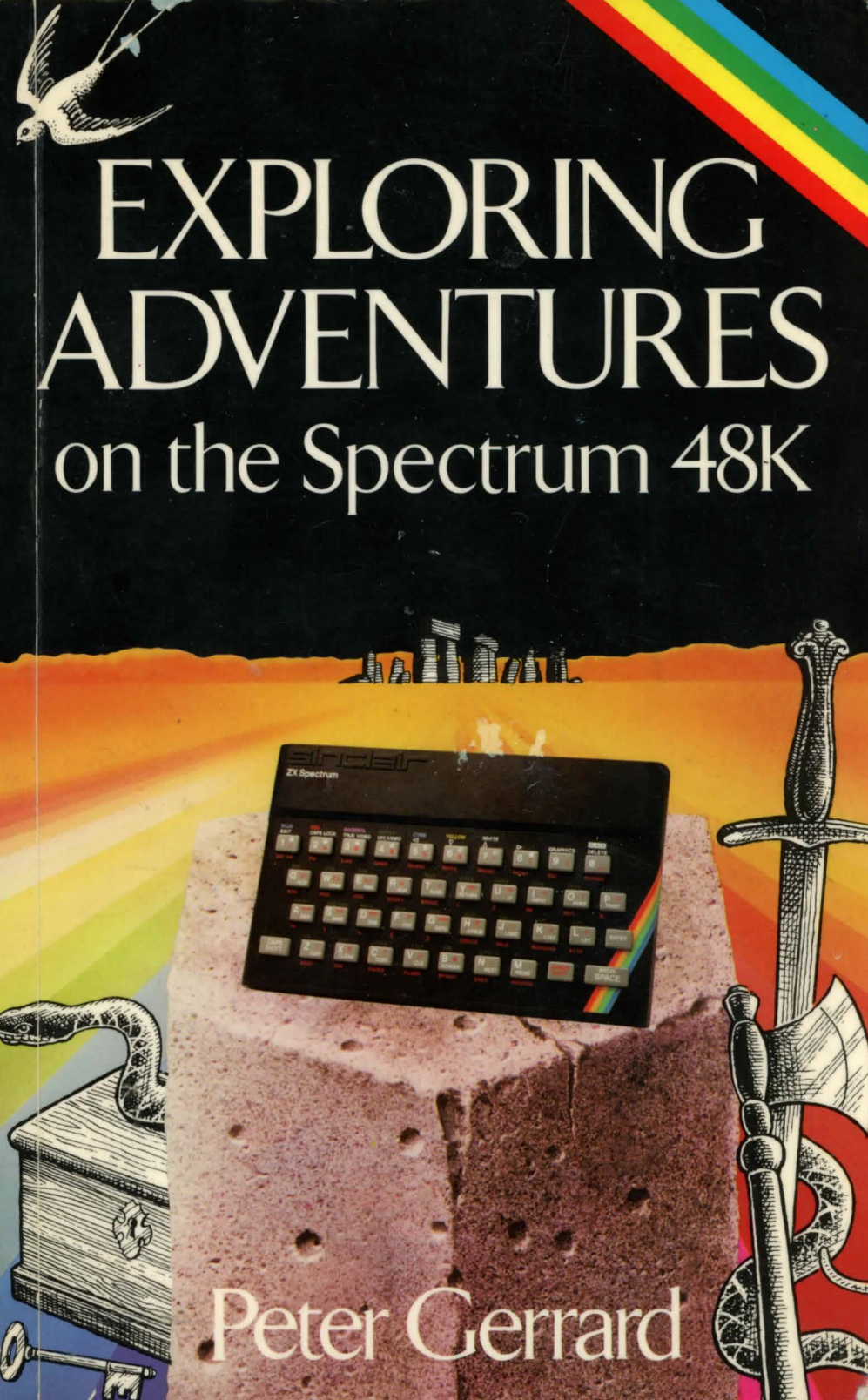
EXPLORING ADVENTURES on the Spectrum 48K



Peter Gerrard

Exploring Adventures on the Spectrum 48K

EXPLORING ADVENTURES
on the Spectrum 48K

Peter Gerrard

Duckworth

First published in 1984 by
Gerald Duckworth & Co. Ltd.
The Old Piano Factory
43 Gloucester Crescent, London NW1

ISBN 0 7156 1796 6

British Library Cataloguing in Publication Data
Gerrard, Peter
Exploring adventures on the Spectrum.—(Duckworth home computing)
1. Electronic games 2. Sinclair ZX Spectrum (Computer) — Programming
I. Title
794.8'028'5404 GV1469.2

ISBN 0-7156-1796-6

Typeset by The Electronic Village, Richmond
from text stored on a Commodore 64
Printed in Great Britain by
Redwood Burn Ltd., Trowbridge
and bound by Pegasus Bookbinding, Melksham

Contents

INTRODUCTION

This book is for anyone interested in the world of adventure games on the Spectrum.

Whether you like to play them, write them, or write about them, this book has been written with you in mind.

More specifically, it is aimed at the person who loves to get absorbed in a game for hours on end, has always wanted to write one of his own, but has taken one look at a listing of someone else's game and thought 'There is no way that I could write something like that!'

This book shows you how to write a fully fledged adventure game, with unique sections on room mapping, data structure, input routines, verb handling, and everything you'll need to know to write an adventure of your own.

The main game in the book, Underground Adventure, is gone through line by line, with each piece of code explained so that you know precisely what is going on.

By the end of this book, you will be in a position to produce your own game for the Spectrum.

Thanks to Steve Darnold, for getting me started in all this (although you didn't know it at the time!).

Thanks also to Jim Butterfield, who gave me my first game of Adventure. And what a game to start with!

Finally, a couple of dedications. Thanks to my wife for doing all the illustrations. Living with her has certainly been an adventure!

And last of all, to the lad with whom I played the longest ever game of Adventure I've played in my life, which probably did more than anything to get me hooked on these games. This single game lasted for about twelve hours, after which time we were still bribing trolls, feeding bears and exploring the bedrock room as we walked to the pub for a pint. Denis Timm, have you managed to get out of the Pirate's maze yet?!

1
An Introduction to Adventure Games

General Introduction

Adventure Games have been played on computers for many years, and are one of the most popular of all types of computer games, if not *the* most popular.

It is sometimes difficult to describe exactly what an adventure game consists of. You're in a magical world of the writer's imagination, doing battle with unknown and often unseen problems, that sometimes appear to defy all logical solutions. You can be placed underground, underwater, in outer space, in colossal caves, or just about anywhere within the known universe, but the ultimate objectives of all the games are usually the same: to survive, and collect all the treasure that is rumoured to exist in these weird and wonderful games.

My own connections with adventure game playing and writing started with the very first game of all - Adventure - playing an abridged version on the Commodore PET 3032 computer, with a 3040 disk drive. One night after a party two of us sat down in front of the computer and, armed with a bottle of whisky in the real world and nothing more than a torch, a bottle of water, a key and some food in the adventure world, began playing a game that was to go on for more than twelve hours!

We simply did not notice that it was now light outside. We were deep underground, trying to cross a bridge with a bear that was too heavy for the bridge, and we didn't care about such commonplace things as sleep!

That early start has led to a lifetime of interest in that game and adventure games as a whole, and my interest in the games is shared by countless other people around the world, who have made this one of the most popular of computer games.

It is hard to explain this popularity to a non-addict. Peculiar looks and pitying stares are the usual response when it is revealed that you spend hours at the keyboard, glued to happenings in an imaginary world.

On the other hand, joining one of the many Adventure user groups will place you amongst many like-minded people who fully understand the frustration at trying to solve a particular problem. 'What *do* you do with the platinum pyramid?!', no longer evokes a 'What on earth are you on about now' attitude, instead you're more likely to get a hundred and one hints and tips on solving the problem of the platinum pyramid.

Adventure enthusiasts even have their own Agony Aunt now in Tony Bridges, who writes a regular weekly column for the microcomputer magazine *Popular Computing Weekly*. Every week he'll take a look at an aspect of adventure playing, or a particular problem in one of the more popular games, and you're welcome to contact him over any problems that you might be experiencing in your own adventure game.

The number of players of these games is legion, and this book has been written to help you write your own adventure programs, and to explain a little bit about the origins of the games, with more than a passing glance at some of the games (and the people) who helped to make this genre of game playing the success it's become today.

We're also giving you three complete adventure game listings at the back of the book, with a full explanation of the Underground Adventure game and how it was written, and brief explanations for the other two.

If the thought of typing in pages and pages of code is a daunting one, you'll be pleased to know that the publishers are also offering these programs on cassette, and that cassette will cost you £7.95, available direct from the publishers.

The listings and sections on programming are all aimed at the Spectrum, using a cassette deck as the storage medium.

It is hoped that, by the end of this book, you'll be more than capable of writing your own games, and perhaps joining the author's Fool's

Gold and Tombs of Xeiops as top-selling adventure programs!

So without further ado, let's take a look at the history of adventure games, and we'll start with the very first one of all, called, simply, Adventure: the game from which all others have taken their generic name.

How It All Began

Although most adventure programs these days seem to be written in Basic, which is the style of writing that we'll be showing you in this book, or machine code, the very first one was written in Fortran, not a language known for its string handling capabilities. Which language you choose is very much up to you, bearing in mind the restrictions of the computer in front of you.

Basic is usually chosen because it's easier than anything else, most Basics have a good set of commands for manipulating strings, and there is no great requirement for speed in this type of game. The essence of these games should always be that you have to think, not act in the frantic fashion of a good arcade game, and because of that we don't have to program everything to happen at lightning speed.

Some adventure games are written in machine code - Zork is a classic example - but the writing of a game like that is beyond the scope of this introductory tome. It is a vast program, usually supplied on three different disks, such is its size.

In Zork, speed is required because of the many and varied ways it can accept the inputting of information from you, the player. Most adventures are restricted to the TAKE STAFF style of commands: one verb and one object, but Zork goes beyond that to the level where you can say something like BURN ALL THE BOOKS EXCEPT THE BLACK ONE, and other complicated instructions.

The first Adventure game used the simple GO NORTH style of instructions: for that game, and for just about everything that's appeared since, credit has to go to Willie Crowther and Don Woods, who wrote the program on a DEC (Digital Equipment Corporation) PDP-10, in , as we've seen, Fortran.

That program required about 300K of computer memory to play it: a great deal more than you get on the Spectrum!

Abridged versions have appeared since then for most of the popular home computers, and it was the work of Jim Butterfield that led to the version now available for all the Commodore range of computers.

Since then, a version has appeared for the IBM Personal Computer, but for some reason it is being marketed commercially. Odd, since it is available free of charge from most user groups!

If you want a copy of that game for your Spectrum, I would suggest getting in touch with one of your local user groups: several names and addresses are given at the back of this book.

If you have never played this, the first ever Adventure program, I would strongly suggest that you do so. Not only is it one of the best adventure games ever written, it is also the origin of every other adventure game. Without it, people like Scott Adams and Greg Hassell would probably have never written their own series of (very good) adventure games.

We'll look at some of theirs later, but for now let's stick to the original.

It is sometimes called the Colossal Cave Adventure, for the opening scenario goes like this: =

'Somewhere nearby is colossal cave, where others have found fortunes in treasure, though it is rumoured that some who enter are never seen again. Magic is said to work in the cave. I will be your eyes and hands. Direct me with commands of 1 or 2 words. I should warn you that I only look at the first five letters of each word, so you'll have to enter "northeast" as "ne" to distinguish it from north. This program was developed by Willie Crowther and Don Woods. This version is abridged for PET disk by Jim Butterfield.'

We'll go into more detail on Adventure (with a capital A to distinguish it from the games as a whole) in chapter 2.

All of this was developed on a mainframe computer with 300K of memory. So how did they get to appear on the microcomputers that we know today ?

The Transition to Microcomputers

The first person to think about putting an adventure onto a small microcomputer was Scott Adams, an American who is commonly ackowledged to be the father of adventure games on small computers.

His story makes interesting reading, and you can find it in the December 1980 edition of the American magazine BYTE, in which there was a special feature on adventure games, and Scott Adams related the story of how it all began.

For the benefit of those who haven't got access to the magazine, here's a brief synopsis:

Scott Adams' first game was written on a Radio Shack TRS-80 level II computer, and came about after he'd already written a few other, non-adventure, games for it.

At the time he was working as a systems programmer for Stromberg Carlson, and he'd been introduced to the original Adventure by a friend. After apparently playing the game for ten days he managed to solve the whole thing, having been totally addicted from that opening scenario given earlier.

However, he realised that not everyone could afford a DEC PDP-10! So, the quest was on to produce a reasonable adventure on a much smaller computer: in his case the TRS-80.

The idea came to him of producing an adventure interpreter. This would allow him to write many different adventures, but at the same time cram an awful lot of information into a very small area of memory.

The programs at the back of this book work along similar lines, in that routines exist within them to move from room to room, store the room descriptions, handle the input of data, and so on, and these routines are common to every listing given. This makes it possible to create adventures with a minimum amount of work from the writer, but at the same time they can be different enough to keep people occupied trying to solve them for many, many hours.

Possibly the most difficult part of writing an adventure, once the actual program structure has been grasped and understood, is getting the original idea in the first place, and working it through as a strong idea that doesn't rely on the impossible happening before the adventure can be solved.

The idea for Scott Adams' first adventure, generally reckoned to be his best, was not particularly brilliant, in that one was doing the usual treasure seeking and problem solving. Nevertheless, it did fit into 16K as opposed to 300K!

After six months of testing his adventure, and of course the interpreter that was driving it all, this first program (called Adventureland) was released through The Software Exchange of Milford, New Hampshire, and Creative Computing Software.

Thus, as he says in his own article, the Scott Adams series of adventures was born.

Apparently it almost died there and then, since his wife was taking great exception to him spending six months locked in a room writing programs! However, all was solved when she decided to write an adventure, and came up with the idea for Pirate Adventure, the second best adventure program he's ever marketed.

In this one the idea is different, in that you have to do slightly more than merely collect treasures and solve problems. You have to build a pirate's ship, and not many people start off with the knowledge to do that!.

And so the transition to microcomputers was complete. It was possible to write an adventure with only a minimal amount of memory, and the market suddenly begain to explode.

The Market Blossoms

Scott Adams has written a large number of adventures now, well into double figures, and we'll be taking a look at some of the better ones later.

But while Scott was doing all his work, there was another, younger, adventure devotee called Greg Hassett, who is now I believe 16 years old, but already the author of at least 8 adventure programs.

Some of his programs, natural enough because of his age, are not worthy competitors to the earlier Adams work, but nevertheless there are some gems to be found from this young schoolboy.

In particular, Enchanted Island Plus is well worth seeking out. Written entirely in machine code (as opposed to his earlier Basic Enchanted Island), solving this one will keep you occupied for a long time to come.

Some of his plots are also refreshingly different. Journeys to the centre of the earth and visiting Atlantis may be fairly run of the mill, but situations where you have to save an almost totally polluted earth from

extinction are much better. World's Edge is possibly the best one that Hassett has written.

Companies producing adventures in these early days tended to be mainly American, and it took a long while before the rest of the globe started producing comparable games, although Britain is now catching up fast.

In those days, Radio Shack themselves started bringing out a couple of adventures, The Programmers Guild took a few pages out of Tolkein's *Lord of the Rings* and had you fighting orcs and spiders, while Mad Hatter Adventures, who started off just handling the Hassett programs, also produced a couple of their own, although these were generally considered to be rather poor when compared to the wonderful program that started off the whole craze.

Since then, of course, many companies have started marketing adventure programs, and now many exist for just about every make of home microcomputer.

Why They Are So Successful

It is true that adventure games generally have captured the computer market to a vast extent. They are one of the most popular types of all computer games, and are now enjoying something of a renaissance, with many new games currently becoming available for all types of computers.

Whilst relatively few will have the long-standing success of the original game, most will probably be worthy of playing, and many will no doubt tie their buyers to the computer for many weeks to come. Tony Bridges is going to be very busy in the months ahead.

But why is there this phenomenal success, and why do so many people spend so long typing in commands on a computer keyboard just to see what appears next upon the screen ?

It is easy to analyse the success of, say, arcade games. The sound effects, the stunning graphics, are obviously pleasing to the human ear and eye, and our society seems to be depressingly heading into a more violent era. Thus the chance to annihilate a few more aliens for a mere 20 pence is not one to be missed.

But adventure games have none of this. There is usually no graphical

display, although we'll see later that games are available that use graphics to one extent or another. Generally, there is no sound being generated by the computer either, although again there are exceptions.

Finally, there is no 'shoot-em-up and zap-em-down' approach to adventures. They are games for the thinker, rather than the person of action.

And perhaps this is part of the secret of their great success. To solve a good adventure like the original Crowther and Woods game requires a lot of logical throught, to say nothing of a lot of time. The first people to start playing the game were computer programmers themselves, and one survey in the States showed that, when an implementation of Adventure appeared on the work's computer, they would lose an estimated two weeks work due to staff playing the game in their free, and not so free, time.

Obviously people tried to put a stop to this, and started restricting access to the game, but it was generally reckoned that whatever a company tried to do, nothing would stop its employees from playing the game. Better to let them have their way for a couple of weeks, and see them emerge contented at having attained the goal of master adventurer.

The same is true for most people who start playing the game. Once you've started, it is virtually impossible to rest until you've completely solved the puzzle.

How can I get past the troll without losing treasure? How do I open the clam? How do I open the treasure chest in the pirate's maze? All these questions have to be solved before attaining the magical status of master adventurer. Sometimes you're setting yourself an impossible task, but that won't prevent people from taking hours trying to solve it, until they give up in disgust.

It becomes a question of pride: 'I am not going to be beaten by a stupid computer!' is the usual response.

Also, pride comes in when you hear of someone else talking about a room, or a particular problem, that you haven't encountered. The desire to find that room, or solve that problem, drives many people back to the keyboard again.

And, strangely enough, you will very rarely get a direct answer when you ask someone how to solve a certain problem. You'll usually get

a cryptic hint, but nothing more. So, you're back to your own logic again, and few people will admit to not being able to solve something.

Finally, adventure games usually have a sense of fun. Take the classic Adventure. The version by Jim Butterfield produces some lovely responses at times. Like this:

FEED BIRD

THE BIRD IS NOT HUNGRY, HE IS MERELY PINING FOR THE FJORDS!

Shades of John Cleese and ex-parrots. If you try typing in the inevitable rude statements, requesting the snake to do the anatomically impossible, again a variety of replies can be generated.

So a combination of problem solving, pride and fun have all contributed to making adventure games required playing for most people.

But what will happen to them in the future, as computers become more and more sophisticated?

A Glimpse into the Future

There will always be a limit to the amount of technology that can be squeezed into a home computer, just because of the sheer size of the thing.

However, there appears to be no limit to the amount of programming talent that can be squeezed into them, and it is this growth of talent that will dictate the course of adventures over the next few years.

We can already see the results of one extremely intelligent set of people in the adventure game Zork, which is in many people's minds a great step forward from the original game.

Again this was developed on a PDP-10, and has now appeared as a three part adventure for a number of home computers. Like a lot of adventures nowadays it is supplied on disk, and thus not everyone will get the chance to play it.

Still, there's always the local user club, and most user clubs have people who are perfectly capable of copying the protected disks on which Zork is supplied! Saying a disk is protected is like waving a red flag

at a bull: sooner rather than later a dedicated programmer is going to crack any form of protection you care to name. As someone once observed: 'You have to have a disk drive to make the protected disk, and you've got precisely the same disk drive as they have. Therefore, you've got the equipment to unprotect the disk.'

I'm not advocating software piracy, by the way, but when you see things like the original Adventure, a public domain program, being sold for anything up to £30, it just invites copying!

So when we get to the stage where all home computers come supplied with built-in disk drives, you can guarantee that there'll be some very sophisticated adventures coming out.

Just as the Crowther and Woods game requires a disk drive, so will many future adventures. Why a disk drive ?

Well, there is a limit to how much memory a computer has, and a disk drive will always have more. Therefore it makes sense to store the core of a program in the computer, and call up the relevant descriptions from the drive.

It will also pave the way for many more graphical adventures. If a computer has got sophisticated graphical capabilities, like many of the current home computers, it makes sense to use them.

However, to utilise all the graphical features on many computers even now can take up to 8K of memory per screen. That's a lot of memory to take up in the computer at one time, and adventures with four rooms in them tend to be solved fairly quickly.

However, hitch up a half a megabyte disk drive and you've got the capacity to handle over sixty rooms. Much more difficult to solve, and as disk drives speed up in terms of access time we can't be too far away from a true animated adventure.

Whether people want animated adventures or not is another question. They say that a picture paints a thousand words, but fifty words can paint a much more graphical image on the mind than an 8K screen display.

That is why *Lord of the Rings*, and other books of that genre, will never make a successful transition to the cinema screen or the home computer. The mind is always capable of imagining far more from a few simple words than can ever be depicted on a screen.

Perhaps that's why the more successful games are always purely textual in their display. Leave it up to the player to imagine it all, and let the computer take care of everything else.

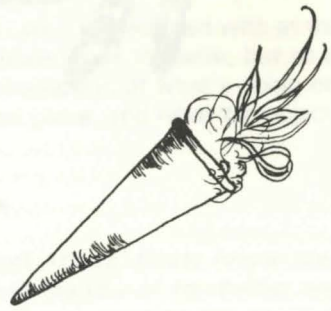
Adventure games that use half-hearted graphics, like the much praised Temple Of Apshai series, from Epyx, tend to be a great disappointment, certainly to this writer anyway.

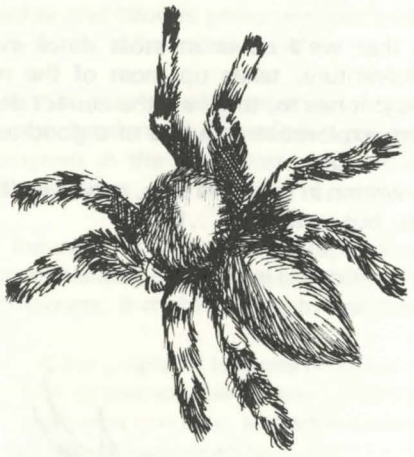
Dungeons and Dragons games in real life are all very well, but the implementation on the home computer hasn't yet arrived.

So in this book we'll stick to textual games with no graphics, on the basis that a) not everyone will have a disk drive, and b) not everyone wants graphics anyway.

The adventure that we'll cover in most detail in this book, the Underground Adventure, takes up most of the memory of your computer anyway: it has to, to give it the correct degree of problem solving and room exploration required of a good adventure.

If this book is re-written in ten years time, maybe we'll be talking about graphical games, but until then. . . !





2

How to Solve Adventures

Adventure Scenarios

Whatever the adventure game that you're playing, you will obviously want to solve all the puzzles presented to you, usually in the minimum amount of time, but if you're a newcomer to the game you'll probably think that the adventure is too difficult and you'll just give up, probably never to play an adventure again.

This chapter is aimed at solving adventures, and as well as some general notes we'll be taking a detailed look at the original Adventure (whilst trying not to give too much away), and also the adventure that forms a large part of this book, the Underground Adventure.

The type of scenario you're presented with at the start of the game will obviously vary from game to game, but as a general rule you'll usually be given a description of what's going on, how you happen to be there in the first place, and what the object of your mission is.

Pirate Adventures

For instance, in Scott Adams' Pirate Adventure you start off in an apartment listening to the roar of the traffic, and only after getting the non-slip sneakers and entering the secret corridor behind the bookcase will you be able to start the adventure properly by saying the magic word and being whisked off to a pirate's island, where you have to build a pirate ship and make your escape.

On the island you'll encounter a wonderfully dotty series of characters, including a drunken pirate and a mongoose, that all add to the charm of this game.

Some of the problems presented to you in various games can at first appear insurmountable. There's one game called Castle Adventure, the object of which is to explore a castle, and make your way safely back with all the treasures.

However, getting into the castle appears to be impossible at first, since it is surrounded by a moat, the drawbridge isn't down, and the moat is full of piranha fish! How do you swim through a shoal of piranhas ?

Sleepy Piranha

The answer is that you don't. You have to roam around outside the castle first of all, finding what you can, and on your travels you eventually discover a set of sleeping pills. Provided that you don't take these yourself you can drop them in the moat, whereupon the piranha obligingly swallow them and go to sleep, thus allowing you to swim across in safety. Of course, you might get your matches wet and soggy in the process, but you had thought of that hadn't you?

Another popular conundrum is the gap in the rocks that is too narrow to squeeze through with whatever you happen to be carrying at the time. The original Adventure has a feature like this, and we've taken that idea and adapted it in the Underground Adventure listing here.

The problem is usually that you can slip through the gap, but nothing that you're carrying can go through with you. As most of these adventures take place underground you require a lit torch to be with you at all times, and if the torch goes out you can't see anything, which means that you just have to blunder around until you fall into a pit and die.

If your torch can't get through the gap, how can you see anything when you're on the other side? The answer usually lies elsewhere in the game, and there will be something that will fit through with you, that begins to glow when you've got through to the other side, thus letting you see whatever happens to be there.

As a final example, Philosopher's Quest for the BBC micro has a delightful problem when it tells you that you no longer have any existence! In other words, you can no longer do anything: if you don't exist, how can you do anything? The answer is one of those horribly obvious ones when you think about it, and that in itself is the answer: if you think, you must exist, as Descartes once said.

Thus by thinking the computer acknowledges your existence, and you can carry on with the game again.

So most adventures follow a fairly standard pattern, although there have been a number of extremely silly adventures that have appeared in recent times, and two of them have both been based on popular television programmes.

There is one adventure based (loosely) on Monty Python's Flying Circus, which has you travelling around on buses, mugging old ladies, and doing all kinds of things in the worst possible taste. Rather like Python itself, really.

Hitch Hiking Around

A second game has now unfortunately been taken off the market, because it was infringing someone's copyright laws. It used to be called Hitch Hiker's Guide to the Galaxy, better known as a radio, television and book series, which found its way into an adventure game by Bob Chappell. All the favourite characters where there, and the plot for this particular game was about as sensible as the series.

However, it did have to be withdrawn, although it has since re-appeared under another name, as a thinly disguised version of its former self.

More usually though, you're exploring caves, or weird haunted castles and houses, and are presented with a reasonably logical set of problems to solve.

Often, these problems will have to be tackled in a specific order, as the solving of one inevitably leads you onto another one that will again have to be solved before you can progress further.

Underground Adventure features this, in that you have to solve some 16 problems before you can complete the entire game, and those problems have to be solved in a set order. In fact, it is usually impossible to progress further if you don't solve them in the right order.

For instance, you can't get past the giant deadly fly until you've found the giant deadly fly-spray, which is itself hidden away behind something else. And so on: solve one problem and you can progress to another.

Some adventures do present almost life-like situations, and your

behaviour has to be judged truly in the light of what you would do if you were actually in that same situation in real life.

Building Ladders

If there's a gap above your head that you can't get to, how would you reach it in real life? Most people would probably go and borrow a ladder, but as adventure games don't usually feature conveniently handy neighbours you're going to have to build one for yourself.

What do you need in order to build a ladder? Nails, wood and some kind of saw are the usual ingredients, so off you go to try and find them all.

Another popular feature is that of having some kind of animal about the place. Bears, snakes and revolting insects are the usual order of the day, and most of them will have two purposes. Bears might eat you alive at first, but tend to calm down when they're fed and perform a number of useful functions.

So the number of possible scenarios is legion, and we can expect just about anything to turn up at one point or another. However, whatever the scenario happens to be you're going to have to solve everything that's thrown at you sooner or later, so let's go about solving an adventure.

Solving Adventures

There are a number of golden rules to be observed when setting out upon a new adventure, and the principal one amongst them must be:

NEVER IGNORE ANYTHING!!

Everything you see will have been put there for a purpose, because writing adventure games on a home computer does restrict the amount of data that can be packed in, and therefore you can't really afford to put in things that will not have a purpose.

Most objects that you enounter will probably only have one role in the game, although this is by no means a hard and fast rule.

In the classic Adventure, you will repeatedly need to keep the axe with you, as little dwarves have a habit of racing out from behind rocks

and engaging you in mortal combat from time to time, and they can only be seen off by throwing the axe at them.

The torch also has to be carried with you most of the time, and in the classic Adventure you have to get a new set of batteries for it after a while, but more of that later.

Although we've said that everything has a purpose, that purpose may only be to annoy and delay you in solving the puzzle.

Life in a Dead End

This is particularly true of some roads and corridors. In Underground Adventure, for instance, there are a number of dead ends. Some of these are purely dead ends and go no further, but others are there to test you, and can be got past.

A giant boulder gets in your way at one point, but can be got round by finding some dynamite, which exists elsewhere in the puzzle, and blowing it up.

Just make sure that you're not carrying the dynamite yourself when you decide to light it, as the only thing you'll blow up then will be yourself.

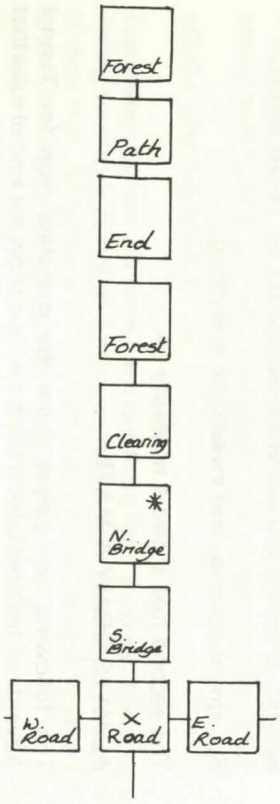
Vast chasms are another popular feature, and Underground Adventure has two of them, which need to be solved in different ways. The Crowther and Woods game also employs a chasm, and if you're carrying the black rod with you when you encounter it you should be all right, provided you can work out the correct verbal syntax.

So, ignore nothing, and investigate everything.

The second rule is also a necessity : =

ALWAYS DRAW A MAP!!

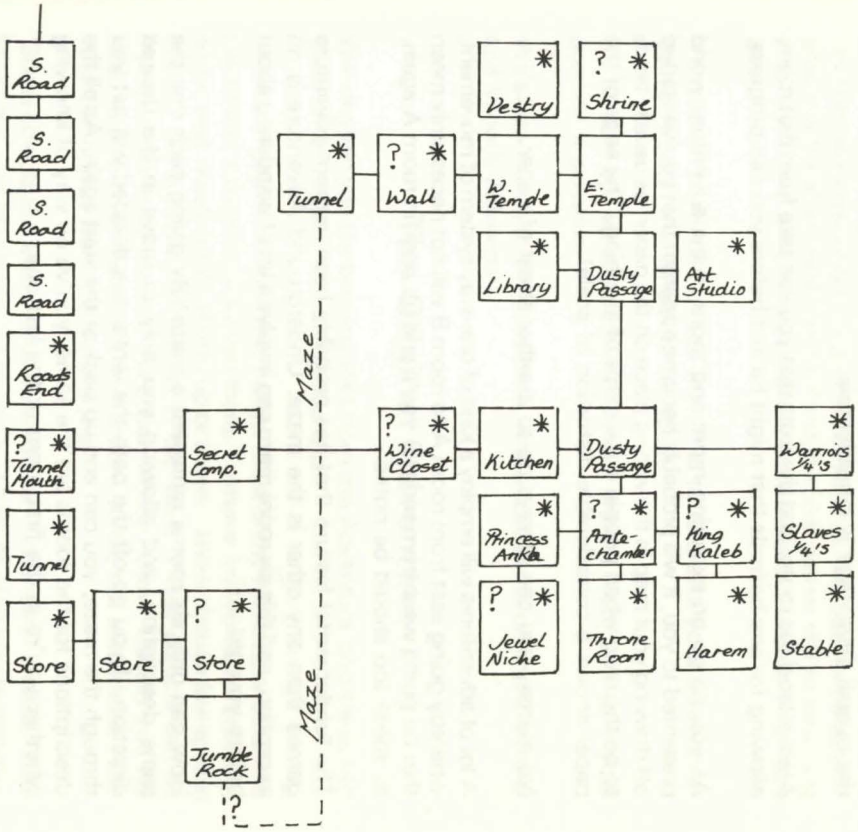
The following two pages show the complete map for Tunnel Adventure, featured later in this book, and show the kind of rules that should be obeyed when drawing a map of your own.



Tunnel.

? _ problem to solve

* _ useful objects / treasures



Drawing a map

Drawing a map will usually speed up your adventure solving process considerably, as it will save a lot of roaming about simply covering the same ground all the time. It will not take long to draw, and thus the overall advantage is considerable.

Always label the room, and the exits that you can take from that room, allowing for any hazards that might be in it before you can progress.

As you do go on exploring more and more of the adventure world presented to you, it will probably become apparent that you've started off drawing your map in the wrong place on the paper, as usually tends to be the case when people draw maps of anything. The edge of the paper shows a considerable distortion of scale!

No matter, you can always go to another sheet of paper.

A lot of adventures will employ a kind of one-way system of movement, whereby going east from room A to room B will not necessarily mean that by going west from room B you'll end up back in room A again, so these too should be noted.

The fundamental feature that has probably been in more adventure games than any other is the maze. Underground Adventure is no exception, and this six room maze can involve a lot of wandering about before you get out.

How can only six rooms represent a maze? By giving each one the same description, and allowing you only to move in the desired direction, if you go off the path the writer has decided will get you through the maze, you can end up back at the start again. As all the descriptions for the rooms are the same, you've no way of knowing whether you're in the first room or the last one.

Make a careful note of the directions you've gone through as you wonder through any mazes. You'll get out in the end, and if you haven't remembered the route it would be a little annoying to encounter the maze again in a later game.

Drawing maps can be fun, and it's useful to note down the initial positions of any objects that you find during play. Some adventures do have a random distribution of objects, but more of them do not,

as the solving of many puzzles depends on finding the correct object in the correct room, and if it isn't there then the game becomes unsolvable.

Finally, if you're playing a game with a LOAD and SAVE feature that allows you to store your current position onto tape for later recall, it's worth saving a game if you're about to do something particularly cavalier, like attacking a dragon or something. The odds are that your attack will end in death, and although some games will re-incarnate you, you'll end up a long way away from where you were when you died.

It's quicker to re-load a tape than it is to re-create your position by going through the whole game again.

So to sum up, ignore nothing, always draw a map, and save your position if possible.

We'll now put all this into practice, with a look at the original Crowther and Woods Adventure.

The Original Adventure

We've given you the opening lines from this Adventure, and the screen goes on to display something like this:

'I know of places, actions and things. Most of my vocabulary describes places and is used to move you there. To move, try words like building, enter, east, west, north, south, up or down. I know about a few special objects like a black rod hidden in the cave. These objects can be manipulated using some of the action words I know. Usually you will need to give both the object and action words, but sometimes I can infer the object from the verb alone.

'Some objects also imply verbs. In particular, "Inventory" implies "Take Inventory", which causes me to give you a list of what you're carrying.

'The objects have side effects. For instance, the rod scares the bird. Usually people having trouble moving just need to try a few more words. Usually people trying unsuccesfully to manipulate an object are trying something beyond their (or my!) capabilities and should try a completely different tack.

'To speed the game you can sometimes move long distances with a

single word. For example, "Building" usually gets you to the building from anywhere above ground, except when lost in the forest. Also, note that cave passages turn a lot, and that leaving a room to the north does not guarantee entering the next from the south. Good luck!'

And finally, you get one last piece of help before being thrown into the game proper:

'Maximum points are earned by leaving treasure in the building. It also helps to get back out in one piece.

'If you think you have found all the treasure, keep moving around until something happens.'

And So We Begin

And with that, the game will begin, and you find yourself in a building known as the well house (since it contains a well!), which houses a number of useful objects like a torch, a bottle, some food and a key.

From the building it is but a short walk to the forest, which is very easy to get lost in, and then the real route into the heart of the game takes you south down a narrow ravine until: =

'You are in a 20 foot depression floored with bare dirt. Set into the dirt is a strong steel grate mounted in concrete. A dry stream bed leads into the depression.'

Opening the gate with the key provided in the building lets you into an underground set of passages, starting off with: =

'You are in a small chamber beneath a 3 by 3 steel grate to the surface. A low crawl over cobbles leads inward to the west.'

Of Black Rods, Birds and Cages

Nearby you can find a black rod, a bird cage and the little bird itself, and your first problem solving comes in actually getting hold of the bird, since it isn't too fond of the rod. You'll also have to light the torch by now as well, as it gets dark this far underground. The torch is, in fact, an electric lamp, and it will sooner or later start running down the batteries you started with. However, you are given the helpful message:

'Your batteries are starting to run low. Better wrap it up soon, unless you can find new ones. I seem to recall that there's a vending machine somewhere in the maze.'

Finding the vending machine in the maze is no easy task, and even when you get there you must be armed with a set of coins which are to be found somewhere within the game, otherwise you won't be able to insert the coins to get the fresh batteries contained in the machine!

But back to the bird, the rod and the cage. Wandering on a little soon brings you to the first major room description of the game, which is when you start to realise why this game is a disk based one: some of these room descriptions can get quite long!

'You are at one end of a vast hall stretching forward out of sight to the west. There are openings to either side. Nearby a wide stone staircase leads downward. The hall is filled with wisps of white mist swaying to and fro almost as if alive. A cold wind blows up the staircase. There is a passage to the top of a dome behind you.'

Round about here you will also encounter a snake, which bars your way and refuses to let you pass.

Snaky Problems

Feeding animals is the usual way to calm them down, but attempting to feed the snake is not a particularly good idea, especially if you're carrying the bird at the time, since the snake eats the bird and then just sits there looking at you, still refusing to let you pass.

You can solve that one for yourself!

Round about here, you have a choice of routes, and one of them leads off across the floor of the hall as far as the aforementioned vast chasm, which is where the rod comes in useful. Going on from there will take you towards the maze with the vending machine in it via a back entrance, but it will also take you near another maze as weil, which is significantly more difficult to get out of.

It also contains something a lot more interesting, but we'll come to that one later.

Going off in another direction leads you to the mysterious Y2 room, and nearby lies the equally mysterious bedrock room, which allows

you to explore around at random.

From Y2 you can go to another one of the game's fine room descriptions, which is one of the more puzzling points on the route for beginners to the game:

'You're at a low window overlooking a huge pit, which extends up out of sight. A floor is indistinctly visible over 50 feet below. Traces of white mist cover the floor of the pit, becoming thicker to the left. Marks in the dust around the window would seem to indicate that someone has been here recently. Directly across the pit from you and 25 feet away there is a similar window looking into a lighted room. A shadowy figure can be seen there peering back at you.'

Who, or what, is the shadowy figure?!

Dwarves and Pirates

From here we have a variety of routes, but by now a couple of things will probably have happened. One is that you will almost certainly have encountered a dwarf: =

'A little dwarf just walked around a corner, saw you, threw a little axe at you which missed, cursed, and ran away.'

Charming!

And the other is a bearded pirate, who lurks about the caves, and who will occassionally appear and steal all your treasure:

'Out from the shadows behind you pounces a bearded pirate! "Har Har", he chortles, "I'll just take all this booty and hide it away with me chest in deep in the maze!" He snatches your treasure and vanishes into the gloom.'

Since some of the treasures have a useful function to fulfil, as well as just being valuable and scoring points when you get them back out to the building, this can be mighty inconvenient!

One of these dual purpose treasures is a trident, which lurks away near the bedrock room. As well as being jewelled, it will also enable you to solve one of the game's more puzzling features.

Mysterious Bivalves

Near Y2 there lives a giant clam, although we later find out that it is in fact an oyster. The program cheerfully tells us that it was never very good at identifying bivalves after this little bit of mistaken identity. Being an oyster, it will probably contain a pearl, and so you attempt to open the clam without success.

You can carry it about with you if you want to, although it is a little heavy, but you won't be able to open it until you find the jewelled trident, which is hidden away in a secret set of rooms, which are themselves reached via the two pit room, or twopit room, as one acquaintance used to call it.

In the two pit room is a plant, and like all plants it likes being watered. Water it enough and it will grow and grow until it reaches the height of a hole way above your head. You can then climb the plant and get into this new set of tunnels and corridors, until you realise that your progress is halted once again as you run into an old rusty door that needs oiling.

Oh well, there is some oil in here somewhere, so having found that you can then get past the door and find the jewelled trident. You'll have to get away from there then, which is none too easy, but can be accomplished.

