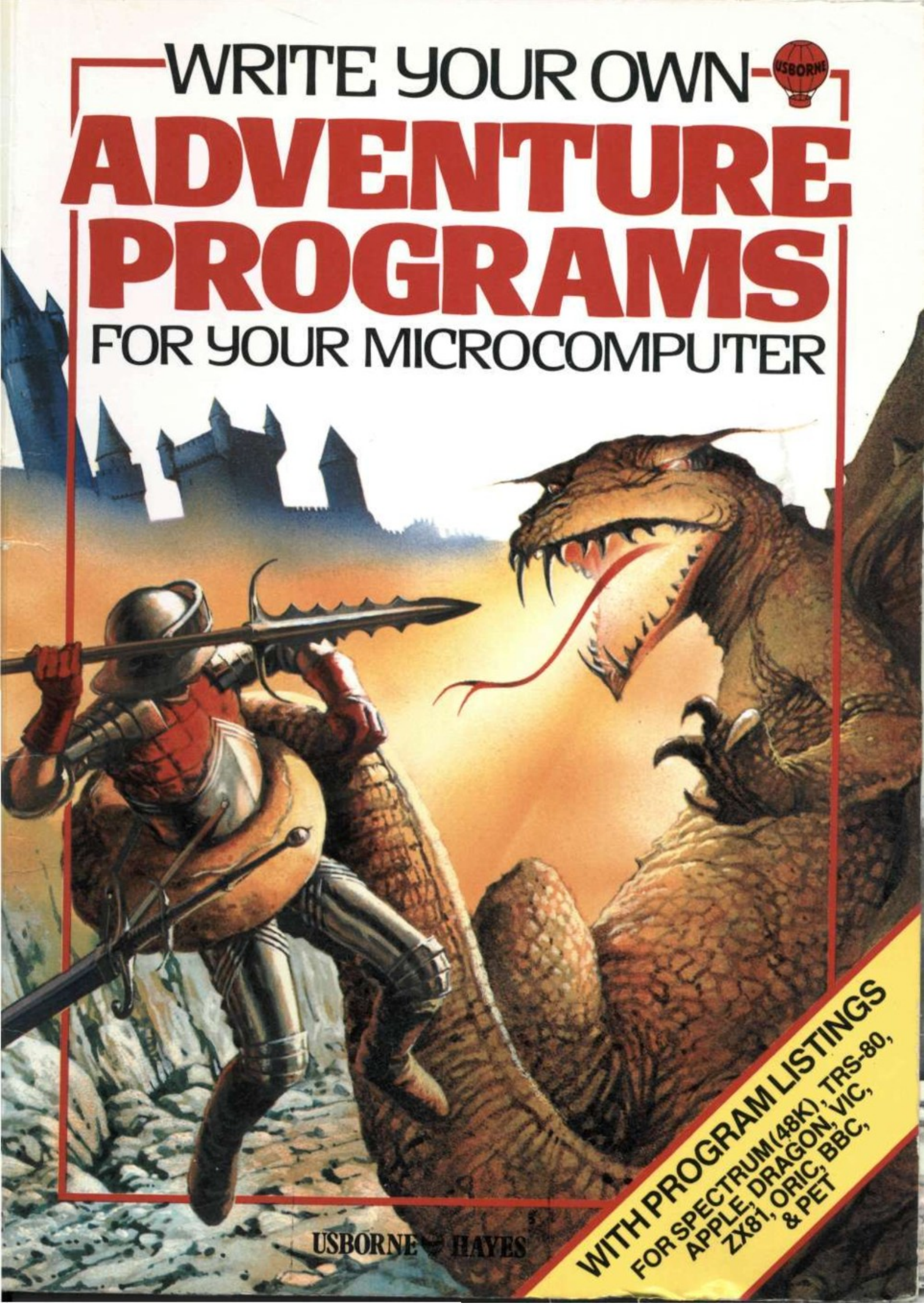
WRITE YOUR OWN

ADVENTURE PROGRAMS

FOR YOUR MICROCOMPUTER



WITH PROGRAM LISTINGS FOR SPECTRUM (48K), TRS-80, APPLE, DRAGON, VIC, ZX81, ORIC, BBC, & PET

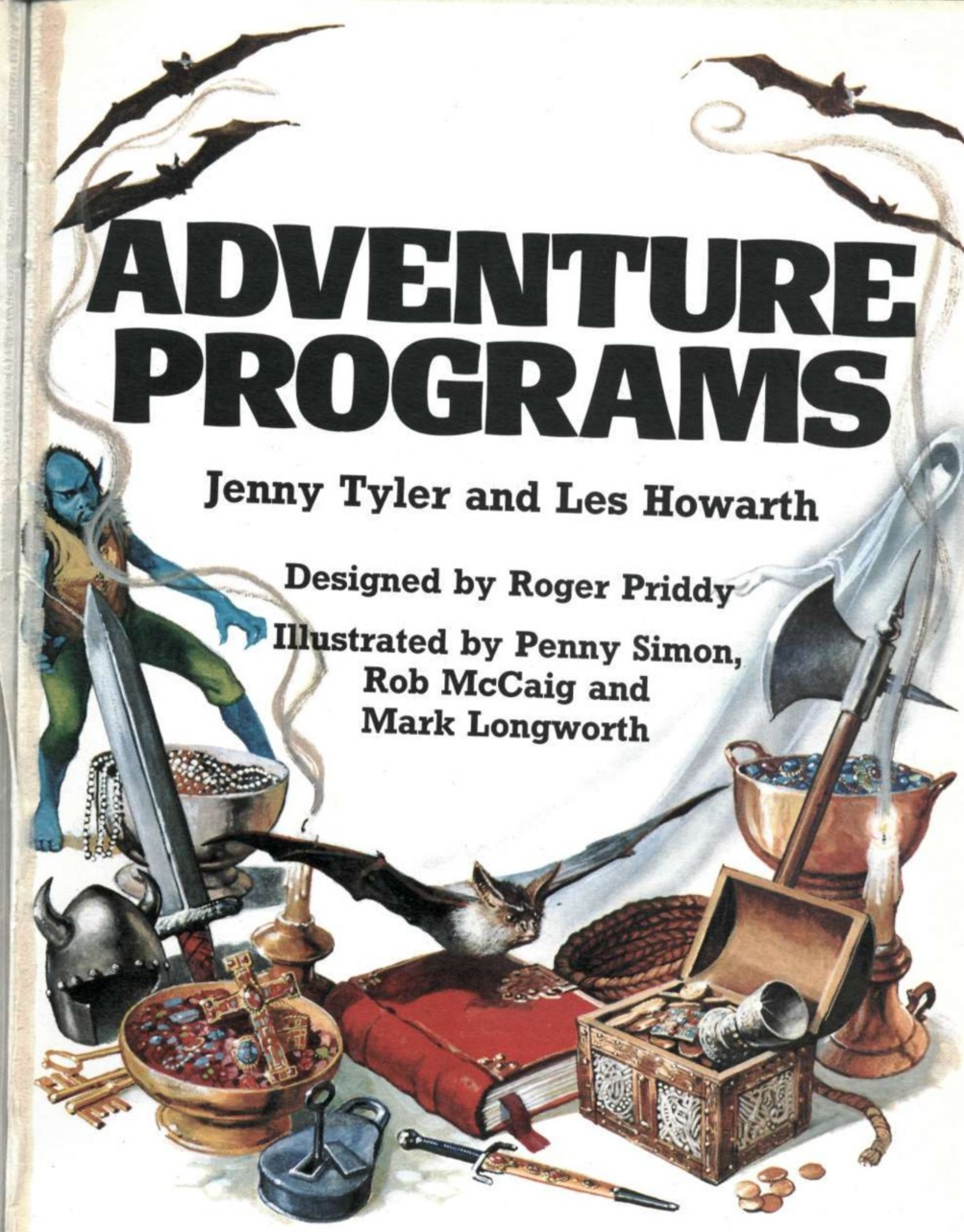
USBORNE ♥ HAYES

ADVENTURE PROGRAMS

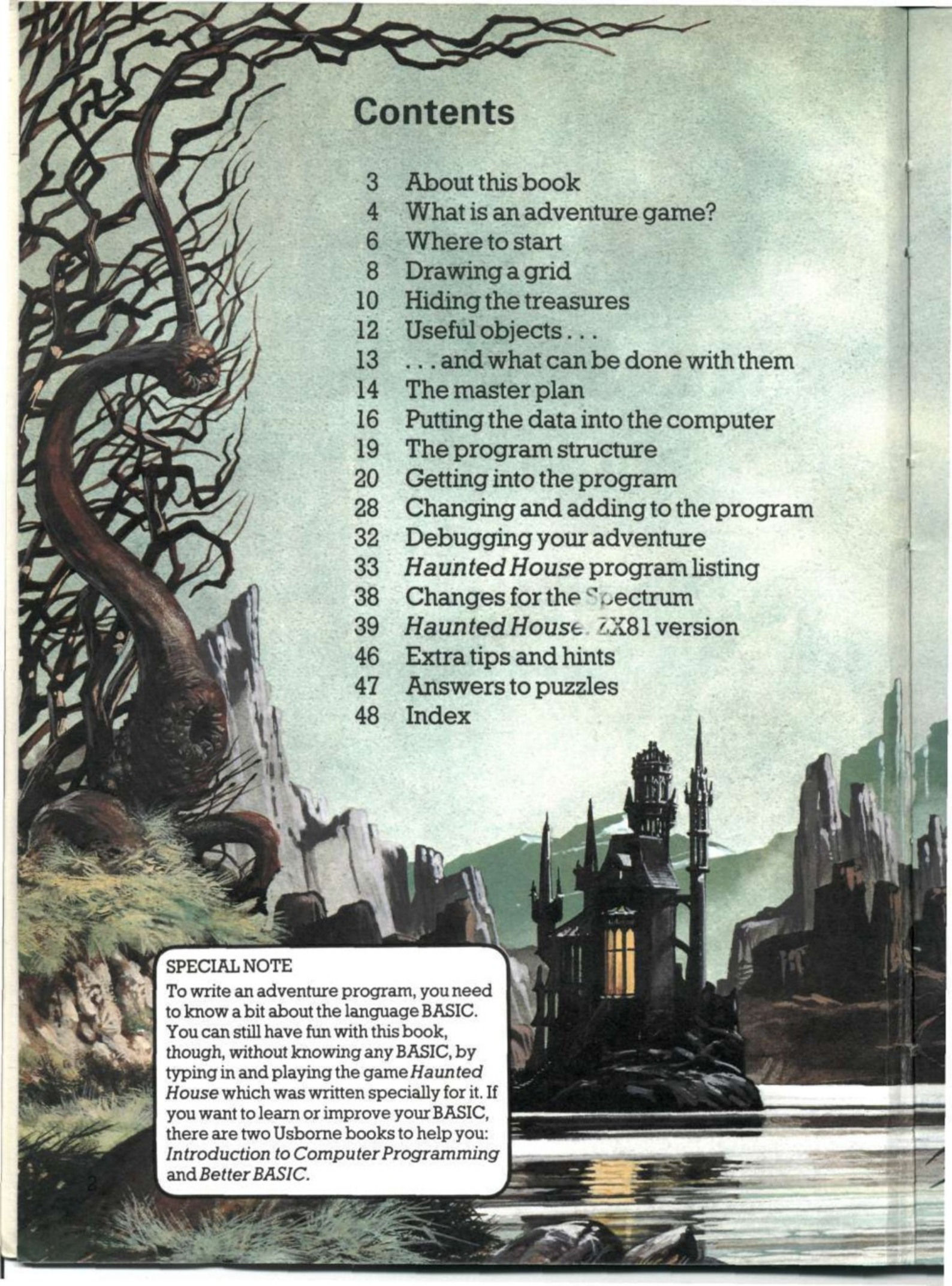
Jenny Tyler and Les Howarth

Designed by Roger Priddy

Illustrated by Penny Simon,
Rob McCaig and
Mark Longworth



ZX81 version of *Haunted House* by Chris Oxlade



Contents

3 About this book
4 What is an adventure game?
6 Where to start
8 Drawing a grid
10 Hiding the treasures
12 Useful objects . . .
13 . . . and what can be done with them
14 The master plan
16 Putting the data into the computer
19 The program structure
20 Getting into the program
28 Changing and adding to the program
32 Debugging your adventure
33 *Haunted House* program listing
38 Changes for the Spectrum
39 *Haunted House*, ZX81 version
46 Extra tips and hints
47 Answers to puzzles
48 Index

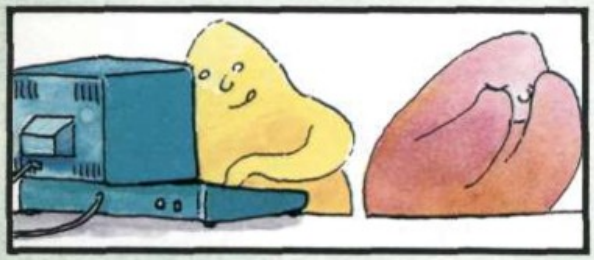
SPECIAL NOTE

To write an adventure program, you need to know a bit about the language BASIC. You can still have fun with this book, though, without knowing any BASIC, by typing in and playing the game *Haunted House* which was written specially for it. If you want to learn or improve your BASIC, there are two Usborne books to help you: *Introduction to Computer Programming* and *Better BASIC*.

About this book

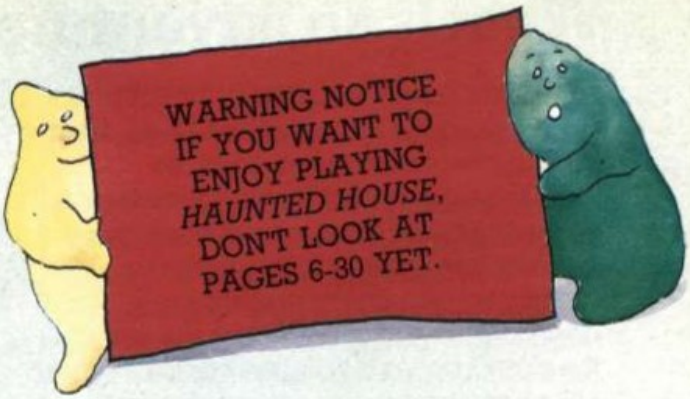
Writing an adventure game from scratch is quite a daunting task, especially if you are new to computer programming. This book allows you to start as gently as you like by giving you an adventure listing which you can type in and play, change and add to as much as you like, or use as a skeleton program for your very own adventure story.

The game written for this book is called *Haunted House* and you will find the main listing for it on pages 33-38. This will work on any computer which uses Microsoft-style BASIC, including BBC, Dragon, Oric, TRS-80 and expanded VIC 20, and has changes to make it work on a 48K Spectrum (Timex 2000). A special ZX81 (Timex 1000) version of the game is listed on pages 39-45. Turn straight to these pages if you want to play the game before you find out how it works. It is a specially good idea to do this if you've never played an adventure game.



If you are used to looking at program listings, you will probably pick up a few clues about how the game works as you type it in. To avoid this, you could try to persuade someone else to type it in for you.

As the program is so long and complicated, you will need to type it in extremely carefully. It is worth typing slowly and checking each line as you go, as you only need do it once. You can save the program on tape for when you want to play it again or produce an adapted version.



On pages 6-15, you will find out how to plan an adventure and on pages 16-27 how to structure and write the program for it. You may find this section quite difficult. Don't worry if you do, just work through each bit slowly and carefully until you feel you have grasped the ideas in it, then go on to the next bit.

As you work through these pages it is a good idea to write a practice game of your own, following each step carefully. Don't worry if it isn't a specially good game; it will help you to understand how the program works and see the problems you need to solve in order to write an adventure. A good adventure needs careful planning to make it interesting and exciting to play. Remember, you don't need to touch your computer until you have planned your game down to the last detail.

You will find some extra tips and hints on adventure writing on page 46, and on page 47 there are answers to the puzzles set in the book.

After playing *Haunted House* a few times, you will probably want to make changes to it. Pages 28-31 give you some ideas for producing your own version.

What are the rules?

If you have played an adventure game before you will know what to expect from *Haunted House*. If you haven't, all you need to know is that the computer will ask you what you want to do and you tell it, using not more than two words.

It is a good idea to pick up anything on the way that looks valuable or useful and to try using these things in any way you can think of to solve the problems you encounter. Type SCORE to find out how many points you have and if you have won.

What is an adventure game?

An adventure game is like a story in which the player is the hero. Unlike a book, where the sequence of events is fixed, an adventure game is different each time it is played because the player chooses what happens at each stage. By giving the computer instructions in response to descriptions which appear on the screen, the player goes on a dangerous journey into an unknown land. The aim is to survive whatever dangers may arise and return with treasures.

The first adventure game was written in 1976 on a mainframe computer at Stanford University in the U.S.A. by William Crowther and Don Woods. It is often referred to as *Colossal Cave, Colossal* or just *Adventure*, and a version of it is now available for most home computers. It was written in the scientific computer language, Fortran, which, unlike BASIC, cannot handle words. All the data for the game had to be indexed and stored on disc.

The first people to play adventures were computer professionals, as home computers did not exist. A version of Crowther & Woods' adventure is still included with most large business computer systems to show people who are not used to computers that they can be "friendly". These disc-based adventures often occupy more than 250K and are very complicated to play.

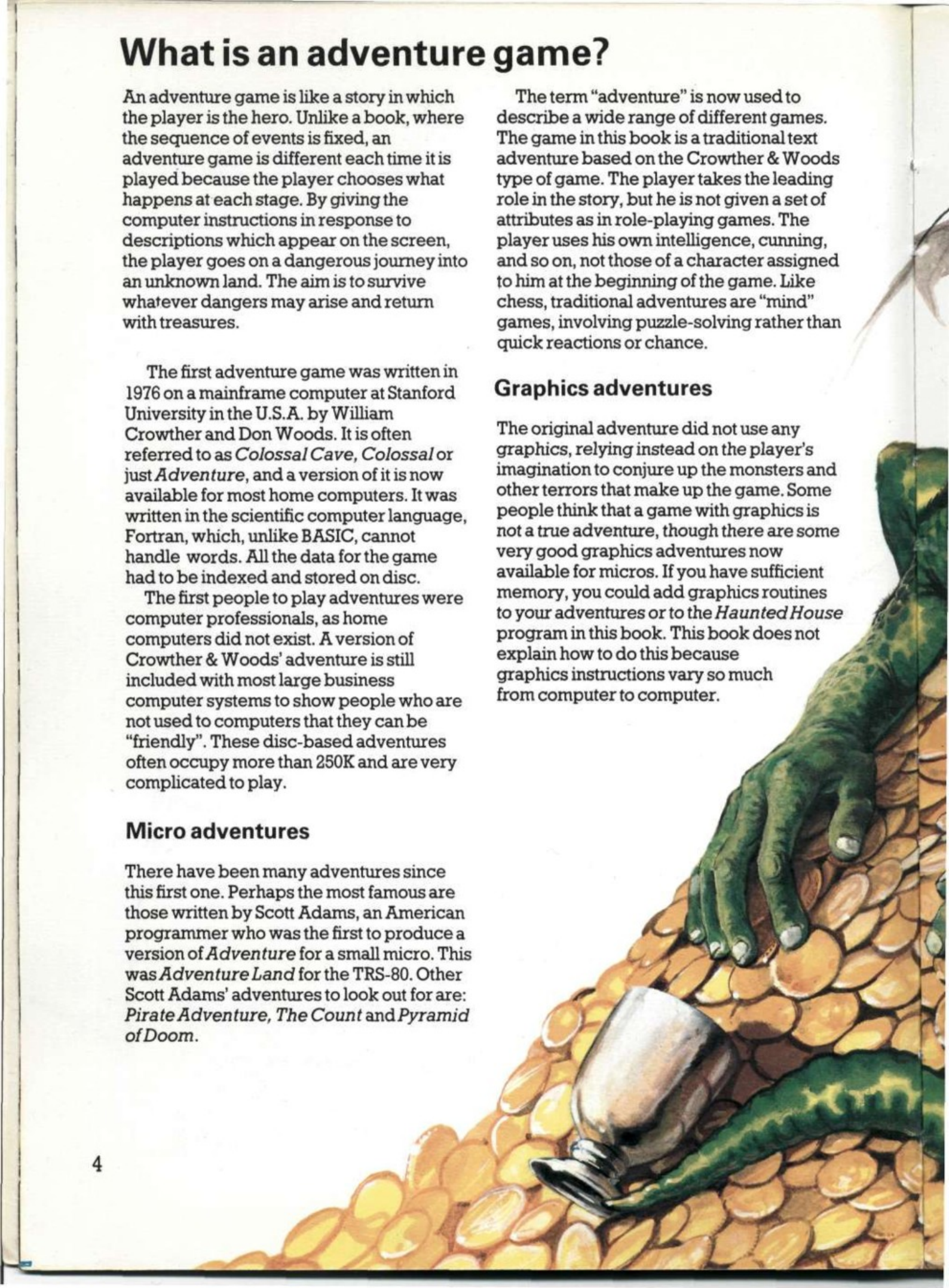
Micro adventures

There have been many adventures since this first one. Perhaps the most famous are those written by Scott Adams, an American programmer who was the first to produce a version of *Adventure* for a small micro. This was *Adventure Land* for the TRS-80. Other Scott Adams' adventures to look out for are: *Pirate Adventure, The Count* and *Pyramid of Doom*.

The term "adventure" is now used to describe a wide range of different games. The game in this book is a traditional text adventure based on the Crowther & Woods type of game. The player takes the leading role in the story, but he is not given a set of attributes as in role-playing games. The player uses his own intelligence, cunning, and so on, not those of a character assigned to him at the beginning of the game. Like chess, traditional adventures are "mind" games, involving puzzle-solving rather than quick reactions or chance.

Graphics adventures

The original adventure did not use any graphics, relying instead on the player's imagination to conjure up the monsters and other terrors that make up the game. Some people think that a game with graphics is not a true adventure, though there are some very good graphics adventures now available for micros. If you have sufficient memory, you could add graphics routines to your adventures or to the *Haunted House* program in this book. This book does not explain how to do this because graphics instructions vary so much from computer to computer.



What kind of program is it?

An adventure program is really a kind of database. A database is a computer filing system which stores information and allows it to be called up in a variety of different ways, and it can have all kinds of serious uses. An adventure program is an "interactive" database. The player moves through it, altering or "updating" information as he does so. As you work through the book, you will see how particular words are used as "keys" to unlock certain items of information. This technique can be used to restrict access to certain information in a "serious" database.

You can learn some useful programming techniques by writing an adventure. As the program is such a complicated one, it shows how important it is to plan it in detail before switching on your computer. It also makes you think of all the things a person using the program might try to do. If you work out a way of making the computer deal with *any* input, however silly, you will be able to write programs which don't crash.

Many of the adventures you can buy on cassette are written, at least partly, in machine code. This allows more information to be packed into the computer and makes the game run faster. If you know a bit about machine code, you could experiment with adding machine code routines to your adventures.



Where to start

When you write an adventure game you are inventing a fantasy world where you make up all the rules. You decide where it is, what sort of creatures and things live there and what these creatures and things can and cannot do. Your world can be an alien city, for instance, or an underground palace



where elves, wizards and trolls live or a mysterious castle which is the home of dragons and other strange monsters. It could even be a time in the past involving actual historical people and facts.

Many adventures use magic of some kind. You can decide how closely your world sticks to the rules of the real world and how much magic is allowed. Whatever you choose to do, try to make sure the rules are logical or players will find the game silly and frustrating.

Having decided on a theme for your adventure world, you then need to decide on the point of your game. The player might have to escape, or return to a certain place, with treasures, or he might have to rescue someone, or find a secret place and do something there (such as defusing the Mad Scientist's Evil Device for Blowing up the World).

Working out the locations

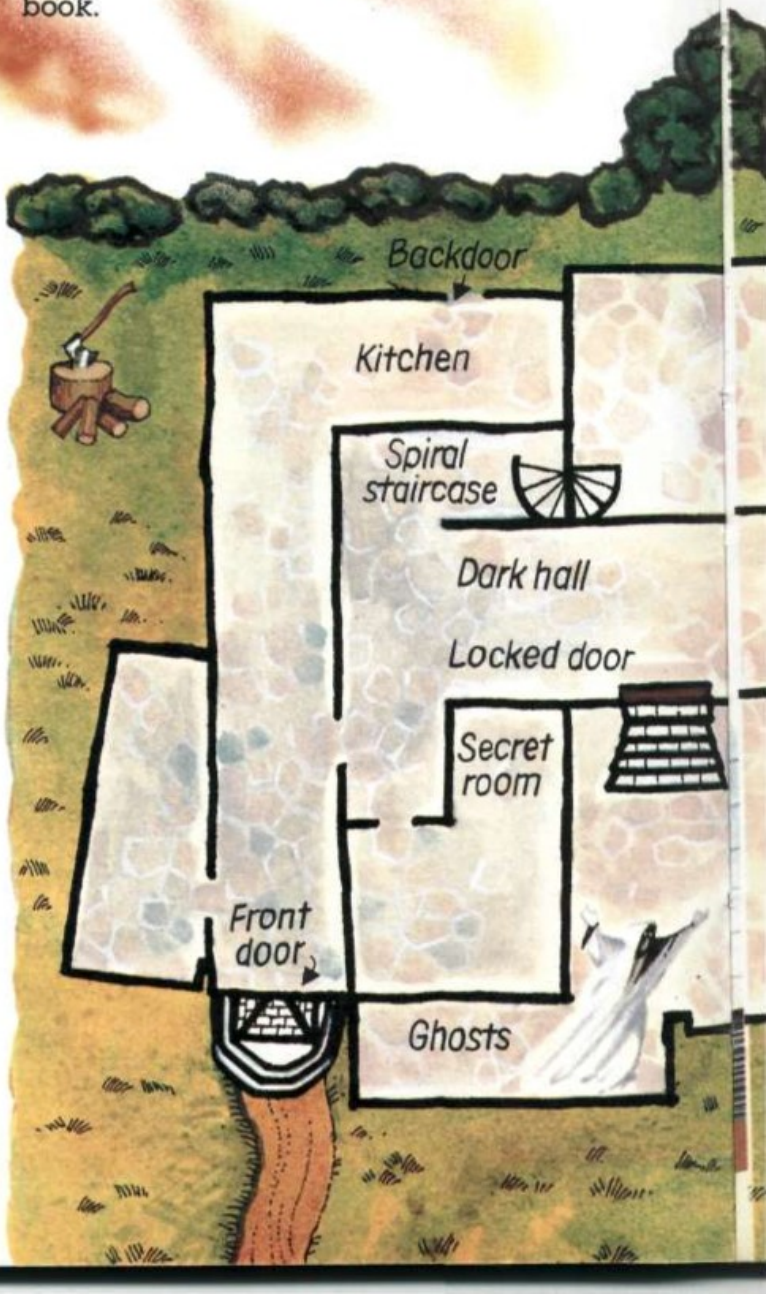
The areas or rooms through which the player moves during an adventure are called locations. Later in the book, you will see how these are numbered to put them in the computer. For the moment, you need to remember that the number of locations you can have depends on the amount of memory your computer has. More locations can make the game more interesting, but leave you less memory space for descriptions of them all. The game in this book has 64 locations with short descriptions.

