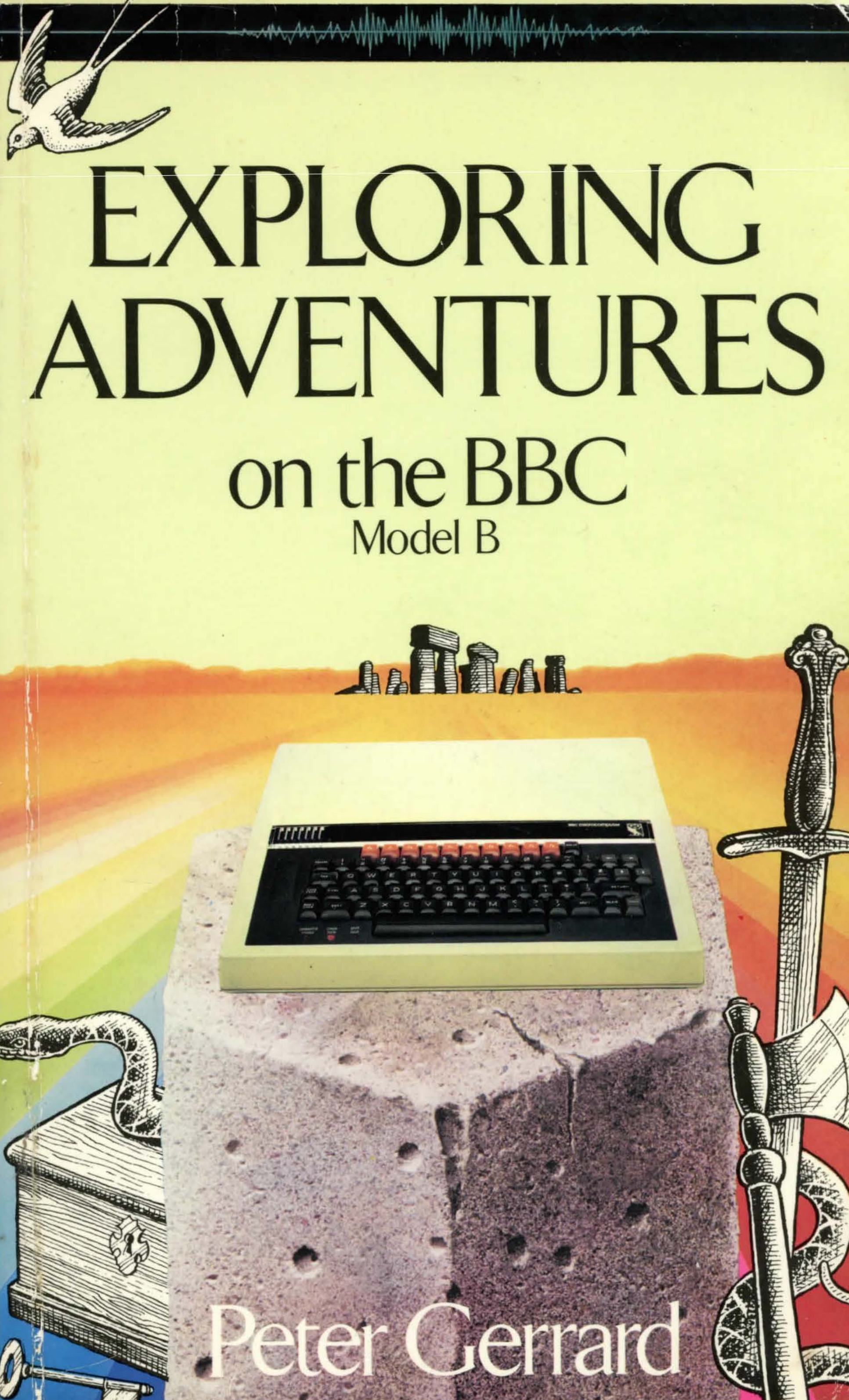
EXPLORING ADVENTURES

on the BBC
Model B



Peter Gerrard

Gerry Sexton

OMudge

Exploring Adventures
on the BBC Model B

Exploring Adventures on the BBC Model B

Peter Gerrard

Duckworth

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

©1984 by Peter Gerrard

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

ISBN 0 7156 1795 8

British Library Cataloguing in Publication Data

Gerrard, Peter
Exploring adventures on the BBC (Model B).
(Duckworth home computing)
1. Electronic games 2. BBC Microcomputer
I. Title
794.8'028'5425 GV1469.2

ISBN 0-7156-1795-8

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 vii

1. An Introduction to Adventure Games 1
2. How to Solve Adventures 13
3. Programming Adventures in Basic 47
4. Writing Your Own Adventures 81
5. Creating Your Own Adventures 115
6. Underground Adventure 135
7. Castlemaze Adventure 211
8. Tunnel Adventure 221
9. Further Information 233

Index 239

Introduction

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

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 your computer.

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 your computer, 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 your computer!

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 computer, 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 reappeared 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.