A Breath-Taking Description

One of the longest of all room descriptions is to be found round about the low room, near bedrock, and is worth repeating here in full just to show you the sort of advantages disk-based systems can give you over programs stored purely in memory, in terms of the use of text to illustrate graphically what a room looks like : =

'You are on the edge of a breath-taking view. Far below you is an active volcano, from which great gouts of molten lava come surging out, cascading back into the depths. The glowing rock fills the farthest reaches of the cavern with a blood-red glare, giving everything an eerie, macabre appearance. The air is filled with flickering sparks of ash and a heavy smell of brimstone. The walls are hot to the touch, and the thundering of the volcano drowns out all other sounds. Embedded in the jagged roof far overhead are myriad twisted formations

composed of pure white alabaster, which scatter the murky light into sinister apparitions upon the walls. To one side is a deep gorge, filled with a bizarre chaos of tortured rock which seems to have been crafted by the devil himself. An immense river of fire crashes out from the depths of the volcano, burns its way through the gorge, and plummets into a bottomless pit far off to your left. To the right, an immense geyser of blistering steam erupts continuously from a barren island in the centre of a sulphurous lake which bubbles ominously. The far right wall is aflame with an incandescence of its own, which lends an additional infernal splendour to the already hellish scene. A dark, foreboding passage exits to the south.'

Wow! Try getting all that into a single picture on the screen. The mind can imagine far more readily what a place looks like from a description like that than it can from a poor graphical illustration on the screen.

Round about here you can also find an extremely narrow crack that you can't get down with anything that you happen to be carrying, and also a troll, who is not too fussed about eating, but who does have a streak of avarice in him.

Trying to attack him produces the response : =

'Trolls are brothers of the rocks and have skin like that of a rhinoceros. He fends off your blows effortlessly.

Even if you try throwing an axe at him, all you'll get is : =

'The troll catches the axe, examines it, and tosses it back to you saying, "Good workmanship, but not very valuable".'

A tricky customer the troll, and you'll have your work cut out to get around him without losing too many points.

On the other side of the troll is another set of passages, including the breathtaking view described earlier, and also including a bear, who can be bribed with some food, and who can then be used to scare away the troll when you want to get back across the bridge again.

However, the bear is heavy, and the bridge is old, and the inevitable happens ... you plunge to your doom on the rocks below.

And so the game continues, through many different rooms and with many different problems to solve, and space dictates that we can't mention them all here. Even with what we've already told you, there's

more than enough in this game to keep you occupied for a long time yet!

But one final feature does deserve mention, and that is the end of the game itself, after you've found all of the treasures and taken them back to the building.

The End Game

As you wander about the caves, convinced that there's nothing more to find, a sepulchral voice booms out and tells you that the caves are closing, and you'd better leave by the main exit.

But where is the main exit?

So off you scurry to try and find a way out, but always its too late, and the caves close! As they do so, mysterious forces snatch your keys out of your posession, and a few other items as well for good measure, and you find yourself:

'. . . at the northeast end of an immense room, even larger than the giant room. It appears to be a repository for the "Adventure" program. Massive torches far overhead bathe the room with smoky yellow light. Scattered about you can see a pile of bottles (all of them empty), a nursery of young beanstalks murmuring quietly, a bed of oysters, a bundle of black rods with rusty stars on their ends, and a collection of brass lanterns. Off to one side a great many dwarves are sleeping on the floor snoring loudly. A sign nearby reads DO NOT DISTURB THE DWARVES! An immense mirror is hanging against one wall, and stretches to the other end of the room, where various other sundry objects can be glimpsed dimly in the distance.'

And if you get out of that room? You enter this one : =

'You are at the southwest end of the repository. To one side is a pit full of fierce green snakes. On the other side is a row of small wicker cages, each of which contains a little sulking bird. In one corner is a bundle of black rods with rusty marks on their ends. A large number of velvet pillows are scattered about on the floor. A vast mirror stretches off to the northeast.

'At your feet is a large steel grate, next to which is a sign which reads TREASURE VAULT. KEYS IN MAIN OFFICE.'

And what happens then? Well, you'll just have to play it all and find out for yourself!

We've given a lot of exposure to Adventure here, because it was the first serious adventure game, and holds many a fond memory for everyone who's ever played it, whether on a PDP-10, or a Commodore PET.

It also contains most of the ideas which have influenced other adventurers over the years, and as such is more than worthy of its place here.

Try your local user group if you're thinking of getting hold of a copy. It'll be worth it, but you won't get much sleep after you've got it.

But since Adventure, there have been many others to solve, so we'll take a look at some of those now.

Other Adventures

The other main contender in the adventure game stakes is obviously Scott Adams, who's done so much to popularise these games on microcomputers.

We've taken a brief look at some of his games earlier on in this book, but to go into a little more detail on some of them, we'll start with the very first one he wrote, Adventureland.

This is a very natural romp, in that most of the features you encounter are perfectly natural, such as bogs, lakes, and a tree (which must become a tree stump before you can get very far into the game), as well as the nasty chiggers. Nasty what? Look it up in the dictionary!

It's all very lighthearted, and a nice sense of humour runs throughout the game. A good starting point for anyone who's fairly new to the adventure world, as solving it is not too complicated. Nevertheless it should keep you entertained for quite a while.

As will the second Adams adventure, Pirate Adventure, with a story line developed by his wife. This one probably more than Adventureland, set the standard that Adams was to adhere to throughout his game writing series.

There are four main locations for this adventure, including a London

Apartment, an Island, a Treasure Island, and Never-Never Land. It was one of the first games of this genre to give you a mission other than pure collection of treasures, in that you have to work out how to build a boat!

Along with some of the characters who inhabit this world, such as the parrot that keeps shouting 'Pieces Of Eight', and who does give you some helpful hints along the way, this is a nicely humourous game.

Mystery Fun House came next, and differs from the usual run of the mill games by taking place in a carnival fun house. All sorts of problems to solve, and many, many corridors to explore, and this was the first Adams adventure to pit you against a time limit, as well as all the other problems.

Mission Impossible has appeared on more computers than possibly any other Adams adventure, and is one of the most difficult ones that he's done. It's also a mission adventure, rather than a treasure collecting one, in that you're on a race against time (as in Mystery Fun House) to try to stop a nuclear reactor from being destroyed by unknown enemies.

Spaced Out

Strange Odyssey is set in another world altogether, as it starts with you all alone on a strange planetoid, with only a shattered spaceship and your own skills as an adventurer to protect you.

Many outer space games have appeared over the years, and in a brief aside we'll take a look at a couple of non-Adams ones, starting with A Stellar Trek.

This is another version of the final frontier, where you boldly go where no computer has gone before, you are in command of the starship Enterprise, and have the simple task of defending the galaxy against the threat of the invading Klingon empire and their friends the Romulans.

This is more of a role-playing game than the true textual adventure, in that you must begin by selecting your crew and adopting various tactics that will stick with you throughout the game.

None the less, our basic rules of ignoring nothing and drawing maps still apply, although as we'll see in another game there are instances

where examining everything in sight can lead you into great trouble!

This is basically a graphical game, and some may not find it to their liking if they're aficionados of the real thing. Still, an enjoyable and frustrating game, that should keep you out of trouble for a while.

Two other games that can dubiously be described as fitting into the adventure world, although really they are more at home with the Dungeons and Dragons fanatics, are Starfleet Orion and Invasion Orion. These are war in space games, with a lot of tactical planning and craft maneouvering going on, and so don't really belong as true adventures. But, like A Stellar Trek, they should keep you amused for a while.

Back to Normal

A new venture for Adams was away from space and into the world of vampires and other assorted nasties.

In The Count you are out to rid the world of Count Dracula once and for all, and, in the best traditions of ancient horror movies, you must race against time to catch the count in his human form before driving the stake home and removing him from the planet.

Voodoo Castle is set along similar lines, with you involved in an attempt to save the cursed Count Christo, which sets you off exploring the hallways and dungeons of Voodoo Castle. An entertaining game, with voodoo dolls, a juju man, and more.

The final two we'll mention from Adams are again set in two totally different worlds, with Pyramid of Doom taking you to an unexplored pyramid somewhere in the depths of Egypt. This is one of the more difficult Adams adventures, and many would say the hardest one he's ever put together.

When you begin writing your own adventures, you'll find that one of the most dfficult things to judge is precisely how difficult you're going to make the game.

Since you control the rooms, the objects in them, and the problems that have to be solved, the game can effectively be made as easy or as difficult as you like. As you're going through it, you may well find yourself thinking that this is a very easy game, and no-one would ever have any problems solving it.

Well, remember that other people haven't got access to your maps, your route diagrams, your list of objects and their original locations, and so on.

The easiest solution to this is to get an adventure playing friend to come around, once you're satisfied that the adventure is complete and bug-free (it won't be, of course - your friend will type in omething you never thought of, and the computer will be equally as stumped), and have him sit down and play the game, while you hover nervously in the background.

From his reports, you can then modify the game, making it more or less difficult, depending on how it's all gone.

The Wild and Woolly West

In an adventure theme that hasn't seen too much experimentation, although Lost Dutchman's Gold comes near the same area, the last of the Adams games, Ghost Town, sets you in an American ghost town that has you expecting John Wayne and Audie Murphy to put in cameo roles.

Good fun, as you encounter saloons, jails, boot hill, piano playing ghosts, and a whole collection of ludicrous characters, this is a suitable Adams game to bow out with. A very enjoyable game.

There are plenty of other games out there that are worthy of exploration, but for our last one in this section we'll take a look at the game that's been described as being as much of an improvement on Adventure as Adventure was on Wumpus.

What's Wumpus? One of the most boring computer games of all time, where you have to walk around a few (typically 24) rooms trying not to bump into the Wumpus, an amiable beast who likes to spend most of his time asleep. A few arrows can be fired now and again, but overall it does not rate very high on the entertainment stakes.

To say a game can improve on the original Adventure by that much is a bold claim, but Zork has captivated everyone who has ever played it.

Now in three parts, sold on three separate disks, each part is a unique adventure in its own right, and pits you against wonderful problems in weird worlds, but with a number of great improvements over that original game.

Zork: the Greatest Adventure?

Zork was the brainchild of four people: Marc Blank, Tim Anderson, Bruce Daniels, and P. David Lebling, and (like our original Adventure), was written on a PDP-10.

However, as Zork grew and grew it began to run out of memory space even on that computer (at the time a giant megabyte, but that doesn't look too much now), and they decided to completely re-write the game for a microcomputer.

A strange decision? Well, not really, because most microcomputers even then had disk drives, and now of course these disk drives are growing in capacity.

However, to re-write Zork in order to make it all fit was no easy task. It might be possible to fit all the data and text required for the game onto a single disk, but what about the program to manipulate it all? Even the original Adventure control program takes up about 13K without running it, and as you probably know, as soon as the program is run, various variables are declared that take up even more memory space.

So Zork had to undergo a few drastic changes.

The first of these was to write a Zork-language, which could be swopped from machine to machine merely by changing that language to suit the machine, and then write all of the program in the Zork-language.

In other words, just as all micros require a different Basic interpreter, so the Zork interpreter swops around from machine to machine. However, the rest of Zork can remain the same, and so the actual work-load on the authors was considerably reduced. Only the interpreter had to keep on being re-written, and now exists for just about every popular make of computer.

The complete story behind all this can be read in a very interesting article in the July 1980 issue of *Creative Computing*, called 'How to fit a large program into a small computer', which was co-written by one of the authors of Zork, Marc Blank.

Having crammed Zork into a small machine, it was now available to many people, and some of its features are truly amazing.

The ability to say more than just DROP BOMB for instance, which can now be said in a variety of ways. For example, TELL THE ROBOT TO PUT THE BOMB ON THE SHELF, and other variations, do much to add to the power, and ease of use, of this game.

Such control over the vocabulary is beyond the scope of this book, although we will be taking an extensive look at string handling on the Spectrum in chapter 3.

Suffice it to say that if you can get hold of a copy of Zork, do so! We've given you a few addresses at the back of the book.

But is Zork the ultimate adventure? With graphical and role-playing games coming more to the fore, let's take a look at some of those, and see if we can guess what will happen over the next few years.

Graphical and Role-Playing Adventures

We've already talked about graphical adventures of the future in an earlier section, and will end our discussion here with the same sort of conclusion as was reached then: not many people want to see fabulous displays on the screen, when fabulous descriptions can conjure up far more in the mind of the player and his alter ego as they wander about the universe created for them.

Instead, the future would tend to lie in the direction of role-playing games, best personified by the original Dungeons and Dragons games, and their variants such as Tunnels and Trolls, Traveller, and the countless other board games that have sprung up since the first game appeared.

In these games there is one great difference over the classic Adventure/Zork scenario: you adopt a character role, rather than just taking on the one that the computer conjures up for you, and your success or failure in the game depends to a very large extent on th type of character adopted.

In Adventure, you know that if you get the bear you can always get back past the troll again, and escape over the rickety bridge to (comparative) safety, but if the same situation were to occur in one of these games that might not always be the case. Your character might be unfriendly towards the bear, and the bear would bite your hand off, or some other dire fate might befall you.

Again, in Adventure, a fight with a dwarf will always have one of two options. You will either win and emerge unscathed, or lose and die.

A fight in a D and D game could have a number of different outcomes, as well as the two simple ones outlined above. You might win the fight, but suffer a gaping wound that leaves you temporarily below your best: an easy victim for the next antman who comes along.

So that is the chief difference: the games are more varied, and indeed one could argue a strong case for there being an infinite number of variations contained within the same game.

However, these advantages are not gained without some other advantages being lost.

In Adventure and Zork you have a vast vocabulary at your disposal (Zork can handle over 600 different words, with about 100 verbs to be used), but in D and D games you're usually restricted to a much smaller number. This is typically of the order of 20 commands, or even less: the much rated Temple of Apshai has a very small vocabulary indeed.

Still, you do have the option of choosing a character who is much more to your liking than a simple 'You are', appearing on the screen every time. It is far more satisfying to watch 'Pete The Great' stalking about the screen (or whatever you would choose, of course), and for some reason it seems to make the game a lot more realistic if you know that it is more specifically YOUR success or failure in the game that's at stake.

From Boards to Computers

One of the less attractive attributes of Dungeons and Dragons is that it takes a referee to make sense of it all, and bribing the referee, all part and illegal parcel of playing the game, has been known to sway many an outcome. The real life 'I'll buy you a pint when we've finished' is far more likely to influence the referee than a simple 'How about 20 gold pieces then?' whilst in the middle of a game.

Also, the referee's job is not an easy one, as most actions have to be decided by a concentrated study of maps, charts and rules, and thus a simple fight between two protagonists could take as long as half an hour, or even more, to resolve.

In computer simulations of these games the computer becomes the referee, and the screen the board on which all the action takes place, and as these games are always played in real time that action can sometimes be decided very rapidly. Our half-hour fight could be over in ten seconds, and it's back to the keyboard in a hurry to see what damage you and your trusty sword have suffered in the duel.

Character Traits

Your character in these games is determined by six factors: three mental (ego, intelligence and intuition), and three physical (dexterity, strength, and constitution).

In the old days these were decided by rolling three dice and adding up the spot scores, and thus any one attribute could range from a low score of three to a high score of eighteen. On the computer, you can usually choose from a total score and divide that score up amongst the six attributes, and since our scores can range from 3 to 18 for each one (or 16 different possibilities), we can create a massive 16 to the power 6 different characters, or over 16 million!

Since we are creating each character as we go along, we can also bring in characters from other games, which helps to explain the popularity of this type of game. If you've survived an exhaustive game of Dungeons and Dragons as Denis The Unsteady it helps to have the same character with you next time you set out to play The Curse of Ra, or whatever.

These six differing character straits interact subtly throughout any one game, and the final outcome of that game always depends on the role you have adopted. A highly intuitive character will find secret trap doors with ease, whereas one with low intuition would only find them by falling into them. Similarly, a high ego would keep going when the going got tough, but a low ego would probably cry and ask for his mum!

And so it goes on, with each attribute perhaps altering slightly as the game progresses and you discover magic potions, bargain in the Apothecary Shoppe, and in any one of a hundred different situations.

Perhaps this is the true way forward for adventure games in the future, and increasingly role playing will play a dominant part in this type of game.

The Ideal Way

The ideal would be to have a combination of the traditional textual adventure with a character playing role as well, as graphics are largely redundant in these games. Thus one would keep the advantages of a large vocabulary, a large number of locations, and a large number of hazards and problems to solve, whilst at the same time having a multitude of variations on the same game by being able to pick your character from one of the 16 million mentioned earlier.

But that must wait for another time, and for now we'll turn our attention to the game Underground Adventure, which will be featured heavily throughout this book, and we'll begin by explaining what it's all about.

Underground Adventure

This is a classic text-only one character game, because for writing your first adventure I don't feel that we should be too optimistic. You may well, after reading this book and understanding everything that's going on, want to go on and develop extremely complicated games, and if you do then the purpose of this book will have been achieved.

But for now, we'll describe a simple, straightforward adventure that is (I hope!) a lot of fun to play and solve.

Underground Adventure starts you off outside a series of caves, with dire warnings about the punishments that await anyone who enters. But, being a brave young lad with a heart for adventure you merrily march off into the caves, take three steps inside and CLANG! A massive gate falls shut against the entrance to the cave, and from then on it's a question of roaming around trying to find the key that will enable you to get out again.

There are a total of 16 problems to solve in this adventure, and I've tried to give you a feeling for the real thing by including a number of scenes that will be familiar to anyone who's ever played an adventure before. In later chapters we'll explore the actual writing of those scenes, and show that it is all possible in Basic, but for now we'll content ourselves with simple descriptions.

You start off immediately with a choice of three routes, heading either east, west or south. North is closed off to you because of the fallen gate.

To the east lies a massive underground tree, which completely blocks your path, so you know that one of the things you'll have to do is to find a way of getting past that tree. What do most people do when they want to remove a tree? They either drag it away or they chop it down, so you know you're looking for either some haulage equipment (unlikely in an underground cave), or something like a sword or an axe.

To the west, your path is blocked by an extremely large boulder, that fills up the whole path and prevents you from going any further. To get past this, you might first of all try pushing and pulling at it, or even attempting to pick it up, but the stone is too heavy for you to move that easily. So, again you must ask yourself the question 'what would anyone do when they wanted to remove a large boulder'. Well, again one could haul it away, but that seems a little unlikely. There could be a large animal around somewhere that might move it for you, or perhaps you'll need to blow the thing up with some dynamite. Of the latter two options, one is the correct one, so we keep an eye out for either a large animal or a keg of dynamite. Beware of large animals though: most of them are not very friendly on the first encounter.

To the south, all we can find is a vast chasm, but en route to it we've already picked up an iron staff, amongst other things. Examining the staff reveals that 'it has some useful properties', so we know that the staff is capable of solving something. Since we can't go west, east or north without finding yet more objects it's reasonable to assume that we'll have to go south somehow. Attempting to jump the chasm is not very rewarding, and in fact leads only to your death, so perhaps if we wave the staff...

Ah, perfect, and a bridge now spans the chasm. Good, we can now head south into the heart of the caves and see what we can find. If we're unlucky, a living gargoyle will appear, and throw a knife or two at you, and he must be engaged in combat before moving on, otherwise he follows you everywhere, continually throwing knives, and one of them may find its mark. How to kill a living gargoyle? Well, there must be something dangerous around somewhere, and sure enough we find an axe eventually. A sure throw with the axe finishes off the gargoyle (temporarily), and we remember that an axe was one of the things we were looking for, as a possible means of chopping the tree down.

Back across the bridge, chop the tree down, and we find some rope, which must come in useful somewhere for climbing up or down something, and a golden bear, who appears none too friendly. Obviously the bear must be calmed down somehow, but how ?

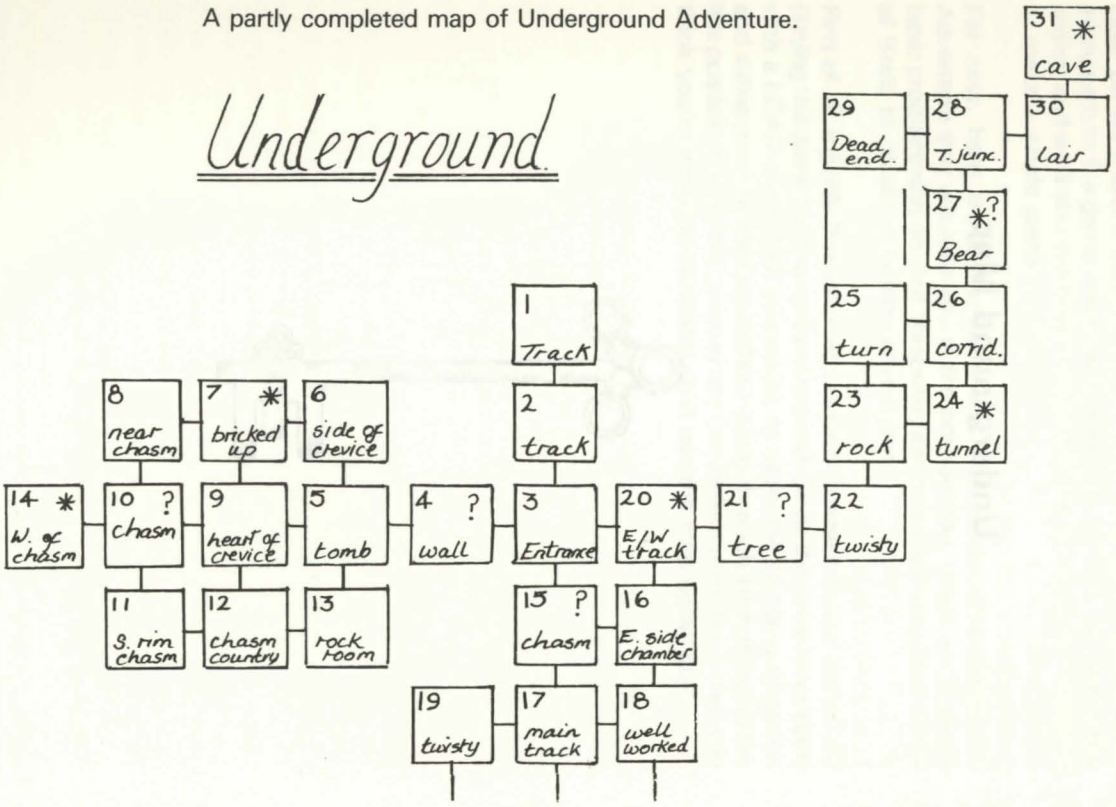
That one, and a few other problems, we'll leave up to you, but you will have begun to get the idea of solving this adventure. Everything is there for a reason, and solutions to problems are usually quite logical. Later on we'll take a very thorough look at this game, and analyse every verb in the game and how it works, along with the rest of the listing, and we'll also see how each part of the listing comes together to make a whole game.

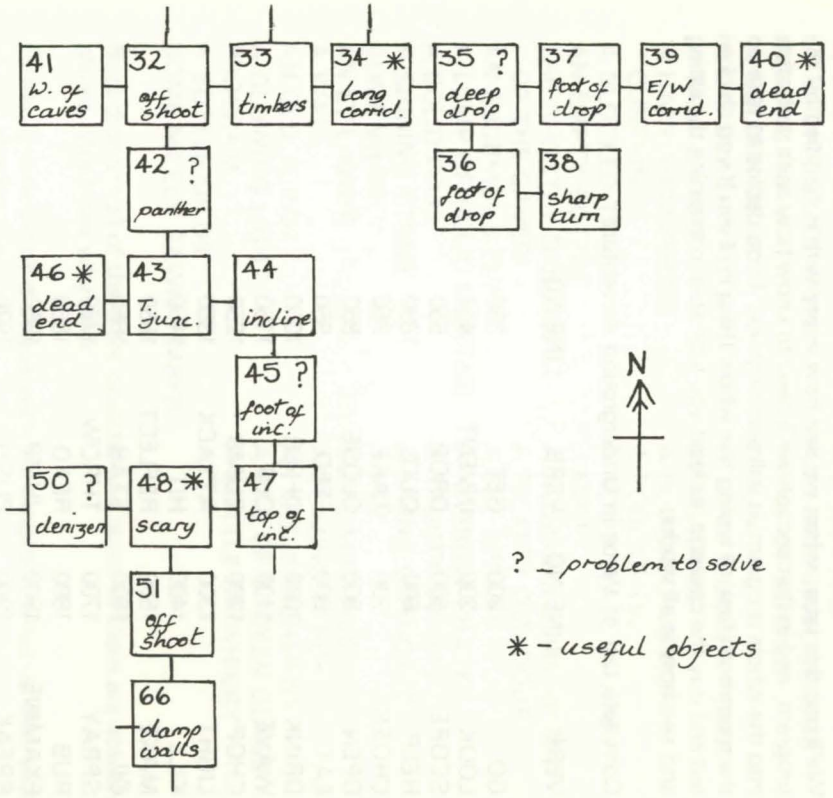
For now, here are a few facts and figures about Underground Adventure that will help you in the next section, when we come to basic programming on your computer, and how we'll use a knowledge of Basic to go about writing adventures.

First of all, a partly finished map drawn by someone who started off playing this game but then ground to a halt. Since the game is equipped with a LOAD and SAVE procedure, to allow the stopping of games and subsequent re-starting without going through everything again, it is possible to stop this game at any convenient point (i.e. when you think you're about to be killed), and use the map again later.

Underground Map

A partly completed map of Underground Adventure.





List of Verbs

We'll use this later, when we see how every verb is handled by the program, and in that section we'll need to know how each verb slots into the whole program. It will also help you if you decide to take on the mammoth task of typing this whole thing in! Even if you chicken out and buy the cassette, at least you'll be able to examine the listing and see how it all works:

Complete List of Verbs in Underground Adventure

VERB	LINE NO.	VERB	LINE NO.
GO	300	GET	350
LOOK	200	INVENT	450
SCORE	500	DROP	550
HELP	600	QUIT	1890
CROSS	700	TAKE	350
OPEN	800	CLOSE	850
EAT	900	FEED	950
DRINK	1000	OFFER	1050
WAVE	1100	CUT	1150
CHOP	1200	CLIMB	1250
LIGHT	1300	ATTACK	1350
KILL	1400	HIT	1450
MAKE	1500	REFLECT	1550
OIL	1600	STAB	1650
SPRAY	1700	THROW	1750
RUB	1800	READ	1850
EXAMINE	1900	JUMP	1950
BREAK	1960	PUSH	1970
SAVE	3000	LOAD	3200

Armed with this, the solving of Underground Adventure will obviously be a lot easier, but it is essential if we're to make sense of the listing!

Complete List of Objects

Equally essential is a list of all the objects in the game, although just to make it a little more difficult we won't tell you where they all start off. However, by the time you've finished this book you'll be able to work it all out for yourself, if you want to cheat!

OBJECT	OBJECT
A VAST CHASM	AN IRON STAFF
A VAST TREE	A STOUT AXE
A THICK COIL OF ROPE	AN ENCHANTED BRIDGE
SOME DYNAMITE!	A PILE OF RUBBLE
A GOLDEN BEAR	A BUN
A BIG BLACK PANTHER	A LONG WOODEN PLANK
A TALL LADDER	SOME NAILS
A HAZY SHIMMERING CURTAIN	A POLISHED MIRROR
A BLOCKED TRACK	A POOL OF OIL
AN EMPTY BOTTLE	A SOLID WALL OF HAZY MIST
THE GHOSTLY DENIZEN OF THE CAVES!	A HUGE BULBOUS SPIDER
AN ENORMOUS FLY!	A RICKETY OLD DOOR
A LUMP OF SOLID MORTAR	A FLY SPRAY!
A SOLID GATE	A NARROW CRACK
A SHINING STONE	A TRUSTY SWORD
SOME WHISKY	A LIVING GARGOYLE!
AN EVIL KNIFE	A KEY
A WALL	SOME MATCHES
AN OLD TORCH	A BLAZING TORCH
A GLOWING LIGHT	AN OLD PARCHMENT
PROGRAM	A PILE OF BROKEN GLASS
A BOTTLE OF OIL	A BOTTLE OF WHISKY
SOME NICELY SAWN TIMBER	

Note that not all of them are objects, and some of them are actually places. We'll see why later.

Finally, a little bit of dialogue with the program, which is the result of first starting the game.

A Dialogue with Underground Adventure

The following is one way that a game night start off, with the computer talking in upper case, and your entries in lower case:

YOU ARE ON AN OLD TRACK HEADING TOWARDS THE CAVES.

YOU CAN SEE :

AN OLD TORCH

YOU CAN GO : SOUTH

WHAT NOW? * (the prompt symbol)

get torch

OK.

WHAT NOW? *

s (or south, or go south)

YOU ARE GETTING EVER NEARER THE CAVES.

YOU CAN GO : NORTH SOUTH

WHAT NOW? *

s

YOU ARE AT THE ENTRANCE OF THE CAVES,WITH PATHS LEADING EVERYWHERE.

YOU CAN SEE :

A SOLID GATE
SOME MATCHES

YOU CAN GO : NORTH SOUTH EAST WEST

WHAT NOW? *

get matches

OK.

WHAT NOW? *

get gate

I CAN'T DO THAT!

WHAT NOW? *

light torch

OK.

inventory

YOU ARE CARRYING :

SOME MATCHES
A BLAZING TORCH

WHAT NOW? *

e

OH DEAR, THE GATE TO THE CAVES APPEARS TO HAVE SLAMMED SHUT!

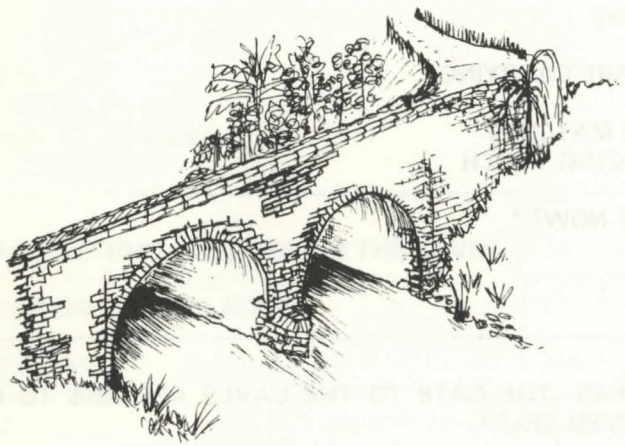
THAT'S TORN IT! YOU'LL HAVE TO FIND THE KEY NOW BEFORE YOU CAN GET OUT.

BUT DON'T WORRY. IT'S IN HERE SOMEWHERE!

WHAT NOW? *

And so it goes on, with your adventure now well and truly under way.

In chapter 3 we'll take a look at some of the knowledge of Basic required to produce this sort of program.



3
Programming Adventures in Basic

Why Bother?

This ranks alongside asking Chris Bonnington why he climbs mountains, or Patrick Moore why he looks at the stars. It's something that they enjoy doing, for some reason that probably they couldn't even explain if asked to sit down and actually state a concrete set of reasons.

So it is with programmers. People enjoy programming, just as much as some enjoy climbing mountains or some enjoy peering through telescopes. As with any other pursuit, there are a variety of ways of programming, and there are a variety of things to write programs about.

This book will teach you all about programming for one subject, that of adventure games. It will also only teach you one style of programming: of necessity, that will be the style adopted by the author.

However, it is to be hoped that originality will shine through on your part, and you'll go on to produce programs that are wildly different from the ones shown here.

Satisfaction

The major reason why people write any sort of program must be for their own satisfaction, rather than anything else, although one is sometimes tempted to think that the mercenary attitude shines through on occasions!

To complete a program that is over 30K long, as some of these adventures undoubtedly will be, is quite an achievement, and even if no one comes along and says 'That was great!', at least you'll still be satisfied with it yourself.

All the better then when someone else does play the game, and congratulates you for writing it. It makes all the hours spent poring over the keyboard desperately trying to solve a programming problem worthwhile.

In return, it's nice to think of the person ultimately playing the adventure taking far longer to solve the program that it took you to write it!

Money

We mentioned mercenary attitudes earlier, and that is obviously one reason why you should bother writing anything, let alone adventure games.

From one point of view there's always the possibility of seeing them go on sale and being marketed by a reputable company, and that is very satisfying.

Another point of view would be that it saves having to buy a lot of adventure games written by other people, but that must be a secondary reason. If you've written the program it won't take you too long to solve it, no matter how many random elements you've put in there.

Level of Skill

One does not have to be the greatest programmer in the world in order to write satisfactory adventure games. You don't need a knowledge of machine code, and the amount of Basic coding that you have to be thoroughly proficient in is not too great: we'll be covering most of what you'll require in the rest of this chapter.

Essentially we're concerned with string handling, and the number of commands in Basic that allow you to manipulate strings is a fairly limited sub-set of the language as a whole.

Once one is au fait with those, the rest of Basic required is mainly standard stuff, with one or two 'tricks of the trade' which we'll be showing you later.

How to Start

The worst thing in the world is to sit down in front of an empty computer, and think 'My God! I've got to write 30K of code!'.

It's akin to the old writer's syndrome of staring at a blank sheet of paper and not having a clue what to write or where to start. Obviously you map things out first, and we'll be looking at that in more detail in chapter 4, as we begin to pull all the separate pieces of knowledge we've learnt together into a coherent whole.

Writing an adventure game is not as daunting a task as you might at first think. Certainly, to look at a listing for an adventure program (perhaps you might care to glance at the listing for Tunnel Adventure, as that is presented in full at the back of the book) is to invite a feeling of nausea as you are confronted by a million and one IF ... THEN, GOTOs and GOSUBs sending program execution careering about all over the place.

However, the listing, when examined carefully and closely, as we shall be doing, does eventually begin to make sense, and you realise that every part of the program is playing its proper role in keeping the whole thing running, whether it's an INPUT routine that stops you entering the wrong type of information, or dropping out of the program; whether it's a routine to handle movement of the character from one room to another; or whatever it is doing, it's all there for a purpose, and later on we'll find out exactly what all those purposes are!

Cooking

What? No, you haven't stumbled into the wrong book, but a useful analogy with programming adventures is to think of the problems posed to a chef, when he/she is presented with a set of ingredients, and told to come up with the finished meal.

We'll present you with a set of program subroutines that handle various tasks, and in this first program we'll also give you the recipe and make them into the finished program.

We may not turn you into the Robert Carrier of the adventure programming world, but at least we'll get you doing more than just making beans on toast!

A Brief Outline

Just to let you know what's coming up, the next section in this chapter will be devoted to learning the commands that are essential to the producing of good adventure games, with obvious emphasis on the string handling ones.

As well as covering the range of commands mentioned earlier (IF . . . THEN, GOSUB and GOTO), we'll also take a brief look at all the other necessary statements taken in conjunction with their use in adventure games.

This is not meant to replace the Basic programming guide in your handbook, but at least will enable you to get going.

This will be followed by a short set of listings, all taken from Underground Adventure, along with a thorough explanation of how they all work, so that you can use them either as they stand, or suitably amended, in your own games.

Our final guide to writing adventures will concentrate on more example listings, together with a set of helpful sections on a good procedure to adopt when sitting down and writing them yourself.

We've even given you a number of scenarios for possible games, which you may like to adapt into your own first adventures!

Underground Adventure is gone through in great detail, with a couple of pages for each verb in our vocabulary, and an explanation of how the code for that verb works, and by following that you should be able to make out what Castle and Tunnel Adventures are doing. You should also learn how to incorporate new words (as you will obviously need to) in your own programs.

Finally, a round up of information on adventures generally, together with a useful set of addresses to contact for further help and information.

Adventure Programming in Basic

In this section we'll look at the commands available to us in Basic for string handling and data handling, and then start tying them up into useful routines.

Input

This is simply a way of typing in information, from a program, that the program will understand and then use in the rest of the program.

However, before we can start using Input, we need to talk about the concept of variables.

Variables

A variable is simply a term used to describe a number, or some text, that can be stored in the computer.

There are two different types of variables that we are interested in, and these are termed numeric and string.

Numeric variables are just numbers, and any of the following is a legal syntax for a variable NAME:

```
A. A5. AZ. BANANA. JAWS. etc.
```

That is, the name must be at least one letter long, and must start with a letter, and anything after that can either be a letter or a number. You can, if you wish, also include spaces in your string name, which will certainly make things easier to read, but the computer will simply discard this when looking at the name.

String variables can be numbers, letters, or a mixture of both, and the restrictions on string variable names are much more severe than those for numeric variables.

String variable names must end with a dollar '$' sign, and they can only be two characters long, including the dollar sign.

Thus we can have names like A$, G$, and so on, but we can only have up to 26 of them! This is a severe limitation when writing adventure games, but we'll show you in this and succeeding chapters how these problems can be overcome.

Back to Input

Input allows you to type some information into the computer from a program, and that information is stored as a variable. For example:

```
10 CLS : PRINT "HELLO, WHAT'S YOUR NAME "
20 INPUT A$
30 PRINT "HELLO ";A$
```

would allow you to enter your name, and then say hello to you.

If you tried typing in 1.45 as your name, you'd have been referred to as 1.45! That's because we specified that we wanted a string to be input (A$). The following example is one way of inputting numerical data, and using that data in part of a calculation.

```
10 CLS : PRINT "HELLO, HOW OLD ARE YOU "
20 INPUT A
30 PRINT "THAT MEANS YOU'RE ";A*365;" DAYS OLD!"
```

Input can also contain some text as well, as in the next example:

```
10 CLS : INPUT "HELLO, WHAT'S YOUR NAME",A$
20 PRINT "HI THERE, ";A$
```

Not only does it make the program shorter, it also makes it neater.

However, there are ways of getting out of an input statement like this one, so we'll be taking a look later on at some more elaborate ways of presenting input statements that stop the player of your adventure from crashing out of the program.

Inputting data into the Spectrum is usually handled on the bottom part of the screen, unless we're using INKEY$ (for example). Getting out of this kind of input is a lot more difficult on the Spectrum than it is on a lot of other home computers!

Data and the Inputting of it

We've already seen that we can get information, or data, into a program by using the input statement, and of course a lot of information could be typed in just by using a lot of input statements.

However, this could get exceedingly tedious if you were using the same information over and over again, hence the need for data statements.

Here the data is typed in as part of a program, read off from within the program, and then acted upon.

Not only does it save you typing in vast amounts of data each time you run the program, but it also allows you to change just one data item, and see how that affects the rest of the program.

In this short example we'll read ten numbers, add them up and then take an average of the whole lot.

```
10 CLS
20 READ A
25 IF A = 0 THEN GO TO 40
30 LET B=B+A
40 LET C=B/10
45 PRINT "THE TOTAL IS ";B
50 PRINT "AND THE AVE. NUMBER READ WAS ";C
60 STOP
70 DATA 1,2,3,4,5,6,7,3,35,80,43,0
```

The IF ... THEN branching statement in line 25 will be explained more fully later, but here it allows us to stop adding up numbers when we've read ten of them, and reached a number of 0: the last data statement.

Data statements can be anywhere in a program, and if you're reading numeric variables, that's what the data statements must contain. If you're reading strings, again they must contain strings, and they must also be enclosed within quotes. For example, "WEST", rather than just WEST. Otherwise you'll get a NONSENSE IN BASIC error message flung at you, and quite right too!

What you must remember is that data is read as it is encountered, so wherever it does happen to be in the program, make sure that it

corresponds to what you want to be read.

Also, make sure that you don't try to read more data than you've actually typed in, otherwise an OUT OF DATA error will occur.

Since we make extensive use of data statements in these adventure listings, always ensure that the right data is being read by the right variable, and that the right amount of data is being read.

If you try to read the same data again, another OUT OF DATA error will take place, unless you use the ...

RESTORE command

This allows you to re-read data, and takes the following syntax:

```
55 RESTORE
56 GO TO 20
```

One concept to explain here. GOTO, which transfers program execution from one part of a program to another, will again be gone into in more detail later.

To finish with data for a while, here's a short example that mixes string and numeric data:

```
10 CLS
20 READ A$,B,C,D
30 PRINT A$;" IS ";B;" YEARS, ";C;"  MONTHS  AND
";D;"  DAYS  OLD!"
40 GO TO 20
50 DATA "PETE",26,1,10
60 DATA "BERYL",25,5,8
```

When run, this will generate an OUT OF DATA error, as we send it back to line 20 to read more data that isn't there, but the concept is, none the less, a sound one.

INKEY$, IF and THEN

We'll confine ourselves to using the keyboard, where we find that

INKEY$ allows us to input one character at a time, without the need to keep pressing the ENTER key.

The following program will illustrate this point:

```
10 CLS : PRINT "PRESS ANY KEY"
20 IF INKEY$ = "" THEN GO TO 20
30 PRINT "YOU PRESSED ";INKEY$;"!"
40 GO TO 20
```

A number of new ideas here.

In line 20, the line is executed as follows:

Step 1) See if a key has been pressed on the keyboard.
Step 2) If it hasn't (i.e. if a null string, "", has been detected) then go back and try again.
Step 3) It has, so we fall through to line 30, where it prints out which key was actually pressed.