A location can be indoors or out and could be a room, a cave, half way along a passage, an area of forest, the middle of a field, or anywhere else you like. It is best to decide on the number of locations early on and stick to it, as this affects the whole structure of the game.

Making a map

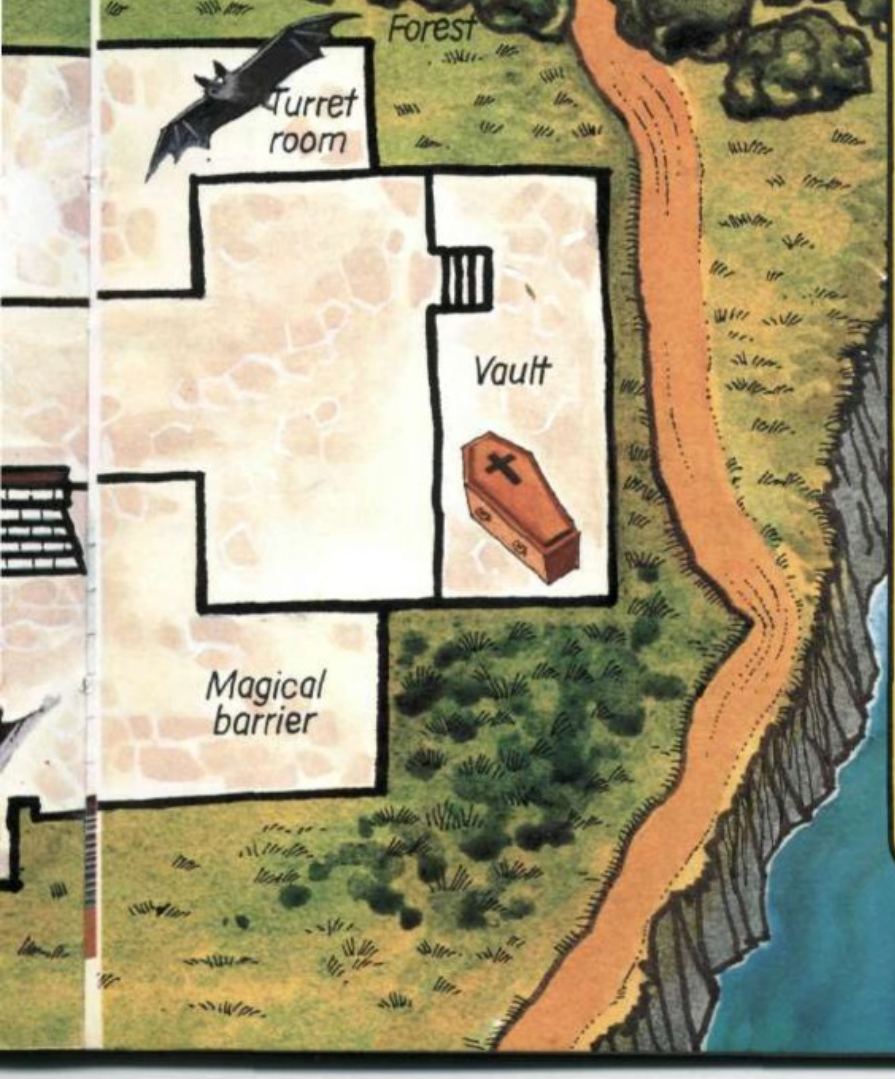
The next stage is to draw a rough sketch map of your world. It need not be accurate to the last detail but should show the overall scale. While you are doing this, think of some ideas for good hiding places for treasures and other objects that the adventurer will need.

Here is a rough sketch map for the *Haunted House* adventure written for this book.



Haunted House is set in a weird house standing on the edge of a sheer cliff. Its strange twisted turrets loom out of an always gloomy sky. No wall seems straight, no corner a right-angle. Not surprisingly no one lives there – well no human that is . . .

People say the richest man in the world spent his last days there. Strangely, no one ever saw his body . . .



Ideas for themes

If you're stuck for an idea for your adventure game, think of films or TV programmes you have seen or books you have read. Remember, though, if you are writing a game to sell, you must not stick too closely to the plot or use the same names for copyright reasons.

Here are some ideas for adventure themes.

DETECTIVE STORY – the player is a detective investigating a terrible crime. The object of the game is to get back to police headquarters with all the evidence. (The items of evidence are the "treasures".)

PREHISTORIC ADVENTURE – the player has travelled through time to the days of cavemen. The object is to return to the present with The Stone, an object of immense power. The adventurer has to make his own weapons and anything else he needs, just as the cavemen do. Prehistoric beasts and cave magic are among the obstacles.

TEMPLE TERROR – the ancient ruins of a temple built by a mysterious, long-lost race are reputed to contain the secret of eternal life. Just hearing about the things that are said to have happened there makes your hair stand on end. The object is to escape with the secret.

Drawing a grid

The first stage in turning your adventure world into a computer game is to transfer your sketch map to a squared up grid. You need one square for each location, so for its 64 locations, the *Haunted House* game needs an 8 × 8 grid.

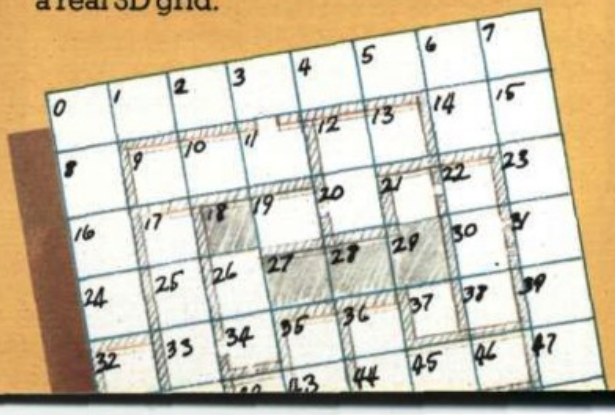
This grid will become the master plan for your adventure, so make it as large and clear as possible. Eventually it will show all the locations and the ways in and out of them, and all the treasures and objects used in the game.

Number each location, starting in the top left-hand corner. Most computers start counting at zero, so use zero as your first location number.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

You may want to change the position of walls and doorways when you work out the routes the player can take, so start by pencilling your map lightly onto the grid. Label each location with a short description, eg "dark cellar" or "dusty room" and then think about the ways in and out of each location. The usual way of marking these is to use points of the compass – north being towards the top of the page, south down, east to the right and west to the left.

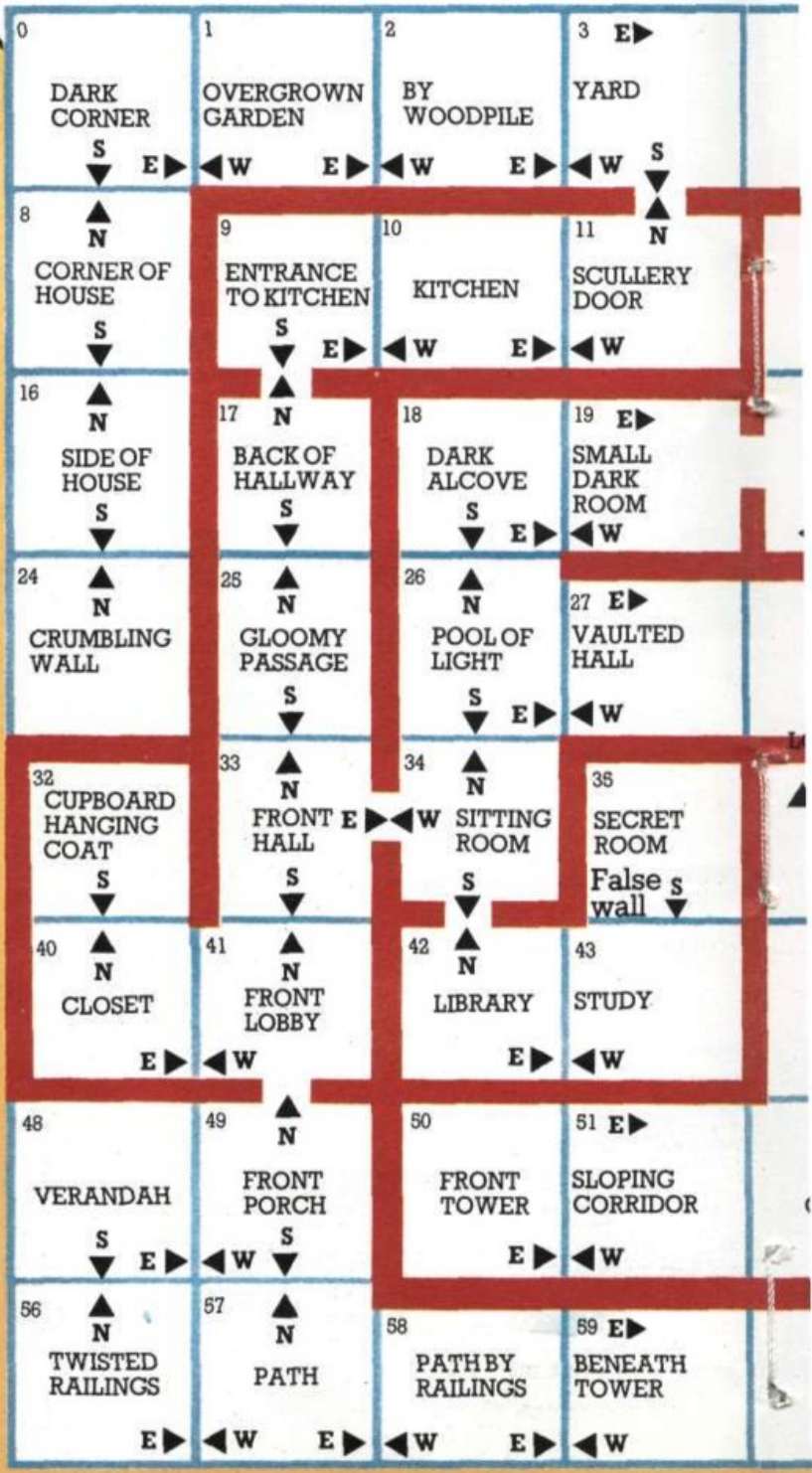
By including staircases, ladders or trapdoors in your descriptions, you can use up and down for some of your routes instead of compass points. This makes the game more interesting without the need for a real 3D grid.



Working out the routes

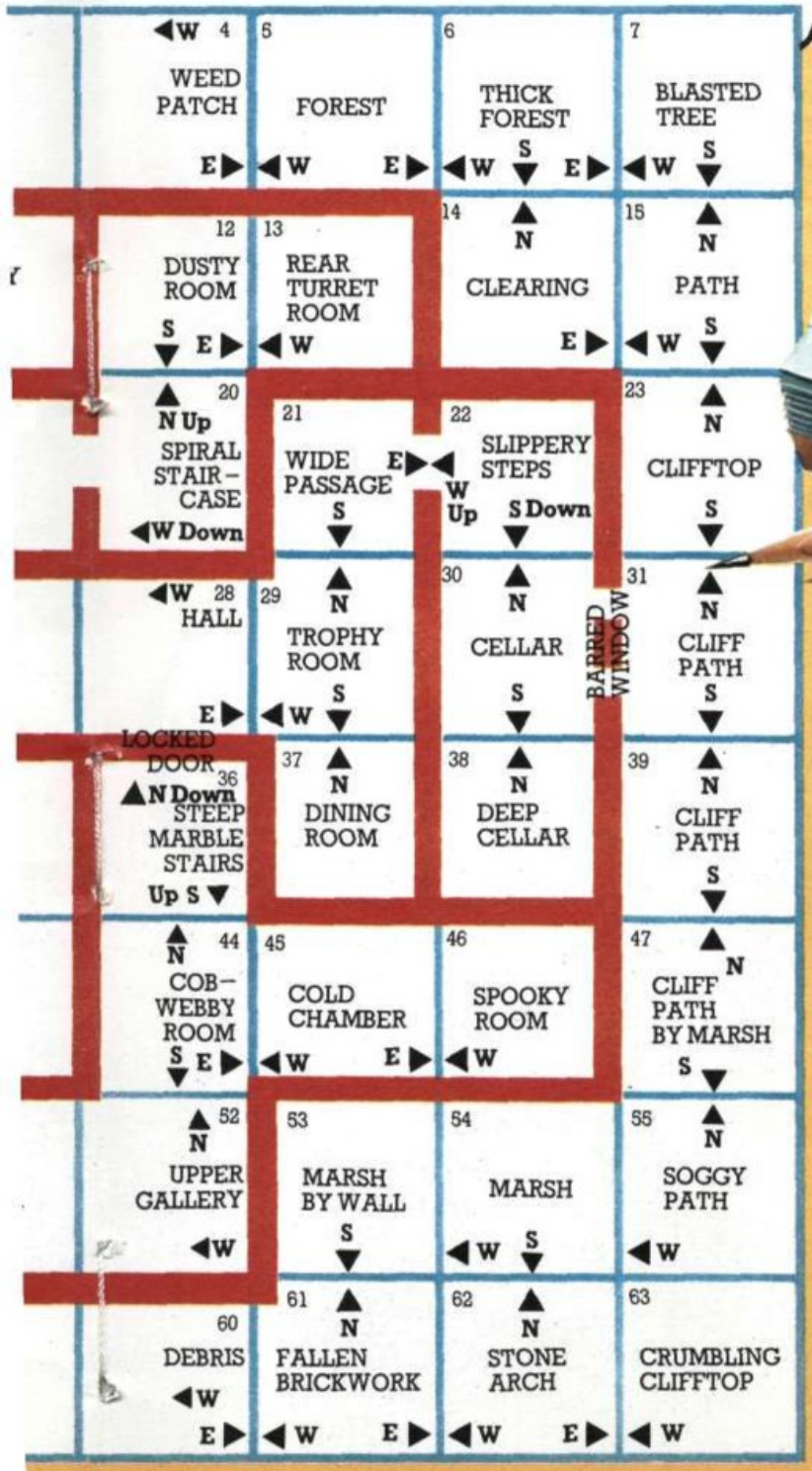
Mark the exits from each location on your grid, like this.

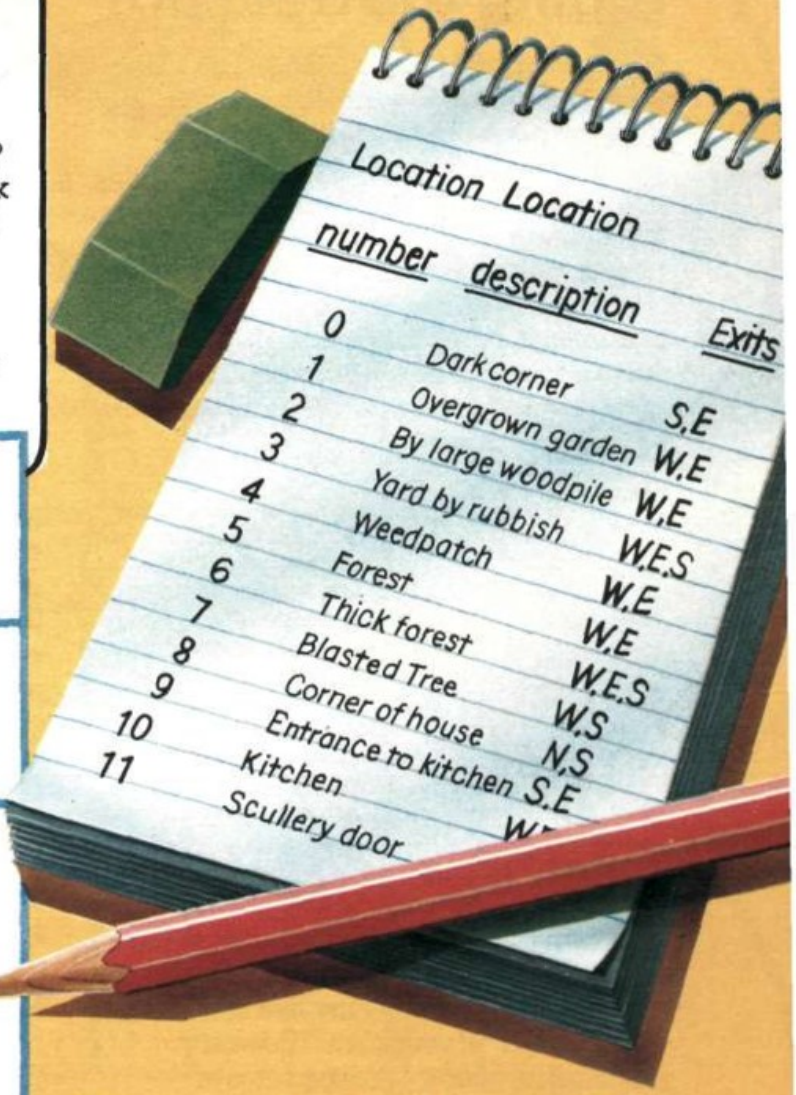
Notice that some locations on this grid have one-way routes, so the player cannot return the way he has come. Make sure there are reasons for these if you use them in your adventure, even if the reasons are magic. In *Haunted House* the front door slams and locks behind the player once he has entered, so he cannot go out again. The



marsh at locations 53 and 54 is also a one-way route, because the boat gets stuck. How many one-way routes can you think of?

When you have settled on your routes, ink in the walls and staircases to fit in with them and make a chart, like the one on the right, listing the location number, its description (this need not be your final version) and its exits. You will find this helps enormously when you start writing the program.





To help you see how the information fits into the program, you could complete this chart using the map on pages 14-15 and check it against the listing on page 37.

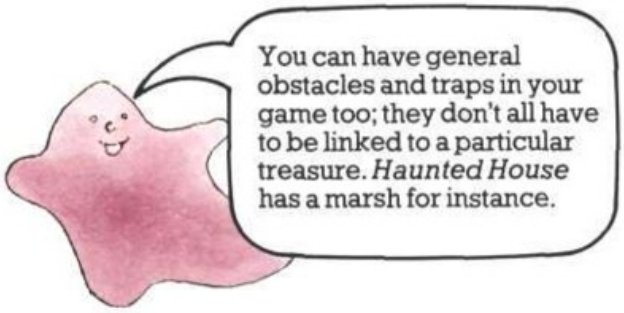
3D games



It is possible to construct real 3D adventures which are set on several levels like the storeys of a house. To do this you need two or more interlocking grids. Such games need a lot of memory (48K is probably the minimum to make it worthwhile) and can get very complicated to write.

Hiding the treasure

Having mapped out your adventure world, you need to come back to thinking about what the player has to do in it. In many adventures, the player has to find valuable objects of some kind and take them somewhere. These could be "real" treasures, like gold and jewels, or they could be something like secret plans and documents, or items of evidence to help solve a crime. If the purpose of your game is to rescue someone, then count this as having one "treasure".



Adding "props"

You need to decide what treasures to have and where to hide them. Hiding the treasures will probably involve including some "props" in your plan. These are pieces of furniture, carpets, items of clothing and so on which the player can open or examine, but which cannot be taken away from the location in which they are found. *Haunted House* has a coffin as one of its props.

Some ideas to think about

Can you think of some "treasures" to fit with these game settings?

1. The headquarters of an international crime syndicate.

2. A distant planet which is known to be more technologically advanced than Earth.

3. A secret scientific research establishment.

Now can you think of some good obstacles to getting them?

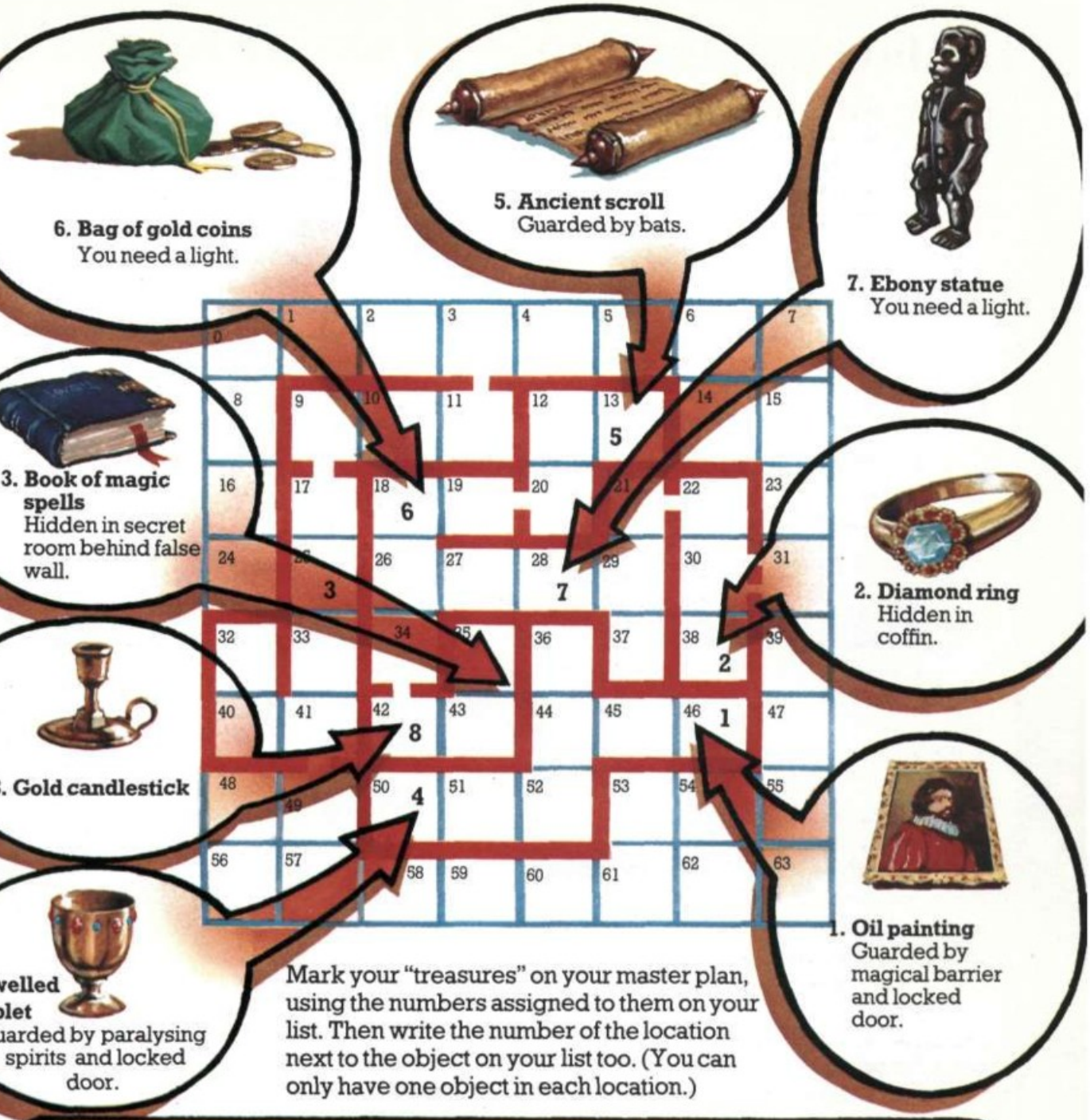
Setting problems for the player

Next you must think about the problems the player will have to solve in order to find and carry away the treasures. The cleverer and more original the problems you invent for the player, the more interesting the game will be to play. The solutions to many of the problems will involve other objects which the player must find and then use in the right way. You will find out about "useful" objects over the page.

Make a list of your valuable objects and number them, starting with 1 this time. (You will find out why on page 16.) List the objects in order of value as this will be useful later on for setting up the scoring system. This is the start of the list of words you want your computer to recognize.

Make a note of the obstacles to getting each treasure too. You might have a monster guard, for example, or a mad axe-wielding troll. Treasures might be in locked drawers, or in safes. They may prove impossible to carry without a container of some kind which is hidden elsewhere. On the left are some puzzles you can think about.