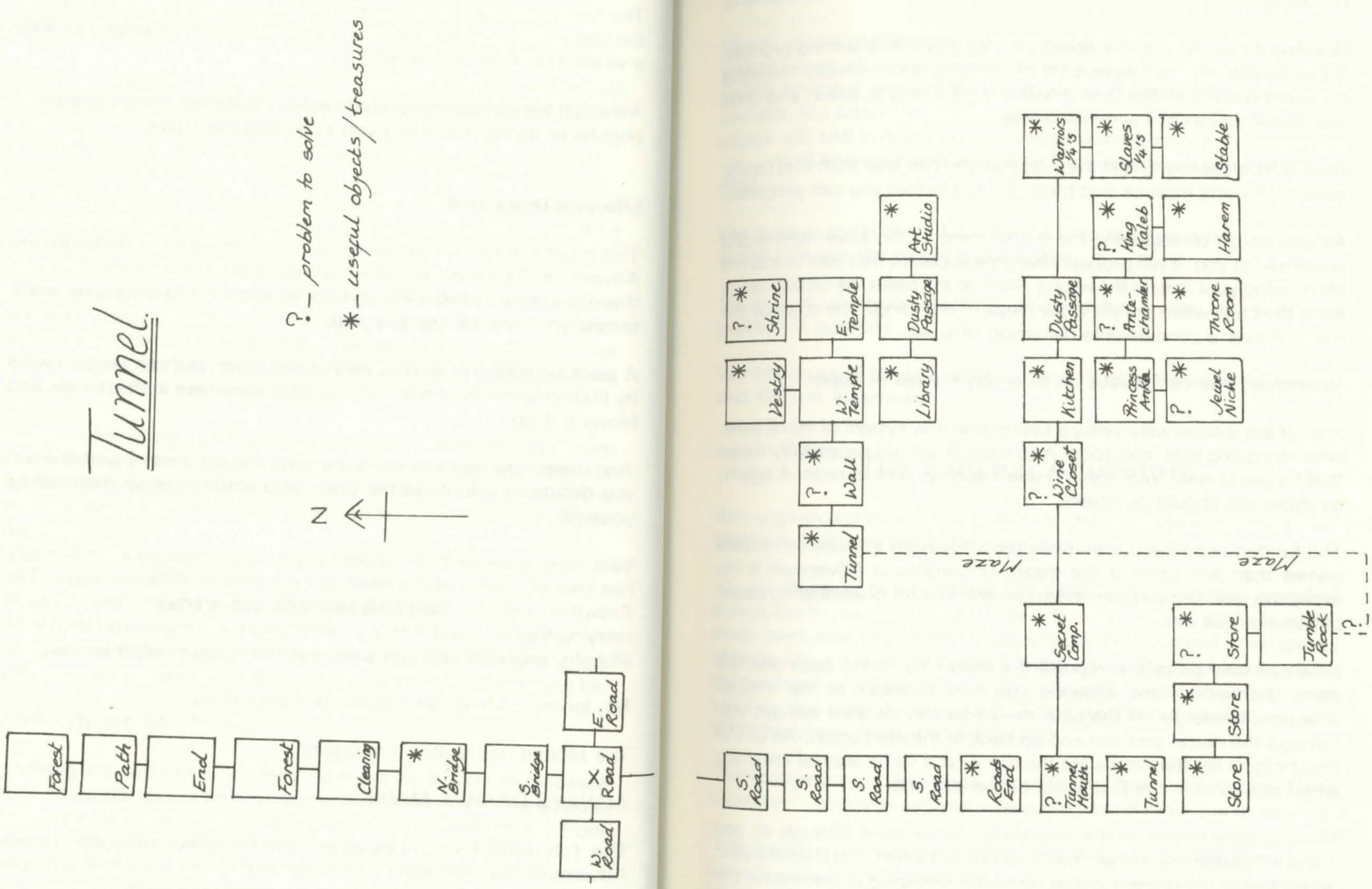
Vast chasms are another popular feature, and Underground Adventure has two of them, which need to be solved in different ways. The Crowther and Woods game also employs a chasm, and if you're carrying the black rod with you when you encounter it you should be all right, provided you can work out the correct 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
N
Forest
Path
End
Forest
Clearing
* N. Bridge
S. Bridge
X Road
E. Road
W. Road
S. Road
S. Road
S. Road
S. Road
* Roads End
? * Tunnel Mouth
* Tunnel
* Store
* Store
? * Store
Jumble Rock
?
* Secret Comp.
Maze
Maze
? * Wine Closet
* Kitchen
? * Dusty Passage
* Warriors 1/4's
* Slaves 1/4's
* Stable
? * King Kaleb
* Harem
? * Ante-chamber
* Throne Room
* Princess Anka
? * Jewel Niche
Dusty Passage
* Library
* Art Studio
E. Temple
W. Temple
* Vestry
? * Shrine
? * Wall
* Tunnel

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 something 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 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.

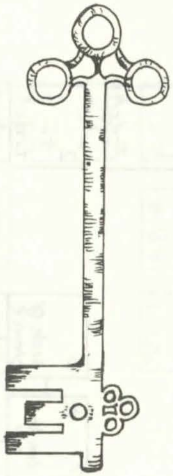
Back across the bridge, chop the tree down, and we find some rope, which must come in useful somewhere for climbing up or down something, and a golden bear, who appears none too friendly. Obviously the bear must be calmed down somehow, but how ?

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.

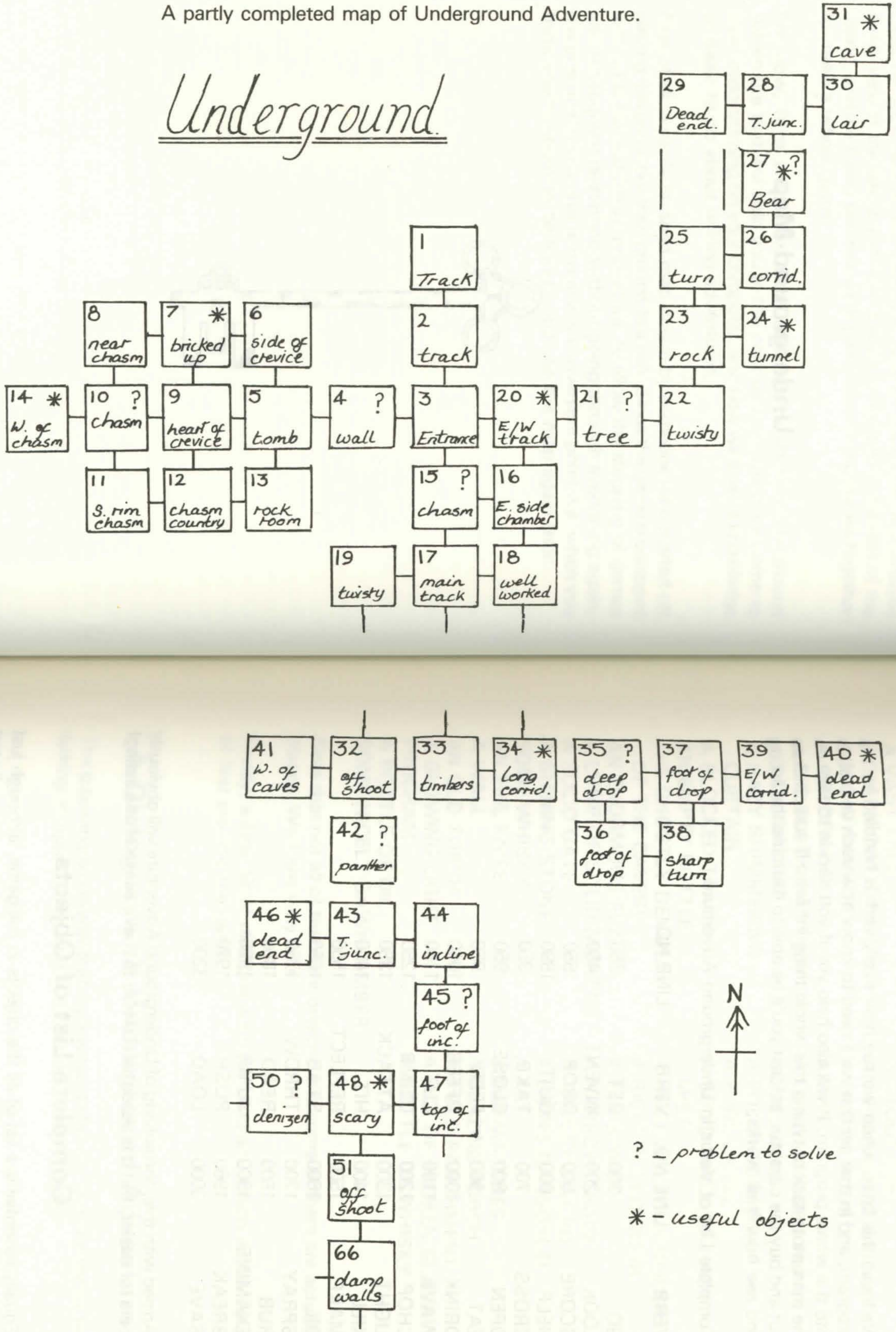
First of all, a partly finished map drawn by someone who started off playing this game but then ground to a halt. Since the game is equipped with a LOAD and SAVE procedure, to allow the stopping of games and subsequent re-starting without going through everything again, it is possible to stop this game 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.

