SPECIAL ASSIGNMENT

In the last instalment of our adventure game project we designed routines to enable the player to pick up objects. Now we must develop the corresponding routines that allow the player to drop any objects he may be carrying. We also look at the first of the 'special' locations.

The DROP subroutine bears many similarities to the TAKE routine described on page 846. Indeed, we can use the same object checking routines that were developed for use with the TAKE command. Three checks on the object are made during the TAKE routine. The first is designed to test whether or not the second part of the command phrase contains a valid object. This is done by checking each word of the command phrase systematically against the object names in the inventory array — IV$(,). If a match is found then a variable, F, is set, giving the position of the matched object within the array. This validity check must also be used in the DROP routine to establish whether the object exists and, if it does, to determine its position in the inventory.

The second check used in the TAKE routine is also used in the DROP routine; this tests whether the player holds the object specified in the command in the inventory of carried objects — IC$(). Obviously, a player cannot drop an object that he is not carrying! The third test used in the TAKE routine checks to ensure that the object to be picked up is at the player's current location, as determined by the position variable, P. However, as the object to be dropped must be held by the player, its position will not appear in the main inventory, and this third test is, therefore, not needed by the DROP routine.

Assuming that both tests result in a favourable outcome, then the following changes must be made to both the main and the player's inventories:

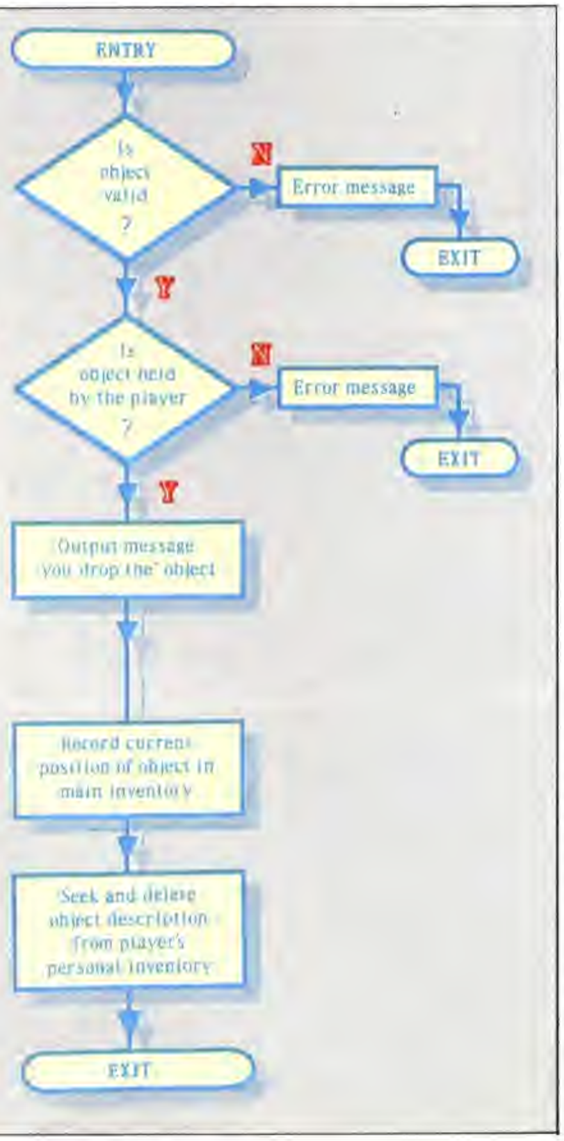
1) The position of the object to be dropped will now be specified by F. The current position, P, must be entered in the main inventory array in position IV$(F,2).
2) The object description must be deleted from the player's personal inventory of objects carried, IC$(). This is best done by searching through the array until the appropriate object is found and replacing it with a null string.

The logic of the DROP routine is shown in the flowchart. Here is the listing for the routine in the Haunted Forest game:

```
3900 REM **** DROP S/R ****
3910 GOSUB5300:REM VALID OBJECT
3920 IF F=0 THEN SN$="THERE IS NO "+W$:GOSUB5500:
RETURN
3930 :
3940 REM ** IS OBJECT IN CARRIED INVENTORY **
3950 OV=F:GOSUB5450
3960 IF HF=0 THEN SN$="YOU DO NOT HAVE THE "+IV$
(F,1):GOSUB5500:RETURN
3970 :
3980 REM ** DROP OBJECT **
3990 SN$="YOU DROP THE "+IV$(F,1):GOSUB5500
4000 IV$(F,2)=STR$(P):REM MAKE ENTRY IN INVENTORY
4010 :
4020 REM ** DELETE OBJECT FROM CARRIED INVENTORY
**
4030 FOR J=1TO2
4040 IF IC$(J)=IV$(F,1) THEN IC$(J)="":J=2
4050 NEXT J
4060 RETURN
```

It can be seen that one of the major advantages of programming in modules is that the same routines can be accessed for different purposes. By using a system of flags, decisions can be made within short

Drop It!
The logic of the DROP routine is similar to that of the TAKE routine (see page 846), but only the validity of the object and its presence in the player's inventory need be checked before the DROP command is executed. The main inventory is then updated to show a new location for the object, and the object name is deleted from the player's inventory


ENTRY
Is object valid ?
N
Error message
EXIT
Y
Is object held by the player ?
N
Error message
EXIT
Y
Output message 'you drop the' object
Record current position of object in main inventory
Seek and delete object description from player's personal inventory
EXIT

IAN McKINNELL