Detective game puzzle

In this game the valuable objects are:

1. Single red hair
2. Brown woollen thread
3. Footprint
4. Set of fingerprints
5. Blood stain
6. Heavy wooden stick

The obstacles to getting these are:

1. The hair will get damaged or lost if you take it as it is. It is on the sleeve of a coat.

2. The thread, invisible to the naked eye, is on the inside edge of a locked drawer.

3. The footprint is in a flower bed outside a window.

4. The fingerprints are on the surface of a large table next to the body. They are invisible.

5. The blood stain is on the carpet.

6. Touching the stick will destroy any prints which might be on it.

What solutions can you think of? There are some suggestions on page 47, though yours might be better.

Useful objects . . .

To help the player solve the problems you set, you will have to include some tools, weapons and other useful objects in the game. The player must find the appropriate objects and use them in the right way to get round the obstacles. You can test the player's ingenuity by not including the most obvious objects. Instead of a key, for example, you could include a hairpin or paperclip for opening a locked door. You can make things more difficult, too, by hiding, say, a torch in one place and the batteries for it in another. The player must find both before he can use them.

Add your objects to the word list you started for your treasures. Don't forget that some of your treasures can have uses too. (You don't need to list them twice.)

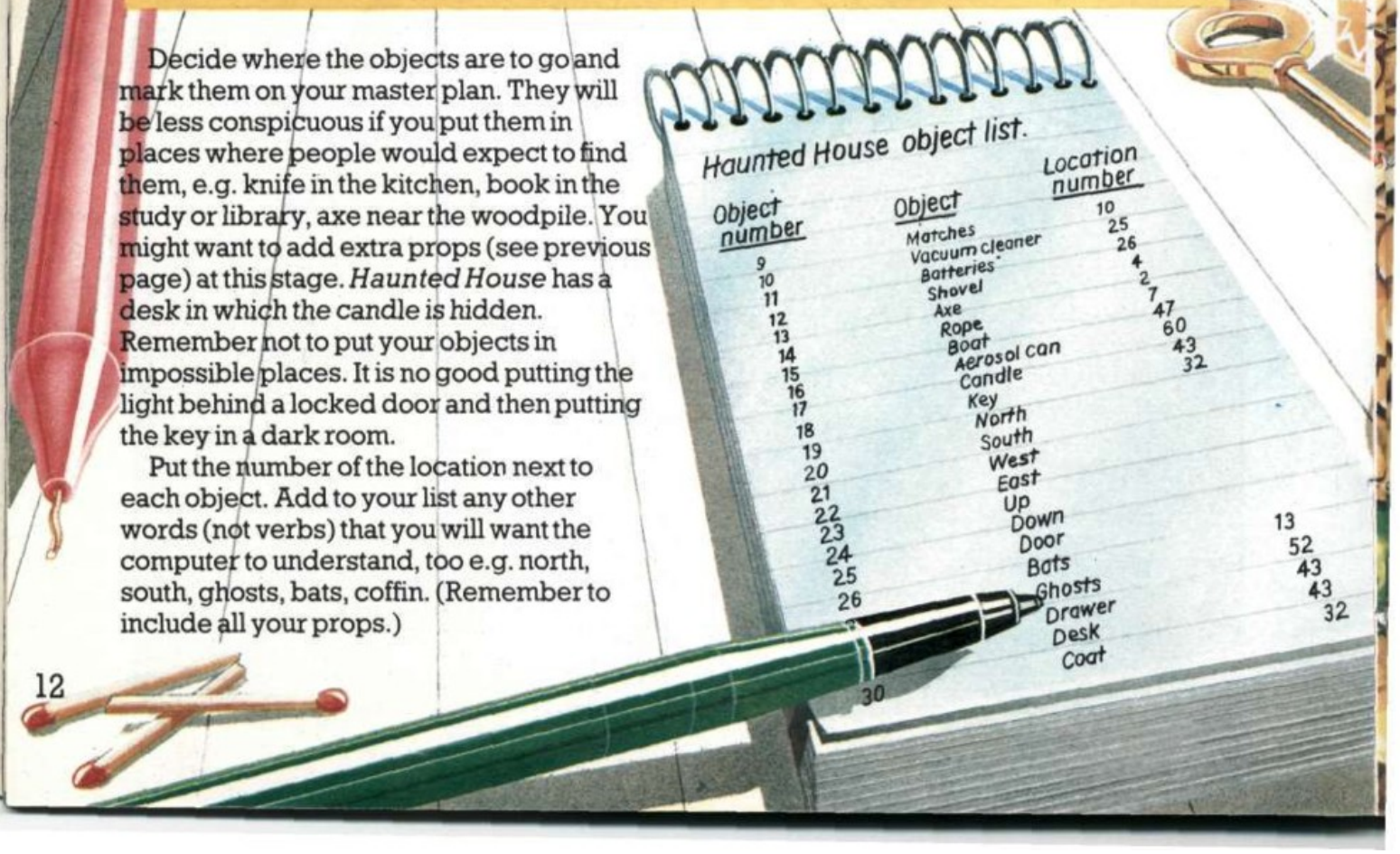
Haunted House problems and solutions

Here are the solutions to the *Haunted House* game problems and the objects needed for them.

Problem	Solution	Objects needed
Too dark to see	Put candle in candlestick Light candle with match	Candle (hidden in desk drawer) Matches Candlestick (already in valuable objects list)
Bats	Spray with "Baticide"	Aerosol can
Secret room	Break down false wall	Axe
Locked door	Unlock	Key
Paralysing ghosts	Suck up with vacuum cleaner	Portable vacuum cleaner Batteries
Magical barrier	Use magic spell	Book of magic spells (already in valuable objects list)
Marsh	Get across in boat (Can only be used once as it gets stuck in mud)	Boat
Barred window	Dig round edge to remove bars	Shovel
Coffin	Open lid	Nothing

Decide where the objects are to go and mark them on your master plan. They will be less conspicuous if you put them in places where people would expect to find them, e.g. knife in the kitchen, book in the study or library, axe near the woodpile. You might want to add extra props (see previous page) at this stage. *Haunted House* has a desk in which the candle is hidden. Remember not to put your objects in impossible places. It is no good putting the light behind a locked door and then putting the key in a dark room.

Put the number of the location next to each object. Add to your list any other words (not verbs) that you will want the computer to understand, too e.g. north, south, ghosts, bats, coffin. (Remember to include all your props.)



. . . and what can be done with them

Now you have decided on the objects to go in your game, what are you going to let the player do with them? You need to make a list of verbs and the things they apply to. This should include "going" verbs too, so that players can give instructions about where they want to move to.

Many adventure programs are constructed so that the computer accepts commands of not more than two words from the player. It checks the first word against a list of verbs you have put in its memory and the second against the object and direction words you have given it. A lot of the fun in writing adventures is trying to think of all the combinations of verbs and objects that the player might try and deciding on what action or reply the computer should give. Writers of business programs need to think in this way too, to prevent their programs crashing because of an unexpected response from the user.

To deal with verbs (and objects) which the computer cannot find in its memory, you can include general replies, such as "Do what with the (object)?". Group together verbs which mean the same thing, such as get and take. You will be able to save memory space by sending the computer to the same routine for both.

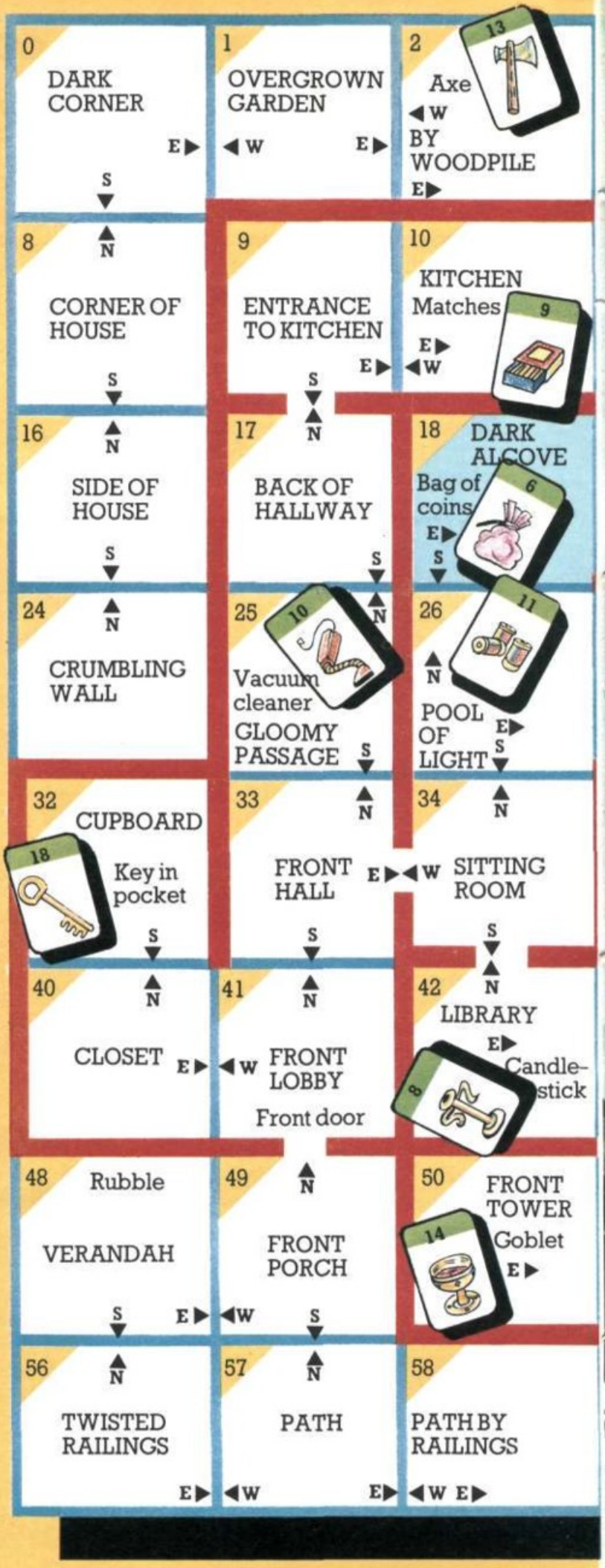
Haunted House verb list

Number your verbs starting with 1. (The computer uses zero for "verb not found", as you will see on page 16.) HELP and INVENTORY (or CARRYING?) are standard adventure game features so include them in the verb list. It is also useful to add shortened forms of GO NORTH etc. (see verb numbers 4-9 in the chart). These reduce the amount of typing the player has to do and make the game faster to play.

Verb number	Verb	Applies to	Action (conditions, if any, in brackets)
0	—	—	Used to indicate "verb not found"
1	HELP	—	Lists all verbs the computer knows
2	CARRYING?	—	Lists all the objects player is carrying
3	GO	DIRECTIONS	Moves position
4	N	—	Shortened form for "GO NORTH"
5	S	—	Shortened form for "GO SOUTH"
6	W	—	Shortened form for "GO WEST"
7	E	—	Shortened form for "GO EAST"
8	U	—	Shortened form for "GO UP"
9	D	—	Shortened form for "GO DOWN"
10	GET	OBJECTS	Pick up object and take it with you (Object must be in the location)
11	TAKE	OBJECTS	Same as GET
12	EXAMINE	ANYTHING	Reveals any concealed object
13	OPEN	DOOR, DESK	Opens door or drawer (must have key for door)
14	READ	BOOKS, SPELL	Displays written clues (must have book of spells or be in library)
15	SAY	ANY WORDS	Says words typed "out loud", e.g. casts spell
16	DIG	—	Makes a hole (must have shovel and be in cellar)
17	SWING	AXE	Breaks down false wall (must have axe and be in study)
18	CLIMB	ROPE	Go up or down rope
19	LIGHT	CANDLE	Turns light on (must have candle, matches and candlestick)
20	UNLIGHT	CANDLE	Turns light off (must be carrying lighted candle)
21	SPRAY	AEROSOL	Removes bats from rear tower (must have aerosol)
22	USE	VACUUM	Sucks up ghosts (must have vacuum and batteries)
23	UNLOCK	KEY, DOOR	Opens door (must have key)
24	LEAVE	ANY WORDS	Leave object behind (must have object)
25	SCORE	—	Prints out score

The master plan

Your master plan and the lists you have made contain all the information, or data, needed for your program. Here is the completed master plan for *Haunted House*. (Don't worry if your master plan doesn't look as elaborate as this.) Over the page, you will find out how to put this data in your computer. Before you touch your computer, though, make sure you have planned out your game to the last detail.



Adventure brain teasers

Here are some situations players might perhaps find themselves in during an adventure. See how many solutions you can think of for each one. There are some suggestions on page 47.

1. You are trapped in a room about three metres square. There are no doors. There is a thick carpet.

2. As you enter a room, a feeling of extreme drowsiness comes over you. You are carrying a small, but quite heavy, rucksack and a handkerchief.

3. You are standing on the battlements of a castle. Beneath you is a horde of angry slaves and behind you armed soldiers. You have a parchment scroll in your hand.

4. You have been invited to dinner by the evil arch-villain. He has taken away all your weapons. As dessert is served, he shows you the remote control for his world decimator weapon.

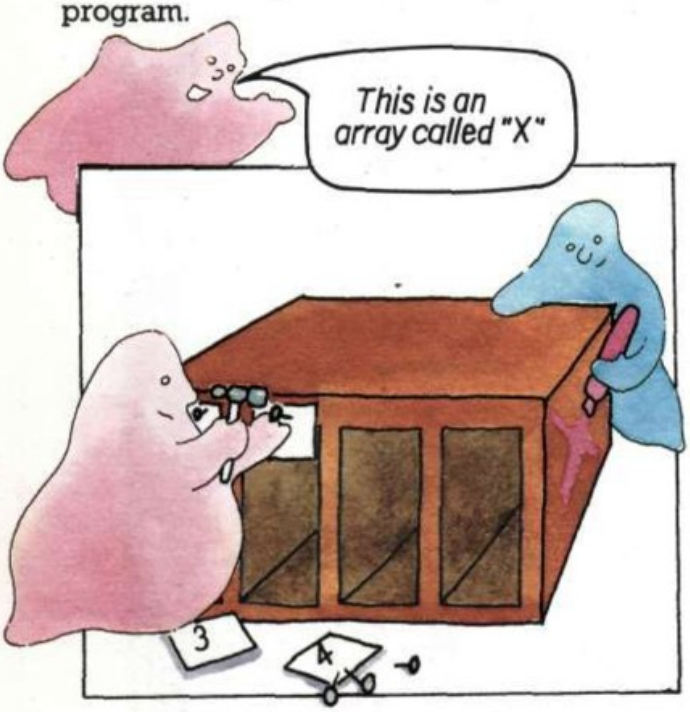



3
Rubbish
YARD
4
Shovel
WEEDPATCH
5
FOREST
6
THICK FOREST
7
BLASTED TREE
Rope
11
SCULLERY DOOR
12
DUSTY ROOM
13
Scroll
Bats
REAR TURRET ROOM
14
CLEARING
15
PATH
19
SMALL DARK ROOM
20
W Down
SPIRAL STAIRCASE
21
WIDE PASSAGE
22
SLIPPERY STEPS
W Up
S Down
23
CLIFFTOP
27
VAULTED HALL
28
Statue
HALL WITH LOCKED DOOR
29
TROPHY ROOM
30
CELLAR
Barred Window
31
CLIFF PATH
35
SECRET ROOM
Book of spells
36
N Down
STEEP MARBLE STAIRS
37
DINING ROOM
38
Ring
Coffin
DEEP CELLAR
39
CLIFF PATH
False wall
43
STUDY
Candle in drawer
44
COBWEBBY ROOM
ONE WAY
45
COLD CHAMBER
46
SPOOKY ROOM
Painting
47
Boat
CLIFF PATH BY MARSH
51
SLOPING CORRIDOR
52
UPPER GALLERY
Paralysing ghosts
53
MARSH BY WALL
54
MARSH
55
SOGGY PATH
59
BENEATH TOWER
60
DEBRIS
Aerosol can
61
FALLEN BRICKWORK
62
STONE ARCH
63
CRUMBLING CLIFFTOP

Putting the data into the computer

You now have all the data for your adventure written out on pieces of paper. The next problem is to work out how to put it into the computer's memory.

The computer needs the data stored in such a way that it can get at each item quickly and update things as the player progresses through the game. To do this, you set up storage areas called "arrays" in the computer's memory. An array is like a set of pigeon holes or filing boxes. You give each array a name and each box in it a number, so the computer can find the box you want when you refer to it in your program.

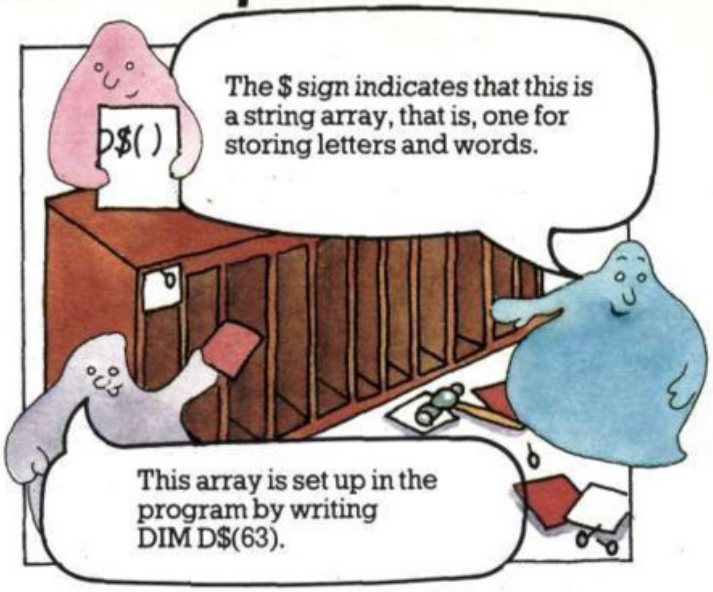


Before you can give the computer any data, you must decide how big each array should be and tell the computer to reserve and label that much space. This is called "dimensioning" the array and is written DIM in BASIC.

The arrays for *Haunted House*

Haunted House needs the following arrays to hold its data. You will need similar arrays whatever the theme of your adventure.

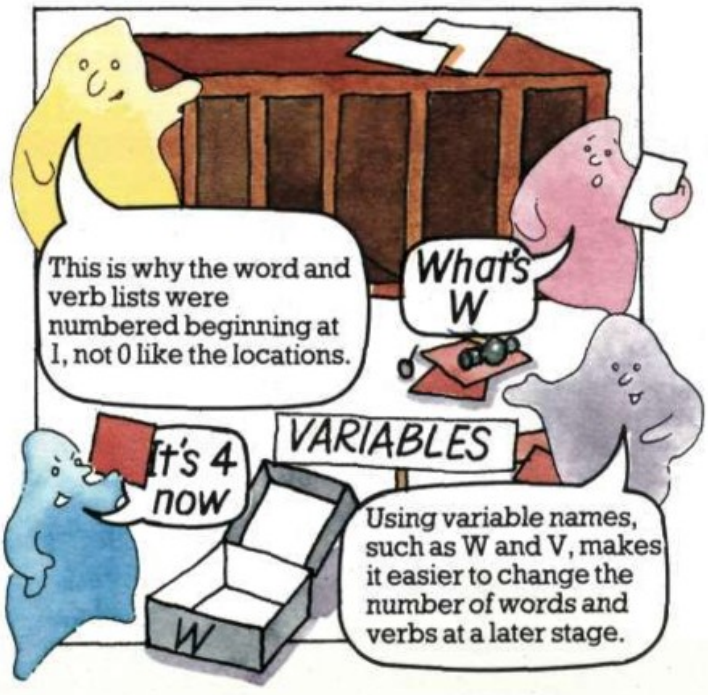
1. An array to hold the descriptions of the locations. It needs 64 pigeon holes (one for each location). We've called it D$( ) and numbered the holes 0 to 63 as on the master plan.



2. An array to hold the information about the routes the player can take from one location to another. This is R$( ). It needs to be the same size and numbered in the same way as D$( ).

3. An array for the objects and other words on the word list. By dimensioning this DIM O$(W), where W is the number of words on your list, the computer will set up an array with one space for each word and an extra space. This is because it always starts numbering with zero and ends with the number in the DIM statement. This is useful, because the zero space can be used for "word not found in memory".

e.g. If W=4, the array would look like this. DIM O$(4) gives five spaces labelled 0 to 4.



4. A verb array. This needs a space for each verb and an extra space for "verb not found". It is called V$( ) and needs to be dimensioned DIM V$(V) where V is the number of verbs on your list.

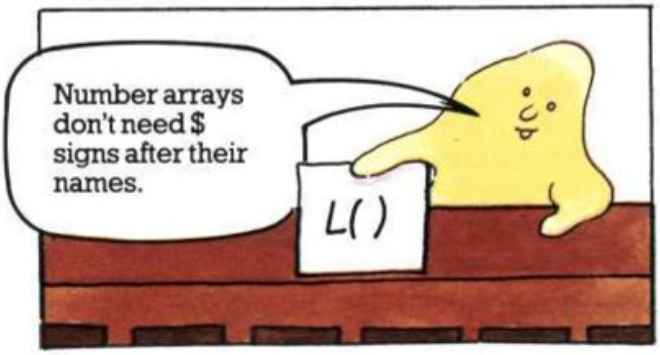
More arrays

Locations, routes, object words and verbs are not the only information that needs to be stored in the computer. You also need arrays to store information about where the objects are, which objects the player is carrying and such things as whether the light is on or off.

There is no need to store the object and location words again. This extra information can be stored as numbers to save space, e.g. object 9 is in location 10.

Array L( ) shows which location each object is in. It only needs spaces for the "gettable" objects such as the key, not the props or other words. If G is the number of gettable objects then this array is dimensioned DIM L(G).

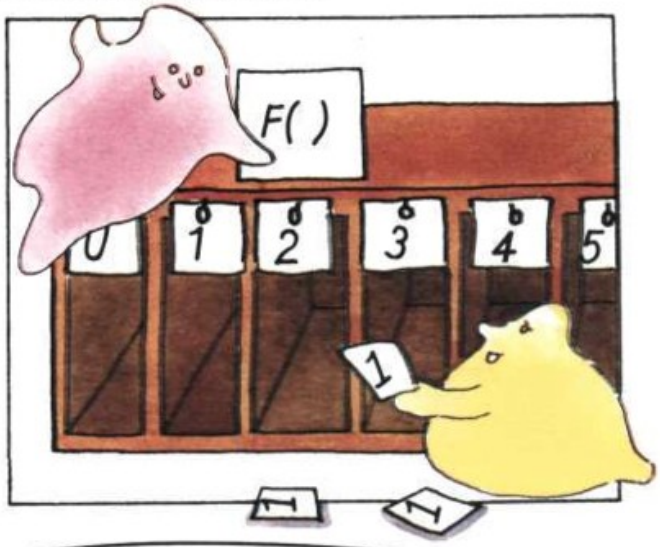
Array C( ) is for information about which objects the player is carrying. This also needs spaces only for the gettable objects, so is dimensioned DIM C(G).

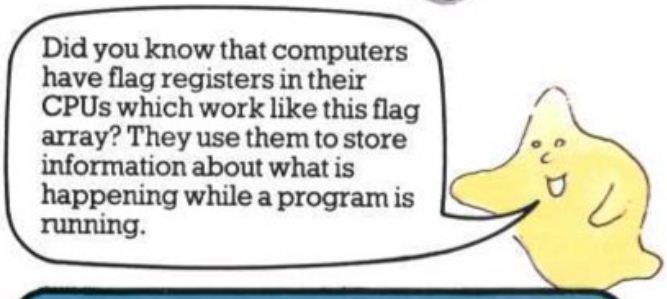


Flags

As well as keeping track of the things the player is carrying, the computer needs to be able to record other changes that happen during the game, e.g. whether the candle is alight, the door locked or the key visible.

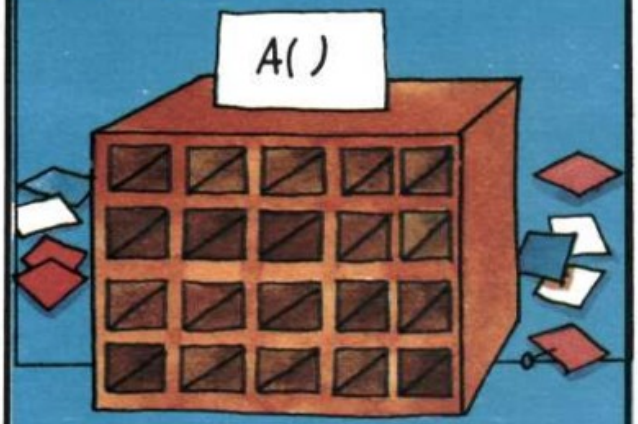
This can be done by using an array, F( ), of markers or "flags", which contains W spaces, i.e. one for each object word. By putting 1s and 0s in these spaces, the computer can see what state the object is in. 0 is used for the "normal" or "inactive" state, such as light off, object visible. 1 shows the "active" or "not normal" state, such as light on and object invisible.





Why not use 2D arrays?