Underground



31 * cave
30 lair
29 Dead end.
28 T.junc.
27 *? Bear
25 turn
26 corrid.
23 rock
24 * tunnel
1 Track
2 track
8 near chasm
7 * bricked up
6 side of crevice
14 * W. of chasm
10 ? chasm
9 heart of crevice
5 tomb
4 ? wall
3 Entrance
20 * E/W track
21 ? tree
22 twisty
11 S. rim chasm
12 chasm country
13 rock room
15 ? chasm
16 E. side chamber
19 twisty
17 main track
18 well worked
41 W. of caves
32 off shoot
33 timbers
34 * Long corrid.
35 ? deep drop
37 foot of drop
39 E/W corrid.
40 * dead end
42 ? panther
36 foot of drop
38 sharp turn
46 * dead end
43 T. junc.
44 incline
45 ? foot of inc.
50 ? denizen
48 * scary
47 top of inc.
51 off shoot
66 damp walls
N
? – problem to solve
* – useful objects

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

A VAST CHASM
A VAST TREE
A THICK COIL OF ROPE
SOME DYNAMITE!
A GOLDEN BEAR
A BIG BLACK PANTHER
A TALL LADDER
A HAZY SHIMMERING CURTAIN
A BLOCKED TRACK
AN EMPTY BOTTLE
THE GHOSTLY DENIZEN OF THE CAVES!
AN ENORMOUS FLY!
A LUMP OF SOLID MORTAR
A SOLID GATE
A SHINING STONE
SOME WHISKY
AN EVIL KNIFE
A WALL
AN OLD TORCH
A GLOWING LIGHT
PROGRAM
A BOTTLE OF OIL
SOME NICELY SAWN TIMBER

OBJECT

AN IRON STAFF
A STOUT AXE
AN ENCHANTED BRIDGE
A PILE OF RUBBLE
A BUN
A LONG WOODEN PLANK
SOME NAILS
A POLISHED MIRROR
A POOL OF OIL
A SOLID WALL OF HAZY MIST
A HUGE BULBOUS SPIDER
A RICKETY OLD DOOR
A FLY SPRAY!
A NARROW CRACK
A TRUSTY SWORD
A LIVING GARGOYLE!
A KEY
SOME MATCHES
A BLAZING TORCH
AN OLD PARCHMENT
A PILE OF BROKEN GLASS
A BOTTLE OF WHISKY

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 somewhere else within that program.

Using Acorn's version of Basic there are a number of ways in which we can input information into the computer.

One of them, and certainly the most straightforward, is the INPUT command, which you would use something like this:

```
10 INPUT A$
20 PRINT A$
```

This would just print a question mark on the screen and wait for you to type something in. If your reply was FRED,BILL the computer would forget the second word (the one after the comma) and print out the word FRED.

However, if you'd typed "FRED,BILL" the computer would have responded by printing out FRED,BILL.

INPUT can also have a statement in it, as in the following example:

```
10 INPUT "HELLO THERE, WHAT'S YOUR NAME",A$
```

This would print up on the screen the words 'HELLO THERE, WHAT'S YOUR NAME' followed by the question mark prompt.

INPUT, however, has a major failing, and that is that it is all too easy just to press the RETURN key and thus input nothing. So, in the adventure games presented here we've used another of the Acorn commands, GET$.

This is used in the form:

```
10 A$=GET$
```

Here, when you press a key, the computer will store the result of that key press in the variable A$.

As we shall see later, using GET$ in this way allows us to keep a much more careful track of what the player is typing in, and by using a simple

*FX call we can disable the movement of the arrow and the copy keys, and check for only numeric and alphabetic keys. *FX4,1 disables the arrow and copy keys, and treats them simply as ordinary keys. Thus, by checking for the ASCII values of whatever keys are pressed, we can selectively accept or reject various keys.

Everything that the player types in will be stored as a variable, so without further ado, let's take a look at 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 three different types of variables: real, string and integer.

Real variables are just numbers, either integer or fractional, and any of the following is a legal syntax for a variable NAME:

A, BB, CcC, DOC2LE, GARgoyLE, 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. What you can't use are punctuation marks, or names that have BASIC keywords in them, such as TO, OR, and so on. On the plus side, the computer will distinguish between upper and lower case letters, so that UPPER and upper are regarded as two different variables.

String variables can be numbers, letters, or a mixture of both, and the restrictions on string variable names are the same as for real variables, with one addition.

String variable names must end with a dollar '$' sign. Thus all of the following are legal string variable names:

A$, A1$, ZZ$, fred$, etc.

Integer variables are again numbers, but this time without the decimal part attached.

Thus an example of a real variable might be A = 12.5, but the integer counterpart would be A% = 12. All integer variable names must end with the '%' sign.

Another couple of rules to remember about integer variables: the 26 variables from A% to Z% are held as resident integer variables, that is they are not cleared when a program is RUN, NEWed, or broken into with the break key, and finally, other integer variable names such as Fred% are lost when one of the above three actions is performed.

The following would all be acceptable variables of their own individual types:

A = 26.45, A% = 54, A$ = "I'M A STRING VARIABLE", b9$ = "I AM 1 2!", etc.

There are a number of variable names, known as reserved variables, that cannot be used on your computer. A couple of these we've already seen, such as TO and OR, but there are others, including the longer reserved words such as PLOT and DRAW. The general rule is that no variable name can start with a BASIC keyword.

Thus, you couldn't use names like:

TIMES$, TOTAL, ORANGE, etc.

as they all begin with one or other of those reserved variable names. Also, logically enough, you can't use any of the reserved Basic words for variables either.

Back to Input

Input allows you to type some information into the computer from a program, and that information is stored as a variable. To go into more detail than earlier, 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 be referred to as 1.45! That's because we specified that we wanted a string to be input (A$). Try the following:

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

If you'd typed in your age as FRED, the computer would have responded with THAT MEANS YOU'RE 0 DAYS OLD, since it would treat the word FRED as a variable with a value of 0.

If you press Return, and nothing else, the program will continue, but it will treat the string as a null one, i.e. one that contains nothing.

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.

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 40
30 B=B+A
35 GOTO 20
40 C=B/10
```

```
45 PRINT "THE TOTAL IS ":B
50 PRINT "AND THE AVERAGE NUMBER READ WAS ";C
60 END
70 DATA 1,2,4,8,16,32,64,128,256,512,0
```

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 real numbers, that's what the data statements must contain. If you're reading strings, again they must contain strings. Otherwise all kinds of very strange things will begin to happen.

What you must remember is that data is read as it is encountered, so wherever it happens to be in the program, make sure that it corresponds to what you want to read.

Also, make sure that you don't try to read more data than you've actually typed in, otherwise the computer will merrily try to read data that isn't there, and again an undesirable chaos will ensue.

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.

One common fault when typing in listings of this length, which contain a lot of data statements, is using a full stop instead of a comma i.e. typing 0.5 instead of 0,5. One data item instead of two!

Again, it is very easy to miss a number out altogether, so do take care.

If you try to read the same data again, another 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 GOTO 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.

There is another option with the RESTORE command, and that is that you can specify a line to restore the data to. For instance, RESTORE 2000, will set the data pointer to line 2000. If no line number is specified, the data pointer goes right back to the start of all the data.

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

```
10 CLS
20 READ A$,B,C,D
25 IF A$=0 THEN END
30 PRINT "[CD]";A$;" IS ";B;" YEARS, ";C;" MONTHS
 AND  ";D;" DAYS OLD!"
40 GOTO20
50 DATA PETE,26,1,21
60 DATA BERYL,25,5,18
70 DATA 0,0,0,0
```

When run, this will print a couple of things up on the screen, until it reaches the data statement in line 70. This puts a value of 0 into A$, and according to the check in line 25, the program will then stop.

GET$, IF, THEN and ELSE

We'll confine ourselves to using the keyboard, where we find that GET$ allows us to input one character at a time, without the need to keep pressing the RETURN key.

The following program will illustrate this point:

