own imagination conjuring up a picture based on a textual description. However, the rise in the popularity of adventure games is almost certainly attributable to the enhanced visual appeal that graphics give, and, although some recent micro games use only simple pictures to enhance the text, others attempt to make the game visual.

In our programming project we shall be looking at the techniques involved in programming an adventure game. During the project, you will be given sections of a listing to an adventure game called Digitaya, which will build into a complete program. In this game, the player is cast as an 'electronic' agent given the task of descending into a microcomputer to locate and rescue the mysterious Digitaya from the clutches of the machine. There are many dangers and difficulties along the way and you have to use all your knowledge of computers to good effect to escape unharmed. The program is, as far as possible, written in 'standard' BASIC, with 'Flavours' given where appropriate. Therefore, provided you have sufficient memory capacity, the program will run on your computer. As we are going to discuss the various programming techniques in detail, it would be difficult not to give away many of the secrets of the game, and this would spoil, to some extent, the pleasure of playing it when it is complete. We will, therefore, construct a shorter game called Haunted Forest, in parallel with Digitaya, which will demonstrate the techniques and algorithms used to build the larger game.

MAKING A MAP

The starting point for the design of our adventure game is to construct a map of the fantasy world that we are imagining. On this map, we mark out the various locations within the world, the position of any objects to be found, and signify those locations that are considered 'special'. Most locations on the map will simply allow the player to move in and out of them, and pick up or drop any objects that are there. Special locations may be perilous (a swamp or a place where a dragon lurks), or they may require a series of special actions to be performed before you can enter into, or exit from, them.

The best way to begin making a map is to consider roughly how many locations are needed for the game. Haunted Forest has 10 locations and was designed on a five by five grid (as shown in the illustration), whereas Digitaya has nearly 60 locations and was originally designed using a 10 by 10 grid.

The grid squares are initially unnumbered and the designer starts by filling in locations on the map. On the Haunted Forest map there is a path, two tunnels, a swamp, a clearing and a village. The positions of several objects are also marked at the bottom of the squares where they are located. Those locations marked with an asterisk (*) are 'special' and will be treated in a different way to the rest of the locations.

Once the layout has been finalised we can number each location. The only special consideration we have taken into account in choosing the location number is that all the special locations have been numbered first. The order in which the others are numbered is not important, but once numbers have been selected it is important that they are not changed later.

PROGRAMMING THE MAP DATA

The first programming task is to convert the information in the map into data for the program. There are many ways of doing this, but what we will do here is use two one-dimensional arrays to hold the map data. The first array, LN$(), holds descriptions of each location. For example, for location 7, LN$(7) will contain 'on a path'. When the data is used later in the program to describe a location it will be prefixed by the words 'You are'.

The second array, EX$(), holds data about the possible moves that can be made out of a location. Both of our games limit themselves to four directions: North, East, South and West. EX$() provides information about the location number to be moved to for each of the four directions. The data is held as a string made up of eight digits. The location number for each direction is entered in the order NESW, using a two-digit number for each direction.

For example, location 7 has exits to the North, South and West, but none to the East. The first two digits of EX$(7) are 08 (not just 8), which shows that location 8 is to the North. The second pair of digits, 00, indicates that there is no exit in this direction (East). The digit pairs 03 and 06 represent the locations found to the South and West of location 7. Using this system, up to 39 locations could be specified; if more than this were required then the data for EX$() would have to be entered as groups of three digits.

The three objects in the Haunted Forest are read into another array — IV$(,). This two-dimensional array keeps track of the position of each object as it is moved around the forest. Each object has a description and its starting location on the map. For example, IV$(C,1) may be GUN, and its position at the start of the game is given by IV$(C,2), As the objects are carried around during the game the position members of the array will be updated accordingly.

At the end of the map data in both of our listings there is another item of data. This is a 'checksum' and is given to ensure that the direction data has been typed in correctly. This is done by calculating a running total of the data values, which is compared against the checksum. If these are not the same then a mistake has been made and the program will stop running. You will notice that in Digitaya two checksums are used. This is because the total sum of all the direction data is too large to be held easily in one checksum, so a total for the left-hand and right-hand four digits is calculated separately. In the next instalment of the project, we will design routines to handle and display the map data assembled here.