If you have come across arrays before, you may have wondered why single dimension arrays are used for the *Haunted House* descriptions and routes instead of two-dimensional arrays, which would look like this:

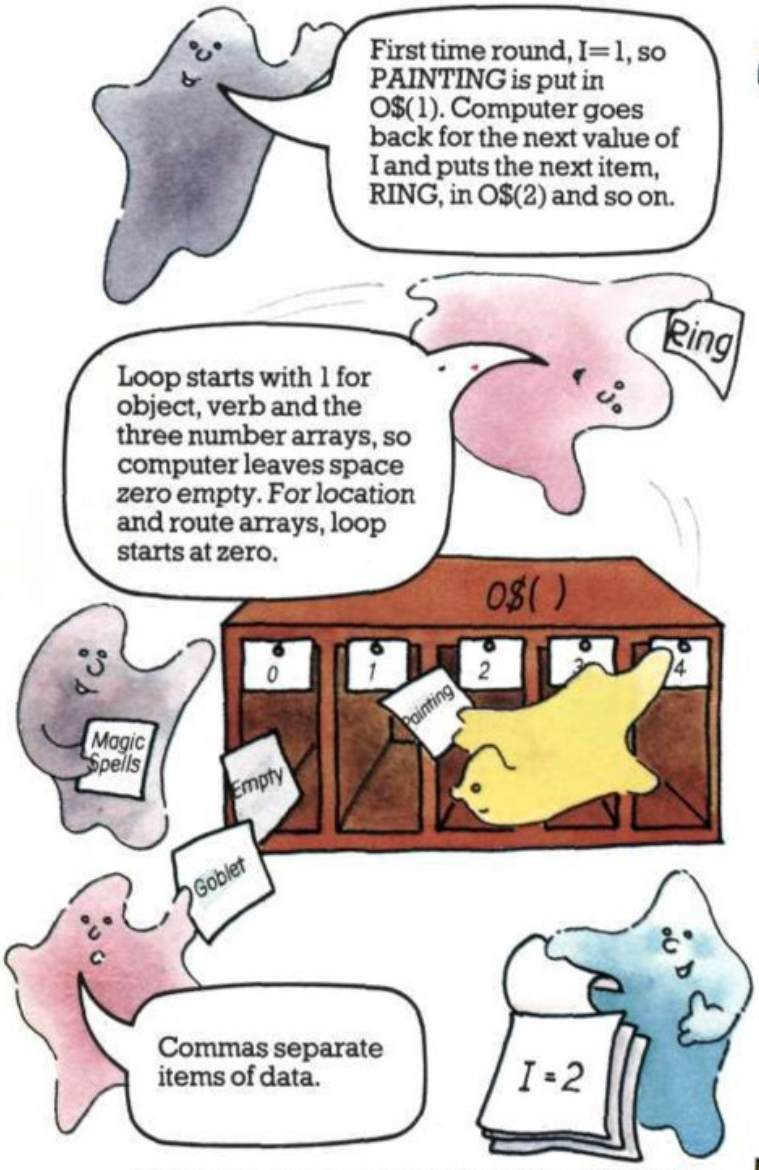


The reason is that single dimension arrays use slightly less memory space. You could use 2D arrays if you wanted to, in which case you would dimension them DIM D$(8,8) and DIM R$(8,8).

Putting the data into the arrays

Having set up labelled storage areas in the computer's memory, you need to tell it what to put in them. One way of doing this is to list the data, in order, and tell the computer to loop round putting one item at a time in the spaces in an array.* Here is how this is written in BASIC:

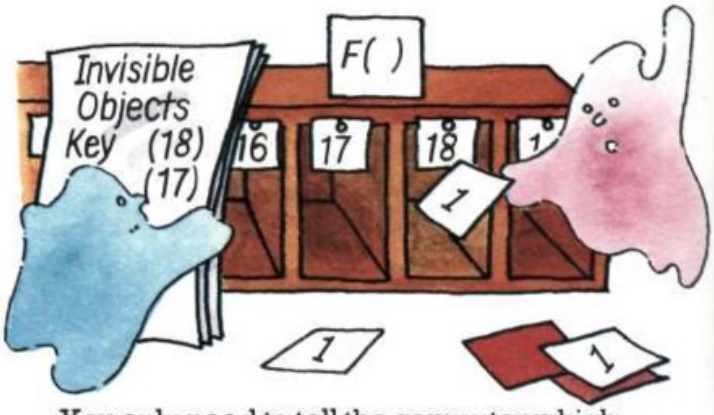
```
DIM O$(W)
DATA PAINTING,RING,MAGIC SPELLS,GOBLET,ETC.
FOR I=1 TO W
READ O$(I)
NEXT I
```



This is the loop for READing the DATA into O$( ). Look at lines 1600 to 2100 in the program listing on pages 36 and 37 and see if you can pick out the data loops for the other arrays.

Data for the flags

The data for the flag array, F( ), consists only of 1s and zeros. Objects which are invisible at the start of the game have 1 in their box in F( ). When they are discovered by the player the flag changes to zero. All the other objects start with zero.



You only need to tell the computer which boxes in F( ) need 1s in them. Leaving the rest empty is the same as filling them with zeros. The easiest way to fill this array is as shown in line 2090 (on page 37).

You may have noticed that some spaces in F( ) are not used because some objects do not change their "state". These spare flags can be used for other things. For instance, in *Haunted House*, F(14) (the rope flag) is used to show whether the player is up the tree. The candle needs two flags – one to show if it is visible and another to show if it is lit. The spare flag F(0) is used for lighting it. If you want a spare flag for something, use the ones for words that won't need them, like "north".

Data for the carrying array

The player isn't carrying anything at the beginning of the game, so to show this the array C( ) is left empty. When an object is picked up the computer puts a 1 into its box. So, no data lines are needed for array C( ).

Initialization

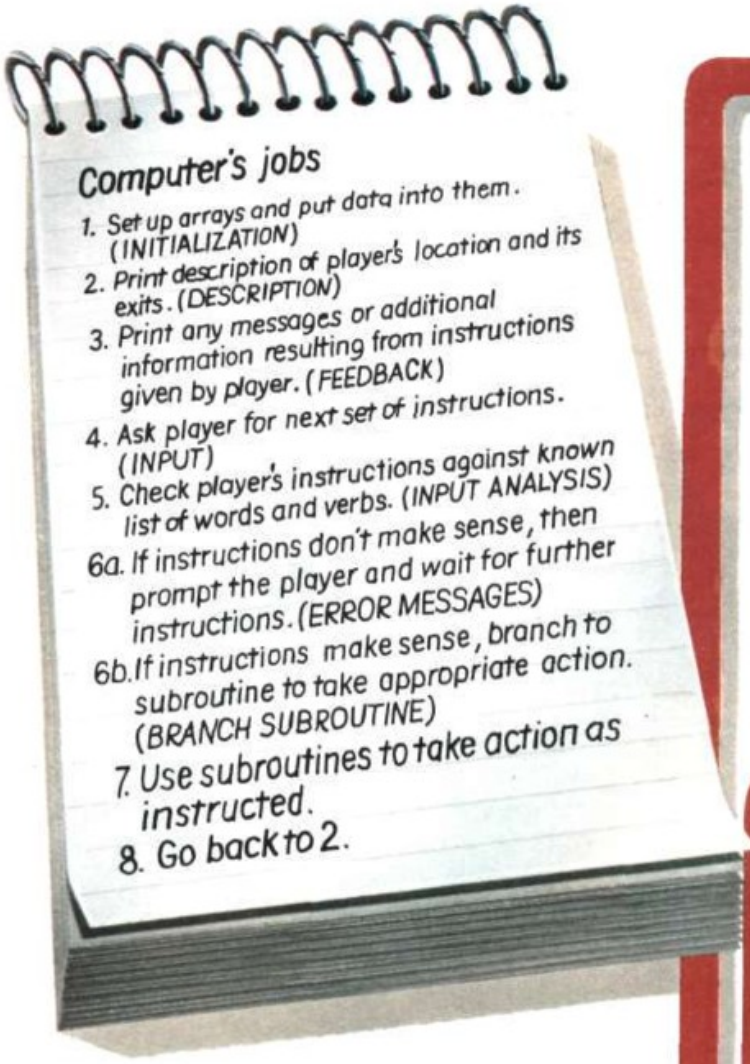
Setting up the arrays and filling them with data is called "initialization". You can see in the next section where this fits into the program structure.

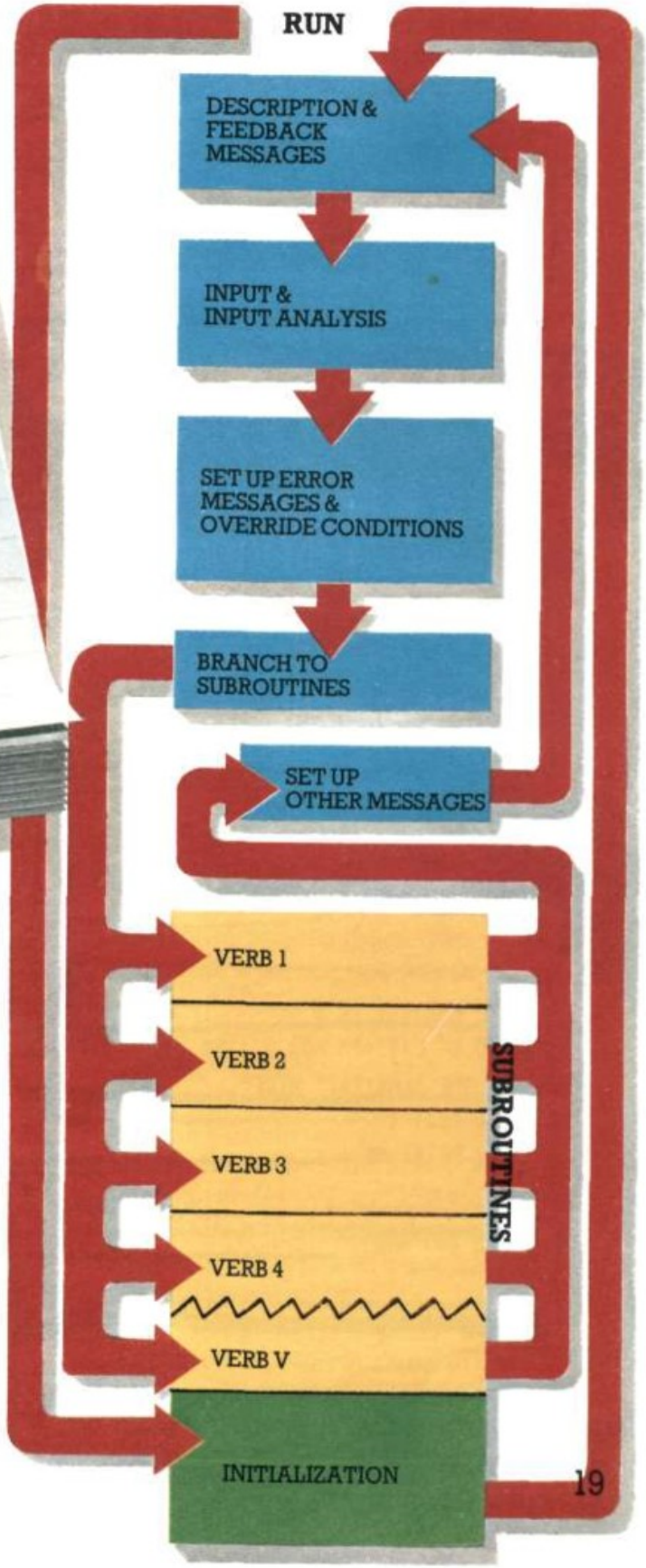
**The ZX81 (Timex 1000) does not do this, see page 39.*

The program structure

In order to arrive at the overall structure of the adventure program, you need to think about the jobs the computer has to do during the game.

The structure of the program actually looks like this. As you can see, the biggest part of it is the subroutine section. There needs to be one subroutine for each verb used in the game. You will find out more about these on page 25.





The list above shows the order in which the computer needs to do things, but not necessarily the order in which they need appear in the program. A large chunk of the program is the initialization routine which is only needed once each game, and, although it is the first thing the computer must do, it is a good idea to put it at the end of the program. This is because every time the computer is told to GOTO or GOSUB it goes back to the beginning of the program and checks through each line number until it finds the one it wants. This can take a noticeable amount of time in a long program. By putting initialization at the end, the computer does not have to check through it each time the player makes a move.

Getting into the program

Now you have an overall idea of what the program will be like, you can start thinking about each part in more detail. You have already seen on pages 16 to 18 how the initialization section works. The next eight pages describe how the other main parts of the program work.

Description and feedback

Every go, the computer must tell the player where he is and the directions in which he can move. It must also tell the player what happened as a result of his last instructions. This is the description and feedback section and it looks something like this. See if you can identify each part in the program listing on pages 35 to 37.

```
90 PRINT "TITLE OF GAME"
100 PRINT "-------------"
110 PRINT "YOUR LOCATION:"
120 PRINT D$(RM)
```

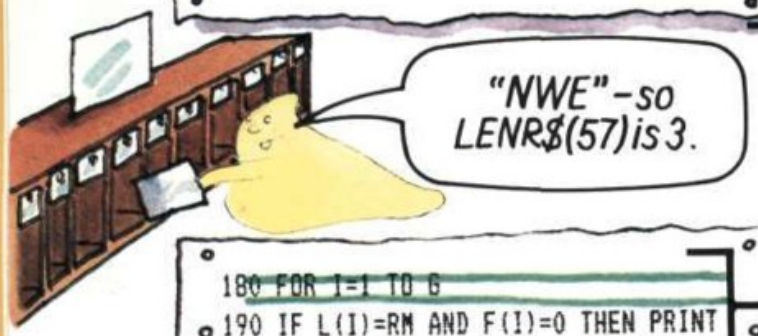
RM is the number of the location the player is in. You must remember to set a starting value for this in the initialization routine. (For *Haunted House* 57 is the starting value for RM, see line 2090).



The computer looks in D$ (the array containing all the descriptions) and prints what it finds in the box with the value RM.

```
130 PRINT "EXITS:";
140 FOR I=1 TO LEN(R$(RM))
150 PRINT MID$(R$(RM),I,1);",";
160 NEXT I
```

Looks at the length of the string in box RM in the routes array, R$. The computer then loops round this number of times, printing out each character in R$(RM) in turn, putting a comma and a space between each one.




```
180 FOR I=1 TO 6
190 IF L(I)=RM AND F(I)=0 THEN PRINT
"I SEE ";O$(I);" HERE"
200 NEXT I
220 PRINT M$

225 M$="WHAT?"
```

Loops round to see if there is an object with a zero flag in the location (i.e. a visible object) and prints out its name if there is.

M$ is a variable used to contain messages the computer has for the player, as a result of instructions given in the previous go. Look for M$ in the program listing on pages 33 to 37 and see how different messages are put into M$ depending on what the player typed.

At the beginning of each go, M$ is set to "WHAT?", so if there is no new message to replace this, the computer just prints "WHAT?"



The input section

An important feature of adventure games is the way the computer responds to instructions typed into it by the player. *Haunted House*, like many adventures, limits the player to two-word sentences, plus a few special one-word commands such as HELP. The next section of your program must ask the player for instructions and then tell the computer what to do with them.

To start with the computer needs to split the player's input into two words which it can then check against the words it has in its memory. The "word-splitter" routine used in *Haunted House* works by scanning the player's input until it finds a gap in the letters. It is listed below with a few extra lines so you can type it in by itself and see it working.*

```
100 CLS
110 PRINT "PLEASE TYPE SOMETHING"
120 INPUT Q$
130 V$=""
140 W$=""
```

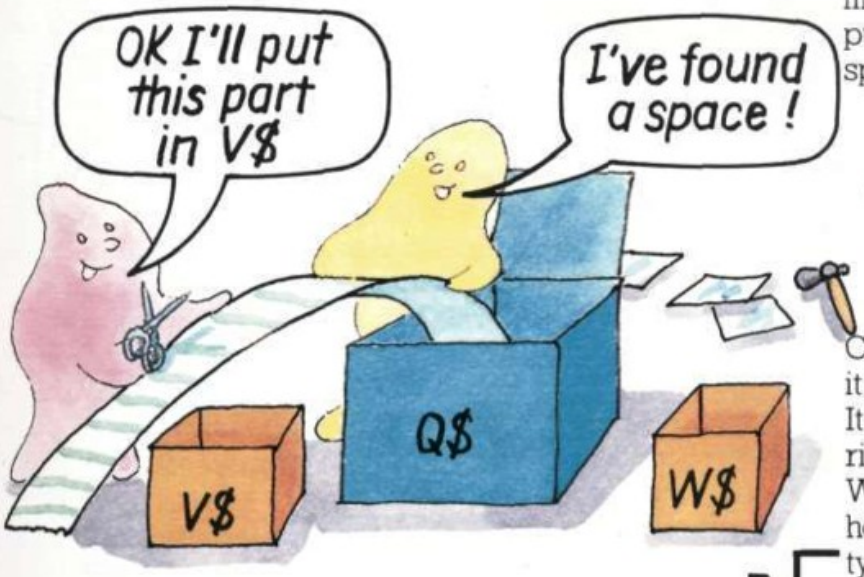
Gets player's instruction and puts it in Q$. Sets up two new string variables: V$ and W$.

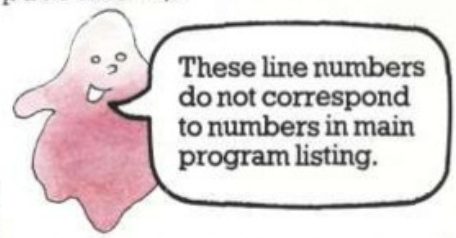
```
150 FOR I=1 TO LEN(Q$)
```

Looks to see how many characters there are in Q$ and starts a loop which goes round this many times.

```
160 IF MID$(Q$,I,1)=" " AND V$=
"" THEN V$=LEFT$(Q$,I-1)
```

Looks through Q$ for a space. If it finds one and V$ is still empty, it puts all the letters to the left of the space into V$.






```
170 IF MID$(Q$,I+1,1)<>" " AND V$<>"" THEN W$=MID$
(Q$,I+1,LEN(Q$)-1):I=LEN(Q$)
```

Continues to look through Q$ until it finds a letter following a space. It then takes everything to the right of this space and puts it in W$. (This means it doesn't matter how many spaces the player types between his two words.)

When V$ and W$ are both filled, the loop counter is set to its maximum value to end the loop.

```
180 NEXT I
190 IF W$="" THEN V$=Q$
```

If the computer didn't find a gap in the letters then V$ and W$ will still be empty when the loop has finished. It then takes the whole of Q$ and puts it in V$.

```
200 M$="THESE ARE YOUR 2 WORDS"
210 PRINT "FIRST WORD=";V$
220 PRINT "SECOND WORD=";W$
230 IF W$="" THEN M$="YOU ONLY TYPED ONE WORD"
240 IF W$="" AND V$="" THEN M$="YOU DIDN'T TYPE
ANYTHING"
250 PRINT M$
260 STOP
```

This section is so you can run the word-splitter by itself. It prints out messages depending on what you typed in. Run the program and see what happens.

**NB This will not work on Sinclair (Timex) computers.* *See pages 38 or 39.*

Analysis of input

The computer now has the player's instructions stored in two strings V$ and W$, which stand for "verb string" and "word string". Its next job is to check these against the words you have given it in the initialization procedure on page 18. It assumes the word in V$ is a verb and loops round seeing if it matches any of the verbs in the array V$( ). (Note the difference between the string variable V$ and the array V$( ) – they are completely different things to the computer, so make sure you don't confuse them.)

The computer then loops round in the same way trying to match up W$ with one of the words in the array O$( ).

Here is the section of program which checks for a match between the player's words and the words in the computer's memory.

```
10 VB=0
```

VB is a new variable set up to record the box number containing the matching verb. If the player typed GET, for example, the computer looks through V$( ) until it reaches box 10 where GET is stored. So, in this case, VB=10.




```
20 FOR I=1 TO V
30 IF V$=V$(I) THEN VB=I
40 NEXT I
```

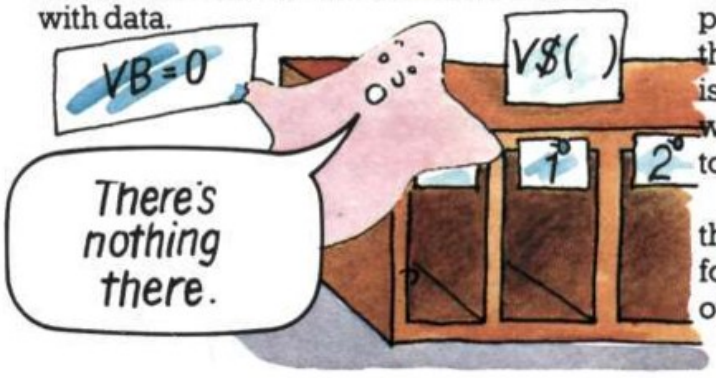
The computer loops round V times (V is the number of verbs in the computer's memory), comparing the player's verb with each of those in its memory. If it finds one that matches, it sets VB to the appropriate number.

```
50 OB=0
60 FOR I=1 TO W
70 IF W$=O$(I) THEN OB=I
80 NEXT I
```

The loop for W$ works in the same way, using OB to record the box number of the matching word.

What if the words don't match?

If no match was found, then VB and OB will still be zero. The computer takes this as meaning that box zero in the array contains the match for the player's word. But when it looks there to find out what the matching word is it doesn't find anything because you left this box empty when filling the arrays with data.



Silly combinations

Notice that this matching-up process only checks if the two words are in the computer's memory. It doesn't check to see if the combination of words makes sense. A silly combination such as UNLOCK CANDLE gets through this stage of the program, but will be rejected later on when the computer tries to carry out the action. It is much quicker just to check the separate words at this stage than to tell the computer to check for valid combinations.

At the end of this section of the program, the computer has a value for VB and a value for OB. You can see what it does with these over the next page.

Setting up error messages

The computer can use the values of VB and OB to see if the player needs to be sent a message saying his instructions are no good. This part of the program is like a filter or grader. The player's instructions are fed through a series of tests. If they don't pass one of the tests, a new message is put in M$. If they pass through all the tests, M$ still contains the message "WHAT?" which was set up in line 220. (Remember that, at this stage, the messages are just set up, they are not printed on the screen and may be changed again later in the program anyway.)

Here are the program lines which set up the error messages in *Haunted House* – see if you can find them in the main listing. You will need similar lines if you are writing your own adventure.



1

```
IF W$>"" AND OB=0 THEN M$="THAT'S SILLY"
```



The first test looks to see if there is a word in W$ (i.e. that the player typed two words) and then checks if the value for OB is zero.

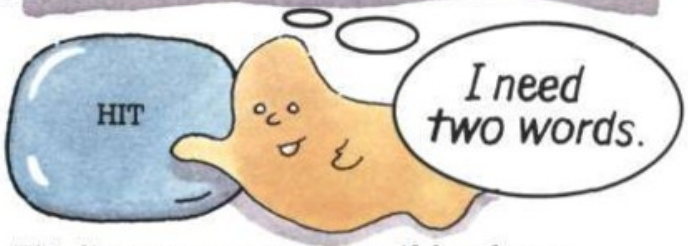
2

```
IF VB=0 THEN VB=V+1
```

This line is to overcome a problem. You cannot GOSUB on a value of zero. But different BASICs vary in the way they cope with being asked to do this. Most of them ignore the GOSUB and carry on to the next line. Some, however, such as the BBC, object to the zero and produce an "on range" error. To get round this, VB is changed from zero to a value greater than V (the number of verbs in the computer's memory), and the computer is sent to a "dummy" subroutine.

3

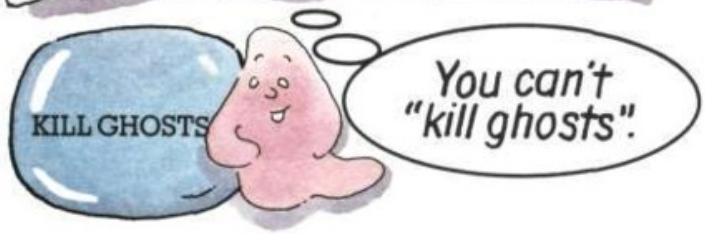
```
IF W$="" THEN M$="I NEED TWO WORDS"
```



This line sets up a message if the player only typed one word and so W$ is empty. (If the word is one of the allowed one-word commands, this message will be changed later in the program.)

4

```
IF VB>V AND OB>0 THEN M$="YOU CAN'T '
"+O$+"'"
```



This line sets up a message if the computer doesn't have the verb in its memory, but does have the object.

5

```
IF VB>V AND OB=0 THEN M$="YOU DON'T
MAKE SENSE"
```



If the computer doesn't have either of the player's words in its memory it sets up this message.

6

```
IF VB<V AND OB>0 AND C(OB)<>1 THEN M$=
"YOU DON'T HAVE "O$(OB)
```



Override conditions

Sometimes things happen in an adventure which prevent the player from doing anything until he has dealt with them. In these circumstances, instructions which would normally be valid need to be overridden, so the computer needs program lines which set flags* in its memory to tell it that special conditions apply.

In *Haunted House*, lines 420 to 450 are override conditions. You can see them on the right, with an explanation of how they work.

```
420 IF F(26)=1 AND RM=13 AND RND(3)<>3
AND VB<>21 THEN M$="BATS ATTACKING!":
GOTO 90
430 IF RM=44 AND RND(2)=1 AND F(24)<>1
THEN F(27)=1
440 IF F(0)=1 THEN LL=LL-1
450 IF LL<1 THEN F(0)=0
```

Line 420
If bats are present, player is in Rear Turret Room, random number is not 3 and player hasn't used verb 21 (SPRAY) in his instructions, then M$ is set to "BATS ATTACKING" and player cannot go any further in the game.

Line 430
If player is in Cobwebby Room, random number value is 1 and vacuum cleaner is switched off, then flag is set for paralysing ghosts to appear, i.e. F(27) is set to 1.