```
10 CLS : PRINT "PRESS ANY KEY"
20 A$=GET$
30 PRINT "YOU PRESSED ";A$;"!"
40 GOTO 20
```

A number of new ideas here.

Line 20 simply sits and waits until something has been pressed on the keyboard. When it has, we fall through to line 30 and print out whatever key it was that was 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 either the ESCAPE key or 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 A$=GET$
105 IF A$<>" " THEN 100
110 carry on ....
```

Here, if A$ is not equal to (the < and > keys together) 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 A$=GET$
110 IFA$="Y"THEN goto another bit of the program.
120 IFA$="N"THEN go somewhere else.
130 GOTO 100
```

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

Or how about this:

```
100 A$=GET$
110 IF A$="Y" THEN 200 ELSE IF A$="N" THEN 300.
120 GOTO 100
```

Here, we sit and wait till a key is pressed. If 'Y' is pressed then we branch to 200, if 'N' is pressed then we branch to 300, but otherwise we go back to 100. 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 ... ELSE statement.

ELSE, as you might gather, is an extension to the scene, as we saw in the last example. Here, IF something is true THEN we do something ELSE we do something different. Like this:

```
10 IF A = 10 THEN 200 ELSE 300
```

So, if A was equal to 10 we'd go to line 200. If it wasn't, the ELSE would be acted on, and the program would branch to line 300.

The other main role for these commands 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 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.

One could also use REPEAT ... UNTIL in a similar way, and we'll take a look at this in a moment.

Subroutines and Procedures

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 A=B+C
20 D=E+F
30 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.

However, you'll note that there are no REM commands in any of the adventures listed in this book.

This is due to the prodigious amount of memory that they take up. If any REMs had been used, the programs probably wouldn't have fitted in!

This should serve to illustrate why it isn't really very good practice to start a subroutine or a procedure with a REM statement. If you decide later on that, because of the amount of memory being taken up, the REM statement is no longer required, when the program is executed it will still attempt to go to the line that was previously REMmed. Finding it not there, it will understandably get a bit miffed, so try to avoid that.

However, given that memory is not at a premium, the use of REM statements can go a long way towards achieving a program listing which is easy to follow when you come back to it after five months of doing something else!

Before continuing with subroutines and procedures, since the majority of such things in these adventure listings rely heavily on the use and manipulation of strings, let's take a side-step for a moment and start with:

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$="ANOTHER STRING"
20 B=LEN(A$)
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$.

Finally, you can take the length of strings other than variables, as in:

PRINT LEN ("ADVENTURER")

LEN comes into its own when taken in conjunction with the next set of string commands.

MID$

This is the most flexible of all the string handling commands, and is taken first because it's probably the one that you'll use most often.

Strings can be manipulated in many ways. As we've seen, they can be added up (more correctly termed 'concatenated'), they can be compared with each other, but MID$ opens up a whole new field.

The command takes the following syntax:

MID$(A$,I,J)

Let us take a typical example.

We'll assign the string A$ to be equal to the name of my home county, Lancashire. So, if we say A$ = "LANCASHIRE", A$ becomes a string of length 10 characters.

The command MID$(A$,I,J) takes the string A$, starts at the Ith character in that string, and takes J characters out of it.

To give a programming example:

```
10 A$="LANCASHIRE"
20 PRINT MID$(A$.4,4)
```

When run, this would print out the new string CASH: A$ is unaffected.

As with LEN, this can also be assigned to another variable. For instance:

```
10 A$="LANCASHIRE"
20 B$=MID$(A$,7,4)
30 PRINT B$
```

would result in the string HIRE being printed out, this being the value now stored in B$.

There is one other way in which MID$ can be used, and this is to take all the characters in a string, starting from a specified point. That is, MID$(A$,I), would start at the Ith character, and take all the remaining ones.

Thus, with our string A$ = "LANCASHIRE", the command:

PRINT MID$(A$,6)

would print out the word SHIRE.

Associated with MID$ are two other string handling commands that are used extensively in our adventures.

LEFT$

Not as flexible as MID$, but none the less a command with its uses when handling strings, is LEFT$.

It is a fairly safe bet to assume that this has something to do with the left-hand side of a string, and indeed it does.

Sticking with counties, we'll assign the string A$ to equal "DEVON".

When we issue the following command:

PRINT LEFT$(A$,4)

the result is printed to the screen as DEVO. Thus, with LEFT$ we always start at the first character in the string, and take as many characters as indicated in the argument.

So, in the following program:

```
10 A$="DEVON"
20 B$=LEFT$(A$,3)
30 PRINT B$
```

we would get the rather strange word DEV being printed out.

As you can see, not as powerful as MID$, but not without its uses.

RIGHT$

Well, you'd never guess would you?

RIGHT$ is concerned with the right-hand side of a string, and works in pretty much the same way as LEFT$.

Thus, if we assign the string A$="KENT", the command:

PRINT RIGHT$(A$,3)

would print out the word ENT, a suitably ADVENTUREous word.

As before, other variables can be assigned using this same command.

For example, the following program will define the variable B$:

```
10 A$="CORNWALL"
20 B$=RIGHT$(A$,7)
30 PRINT B$
```

and print it out as ORNWALL.

Of course, all these commands can be combined in many ways, to make manipulation of strings very easy.

Take the following short program:

```
10 A$="PETER GERRARD"
20 B$=LEFT$(A$,6)
30 C$=MID$(A$,4,5)
40 D$=RIGHT$(A$,7)
50 PRINT B$;C$;D$
```

When run, this would print out:

PETER ER GE GERRARD

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

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

When run, you'll see a rather well known word being printed out.

There are much more elegant ways of doing this kind of thing, as we'll see when we encounter FOR ... NEXT and REPEAT ... UNTIL 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) 12.123.

The command:

PRINT STR$(A)

will print out the string 12.123, although the number A has remained the same.

This command is more useful when assigning variables, as the following program shows:

```
10 A=24.232425
20 A$=STR$(A)
30 PRINT A$
40 PRINT LEN(A$)
50 PRINT MID$(A$,1,2)
60 PRINT MID$(A$,4)
```

When run, this program will print out the following:

```
24.232425
9
24
232425
```

So you can see that 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$ = "88*88*", 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 A=VAL(23.23)
20 PRINTA
30 A=VAL(A)
40 PRINTA
50 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 ASC

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

ASC is short for ASCII, the American Standard Code for the Interchange of Information, although when used on most home computers it is usually anything but! Still, Acorn have stuck fairly rigidly to the standard, with just one or two little delights of their own.

To generate them on the screen the following syntax is used:

PRINT ASC("A")

which would return a value of 65, or:

PRINT ASC(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 returns the character whose ASCII code is the number, variable or expression following CHR$.

This feature is used in all our adventure listings in the INPUT procedure, where we are checking for the ASCII values of keys being pressed. Thus if we find that CHR$ 127 has been pressed, we know that the delete key is being used and so we amend things accordingly. Again, if CHR$ 13 has been pressed, someone has pressed the RETURN key, and again the program has been written to respond in the correct manner.

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

A=ASC("A")

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

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
```

This will simply 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!

