ON LOCATION

So far in our adventure game programming project, we have developed a map of the locations that form the basis of a game and written a utility routine that formats output to the screen. We are now in a position to design routines that describe locations within the game and allow the player to move between locations.

The basic description of each location is held in the array LN$() (see page 767) and can be accessed simply by specifying the number of the location arrived at. In Haunted Forest, the position held by the player at any given time is stored in the variable P, and, therefore, the description of that location is stored in LN$(P). When the location data was first designed the description's final grammatical context was kept in mind; the description always being phrased in such a way that it could be prefixed by 'You are...'. For a given location, P, the description can be formatted and output to the utility developed in the last instalment, by combining 'You are' with the description held for that location in the array LN$(). Line 2010 in the Haunted Forest listing shows this.

In addition to the basic description of the location arrived at, the player will also want to know if any objects are present. The objects used in the game are stored — together with their initial positions in the inventory — in a two-dimensional array, IV$(,). For example, IV$(N,1) holds the description of the Nth object in the inventory, and IV$(N,2) holds its position. If we wish to determine whether or not there is an object at a particular location we must search through the inventory, checking each object's position against the number of the location that is being described. As there are only three objects in Haunted Forest and eight objects in Digitaya, a simple linear search using a FOR...NEXT loop can be implemented.

Lines 2040-2080 show the search loop used in Haunted Forest. The second column of the inventory array is scanned for a match with the current location, P. When a match is found, then the corresponding description is added to the sentence that describes the objects. As more than one object may be present in any one location, we must allow for the construction of a sentence where a list of objects is given, each separated by a comma. By using SP$, initially as a null string, and later as a comma, we can insert the correct punctuation between each item. A flag, F, initially set to zero, is set to one to signal the fact that an object match has been found during the search. If the flag remains at zero at the end of the search, then no objects are present, and this fact can be output to the player — as in line 2090 of Haunted Forest.

```
2000 REM **** DESCRIBE LOCATION ****
2010 SN$="YOU ARE "+LN$(P):GOSUB5500
2020 SN$="YOU SEE "
2030 REM ** CHECK INVENTORY FOR OBJ **
2040 F=0:SP$=""
2050 FOR I=1 TO 3
2060 IF VAL(IV$(I,2))<>P THEN 2080
2070 SN$=SN$+SP$+"A "+IV$(I,1):F=1:SP$=", "
2080 NEXT I
2090 IF F=0 THEN SN$=SN$+"NO OBJECTS"
2100 GOSUB5500:REM FORMAT OUTPUT
2110 RETURN
```

The data containing details of the possible exits from each location is held in the array EX$(). Each string value is made up of eight digits. By subdividing these eight digits into groups of two, we obtain — working from left to right — the

A Room With A View

The details of the locations in our adventure game are held in three string arrays, which contain object names and whereabouts (VS), location exits (EXS) and descriptions (LNS). EXS (34), for example, might contain the eight-digit number 33390027, showing that location 34 connects to locations 33,39 and 27 by its north, east and west exits respectively. LNS(34) contains ' The Middle Of Memory', which describes location 34. IVS(2,2) contains the number 34, showing that IVS(2,1) — The Key — is in location 34. Given the current location number the program assembles this information into a description