Line 440
If candle is lit, then light limit counter, LL, is decreased.

Line 450
If LL is zero, then candle on/off flag, F(0), is set to zero.

Perhaps you can think of other override conditions which could be added here.



Branch to subroutines

The computer's next task is to attempt to carry out what the player wants to do. If it had to search through every possible action until it found the one the player wanted each time, the game would be very slow and boring. To avoid this, you use lots of subroutines – one for almost every verb on the verb list. (A few, such as GET and TAKE, can share the same one.)

You can then use an ON . . . GOSUB line to tell the computer to branch to a different subroutine depending on the value of VB.

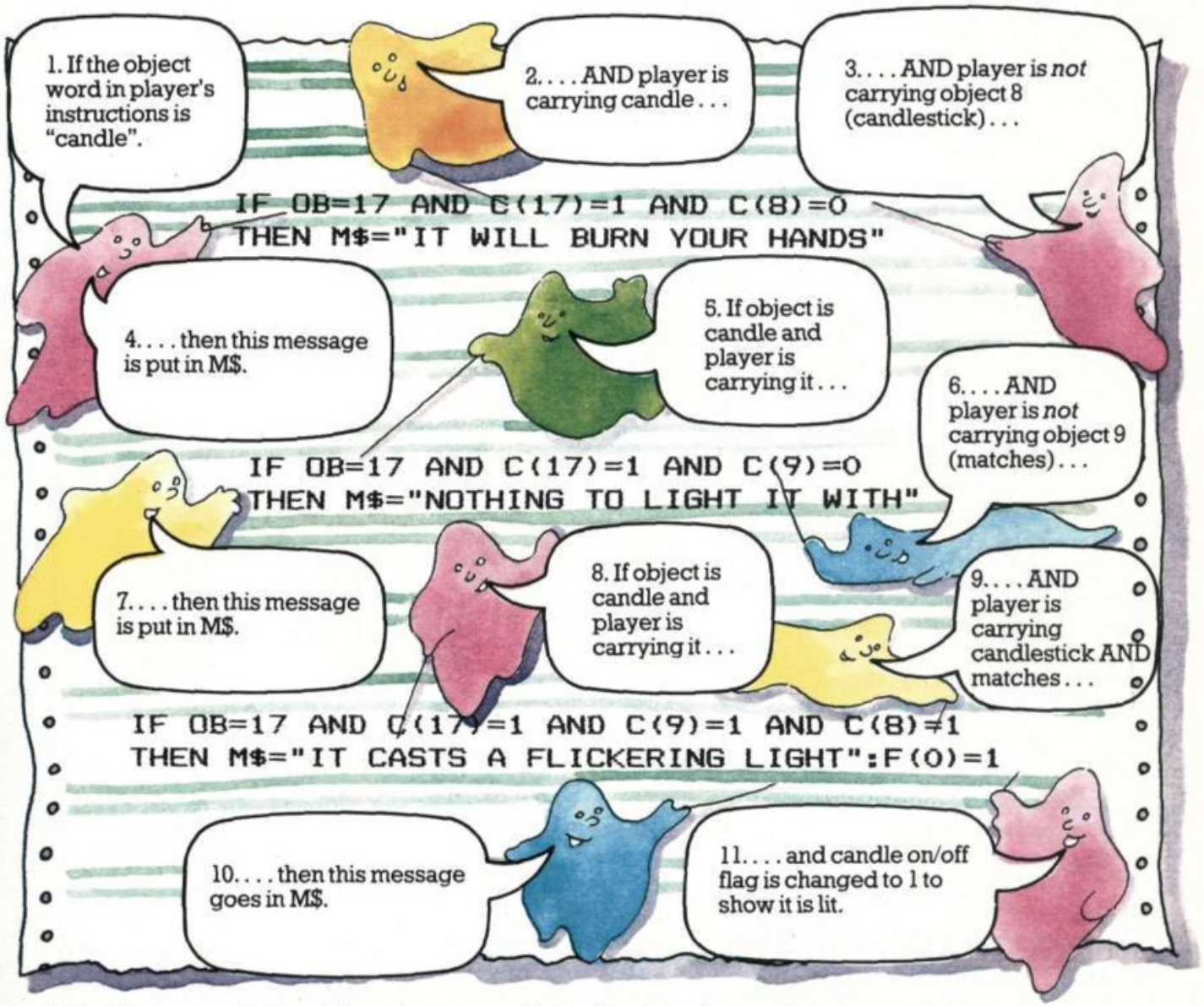
```
ON VB GOSUB 500,570,640,640,640,640,
640,640,640,980,980,1030,1070,1140,
1180,1220,1250,1300,1340,1380,1400,
1430,1460,1490,1510,1590
```


*See page 18 for more about flags.

How the ON . . . GOSUB line works

The ON . . . GOSUB line on the opposite page works like this. If VB=1 the computer goes to the first line number listed, if VB=2 it goes to the second, if VB=3 it goes to the third and so on. Notice that the last line number listed is a "dummy" subroutine for VB=V+1 (the value of VB when no matching verb was found in the computer's memory). The line it is sent to just says RETURN and so sends the computer straight back up the program again.

Look at the subroutines on pages 34 to 36 and see if you can work out what they all do. Here is the procedure for LIGHT (VB=19) as an example. You will find it at lines 1340-1370.



What happens if the object is not candle?

If the object the player wanted to use was not CANDLE, but one of the others in the computer's memory, such as DOOR, then the message in M$ is unchanged from when it was set up in line 220. When the computer returns to the main program and finds the instruction PRINT M$ it will print the message "WHAT?".

Notice that there is no need to set up a message saying the candle isn't there, as this is already covered in the error messages section.

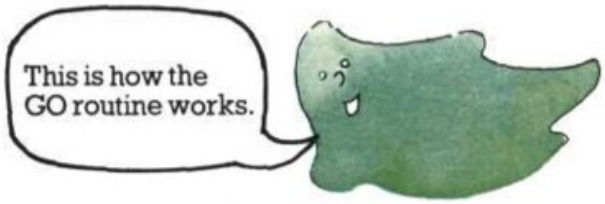
Back to the main program

Although some of the verb routines are longer and more complicated than this one, they all work in a similar way: the value of OB is checked, a special message is set up if necessary and then the computer returns to the main program. It checks the light limit at lines 470 and 480 and is then sent back to the description and feedback section. Here it prints out the message it has put in M$ and waits for the next set of instructions from the player.

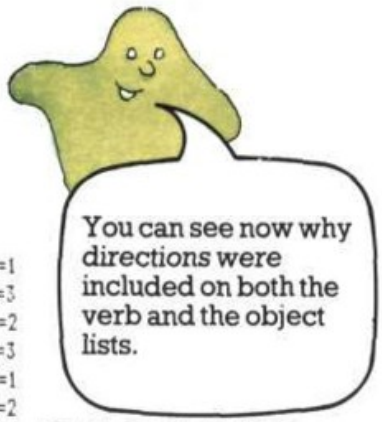
The GO subroutine

The subroutine for the verb GO is so large and important in an adventure game you could almost think of it as a sub-program. Seven verb commands are directed to it – GO, N, S, W, E, U and D. This routine is also special because it responds to single-letter direction commands as well as two-word ones. You don't have to include this facility in your program, but it does help make the game quicker and more interesting to play. If you've played many adventures you will realize how tedious it is to have to type GO NORTH etc. every time.

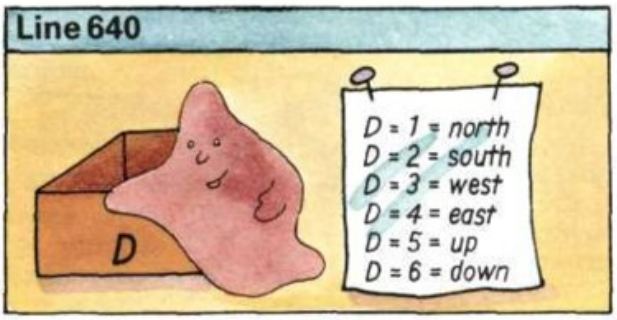
This is how the GO routine works.




```
640 D=0
650 IF OB=0 THEN D=VB-3
660 IF OB=19 THEN D=1
670 IF OB=20 THEN D=2
680 IF OB=21 THEN D=3
690 IF OB=22 THEN D=4
700 IF OB=23 THEN D=5
710 IF OB=24 THEN D=6
720 IF RM=20 AND D=5 THEN D=1
730 IF RM=20 AND D=6 THEN D=3
740 IF RM=22 AND D=6 THEN D=2
750 IF RM=22 AND D=5 THEN D=3
760 IF RM=36 AND D=6 THEN D=1
770 IF RM=36 AND D=5 THEN D=2
780 IF F(14)=1 THEN M$="CRASH! YOU FELL OUT OF THE TREE!"
:F(14)=0:RETURN
790 IF F(27)=1 AND RM=52 THEN M$="GHOSTS WILL NOT LET YOU
MOVE":RETURN
800 IF RM=45 AND C(1)=1 AND F(34)=0 THEN M$="A MAGICAL
BARRIER TO THE WEST":RETURN
810 IF (RM=26 AND F(0)=0) AND (D=1 OR D=4) THEN M$="YOU
NEED A LIGHT":RETURN
820 IF RM=54 AND C(15) <>1 THEN M$="YOU'RE STUCK!":RETURN
830 IF C(15)=1 AND NOT (RM=53 OR RM=54 OR RM=55 OR RM=47)
THEN M$="YOU CAN'T CARRY A BOA
840 IF (RM>26 AND RM<30) AND F(0)=0 THEN M$="TOO DARK TO
MOVE":RETURN
```

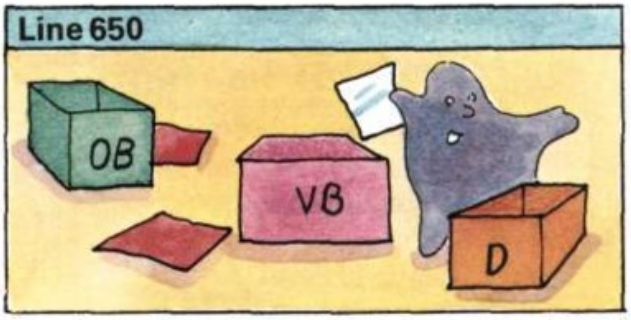


Line 640



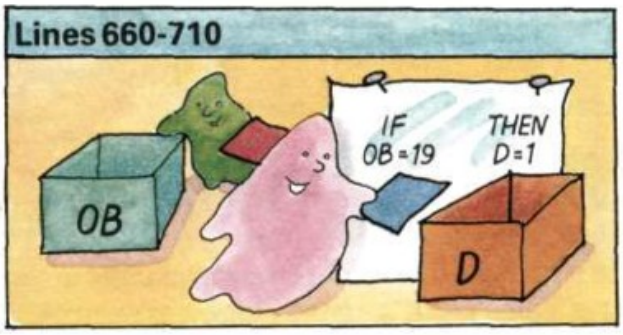
First, a variable D is set up to hold information about the direction in which the player wants to move. Its values 1 to 6 correspond to north, south, west, east, up and down.

Line 650



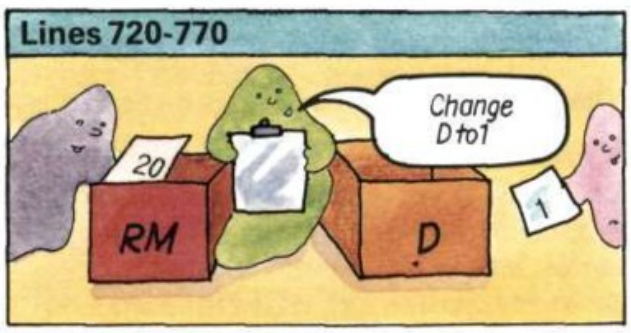
The next line checks to see if the player just typed one word and then gives D a value depending on the value of VB. (Notice that by taking 3 away from VB, the computer gets values for D which correspond to those in lines 660 to 710.)

Lines 660-710



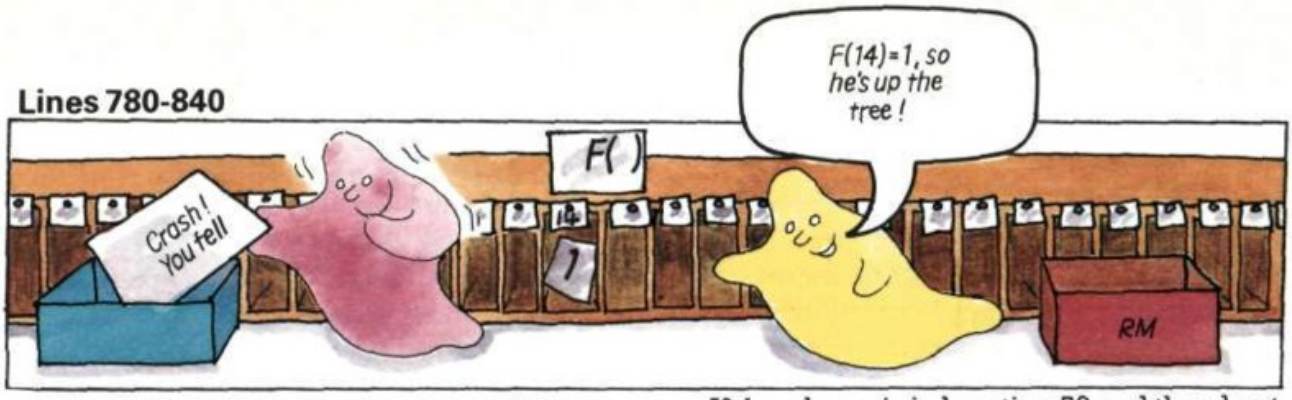
The next six lines check if the player typed a two-word direction instruction. They use the value of OB to set the value of D.

Lines 720-770



As this is not really a 3D adventure, the UP and DOWN instructions need to be converted into north, south, east or west. Lines 720 to 770 do this. (If you check back to the master plan on pages 14 to 15, you will see that this does work.)

Lines 780-840



The computer also needs to check if there are any special conditions which affect the player's ability to move. For example if F(14)=1, then the player is at the top of the tree. If he tries to move without first climbing down, he gets a message saying he has fallen.

If the player is in location 52 and the ghost flag is "on" then a message is sent to say he cannot move. Each of these conditions returns the computer to the main program. See if you can work out what the rest of the lines in this section do.

Checking for walls

If the move has not been stopped by any of these special conditions, the computer must check that there isn't a wall or anything else blocking the way. Here are the lines which do this. They look quite complicated at first sight, but if you look carefully at each part, remembering what all the variables are, you should be able to see what is happening.

```
850 F(35)=0:RL=LEN(R$(RM))
860 FOR I=1 TO RL
870 U$=MID$(R$(RM),I,1)
880 IF (U$="N" AND D=1 AND F(35)=0:
    THEN RM=RM-8:F(35)=1
890 IF (U$="S" AND D=2 AND F(35)=0:
    THEN RM=RM+8:F(35)=1
900 IF (U$="W" AND D=3 AND F(35)=0:
    THEN RM=RM-1:F(35)=1
910 IF (U$="E" AND D=4 AND F(35)=0:
    THEN RM=RM+1:F(35)=1
920 NEXT I
930 M$="OK"
940 IF F(35)=0 THEN M$="CAN'T GO
    THAT WAY"
950 IF D<1 THEN M$="GO WHERE?"
960 IF RM=41 AND F(23)=1
    THEN R$(49)="SW":M$-"THE DOOR
    SLAMS SHUT!":F(23)=0
970 RETURN
```

This is a flag for the computer to use to register whether it has found the exit the player wants.

RL is a new variable which holds the length of the string which it finds in R$(RM). (This string is the routes, NSW etc., for the location the player is in.)

Computer loops round RL times.

Each loop, computer takes one of the characters in R$(RM) and temporarily calls it U$.

It then runs a series of tests on U$ and D. If the player's direction instruction matches an exit in the location he is in, then the value of RM is changed to move him to the appropriate place. F(35) is then set to 1 to stop the computer trying to change RM *again* on another trip through the loop. (If you think carefully, you will see that this could be possible as the computer uses its new value of RM in line 870.)

If you check the master plan, you will see how adding or subtracting 1 or 8 moves the player to the correct next location.

At the end of the loop, M$ is set to "OK". This will replace the "I NEED TWO WORDS" message set in the error messages if the player typed a one-word direction.

If F(35) is still zero, then the direction the player wants to go is not allowed and M$ is changed to say so

If D is less than one (i.e. it wasn't assigned a value in lines 650 to 770), then M$ is changed to "GO WHERE?"

This line makes the front door a "once-only" route. When the player enters location 41 (the lobby), the exits from location 49 (front porch) are changed from "NSW" to "SW", M$ is set to "THE DOOR SLAMS SHUT" and the flag for the front door is set to zero to show the computer it is now closed. (The routes from location 41 do not need to be changed because S wasn't included in them in the first place.)

Changing the program

You can change the program in this book as much as you like, either to produce variations on the haunted house theme or to create games with completely different settings, descriptions, objects, verbs and messages. Remember that the more you change, the more complicated it will get as you will have to think about how everything affects everything else.

If you are going to write a new game, using this program as a guide, then you should plan it as described on pages 6 to 15. It is worth spending the time planning out your game properly as you are less likely to find it full of mistakes when you come to run the program.

It is a good idea to start by making small changes first to see what happens. If you store the master program on tape, you can make changes, test them and adapt them without losing the original.

How much spare memory have you got?



The *Haunted House* program itself occupies about 7K of RAM before it is run. It then needs a further 3½ to 4K for the arrays to store the data. Your computer will take some memory for its own internal use – up to 3K on some models – and it will use a further 1K or more for the screen. (The Spectrum uses 7K which is why *Haunted House* won't fit into the 16K model.) So, if you have a 16K computer, you won't have much memory left over, and most of the changes you make will have to be replacements rather than additions.

Longer descriptions

If you have more than 16K, one of the easiest ways of making the game more interesting to play is to add longer descriptions. Instead of "impressive vaulted hall" for instance, you could say something like "You have entered a vast, vaulted chamber with pillars extending many times your height above you. Light filters in from the east and there appears to be a doorway in the distance to the west . . ."

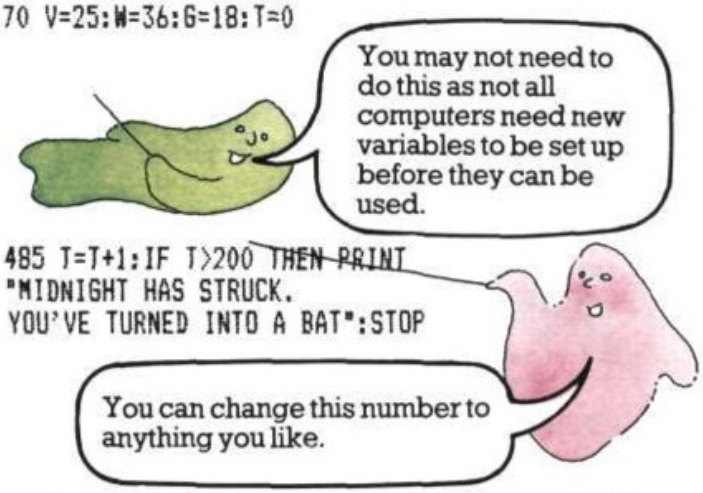


Add a time limit

Haunted House already has a time limit on the life of the candle. You could add an overall time limit to the game as well by getting the computer to count the number of turns the player has had and stop the game at a preset number.

You can do this by adding to line 70 and putting an extra line at 485 like this.

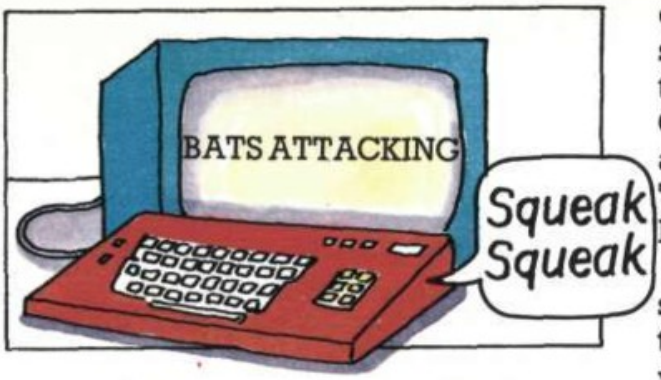
```
70 V=25:W=36:G=18:T=0
```



Puzzle

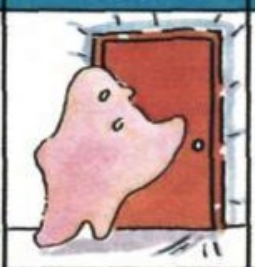
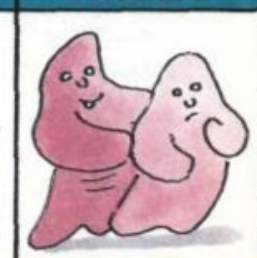
Can you think how to put a limit on the number of objects that can be carried at any one time? (You will have to adjust the scoring routine as well.)

Adding sounds



This is an effective way of adding to the game without having to make complicated changes. You will need to know how your computer's sound instructions work. Test some sound routines out first to make sure they are what you want and then add a GOSUB instruction to the line where the action occurs, eg IF RM=46 AND C(1)=1 THEN M$="SOMETHING SCARY IS HAPPENING": GOSUB 6000

You could add sounds for the front door slamming, the secret wall breaking, the key turning in the door, magic happening when you use the magic word and so on. This chart gives a few sound routines for various computers. The only limit on the number you can add is memory.

	FALL OUT OF TREE	DOOR SLAM	MAGIC	AXE BLOWS	GENERAL PROMPT
					
VIC 20	POKE 36877,130 FOR L=15 TO 0 STEP -1 POKE 36878,L FOR M=1 TO 20:NEXT M NEXT L POKE 36877,0	POKE 36877,130 FOR L=15 TO 0 STEP -1 POKE 36878,L FOR M=1 TO 2:NEXT M NEXT L POKE 36877,0	POKE 36878,15 FOR I=160 TO 240 STEP 5 POKE 36876,I FOR M=1 TO 100:NEXT M POKE 36876,0	POKE 36878,15 FOR I=1 TO 10 POKE 36877,200 POKE 36877,0 FOR M=1 TO 400:NEXT M NEXT I	POKE 36878,15 FOR I=1 TO 2 POKE 36876,200 POKE 36876,0 FOR M=1 TO 400:NEXT M NEXT I
SPECTRUM	—	—	FOR I=5 TO 40 STEP 3 BEEP 0.2,I NEXT I	FOR I=1 TO 10 BEEP 0.01,0.01 FOR M=1 TO 100:NEXT M NEXT I	BEEP 0.5,5 PAUSE 50 BEEP 0.5,5
BBC	FOR L=-15 TO 0 SOUND 0,L,5,1 NEXT L	FOR L=-15 TO -8 SOUND 0,L,5,0.6 NEXT L	FOR I=40 TO 160 STEP 5 SOUND 2,-15,I,5 NEXT I	FOR I=1 TO 10 SOUND 0,-15,5,1 FOR M=1 TO 1000:NEXT M NEXT I	SOUND 2,-15,100,2 FOR M=1 TO 1000:NEXT M SOUND 2,-15,100,2
DRAGON/ TRS-COLOR	—	—	FOR I=50 TO 230 STEP 10 SOUND I,2 NEXT I	—	SOUND 180,1 FOR M=1 TO 500:NEXT M SOUND 180,1
ORIC	FOR L=15 TO 0 STEP -1 PLAY 0,2,6,100 SOUND 5,100,L WAIT 10 NEXT L PLAY 0,0,0,0	FOR L=15 TO 0 STEP -1 PLAY 0,2,6,100 SOUND 5,100,L WAIT 1 NEXT L PLAY 0,0,0,0	FOR L=15 TO 0 STEP -1 PLAY 0,2,6,100 SOUND 5,100,L WAIT 1 NEXT L PLAY 0,0,0,0	FOR I=1 TO 10 PLAY 0,1,1,20 SOUND 4,50,7 PLAY 0,0,0,0 FOR M=1 TO 300:NEXT M NEXT I	FOR I=1 TO 2 PLAY 1,0,6,100 SOUND 1,70,7 WAIT 20 PLAY 0,0,0,0 FOR M=1 TO 300:NEXT M NEXT I

Scoring

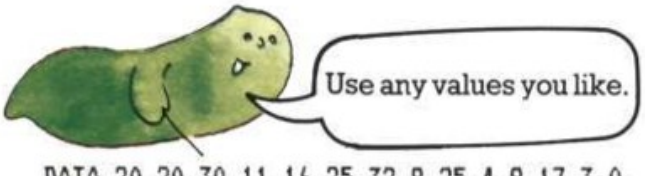
Haunted House has a very simple scoring system, awarding one point for each object the player is carrying. You could change to a more interesting system, such as basing the score on the value of the object. If you assume that the objects are numbered in descending order of value, then the painting will be the most valuable and the key the least. If you change line 1530 like this:

```
1530 IF C(I)=1 THEN S=S+G-I
```

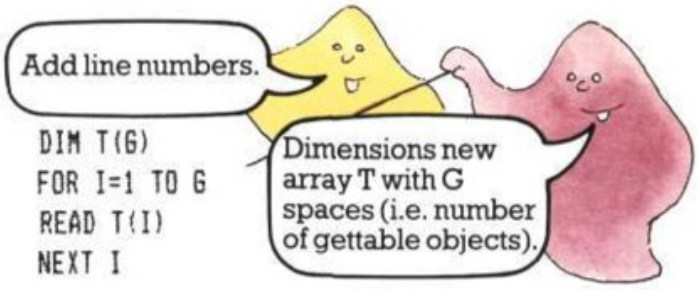
then the painting will be worth 18−1=17 and the key 18−18=0. (G is the number of gettable objects and I is the number of the object the player is carrying.) This makes the key valueless as an item of treasure but of great value as a useful object because without it, the player would not be able to get the painting or the goblet.