The correct syntax in line 40 should have been:

```
40 NEXT I
```

as 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 on to 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 that 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. It is also very easy to generate a 'No FOR' error message if you inadvertently jump out of a FOR ... NEXT loop, so don't do it!

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 A$="ABCDEFG"
20 B$=GET$
30 FORI=1 TO LEN(A$)
40 IFB$=MID$(A$,I,1) THEN PRINTB$::GOTO20
50 NEXTI: GOTO20
```

REPEAT ... UNTIL

Operating in a similar fashion to FOR ... NEXT loops, but much more powerful, are the REPEAT ... UNTIL sequence of commands.

These operate in the following way:

```
10 REPEAT
20 X=X+1
30 PRINT X
40 UNTIL X=7
```

Thus we carry on around the loop UNTIL X is equal to 7.

By careful use of REPEAT ... UNTIL loops it is possible to speed up program execution considerably. Their power comes from being able to end a loop when a certain condition becomes true. With FOR ... NEXT loops things must carry on until the bitter end, since you mustn't jump out of them.

Both have their uses, as we'll see in the listings to come.

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.

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

If it isn't, you'll get a 'No such line' error being generated. The same applies to GOSUBs as well, so take care.

As a short example:

```
10 CLS
20 PRINT "HELLO!"
30 GOTO 20
```

When run, this just prints up hundreds of HELLO!s, until you hit the ESCAPE or 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 'No GOSUB' or a 'Too many GOSUBs' error will take place.

As with FOR ... NEXT loops you can have up to 26 subroutines in action at the same time, but no more. REPEAT ... UNTIL, incidentally, limits you to 20 at a time.

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, and an error will almost certainly occur sooner or later. It will also make for very untidy programs that are almost impossible to follow.

A few examples:

```
10 CLS
20 A=5:B=10
30 GOSUB 100
40 GOSUB 200
50 GOSUB 100
60 PRINT A,B
70 END: REM IMPORTANT, OTHERWISE PROGRAM FALLS THR
OUGH!
100 A=A*A
110 A=A+5
120 RETURN
200 B=B-1
210 RETURN
```

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

```
9059
```

Of course, one can get a lot more complicated than this, although as we've said it isn't really advised.

PROCedures

A PROCedure is a variation on a subroutine, but a much more powerful ally to the programmer - or so they say! GOSUBs allow you to find out where program control is going in terms of line numbers, whereas PROCedures allow you to find out in terms of words. You pays your money and takes your choice, but you'll find that the majority of the routines in these adventures use GOSUBs instead of PROCs.

Nevertheless, let's run through the syntax quickly. A procedure is defined by the statement:

```
10 DEF PROCinput
20 INPUT A$
30 ENDPROC
```

although it is likely that most procedures will get a little bit more complicated than this! To call up a procedure, you simply say PROCinput, and the program trundles off to that part of the program.

Provided that you've matched up names correctly, procedures are useful, in that you don't have to remember where all the subroutines live in terms of line numbers. However, as the object of this book is to teach you how to write adventure games, and not to spend hours looking through a listing for a two line procedure to move the player through a maze, we've stuck to GOSUBs and line numbers.

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 = 0 THEN 100
20 IF A = 1 THEN 200
30 IF A = 2 THEN 300
40 IF A = 3 THEN 400
50 IF A = 4 THEN 500
60 etc.
```

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

In its place we can use the ON ... GOTO command, and the similar ON ... GOSUB. As both work in the same way we'll take the former as an example, although with the latter you do have to take care over matching up RETURNs with GOSUBs.

```
10 ON A GOTO 100,200,300,400,500
```

Here, if A has the value 1, the program continues execution at line 100 onwards, a value of 2 and it goes to line 200, and so on up to a value of 4, when it goes to line 500.

A can be varied, in order to match our earlier IF ... THEN example as follows:

```
10 ON A+1 GOTO 100,200,300,400,500
```

Thus we now have an exact match of the original program, but in four fewer lines! Now, if A equals 0 program execution continues at line 100, and so on.

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

```
10 K$="ABCDE"
20 PRINT"ACTIVITY 'A' : PRESS A
30 PRINT"ACTIVITY 'B' : PRESS B
40 PRINT"ACTIVITY 'C' : PRESS C
50 PRINT"ACTIVITY 'D' : PRESS D
60 PRINT"ACTIVITY 'E' : PRESS E
70 A$=GET$
80 FORI=1 TO LEN(K$)
90 IFA$=MID$(K$,I.1) THEN 1000
100 NEXT I
110 GOTO70
1000 ON (ASC(A$)-64) GOTO 1100.1200,1300.1400,1500
1100 rest of program.
```

RaNDom INTegers

Like most of the home computers currently available, Acorn have seen fit to give us 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.

RND takes a number of forms, as follows:

RND	on its own gives a random number in the range -2147483648 to +2147483647
RND(1)	generates a random number between 0 and 0.999999
RND(0)	repeats the last number generated by RND(1)
RND(A)	generates a number between, and possibly including, 1 and A.

To start things off differently every time, we'll need to use another variation on a theme, using a command of the form:

RND(-A)

which gives the value -A and resets the random number generator to be based on A. So you could generate a random number ordinarily, use the RND(-A) option to start things off somewhere else, and then get something approaching a true random number generator.

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!

Ordinarily this command would have to be used in conjunction with the RND command, as most computers generate a fractional random number. But Acorn (with the exception of RND(0) and RND(1)) have seen fit to do away with this need, and hence we generate whole random numbers every time.

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 A=RND(10)
20 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(0), A(1), etc., up to a limit previously assigned by you, the user. Our 10 numbers, for instance, could be assigned numbers something like this:

```
10 FORI=0TO9
20 A(I)=RND(10)
30 NEXT I
```

Where now we have the ten different numbers stored in A(0), A(1) etc. up to A(9).

These numbers can then be selected at will. For example, PRINT (A(4)) will print the fifth number, or element, in our array A: remember that the first element is referenced as number 0.

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

```
40 FORI=0TO9
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 many elements you might wish to have in your array, be it 1, 10 or 1000, you must specify this before attempting to store any value in any element in that array.

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.

A useful idea, if running low on memory space, is to use something like DIM A(2), if we're only going to need to store a maximum of three numbers in the array A. This can make a lot of difference when running close to the limit of memory, so that we don't waste space with an 'Oh I might end up putting ten elements in it' attitude. Get your arrays right!

Arrays are not limited to one dimension either. You can dimension something as A(7,7) 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!

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 these 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 and 15 define one set of variables, which relate to the gate being open (GF) and door being open (DF), the presence of the bear (TB), and various other flags that change as the game progresses. PROCVAR is the routine that sets up all the variables that are used in the game, such as room descriptions, room direction data and so on. The variable CP that follows is the Character Position, and relates to the room number that you happen to be in at the time. Finally, GA determines whether our hostile gargoyle turns into a mischievous one!

Line 15 just defines a number of messages that are used later on in the program.