Line 40 just sends us back to line 20 again, and waits for another key to be pressed.

The only way to stop this program is by pressing the break key, otherwise it will loop around for ever!

We can be selective in which key we press by moving on only if the correct one is depressed. For instance, suppose we want to halt a program until the space bar is pressed. This part of our program might look something like:

```
100 IF INKEY$ <> " " THEN GO TO 100
110 carry on
```

Here, if A$ is not equal to (the symbol on the W key, achieved with the symbol shift key) a space i.e. the space bar has not been pressed, then go back to line 100 and wait until it has.

This can be extended further, for example if we want someone to make a Yes or No decision, and only want them to press the Y or N keys. There are a number of ways of doing this (although we don't recommend typing in those parts of the program that are in lower case!):

```
100 IF INKEY$ = "" THEN GO TO 100
110 IF INKEY$="Y"THEN go to another bit of the pro
gram
120 IF INKEY$="N"THEN go somewhere else
130 GO TO 100
```

So, if they press Y we go to one part of the program, N and we go to another, but if neither are pressed then we wait until one of them is.

Or how about this:

```
100 IF INKEY$ <> "Y" AND INKEY$ <> "N" THEN GO TO
100
110 IFINKEY$="Y"THEN go to one part of the program
120 this is what happens if INKEY$ is equal to N
```

Here, we sit and wait till either Y or N is pressed. It takes up less program space, and is just another way of doing the same thing.

This kind of selective key pressing is one of the principal uses of the IF ... THEN statement.

Its other main role is in decision making according to the value of string or numerical variables.

Strings or numbers can be compared using the greater than '>' and less than '<' operators, which have the following connotations:

A > B : A greater than B
A >= B : A greater than or equal to B
A = B : A equal to B
A <= B : A less than or equal to B
A < B : A less than B
A <> B : A not equal to B

Thus our program might contain a line something like:

```
100 IF A <= B THEN GO TO 200
```

Thus, if A is less than or equal to B then we go to line 200. If A is greater than B we simply slip through to the next line of the program.

Strings are compared alphabetically. Thus "AAAA" is reckoned to be

less than "ABAA", and so on, and these can also be used in IF ... THEN statements as above.

Subroutines

Some sections of a program have to be performed time and time again, and it would become very tedious, as well as wasting a lot of memory, if you had to keep typing out the following lines every time you wanted the program to execute them:

```
10 LET A=B+C
20 LET D=E+F
30 LET H=A+D
40 PRINT H
50 REM GET ON WITH PROGRAM AGAIN.
```

Of course, if our program segments were only this long there wouldn't be too much trouble, but as we learn more and more commands the complexity of our programs will grow, and the need to perform repetitive calculations will grow with it.

Thus we have subroutines, lines which are used a lot within a main program, and which generally just perform one specific function.

We'll see how to 'call up' subroutines in the next couple of pages, but the point to be made here is that they too, like the rest of the program, should be REMmed.

For instance:

```
5000 REM ************************************
5010 REM * START OF BORDER DRAWING SUBROUTINE *
5020 REM ************************************
5030 CLS : LET A=0 : PRINT  "  ++++++++++++++++++
++++++++++";
5040 LET A=A+1:IFA=24 THEN GO TO 5060
5050 PRINT"+                                  +";
5055 GO TO 5040
5060 PRINT " ++++++++++++++++++++++++++++++++";
5070 REM ************************************
5080 REM * END OF BORDER DRAWING SUBROUTINE *
5090 REM ************************************
```

I don't pretend for a minute that this is the most wonderful program in the world, but at least it shows how the REMark statement can be used to clarify a program listing.

By structuring programs in this way, the REM statement becomes a powerful ally in keeping your programs neat, tidy and intelligible.

More String Commands: LEN

LEN, as you might reasonably guess, is associated with the LENgth of a string.

For instance, if we assign a string A$ to be equal to "A LONG STRING", the command:

```
PRINT LEN A$
```

would return a value of 13, this being the number of characters (including spaces), contained within the string A$.

We can also assign another variable to be equal to the length of a string, thus :

```
10 A$="ANOTHER STRING"
20 LET B=LEN A$
30 PRINTB
```

Running this would give us the result 14, this being the value of the variable B, or in other words the number of characters in the string A$.

LEN comes into its own when taken in conjunction with the other string command available on the Spectrum, namely slicing.

Slicing

Unlike most other computers, which will allow you to use a form of commands called MID$, LEFT$, and RIGHT$, the Spectrum puts you into a completely new ball game by having just one string handling command, and this is usually referred to as 'slicing'.

If handled properly this is an extremely powerful command, but for the newcomer to programming it can, at first, be a little confusing.

The command takes the syntax:

```
A$ (i TO j)
```

If we assign the variable A$ to be equal to "A BIT OF STRING", then the command works in the following way:

```
PRINT A$ (1 TO 4)
```

would give you the result A BIT.

```
PRINT A$ (7 TO 8)
```

would give you the result OF.

The command works as follows. It takes the ith letter of a string (in our last example the seventh letter), and then keeps taking letters out of the string until it reaches the jth character, which was the eighth letter in our last example.

You don't have to put simple numbers in, as numerical variables can be used in their place. For example:

```
PRINT A$ (L to M)
```

would take out the number of letters of the string A$ from the Lth letter to the Mth.

Other commands can be used within the brackets. For instance,

```
PRINT A$ (10 TO LEN A$)
```

would take all the letters from the tenth letter to the end, which in this case would give us the result STRING.

Other strings can also be assigned using this command. If we stay with A$ being equal to A BIT OF STRING, then we might code in something like:

```
LET B$ = A$ (3 TO 5)
```

which would place the string BIT within B$.

As we make extensive use of this command in all of our adventures, it is well worth while playing about with it and understanding it

thoroughly. After all, it's all you've got!

To further illustrate, how about this program to reverse the direction of a word:

```
10 LET A$="SURFING"
20 LET B$=A$ (7 TO 7)
30 LET C$=A$ (6 TO 6)
40 LET D$=A$ (5 TO 5)
50 LET E$=A$ (4 TO 4)
60 LET F$=A$ (3 TO 3)
70 LET G$=A$ (2 TO 2)
80 LET H$=A$ (1 TO 1)
90 LET I$=B$+C$+D$+E$+F$+G$+H$
100 PRINTI$
```

When run, the word GNIFRUS is printed out.

There are much more elegant ways of doing this kind of thing, as we'll see when we encounter FOR ... NEXT loops shortly.

STR$ and VAL

Two functions which are essentially the inverse of each other, and both of which are concerned with string and numeric manipulation.

Take a number A, equal to (say) 12

The command :

```
PRINT STR$ A
```

will print out the string 12, although the number A has remained the same.

This command is more useful when assigning variables, so the following program shows this in action:

```
10 LET A=24.000000
20 LET A$=STR$ A
30 PRINT A$
40 PRINT LEN A$
50 PRINT A$ (1 TO 2)
```


When run, this program will print out the following:

```
24
2
24
```

So you can see, by finding the position of the decimal point, we can split a number up into its two components.

How do we do this?

Well, one way would be to use the inverse function, VAL.

VAL takes a string, and converts it into a number. Thus, if the string A$ was equal to "10", the command:

```
PRINT VAL A$
```

would print out the number 10.

If A$ = "12.123", VAL A$ would also equal 12.123, but of course this time it would be in numerical format.

VAL comes to a halt when it comes across something that is not a number.

Thus, if A$ = "88A88B", VAL A$ would return just 88.

We can also print out straightforward variables. That is, in the following program, we are defining the variable A to be equal to the VALue of various strings:

```
10 LET A=VAL "23.23"
20 PRINTA
30 LET A=VAL "A"
40 PRINTA
50 LET A=VAL "-100.9"
60 PRINTA
```

When run, the results on the screen would be:

```
23.23
0
-100.9
```

So, to split a number up into its component parts, we must find the decimal point by turning the number into a string, taking each number at a time until we find the decimal point, and so on.

CHR$ and CODE

Another two analogous functions, again concerned with string handling, but CODE in particular assumes great importance when talking about communicating from one microcomputer to another.

CODE is concerned with something called ASCII, which itself stands for the American Standard Code for the Interchange of Information, although like all computers the Spectrum does have a few variations on a theme!

Still, to generate these characters on the screen the following syntax is used :

```
PRINT CODE "A"
```

which would return a value of 65, or:

```
PRINT CODE A$
```

which would return the ASCII value of the first character contained in the string A$.

CHR$ is the opposite of this, in that it applies only to numbers, and prints out on the screen the single character string whose code is that number. Some of the CHR$ are special, however. CHR$ 8, for instance, is the code to print a backspace, CHR$ 13 is the code for a carriage return, and so on.

Everything else, like printing in black text, printing letters, etc., can all be done with the CHR$ command.

Both of these commands can again be used to define other variables. For example:

```
LET A = CODE "A"
```

will put the value of 65 into the variable A, and

```
LET A$=CHR$ 13
```


will put the character string 13 (in fact, a carriage return) into the string A$.

FOR ... NEXT

Where would we be without FOR ... NEXT loops?

Although we've been instructing the computer to do the same thing a number of times over, by use of a simple incrementing variable, the 'loop' approach is far better, and far easier to operate.

For instance:

```
10 CLS
20 FOR I = 1 TO 100
30 PRINT I
40 NEXT I
```

This will just print out the numbers from 1 to 100 in rapid succession, but illustrates the point.

Line 20 is the start of our loop, and tells the computer that we want to do something 100 times. In fact, we want to print out the numbers from 1 to 100, and as the value of I increases, it is printed out in line 30. Line 40 then tells the computer NEXT, i.e. there's more to come, and the program branches back to line 20.

It keeps on doing this until I has reached the value of 100, at which point it stops and our short program ceases execution.

Actually, I reaches the value of 101. Why? Well, when it has the value of 100, it prints it out as in line 30, sees the NEXT statement in line 40, and increases the value of I to 101. However, on branching back the computer finds that the limit of the loop is when I is equal to 100, so it stops!

We can have more than one loop active at a time, like this:

```
10 CLS
20 FOR I = 1 TO 20
30 FOR J = 1 TO 3
40 PRINT J,I
50 NEXT J
60 NEXT I
```

The first time around, I is set to 1, and J counts through from 1 to 3. Thus the display goes something like:

```
1          1
2          1
3          1
```

Then J has finished, so we go onto line 60, where I is incremented again, so it's back through the loop once more, for:

```
1          2
2          2
3          2
```

and so on, until we finally reach:

```
1          20
2          20
3          20
```

at which point everything stops.

Lines 50 and 60 could have been abbreviated to the rather more straightforward:

```
50 NEXT J,I
```

Just make sure you keep everything in the right order, and don't have more than 26 loops in action at the same time, otherwise the computer will blow its stack (computing joke).

Loops can be made to count in steps as well, for instance:

```
20 FORI=1TO100STEP2
30 PRINTI
40 NEXTI
```

when run, will print out the numbers 2,4,6, ... 100. We can also go backwards:

```
20 FORI=100TO1STEP-2
30 PRINTI
40 NEXTI
```


when run, will print out the numbers 100,98,96 .... 2.

For an interesting application, using only commands we've seen so far, can you work out what this program is doing (type it in and see, if you can't!)?

```
10 LET A$="ABCDEFG"
20 IF INKEY$ = "" THEN GO TO 20
30 FOR I = 1 TO LEN A$
40 IF INKEY$ = A$(I TO I) THEN PRINT INKEY$:GO
   TO 20
50 NEXT I : GO TO 20
```

GOTO somewhere

We've already encountered this one. Basically it sends command of a program to somewhere else within the program, or back to the same line as in the example on the previous page in line 20.

The syntax used is GOTO xxx, where xxx is an existing line number.

If it isn't, then program execution continues at the next line that it can find. As this usually results in throwing the program into utter chaos, you are well advised to stick to existing line numbers!

As an example

```
10 CLS
20 PRINT "HELLO!"
30 GO TO 20
```

When run, this just prints up hundreds of HELLO!s, until you hit the break key.

Changing line 30 to read GOTO 10, produces a slightly flickering display.

GOSUB and RETURNing

Subroutines have been met before, as small, or maybe even large, segments of programs that have to be repeated many times.

Performing the same function over and over again is a repetitive task, and having to type the code in each time you wanted it actioned would take a lot of time, and a lot of memory.

Thus subroutines were born, and the command used to send program control to them is GOSUB xxx, where xxx is the line number at the start of the subroutine.

Once actioned, the command to send control back to the main program again is RETURN.

Great care must be taken in matching up GOSUBs with RETURNs, otherwise a RETURN WITHOUT GOSUB error will take place sooner rather than later.

As with FOR ... NEXT loops you can have up to 26 subroutines in action at the same time, but no more.

Thus you can jump about from one subroutine to another, and quite often it is necessary to do this, but it isn't really very good programming practice.

A few examples:

```
10 CLS
20 LET A=5:LET B=10
30 GO SUB 100
40 GO SUB 200
50 GO SUB 100
60 PRINT A,B
70 STOP: REM IMPORTANT, OTHERWISE PROGRAM FALLS T
HROUGH!
100 LET A=A*A
110 LET A=A+5
120 RETURN
200 LET B=B-1
210 RETURN
```

When run, the first subroutine is encountered twice, the second once only, and the resultant printout is:

```
905      9
```

Of course, one can get a lot more complicated than this!


```
10 CLS
20 LETA=1:LETB=2:LETC=3
30 GO SUB 100
40 GO SUB 200
50 GO SUB 300
60 PRINT A,B,C
70 STOP
100 LET A=A+B+C
110 GO SUB 200
120 RETURN
200 GO SUB 300
210 LET A=A+B+C
220 GO SUB 300
230 RETURN
300 LET A=A+1
310 RETURN
```

What value will A have when this program is run? Try it and see!

What's GOing on

Quite often within a program, the subroutine or line number you want to go to will depend on the value of a particular variable.

This could be achieved in the following way:

```
10 IF A = 1 THEN GO TO 100
20 IF A = 2 THEN GO TO 200
30 IF A = 3 THEN GO TO 300
40 IF A = 4 THEN GO TO 400
50 IF A = 5 THEN GO TO 500
60 etc.
```

Although this works, it could hardly be described as an elegant way of programming.

In its place we can use a form of computed GOTO command.

As an example:

```
10 GO TO A*100+100
```

Here, if A has the value 0, the program continues execution at line 100 onwards, a value of 1 and it goes to line 200, and so on up to whatever the final value of A is.

Just one example of this command in use could be something like this, which is an interesting use of string handling:

```
10 LET K$="ABCDE"
20 PRINT "ACTIVITY 'A' : PRESS A"
30 PRINT "ACTIVITY 'B' : PRESS B"
40 PRINT "ACTIVITY 'C' : PRESS C"
50 PRINT "ACTIVITY 'D' : PRESS D"
60 PRINT "ACTIVITY 'E' : PRESS E"
70 IF INKEY$ = "" THEN GO TO 70
75 LET A$ = INKEY$
80 FOR I = 1 TO LEN K$
90 IF A$ = K$ (I TO I) THEN GO TO 1000
100 NEXT I
110 GO TO 70
1000 GO TO (CODE A$-65)*100+100
1100 rest of program
```

RaNDom INTegers

Like most of the home computers currently available, the Spectrum is not without a random number generator.

Alas, like most of them it isn't particularly random, and so a few operations have to be done before we can begin setting up 'genuinely' random numbers.

The syntax to be observed is RND, which will give a number in the range 0 to 1.

The INT command comes in useful here, as elsewhere. It chops off the numbers after the decimal point, basically, so INT 2.24 becomes 2, as does INT 2.89 .

INT of a negative number returns the next lower number. Thus, INT -2.24 becomes -3!

So, to generate an integer random number, we could use:

```
10 PRINT INT(RND)
```


However, this will not be very satisfactory for generating future numbers, since RND always returns a number between 0 and 1. So we need to scale things up a little:

```
10 PRINT INT(RND*10+1)
```

which will produce a number in the range 1 to 10.

To generate numbers between a given range, where X is the top limit and Y the lower limit, we must use the formula:

```
10 PRINT INT(RND*(X-Y+1)+Y)
```

This is used in our adventures for producing random events e.g. the appearance of a gargoyle, or the success or failure of throwing a knife.

A New DIMension

We've already seen how numbers and strings can be stored as variables like A, A$, and so on. However, this gets a mite restrictive after a while, and we need to resort to other things. After all, there are only so many letters in the alphabet!

Let's say that we're generating ten random numbers, and we want to store them all as variables.

We could have a very lengthy program to do this:

```
10 LET A=INT(RND*10+1)
20 LET B= ..... etc.
```

but this is extremely space consuming, and there are better ways.

This is where arrays, otherwise called subscripted variables, come in.

The syntax for referring to these is A(1), A(2), etc., up to whatever limit you assign, and these subscripted variables could be assigned numbers something like this (assuming we've dimensioned an array to have ten elements using the DIM A(10) command):

```
10 FORI=1TO10
20 LET A(I)=INT(RND*10+1)
30 NEXT I
```

Where now we have the ten different numbers stored in A(1), A(2) etc. up to A(10).

These numbers can then be selected at will. For example, PRINT (A(5)) will print the fifth number, or element, in our array A.

To prove it, we could print them all out by adding to our program:

```
40 FORI=1TO10
50 PRINTA(I)
60 NEXT I
```

The numbers in an array can be assigned to other variables (e.g. A = A(3)), or even calculated dynamically by using another variable (e.g.PRINT A(B*2)).

However, more often than not we'll be wanting to use a lot more than ten elements in an array, and this is where the DIM statement begins to get a mite more powerful.

The syntax for this is DIM A(199), or whatever, which sets aside a certain amount of room in the computer's memory for storing all the numbers that you might be wanting to save. Whether you use them all or not, that memory is reserved, so use arrays selectively.

Arrays are not limited to one dimension either. You can dimension something as A(8,8) if you like, for instance in a chess game, where you have a board 8 squares by 8.

The elements in that array are referred to as A(1,5), A(6,3), and so on. It is helpful to think of these values as being stored in rows and columns, where the first number refers to the row and the second to the column. Thus A(5,7) is the seventh column of the fifth row. Thinking of it all as boxes of numbers, or strings, stored in rows and columns will always help when you want to reference a particular one within a program.

We'll be using arrays extensively in all our adventures, so it's useful to learn how they operate!

There are two peculiarities within Spectrum Basic when handling arrays, and they concern string arrays. If you dimension a string to have, say, ten elements, then you cannot also use a simple ordinary string with the same letter. Thus you can't have A$(10) and A$.

Secondly, you must specify the length of all string arrays that are used. For instance, if we wanted to hold 100 strings, each with a maximum length of 40 characters, we'd have to say something like DIM A$(100,40). This allocated memory is used up even if one string is only 2 characters long, by padding out the string with spaces up to the length of 40 characters or whatever, so make sure you get your memory's worth out of any dimensioned strings!

Getting Started

Now that we've learnt most of what we'll need to know about strings, data and dimensioning arrays, it's about time we started looking at the results of using this in an actual program.

Our example, as always, will be the Underground Adventure listing, so now we'll start explaining some of the variables that are used in this game, so that we can get an understanding of how the various parts of the program operate.

Lines 2 to 27 define one set of variables, which relate to the gate being open (GF), and door being shut (DF), C$ is set equal to a carriage return, line 10 goes respectively to the subroutines that print up the introduction, and read in all our data, followed by defining the variable CP, which is the Character Position, and relates to the room number that you happen to be in at the time.

There are a few other variables defined here, and we'll come to those as the program goes on.

The next set of 4 variables just contain messages that we'll use later on in the program.

```
   2 LET df=0: LET gf=1: LET gs=0: LET fl=0: LET o
c=0
   3 LET tb=0: LET vb=0: LET pd=0: LET zz=0: LET s
c=0: LET lc=0: LET gs=0: LET br=0: LET np=0
   4 DIM x(9)
   5 LET c$=CHR$ (13)
  10 GO SUB 9000: GO SUB 2001: LET cp=1
  15 LET d$="It is now pitch dark.   Carry onany f
urther and you'll fall intoa deep, deep pit."
  20 LET i$="You can't go that way."
  25 LET g$="The gate is now shut."
  26 LET l$="Going down .....!"
  27 LET f$="The door is now shut."
```


Moving Around

Line 200 sets us off to the routine that checks for character movement:

```
200 GO SUB 5000
```

but before we look at that, we'll jump down to line 2001 and define a few more variables:

```
2001 LET nv=38: LET nn=53: LET z=100: LET lo=53: D
IM p$(z,48): DIM p(z,4): DIM o$(lo,10): DIM o(lo):
 DIM v$(nv,3): DIM j$(nn,3): DIM t$(4,5)
```

This controls all our data reading which takes place in lines 2002 to 2270. These are reproduced in chapter 6, but the variables are set as follows:

NV = the number of verbs we're going to use.

NN = the number of nouns we're going to use.

Z = the number of rooms contained in the adventure.

LO = the number of nouns again, but is used to control the LOcation of every object in the game, there being as many objects as there are nouns.

DIM P$(Z) = dimension the variable P$ to be equal to the number of rooms in the game. P$(I) then contains the description for the Ith room.

DIM P(Z,4) = dimension the variable P to be equal to the number of rooms, with four sub-elements to each level of P. These are used to determine the direction one can take from within a room, and indicate NORTH, SOUTH, EAST and WEST respectively. Thus P(I,3) refers to the direction EAST from room I.

DIM O$(LO) = dimension the variable O$ to be equal to the number of nouns. O$(I) then contains the description of the Ith object.

DIM O(LO) = dimension the variable O to be equal to the number

of nouns. O then contains the position of each object in the game, by referring to its room number. Thus O(I) refers to the Ith object, and if set equal to J puts the Ith object in the Jth room.

DIM V$(NV) = dimension the variable V$ to be equal to the number of verbs. V$ then contains the actual verb itself. Thus V$(I) is the Ith verb. For instance, V$(1) is the verb GO.

DIM J$(NN) = dimension the variable J$ to be equal to the number of nouns. J$(I) then contains the shorthand description for the Ith noun. Thus if O$(I) contained the string "A RICKETY OLD DOOR", J$(I) would contain just "DOO", for door.

Now, let's look at the actual room moving routine, contained in lines 5000 to 5206.

Room Movement Routine

This routine is used to handle all room movement in the game, so we'll take a close look at it.

```
5000 CLS
5001 IF o(46)<>-1 AND (cp>4 AND cp<100) THEN PRINT
 d$: LET pd=1: RETURN
5002 PRINT "You're ";p$(cp): LET pd=0
5003 IF cp=42 AND tb=1 AND p(42,2)=0 THEN GO TO 60
54
5004 LET z$="You can see:                    "
5005 PRINT : PRINT
5006 FOR i=1 TO lo: IF o(i)=cp THEN PRINT z$;o$(i)
: LET z$=""
5008 NEXT i
5009 IF cp=3 AND gf=0 THEN PRINT g$
5010 LET fl=0
5011 PRINT
5012 PRINT : PRINT "You can go:": FOR i=1 TO 4: IF
 p(cp,i)<>0 THEN PRINT t$(i);" ";: LET fl=1
5013 NEXT i
5014 IF fl=0 THEN PRINT "Nowhere."
5015 IF np=1 THEN GO TO 6000
5016 IF (cp>20 AND cp<88) AND (INT (RND*100>96)) T
HEN LET np=1: GO TO 6000
5018 IF cp<>69 THEN RETURN
5020 IF p(69,4)=70 THEN RETURN
5022 IF o(19)<>-1 THEN PRINT "You can't pass yet."
: RETURN
5024 PRINT : PRINT "The shimmering curtain washes
  away the mist and reveals a new tunnel."
5025 LET p(69,4)=70: LET o(15)=0: LET zz=zz-1: LET
 p$(69)="walking past an ice cold spot."
5026 LET o(20)=0: RETURN
```

Explanation of Routine

We'll take this line by line, so:

Line 5000 simply clears the screen.

Line 5001 checks to see if you're holding a blazing torch (O(46)). If the variable is set to -1 it means that you're carrying it. If it's not equal to -1, the line carries on to see if you're in a room lying between room numbers 5 and 99. If you are, it then prints up the variable D$ as defined in line 15 and returns to the WHAT NOW prompt, having set the darkness variable PD equal to 1. Any attempt to move now without lighting the torch will make you fall into a pit and plummet to your doom.

Line 5002 prints up "You're" followed by the description of the room. You are always in room CP. The darkness variable PD is set to 0, since if we've moved, we can't be in darkness.

Line 5003 checks the 'bear following' variable TB. If the bear is following you, and you're in room 42 (which holds a fierce panther to begin with), and there is no path south from room 42, the program transfers execution to line 6054, which prints up what happens when the bear meets the panther!

Line 5004 is the start of the 'you can see' routine, which goes on to:

Line 5006, which checks to see if the location of any object, O, is equal to the current room number CP. If it is, then it tells you that you can see it, but if nothing's there it just prints up nothing.

Line 5009 checks to see if you're in room 3, and if the flag for the state of the gate (open or closed) is set or not (1 or 0), and if it is set prints up the variable G$, as defined in line 25.

Line 5010 just clears the variable FL, used in the next routine:

Lines 5012 to 5014 go through the four possible directions from each room, and check to see whether you can go in any of them, by seeing if the relevant part of the variable P is set to 0, in which case you can't, or something else, in which case you can. It then prints up the right part of the variable T$, which is set to equal the words NORTH, SOUTH, EAST and WEST earlier on in the program.

Line 5015 checks to see if there's a gargoyle chucking a knife at you, and if there is transfers program execution to line 6000, which we'll come to later.

Line 5016 goes through a random number generation, and if that number is greater than 96 (on a scale of 0 to 99), and if you're in a room number greater than 20 but less than 88 sets the gargoyle present flag NP and goes off to line 6000.

Line 5018 checks to see if you're in room 69. If you're not, program execution returns from this subroutine.

Lines 5020 onwards are assuming you are in room 69, which is initially guarded by a hazy mist, through which you cannot pass until various conditions are met. These are checked in lines 5020 to 5026, and I'll let you work out for yourself what they are!

Basically you have to be carrying a certain object before you can get past, and if you are then obviously the hazard doesn't exist any more, and we have to change the relevant parts of the variable P(69) to allow us to move safely through here in future, and the room description P$(69), all of which is done in line 5025.

Line 5026 simply makes the object you must be carrying disappear.

So you can see the checks that have to be made before we can allow our explorer to move through certain areas.

It would be an easy matter to alter this routine to suit your own adventure requirements, just by changing the conditions that have to be met and checking for the right room numbers and the right flags being set.

As we've said, you'll find all the data in chapter 6.

Now we've seen how one routine works. Let's sit down and write an adventure!

4

Writing Your Own Adventures

Let's Get Started

We've seen one of the major routines in the game now, that of handling the movement of the character within the adventure, once we've established from other routines whether or not the character can in fact go in that direction.

That is achieved using the verb GO, which we'll come to along with all the other verbs in chapter 6.

All the data that is necessary for this game, together with a thorough explanation of how it all works, what it all means and how it's all stored in the program, will also be found in chapter 6.

Meanwhile, there's an awful lot of additional coding which doesn't come into either of those sections, and the purpose of this chapter is to present you with the rest of it, including standard routines for the inputting of data, checking on the validity of a move, checking whether the words you've typed in make sense, and one or two other routines which are especially for this game (we couldn't just give you all of the listing bar a couple of lines!), but which could nonetheless be adapted for use in your own games.

You'll know the sort of occasions when it is necessary to include these special routines, what they're doing (and equally important, how they're doing what they're doing), and so you will be able to use variations on them in your own games.

So, between this chapter and chapter 6 you'll get the complete listing

for Underground Adventure, and perhaps by presenting it in small chunks like this you'll feel more inclined to type it all in!

If not, you could always buy the cassette containing the three adventures in this book, configured to run on a Spectrum, and study the listing that way.

Summary So Far

You know what a number of the essential variables in this game are now doing, and can readily adapt them for use in your own games.

The variable CP for instance, which is used to keep track of the room number, and is updated as you move from room to room.

The variable NP, to detect whether or not a living gargoyle has emerged from the rocks and is about to engage you in mortal combat.

The variable PD to check for darkness, and the carrying of the blazing torch.

These, and the others, are the backbone of the game, and without them this adventure could not function. Without similar variables in your own games it would be equally impossible to play and/or write them.

Variables like these are there to make life easier for you. Use them in your own games, and the actual writing of a complete adventure will soon become relatively easy.

However, there's a lot to learn yet, like the drawing of maps, the placing of objects, the positioning of any hazards en route, and everything that goes up to make the total game.

In the next section we'll start again from scratch, and assume that you've sat down with a blank sheet of paper, and want to start writing an adventure game.

So let's get going!

The First Steps

Possibly the most difficult step of all is outlining the story that you're

going to have as the backbone of the adventure.

In effect it will have to be a miniature novel, involving (relatively) realistic concepts, although an ingredient of most adventure worlds is that little touch of magic that sets them aside from the real world.

The plot, just as in a good novel, must flow smoothly from one stage to the next, with no totally unexpected, inexplicable events. One adventure I know suddenly has a sword that you've been happily carrying along turn into a snake in your hands, which then bites you and kills you off.

This is totally inexcusable, and shouldn't find a home in any real adventure. The impossible happens quite often in these games, but at least there should be a warning that it's going to happen, and it should not be sufficient to kill off the character.

So if we're going to have magic, let's keep it on a fairly reasonable level, and stick to iron staffs being waved and causing a bridge to appear over the chasm.

Events that kill off the hero, like crossing a rickety bridge with a heavy bear in tow, should generally be as expected as possible, and only be the fault of the adventurer. In real life, would you expect a rickety bridge to support the weight of a heavy, lumbering bear?

In Underground Adventure, dynamite has to be employed in one instance before you can progress. It is reasonable to assume that lighting the dynamite whilst you're still holding it will not do you any good, and so it should be placed on the ground first of all.

On the other hand, some of the elements in this game, and others, are randomised to give the game some semblance of reality. Not that you'd often bump into a living gargoyle carved out of the rock, who then engages you in a duel to the death every time you meet him, but should such an event take place it is reasonable to assume that the outcome of the fight will not always be the same.

Thus you will sometimes get killed (though not very often, otherwise the game would get very tedious), and sometimes your throws will miss the gargoyle, but again you should conquer him (her?!) most of the time and live to carry on the game.

So anything that happens in the game must have a remote base in reality, and the inexplicable shouldn't really happen without at least

being safe to the player.

Getting the Idea

As we've said, this is possibly the most difficult part of all. Many adventures have now been written, and coming up with an original scenario each time is getting gradually harder and harder. Some possible ideas are presented in chapter 5, where we've gone through a number of adventure scenarios, and described them in some considerable detail.

However, there is of course no constraint on you to use them at all, so your own ideas will have to come from somewhere.

One tried and trusted idea is by dipping into a few books such as *Lord of the Rings*, in which there are a multitude of possible plots which could be turned into very reasonable games. However, as in all implementations of this sort one has to be very careful about the laws of copyright, as we've seen with the Hitch Hiker's Guide to the Galaxy game, so you'll probably have to change a lot of names to protect the innocent, i.e. you!

The traditional thud and blunder adventure, steeped in Gothic names and ancient runes, has been done by many authors, although obviously the scope here is vast for doing variations on a theme.

One possible answer might be to read a few science fiction novels (bearing in mind the author's copyright), such as the works of Michael Moorcock, and obtain a few ideas from there.

To the beginner though it must seem that just about every possible idea has been tried before, including exploring ancient tombs and crypts, jungle adventures that pit you against various natives and native problems, cowboy adventures, outer space adventures, underwater adventures, and the like, and that it would be impossible to come up with a new and original plot-line for your story.

But bear in mind that there have been many more novels written than there have been computer adventures, and people still keep managing to come up with original themes for those, so the ideas are always there: it's just a question of thinking them up.

Visitors from outer space, detective adventures, psychological adventures, biblical adventures, are all relatively new areas, and perhaps

combining one of these new ideas with the character choosing role discussed earlier could pave the way for a whole new set of computer games.

The work is up to you though, and your plot, whatever it consists of, must ring true throughout, and keep the player of the game constantly entertained, forever pitting him against new challenges, new tasks, and keeping the interest by finding out just that little bit extra with each game.

The Hazards

Now there's a television program! But no, nothing to do with car driving American lunatics in an otherwise sleepy mid-western village, one of the most important parts of any adventure game will be the constant search for new problems to set the player, new tasks that have to be accomplished before you can proceed further, and making those hazards solvable, but (preferably) as difficult as possible.

The number of problems set will always vary from game to game, and should to some extent depend on the number of rooms in the game. Perhaps on a 1 to 6 ratio, with a new task to be solved every half dozen rooms or so ?

Some games favour a constant source of worry, and indeed Underground Adventure does the same, with the living gargoyle coming up every now and again, along with a random chance that, as well as fighting with you, he might just nip in and steal a few useful items that you happen to be carrying and hide them in the maze.

As a helping hand, here's a list of the hazards presented in Underground Adventure, and the rooms in which they are first found:

A vast chasm that is too wide to jump: room 15
A massive tree that blocks your path: room 21
A deep drop that is to steep to climb down: room 35
A blocked wall that prevents you from going further: room 4
A golden bear that will not let you pass: room 27
A fierce black panther that stands in your way: room 42
Another deep chasm amongst the rocks: room 10
A steep incline that is to steep to climb up: room 45
A shimmering curtain of light that dazzles you: room 93
An old mining track that is blocked up: room 79
A hazy wall of mist that is too thick to pass through: room 69

The denizen of the caves, who will not let you through: room 50
A giant spider, out to eat you: room 84
A giant fly, out to kill you: room 74
An old door that blocks your path: room 60
A narrow crack, which you can't squeeze through: room 53

There are 100 rooms in Underground Adventure, so we fit nicely into our 1 in 6 ratio, with the above 16 problems to solve. We'll tell you some solutions along the way, but not all of them!

Constant Problems

As well as all of the above, there are a number of constant problems that keep recurring, like the gargoyles, and any reasonable adventure has the same kind of mixture. A good solid set of problems which give the player plenty to chew over, along with a reasonable set of constant events that can also give cause for worry.

However, whatever the kind of problem, be it in a set place or occuring at random, one golden rule of programming this type of game remains the same: if the player solves the problem, make sure the program checks for this and adjusts its variables accordingly.

There is nothing worse for a player than, having spent hours achieving one goal, to throw away the relevant object which has enabled him to do this (or perhaps have it taken away by the program once it has fulfilled its duty), and then to see a bug in the program causing the problem to re-appear!

In other words, don't make your adventures impossible, which is always a problem when you're manipulating a lot of objects. Just placing one of them in the wrong room could cause the program to become unsolvable: a cardinal sin.

One of the more common constant problems is that of a torch. If you're deep underground it's fairly safe to assume that you won't be able to see very much, and so a torch becomes vital.

To light the torch you will also need some matches, and these must also be hidden in the game somewhere.

Finding the torch and lighting it is usually no problem, but keeping it lit often is. A sudden gust of wind perhaps (which could easily be done in the earlier movement routine by checking for, say, room 52

or whatever, and whenever the player walks through there the torch gets blown out), or a swim through some water would do the trick. If you go through water, you would also get the matches wet, so how do you light the torch again?

A torch carries with it another problem. There is usually a limit on how much you can carry at a time, and certain objects will always have to be with you, like torches, axes, and so on, and so the problem becomes what do you carry at the same time.

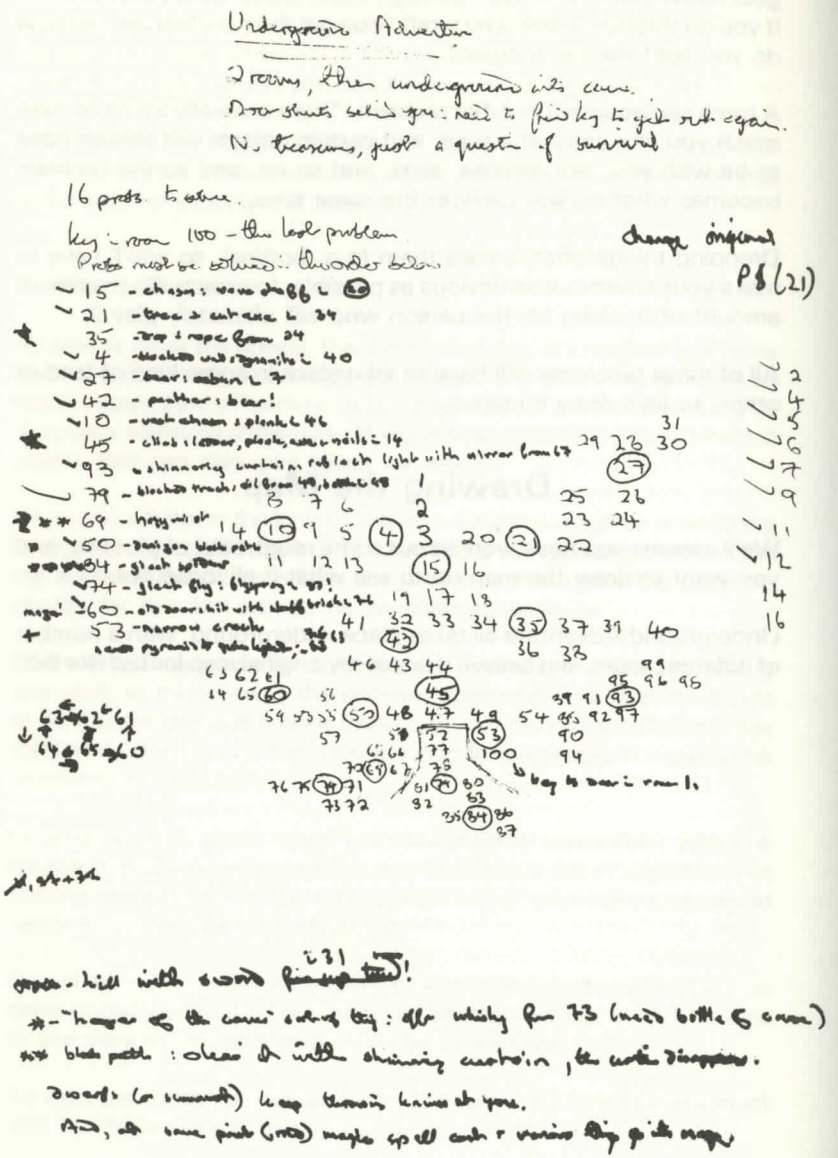
Dropping things often breaks them (e.g. bottles), so you'll have to make your adventure as devious as possible, to ensure the maximum amount of thinking for the person who will ultimately play it.

All of these problems will have to take place in some kind of land or other, so let's draw a map.

Drawing the Map

We'll assume you have worked out some rough kind of plot line, and you want to draw the map up to see what it all looks like.

Underground Adventure all takes place underground, with a number of different areas, and believe it or not my original map looked like this:



Refining the Map

Well, that was certainly nothing to write home about! However, it worked, because having drawn up all the room numbers I then had a much better idea of fitting the adventure together as a whole, and could commence setting up the problems for the player.

The first thing I did was to label 16 rooms (ringed, in the diagram), and decide that this was where the problems would occur. Then, I had to write down what each problem would consist of, and those are the notes at the left of the sheet.

The brief scrawl at the top was an indication of the general outline of the whole thing. There was to be no finding of treasures, it would all be a question of survival, with the all important mission being to find the key to enable you to open the door that had slammed shut, and get out again.

The notes at the bottom where there as guidelines for one or two of the problems, and from that map the whole game was written.

Well, that's not quite true!

A number of changes were made to the original plan, including the location of one or two of the objects in the adventure area, and before I set fingers to keyboard there were a number of other notes to be made first.

We'll see what they were in the next section.

But for now, you'll have drawn your map, however rough it may be, you've got some idea of the general plot for the whole story, and you know (again roughly) where all the hazards are going to present themselves.

You've got a fairly good idea of everything that will happen to our intrepid explorer, and in chapter 6 you'll see one way of turning these ideas into the necessary data statements that form the fabric of the entire game.

But we're concerned with the programming side of it, rather than the sheer slog of getting all the data statements typed in, so let's start making the transition to the computer.

Moving from Paper to Computer

One of the first steps is to draw a much more sensible looking map, as we've shown over the page for one of the other adventures in this book, the Castle Adventure.

This should be big enough to enable you to list everything you want to in each room, including any objects that are to be found in them, and any hazards that may be expecienced in that room.

Having done that, you'll obviously want to know what all of those objects are! So the next step is to look at the list of hazards as you originally drew them up, and decide what the solution would be to each hazard, bearing in mind that you can only move on to the next part of the adventure after you've solved the problem. In other words, don't put the solution further into the game than the problem!