If you wanted a more flexible system (and you have enough memory), you could set up an array to contain object values in the initialization routine, like this:



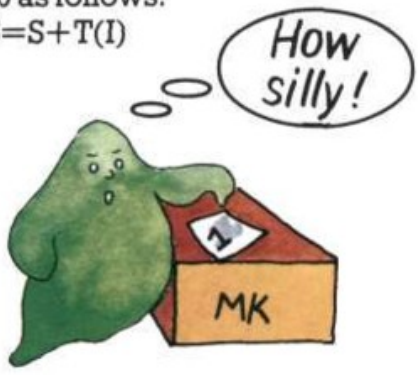

```
DATA 20,20,30,11,16,25,32,8,25,4,9,17,3,0,
10,12,4,9
```



Also change line 1530 as follows:

```
1530 IF C(I)=1 then S=S+T(I)
```

Penalties

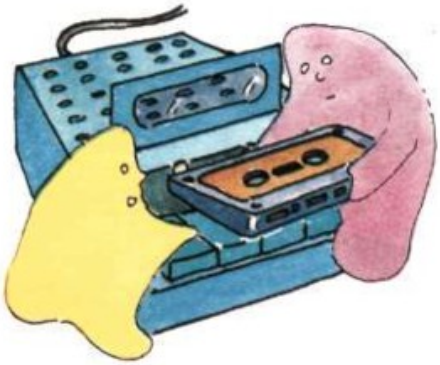


So far, the scoring routine has only counted plus points and not been affected by silly things the player might try to do. You could add a penalty system quite easily by using a counter, say MK, for mistakes. Whenever the player does something really silly, you add to MK and then subtract it from S when the score is worked out at line 1530. If the player falls out of the tree, for instance, you could award one (or more) penalty points like this:

```
780 IF F(14)=1 THEN M$="CRASH YOU FELL OUT
    OF THE TREE":F(14)=0:MK=MK+1:RETURN
```

Don't forget that some computers need new variables to be defined before they can be used. You can do this by adding MK=0 to the variables in line 70.

Saving the game



It would be nice to be able to switch off part way through a game and then carry on later from where you left off. With long, complicated games this is a very important feature and you can include it by adding SAVE and LOAD to the verb list. In line 70, change the value of V to 27 and add the two new verbs, separated by commas, to the end of line 1665. You will also need to change the ON GOSUB line at line 460.

Put the line numbers of the two new subroutines (one for SAVE and one for LOAD) between the last two numbers in line 460 so that they read:

. . . 1510,3000,4000,1590

First new number — Second new number

Then add the new subroutines like this, checking your computer's manual to make sure the wording is correct.

```
*3000 INPUT "IS YOUR CASSETTE
 READY TO RECORD";Y$
 3010 IF Y$<>"Y" THEN 3000
 3020 OPEN FILE FOR OUTPUT FROM
 COMPUTER

 3030 PRINT #1,RM
 3040 FOR I=1 TO G
 3050 PRINT #1,L(I)
 3060 NEXT I
 3070 FOR I=1 TO W
 3080 PRINT #1,C(I),F(I)
 3090 NEXT I
 3100 CLOSE
 3200 RETURN
*4000 INPUT "ARE YOU READY TO LOAD";Y$
 4010 IF Y$<>"Y" THEN 4000
 4020 OPEN FILE FOR INPUT TO COMPUTER
 4030 INPUT #1,RM
 4040 FOR I=1 TO G
 4050 INPUT #1,L(I)
 4060 NEXT I
 4070 FOR I=1 TO W
 4080 INPUT #1,C(I),F(I)
 4090 NEXT I
 4100 CLOSE
 4200 RETURN
```

Replace this line with your computer's own instructions. You may not even need a line here at all.

This saves the room the player is in.

This loop saves positions of gettable objects.

This saves items the player is carrying and the state of the flags.

Note that this save routine does not save the descriptions and routes in the game. This means that the rooms and routes altered by the player's actions will return to their original state – the secret wall will be rebuilt, the door relocked and so on. (The ghosts are probably responsible.) You could save the D$ and R$ arrays if you wanted to, by adding extra loops to each of the SAVE and LOAD routines.

Do you give in?

Like most adventures, *Haunted House* contains traps for the player which can only be avoided by using a certain object in a certain way. If the player doesn't have that object he is stuck. A "quit" feature would be useful in this situation so the player does not have to press BREAK or ESCAPE to end the game. You can do this by adding QUIT to the verb list and putting in a new subroutine, as for SAVE and LOAD described on the left.



You must remember to change the value of V in line 70, add QUIT to the end of line 1665 and insert the new subroutine line number in line 460, putting it in the second to last position.

The QUIT subroutine should be something like this:

```
5000 INPUT "WANT TO QUIT";Q$
5010 IF Q$<>"Y" THEN RETURN
5020 INPUT "LIKE TO SAVE GAME FIRST",Q$
5030 IF Q$="Y" THEN GOSUB 3000
5040 PRINT "THANKS FOR PLAYING"
5050 END
```

You don't need this if you haven't put the SAVE feature in.

Notice that there is no RETURN at the end of this subroutine. This is usually against the rules in BASIC but, in this case, the computer cannot get confused because the program will no longer be running when it reaches line 5050.

If you have a BBC, you may need to replace semi-colon with a comma.

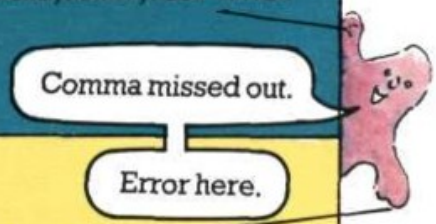
Debugging your adventure

If you write your own version of *Haunted House* or use the routines in it to make a new adventure, then you are quite likely to make mistakes. Finding mistakes and putting them right is called debugging. Here are some of the problems you might come across and some suggestions for fixing them.

Out of data

If the computer gives you an error code which stands for "out of data in line x" then it means that the numbers don't tie up in one of your data-reading sections. Check that the number of items of data is the same as the number in the loop for reading them in. You could have left a comma out in the DATA statement perhaps, or missed out one item altogether or put the wrong number in the loop.

```
DIM A(4)
FOR I=1 TO 4
READ A(I)
NEXT I
DATA SWORD,MONEY,FOOD WATER
```



Array error

If you get an array error, it means that you didn't reserve enough space when you DIMmed the array or you accidentally put an extra item in the DATA statement (perhaps by putting in an extra comma) and then counted this extra item when working out the number for the READ loop.

```
DIM F(3)
FOR I=1 TO 4
READ A(I)
NEXT I
DATA AXE,COFFIN,BLOOD,KEY
```

Objects behave in strange ways

This could happen because the program is being directed to the wrong subroutine by the ON GOSUB line. Check each number in this line against the subroutine with the same number. If these are all correct, check the DATA statement for the verbs to see if their order coincides with the order of the subroutines.

If the program *is* going to the correct verb subroutine and the verbs *are* listed in the correct order, check that there is a RETURN line at the end of each subroutine. If this is missing, the computer will "fall through" the program to the next subroutine down which may produce some strange results.

If none of the above things solves the problem then check through the conditions in the subroutine carefully. You might have missed something out or got a sign wrong or used the wrong variable by mistake. Check the override conditions and flags which occur earlier in the program too.

Exits in funny places

If you find a wall you can walk through or a doorway you can't, you may have made a mistake in planning your routes or in typing in the route data. Check your route map against the data lines to find the mistake.

Objects don't appear where they should

If an object appears in the wrong place then you've probably made a mistake in the data for array L. If an object doesn't appear at all, check the flag array. You must have set the flag with that object number to 1, which means that the object is there but the computer won't tell you. You need to set the flag to zero. Check the initialization routine where the flags are set up and then the flag references throughout the program.

The *Haunted House* listing

This is the program listing for the *Haunted House* adventure. It should run on any computer which uses Microsoft-style BASIC and which has a minimum of 16K of RAM. You may have to make a few minor changes for your computer – look out for comments next to certain lines in the listing. If you have a VIC 20 or an Oric, there are a few extra lines and a line change which you will find at the end of the listing on page 37. If you have a BBC Model A, use mode 7.

This listing will not work as it is on Sinclair (Timex) computers. If you have a Spectrum (Timex 2000), turn to page 38 for changes to make to the program. If you have a ZX81 (Timex 1000) there is a special listing for you on pages 39 to 45.

As this is a long program, you will have to be extremely careful when you type it in. The smallest mistake could prevent it running properly and will be very difficult to find once you've typed the whole program in. Check each line as you go, especially the ON GOSUB and DATA lines. Some of the program lines are so long that they take up two or more lines on the printed page. Look out for these and make sure you do not press RETURN or ENTER until the end of the program line.

```
10 REM  HAUNTED HOUSE ADVENTURE
20 REM  ************************
30 REM  THIS VERSION FOR "MICROSOFT" BASIC
40 REM  REQUIRES A MINIMUM OF 16K
50 REM  SELECT "TEXT MODE" IF NECESSARY
60 REM  ********************************
65 CLEAR 100
70 V=25:W=36:G=18
80 GOSUB 1600
90 CLS:PRINT"HAUNTED HOUSE"
100 PRINT "-------------"
110 PRINT "YOUR LOCATION"
120 PRINT D$(RM)
130 PRINT "EXITS:";
140 FOR I=1 TO LEN(R$(RM))
150 PRINT MID$(R$(RM),I,1);",";
160 NEXT I
170 PRINT
180 FOR I=1 TO G
190 IF L(I)=RM AND F(I)=0 THEN PRINT "YOU CAN SEE ";O$(I);" HERE"
200 NEXT I
210 PRINT "========================"
220 PRINT M$:M$="WHAT"
230 INPUT "WHAT WILL YOU DO NOW";Q$
240 V$="":W$="":VB=0:OB=0
250 FOR I=1 TO LEN(Q$)
260 IF MID$(Q$,I,1)=" " AND V$=""THEN V$=LEFT$(Q$,I-1)
270 IF MID$(Q$,I+1,1)<>" " AND V$<>"" THEN W$=MID$(Q$,I+1,LEN(Q$)-1):I=LEN(Q$)
280 NEXT I
290 IF W$="" THEN V$=Q$
300 FOR I=1 TO V
310 IF V$=V$(I) THEN VB=I
320 NEXT I
330 FOR I=1 TO W
340 IF W$=O$(I) THEN LET OB=I
```

If you have a VIC 20, change CLS to PRINT CHR$(147). If you have an Apple change it to HOME.

Only needed on TRS-80.

Line 70 sets up the variables. V is number of verbs, W is number of object words, G is number of "gettable" objects.

This line sends program to initialization routine.

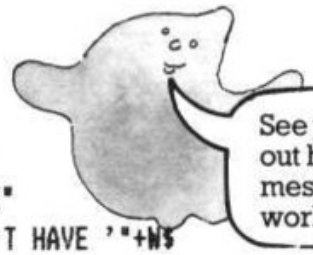
See page 20 to find out how the description and feedback section works.

If you have a BBC, you may need a comma here instead of a semi-colon.

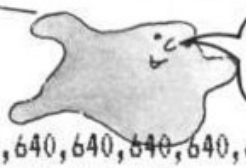
See pages 21-22 to find out how the input section works.

ERROR MESSAGES OVERRIDE CONDITIONS

```
350 NEXT I
360 IF W$>"" AND OB=0 THEN M$="THAT'S SILLY"
370 IF VB=0 THEN VB=V+1
380 IF W$="" THEN M$="I NEED TWO WORDS"
390 IF VB>V AND OB>0 THEN M$="YOU CAN'T '"+Q$+"'"
400 IF VB>V AND OB=0 THEN M$="YOU DON'T MAKE SENSE"
410 IF VB<V AND OB>0 AND C(OB)=0 THEN M$="YOU DON'T HAVE '"+W$
420 IF F(26)=1 AND RM=13 AND RND(3)<>3 AND VB<>21 THEN M$="BATS ATTACKING!":GOTO 90
430 IF RM=44 AND RND(2)=1 AND F(24)<>1 THEN F(27)=1
440 IF F(0)=1 THEN LL=LL-1
450 IF LL<1 THEN F(0)=0
```



See pages 23-24 to find out how the error messages section works.



Use your computer's form of RND here.

BRANCH TO SUBROUTINES

```
460 ON VB GOSUB500,570,640,640,640,640,640,640,640,980,980,1030,1070,1140,1180,1220,1250,
1300,1340,1380,1400,1430,1460,1490,1510,1590
```

VERB 1

```
470 IF LL=10 THEN M$="YOUR CANDLE IS WANING!"
480 IFLL=1 THEN M$="YOUR CANDLE IS OUT!"
490 GOTO90
500 PRINT"WORDS I KNOW:"
510 FOR I=1 TO V
520   PRINT V$(I);",";
530   NEXT I
540 M$="":PRINT
550 GOSUB1580
560 RETURN
```



Change this line for VIC 20 and Oric. See page 37.



Take extra special care to type this line correctly. It will mess up the game if you get it wrong.

VERB 2

```
570 PRINT"YOU ARE CARRYING:"
580 FOR I=1 TO G
590   IFC(I)=1 THEN PRINTO$(I);",";
600   NEXT I
610 M$="":PRINT
620 GOSUB1580
630 RETURN
```



The branch to subroutines section and the verb subroutines are explained on pages 24-25.

VERBS 3-9

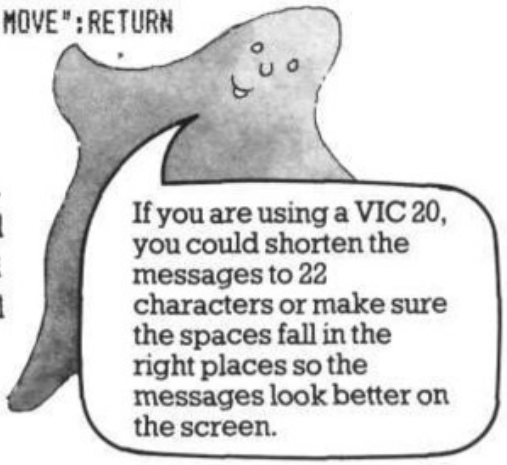
```
640 D=0
650 IF OB=0 THEN D=VB-3
660 IF OB=19 THEN D=1
670 IF OB=20 THEN D=2
680 IF OB=21 THEN D=3
690 IF OB=22 THEN D=4
700 IF OB=23 THEN D=5
710 IF OB=24 THEN D=6
720 IF RM=20 AND D=5 THEN D=1
730 IF RM=20 AND D=6 THEN D=3
740 IF RM=22 AND D=6 THEN D=2
750 IF RM=22 AND D=5 THEN D=3
760 IF RM=36 AND D=6 THEN D=1
770 IF RM=36 AND D=5 THEN D=2
```

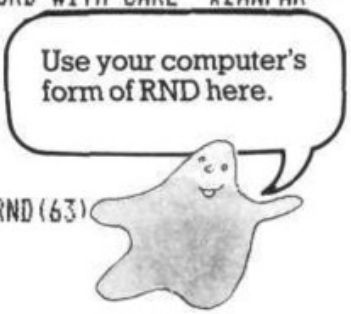


You can find out how the GO subroutine works on pages 26-27.

```
780 IF F(14)=1 THEN M$="CRASH! YOU FELL OUT OF THE TREE!":F(14)=0:RETURN
790 IF F(27)=1 AND RM=52 THEN M$="GHOSTS WILL NOT LET YOU MOVE":RETURN
800 IF RM=45 AND C(1)=1 AND F(34)=0 THEN M$="A MAGICAL BARRIER TO THE WEST":RETURN
810 IF (RM=26 AND F(0)=0) AND (D=1 OR D=4) THEN M$="YOU NEED A LIGHT":RETURN
```

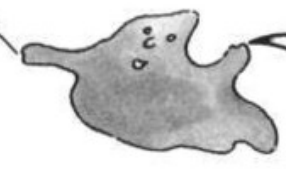
```
 820 IF RM=54 AND C(15) <>1 THEN M$="YOU'RE STUCK!":RETURN
 830 IF C(15)=1 AND NOT (RM=53 OR RM=54 OR RM=55 OR RM=47) THEN M$="YOU CAN'T CARRY A BOA
T!":RETURN
 840 IF (RM>26 AND RM<30) AND F(0)=0 THEN M$="TOO DARK TO MOVE":RETURN
 850 F(35)=0:RL=LEN(R$(RM))
 860 FOR I=1 TO RL
 870   U$=MID$(R$(RM),I,1)
 880   IF (U$="N" AND D=1 AND F(35)=0)THEN RM=RM-8:F(35)=1
 890   IF (U$="S" AND D=2 AND F(35)=0)THEN RM=RM+8:F(35)=1
 900   IF (U$="W" AND D=3 AND F(35)=0)THEN RM=RM-1:F(35)=1
 910   IF (U$="E" AND D=4 AND F(35)=0)THEN RM=RM+1:F(35)=1
 920   NEXT I
 930 M$="OK"
 940 IF F(35)=0 THEN M$="CAN'T GO THAT WAY!"
 950 IF D<1 THEN M$="GO WHERE?"
 960 IF RM=41 AND F(23)=1 THEN R$(49)="SW":M$="THE DOOR SLAMS SHUT!":F(23)=0
 970 RETURN
 980 IF OB>6 THENM$="I CAN'T GET "+W$:RETURN
 985 IF L(OB) <> RM THEN M$="IT ISN'T HERE"
 990 IF F(OB) <>0 THEN M$="WHAT "+W$+"?"
1000 IF C(OB)=1 THEN M$="YOU ALREADY HAVE IT"
1010 IF OB>0 AND L(OB)=RM AND F(OB)=0 THEN C(OB)=1:L(OB)=65:M$="YOU HAVE THE "+W$
1020 RETURN
1030 IF RM=43 AND (OB=28 OR OB=29) THEN F(17)=0:M$="DRAWER OPEN"
1040 IF RM=28 AND OB=25 THEN M$="IT'S LOCKED"
1050 IF RM=38 AND OB=32 THEN M$="THAT'S CREEPY!":F(2)=0
1060 RETURN
1070 IF OB=30 THEN F(18)=0:M$="SOMETHING HERE!"
1080 IF OB=31 THEN M$="THAT'S DISGUSTING!"
1090 IF (OB=28 OR OB=29) THEN M$="THERE IS A DRAWER"
1100 IF OB=33 OR OB=5 THEN GOSUB1140
1110 IF RM=43 AND OB=35 THEN M$="THERE IS SOMETHING BEYOND.."
1120 IF OB=32 THEN GOSUB1030
1130 RETURN
1140 IF RM=42 AND OB=33 THEN M$="THEY ARE DEMONIC WORKS"
1150 IF (OB=3 OR OB=36) AND C(3)=1 AND F(34)=0 THEN M$="USE THIS WORD WITH CARE 'XZANFAR'"
1160 IF C(5)=1 AND OB=5 THEN M$="THE SCRIPT IS IN AN ALIEN TONGUE"
1170 RETURN
1180 M$="OK '"+W$+"'"
1190 IF C(3)=1 AND OB=34THENM$="*MAGIC OCCURS*":IF RM<>45 THEN RM=RND(63)
1200 IF C(3)=1 AND OB=34 AND RM=45 THENF(34)=1
1210 RETURN
1220 IF C(12)=1 THEN M$="YOU MADE A HOLE"
1230 IF C(12)=1 AND RM=30 THEN M$="DUG THE BARS OUT":D$(RM)="HOLE IN WALL":R$(RM)="NSE"
1240 RETURN
1250 IF C(14) <>1 AND RM=7 THEN M$="THIS IS NO TIME TO PLAY GAMES"
1260 IF OB=14 AND C(14)=1 THEN M$="YOU SWUNG IT"
1270 IF OB=13 AND C(13)=1 THEN M$="WHOOSH!"
```





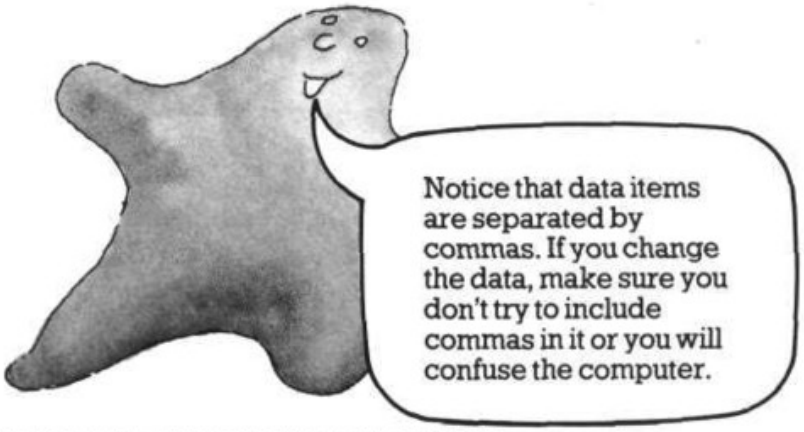
VERBS 10 & 11 VERB 12 VERB 13 VERB 14 VERB 15 VERB 16 VERB 17

```
 1280 IF OB=13 AND C(13)=1 AND RM=43 THEN R$(RM)="WN":D$(RM)="STUDY WITH SECRET ROOM":M$="
YOU BROKE THE THIN WALL"
 1290 RETURN
 1300 IF OB=14 AND C(14)=1 THEN M$="IT ISN'T ATTACHED TO ANYTHING!"
 1310 IF OB=14 AND C(14) <>1 AND RM=7 AND F(14)=0 THEN M$="YOU SEE THICK FOREST AND CLIFF
SOUTH":F(14)=1:RETURN
 1320 IF OB=14 AND C(14) <>1 AND RM=7 AND F(14)=1 THEN M$="GOING DOWN!":F(14)=0
 1330 RETURN
 1340 IF OB=17 AND C(17)=1 AND C(8)=0 THEN M$="IT WILL BURN YOUR HANDS"
 1350 IF OB=17 AND C(17)=1 AND C(9)=0 THEN M$="NOTHING TO LIGHT IT WITH"
 1360 IF OB=17 AND C(17)=1 AND C(9)=1 AND C(8)=1 THEN M$="IT CASTS A FLICKERING LIGHT":F(0)=1
 1370 RETURN
 1380 IF F(0)=1 THEN F(0)=0:M$="EXTINGUISHED"
 1390 RETURN
 1400 IF OB=26 AND C(16)=1 THEN M$="HISSSS"
 1410 IF OB=26 ANDC(16)=1 AND F(26)=1 THEN F(26)=0:M$="PFFT! GOT THEM"
 1420 RETURN
 1430 IF OB=10 AND C(10)=1 AND C(11)=1 THEN M$="SWITCHED ON":F(24)=1
 1440 IF F(27)=1 AND F(24)=1 THEN M$="WHIZZ- VACUUMED THE GHOSTS UP!":F(27)=0
 1450 RETURN
 1460 IF RM=43 AND (OB=27 OR OB=28) THEN GOSUB1030
 1470 IF RM=28 AND OB=25 AND F(25)=0 AND C(18)=1 THEN F(25)=1:R$(RM)="SEW":D$(RM)="HUGE OP
EN DOOR":M$="THE KEY TURNS!"
 1480 RETURN
 1490 IF C(OB)=1 THEN C(OB)=0:L(OB)=RM:M$="DONE"
 1500 RETURN
 1510 S=0
 1520 FOR I=1 TO G
 1530   IF C(I)=1 THENS=S+1
 1540   NEXT I
 1550 IF S=17 AND C(15)<>1 AND RM <>57 THEN PRINT"YOU HAVE EVERYTHING":PRINT"RETURN TO THE
 GATE FOR FINAL SCORE"
 1560 IF S=17 AND RM = 57 THEN PRINT"DOUBLE SCORE FOR REACHING HERE!":S=S$2
 1570 PRINT"YOUR SCORE=";S:IF S>18 THEN PRINT"WELL DONE! YOU FINISHED THE GAME":END
 1580 INPUT"PRESS RETURN TO CONTINUE";Q$
 1590 RETURN
 1600 DIMR$(63),D$(63),O$(W),V$(V)
 1610 DIMC(W),L(G),F(W)
 1620 DATA46,38,35,50,13,18,28,42,10,
25,26,4,2,7,47,60,43,32
 1630 FOR I=1 TO G
 1640   READ L(I)
 1650   NEXT I
 1660 DATAHELP,CARRYING?,GO,N,S,W,E,U,D,GET,TAKE,OPEN,EXAMINE,READ,SAY
 1665 DATADIG,SWING,CLIMB,LIGHT,UNLIGHT,SPRAY,USE,UNLOCK,LEAVE,SCORE
 1680 FOR I=1 TO V
 1690   READ V$(I)
 1700   NEXT I
```