```
    1 MODE7
    2GF=1:DF=0:TB=0:PD=0:BR=0:SC=0:NP=0:ZZ=0:GS=0:
C$=CHR$(8):PROCVAR:CP=1:GA=0
   15PD$="It's now dark.":IM$="Can't go that way."
:GF$="The gate is shut.":D$="Going down ...":DF$="
The door is shut."
```

Moving Around

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

```
200 PROCMOVE:IFTB THENOB%(9)=TB
```

On returning from that procedure, we check for the presence of the bear (IFTB meaning 'if the variable TB has a value other than zero') and update the position of the bear if necessary.

But before we look at that, we'll jump down to line 1998 and define a few more variables:

```
 1998 DEFPROCVAR
 1999 NV=38:NN=53:P=100:LO=53:DIMP$(100),P%(P,3),O
B$(LO),OB%(LO),VB$(NV),NO$(NN)
```

This controls all our data reading which takes place in lines 2000 to 2267. 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.

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

DIM P%(P,3) = 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,2) refers to the direction EAST from room I.

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

DIM OB%(LO) = dimension the variable OB% to be equal to the number of nouns. OB% then contains the position of each object in the game, by referring to its room number. Thus OB%(I) refers to the Ith object, and if set equal to J puts the Ith object in the Jth room.

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

DIM NO$(NN) = dimension the variable NO$ to be equal to the number of nouns. NO$(I) then contains the shorthand description for the Ith noun. Thus if OB$(I) contained the string "A RICKETY OLD DOOR", NO$(I) would contain just "DOO", for door.

Now let's look at the actual room moving routine contained in lines 5000 to 5024.

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 DEFPROCMOVE:CLS:PRINT:PRINT
 5001 IFOB%(46)<>-1AND(CP>4ANDCP<100)THENPRINTPD$:
PD=1:ENDPROC
 5002 PRINT"You're ";P$(CP):PD=0
 5003 IFCP=42ANDTB=1ANDP%(42,1)=0THENPRINT"The bea
r scares the panther.":P%(42,1)=43:OB%(11)=0:P$(42
)="in a comfortable lair."
 5004 PRINT:VB$="You see : ":FORI=1TOLO:IFOB%(I)=C
P THENPRINTVB$;OB$(I):VB$="          "
 5008 NEXT:IFCP=3ANDGF=0THENPRINT:PRINTGF$
 5012PRINT:PRINT"You can go : ":PRINT:FORI=0TO3:IF
P%(CP,I)<>0THENPRINTD$(I);"  ";
 5013 NEXTI
 5015 PRINT:IFNP=1THENPROCGARG
 5016 IF(CP>20ANDCP<88)AND(RND(10)>9)THENNP=1
 5018 IFCP<>69ORP%(69,3)=70THENENDPROC
 5022 IFOB%(15)<>-1 THENPRINT"You can't pass yet."
:ENDPROC
 5024 PRINT"The mist washes away!":P%(69,3)=70:OB%
(15)=0:ZZ=ZZ-1:P$(69)="walking past an icy spot.":
OB%(20)=0:ENDPROC
```

Explanation of Routine

We'll take this line by line.

Line 5000 simply starts the procedure and clears the screen.

Line 5001 checks to see if you're holding a blazing torch (OB%(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 PD$ 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, then the bear frightens the panther and we have to change a few variables.

Line 5004 is the start of the 'You see' routine, which goes on to check if the location of any object, OB%, 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 5008 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 GF$, as defined in line 15.

Lines 5012 to 5013 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 D$, which was 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 PROCGARG at line 5999, which we'll come to later.

Line 5016 goes through a random number generation, and if that number is greater than 9 (on a scale of 1 to 10), 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 PROCGARG.

Line 5018 checks to see if you're in room 69 and if you've got past the obstacle there. If you're not there, or you are there but you've solved the problem, program execution returns from this procedure.

Lines 5022 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 5022 to 5024, 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 5024.

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.

You will see from the above that the listing for Underground Adventure is very tightly packed indeed. This is simply to make it fit into the memory of the computer, as we have a far greater number of rooms, objects and verbs than in either Castlemaze or Tunnel Adventure. Hence the room descriptions are quite short and the routines at times appear incredibly condensed. This is necessary, so don't try to put spaces everywhere if none are in the listing. You can always use LISTO7 afterwards to get a good look at it!

Checking Inputs

This routine is used to check, when playing the game, if the player is trying to do something with an object that isn't in sight, or isn't in his possession.

```
 5400 OB=1:IFOB%(NO)<>-1ANDOB%(NO)<>CP THENPRINT"I
t isn't here.":OB=0
 5402 RETURN
```

Explanation

Line 5400 - Set object flag to 1. If the object isn't in the player's possession, and it isn't in the room, then print a simple message and set the flag to 1.

Line 5402 - return from subroutine.

Now we've seen how a couple of routines work. 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 your computer, 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.

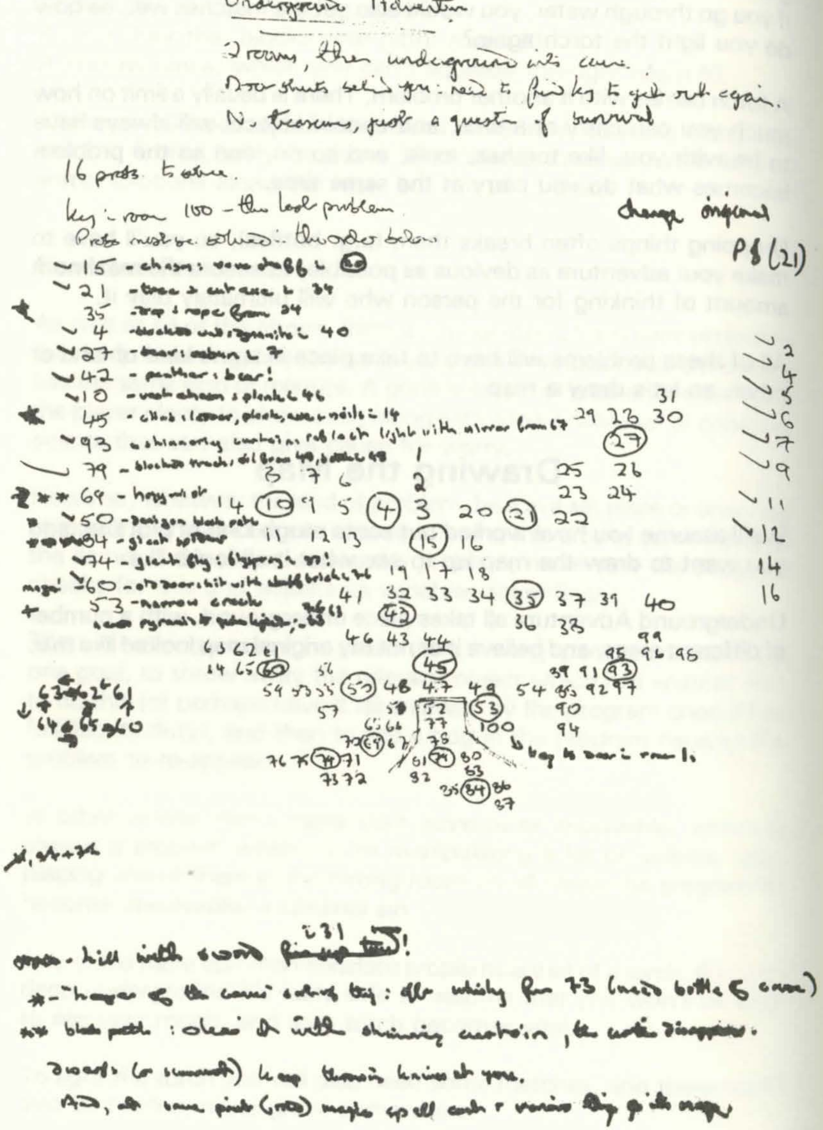
Dropping things often breaks them (e.g. bottles), so you'll have to make your adventure as devious as possible, to ensure the maximum amount of thinking for the person who will ultimately play it.

