has not been recognised by the analysis routine, and the statement is displayed.

In the next instalment of the project, we will deal with subroutines for picking up, dropping and listing objects. For now, we can add a short END command subroutine to our group of normal commands:

```
4170 REM **** END GAME S/R ****
4180 PRINT:PRINT"ARE YOU SURE (Y/N) ?"
4190 GET A$:IF A$<>"Y" AND A$<>"N" THEN 4190
4200 IF A$="N" THEN RETURN
4210 END
```

The LOOK command is also straightforward. To redescribe the current position, we simply need to set the 'move flag', MF, to one and return to the main program loop. Setting MF will cause the main