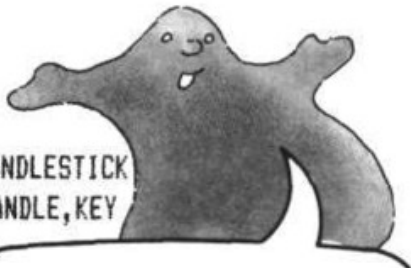


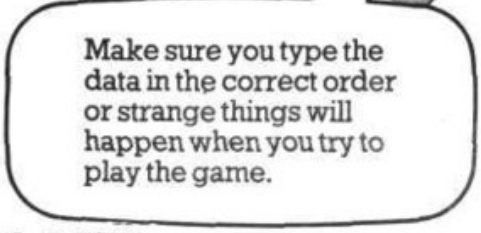
If you have a BBC, you may need a comma here instead of a semi-colon.

```
1710 DATASE,WE,WE,SWE,WE,WE,SWE,WS
1720 DATANS,SE,WE,NW,SE,W,NE,NSW
1730 DATANS,NS,SE,WE,NWUD,SE,WSUD,NS
1740 DATAN,NS,NSE,WE,WE,NSW,NS,NS
1750 DATAS,NSE,NSW,S,NSUD,N,N,NS
1760 DATANE,NW,NE,W,NSE,WE,W,NS
1770 DATASE,NSW,E,WE,NW,S,SW,NW
1780 DATANE,NWE,WE,WE,WE,NWE,NWE,W
1790 FOR I=0 TO 63
1800   READR$(I)
1810   NEXT I
```




```
1820 DATA DARK CORNER,OVERGROWN GARDEN,BY LARGE WOODPILE,YARD BY RUBBISH
1825 DATA WEEDPATCH,FOREST,THICK FOREST,BLASTED TREE
1840 DATA CORNER OF HOUSE,ENTRANCE TO KITCHEN,KITCHEN & GRIMY COOKER,SCULLERY DOOR
1845 DATA ROOM WITH INCHES OF DUST,REAR TURRET ROOM,CLEARING BY HOUSE,PATH
1860 DATA SIDE OF HOUSE,BACK OF HALLWAY,DARK ALCOVE,SMALL DARK ROOM
1865 DATA BOTTOM OF SPIRAL STAIRCASE,WIDE PASSAGE,SLIPPERY STEPS,CLIFFTOP
1880 DATA NEAR CRUMBLING WALL,GLOOMY PASSAGE,POOL OF LIGHT,IMPRESSIVE VAULTED HALLWAY
1885 DATA HALL BY THICK WOODEN DOOR,TROPHY ROOM,CELLAR WITH BARRED WINDOW,CLIFF PATH
1900 DATA CUPBOARD WITH HANGING COAT,FRONT HALL,SITTING ROOM,SECRET ROOM
1905 DATA STEEP MARBLE STAIRS,DINING ROOM,DEEP CELLAR WITH COFFIN,CLIFF PATH
1920 DATA CLOSET,FRONT LOBBY,LIBRARY OF EVIL BOOKS,STUDY WITH DESK & HOLE IN WALL
1925 DATA WEIRD COBWEBBY ROOM,VERY COLD CHAMBER,SPOOKY ROOM,CLIFF PATH BY MARSH
1940 DATA RUBBLE-STREWN VERANDAH,FRONT PORCH,FRONT TOWER,SLOPING CORRIDOR
1945 DATA UPPER GALLERY,MARSH BY WALL,MARSH,SOGGY PATH
1960 DATA BY TWISTED RAILING,PATH THROUGH IRON GATE,BY RAILINGS,BENEATH FRONT TOWER
1965 DATA DEBRIS FROM CRUMBLING FACADE,LARGE FALLEN BRICKWORK,ROTTING STONE ARCH,CRUMBLING
CLIFFTOP
1980 FOR I=0 TO 63
1990   READD$(I)
2000   NEXT I
2010 DATAPAINTING,RING,MAGIC SPELLS,GOBLET,SCROLL,COINS,STATUE,CANDLESTICK
2012 DATAMATCHES,VACUUM,BATTERIES,SHOVEL,AXE,ROPE,BOAT,AEROSOL,CANDLE,KEY
2014 DATANORTH,SOUTH,WEST,EAST,UP,DOWN
2016 DATADOOR,BATS,GHOSTS,DRAWER,DESK,COAT,RUBBISH
2018 DATACOFFIN,BOOKS,XZANFAR,WALL,SPELLS
2060 FOR I=1 TO W
2070   READO$(I)
2080   NEXT I
2090 F(18)=1:F(17)=1:F(2)=1:F(26)=1:F(28)=1:F(23)=1:LL=60:RM=57:M$="OK"
2100 RETURN
```





VIC and Oric changes

If you have a VIC 20 or an Oric, use these lines instead of line 460 in the main listing.

```
455IF VB>14 THEN GOTO465
460ON VB GOSUB 500,570,640,640,640,640,640,640,980,980,1030,1070,1140
463GOTO470
465ON VB-14 GOSUB 1180,1220,1250,1300,1340,1380,1400,1430,1460,1490,1510,1590
```

Changes for the Spectrum (Timex 2000)

Sinclair (Timex) computers use a version of BASIC which differs quite a lot from the BASIC on other popular computers, so you will have to make quite a lot of changes to make it work. These changes make the program slightly too long to fit into a 16K Spectrum. You could, however, try adapting the program to fit by cutting out some of the verbs, for example, and shortening the messages.

Use the lines listed below to replace lines in the main program and also change the main program as follows:

1. The Spectrum needs LET every time you assign a value to a variable e.g. LET V=25. This affects many lines, including all the ones containing IF . . . THEN, so be careful.
2. All the string data in lines 1660, 1665, 1710-1780, 1820-1965 and 2010-2018 must be put in quotes, like this:
1820DATA"DARK CORNER",
"OVERGROWN GARDEN", etc.
3. In lines 1790 and 1980, change the loop to read FOR I=1 TO 64. (The Spectrum won't allow you to use the box labelled zero in an array.)

```
 60 LETF$="                "
120 PRINT D$(RM+1)
140 FOR I=1 TO LEN(R$(RM+1))
150 PRINT R$(RM+1) (I TO I);",";
240 LET X$="":LET W$="":LET VB=0:LET OB=0
250 FOR I=1 TO LEN(Q$)-1
260 IF Q$(I TO I)=" " AND X$="" THEN LET X$=Q$( TO I-1)
270 IF Q$(I+1 TO I+1)<>" " AND X$="" THEN LET W$=Q$(I+1 TO):LET I=LEN(Q$)-1
290 IF W$="" THEN LET X$=Q$
295 IF LEN(X$) > LEN(V$(1)) OR X$="" THEN GOTO 325
296 LET X$=X$+F$( TO (LEN(V$(1))-LEN(X$)))
310 IF X$=V$(I) THEN LET VB=I
325 IF W$="" OR LEN(W$)>LEN(O$(1)) THEN GOTO 360
326 LET W$=W$+F$( TO (LEN(O$(1))-LEN(W$)))
405 IF OB=0 THEN GOTO420
420 IF F(26)=1 AND RM=13 AND INT(RND*3+1)<>3 AND VB <>21 THEN LET M$="BATS ATTACKING"
430 IF RM=44 AND INT(RND*2)=1 AND F(24)<>1 THEN LET F(27)=1
440 IF F(20)=1 THEN LET LL=LL-1
450 IF LL<1THEN LET F(20)=0
460 GOSUB 500*(VB=1)+570*(VB>2 AND VB<10)+980*(VB=10 OR VB=11)+1030*(VB=12)+1070*(VB=13)+1
140*(VB=14)+1180*(VB=15)+1220*(VB=16)+1250*(VB=17)+1300*(VB=18)+1340*(VB=19)+1380*(VB=20)+
1400*(VB=21)+1430*(VB=22)+1460*(VB=23)+1490*(VB=24)+1510*(VB=25)+1590*(VB=26)
800 IF (RM=26 AND F(20)=0) AND (D=1 OR D=4) THEN LET M$="YOU NEED A LIGHT":RETURN
840 IF (RM>26 AND RM<30) AND F(20)=0 THEN LET M$="TOO DARK TO MOVE":RETURN
850 LET F(35)=0:LET RL=LEN(R$(RM+1))
870 LET U$=R$(RM+1) (I TO I)
960 IF RM=41 AND F(23)=1 THEN LET R$(50)="SW":LET M$="THE DOOR SLAMS SHUT!":LET F(23)=0
980 IF OB>6 OR OB=0 THEN LET M$="I CAN'T GET "+W$:RETURN
1190 IF C(3)=1 AND OB=34 THEN LET M$="*MAGIC OCCURS*":IF RM<>45 THEN LET RM=INT(RND*64)
1230 IF C(12)=1 AND RM=30 THEN LET M$="DUG THE BARS OUT":LET D$(RM+1)="HOLE IN WALL":LET
R$(RM+1)="NSE"
1280 IF OB=13 AND C(13)=1 AND RM=43 THEN LET R$(RM+1)="WN":LET D$(RM+1)="STUDY WITH SECRE
T ROOM":LET M$="YOU BROKE THE THIN WALL"
1360 IF OB=17 AND C(17)=1 AND C(9)=1 AND C(8)=1 THEN LET M$="IT CASTS A FLICKERING LIGHT"
:LET F(20)=1
1380 IF F(20)=1 THEN LET F(20)=0:LET M$="EXTINGUISHED"
1470 IF RM=28 AND OB=25 AND F(25)=0 AND C(18)=1 THEN LET F(25)=1:LET R$(RM+1)="SEW":LET D
$(RM+1)="HUGE OPEN DOOR":LET M$="THE KEY TURNS!"
1600 DIM R$(64,4):DIM D$(64,4):DIM O$((W,13):DIM V$(V,9)
```

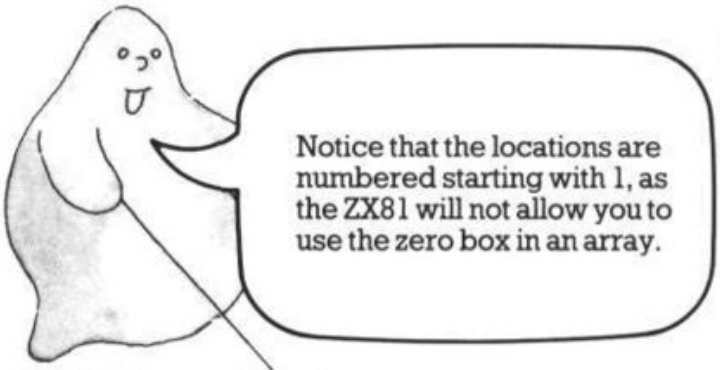
Remember that if you type in the data in capital letters, you must play the game using capitals – the computer does not recognize that "GO WEST" and "go west" are the same thing. It is best to keep the caps lock on all the time.



ZX81 (Timex 1000) version

The program listing on the next six pages is a special version of *Haunted House* for the ZX81. It sticks as closely as possible to the structure of the main program, so you can follow the explanations of the program given throughout this book. The main differences are that the ZX81 will accept only one statement on each line and it does not have the commands READ . . . DATA. The program has been rewritten to take account of these and other differences in the BASIC which you will see pointed out on the listing.

These changes take up quite a lot of memory space. In order to make the game fit, the number of locations has been cut from 64 to 36 and other data changed slightly to fit with this. The ZX81 version of the master plan is shown below. You can find out how the data is put into the computer's memory over the page.



ZX81 master plan

1 ENTRANCE TO KITCHEN	2 KITCHEN	3 SCULLERY	4 DUSTY ROOM	REAR TURRET ROOM	6 CLOSET WITH COAT
7 BACK OF HALLWAY	8 DARK ALCOVE	9 SMALL ROOM WITH RUBBISH	10 SPIRAL STAIRCASE	11 WIDE PASSAGE	12 SLIPPERY STEPS
13 HALLWAY TO REAR	14 POOL OF LIGHT	15 VAULTED HALL	16 HALL WITH LOCKED DOOR	17 TROPHY ROOM	18 CELLAR
19 FRONT HALL	20 SITTING ROOM	21 SECRET ROOM	22 STEEP MARBLE STAIRS	23 DINING ROOM	24 VAULT WITH COFFIN
25 FRONT LOBBY	26 LIBRARY	27 STUDY	28 COBWEBBY ROOM	29 COLD CHAMBER	30 SPOOKY ROOM
31 FRONT PORCH	32 FRONT TOWER	33 SLOPING CORRIDOR	34 UPPER GALLERY	35 BOAT HOUSE	36 SOGGY PATH

How to use the program

If you look through this listing, you will notice that the data for the game is not incorporated in the program. The program works by asking you to type in the data and then saving the whole program, including the data, on tape. You only need do this once – next time you want to play the game, all you have to do is load the tape.

Follow these instructions to use the program:

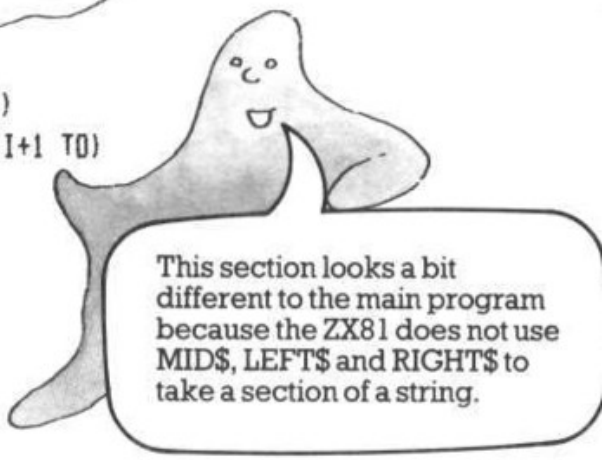
1. Type in the program (very carefully).
2. Type RUN 2440.
3. Now type in the data in the following order (see page 45 for lists of data):
   a) location descriptions
   b) routes
   c) object words
   d) verbs
   The program stops after each section so you can re-enter any data which you put in incorrectly. If, for instance, you want to put the verb data in again, type GOTO 2720. If you want to carry on to the next input section, type CONT, followed by NEWLINE.
4. Now SAVE the program on tape. This will save all the data as well.
5. To start the game, type GOTO 10. DO NOT TYPE RUN, as this will destroy all the variables.
6. Now input the starting positions of the objects. When you input the last of these (18) the program will give you your starting location.
7. For a new game, repeat steps 5 and 6.
8. When you load the program from tape, start these instructions at step 5.

DESCRIPTION & FEEDBACK

INPUT & INPUT ANALYSIS

```
10 GOSUB 2200
20 CLS
30 PRINT "HAUNTED HOUSE ADVENTURE"
40 PRINT "-----------------------"
50 PRINT "YOUR LOCATION:"
60 PRINT D$(RM)
70 PRINT "EXITS:"
80 FOR I=1 TO LEN(R$(RM))
90 PRINT R$(RM)(I TO I);",";
100 NEXT I
110 PRINT
120 FOR I=1 TO 6
130 IF L(I)=RM AND F(I)=0 THEN PRINT "YOU CAN SEE ";O$(I);" HERE"
140 NEXT I
150 PRINT "===================="
160 PRINT M$
170 LET M$="WHAT ?"
180 PRINT "WHAT WILL YOU DO NOW"
190 INPUT Q$
200 LET X$=""
210 LET W$=""
220 LET VB=0
230 LET OB=0
240 FOR I=1 TO LEN(Q$)-1
250 IF Q$(I TO I)=" " AND X$="" THEN LET X$=Q$(TO I-1)
260 IF Q$(I+1 TO I+1)<>" " AND X$<>"" THEN LET W$=Q$(I+1 TO)
270 IF W$<>"" THEN LET I=LEN(Q$)-1
280 NEXT I
290 IF W$="" THEN LET X$=Q$
300 IF LEN(X$)>LEN(V$(1)) OR X$="" THEN GOTO 420
310 LET F=LEN(V$(1))-LEN(X$)
320 LET X$=X$+F$(TO F)
330 FOR I=1 TO V
340 IF X$=V$(I) THEN LET VB=I
350 NEXT I
360 IF W$="" OR LEN(W$)>LEN(O$(1)) THEN GOTO 430
```





ERROR MESSAGES
OVERRIDE CONDITIONS

```
370 LET F=LEN(O$(1))-LEN(W$)
380 LET W$=W$+F$(TO F)
390 FOR I=1 TO W
400 IF W$=O$(I) THEN LET OB=I
410 NEXT I
420 IF W$>"" AND OB=0 THEN LET M$="THATS SILLY"
430 IF VB=0 THEN LET VB=V+1
440 IF W$="" THEN LET M$="I NEED TWO WORDS"
450 IF VB>V AND OB>0 THEN LET M$="YOU CANT "+Q$
460 IF VB>V AND OB=0 THEN LET M$="YOU DONT MAKE SENSE"
470 IF OB=0 OR OB>6 THEN GOTO 490
480 IF VB<V AND OB>0 AND C(OB)=0 THEN LET M$="YOU DONT HAVE "+W$
490 IF F(26)=0 OR RM<>5 OR INT(RND*3)=2 OR VB=21 THEN GOTO 520
500 LET M$="BATS ATTACKING"
510 GOTO 20
520 IF RM=28 AND INT(RND*2)=1 AND F(24)=0 THEN LET F(27)=1
530 IF F(20)=1 THEN LET LL=LL-1
540 IF LL<1 THEN LET F(20)=0
```

BRANCH TO SUBROUTINES

```
550 GOSUB 590*(VB=1)+660*(VB=2)+730*(VB)2 AND VB(10)+1160*(VB=10 OR VB=11)+1270*(VB=12)
+1350*(VB=13)+1440*(VB=14)+1480*(VB=15)+1540* (VB=16)+1560*(VB=17)+1640*(V B=18)+1700*(VB=19)
+1760*(VB=20)+1800*(VB=21)+1850*(VB=22)+1920*(VB=23)+1990*(VB=24)+2040*(VB=25)+2190*(VB=26)
560 IF LL=10 THEN LET M$="YOUR CANDLE IS WANING"
570 IF LL=1 THEN LET M$="YOUR CANDLE IS OUT"
580 GOTO 20
```

VERB 1

```
590 PRINT "WORDS I KNOW"
600 FOR I=1 TO V
610 PRINT V$(I);",";
620 NEXT I
630 LET M$=""
640 GOSUB 2160
650 RETURN
```

VERB 2

```
660 PRINT "YOU ARE CARRYING:"
670 FOR I=1 TO 6
680 IF C(I)=1 THEN PRINT O$(I);",";
690 NEXT I
700 LET M$=""
710 GOSUB 2160
720 RETURN
```

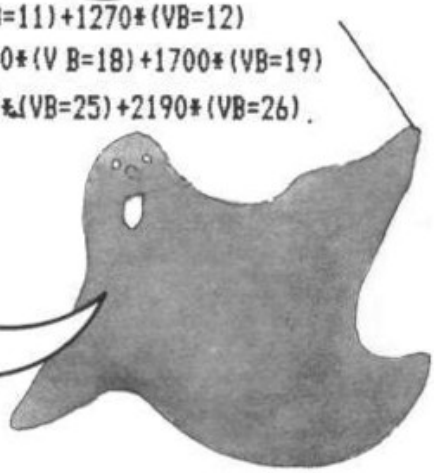
VERBS 3-9

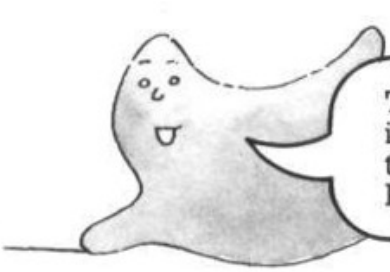
```
730 LET D=0
740 IF OB=0 THEN LET D=VB-3
750 IF OB>18 AND OB<25 THEN LET D=OB-18
760 IF RM=10 AND D=5 THEN LET D=1
770 IF RM=10 AND D=6 THEN LET D=3
780 IF RM=12 AND D=6 THEN LET D=2
790 IF RM=12 AND D=5 THEN LET D=3
800 IF RM=22 AND D=6 THEN LET D=1
810 IF RM=22 AND D=5 THEN LET D=2
820 IF RM<>32 OR D<>3 THEN GOTO 850
830 LET M$="ITS A LONG DROP"
```

Notice that some of the program lines are longer than the printed lines on the page. Make sure you don't press NEWLINE before the end of the program line.



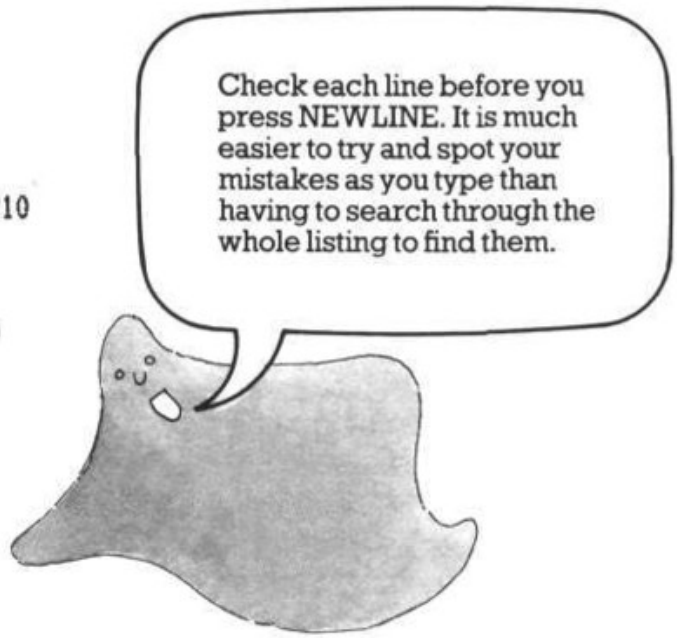
This line replaces the ON GOSUB line, which the ZX81 can't do. It works like one long calculation, using the value of VB. The computer looks at each of the brackets containing "VB=" and puts a 1 if the bracket is true and a zero if it isn't. Try working through the calculation using a particular value of VB to see how it works.

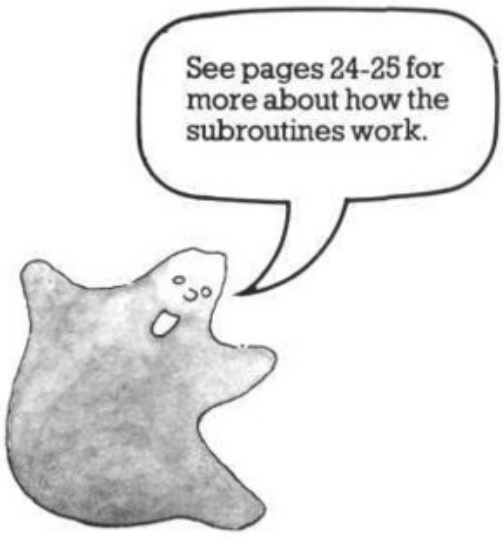


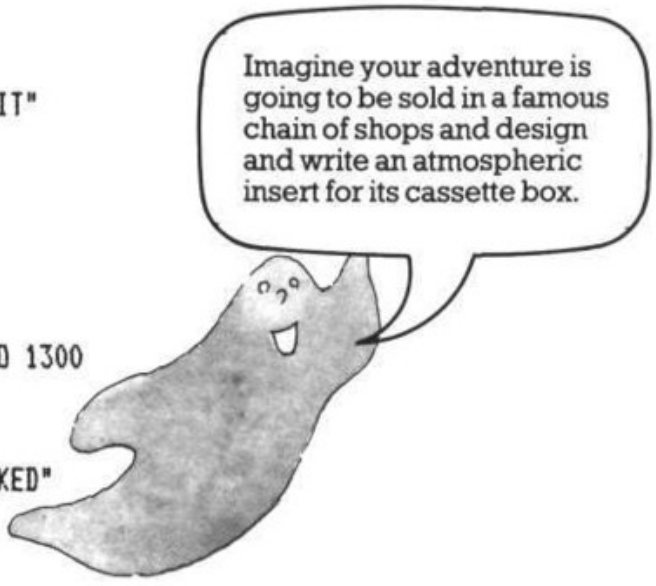


This replaces the tree section in the main program. Check the plan if you want to see what location 32 is.

```
840 RETURN
850 IF F(27)=0 OR RM<>34 THEN GOTO 880
860 LET M$="GHOSTS WILL NOT LET YOU MOVE"
870 RETURN
880 IF RM<>29 OR C(1)=0 OR F(34)=1 THEN GOTO 910
890 LET M$="MAGICAL BARRIER TO THE WEST"
900 RETURN
910 IF RM<14 OR RM>17 OR F(20)=1 THEN GOTO 950
920 IF RM=14 AND D<>1 AND D<>4 THEN GOTO 950
930 LET M$="TOO DARK TO MOVE"
940 RETURN
950 IF C(15)=0 OR RM<>36 THEN GOTO 980
960 LET M$="THE BOAT IS TOO HEAVY"
970 RETURN
980 LET RL=LEN(R$(RM))
990 LET OM=RM
1000 FOR I=1 TO RL
1010 LET U$=R$(RM)(I TO I)
1020 IF U$="N" AND D=1 THEN LET OM=OM-6
1030 IF U$="S" AND D=2 THEN LET OM=OM+6
1040 IF U$="W" AND D=3 THEN LET OM=OM-1
1050 IF U$="E" AND D=4 THEN LET OM=OM+1
1060 NEXT I
1070 LET M$="OK"
1080 IF RM=OM THEN LET M$="CANT GO THAT WAY"
1090 LET RM=OM
1100 IF D<1 THEN LET M$="GO WHERE ?"
1110 IF RM<>25 OR F(23)=0 THEN GOTO 1150
1120 LET R$(31)=""
1130 LET M$="THE DOOR SLAMS SHUT BEHIND YOU"
1140 LET F(23)=0
1150 RETURN
1160 IF OB>0 AND OB<=6 THEN GOTO 1190
1170 LET M$="YOU CANT GET "+W$
1180 RETURN
1190 IF L(OB)<>RM THEN LET M$="ITS NOT HERE"
1200 IF F(OB)=1 THEN LET M$="WHAT "+W$+" ?"
1210 IF C(OB)=1 THEN LET M$="YOU ALREADY HAVE IT"
1220 IF L(OB)<>RM OR F(OB)=1 THEN GOTO 1260
1230 LET C(OB)=1
1240 LET M$="YOU HAVE THE "+W$
1250 LET L(OB)=37
1260 RETURN
1270 IF RM<>27 OR (OB<>28 AND OB<>29) THEN GOTO 1300
1280 LET M$="DRAWER OPEN"
1290 LET F(17)=0
1300 IF RM=16 AND OB=25 THEN LET M$="IT IS LOCKED"
1310 IF RM<>24 OR OB<>32 THEN GOTO 1340
1320 LET M$="CREEPY"
```