A list of solutions will give you a healthy list of objects, and these will then form the basis of the list that we'll type into our program later.

With the program set up as it is, although obviously you could modify it if you want to, the routine that checks your data entry only looks at the first three letters of each word. Thus if you had a TRACK and a TRAM in your adventure the program listing would interpret them to be the same object, and you would get some very strange displays being shown up on the screen!

So, if you're going to follow the methods outlined in this book, it helps to give all the objects individual names. As we'll see later, there are enough problems coping with EMPTY BOTTLE, BOTTLE OF OIL and BOTTLE OF WHISKY in Underground Adventure as it is, so we don't want to encourage more of them!

This list of objects will have to be extended beyond a simple list of those generated by the problems and their solutions. We haven't mentioned lamps, or anything like that, so you'll have to have words for LAMP.

What happens if you drop a bottle? If you're going to have it break, you'll also need to have an object something like A PILE OF BROKEN GLASS.

These, and other problems will all have to be thought of before we

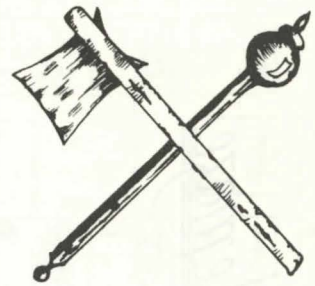
start typing anything in, but inevitably we'll have to add objects to our list as we go along developing the program, but in Basic that is no great difficulty.

Underground Adventure originally started out life with about 48 objects, but ended up having 53, due to circumstances arising during testing of the program that I just hadn't envisaged beforehand. It's nice to track everything first before you start though.

A New and Better Map

This is the final map for Castle Adventure, as re-drawn from an initial scrawl on a tiny sheet of paper.

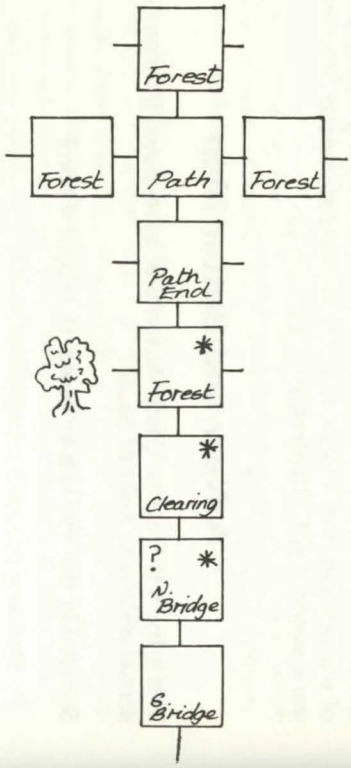
Something like this is a lot easier to program from!

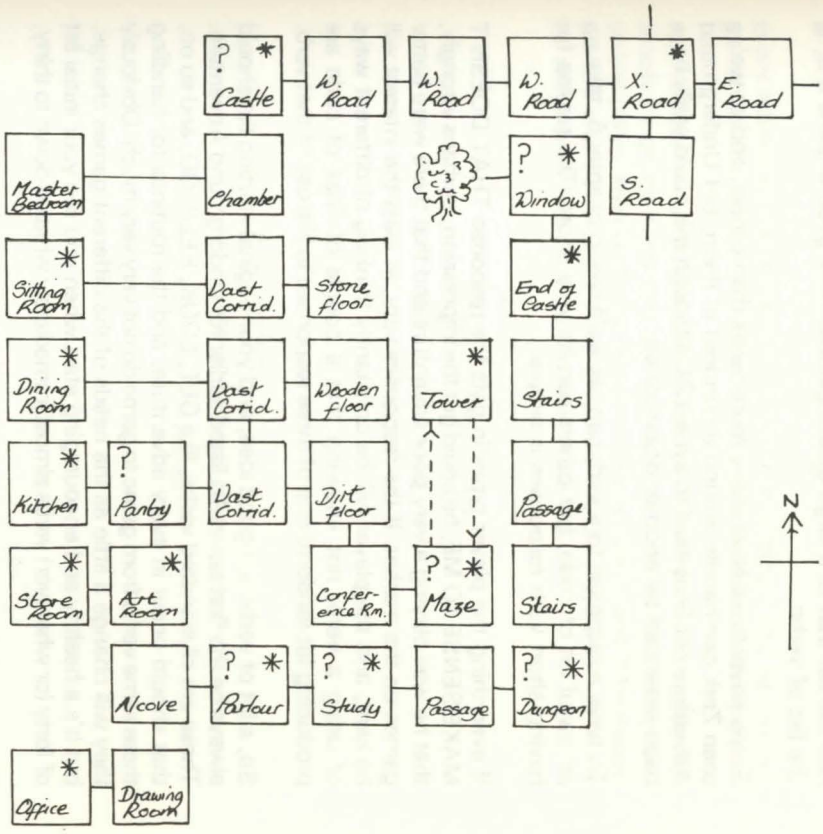


Castlemaze.

?— problem to solve

*— useful objects / treasures





And on to Verbs

As well as our list of nouns, the other great list in any adventure games, and the list that to a large extent dictates how good a game it is, is the list of verbs.

Some adventures have many more verbs than others, and as we've seen Zork can handle around a hundred of them, but Underground Adventure confines itself to a mere 38, although this could easily have been extended by another dozen or so.

To have a response to a verb can, as we'll see in chapter 6, take up an awful lot of code, but others can be very short. The reason for having short verb responses is simple.

If everything the player types in gets the response THAT DOESN'T MAKE SENSE TO ME, he could get the impression, perhaps wrongly, that he was playing a very poor adventure and that there were better games on the market. If the responses vary, at least the interest will be kept, and the player will be constantly thinking of different ways of using a verb, not knowing that a couple of lines of code are producing (at random) one of three responses to the use of that verb.

So, a lot of verbs is a good idea, and your original starting list should always be the first ten verbs listed earlier for Underground Adventure. These are all standard verbs, like GET, LOOK, HELP, GO, and so on, that should occur in every adventure, and the routines for handling these same verbs from game to game do not vary very much. Obviously they will change a little as the needs of the different games change, but it's a healthy and encouraging start when you see your initial list of forty (or whatever) verbs almost immediately whittled down to thirty.

The rest of the verbs are very much up to you, but again they will to a large extent be dicated by the problems that have to be solved.

There is no point in having a can of fly spray to kill the giant fly if the verb SPRAY is not included in the vocabulary. KILL is too woolly a word, and could produce the wrong response if the spray was not being held.

Additional verbs should also be there, just to encourage diversification of response from the computer, and keep the player's interest. A good idea is to give bizarre ideas on the part of the player equally bizarre

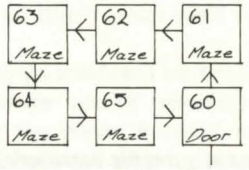
responses from the computer.

It all adds to the humour of playing this type of game.

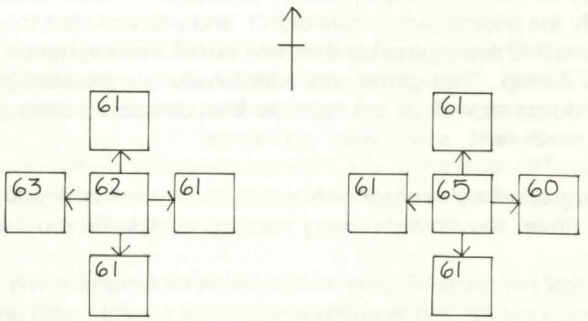
Amazing

Every adventure has a maze of one sort or another, and having got our verbs and nouns, it makes sense to put a maze somewhere.

As the diagram below shows, hard mazes are very easy to construct, simply by giving every one of (say) six rooms the same description, so the player always thinks he's in the same room, and if he makes a move in any one of the three directions you don't want him to move in, why, send him back to the start! Like this :=



Construction of a simple maze using a one-way system.
Taking a wrong turning results in the player returning to room 61.



The only way through the maze is to go W→W→S→E→E.

Some General Rules

Although we've been looking at specifics for the last few pages, for the next half dozen pages or so we'll turn our attention to some general rules when writing these games, and concentrate on five of the most important parts of every adventure game:

1) Movement of characters

2) Responses to inputs

3) Screen displays

4) Picking things up & dropping them down

5) Problem solving

Movement

As your character moves around his wonderful adventure world, there are obviously certain rooms he will and will not be able to go into straight away.

Some rooms will be purely east-west or north-south corridors, in which case it would be rather silly to tell your character that he could move north/south and east/west respectively.

You may or may not display which directions he can move in at all. Certainly the original Adventure didn't, and you were left to your own devices to find every possible direction out of a room, hence the need to draw a map. That game was additionally complicated by having up and down as well as the four cardinal compass points, and also having north-east, south-west and so on.

In Underground we've stuck purely to the four cardinal directions, with up and down movements being handled in specific problem areas.

If you don't want to display the possible directions it will certainly prompt the player into drawing a map, and it might well annoy him considerably to be told over and over again 'YOU CAN'T GO THAT WAY', although interest could be sustained by the addition of the little word YET, thus making him think Aha! perhaps I can go along there later.

Personally, I'm in favour of displaying the available choice of directions, as it speeds up the playing process, but if necessary you can just resort to hints like 'A VAGUE TRACK HEADS OFF TO THE SOUTH', and the like.

It's up to you, but whatever style you pick, make sure that you stick to it throughout the game.

Screen Responses

This is obviously the factor that is most important in keeping the interest and attention of the player throughout the game, and if you want to resort to sound, colour and graphics that's up to you.

However, the simple text-only game without any sound has been used throughout this book, so that's what we'll concentrate on here.

In designing and writing your adventure there is an important factor to bear in mind whenever you're planning the responses to the statements typed in by the player in response to the WHAT NOW prompts, and that is that people playing adventures will never, ever type in what you want them to.

You may have a situation where a player comes to a halt in front of a gate that he can't climb over because the top of it is riddled with barbed wire (an escape from Colditz type adventure?), until he gets hold of a set of wire cutters. You have programmed all your responses to GET GATE, GET WIRE, and so on, and are waiting for the player to get the cutters and type CUT WIRE.

What if he types CUT GATE? What happens then? Or what about something typed in in sheer desperation, as people do, like EAT GATE? Does the gate get swallowed up in a display of apparent relish?

Anticipating people's lines of enquiry is one of the most difficult things to allow for, and will take up an awful lot of program code that will probably never be used.

Still, even if it is used only once at least you'll have the satisfaction of knowing that someone out there will consider that the game that he's playing is an extremely robust, well thought-out adventure.

Always try to anticipate the impossible. You'll never manage all of it, of course, and will have to rely on some stock I DON'T UNDERSTAND

type responses, but a few of those mixed up and one picked out at random will keep the interest from flagging.

And never forget the use of the word YET. It will keep a player trying long after the more straightforward 'YOU CAN'T OPEN THE GATE' will.

So the golden rule here must be to keep it interesting, and try to anticipate everything that the player might type in. You won't get them all, but at least you can conjure up some different responses.

Also, a large list of verbs is a great help here: even if the responses are only short and sweet, at least the player will be seeing something different on the screen.

Screen Displays

To a small extent we've covered this one already, but it's worth going over some of the ground again.

The use of graphics has been deplored often enough before now to render any comment here redundant, although you might think the odd display of a sword or amulet every now and again might liven things up a little. But nothing can beat the written word.

Sound is a different question, and the arguments concerning this are almost as legion as those concerning the use of graphics.

My own view is that if you're going to use sound, it must be done extremely well, as the computer is capable of a very complex series of sound outputs. If you're only going to give a little beep every now and again, it's hardly worth the effort of putting it in there in the first place, and you'll soon have people racing for the volume control and a blessed silence.

If done well, it can greatly enhance a game, as people who have played the Temple of Apshai on a Commodore 64 will know: the use of sound is very good here, and the whole atmosphere of moody, omnipresent danger is well presented.

On the other hand, all their programming efforts are wasted if somebody turns the volume down. Be prepared to have sound in your programs if you wish, but don't be disappointed if everyone immediately adopts to play out the game in silence.

The words that are displayed in the screen are obviously dictated by the responses you've allowed for, but an overall attractive layout is to be desired, usually using lower case, since most people seem to prefer that for some reason. Perhaps it's more restful on the eyes as you do battle against a giant troll!

Silly little things can so easily spoil a game in this area - if your room descriptions overlap the edge of the screen so that words are split up, or an inventory list causes some of the objects to be displaced against each other, or even if your output is riddled with spelling errors.

It doesn't take too long to check all of these things, and the results are well worth the trouble. A neat adventure is more likely to be played than a badly spelt, badly laid out one.

The golden rule here? Keep it simple, but keep it tidy.

Picking Things Up and Putting Them Down

Two of the most important words in the adventurer's catalogue are GET and DROP, and in chapter 6 we take a more detailed look at these two words as they apply to the game Underground Adventure. However, a few general words of advice before we get to that chapter.

Obviously, in any game there will be a number of things that you can pick up, and a number that you can't, with the former probably far outweighing the latter. Nevertheless, all possible occurences must be taken into account, and just because you know that the BARRED GATE is too heavy to carry, that won't stop virtually every player who comes along from attempting to pick it up and walk off with it.

Another annoying thing to find in any adventure program is a description that might read something like 'YOUR PROGRESS IS HALTED BY A SOLID WALL OF ROCK', and when you type in GET WALL, the only response is 'I CAN'T SEE ANY WALL HERE', or 'I DON'T KNOW WHAT A WALL IS'.

Look out for that one, for although it can be covered by a blanket response of NO!, that is not very good practice and will certainly not produce an excellent adventure game. Far better to have a response actually geared to the request like 'THE WALL CANNOT BE CARRIED', or something like that.

Some things in a game are only meant to be carried after certain actions

have taken place, in which case you'll need a number of variables to flag the progress of the adventurer, and you'll also have to use the word YET to keep the level of interest there. 'YOU CAN'T CARRY IT YET', will have someone attempting to carry whatever IT is until the cows come home, even if they never can carry it.

When dropping things, a subtle level of difficulty comes into the game. In Underground, after you've made friends with the bear and he's happily trundling around the caves after you, dropping anything will cause him to think that you're throwing things at him, and he'll disappear in a sulk to a random part of the caves, never to be GOT again.

Dropping bottles is usually a good one, since you can have them break on your adventurer, thus rendering them useless for the rest of the game. The original Adventure had as one of its treasures a Ming Vase, but dropping it caused it to smash into delicate little pieces, unless (of course!) you'd taken the precaution of placing a pillow underneath it.

GET and DROP are fun, and don't confuse GET with TAKE. The two words are not the same! For instance, people talk about TAKEing medicine, not GETting it!

Problem Solving

The key to any adventure is how good and how complicated the problems may be in a game, but don't make it too complicated to get started, or your adventurer might give up in disgust and never play an adventure game again.

Encourage people by at least letting them get started, and then pile the problems on, preferably making the first few lean towards the easy side, and have them get harder as the game gradually progresses.

The Scott Adams games are particularly good here, as it is always possible to get somewhere at a first sitting, even if that somewhere isn't very far, and you can gradually improve your progress just about every time you play the game.

Problems usually have to be solved in a set order too, in that solving one leads you to another, which gives you a clue to an earlier hazard you were puzzling over, which in turn sets you off somewhere else, and so on.

The number of problems in a game is obviously up to the writer of the game, but too many will soon discourage people. A problem every room will become totally boring after only a short playing session, but the intervention of a few rooms between hazards will soon perk up the player, even if he does walk into another one almost immediately.

Some problems will have to rely on a number of events taking place. In Underground Adventure, one of the hazards you're faced with is a very steep incline that you can't climb up by yourself, and the rope that you've previously used to shimmy down a steep drop isn't of any use to you here.

A little thought, or a read of the old parchment if you find it, leads you to conclude that you must build yourself a ladder, for which you need some wood (you recall a plank somewhere), some nails, and something to hit it all into shape with. Aha! The axe. But the wood has to be cut into shape first, before you can make a ladder. Only when you've got a collection of neatly cut timber can you make the ladder, and proceed to the next set of problems.

So, keep up the interest, and let people get a little further each time. And above all, don't make it an unsolvable adventure!

Program Listings From Underground Adventure

In this section of chapter 4 we're going to give you all the lines of code that you haven't already seen, and which won't be found in the sections on verbs and data in chapter 6. So if you're going to type the whole thing in, this is the place to look for that missing piece of code that's been puzzling you.

Of course, in common with the rest of the book we're not going to present the code without any sort of explanation.

Each line will, where appropriate, be fully explained, along with an idea of how that line could be incorporated into a program of your own.

Some of the sections of the program that we'll be covering here include the rules about what happens when the bear is following you, the fights between the gargoyle and yourself, the checks to see whether you're carrying a bottle of oil, a bottle of whisky or just a plain old empty bottle, and most important of all the lines that deal with inputting data and analysing that data as it is typed in.

We'll take each section as it comes in the game, rather than diving about all over the place, so that you'll be able to see a coherent whole being slowly built up, with all the missing parts slotting logically into place, bearing in mind of course that you've already seen the movement listing, and that the data comes later on.

So, without further ado, let's get into the game.

If, by the way, you think that we've sometimes left rather large gaps on the pages, this is very true, but it's there for you to use to put your own notes in when adapting these routines for your own purposes, so the book builds up to become more YOUR book of exploring adventures rather than just a text book.

Don't worry: we'd have charged you the same even if we had filled up every page!

The Bear and the Verbs

This part of the program deals with the presence of the bear, and the handling of the verb number as it comes back from the routine in lines 390 to 407, which we'll come to in a minute.

```
 206 IF tb THEN LET o(9)=cp
 208 IF tb=1 THEN PRINT : PRINT "You're being foll
owed by a tame bear."
 209 IF tb AND cp=45 THEN PRINT "The ladder snaps
in two under   the weight of the bear!!": LET o(13
)=0: LET p(45,2)=0: LET cp=45
 210 GO SUB 390
 220 IF vb=34 THEN GO TO 1950
 225 IF vb>9 AND a$="" THEN PRINT "You need a dire
ct object.": GO TO 210
 226 IF (vb=2 OR vb=6) AND a$="" THEN PRINT "You'r
e not helping me here!": GO TO 210
 227 IF a$<>"" AND vb=1 AND no=0 THEN PRINT "That
doesn't make any sense.": GO TO 210
 240 GO TO vb*50+250
```


Explanation

Line 206: check the bear flag (TB) and if this is set put the bear (O(9)) into the current room (CP).

Line 208: check for bear again, and if present print up a simple message.

Line 209: check the bear is there, and if he is and you're climbing up the ladder in room 45, then the ladder snaps in two! The ladder disappears (O(13) = 0), the south exit from room 45 is closed (P(45,2) = 0), and you are back in room 45.

Line 210: gosub to the input routine.

Line 220: if the verb number is equal to 34 (JUMP), then go to line 1950.

Line 225: if the verb number is greater than 9, but you've only typed in one word, print out a simple message and start again.

Line 226: check for GET and DROP.

Line 227: if the verb number is 1 (GO), and you've typed in a non-recognised word (NO = 0, and A$ = "something"), then print a simple message and try again.

Line 240: take the verb number and go to the appropriate line in the program.

Data Validation Routine

This checks to see what you've typed in from the input routine in lines 8000 to 8002, which we'll get to later, and splits your input up into a verb and a noun, where applicable.

```
 390 PRINT : PRINT : PRINT "What Now * ?";: GO SUB
 8000: PRINT m$: PRINT
 391 LET a$="": LET b$="": LET vb=0: LET no=0
 392 LET lc=LEN m$: FOR i=1 TO lc: IF m$(i TO i)<>
" " THEN LET b$=b$+m$(i TO i): NEXT i
 393 LET e$=b$: IF e$="go" THEN GO TO 397
 394 IF LEN e$<3 THEN PRINT "I don't understand yo
u at times!": GO TO 210
 395 LET b$=e$(1 TO 3): FOR i=1 TO nv: IF b$=v$(i)
 THEN LET vb=i: GO TO 398
 396 NEXT i
 397 LET vb=1: LET a$=b$: GO TO 402
 398 IF LEN e$+1>=LEN m$ THEN LET no=0: RETURN
 399 GO TO 402
 400 GO TO 200
 402 LET a$=m$(LEN e$+2 TO LEN m$)
 403 IF LEN a$<3 THEN PRINT "Je ne comprends pas!"
: GO TO 210
 404 LET h$=a$: LET a$=a$(1 TO 3): FOR i=1 TO nn:
IF a$=j$(i) THEN GO TO 407
 405 NEXT i
 406 LET no=0: RETURN
 407 LET no=i: RETURN
```

Explanation

Line 390: print up the WHAT NOW? * prompt, and go to the routine at 8000 to get the input of data.

Line 391: declare a few variables (length of noun, length of verb, verb number and noun number) to equal zero.

Line 392: perform a loop LC times, where LC is the length of the input string M$. Carry on until you find a space in M$, by searching through one character at a time.

Line 393: see if the verb is GO, in which case assign a variable and jump to later in the routine.

Line 394: perform a check for the length of the verb, otherwise the slicing action in line 395 won't work!

Line 395: set E$ equal to the verb, and take the first three letters of it, since that's all we analyse. Perform a loop NV (number of verbs) times, to see if we recognise the verb, and if we do set the verb number equal to I : the Ith verb. Then GOTO line 398

Line 396: carry on the NV loop, because we don't recognise the verb yet.

Line 397: there's no verb, therefore only one word was typed in. Assume the verb is an implied GO, as in GO NORTH. Set the noun string equal to the verb string (i.e. that which was typed in as M$). GOTO line 402

Line 398: if the length of the string plus 1 is equal to or greater than the length of the input string, i.e. we've only typed in one word, then there is no noun, and we return from the subroutine with a single verb.

Line 399: GOTO line 402, because of interruption in next line!

Line 400: action the verb LOOK by going to line 200 (due to use of computed GOTO in line 240)

Line 402: find the noun A$ from the original input string M$, by taking the right side of M$, starting at the character after the space.

Line 403: perform check for length of string again (due to slicing in line 402) and action accordingly.

Line 404: set H$ equal to the noun. Check to see if we recognise it by going through the loop NN (number of nouns) times and checking to see if it's equal to a known noun. If it is go to line 407.

Line 405: continue loop because we don't recognise the noun.

Line 406: unrecognised noun, so set noun number to zero and return from subroutine.

Line 407: recognised noun, so set noun number to that of the Ith noun, and return from subroutine.

Death or Glory!

This is the death routine, and is called up from a number of spots in the program in case of an untimely demise.

```
612 PRINT "You appear to have blown it.    You're
dead.": GO SUB 6800
614 PRINT "Do you want another game (YorN)?"
616 IF INKEY$="" THEN GO TO 616
617 IF INKEY$="y" THEN RUN
618 IF INKEY$="n" THEN CLS : PRINT "Bye.": STOP
620 GO TO 616

6800 BEEP 1,0: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEE
P 1,0
6801 BEEP 1,0: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEE
P 1,0
6802 BEEP 1,3: BEEP 1,5: BEEP 2,7
6803 BEEP 1,3: BEEP 1,5: BEEP 2,7
6804 BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5:
 BEEP .5,3: BEEP .5,2: BEEP 1,0
6805 BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5:
 BEEP .5,3: BEEP .5,2: BEEP 1,0
6806 BEEP 1,0: BEEP 1,-5: BEEP 2,0
6807 BEEP 1,0: BEEP 1,-5: BEEP 2,0
6808 RETURN
```

Explanation

Line 612: print the 'you're dead!' message, and GOTO subroutine at 6800.

Line 614: ask for another game.

Line 616: wait for a key to be pressed.

Line 617: check for Y, and if pressed RUN the program again.

Line 618: if they've typed 'N' then print out a goodbye, and stop the program.

Line 620: no one's pressed anything, so loop back to line 616 and continue to do so until they do.

Lines 6800-6807: subroutine to lament your death. Plays Mahler's funeral march.

The Start and the End

These lines appear at the very start of the program, as you get the door slamming shut behind you, and the very end, if you ever manage to get out alive. In reverse order we have:

```
2510 CLS : PRINT "And now you can leave the caves!
": PRINT : PRINT "Congratulations!": PRINT : PRINT
 "And have a safe journey home!"
2512 STOP
```

Explanation

Line 2510: you're out, called up from another line in the program, in the OPEN routine, lines 800 to 816, so print a message of congratulations and stop the program!

```
5200 CLS : PRINT "Oh dear, the gate to the caves
appears to have slam shut!"
5202 PRINT : PRINT "That's torn it.  You'll have t
o find the key now before you can get out."
5204 PRINT : PRINT "But don't worry.  It's in here
  somewhere."
5205 LET p(3,1)=0
5206 LET gf=0: GO TO 210
```

Explanation

Lines 5200-5204: print message to say gate's closed behind you, called from line 310 in the GO routine.

Line 5205: close off the north exit from room 3 (P(3,1))

Line 5206: set the gate flag, GF, and return back to line 210 again.

Checking for Bottles and Torches

This routine is called up many times in the program, and is used to check to see whether you mean a lit or an unlit torch, or a bottle of whisky, a bottle of oil, or an empty bottle.

This is necessary because the data checking routine covered earlier will halt at the first noun it recognises, and the response in all the verbs will obviously depend on whether you've got the relevant torch or bottle. So we must adjust the noun number No accordingly.

```
5300 IF no=45 AND o(46)=-1 THEN LET no=46: RETURN
5302 IF no=19 AND o(51)=-1 THEN LET no=51: RETURN
5304 IF no=19 AND o(52)=-1 THEN LET no=52: RETURN
5306 IF no=18 AND o(51)=-1 THEN LET no=51: RETURN
5308 IF no=39 AND o(52)=-1 THEN LET no=52: RETURN
5310 RETURN
```

Explanation

Line 5300: if the object number is for the old torch (O(45)), and you're carrying the blazing torch (O(46) = -1) then change the noun number accordingly. Return from this subroutine to whatever part of the program called it up.

Line 5302: if the object number is for the empty bottle (O(19)), and you're carrying the bottle of oil (O(51) = -1) then change the noun number accordingly. Return from this subroutine to whatever part of the program called it up.

Line 5304: if the object number is for the empty bottle (O(19)), and you're carrying the bottle of whisky (O(52) = -1) then change the noun number accordingly. Return from this subroutine to whatever part of the program called it up.

Line 5306: if the object number is for the pool of oil (O(18)), and you're carrying the bottle of oil (O(51) = -1) then change the noun number accordingly. Return from this subroutine to whatever part of the program called it up.

Line 5308: if the object number is for the pool of whisky (O(45)), and you're carrying the bottle of whisky (O(52) = -1) then change the noun number accordingly. Return from this subroutine to whatever part of the program called it up.

Line 5310: none of these options, so return from the subroutine.

The Hostile Gargoyle

This is the routine that handles the hostile gargoyle, and checks to see whether he or you have been successful in your knife and axe throwing attempts.

```
6000 PRINT : PRINT "There's a hostile gargoyle pee
r-ing at you from the shadows!"
6001 IF INT (RND*100)>98 THEN GO TO 6020
6002 PRINT "He has a knife!  He throws it atyou!":
 LET o(40)=cp
6004 IF INT (RND*100)>98 THEN PRINT "He's killed y
ou!": GO TO 612
6006 PRINT "It missed!": RETURN
6010 IF INT (RND*11)>1 THEN PRINT "You've killed a
 gargoyle!": LET o(40)=0: GO TO 6014
6011 PRINT "You missed him!": LET o(40)=cp
6012 LET o(4)=cp: LET zz=zz-1
6013 FOR i=1 TO 50: NEXT i: GO TO 200
6014 LET np=0: GO TO 6012
```

Explanation

Line 6000: print out a hostile message.

Line 6001: if the random number generated is greater than 98, in a range of 0 to 99, then the gargoyle turns into a thief at lines 6020 to 6040.

Line 6002: print out 'He's got a knife and throws it at you', and place the gargoyle in the room CP, meaning that he's here until the bitter end.

Line 6004: if the random number generated is greater than 98, on a scale of 0 to 99, the gargoyle has been successful and killed you. GOTO the death routine.

Line 6006: a shoddy shot and he missed, so return from this subroutine.

Line 6010: if the random number generated is greater than 1, on a scale of 0 to 10, then you've killed him, so jump to line 6014 and remove the gargoyle (0(40) = 0).

Line 6011: yah boo! you missed, so the gargoyle stays there.

Line 6012: your axe (O(4)) is placed in the room CP, the number of objects that you're carrying (ZZ) is therefore reduced by 1

Line 6013: delay to enable reading off message on screen, then GOTO line 200

Line 6014: clear gargoyle present flag, since you've killed him, and GOTO line 6012

The Thieving Gargoyle

The gargoyle has turned into a thief, and here we check to see what he can take.

```
6020 PRINT "He appears from the shadows and steals
:": LET gs=0
6022 IF o(2)=-1 THEN LET o(2)=63: PRINT o$(2): LET
 gs=gs+1
6024 IF o(7)=-1 THEN LET o(7)=63: PRINT o$(7): LET
 gs=gs+1
6026 IF o(14)=-1 THEN LET o(14)=63: PRINT o$(14):
LET gs=gs+1
6028 IF o(16)=-1 THEN LET o(16)=63: PRINT o$(16):
LET gs=gs+1
6030 IF o(19)=-1 THEN LET o(19)=63: PRINT o$(19):
LET gs=gs+1
6031 IF o(33)=-1 THEN LET o(33)=63: PRINT o$(33):
LET gs=gs+1
6032 IF o(34)=-1 THEN LET o(34)=63: PRINT o$(34):
LET gs=gs+1
6034 IF o(38)=-1 THEN LET o(38)=63: PRINT o$(38):
LET gs=gs+1
6036 IF o(44)=-1 THEN LET o(44)=63: PRINT o$(44):
LET gs=gs+1
6038 IF gs=0 THEN PRINT "Nothing!  You were lucky!
"
6040 RETURN
```

Explanation

Line 6020: print simple message.

Line 6022: if you're carrying the staff (O(2)), then place it in the maze (CP = 63) and increase the thief counter GS.

Line 6024: if you're carrying the dynamite (O(7)), then place it in the maze (CP = 63) and increase the thief counter GS.

Line 6026: if you're carrying the nails (O(14)),then place them in the maze (CP = 63) and increase the thief counter GS.

Line 6028: if you're carrying the mirror (O(16)), then place it in the maze (CP = 63) and increase the thief counter GS.

Line 6030: if you're carrying the bottle (O(19)), then place it in the maze (CP = 63) and increase the thief counter GS.

Line 6031: if you're carrying the brick (O(33)),then place it in the maze (CP = 63) and increase the thief counter GS.

Line 6032: if you're carrying the fly spray (O(34)), then place it in the maze (CP = 63) and increase the thief counter GS.

Line 6034: if you're carrying the sword (O(38)), then place it in the maze (CP = 63) and increase the thief counter GS.

Line 6036: if you're carrying the matches (O(44)), then place them in the maze (CP = 63) and increase the thief counter GS.

Line 6038: if the thief counter GS hasn't been set, then nothing has been stolen, so print a simple message on the screen.

Line 6040: return from the thieving subroutine.

Of Panthers and Crevices

Two separate routines here, one for dealing with the panther in the presence of the bear, and one for the problem encountered in room 53: the narrow crevice.

```
6054 PRINT "The panther flees at the sight  of the
 bear!": LET p(42,2)=43: LET o(11)=0
6055 LET p$(42)="walking past the scent ofold pant
her.": GO TO 5004
```

Explanation

Line 6054: print appropriate message, clear south path from room 42, and remove the panther (O(11))

Line 6055: change the room message, and back to line 5004.

```
6300 LET oc=0: FOR i=1 TO lo: IF o(i)=-1 THEN LET
oc=oc+1
6302 NEXT i
6304 IF oc>1 THEN PRINT "Something won't fit throu
gh.": GO TO 210
6306 IF o(37)<>-1 THEN PRINT "Sorry.  You can't fi
t through.": GO TO 210
6308 LET cp=100: PRINT "The stone glows with a shi
ny    light and lets you through."
6310 GO TO 210
```

Explanation

Line 6300: set object counter OC to zero, and go through a loop LO times to check for the presence of every object. If you find one, increase the variable OC.

Line 6302: next time around!

Line 6304: if you're carrying more than one thing, then print suitable message and go to line 210

Line 6306: if you're not carrying object 37 print suitable message and GOTO 210.

Line 6308: puts you in room 100, prints message.

Line 6310: back to line 210 again.

May I Introduce You?

This is just the introduction to the game, and doesn't really need any explanation. The first line just sets the paper, ink and border colours.

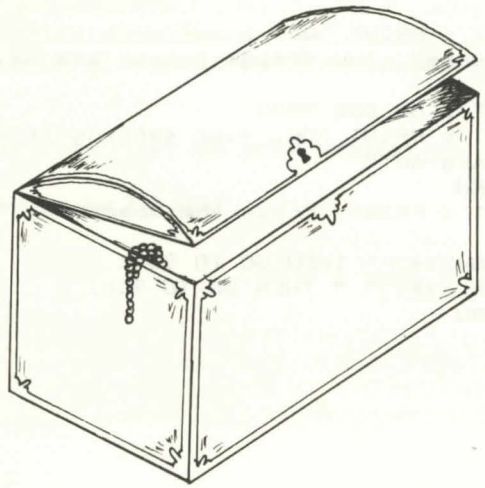
```
9000 CLS : FOR i=1 TO 88: PRINT "DuckSoft";: NEXT
i: INVERSE 1: PRINT AT 11,9;" HELLO THERE! ": INVE
RSE 0: FOR i=1 TO 250: NEXT i
9002 PAPER 6: INK 0: BORDER 1: CLS
9004 PRINT "Welcome to Underground Adventure": PRI
NT : PRINT "Here you are,miles away from    home,
trying to decide how to   spend your afternoons."
9006 PRINT : PRINT "Do you look for peace and
  solitude, or do you look  for   danger and adven
ture?"
9008 PRINT : PRINT "Of course,you decide to look f
oradventure!  Life's too short!": GO SUB 9500
9010 CLS : PRINT "You are on a dusty old beaten
track, heading south towards    some caves hidden
deep in the   hillside away in the distance."
9012 PRINT : PRINT "It is rumoured that the caves
  are dangerous.  Pah! you say,   and quite right
to."
9014 PRINT : PRINT "Who knows what is to be found
  inside them?  You decide to go  and have a look.
"
9016 PRINT : GO SUB 9500
9018 PRINT : PRINT "OK, just setting it all up for
  you.  Hang on!"
9020 RETURN
9500 PRINT : PRINT "Press the space key to continu
e."
9501 IF INKEY$="" THEN GO TO 9501
9502 IF INKEY$<>" " THEN GO TO 9501
9504 RETURN
```

Input Subroutine

This is simply using the built-in input routine of the Spectrum, since this is adequate to prevent anyone escaping from the program by any way other than using the break key while the program is doing a spot of decision making.

The response typed in is echoed onto the screen again in the What Now routine in line 390 onwards.

```
8000 INPUT LINE m$
8002 RETURN
```



5
Creating Your Own Adventures

Introduction

We've already mentioned that one of the hardest parts of creating an individual adventure game is making it just that: individual.

More and more brave new worlds are being explored every day, and a glance at any computer magazine, particularly the advertisements inside it, will reveal that there are many, many adventures on the market for all kinds of machines, and the themes used seem to range from the sublime to the ridiculous, from Colossal Caves to Pi-Men.

Five New Adventures

To the newcomer, eyeing this vast range of adventure games, it must seem that there is nothing new under the sun, and that any attempt to create a new, wonderfully different, adventure world is doomed to failure.

Nothing could be further from the truth, and in this section we're going to outline five full adventures for you, some old, some new, but all with one thing in common: they haven't been written yet.

Acknowledgment

So, if any of you take up the challenge, I hope one day to see adventures based on these themes on the market. No royalty would be charged, no copyright laws infringed, but an acknowledgment would be nice!

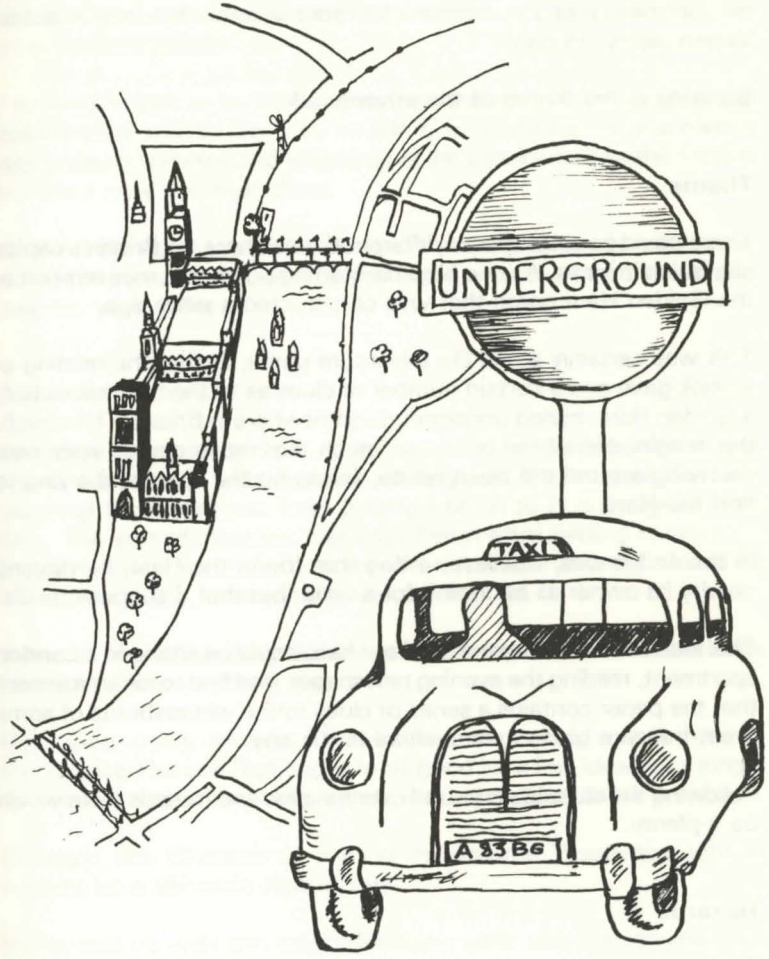
The five areas that we'll look at are all individual in their way, and none of them cross over into any of the others: they are five unique scenarios that could easily be built up into complete and enjoyable games.

We won't be giving you any maps, so that you can construct the entire game for yourself, but an overview of the game, along with a selection of possible problems, and the corresponding objects to go with them will be given.

To round off this section, we'll give a complete overview of the art of designing a new adventure.

But for now, let's head off in search of fame and glory, and arrive in...

The Streets of London



Introduction

This would be a relatively easy map to construct, since London is a well documented town. Of course, you could always choose your own town as the base for a game if you wanted to, but an adventure based on London is probably destined for more success than one based on Wigan: sorry, Wigan!

So what is the theme of the adventure?

Theme

There could be a number of different themes here, as Britain's capital city is rich in ideas. As one possible starting point, you may remember the Golden Hare game that was constructed a while ago.

This was certainly a real life adventure game, in that the reading of a book gave one a certain number of clues as to the whereabouts of a Golden Hare, buried underground somewhere in Britain. This caught the imagination of the public so much that many people were sent scurrying around the countryside, following the clues and trying to find the Hare.

In the end it was, I believe, a dog that found the Hare, by digging nearby its owner as he took it for a walk, but that, I suppose, is life!

This idea could be adapted, and our hero could be sitting in a London apartment, reading the evening newspaper, and find to his amazement that the paper contains a series of clues to the whereabouts of some great treasure hidden somewhere in the city.

Following the clues leads you all over the city, and hazards there would be a-plenty.

Hazards

The underground could go on strike, and you'd find yourself having to take a bus. None come for hours, thus losing valuable time, and then four of them turn up at once, only one going in the right direction. Which one do you catch?

You could try taking a taxi, but the taxi takes you on a scenic tour of London that takes hours before you get to your proper destination. Then the fare is too high, you haven't got enough money, and you have to haggle with a noisy taxi driver in the middle of the streets of London.

There are many other possible problems that one could construct, all based very much on real life in this re-construction of a real town into an adventure game.

You would have to be careful that the details about the locations of objects were true to life. You couldn't, for instance, have someone taking the Victoria line and ending up at the Barbican, since the Victoria line goes nowhere near there.

On the other hand, just about every diary ever printed contains a map of the London underground, so you could soon chart up a reasonable map for your game.

Other Adventures

Or indeed, the underground could also be used as the basis for your whole adventure, with a series of Reginald Perrin type disasters occuring to prevent you from getting from A to B in the given time limit. The sort of disasters that kept Perrin from getting to work on time every day: a wombat escapes from London Zoo and chews its way through the underground line, and so on.