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:



Underground Adventure

3 rooms, then underground into cave.
Door shuts behind you: need to find key to get out again.
No treasures, just a question of survival

16 probs to solve.

key in room 100 – the last problem

Probs must be solved in the order below

change original

P$(21)

15 – chasm: [illegible] 86 [illegible]
21 – tree + entrance [illegible] 34
35 – trap + rope from 24
4 – blocked [illegible] dynamite [illegible] 40
27 – bear: [illegible] 7
42 – panther: bear!
10 – [illegible] chasm: plank [illegible] 46
45 – [illegible] ladder, planks, axe + nails [illegible] 14
93 – shimmering curtain [illegible] light with mirror from 67
79 – blocked [illegible] oil from 49, bottle [illegible] 48
69 – [illegible]
50 – [illegible] black path
84 – giant [illegible]
74 – giant [illegible]
60 – [illegible]
53 – narrow crack

[illegible] – kill with sword [illegible]!

[illegible] – "keeper of the caves" [illegible] key: offer whisky from 73 ([illegible] bottle [illegible])

[illegible] black path: clear it with shimmering curtain, the curtain disappears.

Dwarfs (or [illegible]) keep throwing knives at you.

[illegible], at same point ([illegible]) [illegible] spell [illegible]

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.

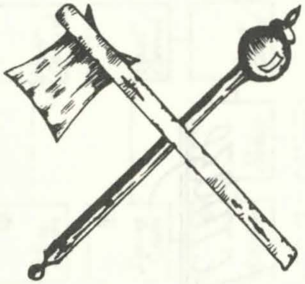
These, and other problems will all have to be thought of before we start typing anything in, but inevitably we'll have to add objects to our list as we go along developing the program, but in Basic that is no great difficulty.

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.

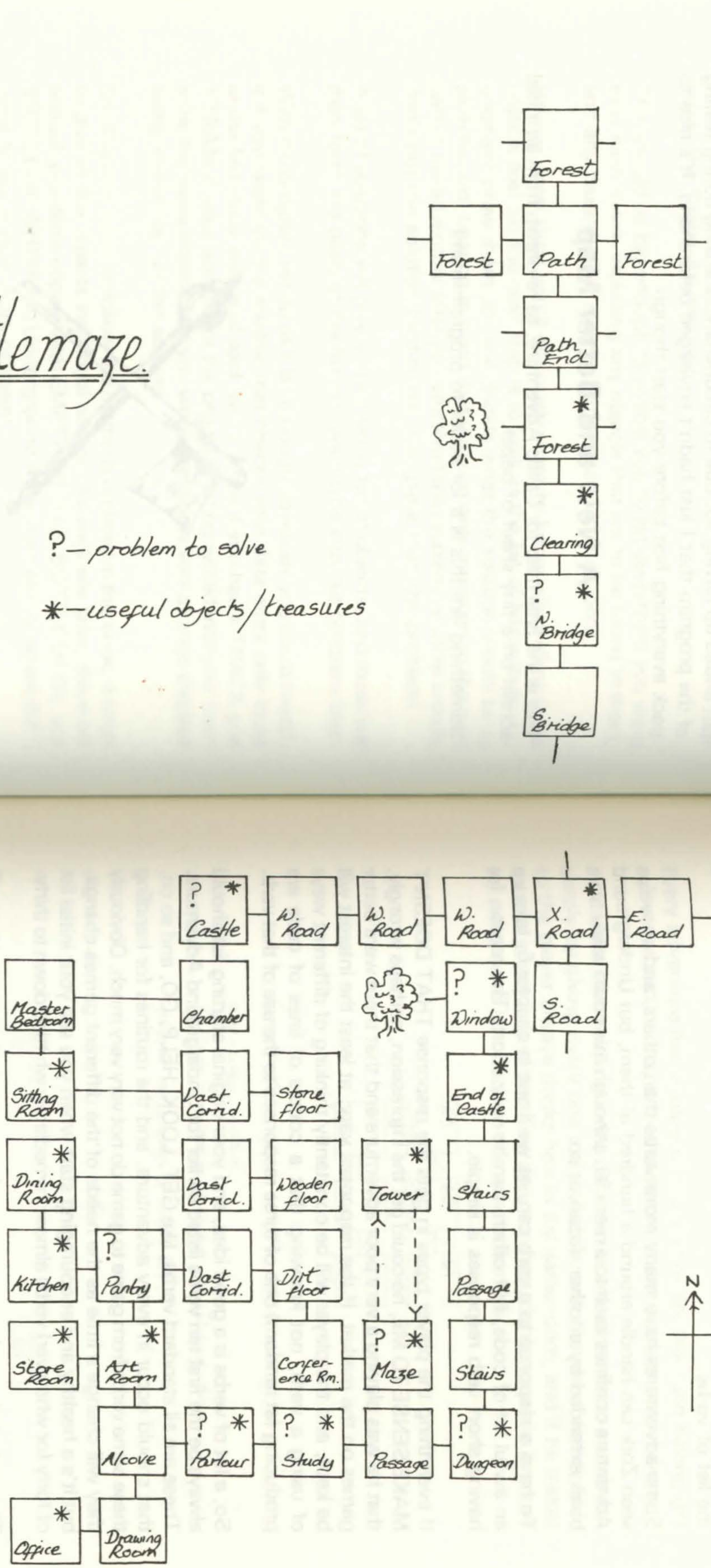
A New and Better Map

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
N
Forest
Forest
Path
Forest
Path End
✱ Forest
✱ Clearing
? ✱ N. Bridge
S. Bridge
? ✱ Castle
W. Road
W. Road
W. Road
✱ X. Road
E. Road
? ✱ Window
S. Road
✱ End of Castle
Stairs
Passage
Stairs
? ✱ Dungeon
Master Bedroom
Chamber
✱ Sitting Room
Vast Corrid.
Stone floor
✱ Dining Room
Vast Corrid.
Wooden floor
✱ Tower
✱ Kitchen
? Pantry
Vast Corrid.
Dirt floor
✱ Store Room
✱ Art Room
Conference Rm.
? ✱ Maze
Alcove
? ✱ Parlour
? ✱ Study
Passage
✱ Office
Drawing Room

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.

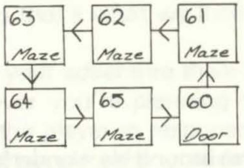
Additional verbs should also be there, just to encourage diversification of response from the computer, and keep the player's interest. A good idea is to give bizarre ideas on the part of the player equally bizarre responses from the computer.