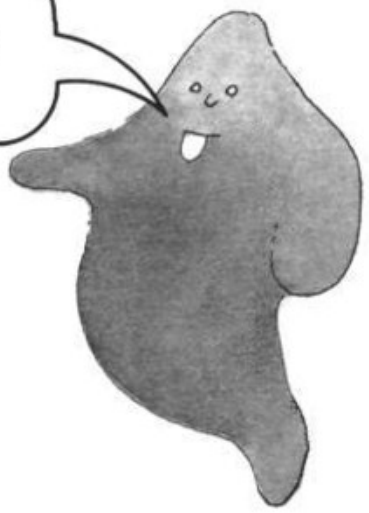
VERBS 10 & 11

VERB 12

```
1330 LET F(2)=0
1340 RETURN
1350 IF OB<>30 THEN GOTO 1380
1360 LET M$="SOMETHING HERE"
1370 LET F(18)=0
1380 IF OB=28 OR OB=29 THEN LET M$="THERE IS A DRAWER"
1390 IF OB=33 OR OB=5 THEN GOSUB 1440
1400 IF RM=27 AND OB=35 THEN LET M$="SOMETHING BEYOND"
1410 IF OB=32 THEN GOSUB 1270
1420 IF RM=9 AND OB=31 THEN LET M$="THATS DISGUSTING"
1430 RETURN
1440 IF RM=26 AND OB=33 THEN LET M$="THEY ARE DEMONIC WORKS"
1450 IF (OB=3 OR OB=36) AND C(3)=1 AND F(34)=0 THEN LET M$="USE
THIS WORD WITH CARE - XZANFAR"
1460 IF C(5)=1 AND OB=5 THEN LET M$="AN ALIEN TONGUE"
1470 RETURN
1480 LET M$="OK "+W$
1490 IF C(3)=0 OR OB<>34 THEN GOTO 1520
1500 LET M$="MAGIC OCCURS"
1510 IF RM<>29 THEN LET RM=INT(RND*36)+1
1520 IF C(3)=1 AND OB=34 AND RM=29 THEN LET F(34)=1
1530 RETURN
1540 IF C(12)=1 THEN LET M$="YOU HAVE MADE A HOLE"
1550 RETURN
1560 IF C(14)<>1 AND RM=11 THEN LET M$="THIS IS NO TIME FOR GAMES"
1570 IF OB=14 AND C(14)=1 THEN LET M$="YOU SWUNG IT"
1580 IF OB=13 AND C(13)=1 THEN LET M$="WHOOSH"
1590 IF OB<>13 OR C(13)=0 OR RM<>27 THEN GOTO 1630
1600 LET R$(RM)="WN"
1610 LET D$(RM)="STUDY WITH SECRET ROOM"
1620 LET M$="YOU BROKE THROUGH"
1630 RETURN
1640 IF RM<>32 OR C(14)<>1 OR OB<>14 THEN GOTO 1680
1650 LET M$="GOING DOWN"
1660 LET RM=RM-1
1670 GOTO 1690
1680 LET M$="WHERE TO ?"
1690 RETURN
1700 IF OB=17 AND C(17)=1 AND C(8)=0 THEN LET M$="IT WILL BURN YOUR HANDS"
1710 IF OB=17 AND C(17)=1 AND C(9)=0 THEN LET M$="WHAT WITH ?"
1720 IF OB<>17 OR C(17)=0 OR C(9)=0 OR C(8)=0 THEN GOTO 1750
1730 LET M$="IT CASTS A FLICKERING LIGHT"
1740 LET F(20)=1
1750 RETURN
1760 IF F(20)=0 THEN GOTO 1790
1770 LET M$="EXTINGUISHED"
1780 LET F(20)=0
1790 RETURN
1800 IF OB=26 AND C(16)=1 THEN LET M$="HISSSS"
```

VERB 13 VERB 14 VERB 15 16 VERB 17 VERB 18 VERB 19 VERB 20

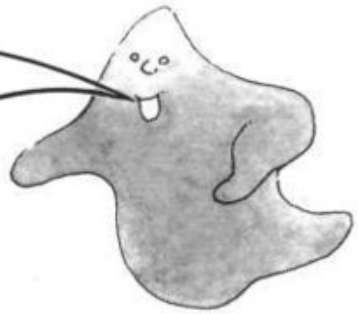
Check the two versions of *Haunted House* against each other to see where the programs differ.



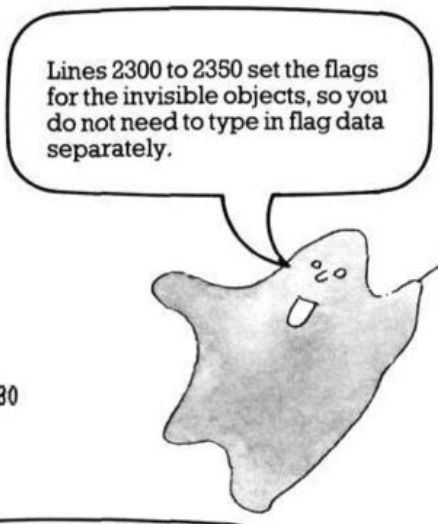
Don't forget, you can add extra verbs without adding extra subroutines. You could add LOOK for example and make it use the EXAMINE subroutine.

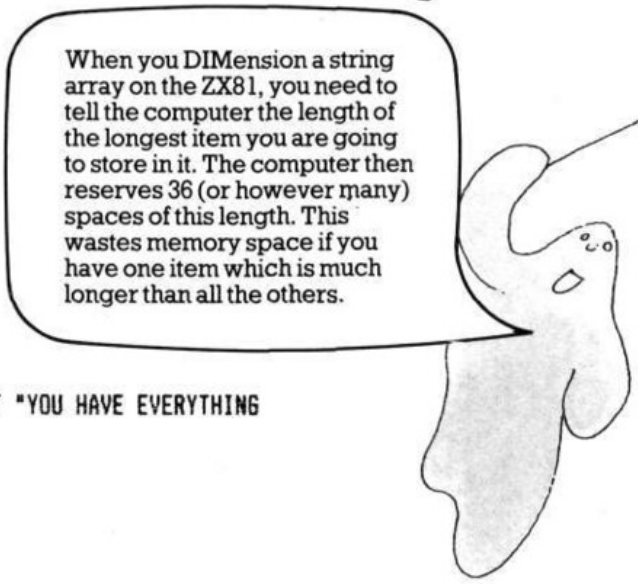


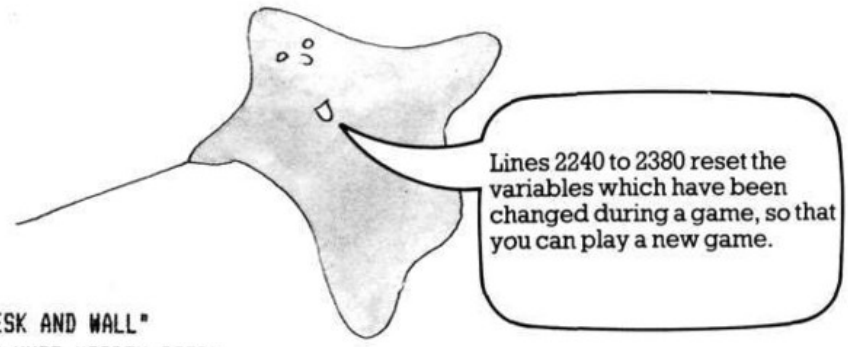
Perhaps you can think of a better verb than "unlight".



```
1810 IF OB<>26 OR C(16)<>1 OR F(26)=0 THEN GOTO 1840
1820 LET M$="PFFT - GOT THEM"
1830 LET F(26)=0
1840 RETURN
1850 IF OB<>10 OR C(10)=0 OR C(11)=0 THEN GOTO 1910
1860 LET F(24)=1
1870 LET M$="SWITCHED ON"
1880 IF F(27)=0 OR F(24)=0 THEN GOTO 1910
1890 LET M$="YOU VACUUMED THEM UP"
1900 LET F(27)=0
1910 RETURN
1920 IF RM=27 AND (OB=27 OR OB=28) THEN GOSUB 1270
1930 IF RM<>16 OR OB<>25 OR F(25)=1 OR C(18)=0 THEN GOTO 1980
1940 LET F(25)=1
1950 LET M$="THE KEY TURNS - CLUNK"
1960 LET R$(RM)="SEW"
1970 LET D$(RM)="HUGE OPEN DOOR"
1980 RETURN
1990 IF C(OB)=0 THEN GOTO 2030
2000 LET C(OB)=0
2010 LET M$="DONE"
2020 LET L(OB)=RM
2030 RETURN
2040 LET S=0
2050 FOR I=1 TO 6
2060 IF C(I)=1 THEN LET S=S+1
2070 NEXT I
2080 IF S=17 AND C(15)=0 AND RM<>31 THEN PRINT "YOU HAVE EVERYTHING
RETURN TO PORCH FOR FINAL SCORE"
2090 IF S<>17 OR RM<>31 THEN GOTO 2120
2100 PRINT "DOUBLE SCORE"
2110 LET S=S*2
2120 PRINT "YOUR SCORE ";S
2130 IF S<18 THEN GOTO 2160
2140 PRINT "WELL DONE - YOU HAVE FINISHED"
2150 STOP
2160 PRINT "PRESS NEW LINE TO CONTINUE"
2170 INPUT Q$
2180 LET M$="OK"
2190 RETURN
2200 FOR I=1 TO W
2210 LET F(I)=0
2220 LET C(I)=0
2230 NEXT I
2240 LET R$(31)="N"
2250 LET R$(27)="W"
2260 LET R$(16)="NE"
2270 LET D$(27)="STUDY, DESK AND WALL"
2280 LET D$(16)="HALL WITH HUGE WOODEN DOOR"
```







DATA INPUT SECTION

```
2290 LET M$="OK"
2300 LET F(18)=1
2310 LET F(17)=1
2320 LET F(27)=1
2330 LET F(2)=1
2340 LET F(26)=1
2350 LET F(23)=1
2360 LET LL=60
2370 LET RM=31
2380 LET F$="             "
2390 FOR I=1 TO 6
2400 PRINT I
2410 INPUT L(I)
2420 NEXT I
2430 RETURN
2440 DIM R$(36,4)
2450 DIM D$(36,30)
2460 LET V=25
2470 DIM V$(V,9)
2480 LET W=36
2490 DIM O$(W,13)
2500 DIM C(W)
2510 DIM F(W)
2520 LET G=18
2530 DIM L(G)
2540 PRINT "DESCRIPTIONS"
2550 FOR I=1 TO 36
2560 PRINT I
2570 INPUT D$(I)
2580 NEXT I
2590 STOP
2600 PRINT "ROUTES"
2610 FOR I=1 TO 36
2620 PRINT I
2630 INPUT R$(I)
2640 NEXT I
2650 STOP
2660 PRINT "OBJECTS"
2670 FOR I=1 TO W
2680 PRINT I
2690 INPUT O$(I)
2700 NEXT I
2710 STOP
2720 PRINT "VERBS"
2730 FOR I=1 TO V
2740 PRINT I
2750 INPUT V$(I)
2760 NEXT I
2770 STOP
```

13 SPACES

The data: Load the data in this order. (See page 40.)

DESCRIPTIONS

ENTRANCE TO KITCHEN, KITCHEN WITH GRIMY COOKER, SCULLERY, DUSTY ROOM, REAR TURRET ROOM, CLOSET WITH COAT, BACK HALLWAY, DARK ALCOVE, SMALL ROOM WITH RUBBISH, SPIRAL STAIRCASE, WIDE PASSAGE, SLIPPERY STEPS, HALLWAY TO REAR, POOL OF LIGHT, VAULTED HALL, HALL WITH HUGE WOODEN DOOR, TROPHY ROOM, CELLAR ROOM, FRONT HALL, SITTING ROOM, SECRET ROOM, STEEP MARBLE STAIRS, DINING ROOM, VAULT WITH COFFIN, FRONT LOBBY, LIBRARY OF EVIL BOOKS, STUDY WITH DESK. HOLE IN WALL, COBWEBBY ROOM, VERY COLD CHAMBER, SPOOKY ROOM, FRONT PORCH, TOP OF FRONT TOWER, SLOPING CORRIDOR, UPPER GALLERY, BOAT HOUSE. SOGGY PATH

ROUTES

SE, WE, W, SE, WE, W,
NS, SE, WE, NWUD, SE, SWUD,
NS, NSE, WE, WE, NSW, NS,
NSE, NSW, S, NSUD, N, N,
N, NE, W, NSE, WE, SW,
N, WE, WE, NW, E, NW

The commas show where you should press NEWLINE between data items. DON'T TYPE THE COMMAS.



OBJECTS

PAINTING, RING, MAGIC SPELLS, GOBLET, SCROLL, COINS, STATUE, CANDLESTICK, MATCHES, VACUUM, BATTERIES, SHOVEL, AXE, ROPE, BOAT, AEROSOL, CANDLE, KEY, NORTH, SOUTH, WEST, EAST, UP, DOWN, DOOR, BATS, GHOSTS, DRAWER, DESK, COAT, RUBBISH, COFFIN, BOOKS, XZANFAR, WALL, SPELLS

VERBS

HELP, CARRYING?, GO, N, S, W, E, U, D,
GET, TAKE, OPEN, EXAMINE, READ, SAY,
DIG, SWING, CLIMB, LIGHT, UNLIGHT, SPRAY,
USE, UNLOCK, LEAVE, SCORE

STARTING LOCATIONS FOR OBJECTS

(You must type in this set of data each time you run the program.)
30, 24, 21, 32, 5, 8,
16, 26, 2, 13, 14, 36,
18, 11, 35, 3, 27, 6

Extra tips and hints

1. Use integer variables

On some computers you can put a % sign after number variable names to show that you only want to put integers or whole numbers in them (numbers without anything to the right of the decimal point that is). So variable V becomes V% and so on. You can do this on the BBC, TRS-80, Dragon, TRS-Color and Oric. It is useful to do this because it saves memory space and increases the speed by as much as 50%. The speed is particularly noticeable when the computer is executing long loops.

2. Think about screen presentation

If you have enough spare memory, you could improve the way the game looks on the screen. You could add a graphics routine for the opening title for instance and make the text flash on and off at particular points in the game, such as when the candle flickers or the ghosts appear. The text need not be printed at the edge of the screen, nor need it all be the same colour. You could make use of coloured borders and backgrounds too.

3. Watch your spelling

If you are not quite sure how to spell a word you want to use in the game, check it in a dictionary. Your computer doesn't know how to spell and will store whatever you tell it in its memory. This could be very frustrating for the player who is using the correct version and keeps getting error messages because the computer doesn't recognize the word.

4. Spread the action

Some adventure games are a bit boring to play because everything happens in the same place. Try to make sure there are interesting things all through the game.

5. Use REM statements

When you are writing a program as long and complicated as an adventure, it is a very good idea to put REM statements in front of each section. You are quite likely to get confused as to which section is which if you don't. When you have finished the program, though, take the REMs out – they take up memory space, slow the program up and allow unscrupulous players to cheat.

6. Use helpful variable names

Try to name your variables so that it is easy to remember what each one is e.g. OB for objects, MK for mistakes, and so on. If you have plenty of memory space and your computer will allow you, it is a good idea to use long variable names to help you remember what each variable is, e.g. instead of V use VERB. Make a list of your variables and what they are anyway, so you don't mix them up while you are writing the program.

7. Keep it simple

Don't be too ambitious with your first games. A simple, well-thought-out game will be more fun to play than a confused, complicated one. Not everyone wants to play a game which goes on for days.

8. Keep it friendly

When you have written your game, look at the comments to make sure they are not ambiguous or misleading. Instead of "TOO DARK", for instance, you could say "YOU NEED A LIGHT TO GO HERE". Remember, something that is obvious to you will not be at all obvious to a player. Make some of your comments funny too as this will help the player feel the computer is really talking to him.

Answers to puzzles

Detective game puzzle (page 13)

Here are some suggested solutions for the problems in the detective game. See how they compare with the solutions you thought of.

1. You will only see the hair if you instruct computer to examine coat. You cannot take hair unless you have a clean envelope to put it in.

2. You need a key to open the drawer, a magnifying glass to see the thread and a second clean envelope to put it in.

3. You need plaster and a container of water to make a plaster cast of the footprint.

4. You need talcum powder to show up prints and sticky tape to lift print off surface to take away.

5. You need a portable blood analysis kit (described in game as a box containing bottles and other scientific equipment).

6. You need a handkerchief to pick up the stick and a polythene bag to carry it in.

Adventure brain teasers (page 15)

Remember there are no "correct" answers to these puzzles. Here are some suggested solutions.

1. Lift the carpet and find a trap door.

2. Use the handkerchief as a mask (assuming drowsiness is caused by a gas in the room), look inside rucksack and find a flask. Open the flask and find black coffee. Drink coffee.

3. Read scroll (which is a proclamation to free the slaves).

4. Throw the dessert (which happens to be custard pie) in the arch-villain's face. Grab the remote control and escape.

Puzzle (page 28)

Here is how you can change the program to limit the number of objects that can be carried at one time.

You need two new variables, here they are called CO (which stands for "carried objects") and CL (which stands for "carrying limit"). Add these to the end of line 70 like this:

```
70 . . . :CO=0:CL=8
```

You then need to tell the computer to add one to CO in the GET routine when the player picks up an object and subtract one from it in the LEAVE routine if he drops an object. Do this by adding to the ends of lines 1010 and 1490 like this:

```
1010 . . . :CO=CO+1
1490 . . . :CO=CO-1
```

Now add a new line to the GET routine to check if CO equals the limit before proceeding with the rest of the routine.

(CL need not be 8, but it cannot be less or the player would not be able to carry all the treasures to the finish.)

Going further

Once you have written an adventure, you could join the BBC Micro Adventure/Fantasy Club. This is a postal club and it provides a library of adventure and fantasy games written by members for the use of other members. To find out more, write to:
BBC Micro Adventure Club, 29 Blackthorne Drive, Larkfield, Kent ME20 6NR, England.

Index

Adams, Scott, 4
Adventure, 4
adventure games, different types, 4
array error, 32
arrays, 16, 17, 18, 31, 38
BASIC 3, 4, 16, 18, 23, 38, 39, 40
BBC, 3, 23, 29, 31, 33, 36, 46
BBC Micro Adventure Club, 47
branch to subroutines, 19, 24, 34
caps lock, use of, 38
carrying array, 17, 18
changing the program, 28-32, 38
Colossal Cave, 4
combinations of words, 22, 23
commas, in data, 18, 37, 45
CPUs, 17
Crowther and Woods, 4
2D arrays, 17
2D games, 26
3D games, 9, 26
data, 16, 17, 18
  loops, 18
  out of, 32
  storing the, 16, 17, 18
  for ZX81 game , 45
database, 5
debugging, 32
descriptions, 19, 20, 25, 28, 33
descriptions of the locations array, 16, 17, 20, 31, 39
detective story adventure, 7, 11
DIM, 16, 17, 32, 44
dimensioning, 16, 17
disc-based adventures, 4
Dragon, 3, 29, 46
dummy subroutine, 23
error code, 32
error messages, setting up in game, 19, 23, 24, 25, 34
feedback, 19, 20, 25, 33
first adventure game, 4
flag arrays, 17, 18, 24, 27, 32, 44
flag registers, 17
Fortran, 4
"gettable" objects, 17
Go subroutine, 26, 27, 34, 41
GOSUB, 19, 23, 29
GOTO, 19
graphics, 4, 46
grid, drawing a, 8, 9
HELP, 13, 21
hiding places, 6
HOME, 33
IF . . . THEN, 38
initialization, 18, 19, 20, 30, 32, 33
input, 19, 21, 33
input analysis, 19, 22
instructions, player's, 21
integer variables, 46
interactive database, 5
INVENTORY, 13
invisible objects, 17, 18
LEFT$, 40
LET, 38, 40
listing, program, 33-37
  Spectrum (Timex 2000) version, 38
  ZX81 (Timex 1000) version, 39-45
LOAD subroutine, 30, 31
locations, 6, 8, 16, 17, 40, 41
  numbering of, 8, 12
locations array, 16, 17, 20, 39
loop, 18, 20
machine code, 5
magic, use of, 6
map, of adventure world, 6-7
master plan, 8, 11, 12, 14, 15, 16, 26
  ZX81 version, 39
memory, amount used up by game, 28
Microsoft-style BASIC, 3, 33
MID$, 40
mistakes, correcting, 32
no match, 22
number arrays, 17
numbering of locations, 8
  objects, 10, 12
object word array, 17, 18
object words, 40
one-way routes, 8, 9
one-word commands, 21, 23, 26
ON . . . GOSUB, 24, 25, 30, 32, 33, 41
on range error, 23
Oric, 3, 29, 33, 34, 37, 46
out of data, 32
override conditions, 19, 24, 32
penalties, 30
planning, 5-15
point of the game, 6
problems for player, 10
program
  changing the, 28-32
  structure, 19
  writing the, 19-27
props, 10, 12
QUIT, 31
READ . . . DATA, 18, 39
REM statements, 46
RIGHT$, 40
RND, 34, 35
routes, 8, 9, 16, 17, 40
  one-way, 8, 9
routes array, 17, 18, 20, 31
rules, 3
saving the game, 30, 31
SCORE, 3
  subroutine, 30, 36, 44
scoring, 10, 30
screen presentation, 46
Sinclair (Timex) computers, 3, 18, 21, 33, 38, 40
single-letter commands, 26
sketch map, 6
sounds, 29
Spectrum, 3, 29, 33
  version, 38
spelling, 46
Stanford University, 4
storing the data, 16, 17, 18
string arrays, 16
string data (for Spectrum), 38
string variables, 21, 22
subroutines, 19, 24, 25, 26, 31, 34, 42, 43
  dummy, 23
themes for games, 7
time limit, 28
Timex 1000, 3, 18, 21, 33, 39-45
Timex 2000, 3, 21, 33, 38
tools, 12
treasures, 8, 10, 11
TRS-80, 46
TRS-Color, 29, 46
two-word sentences, 3, 21
useful objects, 10, 11, 12, 13
variable names, 46
variables, 16, 20, 26, 30, 33, 38, 40, 44
  string, 21
verbs, 13, 17, 24, 25, 34, 40, 43
verbs array, 17, 18
verb string, 16, 22
VIC 20, 29, 33, 34, 35, 37
walls, checking for, 27
weapons, 12
word list, 10, 12, 16
word not found in memory, 16
word-splitter routine, 21
word string, 16, 22
writing the program, 19, 20, 21, 22, 23, 24, 25, 26, 27
zero space, use of, 16
ZX81, 3, 18, 21, 33
  version, 39-45

First published 1983 by Usborne Publishing Ltd, 20 Garrick Street, London WC2E 9BJ, England.

Copyright © 1983 Usborne Publishing

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publisher.

The name Usborne and the device  are Trade Marks of Usborne Publishing Ltd.

Printed in Spain by CEDAG S.A., Barcelona. D.L.B. 29678-83.

Usborne Computer Books

Usborne Computer Books are colourful, straightforward and easy-to-understand guides to the world of home computing for beginners of all ages.

Usborne Guide to Computers A colourful introduction to the world of computers. *"Without question the best general introduction to computing I have ever seen."* Personal Computer World

Understanding the Micro A beginner's guide to microcomputers, how to use them and how they work. *"This introduction to the subject seems to get everything right."* Guardian

Computer Programming A simple introduction to BASIC for absolute beginners. *"... lucid and entertaining ..."* Guardian

Computer and Video Games All about electronic games and how they work, with expert's tips on how to win. *"The ideal book to convert the arcade games freak to real computing."* Computing Today

Computer Spacegames, Computer Battlegames Listings to run on the ZX81, Spectrum, BBC, TRS-80, Apple, VIC 20 and PET. *"Highly recommended to anyone of any age."* Computing Today

Practical Things to do with a Microcomputer Lots of programs to run and a robot to build which will work with most micros.

Computer Jargon An illustrated guide to all the jargon.

Computer Graphics Superbly illustrated introduction to computer graphics with programs and a graphics conversion chart for most micros.

Write Your Own Adventure Programs Step-by-step guide to writing adventure games programs, with lots of expert's tips.

Machine Code for Beginners A really simple introduction to machine code for the Z80 and 6502.

Better BASIC A beginner's guide to writing programs in BASIC.

Inside the Chip A simple and colourful account of how the chip works and what it can do.

edc PUBLISHING
Published in the USA by
EDC PUBLISHING, 8141 E. 44th Street,
Tulsa, Oklahoma 74145, USA.

Published in Canada by Hayes Publishing Ltd, 3312 Mainway, Burlington, Ontario, Canada, L7M 1A7.

Published in Australia by Rigby Publishing Ltd, Adelaide, Sydney, Melbourne and Brisbane.

ISBN 0 86020 741 2