A tour of London could give the would be adventure writer more ideas than just about anything else.

How about going down to Kew Gardens, and taking a walk through the Tropical House? That ought to be good for a few ideas for a jungle adventure, with man-eating plants and other hazards to avoid.

Or again, the Chamber of Horrors in Madame Tussauds ought to conjure up a demonic idea or two.

But to end up with one solid adventure, we'll take that original idea of some treasure being buried under the streets of London, and all you know is that it's in London somewhere.

Scenario

Reading the evening paper one Monday night in your apartment, you discover a strange article that seems to point to the location of a buried treasure buried deep underground somewhere in the city of London.

The only clue that the article gives to this location is that the treasure originally came from 'Underneath the Arches', and was moved from there many years ago.

You decide to set off in search of adventure, and head towards the arches.

Thus we could start off, and the first problem could be to get from the apartment in Muswell Hill to the Arches, which (in our adventurer's mind) would presumably be the arches behind Charing Cross Station.

After solving that problem (GET BUS, BUY TICKET, and so on), arriving at the arches would reveal a pub called the Ship and Shovel.

Is this the next clue? Does our intrepid hero have to go off and acquire a shovel and find a ship? Or does he merely go into the pub?

ENTER PUB

OK.

THE BARTENDER IS AUSTRALIAN, AND SAYS THAT 'DOWN UNDER IS THE ONLY PLACE TO BE'

WHAT NOW *

Down under? Another clue, and so we go off in search of a shovel, and somewhere to dig underground.

This could be the start of a very intriguing adventure, set as it is in real life situations (one of the bartenders really is Australian!) that would give the player a sense of familiarity, but pitching those situations into a different role from the norm.

The game could encompass many famous London landmarks, each holding a clue on the trail, and each presenting its own particular problems. Big Ben would presumably feature somewhere, and, as in

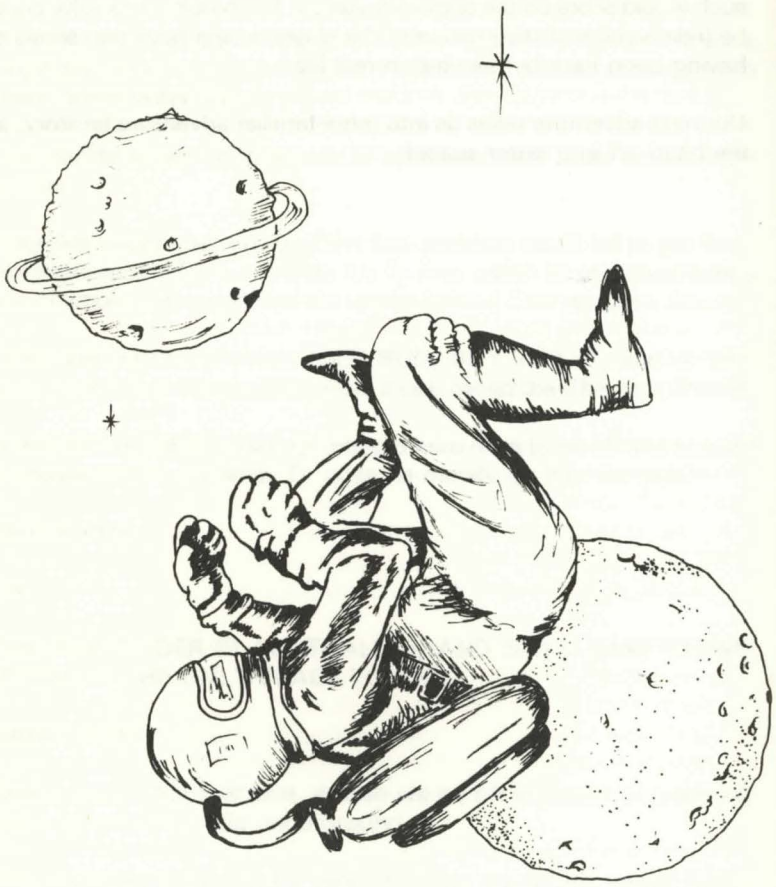
the famous scene in the re-make of the Thirty Nine Steps, a hazardous climb out onto the clock face could be another hazard to overcome.

Conclusion

An adventure like this is a departure from the usual themes, and as such would score on the originality stakes. The problems to solve could be (relatively) realistic ones, and the player would have that sense of having been here before, but in real life.

Our next adventure takes us into more familiar adventure territory, as we head off into outer space!

Lost in Space



Introduction

There have been a number of adventures set in outer space, and the classic Star Trek series of games that have been written for every computer under the sun, were probably the inspiration for a number of early games in this genre.

However, most of the Star Trek ones tend to be tactical battles, rather than true adventure games, and one has to go beyond the usual 'You are in command of the US Enterprise, and your mission is to destroy the Klingons' type of game, and put the player into a true adventure setting.

Theme

One possible idea would be to have your hero cast up on a dim and distant planet, deep in space, with a damaged spacecraft that needs rebuilding before he can take off again and get back home.

Here we could use some of the more traditional ideas of adventure games, but put into a modern setting. For example, the majority of thud and blunder adventures require that you carry a torch around with you. This could be replaced in this game by an oxygen tank, with a limited amount of gas, so that the mission would have to be completed in a set time.

There would be a number of different settings in this sort of adventure. One part would take place on board the damaged ship, in a search for plans, more oxygen, and equipment to repair the damage, and if the hero was silly enough to be wearing the oxygen tank on board he would lose valuable time when it came to going out onto the planet's surface.

Having thoroughly explored the ship, and cut past tangled metal, opened locked doors, and any other hazards you could dream up, the time would come to go outside, with oxygen, and the living gargoyles and little dwarfs that inhabit older adventure worlds could be replaced by hostile aliens and strange life forms.

Alien Hazards

To any reader of science fiction there should be no problem in coming up with a million and one problems for an adventurer to solve as he explores the surface of a hitherto undiscovered planet. Undiscovered, because then he won't be able to anticipate any of the problems that might arise.

Here too, as in the Streets of London, a reasonable amount of realism must come into the game, but your imagination can have a much freer rein deep in outer space.

Perhaps one could use the discovery of planet-like bodies around the star Vega, in the constellation of Lyra. A mission could be sent to explore, but a technical hitch causes the ship to crash and leaves you as the sole survivor. Being a good few light years away from earth it's impossible to signal for help, and in any case the radio probably wouldn't work, so you'd be on your own in a do-or-die mission oriented adventure.

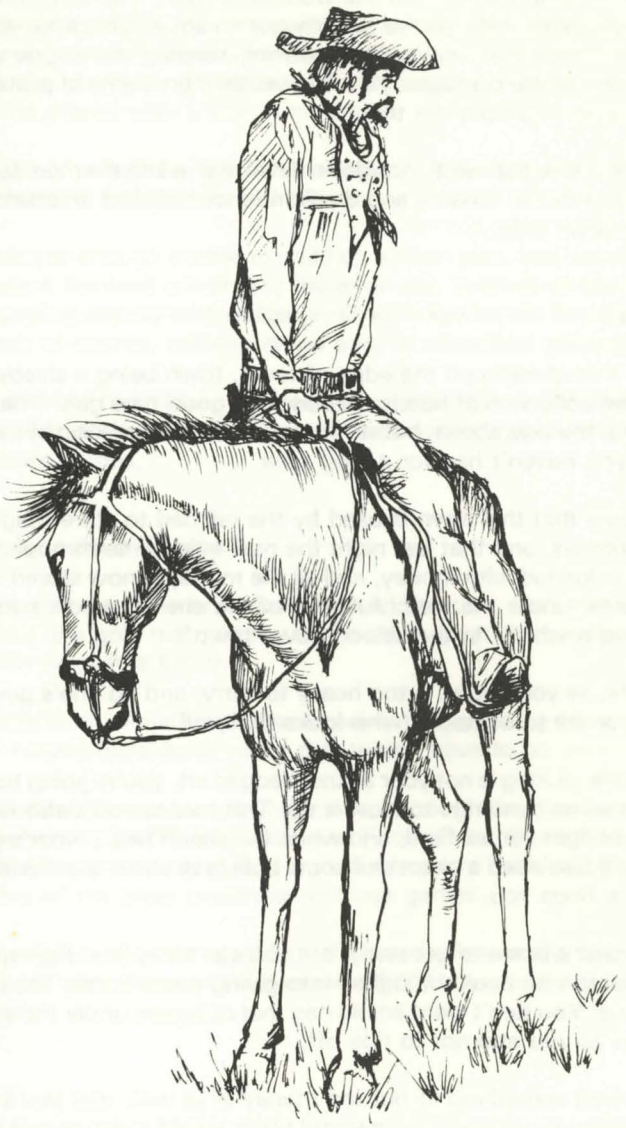
This could even be written as a two-stage adventure, in that you get the spaceship working again, but instead of steering your course for home you head off into the wilds of outer space, since the steering device hasn't been fixed properly, and then the exploration would take place aboard the ship in an effort to correct the mistake before it was too late, and you ended up in Andromeda or something. I knew I should have turned left at the Pleiades!

Conclusion

Outer space is rich in many things, and it is certainly a rich source of inspiration for the would-be adventure writer. A nice touch could be added by having various cameo roles from E.T., Darth Vader, Patrick Moore, and other stars of screen and space.

But now we'll turn our attention down home again, and travel back in time to the wild west!

Go West



Introduction

To anyone who's ever seen the wonderful Marx Brothers movie of the same name, well, you've already got an adventure game written for you! Trains that come off the tracks, keeping the engine going by burning all the carriages, all the essential ingredients of problems, disasters and humour are there.

But for the idea that we'll consider in detail, we're into the more familiar territory of Butch Cassidy and the Sundance Kid, and an attempt to rob the town safe.

Theme

You're a desperado on the edge of town, town being a sleepy little mid-west collection of hotels, saloons and good-time gals. The stars twinkle in the skys above, but are not joined by the twinkling of money, which you haven't had for a long time.

You know that this town is used by the railroad to store freight on long journeys, and that last night the mail train came through. That train was loaded with money, and all the money is now stored in the town safe, under the watchful eyes of the sheriff, who's currently watching a whisky in the saloon down town.

The safe, as you know, is too heavy to carry, and no one's going to sell dynamite to someone who looks like you!

Since safe-picking is not your acknowledged art, you're going to have to steal some dynamite to blow it up. This means you'll also need a source of light somewhere, and when the sheriff hears what's going on, you'll also need a pistol and some bullets to shoot it out with him when he finds you.

You'll need a horse to get away, but you can't buy one. Perhaps the local blacksmith could be bribed into giving you a horse, but only a good one. You don't want an old nag that collapses under the weight as soon as you attempt to ride off.

You'll need something to put the money in as well, and you'll need a small light to work by. A powerful torch would make people come and investigate, and the game would be up, you'd be slung in jail,

and somehow you'd have to get out again.

Building up the Game

The above scenario could be built into a long and enjoyable game, with many more hazards than the ones we've detailed above. The pitfalls are obviously immense, and the number of different scenes could be played with a fine humour.

Perhaps some real characters from days of old could be included, like Doc Holliday, Buffalo Bill and the rest.

It's a simple enough matter to build up a town plan, and some of the characters involved are already there for you, in terms of the sheriff, the bungling deputy who obligingly drops a key on the floor: just out of reach of course, nothing is too easy in adventure games.

From this one basic idea, there are many other themes that could be developed, and which readily lend themselves into adaptation as adventure games.

Variations on a Theme

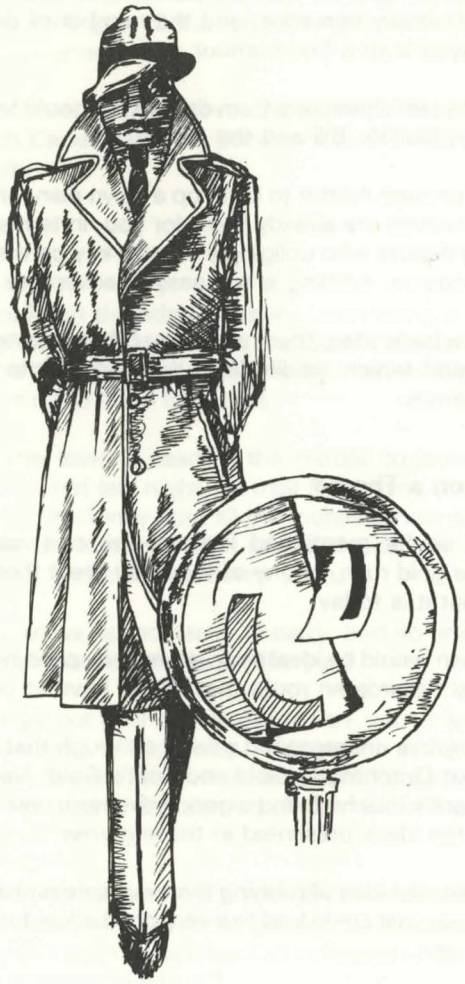
We haven't so far mentioned Indians, the civil war, the railroad pioneers, the gold rush, or any of the other great themes that made America what it is today.

The Gold Rush would be ideal as an adventure, panning for gold, with many natural hazards en route that would have to be overcome.

You could explore underground mines, although that has been done before in Lost Dutchman's Gold and Fool's Gold. Nevertheless, the area is still barely touched, and a good adventure could still make use of some of the ideas presented in these games.

But for all that, the idea of robbing the town safe is probably the best, untapped, idea, that could lead to a very good adventure indeed. Good writing!

Murder Mystery



Introduction

One of the great untouched ideas in adventure game writing is the solving of a mystery, not necessarily a murder, although that is what we'll look at here, but any mystery.

It's hard to explain why this should be so.

Certainly detective novels sell in vast quantities year after year, and there would definitely be no shortage of plots for the adventure writer who would like to concentrate on creating a series of mystery adventures, perhaps with a connecting link like Agatha Christie's Hercule Poirot, or Conan Doyle's Sherlock Holmes (not forgetting Doctor Watson!), so that the games are linked together as a whole, although each one enjoys a separate identity as a full adventure game.

The sort of game that could be created would depend to some extent on the character adopted as the adventurer. 'Of all the adventure games in all the world, you had to walk into mine' players would enjoy a different game from 'it's all part of life's rich pageant' bungling French detectives, so the game itself would have to take on a character akin to that of the adventurer solving it.

The Story

As the great detective, a new case is brought to your attention. In the old manor belonging to the squire of the local village, a few village notables were sitting down to a pleasant evening meal when one of them pitched over, dead!

Obviously, the body is examined and found to contain an overdose of some poison, which narrows the number of suspects down to the people who were sitting down to the meal, plus all the servants who normally attend the house. In total, a dozen people are suspected, and you have to find out who the real villain was.

Developing the Story

In essence, this is a variation on the old Cluedo theme, the popular board game from Waddington's, in that there are a number of suspects within a confined area, and you have to eliminate everyone bar one

person: the murderer.

Exploration of the manor in search of clues could provide the basic adventure scenario, whilst the questioning of the suspects could be kept on a very simple level, in order to accomodate our two-word adventure type of game.

In a more advanced game of the Zork variety one could well indulge in elaborate question and answer routines, but here we'd have to restrict ourselves to much simpler ideas, perhaps using TAKE STATEMENT when you're in the same room as one of the suspects, or something like that. EXAMINE SMITH, or EXAMINE SQUIRE, might reveal some vital clue about their person.

Building the story up in this way could then provide the basis for an enjoyable romp, with the detective having to do an awful lot of work to uncover the truth.

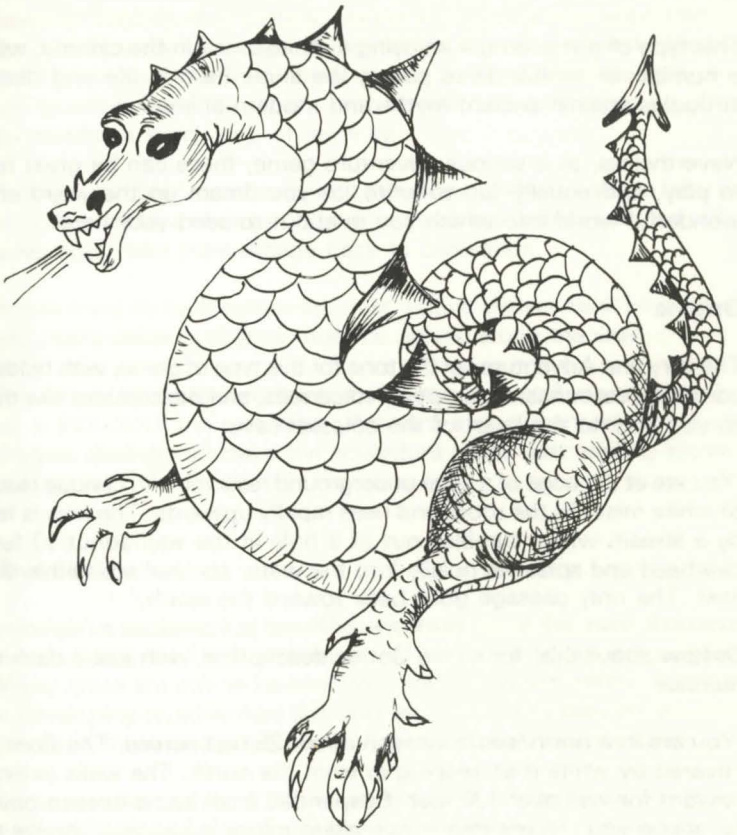
Conclusion

Detective games of this nature, that is, combining an adventure with a little bit of amateur sleuthing, have been very much neglected, and could lead to some good games if developed properly.

Not only would the exploration of the manor, or whatever environment you pitch our adventurer into, provide some entertaining diversions, by way of locked doors, guard dogs, hidden tunnels, and other hazards, but the level of brainwork required could combine to produce a good few hours entertainment.

But now, a much more traditional theme, as we enter the Valley of Death!

The Valley of Death



Introduction

The Valley of Death! You can tell from the title alone just what sort of world we're about to enter, and it is very much the traditional home of the adventure writer, with mythical beasts and dragons, hobgoblins, orcs and trolls, necromancers and black riders, and a myriad of other illustrious villains from the halls of the mountain king, or more specifically the pages of books such as *Lord of the Rings*!

This type of game is now enjoying a renaissance in the cinema, with a number of terrible films pitting the super-hero in life and death struggles against ancient myths and modern animation.

Nevertheless, as a serious adventure game, these can be great fun to play, and equally fun to write, as you dream up the weird and wonderful world into which you're about to send your hero.

Origins

The very first Adventure set the tone for this type of game, with hidden corridors, vast chasms, erupting volcanoes, and descriptions like this as you go into the heart of the colossal cave:

'You are at the edge of a large underground reservoir. An opaque cloud of white mist fills the room and rises rapidly upwards. The lake is fed by a stream which tumbles out of a hole in the wall about 10 feet overhead and splashes noisily into the water somewhere within the mist. The only passage goes back toward the south.'

Or how about this, for a true Gothic description, with just a dash of humour:

'You are in a north/south canyon about 25 feet across. The floor is covered by white mist seeping in from the north. The walls extend upward for well over 100 feet. Suspended from some unseen point far above you, an enormous two-sided mirror is hanging parallel to and midway between the canyon walls. (The mirror is obviously provided for use by the dwarves, who, as you know, are extremely vain.) A small window can be seen in either wall some 50 feet up.'

Tremendous stuff! You know straightaway the kind of world you're walking in, where characters from a Jules Verne novel like *Journey*

to the Centre of the Earth might be expected to appear at any moment.

The Story

All good, traditional stuff, but the area is so vast that many adventures are still to be written that put the adventurer into a world filled with strange creatures, and countless hazards to overcome.

The story of the valley is a simple one. Stranded (you can work out how!) at the top of the valley, you have to make your way down to the mouth, walking alongside the river as it gushes down to the sea, sinking into quicksand, building canoes that do little more than pitch you headlong into the rapids, with hostile natives stalking you from the shadows every step of the way.

Strange, terrible creatures inhabit the valley, and you have to kill them all with a mixture of dexterity, wit and courage before you can safely leave and make your escape back to civilisation.

Ropes must be built across the river, native arrows must be avoided, and many other problems must be solved along the way.

The range of story lines in this sort of field is vast, and one could conjure up a thousand and one tales of sword and sorcery, dungeon and dragon, that would leave the adventure player just waiting for your next game.

Conclusion

Here we've explored just five different areas out of the many thousands that could be used to form the basis of a good, solid, adventure game. Many areas are still to be touched, and it is worth taking your time in developing an adventure scenario, as the plot and story line are major points in the success or failure of writing an adventure game.

So too are the problems that must be solved, and the ease or difficulty with which the player can progress to other levels in the game, but none the less it is the story line that will initially attract a player, and start him playing your game rather than any other.

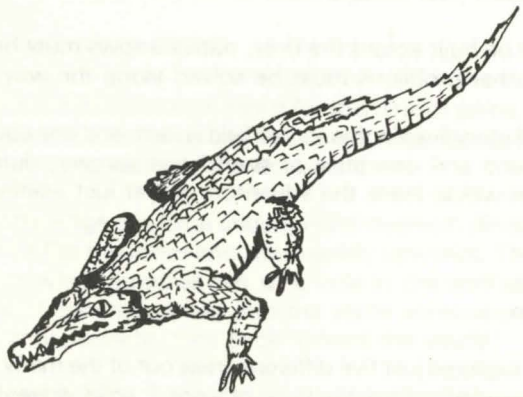
We mentioned earlier the Bible as a source of inspiration, and there are an infinite number of stories in there that could be turned into long adventure games. I'm not suggesting you wander across the desert

for forty years, but you might have fun trying to cross the Red Sea.

In the end, it is your own mind that is going to conjure up a good or a bad adventure, and the story must hold true throughout the entire game, or people will just tire of it and not consider any more of your games, not matter how good.

It is a lot easier to bore people than it is to entertain them!

So, at the risk of boring you with a lot of writing, let's take a look at the construction of Underground Adventure, and the entire selection of verbs that are used in the game.



6
Underground Adventure

In this chapter we are going to present you with the rest of Underground Adventure, to complement the listings that you've already seen in chapters 3 and 4.

All that's left to do now is to look at the data, which we'll list in full, followed by three pages of explanations for the verb data, the objects data, and the rooms data, and the entire list of verbs that are used in the game.

As you've probably never written an adventure before, we're going to go through each verb in turn, giving on one page the listing for that verb, and every part of the program that handles it, and on the opposite page an explanation for the listing, line by line.

Some verbs take up more room than others, and in particular the GET and OFFER routines are quite long. Others do not take up so much space in this adventure, and so there will be a fair amount of blank space left on a number of pages. This is there for your own notes, because in many instances the verb will require a lot more code in your own games than we've used here.

Thus the space can be used to amplify on the original listing, without having to have lots of separate sheets of paper lying around everywhere.

The Scenario

You are outside a set of caves that look invitingly out at you. They

seem worthy of exploration, and so off you go into the caves and the darkness within. Finding an old torch and some matches, you light the torch, and the blazing light fills the caves. As you step further inside the gates are rocked by the reverberating sound of a solid gate being slammed shut behind you, and your avenue of escape is blocked.

Somewhere in the caves lies the key to the gate, which you must find before you can escape. You got yourself into the caves, now only you can get yourself out.

We took a fairly detailed look at this adventure earlier on, so the description of the perils involved in finding the key can be read there, but it's worth pondering a while on the story line as we've got it set out here.

The Story Line

This game is set in traditional adventure territory, deep underground, fighting off mythical creatures and exploring some unusual terrain.

The tunnels and corridors much loved by Crowther and Woods have been incorporated here, together with a few swamps, a little touch of magic, and a hazy, misty land that is difficult to pass through. Some of the hazards will be familiar to players of other adventures, while some will be new, as will be the manner in which these puzzles have to be solved.

This mixture of old and new has been adopted a) to put the player at ease with familiar territory, and the writer with a good stock of useful verbs and subroutines that can be used in other stories, and b) to have enough new material to keep the player interested, and give the writer some ideas of how new verbs can be accomodated into his own adventures.

The Writing

This is not to say that this is the only way to write adventure games, of course it isn't. But it does produce a fairly fast response from the computer, and it does allow a large range of verbs and nouns to be accommodated quite easily.

One of its weaknesses is the length of the room descriptions: these tend to be rather short, and because of this it is sometimes difficult

to produce a different and meaningful description for each room. This problem could be surmounted by the addition of a few extra lines of code in the routine from line 8000 onwards, e.g.

```
5011 IF CP=24 THEN 8000
.
.
.
8000 PRINT "IN A LONG DARK TUNNEL THAT HAS BEEN  C
ARVED  OUT OF THE ROCKS."
8002 PRINT "THE ROCKS HAVE WEATHERED OVER THE  YEA
RS  INTO  A THOUSAND AND ONE"
8004 PRINT "FANTASTIC FORMATIONS.  THE LIGHT FROM
YOUR  TORCH FLICKERS EERILY"
8006 PRINT "AMONGST THE SHADOWS, CAUSING THE LIGHT
 TO  DANCE ABOUT FROM THE ROCKS
and so on, before returning back to the main progr
am again.
```

Other than that, it works, so let's look at the verbs.

The Complete List of Verbs

These verbs are to be covered one at a time, with two pages reserved for each verb, one for the listing and one for the explanation of that listing.

GO

This verb covers all movement in the game, in the four cardinal directions.

```
 300 IF a$<>"" AND no=0 THEN PRINT "You have me ba
ffled.": GO TO 210
 302 IF no>28 OR no<21 THEN PRINT "I don't underst
and you.": GO TO 210
 304 IF no>24 THEN LET no=no-4
 306 LET no=no-20
 308 IF no AND cp=3 AND gf=1 THEN GO TO 5200
 310 IF no AND pd THEN PRINT "You have fallen into
 a very deeppit.  This could be trouble.": GO TO 6
12
 312 IF p(cp,no)=0 THEN PRINT "Nope!  You can't go
 that way.": GO TO 210
 314 IF cp=53 AND no=2 THEN GO TO 6300
 316 LET cp=p(cp,no): GO TO 200
```


Explanation

Line 300 - if the noun string is not equal to zero, but the noun number is, then the word is not recognised, a message is printed, and back for another input.

Line 302 - if the noun number NO is greater than 28, or less than 21, then it is not one of the eight movement nouns (NORTH, SOUTH, EAST, WEST, N, S, E, W), and so the computer doesn't understand!

Line 304 - just adjust NO, if it's greater than 24, to lie between 21 and 25.

Line 306 - adjust NO to lie between 1 and 4.

Line 308 - if we're moving in room 3 and the gate is open (GF = 1), then it's the start of the game, so GOTO5200 to set the start up by shutting the gate.

Line 310 - if we're moving but it's pitch dark (PD is set), then print message and GOTO death routine.

Line 312 - if P(room number,direction) is equal to zero, then we can't go that way, so print out message and back for more input.

Line 314 - if we're in room 53, and we're trying to go south, then GOTO 6300

Line 316 - update the room number CP from the variable P, then GOTO 200

GET

This verb handles the picking up of all objects in the game.

```
350 IF no=0 THEN GO TO 1900
352 GO SUB 5300
354 IF o(no)=-1 THEN PRINT "You've already got it
!": GO TO 210
356 IF o(no)<>cp THEN PRINT "I can't see it here.
": GO TO 210
358 IF no=18 AND o(19)<>-1 THEN PRINT "You haven'
t got a container.": GO TO 210
360 IF no=39 AND o(19)<>-1 THEN PRINT "You haven'
t got a container.": GO TO 210
362 IF no=39 AND o(19)=-1 THEN LET o(39)=0: LET n
o=52: LET o(19)=0: LET zz=zz-1: GO TO 380
364 IF no=18 AND o(19)=-1 THEN LET o(18)=0: LET n
o=51: LET o(19)=0: LET zz=zz-1: GO TO 380
366 IF no=1 OR no=3 OR no=6 OR no=9 OR no=11 OR n
o=17 THEN PRINT "Don't be ridiculous!": GO TO 210
368 IF no=20 OR no=29 OR no=30 OR no=31 OR no=32
OR no=35 OR no=36 THEN PRINT "I can't do that!": G
O TO 210
370 IF no=40 OR no=41 OR no=43 OR no=49 THEN PRIN
T "It can't be done!": GO TO 210
372 IF no=8 OR no=50 THEN PRINT "There's no point
carrying that, so I won't!": GO TO 210
374 IF zz>4 THEN PRINT "You re carrving too much.
": GO TO 210
376 IF no=12 AND cp=10 THEN LET p(10,4)=0: LET p$
(10)="faced by a vast chasm.":
378 IF no=15 AND sc=0 THEN PRINT "You can't get i
t yet.": GO TO 210
380 LET zz=zz+1: LET o(no)=-1: PRINT "OK.": GO TO
210
```


Explanation

Line 350 - if the noun number is zero, then we don't know what the noun is, so GOTO 1900 to print out message.

Line 352 - GOSUB to routine to check bottles and torches.

Line 354 - if the object number is set to minus 1, we're already carrying it!

Line 356 - if the object number isn't equal to the room number, then it isn't here, so print message and try again.

Line 358 - if you're trying to get object 18, the pool of oil, but you're not carrying the empty bottle, object number 19, then you can't have it!

Line 360 - ditto for trying to get object 39, the whisky.

Line 362 - on the other hand, if you want the whisky and you are carrying the bottle, then you can have it. The pool of whisky disappears, change the noun number to refer to the bottle of whisky, object 52, set the empty bottle to disappear, and decrement the number of objects being carried counter ZZ before GOTO 380

Line 364 - ditto for the pool of oil

Line 366 - list of objects (see data tables later) that you can't carry: mainly big things that would be too heavy, so if you're trying to get one of them, print out a suitable message and go back for another input.

Line 368 - more unobtainable objects.

Line 370 - and yet more, including object 49, the word PROGRAM: someone would type it in!

Line 372 - silly objects that people might try to pick up, a pile of rubble and some broken glass.

Line 374 - check to see how much is already being carried.

Line 376 - if you pick the plank up from room 10, then you can't get past the chasm again, so adjust everything accordingly.

Line 378 - if you're trying to get the shimmering curtain, but you haven't

worked out how to get past it (in which case the shimmering curtain counter isn't set), then you can't have it!

Line 380 - everything's OK, increase the number of objects being carried counter, put the object in your possession, and GOTO line 210

INVENTORY

This verb is used to give a list of everything that you're carrying, so you can take stock of a given situation, and decide what to leave behind.

```
 450 PRINT "You are carrying the following:": LET
gs=0: LET zz=0
 452 FOR i=1 TO lo: IF o(i)=-1 THEN PRINT o$(i): L
ET gs=gs+1: LET zz=zz+1
 454 NEXT i
 456 IF gs=0 THEN PRINT "Not a lot!"
 458 GO TO 210
```


Explanation

Line 450 - print out a simple message, and set the variable GS to zero, and also the number of objects being carried to zero.

Line 452 - start a loop up that will be gone through LO (number of objects) times, and check to see if the object is being carried i.e. if O(I) is equal to minus one. If it is, then print up on the screen the objects description from the variable O$, and increment the two counters GS and ZZ.

Line 454 - NEXT step through the loop.

Line 456 - if GS is equal to zero, then you can't be carrying anything, so just print out the word NOTHING.

Line 458 - go back and get another input.

DROP

This verb is used to drop anything that you might be carrying.

```
 550 IF no=0 THEN GO TO 1900
 552 GO SUB 5300
 554 IF o(no)<>-1 THEN PRINT "You aren't even carr
ying it!": GO TO 210
 556 IF no=19 THEN PRINT "Smash...": LET o(19)=0:
LET o(50)=cp: LET zz=zz-1: GO TO 210
 558 IF no=51 THEN PRINT "Smash...": LET o(19)=0:
LET o(51)=cp: LET zz=zz-1: GO TO 210
 560 IF no=52 THEN PRINT "Smash...": LET o(19)=0:
LET o(52)=cp: LET zz=zz-1: GO TO 210
 562 IF no=16 THEN PRINT "Oh dear!  It vanishes in
 a      sparkle of shattered glass!"
 564 IF no=16 THEN LET o(16)=0: LET zz=zz-1: GO TO
 210
 566 IF no=46 THEN LET o(no)=0: LET o(45)=cp: LET
zz=zz-1: PRINT "OK.": GO TO 210
 568 IF no<>12 THEN GO TO 576
 570 IF cp<>10 THEN GO TO 576
 572 PRINT "Brilliant!  Now you can walk    across
 the plank!"
 574 LET o(12)=cp: LET p(10,4)=14: LET p$(10)="wal
king across the plank.": LET zz=zz-1: GO TO 210
 576 PRINT "OK.": LET zz=zz-1: LET o(no)=cp: IF tb
=1 THEN GO TO 580
 578 GO TO 210
 580 PRINT "The bear glares at you and runs awav i
n a fit of pique.": LET tb=0: LET o(9)=INT (RND*40
)+1
 582 LET zz=zz-1: GO TO 210
```


Explanation

Line 550 - noun not recognised, so GOTO 1900

Line 552 - GOSUB 5300 to check the bottle and torch situation.

Line 554 - if you're not carrying it then you can't drop it!

Line 556 - if you drop the bottle, then it breaks, the empty bottle disappears, a pile of broken glass appears in the room CP, the object counter is decreased, and it's back for another input.

Line 558 - ditto for the bottle of oil.

Line 560 - and for the bottle of whisky.

Line 562 - if you attempt to drop object number 16, the mirror, it vanishes, so print out a suitable message.

Line 564 - remove mirror, and decrement object counter, then GOTO 210 for more input.

Line 566 - dropping the blazing torch causes the blazing torch to disappear, the old torch to appear in the room, the object counter to decrease, the word OK to be printed on the screen, and GOTO210

Line 568 - if you're not trying to drop object 12, the plank, then off to line 576.

Line 570 - if you're not trying to drop the plank in room 10, then off to line 576 as well.

Line 572 - print out a message of congratulations at doing something right.

Line 574 - put the plank in the room, enable you to go west from room 10, change the description for room 10, decrement the object counter, and GOTO 210

Line 576 - everything's all right, we can drop somehing! Print OK, decrement the object counter, put the object in the room, and check to see if we've got the bear with us. If we have then GOTO 580

Line 578 - back to 210 for more input.

Line 580 - the bear thinks you're throwing something at it, so runs away! Set the bear flag to zero, put the bear (object 9), in a room somewhere between rooms 1 and 41.

Line 582 - decrement the object counter, and back to 210 again.

QUIT

This is the verb used to end a game, and has to ask you a couple of questions before you can actually leave the game.

It's used to give you the chance of saving your progress onto tape, should you choose to do so.

```
650 GO TO 1890

1890 PRINT "OK.": PRINT "Do you want to save your
      progress onto tape (Y or N)?"
1892 IF INKEY$="" THEN GO TO 1892
1893 IF INKEY$="y" THEN GO TO 3000
1894 IF INKEY$="n" THEN GO TO 614
1896 GO TO 1892
```


Explanation

Line 650 - GOTO 1890.

Line 1890 - print out a message, and give the player the chance of saving the game onto tape, for starting again at the next session without having to go through the whole rigmarole of playing the game again!

Line 1892 - wait for a key to be pressed on the keyboard.

Line 1893 - if key is a Y then GOTO save routine at 3000.

Line 1894 - if they've pressed 'N' then GOTO 614, which gives you the chance of running through the game again before definitely finishing.

Line 1896 - they haven't pressed anything, so go back and wait until they have!

CROSS

This verb is used whenever the player wants to get across something, and can't be bothered to type in a direction.

In this game the verb doesn't really have any use, but in other adventures it could be a very useful way of getting from one place to another, which just by a logical NORTH or whatever they couldn't do.

In that case, you'd have to check the room number CP, and provided that there's something in place that they can cross, whisk them across to the other side by changing CP to the appropriate value.

```
 700 IF no=0 THEN PRINT "Cross what?!": GO TO 210
 702 IF cp<>15 AND cp<>10 THEN PRINT "There's noth
ing here to cross.": GO TO 210
 704 IF no<>1 AND no<>6 AND no<>12 THEN PRINT "Mmm
mm.  What a strange idea.": GO TO 210
 706 PRINT "Why don't you just type in a    direct
ion instead?": GO TO 210
```


Explanation

Line 700 - if you don't understand what noun has been typed in, ask them what it is they want to cross, and go back to line 210 again.

Line 702 - if they're not in rooms 15 or 10, the only two rooms that have got chasms in them, then there's nothing to cross, so tell the player so and go back to 210 again.

Line 704 - if they're not trying to cross the chasm, the bridge or the plank, then assume they're trying some strange breeding program, print out a suitable response on the screen, and go back to line 210 again.

Line 706 - a cop-out, since we don't really use this verb in the program, so suggest that they type in a direction instead, and then back to 210.

OPEN

There are a number of things that can be OPENed in this game, or at least that the player can try and open, like gates and doors, so this verb deals with all of that.

If you had treasure chests or something in your games, the relevant lines to handle opening of the chest could be used here.

```
 800 IF no=0 THEN PRINT "Open what?!": GO TO 210
 802 IF cp<>60 AND cp<>3 THEN PRINT "There's nothi
ng here to open.": GO TO 210
 804 IF cp=60 THEN GO TO 812
 806 IF gf=1 THEN PRINT "But it is open!": GO TO 2
10
 808 IF o(42)<>-1 THEN PRINT "But you haven't got
the key.": GO TO 210
 810 PRINT "The gate swings slowly open.": LET gf=
1: LET p(3,1)=2: GO TO 2510
 812 IF df=1 THEN PRINT "But it's already open.":
GO TO 210
 814 IF o(33)<>-1 THEN PRINT "You've nothinf stron
g enough to open it with.": GO TO 210
 816 PRINT "You'll just have to try and do  this s
ome other way.": GO TO 210
```


Explanation

Line 800 - if you don't recognise the noun, ask them what they're trying to open, and GOTO210

Line 802 - if they're not in rooms 60 or 3 then there's nothing to open, so tell them so, and GOTO210

Line 804 - if they're in room 60 then GOTO812

Line 806 - if the gate flag is set then the gate is already open, so tell them, and GOTO210

Line 808 - if they're not carrying object 42, the key, then they can't open it anyway, so tell them again, and GOTO210

Line 810 - they can open the gate, so print a suitable message. This signifies the end of the game, so set the gate flag, let them go north from room 3 to room 2, and GOTO2510 to print a congratulatory message and end the game.

Line 812 - if the door flag is set then it's already open, so tell them so and GOTO210

Line 814 - if they're not carrying the lump of mortar they've nothing strong enough to open the door with, so tell them so by printing a message on the screen, and GOTO 210

Line 816 - you don't open a door with a lump of mortar, you have to do something else, so tell them so, and GOTO 210

CLOSE

This is used whenever the player attempts to close something in the game. In Underground Adventure the only things that they can close are the gate or the door, so we check for that accordingly.

```
 850 IF no=0 THEN PRINT "Close what?!": GO TO 210
 852 IF no<>32 AND no<>35 THEN PRINT "Que?": GO TO
 210
 854 IF cp=3 THEN GO TO 860
 856 IF df=0 THEN PRINT "It's already closed.": GO
 TO 210
 858 PRINT "OK.": LET p(60,2)=0: LET df=0: LET p$(
60)="faced with a closed door.": GO TO 210
 860 IF gf=0 THEN PRINT "But it's already closed."
: GO TO 210
 862 PRINT "The gate is a magical one and,  once o
pened, it can never be    closed by mere humans li
ke you.": GO TO 210
```


Explanation

Line 850 - if you don't recognise the noun, ask them what it is they want to close, and GOTO210

Line 852 - if they're not trying to close the old door or the gate, then tell them that you don't understand the request, and GOTO 210

Line 854 - if they're in room 3 then GOTO 860

Line 856 - if the door flag is set to zero then the door is already closed, so tell them so and GOTO 210

Line 858 - print OK, close off the south exit from room 60, set DF equal to zero, change the message for room 60, and GOTO 210

Line 860 - if the gate flag is set to zero then it's already closed, so tell them so and GOTO 210

Line 862 - just print out a simple message telling them that the gate is a magical one, and cannot be closed by you! Then off to 210 again for another input.

EAT

Most adventure games seem to feature food of one sort or another, and although this food is very rarely intended for the consumption of the player, it is inevitable that sooner or later someone is going to attempt to eat it themselves.

Hence this routine, which copes with greedy adventure players!

```
 900 IF no=0 THEN GO TO 1900
 902 GO SUB 5300
 904 IF o(no)<>-1 AND o(no)<>cp THEN PRINT "I can'
t see it here.": GO TO 210
 906 IF no<>10 THEN PRINT "I don't think so someho
w.": GO TO 210
 908 PRINT "Mmmm-mmmm!  That was delicious.": LET
o(10)=0: LET zz=zz-1: GO TO 210
```


Explanation

Line 900 - if you don't recognise the noun then GOTO the routine at 1900 to print out a suitable message.

Line 902 - check to see if anyone's attempting to eat the bottle or the torch. They do, they do!!

Line 904 - if the object that they want to eat isn't in their possession, and it isn't in the room either, then they can't have it, so print out a suitable message and GOTO 210

Line 906 - if they're attempting to eat anything other than object number 10, a bun, then warn them off with a suitable message.

Line 908 - fair enough, the delicious bun is eaten, with an appropriate message, the bun then disappears (inside the player's stomach), the object counter is decremented, and we go off to 210 for another input.

FEED

Since there is some food about, someone has obviously got to feed it to something, and you'd be surprised at the things some adventure players try to force on the unsuspecting occupants of the adventure world.

In Underground Adventure, the only thing that's interested in eating is the bear, and the only thing it wants to eat is the bun, apart from you, perhaps.

```
 950 IF no=0 THEN PRINT "I don't understand you.":
 GO TO 210
 952 IF no<>9 THEN PRINT "It isn't hungry!": GO TO
 210
 954 IF o(10)<>-1 THEN PRINT "You've nothing to fe
ed it with.": GO TO 210
 956 GO TO 1072
```


Explanation

Line 950 - if you don't recognise the noun, then tell the player so, and go off back to line 210 again for another try.

Line 952 - if you're not carrying the bun, object 10, then whatever you're trying to feed fobs you off with an excuse, and GOTO210

Line 954 - if you're trying to feed anything other than the bear, then it suddenly feigns a lack of hunger, so we print a suitable message on the screen and go back to 210 again.

Line 956 - Aha! We're trying to feed the bear, so we go to line 1072, where this same sequence of events is handled by another verb, OFFER, in case someone decides to OFFER BUN, rather than feeding the bear.

DRINK

An occupation favoured by many adventure players, but when it comes to actually playing a game of adventure people will try to drink some very odd things indeed.

Like eating in adventures, the drink is usually reserved for someone else's use rather than that of the player, and consumption by the player will, in the end, result in an adventure that can't be finished.

Still, they don't know this when they start, and so the appropriate routine has to be inserted to handle this.

```
1000 IF no=0 THEN GO TO 1900
1002 GO SUB 5300
1004 IF no<>51 AND no<>52 THEN PRINT "You must be
joking!": GO TO 210
1006 IF no=51 THEN PRINT "Urggh!": LET o(51)=0: LE
T o(19)=-1: GO TO 210
1008 PRINT "Glug-glug-glug .... hic!": LET o(52)=0
: LET o(19)=-1: GO TO 210
```


Explanation

Line 1000 - if you don't understand the noun, then it's off to the subroutine at line 1900

Line 1002 - GOSUB 5300 to check we're not mixing up bottles and torches.

Line 1004 - if they're not trying to drink either of the two liquids in the program, i.e. the oil or the whisky, then print out a suitable statement and GOTO 210 as usual.

Line 1006 - some people try to drink strange things in these games, and oil is one of them. However, if that is what they want to drink they must face the consequences, so print a 'this is revolting' message on the screen, remove the bottle of oil from the player's possession, replace it with an empty bottle, and GOTO 210.

Line 1008 - if you will drink whisky! Print out the message, remove the bottle of whisky, replace it with an empty bottle, and GOTO 210.

OFFER

This is one of the commonest ways of transferring possessions from the player to someone else, and in this adventure there are two things that change hands, and get you through a couple of awkward spots.

```
1050 IF no=0 THEN PRINT "Offer what?!": GO TO 210
1052 GO SUB 5300
1054 IF o(no)<>-1 THEN PRINT "You've got to have i
t to offer  it!": GO TO 210
1056 IF no=10 THEN GO TO 1070
1058 IF no<>52 THEN PRINT "You've nothing worth of
fering.": GO TO 210
1060 IF cp<>50 THEN PRINT "There's no-one here who
 wants it(except perhaps for you)": GO TO 210
1062 PRINT "The denizen of the caverns downsit in
one draught, and          gratefully shows you a n
ew      tunnel."
1064 PRINT : PRINT "He then stumbles away to sleep
  it all off!"
1066 LET o(52)=0: LET o(19)=-1: LET p(50,4)=55: LE
T p$(50)="walking past old spirits."
1067 LET o(29)=0: GO TO 210
1070 IF cp<>27 THEN PRINT "There's nothing here th
at wants it!": GO TO 210
1072 PRINT "The bear gratefully accepts the bun, a
nd stands aside to reveal a new path way."
1074 PRINT : PRINT "In a show of gratitude, he now
  attatches himself to you like a limpet!": LET p(
27,1)=28
1075 LET p$(27)="walking past the scent ofold bear
."
1076 LET o(10)=0: LET zz=zz-1: LET tb=1: GO TO 210
```


Explanation

Line 1050 - if you don't recognise the noun, then ask them what they want to offer, and GOTO 210

Line 1052 - check that we're not confusing the various bottles and torches by the subroutine at 5300

Line 1054 - if they're not carrying the object you can't offer it, so print out the message and GOTO 210

Line 1056 - if they're talking about the bun then GOTO1070

Line 1058 - if they're not carrying the bottle of whisky, then forget it! Print out message and GOTO 210

Line 1060 - if they're not in room 50 then there's no one who's interested in the whisky, except them, so print out the message and GOTO 210

Line 1062 - aha! The denizen of the caves gratefully accepts their kind present, so print out a suitable message.

Line 1064 - rest of message.

Line 1066 - remove the bottle of whisky, and replace with an empty one. Allow them to go west from room 50 to room 55. Change the room description for room 50.

Line 1067 - remove the denizen of the caves, and GOTO 210

Line 1070 - if they're not in room 27, then no one's interested. GOTO 210

Line 1072 - the bear eats the bun! Print out message.

Line 1074 - print rest of message and allow them to go north from room 27 to room 28.

Line 1075 - change the description for room 27.

Line 1076 - remove the bun, decrement the object counter, set the bear flag, so that he tags along behind the player, and GOTO 210

WAVE

One of the key features of most adventures is waving something, which can quite often cause a magical feat, and usually this happens relatively early on in a game.

This early success seems to go to some people's heads, who then merrily wave anything they can get their hands on, so we have to check for all of that.

```
1100 IF no=0 THEN GO TO 1900
1102 GO SUB 5300
1104 IF o(no)<>-1 AND o(no)<>cp THEN PRINT "But it
 ain't here!": GO TO 210
1106 IF no<>2 THEN PRINT "Wave,wave,wave, but noth
ing     happens.": GO TO 210
1108 IF cp<>15 THEN PRINT "Nothing happens.": GO T
O 210
1110 IF br=1 THEN PRINT "You've already done that.
": GO TO 210
1112 PRINT "A crystal bridge now spans the  chasm
(lucky, aren't you)": LET o(6)=cp: LET p(15,2)=17
1114 LET p(15,3)=16: LET p$(15)="walking across th
e chasm.": LET br=1: GO TO 210
```


Explanation

Line 1100 - if you don't recognise the noun, then GOTO 1900.

Line 1102 - our usual trip to the subroutine at line 5300.

Line 1104 - if the object is not in the player's possession, nor is it in the room, then print out a suitable message and GOTO 210

Line 1106 - if they're not waving object number 2, the staff, then print out silly message and GOTO 210

Line 1108 - even if they're waving the staff, nothing will happen unless they're also in room 15, so print out the message and GOTO 210 as usual.

Line 1110 - if the bridge flag is set, then tell them that they've already stood here and waved a staff, before going to 210 again.

Line 1112 - print the magic message, put the bridge in the room, allow them to go south from room 15 to room 17, and set the bridge flag.

Line 1114 - allow the player to go east from room 15 to room 16, change the description for room 15, and finally GOTO 210

CUT and CHOP

In this adventure the two words are synonymous, in that both achieve the same object in the same way.

However, some games may care to give them a different meaning, so we've left them both in here.

Usually used to cut something up or chop it down, like a tree, or a tangled mass of vines, or something of that ilk.

```
1150 GO TO 1200:
1200 IF no=0 THEN GO TO 1900
1202 GO SUB 5300
1204 IF o(no)<>-1 AND o(no)<>cp THEN PRINT "I can'
t see that here.": GO TO 210
1205 IF no<>3 AND no<>15 AND no<>5 AND no<>12 AND
no<>32 THEN PRINT "Not a chance!": GO TO 210
1206 IF o(4)<>-1 THEN PRINT "You've got nothing to
 cut it    with.": GO TO 210
1208 IF no<>3 AND no<>12 THEN PRINT "Your axw is n
ot strong enough.": GO TO 210
1210 IF no=3 THEN GO TO 1220
1212 PRINT "The plank is now nicely hewn,  but yo
u'll need something else  before you can make a la
dder."
1214 LET o(12)=0: LET o(53)=-1: GO TO 210
1220 PRINT "Timberrrr!  The tree crashes to the gr
ound."
1222 LET p(21,3)=22: LET p$(21)="walking past a de
ad tree.": LET o$(3)="an ex-tree": GO TO 210
```


Explanation

Line 1150 - speaks for itself!

Line 1200 - if you don't recognise the noun, off to 1900

Line 1202 - as usual, check with the subroutine at 5300 before proceeding further.

Line 1204 - if the object they're trying to cut or chop isn't in the player's possession, and it isn't in the room, then print out a suitable message and GOTO 210

Line 1205 - if they're not trying to chop the tree, the shimmering curtain, the rope, the plank, or the old door, then tell them that it can't be done, and GOTO 210

Line 1206 - if the player is not carrying object number 4, the axe, then they've nothing to chop anything with, so tell them so and GOTO 210 again.

Line 1208 - unless they're trying to cut the tree or the plank, then the axe isn't strong enough and they'll have to try a different tack, so tell them so, and GOTO 210

Line 1210 - if it's the tree they're after, then line 1220 handles it.

Line 1212 - print a message about the plank.

Line 1214 - rest of message, remove the plank, put the neatly sawn timber in their possession (a fine piece of axemanship!), and GOTO 210

Line 1220 - print message about the tree.

Line 1222 - let them go east from room 21 to room 22, change the description for room 21, change the description of object 3, and GOTO 210

As it stands, this will let players repeatedly chop down the ex-tree, should they choose to do so, but a simple test could be carried out to disable this.

CLIMB

In most adventures there is a degree of climbing somewhere along the way, but the ability to climb something usually depends on the player having already collected or made something else.

Such is the case with Underground, where we need to a) find a rope, and b) build a ladder before we can climb the two obstacles presented to us.

```
1250 IF no<>3 AND no<>5 AND no<>13 THEN PRINT "I b
eg your pardon?": GO TO 210
1252 IF no=3 THEN PRINT "Oh, these old war wounds!
  Sorrybut it can't be done.": GO TO 210
1254 IF no=5 THEN GO TO 1266
1256 IF o(13)<>cp THEN PRINT "I can't see it anywh
ere around  here.": GO TO 210
1257 IF cp<>45 AND cp<>47 THEN PRINT "There's no p
oint in climbing theladder here.": GO TO 210
1258 IF cp=45 THEN LET o(13)=47: LET cp=47: GO TO
200
1260 LET o(13)=45: LET cp=45: GO TO 200
1266 IF o(5)<>cp THEN PRINT "I don't see it on the
 ground    anywhere.": GO TO 210
1267 IF cp<>35 AND cp<>36 THEN PRINT "There's no p
oint climbing the   rope here.": GO TO 210
1268 IF cp=35 THEN LET o(5)=36: LET cp=36: GO TO 2
00
1270 LET o(5)=35: LET cp=35: GO TO 200
```


Explanation

Line 1250 - if the object concerned isn't the tree, the rope or the ladder, then the player can't climb it, so a suitable response is given before going back to 210

Line 1252 - if the player is attempting to climb object 3, the tree, an excuse is given as to why he can't, and back to 210

Line 1254 - if the player is trying to climb object 5, the rope, then off to 1266

Line 1256 - if the ladder isn't in the room then the player can't climb it, so print a message and GOTO 210

Line 1257 - if the player isn't himself in rooms 45 or 47 then there's no point in climbing the ladder, so print a message and GOTO 210

Line 1258 - if the player's in room 45, then put him in room 47, put the ladder in room 47, then GOTO 200 for a LOOK.

Line 1260 - otherwise, put the ladder and the player in room 45 again, and GOTO 200

Line 1266 -if the rope isn't in the room, then the player can't climb it, so print a message and GOTO 210

Line 1267 - if the player himself isn't in rooms 35 or 36 then there's no point in climbing the rope, so print a message out and GOTO 210

Line 1268 - if the player's in room 35 then put him and the rope in room 36, print a message and GOTO line 200

Line 1270 - otherwise, put the player and the rope in room 35, and GOTO 200

LIGHT

Torches are quite a common feature of adventures, and obviously they'll have to be lit at some time or other during the course of the game.

Occasionally other objects will have to be lit as well, as in the case of Underground where the dynamite has to be used, and checks must be made to see what the player is trying to light, and if he's got the necessary equipment to light something with : usually matches.

```
1300 IF no=0 THEN GO TO 1900
1302 GO SUB 5300
1304 IF o(no)<>cp AND o(no)<>-1 THEN PRINT "I can
t see it here.": GO TO 210
1306 IF o(44)<>-1 THEN PRINT "You've nothing to li
ght it with.": GO TO 210
1308 IF no<>45 AND no<>7 THEN PRINT "Don't be sill
y.": GO TO 210
1310 IF no=7 THEN GO TO 1320
1312 IF o(46)=-1 THEN PRINT "It's already lit!": G
O TO 210
1314 PRINT "OK.": LET o(46)=-1: LET o(45)=0: LET p
d=0: GO TO 210
1320 IF o(7)=-1 THEN PRINT "Booooom!  You've just
blown     yourself to bits!": GO TO 612
1322 IF cp<>4 THEN PRINT "Kabooooom.  The dust slow
ly      clears to reveal ...  nothing's changed at
 all.": LET o(7)=0
1324 PRINT "Kabooooom!  The wall's been blownto smi
thereens!"
1326 LET o(7)=0: LET zz=zz-1: LET p(4,4)=5: LET p$
(4)="walking along a dusty    track.": GO TO 210
```


Explanation

Line 1300 - unrecognised noun, so GOTO 1900

Line 1302 - the usual check using the subroutine at 5300

Line 1304 - check to see if the object they're trying to light is either being held or in the room, and if not print a message and GOTO 210

Line 1306 - if the player isn't holding object 44, i.e. the matches, then he can't light anything, so print message and GOTO 210

Line 1308 - if they're not trying to light the torch or the dynamite, then it can't be lit, so print message and GOTO 210

Line 1310 - if the object they're trying to light is the dynamite then GOTO 1320

Line 1312 - if they're carrying the blazing torch, object 46, then there's no point lighting the torch, so say so and GOTO 210

Line 1314 - OK, the player can light the torch, the old torch disappears, the blazing torch is placed in the player's possession, the darkness flag PD is set to zero, and we can GOTO 210

Line 1320 - if the player is holding the dynamite while trying to light it, this is understandably fatal, so GOTO 612 for the death routine.

Line 1322 - if the player isn't in room 4, then the dynamite blows up but nothing else happens, so make the dynamite disappear.

Line 1324 - print a suitable message.

Line 1326 - remove the dynamite, decrement the object counter, enable the player to go west from room 4 to room 5, change the description of room 4, and GOTO 210

ATTACK

Adventure players seem to be a bloodthirsty lot when they get a keyboard in front of them, and quite often like to attack things.

Usually it doesn't do any good, although here we've let them off with a mild warning. However, the routine could easily be adapted to include things like killing the player if he attempts to attack a dragon, or something of that ilk.

```
1350 IF no=0 THEN PRINT "Attack what?": GO TO 210
1351 GO SUB 5300: IF o(no)<>cp AND o(no)<>-1 THEN
PRINT "Where is it?!": GO TO 210
1352 IF no<>9 AND no<>11 AND no<>29 AND no<>30 AND
 no<>31 THEN PRINT "What an odd request.": GO TO 2
10
1354 PRINT "This is not one of your better  sugges
tions!": GO TO 210
```


Explanation

Line 1350 - the noun isn't recognised, so go to the routine at line 1900 and print an appropriate message.

Line 1351 - GOSUB the subroutine at 5300 to clear up any doubts about bottles or torches, and then check that the thing to be attacked is either in the player's possession or in the room. If it isn't, print a suitable message and GOTO line 210

Line 1352 - if the player isn't trying to attack the bear, the panther, the denizen of the caves, the giant spider or the giant fly, then he's probably trying to attack an inanimate object, so print out the message and go back to 210 for another input.

Line 1354 - warn the player gently that this is not a very good idea, and go back to 210 and try something else.

KILL

Another verb that people try to use quite a lot, although in Underground Adventure we haven't used it at all.

Nevertheless, in case people do try to type in the command to KILL BEAR, or whatever, a few lines of code are necessary in order to deal with the request.

This is certainly one of the routines that could be expanded in your own games. For instance, attempting to kill the dragon in the original Adventure game produces the following series of responses : =

kill dragon

WITH WHAT? YOUR BARE HANDS?

yes

CONGRATULATIONS! YOU HAVE JUST KILLED A 20 TON DRAGON! HARD TO BELIEVE, ISN'T IT?

```
1400 IF no=0 THEN PRINT "Kill what?": GO TO 210
1401 GO SUB 5300: IF o(no)<>cp AND o(no)<>-1 THEN
PRINT "Where is it?!": GO TO 210
1402 GO TO 1352
```


Explanation

Line 1400 - if it's an unknown noun, request the player to type in what it is he really wants to try to kill, then go back to line 210

Line 1401 - GOSUB the routine at 5300 to clear up the problem of the bottles and the torches, and then check to see whether the object of the player's affections is either in the room or in his possession. If it isn't, then ask him where it is and go back to 210 again.

Line 1402 - GOTO line 1352 and deal with everything from there.

Killing with a straightforward command like this isn't allowed in this game!

HIT

Another common verb, this one is usually used when people are getting fed up with not being able to get anywhere, and are typing in commands almost at random in a vain hope of getting something to happen.

In one of the games printed in this book, hitting a wall does produce some response other than getting a sore hand. However, it isn't this one, so all this routine consists of in Underground Adventure is :=

```
1450 IF no=0 THEN PRINT "Hit what?": GO TO 210
1452 GO SUB 5300: IF o(no)<>cp AND o(no)<>-1 THEN
PRINT "Where is it?!": GO TO 210
1454 PRINT "You feel a slight twinge of painbut ot
herwise nothing happens.": GO TO 210
```


Explanation

Line 1450 - if it's an unknown noun, request the player to type in what it is he really wants to try to hit, then go back to line 210

Line 1452 - GOSUB the routine at 5300 to clear up the problem of the bottles and the torches, and then check to see whether the object of the player's affections is either in the room or in his possession. If it isn't, then ask him where it is and go back to 210 again.

Line 1454 - just print a message and go back to 210 again.

Hitting anything in Underground Adventure gets you nowhere, other than hurting your hand!

MAKE

Most adventures require you to do a lot more than just trundle around, solve a few problems and find a few treasures. In order to complete the adventure, you'll usually have to make something along the way in order to get from one location to another.

In Pirate Adventure, for instance, you have to build a boat, and in one of the other games listed in this book you have to make your own dynamite, since it isn't provided for you.

In this one, you have to make a ladder, and the materials to do so are fairly obvious: an axe to chop the wood with, some nails to hold it all together, and of course the wood itself.

```
1500 IF no=0 THEN PRINT "Make what?": GO TO 210
1502 IF no<>13 THEN PRINT "I despair of you someti
mes.": GO TO 210
1504 IF o(53)<>-1 OR o(14)<>-1 OR o(4)<>-1 THEN PR
INT "You need more materials.": GO TO 210
1506 PRINT "You have a brand new ladder.": LET o(1
3)=-1: LET o(14)=0: LET o(53)=0: LET zz=zz-1
1508 GO TO 210
```


Explanation

Line 1500 - if it's an unrecognised noun then print up a simple statement to that effect and go to line 210

Line 1502 - if the player isn't trying to make a ladder, admit that you've lost faith in his ability as an adventurer and GOTO line 210 again.

Line 1504 - check to see if the player is holding the nicely sawn timber, the nails and the axe, and if he isn't inform him that he needs to collect something else yet before he can make a ladder, and then GOTO 210 again.

Line 1506 - brilliant! you make a ladder, so print out the right message, put the ladder in the player's possession, remove the nails and the timber, and decrement the object counter by one, since we've swopped some nails and some timber for a ladder (two objects for one).

Line 1508 - back to 210 again.

REFLECT

This is a verb that I haven't seen in any other adventure, and is used to solve a problem peculiar to this one.

It is illustrated here as an example of how easy it is to add new commands to the player's vocabulary, but like all commands there must be some clue as to the actual word involved. Most players wouldn't try to REFLECT an axe, for example, but give them a mirror and it is a word that they might well try to use.

Having used it once, they will then try to reflect everything under and probably including the sun, so have a few suitable responses ready.

```
1550 IF no=0 THEN PRINT "Reflect what?": GO TO 210
1552 IF no<>47 THEN PRINT "I don't compute this in
struction": GO TO 210
1554 IF cp<>93 THEN PRINT "Nothing happens.": GO T
O 210
1556 IF sc=1 THEN PRINT "You've already done that.
": GO TO 210
1558 PRINT : PRINT "The light is reflected back an
d the shimmering curtain falls    aside."
1560 LET p(93,1)=95: LET o(15)=cp: LET p$(93)="wal
king past a shimmeringlight.": LET sc=1: GO TO 210
```


Explanation

Line 1550 - an unrecognised noun, so print an appropriate message and GOTO 210 to try again.

Line 1552 - if the object that the player is trying to reflect isn't the LIGHT, then print an 'I don't understand' message and try again.

Line 1554 - if the player isn't in room 93, where the shimmering curtain is, nothing happens, so print the message and back to 210

Line 1556 - if the SC counter has been set, then print a message to the effect that the player is repeating himself, and go back and try again.

Line 1558 - print the all important message.

Line 1560 - allow the player to go north from room 93 to room 95, put the shimmering curtain in room 93 (CP = 93 of course, since we're in that room), change the room description for room 93, set the SC counter, and GOTO 210

Astute readers will realise that we should also have a line like :

1551 GO SUB5300 : IFOB%(16)< >-1THEN?"YOU AREN'T HOLDING THE MIRROR":GO TO210

To check that the mirror is in the player's possession!

OIL

Oil frequently occurs in adventure games, and is usually used to remove something that is being sticky and refusing to budge.

Obviously, players will attempt to oil everything, so suitable responses must be made. If a player makes a mistake and oils the wrong thing, then some kind of message must be printed up, and the oil must slowly trickle away, never to be found again, thus rendering the adventure unsolvable through the fault of the player.

```
1600 IF no=0 THEN PRINT "Oil what?": GO TO 210
1602 IF o(51)<>-1 THEN PRINT "But you've no oil.":
 GO TO 210
1604 IF cp<>79 THEN PRINT "Nothing worth oiling ar
ound here": GO TO 210
1605 IF no<>17 THEN PRINT "You've just wasted a lo
t of oil.": LET o(51)=0: LET o(19)=-1: GO TO 210
1606 PRINT "The track slides noiselessly    away,
to reveal more tunnels!"
1608 LET o(51)=0: LET o(19)=-1: LET p(79,3)=80: LE
T p(79,4)=81
1610 LET p$(79)="walking past a smooth    track.":
 LET o(17)=0: GO TO 210
```


Explanation

Line 1600 - usual check for an unrecognised noun.

Line 1602 - if the player isn't carrying the bottle of oil, then he can't oil anything, so print the message up and GOTO 210

Line 1604 - if we're not in room 79 then there isn't anything worth oiling, so we inform the player and then go back to line 210. This is being kind to the player really, since he could have wasted his oil. Such largesse!

Line 1605 - nevertheless, if he now doesn't oil the track, object 17, he wastes all the oil, so we print up the message, remove the bottle of oil and replace it with an empty bottle, and go back to 210

Line 1606 - print the message of success.

Line 1608 - remove the bottle of oil, replace it with an empty bottle, and allow the player to go east from room 79 to room 80, and west to room 81.

Line 1610 - change the room description for room 79, get rid of the track now that we've solved the problem, and GOTO 210

STAB

Not a verb that is commonly encountered, and again the use here should serve to show how easy it really is to add tailor-made commands to any adventure scenario.

It is not a word that a lot of people would at first think of, although the presence of a sword should trigger off the idea in the minds of a few players.

Still, those familiar with *Lord of the Rings*, who will have read the passage about Shelob, should know that every good Hobbit always stabs a nasty spider with his sword, and that is indeed the use of the verb in Underground Adventure.

```
1650 IF no=0 THEN GO TO 1900
1651 GO SUB 5300
1652 IF o(no)<>cp AND o(no)<>-1 THEN PRINT "Where
is it?!": GO TO 210
1653 IF o(38)<>-1 THEN PRINT "You've nothing to st
ab it with.": GO TO 210
1654 IF no<>30 THEN PRINT o$(no): PRINT "is not im
pressed!": GO TO 210
1656 PRINT "The spider dies in a glorious  displa
y of bit-acting, and     reveals:"
1658 PRINT "a new passage way!": LET p(84,3)=86: L
ET p(84,4)=85: LET o$(30)="a dead spider"
1660 LET p$(84)="walking past a dead       spider."
: GO TO 210
```


Explanation

Line 1650 - usual check for an unrecognised noun.

Line 1651 - usual check for bottles and torches.

Line 1652 - if the object the player is trying to stab isn't in his possession and isn't in the room, then inform him of that fact and GOTO 210

Line 1653 - if you're not holding the sword, object 38, then you can't stab anything, so print the message up and go back to line 210

Line 1654 - if the player isn't trying to stab the spider, then print a suitable message and GOTO 210

Line 1656 - print the message.

Line 1658 - continue the message. Allow the player to go east from room 84 to room 86, and west to room 85. Change the description of object number 30, the spider.

Line 1660 - change the room description of room 84, and go back to line 210.

Again we didn't put in any checks to make sure that the player was in the correct room (IF CP< > etc.), but the checking in line 1652 takes care of that.

SPRAY

This could well be the first adventure to feature this word! I certainly can't think of any others with it, although there are no doubt some floating around somewhere.

Being an unusual word, one has to give the player some encouragement to use it, and the finding of the can of fly spray after eliminating the spider should give most people the right kind of idea.

A check is made to see if it is the fly that you're trying to spray, but as usual we've been kind to the player and not exhausted the fly spray if he sprays the wrong thing.

```
1700 IF no=0 THEN GO TO 1900
1701 GO SUB 5300
1702 IF o(no)<>cp AND o(no)<>-1 THEN PRINT "Where
is it?!": GO TO 210
1703 IF o(34)<>-1 THEN PRINT "You're not carrying
any spray.": GO TO 210
1704 IF no<>31 THEN PRINT "Cough-cough Splutter-sp
lutter!": GO TO 210
1706 PRINT "The evil fly coughs its last andreveal
s a hidden tunnel."
1708 LET p(74,4)=75: LET p$(74)="walking past a de
ad fly!": LET o$(31)="a dead fly": GO TO 210
```


Explanation

Line 1700 - you should be used to this by now!

Line 1701 - ditto!

Line 1702 - and again, our usual check for the presence of an object.

Line 1703 - if the player isn't holding the fly spray then he can't spray anything, so we print up the message and GOTO 210 as usual.

Line 1704 - if the object to be sprayed is not the fly, object 31, then the player only succeeds in making himself cough, and we go back to 210

Line 1706 - print the message.

Line 1708 - allow the player to go west from room 74 to room 75, change the room description of room 74, change the object description of object 31, and then GOTO 210

THROW

This is often given the same meaning as drop, but just as in real life, here we differentiate between a simple dropping of something, and a determined throw into the middle distance.

If we attempt to throw anything other than the lump of mortar or the axe, then it is treated as if the player just wished to drop the object in question, but those two particular objects have two very important roles to play, as we shall see :=

```
1735 IF no=4 AND np=1 THEN GO TO 6010
1750 IF no=0 THEN PRINT "Throw what?": GO TO 210
1751 IF no<>33 AND no<>4 THEN GO TO 552
1752 IF no=4 AND np=0 THEN GO TO 552
1753 IF no=4 AND np=1 THEN GO TO 6010
1754 IF o(33)<>-1 THEN PRINT "But you haven't got
it!": GO TO 210
1755 IF cp<>60 THEN PRINT "OK!  It vanishes in a c
loud of  dust.": LET o(33)=0: LET zz=zz-1: GO TO 2
10
1756 PRINT "The door shatters under the     force
of the blow, and reveals anew corridor."
1758 LET p(60,1)=61: LET p(60,4)=65: LET p$(60)="w
alking past the door.": LET o(33)=0
1760 LET zz=zz-1: LET df=1: GO TO 210
```


Explanation

Line 1750 - as usual.

Line 1751 - if the player isn't throwing the mortar or the axe then transfer program execution to the drop routine starting at line 552

Line 1752 - if the player is throwing the axe and the gargoyle present flag isn't set, then assume he just wants to drop it, so go to line 552 again.

Line 1753 - if the player is throwing the axe, and there is a gargoyle hanging around, then we go to line 6010 and continue the fight between player and gargoyle.

Line 1754 - check that the mortar is in the player's posession.

Line 1755 - if the lump of mortar is thrown anywhere other than in room 60, the room with the old door, then it just disappears in a cloud of dust, nothing else happens, we decrement the object counter, and GOTO 210.

Line 1756 - it's been thrown in the right place, so print the appropriate message.

Line 1758 - allow the player to go north from room 60 to room 61, and west to room 65, change the description of room 60, and remove the lump of mortar from the game.

Line 1760 - decrement the object counter, and set the door flag, before going off to line 210 again.

RUB

There are a number of things that one might be inclined to rub during an adventure, but the usual one is a lamp or torch, perhaps mindful of Aladdin and his lamp.

Indeed, rubbing the lamp in the original Adventure produces an interesting response, when you're told for the first time that the lamp is, in fact, an electrical one, so nothing much happens.

In Underground Adventure, nothing happens either, but people are wont to type in anything they can think of, so the listing goes something like :=

```
1800 IF no=0 THEN GO TO 1900
1801 GO SUB 5300
1802 IF o(no)<>cp AND o(no)<>-1 THEN PRINT "Where
is it?!": GO TO 210
1804 PRINT "Interesting, but unrewarding.": GO TO
210
```


Explanation

Line 1800 - usual check for the presence of an unknown noun.

Line 1801 - usual check in subroutine at 5300 for bottles and torches.

Line 1802 - usual check to see if an object is in the player's possession, or is in the room, and if it isn't print some kind of response and GOTO 210 again for another input.

Line 1804 - print the standard response to all RUBbing suggestions.

READ

Quite often one will find objects scattered about inside an adventure that look as if they might have something written on them, so the obvious command is to read object, to see what it says.

The replies are usually meant as helpful hints for the playing of the game, and set you thinking in a direction you might otherwise not have thought of.

Sometimes, however, they are anything but, and give you something like the weather forecast for five years ago, although even that usually makes you think of something. Occasionally they're not even written in English, as is the case with Spelunker Today, the magazine to be found in the original Adventure, which is written in Dwarvish!

```
1850 IF no=0 THEN GO TO 1900
1851 GO SUB 5300
1852 IF no<>48 THEN PRINT "There's nothing here to
 read.": GO TO 210
1854 IF o(48)<>-1 THEN PRINT "You're not holding i
t.": GO TO 210
1856 PRINT "There's material in here to make"
1858 PRINT "a ladder, like nails and planks"
1859 PRINT "and axes and things."
1860 PRINT "There's also more than a touch "
1862 PRINT "of magic in the air!": GO TO 210
```


Explanation

Line 1850 - usual check for an unrecognised word.

Line 1851 - usual check for bottles and torches with the subroutine at line 5300

Line 1852 - if they're not trying to read object 48, the old parchment, then tell them that nothing is written on it, and go back to line 210

Line 1854 - check to see if the object is in the player's possession, and if it isn't print a suitable message and GOTO 320 again.

Line 1856 - print the first part of the message contained on the old parchment.

Line 1858 - continued.

Line 1859 - continued.

Line 1860 - continued.

Line 1862 - end of message, and back to line 210 for more input.

EXAMINE

This is one of the most useful words in the adventurer's vocabulary, as any object should be able to be examined, and the examination of it will reveal valuable clues about it.

Even if the result of EXAMINE TORCH reveals nothing more than IT'S JUST AN OLD TORCH, it at least tells you that the torch has no magical powers (although someone might be fooling ...!)

More often, you'll be told something about the object, about its value, its usefulness, or its actual design.

In Underground Adventure, you're just told whether it's magical or not.

```
1900 IF no=0 THEN PRINT "What's a ";h$;"!": GO TO
210
1901 IF no=43 OR no=1 OR no=6 THEN PRINT "There's
nothing interesting here.": GO TO 210
1903 GO SUB 5300
1904 IF o(no)<>cp AND o(no)<>-1 THEN PRINT "Where
is it?!": GO TO 210
1905 IF no=2 OR no=16 OR no=33 OR no=37 OR no=38 T
HEN PRINT "This has useful powers.": GO TO 210
1906 PRINT "It's nothing more than:": PRINT o$(no)
: GO TO 210
```


Explanation

Line 1900 - usual check for an unrecognised noun.

Line 1901 - if they want to examine the wall, the chasm, or the bridge, then tell them there's nothing interesting here, and GOTO 210

Line 1903 - off to 5300 to check for torches and bottles.

Line 1904 - if the object is not in the player's possession, and isn't in the room, then it can't be examined, and so back to 210.

Line 1905 - if the player is examining the staff, the mirror, the brick, the shining stone or the sword, then he's told that it has useful powers, before GOTO 210

Line 1906 - otherwise just print that it's nothing more than : and the object description from O$(NO). Then, off to 210 again.

JUMP and BREAK

These are grouped together here because they don't take up much code, and they don't perform a great function in this particular game.

Nevertheless, JUMP could be a useful command in some games, enabling a player to jump across gaps that he couldn't simply walk across, if any player chose to take the risk.

Break is again not used here, but sometimes it could be used as a test of the player's ingenuity. Something could only be broken if, say, the bear was following the player, in which case the bear would have the strength to break the object for the player.

```
1950 IF cp=15 OR cp=10 OR cp=45 THEN PRINT "I told
 you so ....": PRINT l$: GO TO 612
1952 PRINT "Wheee!": GO TO 210
1960 IF no=0 THEN PRINT "Break what?": GO TO 210
1962 GO SUB 5300: IF o(no)<>cp AND o(no)<>-1 THEN
PRINT "I can't.  It isn't here.": GO TO 210
1964 PRINT "You're not strong enough to     break
anything by yourself!": GO TO 210

2000 GO TO 1960
```


Explanation

Line 1950 - if the player tries to jump in room 15, 10 or 45 (i.e. across a chasm or down a steep incline) then print a sarcastic message, print the variable L$, and go into the death routine.

Line 1952 - otherwise, print out a silly message and GOTO 210 again.

Line 1960 - usual check for unknown noun.

Line 1962 - if the object isn't in the player's possession and isn't in the room then he can't break it, so tell him so and then go back to 210

Line 1964 - tell the player the sad news, and GOTO 210 again.

Line 2000 - GOTO 1960 (because of computed GOTO)

PUSH

A verb that is used in a number of games, and one that could have been used in this one. As it is, an attempt to push the one thing that moves results only in the player being told to try doing this in another way.

```
1970 IF no=0 THEN PRINT "Push what?": GO TO 210
1971 GO SUB 5300
1972 IF o(no)<>cp AND o(no)<>-1 THEN PRINT "I can'
t.  It isn't here.": GO TO 210
1974 IF cp<>79 THEN PRINT "You can't!": GO TO 210
1976 PRINT "Try doing this another way, by  using
something else.": GO TO 210
1999 STOP

2050 GO TO 1970
```


Explanation

Line 1970 - familiar!

Line 1971 - familiar again!

Line 1972 - and again!

Line 1974 - if the player isn't in room 79, i.e. the one where the track is stuck, then there's nothing to move, so the player is told before sending program control back to line 210 again.

Line 1976 - print out a helpful message before going to line 210

Line 1999 - the last line before the great assembly of data begins!

Line 2050 - GOTO 1970 (because of computed GOTO)

SAVE

A useful, and one could say vital part of any adventure game, is the ability to stop a game in mid-flight and save one's progress onto tape, including all the room descriptions that change, the object descriptions, the positions of all the objects that have moved, the flags that indicate the successful or otherwise completion of a problem, and of course the room number.

In Underground Adventure, this is achieved by typing in SAVE PROG, in response to the WHAT NOW * prompt. It does just save the data, not the whole program!

```
2100 GO TO 3000

3000 CLS : PRINT "Insert cassette in tape unit,
and tap the space bar when ready"
3001 IF INKEY$="" THEN GO TO 3001
3002 IF INKEY$<>"" THEN GO TO 3002
3003 LET x(1)=cp: LET x(2)=tb: LET x(3)=gf: LET x(
4)=pd: LET x(5)=zz: LET x(6)=sc: LET x(7)=df: LET
x(8)=br: LET x(9)=np
3004 PRINT "OK."
3006 SAVE "data1" DATA p$(): GO SUB 6900
3007 SAVE "data2" DATA p(): GO SUB 6900
3008 SAVE "data3" DATA o$(): GO SUB 6900
3009 SAVE "data4" DATA o(): GO SUB 6900
3010 SAVE "data5" DATA x()
3026 GO TO 614

6900 PRINT "Stop the tape but do NOT rewind it yet
.": PRINT : PRINT "Press space bar when ready to
 continue."
6902 IF INKEY$="" THEN GO TO 6902
6903 IF INKEY$<>"" THEN GO TO 6903
6904 RETURN
```


Explanation

Line 2100 - GOTO 3000 (because of computed GOTO)

Line 3000 - tell the player to put a tape in the cassette unit and press space when ready.

Line 3002 - wait for the space bar to be pressed.

Line 3003 - put all the variables into an array for cassette storage.

Line 3004 - print OK.

Line 3006 - save all the room descriptions and GOSUB 6900.

Line 3007 - save the room direction data.

Line 3008 - save all the object descriptions and GOSUB 6900.

Line 3009 - save the position of all the objects, then GOSUB 6900 for next pressing of space bar.

Line 3010 - save all the variable flags and counters and GOSUB 6900.

Line 3026 - go to the routine at 614 requesting another game. The player may just have saved the data because he's about to try something risky!

Lines 6900 onwards - print message on screen and wait for space bar to be pressed before saving next lot of data.

LOAD

Vital of course, since we have got a save routine, and this just reads all the data back and starts the game off again at the point where it had finished.

To use this, just type in LOAD PROG in response to the first WHAT NOW * prompt.

```
2150 GO TO 3200

3200 CLS : PRINT "Insert cassette in tape unit,
and tap the space bar when ready"
3201 IF INKEY$="" THEN GO TO 3201
3202 IF INKEY$<>"" THEN GO TO 3202
3204 PRINT "OK"
3206 LOAD "data1" DATA p$()
3207 LOAD "data2" DATA p()
3208 LOAD "data3" DATA o$()
3209 LOAD "data4" DATA o()
3210 LOAD "data5" DATA x()
3212 LET cp=x(1): LET tb=x(2): LET gf=x(3): LET pd
=x(4): LET zz=x(5): LET sc=x(6): LET df=x(7): LET
br=x(8): LET np=x(9)
3226 GO TO 200
```


Explanation

Line 2150 - GOTO 3200, because of computed GOTO.

Line 3200 - tell the player to put a tape in the cassette unit and press space when ready.

Line 3202 - wait for the space bar to be pressed.

Line 3204 - print OK.

Line 3206 onwards - load in data in order saved, and convert variable flags back from array X$.

Line 3226 - go to the routine at 200 that starts the LOOK sequence.

The Rest of the Verbs

Just four more to go now, and they all perform fairly minor functions, so we'll group these up two to a page. There should still be enough space for your own notes later.

LOOK

This doesn't even have a line of its own, but just goes to line 200. This sends it off to the subroutine at line 5000.

All it consists of, to keep the computed GOTO happy, is:

```
400 GOTO 200
```

SCORE

Two lines only, and these are :

```
 500 PRINT "There are no points to be scoredin thi
s game, old chum.          You've just got to find
a way   out of here."
 502 GO TO 210
```

Just a simple message, and no points to be scored at all in this game. All you have to do is survive and get out!

In adventures where you are keeping a score, there are a number of ways of handling things. You'll need a score counter, say LS for latest score, and whenever the player requests his score, perhaps go through a loop checking to see if each treasure is stored in the correct place.

If it is, then add a value (say 10) to the score counter. You could also add a value to the score counter for every hazard successfully negotiated.

HELP

Another simple one, this could be used to great effect in some games, in giving vital clues for the sake of taking points away, but in Underground Adventure you get no help at all, like this :

```
 600 PRINT "I'm afraid that you won't get  much h
elp from me, so just keep on trying things.  If no
thing's happening, try using different  words inst
ead."
 602 GO TO 210
```

Only a simple message that tells you to keep examining things.

TAKE

In this game, TAKE functions in exactly the same way as GET, so program execution is just transferred to line 350 and everything dealt with in the usual way.

To keep the computed GOTO happy, all we have is:

```
750 GO TO 350
```

TAKE could be useful in some ways, as we've already mentioned, in that one talks of TAKEing medicine, rather than GETting it, and there are other ways in which the two words are different.

However, in Underground Adventure they behave in the same way.

That's the end of the verbs!

Let's get on and look at some data now.

Linking Everything Together

We've had to split up the various separate parts of Undergound Adventure in order to be able to explain properly how each section works.

Consequently, the listing is split up into a vast number of different sections scattered around the length and breadth of this book. However, every single line is in here somewhere, and the only section that we haven't yet seen is the data, and this follows immediately after this page.

It includes the data for the 100 rooms contained in the game, although some of these rooms are little more than tunnels and corridors. Whether you have this many in your games is up to you, since some people prefer the 'less rooms, more objects' principle of adventure writing.

This is all very well, and the trade-off in memory space saved is usually the equivalent of something like four or five rooms per object on the kind of system that we've been employing throughout the book. By all means have more objects than we've used here, but do realise that this will mean a corresponding rise in the number of verbs used.

No bad thing, but it all takes up memory space, and whether you want a lot of rooms, or a lot of objects, is up to you.

Personally, I prefer the more rooms approach. It gives you lots of space to explore about in, and means that the problems presented can be spaced out at reasonable intervals, rather than coming one after the other, with little chance for the adventure player to get a good feel for the game, and for the area he is exploring.

It also seems more realistic, in that a stroll underground in a set of caves is hardly likely to throw up hundreds of objects in each room, but will provide a lot of cross-linking tunnels and corridors for you to walk along.

But we've elected to go for a hundred rooms in total, and as we've seen we'll be giving you all the descriptions in a moment.

To sum up the job of typing in this entire listing: it is scattered about all over the place, but it is all here somewhere, with the data here,

the verbs earlier on in this chapter, most of the routines in the last section of chapter 4, and the moving room routine in the last section of chapter 3.

Of course, you can always buy the cassette of the game and save yourself a lot of time and trouble!

The Data

This is the complete collection of data for the entire adventure, and runs through the room data first, including description and direction, the initial locations and descriptions of the objects, the shortened forms of the object names, and of course the all important verbs.

A description of how each piece of data is used follows the listing.

```
2002 LET p$(1)="on an old track heading  towards t
he caves.": DATA 0,2,0,0
2003 LET p$(2)="getting ever nearer the  caves.":
DATA 1,3,0,0
2004 LET p$(3)="at the mouth of the caveswith path
s everywhere."
2005 LET p$(5)="in a subterranean tomb,  dotted wi
th crevices."
2006 LET p$(4)="in front of a solid wall of rock!"
2010 LET p$(6)="walking around the side  of the cr
evice room."
2012 LET p$(7)="surrounded by bricked up walls."
2013 DATA 2,15,20,4,0,0,3,0,6,13,4,9,0,5,0,7,0,9,6
,8,0,10,7,0,7,12,5,10
2014 LET p$(8)="near the great chasm in  the rock,
 which plunges down    hundreds of feet."
2016 LET p$(9)="in the heart of crevice  room,with
 many tunnels."
2018 LET p$(10)="in front of a great chasmthat you
 cannot jump."
2019 DATA 8,11,9,0,10,0,12,0,9,0,13,11,5,0,0,12,0,
0,10,0,3,0,0,0,0,18,0,15
2020 LET p$(11)="on the southern rim of   the chas
m."
2021 DATA 15,33,18,19,16,34,0,17,0,32,17,0,0,0,21,
3,0,0,0,20,23,0,0,21
2022 LET p$(12)="lost in chasm country!"
2024 LET p$(13)="in a room full of rocks, rocks, r
ocks and rocks."
2026 LET p$(14)="on the west side of the  chasm."
```

```
2028 LET p$(15)="faced with a crack that  is too w
ide to jump."
2030 LET p$(16)="in an east side chamber."
2032 LET p$(17)="on the main track throughthe cave
s."
2034 LET p$(18)="away from the main path  with a c
hoice of route."
2037 LET p$(19)="in a sharply twisting    corridor
."
2038 LET p$(20)="on a long east-west trackinto the
 depths!"
2040 LET p$(21)="forced to a halt by a    large un
derground tree."
2042 LET p$(22)="heading down a twisting  path int
o an old lair."
2044 LET p$(23)="surrounded by rock in a  mixture
of corridors.": DATA 25,22,24,0
2045 LET p$(24)="walking along an old     tunnel i
n the rocks."
2046 LET p$(25)="forced to turn sharply asthe path
 bends round.": DATA 26,0,0,23,0,23,26,0,27,24,0,2
5
2048 LET p$(26)="walking along a fairly   large co
rridor."
2049 GO TO 2051


2051 LET p$(27)="face to face with a very large be
ar!"
2052 LET p$(28)="at a t-junction behind   the bear
's lair.": DATA 0,26,0,0,0,27,30,29
2054 LET p$(29)="at a dead end.": DATA 0,0,28,0,31
,0,0,28,0,30,0,0
2056 LET p$(30)="near the heart of the    bear's h
iding place."
2057 LET p$(31)="in an old cave used as a bear's r
esting place."
2058 LET p$(32)="heading down an offshoot from the
 main mine.": DATA 19,42,33,41
2060 LET p$(33)="surrounded by shored up  timbers
and walls."
2062 LET p$(34)="crawling over stones and rubble o
n a low path.": DATA 17,0,34,32
2064 LET p$(35)="faced with a very deep   drop."
2065 DATA 18,0,35,33,0,0,0,34,0,0,38,0,0,38,39,0,3
7,0,0,36,0,0,40,37,0,0,0,39
2066 LET p$(36)="at the foot of the drop  with pat
hs everywhere."
2068 LET p$(37)="walking along an easterlycorridor
."
2070 LET p$(38)="forced into a sharp turn as the p
ath bends."
```



```
2072 LET p$(39)="in a long, low east-west corridor
."
2074 LET p$(40)="in a dead end and can go no furth
er."
2075 DATA 0,0,32,0,32,0,0,0,42,0,44,46,0,45,0,43,4
4,0,0,0,0,0,43,0,0,52,49,48
2076 LET p$(41)="well and truly stopped bya vast w
all of rock."
2078 LET p$(42)="face to face with a very angry pa
nther!"
2080 LET p$(43)="at an underground t-     junction
."
2082 LET p$(44)="near a great incline."
2084 LET p$(45)="at the foot of a great   incline.
"
2086 LET p$(46)="in a dead end."
2088 LET p$(47)="at the top of the great  incline.
"
2090 LET p$(48)="near an old scary part ofthe cave
s."
2092 LET p$(49)="near a reputedly magical part of
the caves ...."
2093 DATA 0,51,47,50,0,53,54,47,0,0,48,0,48,66,0,0
,47,77,0,0,49,100,0,0,0,0,88,49
2094 LET p$(50)="halted by the  ghostly   spirit o
f the caves."
2096 LET p$(51)="on an offshoot from the  main tra
ck."
2097 LET p$(53)="stopped by an extremely  narrow s
queeze."
2098 LET p$(52)="on an old path heading   north-so
uth."
2099 GO TO 2101
2101 DATA 56,57,50,58,0,55,0,0,55,0,0,0,0,0,55,59,
60,0,58,0,0,59,0,0
2102 LET p$(55)="in an open corridor with many exi
ts running off."
2103 LET p$(54)="near the magical caves."
2104 LET p$(56)="stuck in a dead end."
2106 LET p$(57)=p$(56)
2108 LET p$(58)="on a well trodden path   running
east-west."
2110 LET p$(59)="forced to turn as the    path bob
s and weaves."
2112 LET p$(60)="faced with door  marked  'begone
stranger'."
2114 DATA 61,61,61,62,61,61,61,63,61,64,61,61,61,6
1,65,61,61,61,60,61
2115 LET p$(61)="in a tortured maze of    little p
assages."
2116 LET p$(62)=p$(61)
```

```
2117 LET p$(63)=p$(61)
2118 LET p$(64)=p$(61)
2119 LET p$(65)=p$(61)
2120 LET p$(66)="walking along a dim path with dam
p walls."
2122 DATA 51,67,0,68,66,0,0,69,0,69,66,0,68,0,67,0
,0,71,69,0,70,72,0,74,71,0,0,73
2124 LET p$(67)="in a low,damp corridor."
2126 LET p$(68)="in a low corridor.It all seems ve
ry damp here."
2128 LET p$(69)="stopped by a wall of mistthat obs
cures light."
2130 LET p$(70)="on the south side of the mist in
clearer air."
2132 LET p$(71)="heading along a good pathcut from
 living rock."
2134 LET p$(72)="in a sharply twisting    corridor
."
2136 LET p$(73)="twisting and turning nearthe FLY
room!"
2138 LET p$(74)="face to face with a giantfly bloc
king the path!"
2140 DATA 74,0,72,0,0,73,71,0,0,0,74,76,0,0,75,0
2142 LET p$(75)="in a low east-west tunneldevoid o
f insects!"
2144 LET p$(76)="in a complete dead end   and can
go no further."
2146 DATA 52,78,0,0,77,79,0,0,78,0,0,0,0,83,0,79,0
,82,79,0,81,0,0,0,80,84,0,0
2148 LET p$(77)="still heading north-south"
2149 GO TO 2151
2151 LET p$(78)="at the bottom of a long  low nort
h-south path.."
2152 LET p$(79)="halted by an old seized  up minin
g track."
2154 LET p$(80)="weaving around old and   dusty co
bwebs."
2156 LET p$(81)="on the west side of the  track."
2158 LET p$(82)="in what was once known asthe salv
age room."
2160 LET p$(83)="near to the SPIDER room!"
2162 LET p$(84)="in spider-land.The vast  spider h
ere halts you!"
2164 LET p$(85)="in an old chamber known  as spide
rs' grave!"
2166 LET p$(86)="near to the SPIDER room!"
2168 LET p$(87)="in a total dead end. Yourroute en
ds here."
2169 DATA 83,0,0,0,0,0,84,0,0,87,0,84,86,0,0,0
2170 DATA 89,90,92,54,0,88,91,0,88,94,0,0,0,92,93,
89,91,0,97,88,0,97,0,91
```



```
2172 LET p$(88)="in the heart of the magiccaverns.
"
2173 LET p$(89)="in a northern offshoot   from the
 main path."
2174 LET p$(90)="walking along a magical  corridor
."
2176 LET p$(91)="in a dimly lit tunnel."
2178 LET p$(92)="near to the source of themagic."
2180 LET p$(93)="halted by a magical shi- mmering
curtain."
2182 DATA 90,0,0,0,0,0,93,96,0,99,0,98,95,93,0,0,92,
0,0,0,0,96,0,96,0,0,53,0,0,0
2184 LET p$(94)="in no-mans land.  The    path end
s here."
2186 LET p$(95)="on the northern side of  the shim
mering curtain."
2188 LET p$(96)="in a low corridor."
2189 LET p$(97)="treading over dimly lit  rocks an
d rubble."
2190 LET p$(98)="in a dead end.  The wall is brick
ed up here."
2191 LET p$(99)="in a northern offshoot   from the
 main path."
2192 LET p$(100)="in an old warehouse once used to
 store tools."
2200 FOR i=1 TO z: FOR j=1 TO 4: READ p(i,j): NEXT
 j: NEXT i
2210 DATA 15,20,21,34,24,0,40,0,27,7,42,46,0,14,93
,67,79,48,98,69
2212 DATA "vast chasm","a staff","a tree","an axe"
,"a rope"
2214 DATA "a bridge","dynamite","rubble","a bear"
2216 DATA "a bun","a panther","a plank","a ladder"
,"some nails"
2218 DATA "a curtain","a mirror","a track"
2220 DATA "some oil","a bottle","misty wall"
2222 FOR i=1 TO 20: READ o(i): NEXT i: FOR i=1 TO
20: READ o$(i): NEXT i
2224 DATA 50,84,74,60,76,87,3,53,63,31,73,0,0,100,
0,3,1,0,0,39,0,0,0,0,0
2226 DATA "the ghost!","a spider!"
2228 DATA "a fly!","a door","a mortar","fly spray"
2230 DATA "a gate","a crack","a stone","a sword","
whisky!"
2232 DATA "a gargoyle","an knife","a key","a wall"
,"matches","a  torch"
2234 DATA "lit torch","a light","parchment","progr
am"
2236 DATA "glass"
2238 DATA "oilbottle","whisky jar","timber"
2239 FOR i=29 TO lo: READ o(i): NEXT i
```

```
2240 FOR i=29 TO lo: READ o$(i): NEXT i
2250 DATA "cha","sta","tre","axe","rop","bri","dyn
","rub","bea","bun","pan","pla","lad","nai","cur",
"mir","tra"
2252 DATA "oil","bot","mis","nor","sou","eas","wes
","n","s","e","w","gho","spi","fly","doo","mor","s
pr","gat","cra"
2254 DATA "sto","swo","whi","gar","kni","key","wal
","mat","tor","tor","lig","par","pro","gla","bot",
"bot","tim"
2256 DATA "go","get","loo","inv","sco","dro","hel"
,"qui","cro","tak","ope","clo","eat","fee","dri","
off","wav"
2258 DATA "cut","cho","cli","lig","att","kil","hit
","mak","ref","oil","sta","spr","thr","rub","rea",
"exa","jum"
2260 DATA "bre","pus","sav","loa"
2262 FOR i=1 TO nn: READ j$(i): NEXT i
2264 FOR i=1 TO nv: READ v$(i): NEXT i
2266 DATA "north","south","east","west"
2268 FOR i=1 TO 4: READ t$(i): NEXT i
2270 RETURN
```

Using the Data

Here we'll explain how all the data is used, and how it all works. In other words, what are all those words and numbers that you've just typed in!

We'll start off with the room data.

Data for the Rooms

There are one hundred rooms in all, and each one is given a description. Some of these descriptions are used for a number of different rooms, in particular in the maze where we want to confuse the player totally.

The room descriptions are stored in the variable P$(I), where P$(I) contains the description of the Ith room, which is used in the routine from line 5000 onwards when actually printing the description onto the screen.

Using strings in this way naturally limits the length of description that we can give to a room, since we had to define a maximum string length of 40 characters when dimensioning our array. The Spectrum works in mysterious ways its arrays to handle!

Associated with each room are four numbers, stored in the variable P(I,J), where P(I,J) refers to the Jth direction from room I.

For instance, the four values for room 1 are 0,2,0,0. This means the player cannot go north, east or west, but can go south. Moving south will take him to room 2, which has the data 1,3,0,0. This signifies that the player can move north to room 1, south to room 3, but cannot move east and west.

In room three, we have our first choice of routes, since the data for room three is 2,15,20,4 : the player can go north to room 2, south to 15, east to 20, and west to 4.

Judicious use of room numbering can greatly enhance an adventure, although this is by no means the only system in use today. However, it is possibly one of the easiest to master, and is certainly easy to program.

Data for the Nouns

Just like the rooms, each noun, or object, has two variables associated with it, and these are O$(I), used to refer to the description of the Ith object, and O(I), which holds the current room number of the Ith object. If this number is a zero it isn't currently in the game, and if it is equal to minus 1, it is in the possession of the player.

In line 2222 we read in this data for the first 20 objects, position first, and then the lengthy description.

There then follows a gap of eight object descriptions and positions, as these are used to hold the words NORTH, SOUTH, EAST, WEST, N, S, E, W respectively. This is so that the program can actually understand the command GO NORTH, etc.

In lines 2239 and 2240 the next set of descriptions and locations are read in for the objects from 29 up to the upper limit set by the variable LO, as defined in line 2001.

The shortened forms for the nouns, that is, the words that we use when analysing any data that has been typed in, are stored in lines 2250 to 2254, and are read in as J(I) in line 2262.

Data for the Verbs

This is only of use when analysing what has been typed in, and obtaining a verb number, which is then used in line 240 of the program in order to send program execution off to the correct part of the program.

The data, in three-letter format for speed of verb identification, is stored in lines 2256 to 2260, and is read into the variable V$(I) in line 2264.

This data is used throughout the program to keep the adventurer on the move, and the large number of verbs provided ensures that a reasonable degree of interest should be maintained throughout the duration of the game.

The final lot of data, in line 2266, is only used once, in the routine starting up at line 5000, to print out the directions which our intrepid explorer can take.

It is read in in line 2268, in the order that the numbers in the variable P(I,J) are read. That is, NORTH first, then SOUTH, EAST and WEST.

And that's it! A whole adventure!

Conclusion on Underground Adventure

It is not the world's greatest adventure, simply because we have explained it all in great detail, so that you now know precisely how it all works, and could probably solve it in a matter of one or two sittings.

Nevertheless, it is not to be decried because of that, if it achieves its aim: that of presenting clearly and logically a complete adventure game listing, that anyone could take and adapt to produce his own compelling adventure games.

Machine Code Adventuring

This approach, in Basic, is obviously limited, and it would be possible to write much faster games in machine code. However, to write an adventure in machine code would be the work of many, many months,

possibly even years, and most of us want to see results in far less time than that!

Using the approach outlined here, it should be possible to produce adventure games at a reasonable rate, although a programmer's utility is virtually essential for writing a program this long.

Finding all the occurrences of the variable P(54, anything), and others, are problems you want answers to all the time, and most Basics aren't equipped with such useful functions as these!

Role-Playing Adventures

We also haven't really considered adventure role-playing games, although it is a subject I may tackle at a later date. Still, we have given a few brief outlines here, and even the simple approach followed throughout this book could be used as the model for a role-playing game.

The number of rooms would have to be a little less, but within reason, and with some competent programming, the same level of difficulty, the same kind of vocabulary, and the same number of objects, could all be retained to provide a fascinating game.

The one real limitation of this approach is that of the acceptance of an input from the user. We have restricted ourselves to the purely VERB OBJECT school, although this hasn't stopped a large number of adventures from being very successful programs in the past, viz. those of Crowther and Woods, Adams, *et al.*

Verbal Adventures

To go in for a greater level of response is possibly beyond Basic, as it would take a long time to sort through the response and break it down into its proper component parts. Just because the program can accept something like VERB OBJECT ACTION, i.e. like 'Take the Box and Close the Lid', doesn't mean that the player will always want to use all of those options, and the program, unless cleverly and quickly written, could find itself getting into a terrible muddle.

But the purpose of this book, and the game Underground Adventure, was to get you exploring adventures and writing them, and on a good level we have, I hope, succeeded.

Have fun adventuring, and we'll leave you with two final listings, Tunnel Adventure and Castle Adventure.

7

Castlemaze Adventure

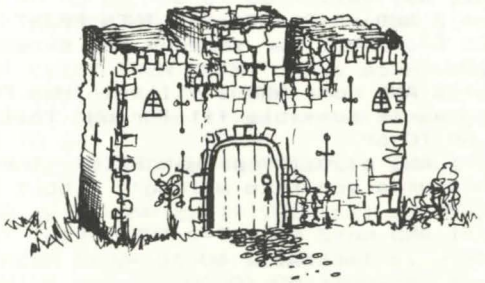
Introduction

This is a full-blown adventure listing, written using the same routines as Underground Adventure, so you should be able to follow what's going on.

It isn't as sophisticated in looks as the first game, but it is a challenging adventure that should keep you occupied for many a long day. Of course, if you cheat by looking at the listing you'll solve it very quickly, but you wouldn't do that, would you...!

We've already given you the map for this, so you should know what's going on, but watch out for the evil sorceror and the Black Knight. Oh yes, and the deadly maze is VERY deadly!

Have fun!



```
  1 GO SUB 4000
  2 LET g$="A Gold Bar falls out!": LET d$="Gulp-
Gulp-Gulp! You're shrinking"
  3 LET i$="You haven't got it."
  4 LET x=RND: LET x=0: LET zz=1: LET pi=12
  5 LET kn=0: LET t=0: LET df=0: LET cf=0: LET pf
=0: LET vf=0: LET sh=0: LET wf=0: LET sp=1
  9 LET v$="Behind the sign is a vault in   the w
all.  The vault is locked."
 15 LET b$="You must supply a direct object."
 17 DEF FN r(q)=INT (RND*q+1)
 19 LET cp=49: LET s$="I don't see it here.": LET
 r$="Don't be ridiculous.": GO TO 1700
 20 GO TO 1500
 30 IF cp=52 AND kn=0 THEN GO TO 1170
 40 IF o(2,1)=-1 AND pi=cp THEN GO TO 1240
 50 IF cp=29 AND sp=0 THEN GO TO 1330
 60 LET t=t+1: GO SUB 1430: IF a$="3.1" THEN LET
p(30,3)=31: GO TO 20
 70 IF vb=-1 AND (no>21 AND no<30) THEN LET vb=1
 90 IF a$="cro" THEN IF (cp=52 OR cp=53) THEN LET
 cp=105-cp: GO TO 20
 110 IF vb<>30 AND (vb>10 OR vb=20 OR vb=6) AND n$
=="" THEN PRINT b$: GO TO 60
 140 IF vb=30 THEN GO TO 1100
 160 IF vb=-1 AND no<>0 AND (no<22 OR no>29) THEN
PRINT "You must supply a verb.": GO TO 60
 170 IF vb<1 AND no=0 THEN PRINT "I don't know how
 to do that.": GO TO 60
 190 IF no=0 AND (vb>10 OR vb=6 OR vb=2) THEN PRIN
T "What's a ";c$;"?": GO TO 60
 200 GO TO vb*30+200
 210 IF (no<22 OR no>29) AND a$<>"" THEN PRINT "Yo
u have me completely baffled.": GO TO 60
 220 IF n$="" THEN PRINT "Where?": GO TO 60
 230 IF (no<22 OR no>29) AND a$<>"" THEN PRINT "Yo
u have me completely baffled.": GO TO 60
 231 IF a$="" THEN PRINT "Where?": GO TO 60
 232 IF no>25 THEN LET no=no-4
 233 LET no=no-21: IF p(cp,no)=0 THEN PRINT "You c
an't go that way yet.": GO TO 60
 234 IF cp=17 AND no=2 AND cf=0 THEN PRINT "The cr
ack is much too small for someone the size of you!
": GO TO 60
 235 IF cp=18 AND no=1 AND o(9,1)=-1 THEN PRINT "T
he painting won't possibly fit through that little
 crack.": GO TO 60
 236 IF no=1 AND o(20,1)=cp THEN PRINT "The Sinist
er Sorceror turns you into a frog!  Ribbit Ribbit.
": GO TO 1220
 237 IF cp=1 AND no=2 AND df=0 THEN PRINT "HaHa!Th
e castle door is locked.": GO TO 60
 238 LET cp=p(cp,no): GO TO 20
 260 IF o(no,1)=-1 THEN PRINT "You've already got
it!": GO TO 60
```



```
 262 IF no=0 THEN PRINT "What's a ";c$;"?": GO TO
60
 264 IF o(no,1)<>cp THEN PRINT s$: GO TO 60
 266 IF no=17 OR no=21 OR no=20 OR no=16 THEN PRIN
T r$: GO TO 60
 268 IF zz>4 THEN PRINT "You're carrying too much
already": GO TO 60
 270 IF no=19 AND pf=0 THEN PRINT v$: LET pf=1: LE
T o(16,1)=cp: LET o(19,1)=-1: LET zz=zz+1: GO TO 6
0
 272 PRINT "OK.": LET zz=zz+1: LET o(no,1)=-1: GO
TO 60
 290 GO TO 20
 320 IF zz=0 THEN PRINT "You aren't carrying anyth
ing.": GO TO 60
 322 PRINT "You are carrving the following:": FOR
i=1 TO lo: IF o(i,1)=-1 THEN PRINT o$(i)
 324 NEXT i: PRINT : GO TO 60
 350 GO SUB 352: GO TO 60
 352 LET j=0: FOR i=1 TO lo: IF o(i,1)=1 THEN LET
j=j+o(i,2)
 354 NEXT i: PRINT "You have scored ";j;" points":
 PRINT "out of a possible hundred.": IF j<100 THEN
 RETURN
 356 PRINT : PRINT "Well Done!": STOP
 380 IF no<>0 AND o(no,1)<>-1 THEN GO TO 980
 382 IF no=0 THEN PRINT "I've never heard of a ":c
$: GO TO 60
 384 IF no=18 AND o(13,1)<>cp THEN LET o$(18)="A s
hattered vase": LET o(18,2)=0
 386 LET o(no,1)=cp: LET zz=zz-1: PRINT "OK.": GO
TO 60
 410 IF cp<8 THEN PRINT "Be persistent!": GO TO 60
 412 IF cp<20 THEN PRINT "Examine things.": GO TO
60
 414 IF cp<24 THEN PRINT "What goes up must come d
own!": GO TO 60
 416 IF cp<34 THEN PRINT "Try valuing a few things
.": GO TO 60
 418 IF cp<41 THEN PRINT "Do as Hansel and Gretel
did.": GO TO 60
 420 IF cp<45 THEN PRINT "Just try thinking a litt
le!": GO TO 60
 422 IF cp<52 THEN PRINT "This adventure has a vio
lent    beginning.": GO TO 60
 424 PRINT "Why don't you try crossing the  bridge
?": GO TO 60
 440 IF cp<49 AND cp>44 THEN LET cp=cp-25: GO TO 2
0
 442 IF cp<24 AND cp>19 THEN LET cp=cp+25: GO TO 2
0
 444 PRINT "That isn't possible.": GO TO 30
 470 PRINT "You need some kind of tool.": GO TO 60
 500 CLS : PRINT "Byeee!": GO SUB 352: STOP
 530 PRINT "All right then - ";c$: GO TO 60
```

```
 560 IF o(no,1)<>-1 THEN PRINT i$: GO TO 60
 562 IF no<>7 THEN PRINT r$: GO TO 60
 564 PRINT d$: LET zz=zz-1: LET o(7,1)=0: LET cf=1
: GO TO 60
 590 IF no<>31 AND no<>16 AND no<>30 THEN PRINT "I
 don't know how to open such a thing.": GO TO 60
 592 IF no=16 AND o(16,1)<>cp THEN PRINT "What vau
lt?": GO TO 60
 594 IF no=16 AND o(2,1)<>-1 THEN PRINT "You don't
 appear to possess the key.": GO TO 60
 596 IF no=16 THEN PRINT "The vault is now open.":
 LET vf=1: IF o(15,1)=0 THEN PRINT g$: LET o(15,1)
=cp
 598 IF no=16 THEN GO TO 60
 600 IF no=31 THEN GO TO 1140
 602 IF cp<>1 THEN PRINT "What door?": GO TO 60
 604 IF o(2,1)<>-1 THEN PRINT "You appear not to h
ave the key.": GO TO 60
 606 PRINT "The door is now open.": LET df=1: GO T
O 60
 620 GO TO 280
 650 PRINT "And just how do you expect me todo tha
t?": GO TO 60
 680 IF o(no,1)<>-1 THEN PRINT "You don't seem to
have it.": GO TO 60
 682 IF no<>3 THEN PRINT "How am I supposed to rea
d that?!": GO TO 60
 684 PRINT "It says 'A secret passage lies  nearby
.  It opens if you number PI'": GO TO 60
 710 IF o(no,1)<>-1 AND o(no,1)<>cp THEN PRINT "I
 can't see it here.": GO TO 60
 712 IF o(1,1)<>-1 THEN PRINT "You haven't got a b
ow!": GO TO 60
 714 IF o(4,1)<>-1 THEN PRINT "You haven't got an
arrow!": GO TO 60
 716 PRINT "OK.": LET zz=zz-1: LET o(4,1)=cp: GO T
O 60
 740 GO TO 650
 770 IF no=16 OR no=30 OR no=31 THEN GO TO 776
 772 IF o(no,1)<>-1 THEN PRINT "You haven't got it
!": GO TO 60
 774 PRINT "I don't quite know how to close that."
: GO TO 60
 776 IF no=16 AND o(16,1)<>cp THEN PRINT "What vau
lt?": GO TO 60
 778 IF no=16 THEN PRINT "The vault is now well an
d truly closed.": LET vf=0: GO TO 60
 780 IF no=31 THEN GO TO 1119
 782 IF cp<>1 THEN PRINT "What door?": GO TO 60
 784 PRINT "The door is now closed and      locked
.": LET df=0: GO TO 60
 800 GO TO 770
 830 GO TO 590
 860 GO TO 380
 890 IF no=8 OR no=14 THEN PRINT "Try 'swing'": GO
```



```
 TO 60
 892 IF no=1 OR no=4 THEN PRINT "Try 'shoot'": GO
TO 60
 894 IF no=10 THEN PRINT "Try 'sharpen'": GO TO 60
 896 IF no=31 THEN PRINT "Try 'jump'": GO TO 60
 898 IF no=13 THEN PRINT "Just 'drop' it where you
 want it": GO TO 60
 900 PRINT "I don't follow you.": GO TO 60
 920 PRINT "It has a value of ";o(no,2): PRINT "po
ints": GO TO 60
 950 IF o(no,1)<>-1 THEN PRINT "You don't seem to
have it.": GO TO 60
 952 IF no<>14 THEN GO TO 960
 954 FOR i=1 TO 19: IF o(i,1)=-1 THEN LET o(i,1)=c
p
 956 NEXT i: LET zz=0: LET cp=23: GO TO 20
 960 IF no<>8 THEN PRINT "Wow!  This is fun!": GO
TO 60
 962 IF o(29,1)<>cp THEN PRINT "Swoosh!": GO TO 60
 964 IF sh=0 THEN PRINT "The sword bounces off the
      sorceror and hits you!": GO TO 1220
 968 PRINT "The sharp sword slices the      sorcer
or.": LET sh=sh+1: IF sh<4 THEN GO TO 30
 970 LET o(20,1)=0: LET o(14,1)=cp: PRINT "The sor
ceror disappears in a    cloud of greasy black smo
ke.": GO TO 30
 980 IF o(no,1)<>-1 THEN PRINT "You aren't carrvin
g it.": GO TO 60
 982 IF no<>8 THEN PRINT r$: GO TO 60
 984 IF o(10,1)<>-1 THEN GO TO 470
 986 PRINT "Your sword is now razor sharp.": LET s
h=1: GO TO 60
1010 IF (cp=1 AND no=30) OR (cp=44 AND no=31) THEN
 GO TO 1022
1011 IF o(no,1)<>-1 AND o(no,1)<>cp THEN PRINT s$:
 GO TO 60
1012 IF no=17 AND o(2,1)=0 THEN PRINT "Something's
 in his pocket!": LET o(2,1)=cp: GO TO 60
1013 IF no=21 AND o(6,1)=0 THEN PRINT "Something's
 in it's stomach!": LET o(6,1)=cp: GO TO 60
1014 IF no=8 AND sh=0 THEN PRINT "It's a bit blunt
!": GO TO 60
1016 IF no=8 THEN PRINT "It's very sharp.": GO TO
60
1018 IF no=7 THEN PRINT "On the bottom it says 'dr
ink me'": GO TO 60
1020 IF no=18 THEN PRINT "It's very fragile.": GO
TO 60
1022 IF no=31 THEN PRINT "It's big enough to jump
out of.": GO TO 60
1024 IF no=1 OR no=30 THEN PRINT "It's made of woo
d.": GO TO 60
1025 IF no=19 AND o(19,1)=35 THEN PRINT "It's just
 hanging there.": GO TO 60
1026 IF no=20 THEN PRINT "He's preparing to cast a
```

```
 spell  on you!": GO TO 60
1028 IF no=13 THEN PRINT "It's soft.": GO TO 60
1030 IF no=10 THEN PRINT "It's grey and gritty.":
GO TO 60
1038 PRINT "It's nothing more than:": PRINT o$(no)
: GO TO 60
1040 GO TO 380
1070 IF no=4 AND o(4,1)=22 THEN PRINT "Look for it
 in the forest.": GO TO 60
1072 IF no=2 AND o(2,1)=0 THEN PRINT "Examine thin
gs.": GO TO 60
1074 IF o(no,1)=-1 THEN PRINT "You're carrying it,
 stupid!": GO TO 60
1076 IF o(no,1)=cp THEN PRINT "It's right in front
 of you,     stupid!": GO TO 60
1078 IF no<>20 THEN PRINT "Just pull yourself toge
ther and look for it.": GO TO 60
1080 PRINT "You're in the Sorceror's torturechambe
r -- he's armed with a redhot poker and he's comin
g       towards you!": FOR j=1 TO 3: GO SUB 1430
1082 IF vb=25 AND no=14 AND o(14,1)=-1 THEN GO TO
950
1084 PRINT "The Sorceror thrusts the sword  at you
!": NEXT j: GO TO 1220
1110 IF (cp>19 AND cp<24) OR cp=34 THEN PRINT "Goi
ng down .....": GO TO 1220
1111 IF cp<>44 THEN PRINT "Wheee!": GO TO 60
1112 IF wf=0 THEN LET cp=43: GO TO 1111
1113 PRINT "You land safely in the branches of a t
ree.": LET cp=21: GO TO 60
1119 IF cp<>44 THEN PRINT "I see no windows.": GO
TO 60
1120 IF wf=0 THEN PRINT "It's already closed.": GO
 TO 60
1130 PRINT "It's stuck!": GO TO 60
1140 IF cp<>44 THEN GO TO 1119
1150 IF wf=1 THEN PRINT "It's already open!": GO T
O 60
1160 PRINT "It's not easy, but you manage toget th
e window open.  You see a big leafy tree about 2 m
etres   below the window.": LET wf=1: GO TO 60
1170 PRINT "A Black Knight is riding across the br
idge towards you!": GO SUB 1430
1180 IF vb<>17 OR no<>17 THEN GO TO 1210
1190 IF o(1,1)<>-1 THEN PRINT "You have no bow!":
GO TO 1210
1195 IF o(4,1)<>-1 THEN PRINT "You have no arrow!"
: GO TO 1210
1200 PRINT "The arrow finds a chink in the  knight
's armour, and he falls tohis death."
1205 LET kn=1: LET zz=zz-1: LET o(4,1)=52: LET o(1
7,1)=52: GO TO 60
1210 PRINT "The knight skewers you with his lance.
"
1220 PRINT : PRINT "You're dead, old bean.": GO TO
```



```
 1370
1240 PRINT "A Pirate sneaks up on you and  steals
 the key.  'Har Har Har'  he chortles.  'I'll hide
 this   deep in me maze!"
1290 LET o(2,1)=34: LET zz=zz-1: GO TO 60
1330 PRINT "A giant spider drops down from  the ce
iling!"
1335 PRINT "It's moving towards you!": GO SUB 1430
1340 IF o(1,1)<>-1 THEN PRINT "You have no bow!":
GO TO 1350
1342 IF o(4,1)<>-1 THEN PRINT "You have no arrow!"
: GO TO 1350
1345 PRINT "The arrow rips into the spider  and de
als him a gory death!": LET sp=1: LET zz=zz-1
1347 LET o(21,1)=29: LET o(4,1)=0: LET o(5,1)=29:
GO TO 60
1350 PRINT "The spider pounces on you and  sinks
its fangs into your neck!": GO TO 1220
1370 GO SUB 352: PRINT "Bye!": STOP
1430 PRINT "What Now * ? ";: GO SUB 5000: PRINT m$
: PRINT
1440 LET c$="": LET e$="": LET no=0: LET vb=0: LET
 n$="": LET a$=""
1450 LET cm=LEN m$: FOR i=1 TO cm: IF m$(i TO i)<>
" " THEN LET e$=e$+m$(i TO i): NEXT i
1452 IF e$="go" THEN LET vb=1: LET a$="go": GO TO
1480
1454 IF LEN e$<3 THEN PRINT "Que?": GO TO 60
1460 LET a$=e$(1 TO 3): FOR i=1 TO nv: IF a$=x$(i)
 THEN LET vb=i: GO TO 1480
1465 NEXT i
1470 LET vb=-1: LET c$=e$: GO TO 1490
1480 IF LEN e$+1>cm THEN LET no=0: RETURN
1485 LET c$=m$(LEN e$+2 TO cm)
1488 IF LEN c$<3 THEN PRINT "You can't mean that?!
": GO TO 60
1490 LET n$=c$(1 TO 3): FOR i=1 TO nn: IF n$=y$(i)
 THEN LET no=i: RETURN
1493 NEXT i
1495 LET no=0: RETURN
1500 CLS : PRINT "You're ";p$(cp): PRINT : LET z$=
"You can see:                    ": FOR i=1 TO lo
1510 IF o(i,1)=cp THEN PRINT z$;o$(i): LET z$=""
1520 NEXT i
1530 IF cp=1 AND df=0 THEN PRINT "The door is lock
ed."
1540 IF cp=18 AND vf=0 AND o(16,1)=18 THEN PRINT "
The vault is locked."
1550 IF cp=17 AND cf=0 THEN PRINT "A narrow crack
leads southwards."
1560 IF cp=1 AND df=1 THEN PRINT "The door is open
."
1570 IF cp=35 AND vf=1 AND o(16,1)=35 THEN PRINT "
The vault is open."
1590 IF cp=17 AND cf=1 THEN PRINT "A wide crack le
```

```
ads off to the   south."
1600 IF cf=0 THEN LET p(17,2)=0
1610 IF cp=44 AND wf=1 THEN PRINT "The window is o
pen.  A tree lies2 metres below you."
1620 LET k=0: PRINT : PRINT "You can go:": PRINT :
 FOR i=1 TO 4: IF p(cp,i)=0 THEN GO TO 1650
1640 PRINT t$(i);" ";: LET k=1
1650 NEXT i: IF k=0 THEN PRINT "Nowhere."
1660 IF k=1 THEN PRINT
1670 PRINT : LET p(17,2)=18: GO TO 30
1700 LET np=53: LET lo=35: LET nn=31: LET nv=30: D
IM p(np,4): DIM p$(np,80): DIM x$(nv,3): DIM y$(nn
,3): DIM o(lo,2): DIM o$(lo,31): DIM t$(4,5)
1710 LET p$(1)="outside a medieval castleThe pavem
ent bears the message  'leave treasures here."
1715 DATA 0,8,4,0,53,7,3,6,0,0,3,2
1720 LET p$(2)="standing at a great crossroads, wi
th roads stretching outof sight everywhere."
1730 LET p$(3)="walking along the great  east road
.": LET p$(4)="walking along the great  west road.
": DATA 0,0,2,1,0,0,2,4
1740 LET p$(5)=p$(4): LET p$(6)=p$(4): DATA 0,0,2,
5
1755 LET p$(7)="walking along the great  south roa
d.": DATA 2,7,0,0
1760 LET p$(8)="standing in a splendid   chamber t
hat is at least thirty feet high.": DATA 1,11,0,10
1770 LET p$(9)="in a cosy and comfortablesitting r
oom.": DATA 10,0,11,0
1780 LET p$(10)="standing in the superb   master b
edroom. The decoration  is a bit gaudy!": DATA 0,9
,8,0
1790 LET p$(11)="in a vast corridor that  stretche
s out of sight to the   south.": DATA 8,12,14,9
1800 LET p$(12)="in a vast corridor that  stretche
s out of sight to the   north and south.": DATA 11
,13,15,25
1810 LET p$(13)="in a vast corridor that  stretche
s out of sight to the   north.": DATA 12,0,16,17
1820 LET p$(14)="in a cold bedroom with a stone fl
oor."
1822 LET p$(15)="in a cosy bedroom with a wooden f
loor."
1824 LET p$(16)="in a dusty bedroom with adirty fl
oor."
1826 DATA 0,0,0,11,0,0,0,12,0,33,0,13
1830 LET p$(17)="standing in a very dusty pantry t
hat looks like it hasn'tbeen touched for years.":
DATA 0,18,13,24
1840 LET p$(18)="in a private art gallery.Great po
rtraits adorn all the   walls.": DATA 17,26,0,19
1850 LET p$(19)="in an old storeroom, withcobwebs
hanging everywhere.": DATA 0,0,18,0
1860 LET p$(20)="at the top of a large    bushy tr
ee.": DATA 0,0,0,0
```



```
1870 LET p$(21)=p$(20): DATA 0,0,0,0
1880 LET p$(22)=p$(20): DATA 0,0,0,0
1890 LET p$(23)=p$(20): DATA 0,0,0,0
1910 LET p$(24)="in the ancient kitchen.  It has o
bviously seen better    days in the past.": DATA 2
5,0,17,0,0,24,12,0,18,28,29,0
1920 LET p$(25)="in the old dining room,  now no l
onger full of old       diners.": LET p$(26)="in a
 shadowy alcove."
1925 LET p$(27)="in a suprisingly austere office,
that has survived the   passage of time well.": DA
TA 0,0,28,0
1930 LET p$(28)="in the old drawing room.": DATA 2
6,0,0,27,0,0,30,26,0,0,0,29
1940 LET p$(29)="standing in what was oncethe parl
or.": LET p$(30)="standing in what was oncethe mas
ter's study."
1945 LET p$(31)="in a damp stone passage  that chi
lls you to your bones."
1950 LET p$(32)="in the deep, the dark,   the dism
al, the dank, the nasty nasty dungeon!": DATA 0,0,
32,30,41,0,0,31
1960 FOR i=1 TO np: FOR j=1 TO 4: READ p(i,j): NEX
T j: NEXT i
1970 LET p$(33)="standing in the ancient  conferen
ce room,overcome by con-ference room lassitude.":
DATA 16,0,35,0
1980 LET p$(34)="in a tower overlooking a huge kin
gdom down a monstrous   mountain!": DATA 40,0,0,0
1990 LET p$(35)="in a familiar- looking   maze of
twisty little passages. They all look the same."
1995 FOR i=36 TO 40: LET p$(i)=p$(35): NEXT i: DAT
A 36,36,36,33,37,35,35,35,36,36,38,36
1996 DATA 36,36,36,39,40,36,36,36,36,34,36,36,42,3
2,0,0
2000 LET p$(41)="on a long,long flight of stairs g
oing down."
2001 LET p$(42)="in a mile-long passage   with hor
rible stagnant water    lapping at your feet."
2002 LET p$(43)="on a long,long flight of stairs g
oing up."
2010 LET p$(44)="at the end of the castle.You can
see a forest out of a   small window."
2015 DATA 43,41,0,0,44,42,0,0,0,43,0,0
2020 LET p$(45)="in a dense dark forest."
2025 FOR i=46 TO 48: LET p$(i)=p$(45): NEXT i: DAT
A 45,49,46,48,45,47,46,49,50,51,46,48
2026 DATA 45,47,49,48
2030 LET p$(49)="on a familiar little     path, on
ce trod by horses."
2035 LET p$(51)="in the middle of a clear-ing.  To
 the south is your fav- ourite bridge."
2037 LET p$(52)="standing on the north    side of
the bridge."
2038 LET p$(53)="standing on the north    side of
```

```
the bridge."
2039 DATA 45,50,46,48,49,47,46,48,47,52,0,0,51,0,0
,0,0,2,0,0
2040 DATA "bow","key","boo","arr","bro","sap","liq
","swo","pai","whe","sil","pen","pil","sce"
2045 DATA "bar","vau","kni","vas","sig","sor","spi
","nor","sou","eas","wes","n","s","e","w","doo","w
in"
2050 DATA "go","get","loo","inv","sco","dro","hel"
,"cli","dig","qui","say","dri","ope","tak","kil","
rea","sho","att"
2055 DATA "clo","loc","unl","giv","use","val","swi
","sha","exa","thr","fin","jum"
2060 FOR i=1 TO nn: READ y$(i): NEXT i: FOR i=1 TO
 nv: READ x$(i): NEXT i
2070 DATA "a long bow","a bronze key","a leather-b
ound book","a silver arrow","a broken arrow","a gi
gantic sapphire"
2072 DATA "a vial of amber liquid","a golden sword
","a large Rembrandt painting","a whetstone","a se
t of silverware","a platinum pen","a velvet pillow
"
2074 DATA "the sorceror's sceptre","a gold bar","a
 vault in the wall","a dead knight","a ming vase",
"a sign saying 'diabolical maze'","a wicked sorcer
or","a dead spider"
2076 DATA -1,0,0,0,30,0,22,10,0,10,0,10,24,0,34,10
,18,20,19,0,25,10,27,10,44,0,0,10,0,10,0,0,0,0,9,1
0,35,0,32,0,0,0
2100 FOR i=1 TO 21: READ o$(i): NEXT i: FOR i=1 TO
 21: READ o(i,1),o(i,2): NEXT i
2110 DATA "north","south","east","west"
2120 FOR i=1 TO 4: READ t$(i): NEXT i
2150 LET p$(50)="at the end of a path,     with for
est surrounding you in  ALL directions."
2160 LET p$(47)=p$(47)+"  With a vague light to th
e south."
2270 GO TO 20
3999 STOP
4000 FOR i=1 TO 88: PRINT "DuckSoft";: NEXT i: INV
ERSE 1: PRINT AT 11,9;" HELLO THERE! ": INVERSE 0
4001 FOR i=1 TO 250: NEXT i
4002 PAPER 6: INK 0: BORDER 1: CLS
4004 PRINT "Welcome to Castlemaze adventure!"
4006 PRINT : PRINT : PRINT "Dare you explore the a
ncient": PRINT "castle?"
4008 PRINT : PRINT "Do battle with Savage Spiders
  and Sinister Sorcerors?"
4010 PRINT : PRINT "Of course you do!": PRINT : PR
INT "Hang on while I set up the data tables."
4012 RETURN
5000 INPUT LINE m$
5002 RETURN
```

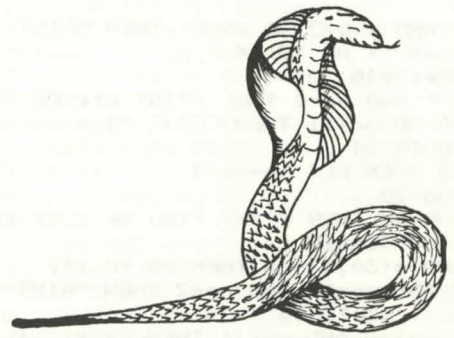

8

Tunnel Adventure

Another full-blown adventure, and again written in the same style as Castlemaze Adventure and Underground Adventure. This should serve to illustrate how easy it is to produce a large number of different games from the same basic rules.

This again is challenging, although it doesn't have the glossy edges of Underground. However it should keep you very busy trying to solve the many problems presented along the way.

Watch out for the vicious cat, and the evil hooded cobra, and the affectionate turtle encrusted with diamonds isn't all he seems either, in the ancient city of Kez!



```
  1 GO SUB 900
  2 LET bi=0: LET no=0
  4 LET w$="The Panther sees the Snake and  flees
.": LET s$="You're not holding it."
  5 LET tg=0: LET mf=0: LET m2=0: LET pd=0: LET g
f=0: LET no=0: LET zz=0: LET j=0: LET bi=0: LET mi
=0: LET t=0
  6 LET b$="You're out of matches.": LET d$="It's
 very drafty here."
  8 LET g$="A Bird swoops down out of the   sky a
nd lands in front of you."
 10 LET i$="You need a direct object."
 14 LET e$="You cannot break that."
 16 LET f$="It doesn't burn."
 18 LET o$="That is not possible."
 20 LET h$="It's pitch dark.":
 21 LET j$="The Javelin glides through the  air a
s if pulled by magic."
 22 LET l$="I can't see it here.": LET n$="Don't
be ridiculous.": LET cp=39
 24 LET m$="I don't know that word.": GO TO 446
 26 GO SUB 414
 28 IF tg THEN LET o(29,1)=cp: IF cp=36 THEN LET
tg=0
 30 IF tg THEN PRINT "The Turtle is following you
."
 32 IF tg AND cp=11 THEN GO SUB 148: CLS : PRINT
"Cave-in!!": LET p(13,4)=0: LET p(9,2)=0: LET cp=1
3
 34 GO SUB 390
 36 IF mf=1 AND m2=0 THEN LET m2=1
 38 IF vb>9 AND vb<>20 AND a$="" THEN PRINT i$: G
O TO 34
 40 IF a$<>"" AND vb=1 AND no=0 THEN PRINT "That
doesn't make much sense to me.": GO TO 34
 42 IF o(35,1) AND no=32 THEN LET no=35
 44 IF vb>10 AND no=0 THEN PRINT "What's a ";z$;"
!": GO TO 34
 45 IF no=0 AND (vb=2 OR vb=6) THEN PRINT "What A
RE you on about?": GO TO 34
 46 GO TO vb*10+40
 50 IF a$<>"" AND no=0 THEN PRINT m$: GO TO 34
 51 IF no>28 OR no<21 THEN PRINT "I don't underst
and you.": GO TO 34
 52 IF no>24 THEN LET no=no-4
 53 LET no=no-20
 54 IF no AND pd THEN PRINT "You've just fallen i
nto a pit.": GO TO 612
 55 IF no AND o(30,1)=cp THEN GO TO 142
 56 IF gf=0 AND cp=18 AND no=2 THEN PRINT "The ga
te is locked.": GO TO 34
 57 IF p(cp,no)=0 AND cp<>1 THEN PRINT o$: GO TO
34
 58 IF p(cp,no)=0 THEN PRINT "You can't go that w
ay.": GO TO 34
```



```
 59 LET cp=p(cp,no): GO TO 26
 60 IF a$="" THEN PRINT i$: GO TO 34
 61 IF o(no,1)=-1 THEN PRINT "You've already got
it!": GO TO 34
 62 IF no=0 THEN PRINT m$: GO TO 34
 63 IF cp=18 AND no=31 THEN GO TO 68
 64 IF no<>37 OR cp<>29 THEN GO TO 67
 65 IF o(17,1)+1 THEN PRINT "You need a container
.": GO TO 34
 66 LET o(17,1)=0: GO TO 74
 67 IF o(no,1)<>cp THEN PRINT l$: GO TO 34
 68 IF (no>18 AND no<32) OR no>49 THEN PRINT "You
 can't. It's too heavy.": GO TO 34
 69 GO TO 71
 70 GO TO 26
 71 IF no=12 THEN GO SUB 141
 72 IF zz>3 THEN PRINT "You're carrying too much.
": GO TO 34
 73 LET zz=zz+1
 74 PRINT "OK.": LET o(no,1)=-1: GO TO 34
 75 PRINT "You are carrying:": LET zz=0
 76 FOR i=1 TO lo: IF o(i,1)=-1 THEN PRINT t$(i):
 LET zz=zz+1
 77 NEXT i: IF zz=0 THEN PRINT "Not a lot!"
 78 GO TO 34
 80 GO TO 75
 90 PRINT "Points are scored by leaving    valuab
les at the mouth of the   tunnel."
 92 GO SUB 378: GO TO 34
100 IF no=0 AND a$<>"" THEN PRINT "What's a ": PR
INT z$;"?": GO TO 34
101 IF no=0 THEN PRINT "Que?": GO TO 34
102 IF o(no,1)<>-1 THEN PRINT "You haven't got a
": PRINT z$: GO TO 34
103 IF no=35 THEN LET o(35,1)=0: LET no=32
104 IF no=15 THEN PRINT "You can't.  It's stuck t
o your  hand.": GO TO 34
105 LET o(no,1)=cp: LET zz=zz-1
106 IF no=17 THEN LET o(17,1)=0: LET o(38,1)=cp:
PRINT "Crash!!": GO TO 34
107 IF o(12,1)=o(30,1) THEN PRINT w$: LET o(30,1)
=0: GO TO 34
108 PRINT "OK.": GO TO 34
110 IF cp=31 THEN PRINT "Try reading the medallio
n.": GO TO 34
112 IF cp=7 THEN PRINT "Try using prime numbers."
: GO TO 34
114 IF j=90 AND o(41,1)=0 THEN PRINT "Some music
would be nice.": GO TO 34
116 PRINT "Try examining things.": GO TO 34
120 GO SUB 378: GO TO 388
130 IF cp=43 OR cp=44 THEN LET cp=87-cp: GO TO 26
132 PRINT o$: GO TO 34
140 GO TO 60
141 PRINT "The Snake bites you.": LET bi=bi+8: RE
```

```
TURN
 142 PRINT "The Panther pounces on you!!": GO TO 6
12
 144 IF o(32,1) THEN RETURN
 145 LET o(32,1)=o(35,1): LET o(35,1)=0: RETURN
 146 FOR i=1 TO o(31,2): NEXT i: GO SUB 144: LET o
(31,2)=100: RETURN
 147 FOR i=1 TO 54: IF o(i,1)=13 THEN LET o(i,1)=5
 148 NEXT i: RETURN
 150 IF no=31 AND cp=18 THEN GO TO 154
 151 IF no=31 THEN PRINT "I see no gate here.": GO
 TO 34
 152 IF o(no,1)<>-1 THEN PRINT s$: GO TO 34
 153 PRINT "That isn't necessary.": GO TO 34
 154 IF gf=1 THEN PRINT "It's already open!!": GO
TO 34
 155 IF o(4,1)=-1 THEN LET gf=1: PRINT "The gate s
wings open.": GO TO 34
 156 PRINT "You need a key to open a locked gate."
: GO TO 34
 157 IF no<>46 OR o(46,1)<>-1 THEN GO TO 100
 158 LET o(46,1)=17: LET zz=zz-1: PRINT j$: GO TO
34
 160 PRINT "Try 'Push'.": GO TO 34
 170 IF o(no,1)<>-1 THEN PRINT s$: GO TO 34
 171 IF no<>2 AND no<>16 AND no<>18 AND no<>5 THEN
 PRINT "There's nothing written on it!": GO TO 34
 172 PRINT "It says:": PRINT : IF no<>2 THEN GO TO
 176
 173 PRINT "The hidden city of Kez has been looked
 for by many men, but nonehave yet found it.": GO
TO 34
 176 IF no=18 THEN PRINT "Even felines have enemie
s.": GO TO 34
 177 IF no=16 THEN PRINT "Fermented juice is alexi
pharmic.": GO TO 34
 178 PRINT "Take the first six letters,throwaway t
he left half, double the  middle and turn it aroun
d.": GO TO 34
 180 IF o(no,1)<>-1 THEN PRINT l$: GO TO 34
 182 IF no<>13 THEN PRINT n$: GO TO 34
 184 IF o(13,1)<>-1 THEN GO TO 194
 186 LET zz=zz-1: PRINT "Urghh!! It tastes horribl
e!": LET o(13,1)=0: GO TO 34
 190 IF o(no,1)<>-1 AND o(no,1)<>cp THEN PRINT l$:
 GO TO 34
 191 IF no<>12 AND no<>29 AND no<>30 AND no<>41 TH
EN GO TO 201
 192 IF no=30 THEN GO TO 142
 193 IF no=12 THEN GO SUB 141: GO TO 34
 194 IF o(13,1)<>-1 THEN PRINT "You have no food."
: GO TO 34
 195 IF o(29,1)<>cp THEN PRINT "What Turtle?": GO
TO 34
 196 LET zz=zz-1: LET o(13,1)=0: PRINT "The Turtle
```



```
 rubs your leg affect-ionately, and clings to you
likea limpet": LET tg=1: GO TO 34
 200 IF o(no,1)<>-1 AND o(no,1)<>cp THEN PRINT l$:
 GO TO 34
 201 IF no<>12 AND no<>41 AND no<>29 AND no<>30 AN
D no<>41 THEN PRINT "It isn't alive!": GO TO 34
 202 IF no=12 THEN GO SUB 141: GO TO 34
 203 IF no=30 THEN GO TO 142
 204 PRINT "It's immortal.": GO TO 34
 210 IF no<>22 AND no<>11 THEN GO TO 214
 212 IF cp=21 AND p(21,2)=0 THEN LET p(21,2)=9: PR
INT "You've broken through!!": GO TO 34
 214 IF o(12,1)=cp THEN GO SUB 141: GO TO 34
 216 PRINT "Nothing happens.": GO TO 34
 220 PRINT "You don't have enough charisma!": GO T
O 34
 230 PRINT "Try 'Open'": GO TO 34
 240 IF no<>51 THEN PRINT "Nothing happens.": GO T
O 34
 242 IF cp<>22 THEN PRINT "What mirror?": GO TO 34
 244 IF mi THEN LET p(22,2)=4-p(22,2): PRINT "It r
olls easily.": GO TO 34
 246 PRINT "It's stuck.": GO TO 34
 250 PRINT "Try 'Use'": GO TO 34
 260 IF no<>39 AND no<>37 THEN PRINT "Try expressi
ng that another way.": GO TO 34
 261 IF o(no,1)<>-1 THEN PRINT s$: GO TO 34
 262 IF no=39 THEN GO TO 265
 263 IF cp<>22 THEN PRINT "There's no use for oil
here.": GO TO 34
 264 LET mi=1: PRINT "The rollers are now oiled.":
 GO TO 34
 265 IF o(15,1)+1 THEN PRINT "Your nails are now n
ice and     clean!": GO TO 34
 266 LET o(15,1)=cp: LET zz=zz-1: PRINT "The Statu
ette slips from your   grasp.": GO TO 34
 270 GO TO 250
 280 IF no<>36 THEN PRINT n$: GO TO 34
 281 IF o(36,1)<>-1 THEN PRINT "You have no wine."
: GO TO 34
 282 PRINT "Glug-glug-glug!": LET o(36,1)=0: LET o
(17,1)=-1
 283 IF bi THEN PRINT "Aaaahh.  It cures the snake
bite.": LET bi=0
 284 GO TO 34
 290 PRINT e$: GO TO 34
 300 IF no<32 OR no>35 THEN PRINT o$: GO TO 34
 301 IF o(33,1)<>-1 THEN PRINT "You have no matche
s.": GO TO 34
 302 IF no=33 THEN PRINT "The matches burn brightl
y.": LET zz=zz-1: LET o(33,1)=0: GO TO 34
 303 IF no<>34 THEN GO TO 308
 304 IF o(34,1)=-1 THEN GO SUB 146: PRINT "You are
 blown to bits.": GO TO 612
 305 IF o(34,1)<>cp THEN PRINT l$: GO TO 34
```

```
 306 IF cp=13 THEN LET p(13,3)=24: LET p$(13)=p$(9
): LET cp=11: PRINT "The force of the blast leaves
 a gaping hole in the wall.": GO SUB 147
 307 LET o(34,1)=0: GO SUB 146: PRINT "The smoke g
ets everywhere!": GO TO 34
 308 IF o(35,1) THEN PRINT "It's already lit.": GO
 TO 34
 309 GO TO 314
 310 IF no<>48 THEN PRINT "Pardon?": GO TO 34
 311 IF o(48,1)<>-1 THEN PRINT s$: GO TO 34
 312 IF cp>35 THEN PRINT g$: LET o(41,1)=cp: GO TO
 34
 313 GO SUB 146: CLS : PRINT "Cave-in!!": GO TO 61
2
 314 IF o(32,1)=-1 THEN LET o(35,1)=-1: LET o(32,1
)=0: LET pd=0: GO TO 26
 315 PRINT "You have no torch.": GO TO 34
 320 IF no<>11 THEN GO TO 325
 321 IF cp=21 THEN PRINT "The south wall is badly
eroded.": GO TO 34
 322 IF cp=17 AND p(17,4)=0 THEN LET p(17,4)=34: P
RINT "You find a secret passage.": GO TO 34
 323 IF cp=34 AND p(34,4)=0 THEN LET p(34,4)=35: L
ET p(35,3)=34: PRINT "You've found a secret passge
.": GO TO 34
 324 PRINT "You find nothing special.": GO TO 34
 325 IF no=31 AND cp=18 THEN GO TO 328
 326 IF no=37 AND cp=29 THEN PRINT "It's just oil.
": GO TO 34
 327 IF o(no,1)<>cp AND o(no,1)<>-1 THEN PRINT l$:
 GO TO 34
 328 IF no=2 OR no=16 OR no=18 THEN GO TO 170
 329 GO TO 800
 330 GO TO 210
 340 IF no<>34 THEN PRINT "I don't know how to do
that.": GO TO 34
 341 IF o(3,1)=-1 AND o(6,1)=-1 AND o(14,1)=-1 THE
N GO TO 343
 342 PRINT "You haven't got all the ingred- ients.
": GO TO 34
 343 LET o(3,1)=0: LET o(6,1)=0: LET o(14,1)=0: LE
T o(34,1)=-1: LET zz=zz-2: PRINT "Done it.": GO TO
 34
 350 PRINT "Try 'Make'.": GO TO 34
 351 PRINT "Which compartment number?";: GO SUB 61
4: LET no=VAL v$: IF no=0 OR no>100 THEN GO TO 34
 352 IF no=13 AND o(49,1)=0 THEN LET o(49,1)=7: GO
 TO 355
 353 IF no=13 AND o(8,1)=0 THEN LET o(8,1)=7: GO T
O 355
 354 PRINT "That compartment is empty.": GO TO 34
 355 PRINT "Something fell out.": GO TO 34
 360 GO TO 157
 376 GO SUB 378: GO TO 388
 378 LET j=0: FOR i=1 TO lo: IF o(i,1)=36 THEN LET
```



```
 j=j+o(i,2)
 382 NEXT i
 384 PRINT "You have scored ";j;" points": PRINT "
out of one hundred.": IF j<99 THEN RETURN
 386 CLS : PRINT "Well done!":
 388 STOP
 390 PRINT "What Now  * ? ";: GO SUB 614: PRINT x$
: PRINT : IF bi>0 THEN LET bi=bi+1
 392 LET a$="": LET k$="": LET vb=0: LET no=0
 394 LET lc=LEN x$: FOR i=1 TO lc: IF x$(i TO i)<>
" " THEN LET k$=k$+x$(i TO i): NEXT i
 395 LET y$=k$: IF y$="go" THEN LET vb=1: GO TO 40
2
 396 IF LEN v$<3 THEN PRINT "I don't understand yo
u.": GO TO 34
 397 LET k$=y$(1 TO 3): FOR i=1 TO nv: IF u$(i)=k$
 THEN LET vb=i: GO TO 402
 398 NEXT i
 400 LET vb=1: LET a$=k$: GO TO 406
 402 IF (LEN y$+1)>=LEN x$ THEN LET no=0: RETURN
 404 LET a$=x$(LEN y$+2 TO LEN x$)
 405 IF LEN a$<3 THEN PRINT "You're not making sen
se.": GO TO 34
 406 LET z$=a$: LET a$=a$(1 TO 3): FOR i=1 TO nn:
IF a$=c$(i) THEN GO TO 412
 408 NEXT i
 410 LET no=0: RETURN
 412 LET no=i: RETURN
 414 CLS : IF cp=16 THEN LET t=t+1: IF t>2 THEN PR
INT d$: IF t>3 AND 1*RND<t*.1 THEN GO SUB 144: LET
 t=0
 416 IF o(35,1)+1 AND cp<35 THEN PRINT h$: LET pd=
1: RETURN
 418 PRINT "You're ";p$(cp): LET pd=0
 420 PRINT : LET x$="You can see:"+CHR$ (13)
 422 FOR i=1 TO lo: IF o(i,1)=cp THEN PRINT x$;t$(
i): LET x$=""
 424 NEXT i
 425 PRINT : PRINT
 426 LET fl=0
 428 PRINT "You can go: ": FOR i=1 TO 4: IF p(cp,i
)<>0 THEN PRINT r$(i);" ";: LET fl=1
 430 NEXT i
 432 PRINT : PRINT
 434 IF bi>12 THEN PRINT ">The bite's throbbing<"
 436 IF bi>23 THEN PRINT ">You're getting dizzy<"
 438 IF bi>34 THEN PRINT ">It's hard to breathe<":
 IF bi>42 THEN GO TO 612
 440 IF cp<>18 THEN RETURN
 442 IF gf=1 THEN PRINT "The gate is open.": RETUR
N
 444 PRINT "The gate in the grill is locked.": RET
URN
 446 LET nn=54: LET nv=32: LET p=51: LET lo=54: DI
M p$(p,80): DIM p(p,4): DIM t$(lo,30): DIM o(lo,2)
```

```
: DIM u$(nv,3): DIM c$(nn,3): DIM r$(4,5)
 448 LET p$(1)="in a storeroom. The wallsare made
of concrete."
 450 DATA 18,0,5,0,25,33,8,12,0,7,31,0
 452 LET p$(2)="in a dusty passageway."
 454 LET p$(3)="in the quarters of        Princess
Anka."
 456 LET p$(4)="in the King's Harem:      doesn't t
his sound like a fun    place to be!": DATA 22,0,0,
0,0,0,21,1,6,15,6,19
 458 LET p$(5)=p$(1)
 460 LET p$(6)="in a twisty little tunnelleading .
.. who knows where?"
 462 LET p$(7)="in a jewelry niche that  obviously
 once housed belongingsof great value.": DATA 3,0,
0,0
 464 LET p$(8)="in an old artist's studiobedecked
with painter's wares.": DATA 0,0,0,2
 466 LET p$(9)="crawling over a jumble ofbroken ro
ck.": DATA 21,10,0,0
 468 LET p$(10)=p$(6): DATA 9,6,6,6
 470 LET p$(11)="in a long low tunnel.": DATA 0,20
,13,0
 472 LET p$(12)="in an ancient library    which ha
s now run to ruin.": DATA 0,0,2,0
 474 LET p$(13)="in a long low tunnel.  A thick br
ick wall bars your path."
 476 LET p$(14)=p$(6): DATA 0,0,0,11,19,19,20,19
 478 LET p$(15)=p$(6): DATA 6,6,6,10
 480 LET p$(16)=p$(6): DATA 14,6,6,6
 482 LET p$(17)="in an old wine closet    which is
 now old with decay and ruin.": DATA 0,0,30,0
 484 LET p$(18)="in a long low tunnel.  A metal gr
ill halts your progress.": DATA 35,1,0,0
 486 LET p$(19)=p$(6): DATA 6,16,6,6
 488 LET p$(20)=p$(6): DATA 11,16,16,16
 490 LET p$(21)=p$(1): DATA 0,0,0,5
 492 LET p$(22)="in the sumptious bedroom of King
Kaleb.  The interior    decor is lavish!"
 494 LET p$(23)="in the slaves' quarters. This is
very poorly and shabbilydecorated!": DATA 0,0,0,31
,26,27,0,0,28,0,25,13
 496 LET p$(24)="at the west end of the   Temple.
 An ugly hole is in the west wall."
 498 LET p$(25)="at the eastern end of thebeautifu
l Temple.": DATA 29,2,0,24
 500 LET p$(26)="in the quarters that oncebelonged
 to the warriors.  They were a scruffy lot.": DATA
0,23,0,33
 502 LET p$(27)="in an old ramshackle     stable."
: DATA 23,0,0,0
 504 LET p$(28)="in the High Priest's     Vestry.
High Priest's vests hangeverywhere.": DATA 0,24,0,
24
 506 LET p$(29)="standing at the shrine ofthe grea
```



```
t godess Isis.": DATA 0,25,0,0
 508 LET p$(30)="standing in an old       kitchen.
 which has obviously    seen better days.": DATA 0
,0,33,17
 510 LET p$(31)="in an ante-chamber.": DATA 33,32,
22,3
 512 LET p$(32)="standing in the sacred   Throne r
oom, where none have    stood for aeons.": DATA 31
,0,0,0
 514 LET p$(33)=p$(2): DATA 2,31,26,30
 516 LET p$(34)="in an old secret compart-ment.":
DATA 0,0,17,0
 518 LET p$(35)="in a long low tunnel withdaylight
 beckoning from the     north.": DATA 36,18,0,0
 520 LET p$(36)="standing at the mouth of a long a
nd imposing tunnel.": DATA 37,35,0,0,51,36,0,0
 522 LET p$(37)="at the end of the road.  You can
see a brooding mountain in the distance."
 524 LET p$(38)="in a dense dark forest.  with tre
es around you in *ALL*  directions.": DATA 38,39,3
8,38
 526 LET p$(39)="on an old, well trodden  path, us
ed by horses long ago.": DATA 38,40,38,38,39,41,38
,38
 528 LET p$(40)="at the end of a path withforest s
urrounding you in *ALL* directions."
 530 LET p$(41)="in a dense, dark forest. There ap
pears to be light to thesouth.": DATA 40,42,38,38
 532 LET p$(42)="in the middle of a clear-ing.  To
 the south you can see arickety bridge."
 534 DATA 41,43,0,0,42,0,0,0: LET p$(43)="on the n
orth side of the bridge."
 536 LET p$(44)="standing on the west sideof the b
ridge.": DATA 0,45,0,0
 538 LET p$(45)="at a cross roads with    roads st
retching out of sight   everywhere.": DATA 44,48,4
7,46
 540 LET p$(46)="walking along the great  west roa
d.": DATA 0,0,45,46
 542 LET p$(47)="walking along the great  east roa
d.": DATA 0,0,47,45
 544 FOR i=48 TO 51: LET p$(i)="walking along the
great  south road.": NEXT i
 546 DATA 45,49,0,0,45,50,0,0,45,51,0,0,45,37,0,0
 548 FOR i=1 TO p: FOR j=1 TO 4: READ p(i,j): NEXT
 j: NEXT i
 550 DATA 28,10,37,0,1,0,0,10,28,0,5,0,4,10,0,10,1
3,0,18,10,0,0,36,0,30,0,21,0
 552 DATA 8,10,12,0,0,0,34,10,32,0,43,0
 554 DATA "an ephod","a scrap of newspaper","a keg
 of charcoal"
 556 DATA "a silver key","a parchment scoll","a ke
g of saltpetre"
 558 DATA "a platinum chastity belt","a ruby earri
ng"
```

```
 560  DATA "a green pebble","a blue stone","a wall
","a vicious cobra","a shrivelled carrot"
 562 DATA "a keg of sulphur","a jade statuette","a
n old medical book","an empty bottle"
 564 DATA "a gold medallion","a throne made of sol
id gold","a dead knight",27,10
 566 FOR i=1 TO 20: READ o(i,1): READ o(i,2): NEXT
 i
 568 FOR i=1 TO 20: READ t$(i): NEXT i
 570 DATA 31,0,0,20,35,0,0,0,0,0,0,0,17,0,0,0,0,0,
3,0,11,10,0,10,13,0,30,0,8,0
 572 DATA 23,0,26,0,27,0,32,0,0,0,29,0,22,0,7,0,22
,0,3,0
 574 DATA "a giant turtle!"
 576 FOR i=29 TO 54: READ o(i,1): READ o(i,2): NEX
T i
 578 DATA "a hungry panther","a gate","an old torc
h"
 580 DATA "some matches","three kegs of gunpowder"
,"a shining torch","a bottle of wine"
 582 DATA "a bottle of oil","some broken glass","a
 jar of nail-polish remover"
 584 DATA "some brown and pink gravel","a bird","a
 gold nugget"
 586 DATA "a wooden spoon","a block of marble","a
set of manacles","a rusty javelin"
 588 DATA "straw and dung","a brass clarion","a sa
tin ribbon","a marble font"
 590 DATA "a huge mirror on the south wall","a hun
dred small compartments"
 592 DATA "a king-sized bed","a triclinium"
 594 FOR i=29 TO 54: READ t$(i): NEXT i
 596 DATA "eph","new","cha","kev","scr","sal","bel
","ear","peb","sto","wal","cob","car","sul","sta",
"boo","bot"
 598 DATA "med","thr","kni","nor","sou","eas","wes
","n","s","e","w","tur","pan","gat","tor","mat","g
un","tor","win"
 600 DATA "oil","gla","rem","gra","bir","nug","spo
","blo","man","jav","str","cla","rib","fon","mir",
"com","bed"
 602 DATA "tri","go","get","loo","inv","sco","dro"
,"hel","qui","cro","tak","ope","mov","rea","eat","
fee","kil","hit"
 604 DATA "cha","unl","pus","rem","use","oil","dri
","bre","lig","pla","exa","kic","mak","mix","thr"
 606 FOR i=1 TO nn: READ c$(i): NEXT i: FOR i=1 TO
 nv: READ u$(i): NEXT i
 608 DATA "north","south","east","west"
 610 FOR i=1 TO 4: READ r$(i): NEXT i
 611 GO TO 26
 612 PRINT "You are dead!": GO TO 376
 614 INPUT LINE x$
 616 RETURN
 700 CLS : PRINT "Welcome to **Tunnel Adventure**"
```



```
:
 702 PRINT : PRINT : PRINT "Are you prepared to se
arch for  the lost city of Kez?"
 704 PRINT : PRINT : PRINT "Your adventure is abou
t to begin"
 706 PRINT : PRINT : PRINT "Hang on while I read t
he data   tables ..."
 708 RETURN
 800 IF no=13 THEN PRINT "It isn't fit for human
       consumption.": GO TO 34
 801 IF no=40 THEN PRINT "It's Topaz.": GO TO 34
 802 IF no=9 THEN PRINT "It's Malachite.": GO TO 3
4
 803 IF no=41 THEN PRINT "It's made of Gold!": GO
TO 34
 804 IF no=10 THEN PRINT "It's Lapis Lazuli.": GO
TO 34
 805 IF no=42 THEN PRINT "It's Pyrite.": GO TO 34
 806 IF no=12 THEN GO SUB 141: GO TO 34
 807 IF no=30 THEN GO TO 142
 808 IF no=1 THEN PRINT "It's embroidered with Gol
d      thread!": GO TO 34
 809 IF no=15 THEN PRINT "It glistens.": GO TO 34
 810 IF no=46 AND o(46,1)=17 THEN PRINT "It's poin
ting towards the west.": GO TO 34
 811 IF no=50 THEN PRINT "It contains sacred oil."
: GO TO 34
 812 IF no=51 THEN PRINT "It's on rollers.": GO TO
 34
 813 IF no=52 THEN GO TO 351
 814 IF no=20 AND o(33,1)=0 THEN PRINT "There's so
mething in his pocket!": LET o(33,1)=43: GO TO 34
 815 IF no=20 AND o(4,1)=0 THEN PRINT "What?! Ther
e's something else!": LET o(4,1)=43: GO TO 34
 816 PRINT "It's just ";t$(no);: GO TO 34
 900 CLS : FOR i=1 TO 88: PRINT "DuckSoft";: NEXT
i: INVERSE 1: PRINT AT 11,9;" HELLO THERE! ": INVE
RSE 0
 901 BORDER 1: PAPER 6: INK 0
 902 FOR i=1 TO 250: NEXT i: GO TO 700
```

9

Further Information

Introduction

We've presented you with information on various adventures from both the U.K. and the U.S.A. over the pages of this book, but most of the games mentioned so far have been fairly old, in that they go back as far as some of the earliest microcomputers like the Apple and the Commodore PET.

In this last section we'd like to round off by going through a few currently available adventures for various microcomputers that are relatively recent, at least at the time of writing.

Some are classics, some are obviously destined to be so, and some will probably fade over the years into a delightful obscurity and never be heard of again.

The rest of this chapter will give you some useful information on where to find out more about adventures generally, as well as listing a number of popular newstand magazines that do sometimes carry features about this sort of game.

Finally, a few useful names and addresses, and especially for those of you who own Commodore kit and want to acquire a copy of the legendary Adventure by Crowther and Woods that has featured prominently in this book, the name and address of the person to contact at the Independent Commodore Products Users' Group.

For owners of other machines, it's worth asking around to see if a copy exists for your particular machine, but if you haven't got disk

drives, forget it! This game relies almost entirely on a disk-based mode of operation, and would require an awful lot of memory before it would function on a micro that was sans disks.

That's all for now, except to say thanks to a few people. Obviously Crowther and Woods, but also Jim Butterfield, for producing the orginal PET version, and to Steve Darnold, for inadvertently getting me started on this whole adventure writing lark in the first place, and who provided the original code for Castlemaze Adventure and Tunnel Adventure.

Current Adventure Games

All the names and addresses of the companies involved can be found in most of the current popular magazines, as most of them seem to advertise quite extensively.

If not, a copy of *Personal Computer News,* the (at the moment!) 50 pence weekly, has a tri-weekly round up of software available, and covers most of the adventure games around.

So, to get the ball rolling, how about The Hobbit, which must rank as one of the classic modern games of adventure, which is available from Melbourne House for the 48K Spectrum.

A complete solving of this would take a very long time indeed, and I've yet to hear of anyone who has actually solved the entire thing. A nice style of entering your commands here as well.

PIMania seems to be the other game currently 'in vogue' as it were, although I think I'd like it a lot better if it wasn't for the inept advertising by the company who handle it, namely Automata UK. Are they really trying to produce the worst advertising in the microcomputer industry?!

Still, at least the game is good, and has the virtue of working on the Spectrum, Dragon and BBC.

Sphinx, for the BBC model B, from John Wiley and Sons is also quite a good, classical adventure, involving all the usual thud and blunder techniques beloved by writers of this particular type of adventure.

John Wiley also do a few more for the model B as well, so they're worth checking out if you're tuned into Auntie Beeb.

Microdeal have inevitably produced a series of adventures for the Dragon, including Escape, Flipper, and Mansion Adventure, or at least they call them adventure games. Personally the only one I thought was of lasting interest was the Mansion Adventure, but then we all have our different tastes.

For the Commodore 64, well, Romik have produced a couple of games, and modesty prevents me from telling you how wonderful they are, but I would like to thank Kevin Bergin for some last minute programming on those!

And the Vic 20 ? Well, there are always the cartridge versions of the Scott Adams games, and Kayde Electronics have produced the Swamp (...In the Swamp, no one can hear you scream..., runs the advertising. Yawn...), although it, not suprisingly, requires a minimum of 16K expansion.

Those are just some, but any periodical should give you details of many more.

More Information

Strangely enough, the general magazines don't appear to have picked up too strongly on this resurgence of interest in adventures, although *Personal Computer News* regularly carries a number of reviews for all kinds of machines, and most of the others mention them every now and again.

However, there are three classic issues of old magazines which the serious adventure freak must have.

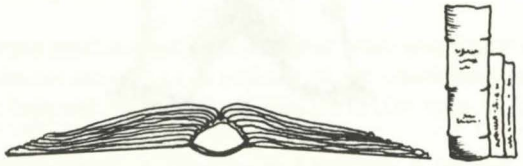
The December 1980 issue of *Byte* magazine, the one Daley Thompson does weight training with, is mainly devoted to adventuring, and features a whole host of excellent articles by many of the top authors around at the time, including Scott Adams, P. Lebling, Bob Liddell, and many more. A great issue, if you can dig it out.

The other two are different issues of the same magazine, but finding them is not going to be easy.

The magazine in question is *Creative Computing*, and the first major article appeared in August 1979, when the data structure behind the Scott Adams series of adventures was explained in full. This has inspired a number of people to begin writing their own adventures, including David Malmberg, who went on to write the very good Castle Adventure (the one with the sleepy piranha in it that I mentioned earlier!).

July 1980 was another good issue, including the article that explained the working of the program Zork, in the excellent 'How to fit a large program into a small computer'.

All required reading for the serious adventure fan, but keep your eyes on the newstands for other, newer issues of magazines.



Who to Contact

User Groups are the people to contact, and the following covers most of the popular makes of home computers.

BBC:
Laserbug
Paul Barbour
10 Dawley Ride
Colnbrook
Slough
Berkshire

or
Beebug
Sheridan Williams/David Graham
P.O. Box 50
St. Albans
Hertfordshire

Dragon:
Brixham Dragon Owners Club
Ian Chipperfield
22 Brookdale Court
Brixham
Devon

Commodore:
ICPUG
Mick Ryan
Riverhead
154 Chesterfield Drive
Sevenoaks
Kent

Spectrum:
Sinclair User Group
Irving Brand
Polytechnic of North London
Holloway Road
London N7

Writing to the appropriate address for your machine should produce the desired response.

Index

This index usually only shows the first appearance of a subject in the book, but if a second (and subsequent) entry is important, it is also noted down.

Duckworth Home Computing

EXPLORING ADVENTURES ON THE SPECTRUM 48K
by Peter Gerrard

This is a complete look at the fabulous world of Adventure Games for the Spectrum 48K Computer. Starting with an introduction to adventures, and their early history, it takes you gently through the basic programming necessary on the Spectrum before you can start writing your own games.

Inputting information, room mapping, movement, vocabulary – everything required to write an adventure game is explored in detail. There follow a number of adventure scenarios, just to get you started, and finally three complete listings written specially for the Spectrum, which will send you off into wonderful worlds where almost anything can happen.

The three games listed in this book are available on one cassette.

Duckworth
The Old Piano Factory
43 Gloucester Crescent, London NW1

ISBN 0 7156 1796 6

IN UK ONLY £6.95 NET