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

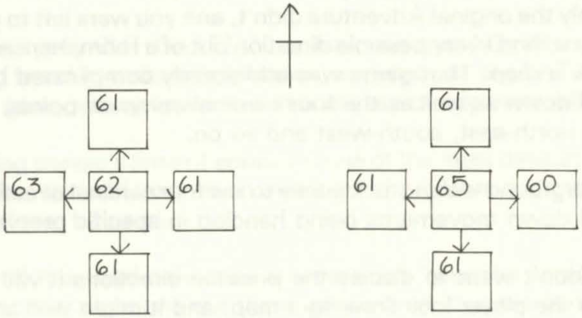
Amazing

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've not already seen, and which won't be found in the sections on verbs and data on chapter 6. So, if you're going to type the whole thing in, this is the place to look at 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 the inputting of data, and the analysing if 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 391 to 412, which we'll come to in a minute.

There's also a simple check on what you've typed in just to see if it makes any sense. This eliminates the ridiculous before going off to the appropriate routines that deal with each verb.

```
  200PROCMOVE:IFTB THENOB%(9)=CP
  208IFTB THENPRINT:PRINT"Beware the bear!"
  209IFTB ANDCP=45ANDOB%(13)=-1THENPRINT"The ladde
  snaps!":OB%(13)=0:P%(45,1)=0
  210PROCINPUT:IFVB=34THEN1950
  225IFVB>9ANDNO$=""THENPRINTV1$" what?":GOTO210
  240ONVB GOTO270,300,200,500,540,560,650,1890,690
,300,780,880,900,950,1000,1050,1100,1200,1200,1250
,1300,1400,1400,1400,1500,1550,1600,1650,1700,1750
,1800,1850,1900,0,1960,1970,3000,3200
```

Explanation

Line 200: we've already seen this, as it starts off the PROCMOVE procedure, but briefly the rest of the line checks the bear flag (TB) and if this is set puts the bear (OB%(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! Understandable, under the weight of a hefty bear. The ladder disappears (OB%(13) = 0) and the south exit from room 45 is closed (P%(45,1) = 0).

Line 210: goes to the input procedure, and on returning also checks to see if the verb number generated by that routine is equal to 34 (the verb JUMP). This special check is necessary as JUMP is the only verb not checked for in the next line that doesn't require a direct object to go with it (e.g. GET AXE, DROP TORCH). If the verb number is 34, then off to line 1950 for the jump routine.

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. In other words, you've just typed in a verb it understands, but you haven't given that verb an object to work with.

Line 240 : take the verb number and go to the appropriate line in the program by using the ON GOTO statement we explained earlier. Yes, it does look cluttered, but Acorn requires it that way. It's also quite useful, since each new line number takes up four bytes in memory, and space is at a premium!

Data Validation Routine

This checks to see what you've typed in from the subroutine in lines 30000 to 30020, which we'll get to later, and splits your input up into a verb and a noun, where applicable.

```
  391DEFPROCINPUT:PRINT:PRINT"What now ? ";:PROCIN
FO:PRINT:NO$="":VB$="":VB=0:NO=0:H=0:CM=LEN(CM$):F
ORI=1TOCM:IFMID$(CM$,I,1)=" "THEN H=I-1
  395NEXT
  396IFH=0THENH=LEN(CM$)
  397IFH=1THENV1$=CM$:GOTO 399
  398V1$=LEFT$(CM$,H)
  399VB$=LEFT$(V1$,3):FORJ=1TONV:IFVB$(J)=VB$THENV
B=J
  400NEXTJ:IFVB>0THEN406
  404VB=1:N1$=Vi$:GOTO 410
  406IFLEN(V1$)+1>LEN(CM$)THENNO=0:ENDPROC
  408N1$=RIGHT$(CM$,LEN(CM$)-1-LEN(V1$))
  410NO$=LEFT$(N1$,3):FORI=1TONN:IFNO$(I)=NO$THENN
O=I
  411NEXTI:IFNO=52THENNO=19
  412GOSUB5300:ENDPROC
```

Explanation

Line 391: print up the WHAT NOW? * prompt, and go to the procedure at 30000 to get the input of data. Then declare a few variables (length of noun, length of verb, verb number and noun number) to equal zero. Finally, perform a loop LC times, where LC is the length of the input string CM$. Carry on until you find a space in CM$, by searching through one character at a time.

Line 395: finish off loop.

Line 396: if H = 0, i.e. no spaces have been found, then put H equal to the length of the input string CM$.

Line 397: now, if H = 0, set VI$ equal to the input string, and branch on.

Line 398: set VI$ equal to the verb.

Line 399: 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.

Line 400: carry on the NV loop, because we don't recognise the verb yet. Then, a check to see if we do recognise the verb.

Line 404: 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 CM$). GOTO line 410.

Line 406: 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 procedure with a single verb.

Line 408: find the noun N1$ from the original input string CM$, by taking the RIGHT$ of CM$, starting at the character after the space.

Line 410: set No$ 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.

Line 411: continue loop because we don't recognise the noun, and when finished, if the noun number equals 52 (bottle of whisky) change it to 19 (the empty bottle) to avoid confusion over all these bottles.

Line 412: go to the subroutine at 5300 which sorts out the confusion over bottles, torches etc., and return from procedure.

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're dead!"
614PRINT"Play again?"
616 PA$=GET$
617 IFPA$="Y"THEN RUN ELSEPRINT"Bye.":END
```

Explanation

Line 612: print the 'you're dead!' message.

Line 614: ask for another game.

Line 616: wait for a key to be pressed.

Line 618: if they've typed 'Y' then run the program again, otherwise print out a goodbye, and END the program.

The Start and the End

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

```
2510 PRINT:PRINT"Well done! You're out!":END
```

Explanation

Line 2510: you're out, called up from another line in the program, in the OPEN routine in lines 788 and 799, so print a message of congratulations and end the program!

```
5200 CLS:PRINT:PRINT:PRINT"Ooops.  The gate's shu
t!":P%(3,0)=0:GF=0:GOTO210
```

Explanation

Line 5200: print message to say gate's closed behind you, called from line 277 in the GO routine, close off the north exit from room 4 (P%(3,0)), set the gate flag GF equal to zero (i.e. the gate has now closed), and return from the subroutine.

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, a bottle of whisky, a bottle of oil, or an empty bottle.

This is necessary because the data checking routine covered earlier will give the last noun that 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 IFNO=45ANDOB%(46)=-1THENNO=46:RETURN
5301 IFNO=46ANDOB%(46)<>-1THENNO=45:RETURN
5302 IFNO=19ANDOB%(51)=-1THENNO=51:RETURN
5304 IFNO=19ANDOB%(52)=-1THENNO=52:RETURN
5306 IFNO=18ANDOB%(51)=-1THENNO=51:RETURN
5308 IFNO=39ANDOB%(52)=-1THENNO=52:RETURN
5310 RETURN
```

Explanation

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

Line 5301: if the object number is for the lit torch (OB%(46)), and you're not carrying the lit torch 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 (OB%(19)), and you're carrying the bottle of oil (OB%(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 (OB%(19)), and you're carrying the bottle of whisky (OB%(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 (OB%(18)), and you're carrying the bottle of oil (OB%(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 (OB%(39)), and you're carrying the bottle of whisky (OB%(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.

```
 5999DEFPROCGARG:IFGA=1THEN6010
 6000PRINT"There's a gargoyle nearby!":IFRND(100)>
85THEN6002ELSEENDPROC
 6002PRINT"He throws a knife!":IFRND(100)>99THEN61
2
 6006 PRINT"Missed!":ENDPROC
 6010 IFRND(10)>1THENPRINT"You've got him!":OB%(40
)=0:GA=0:NP=0:GOTO6012
 6011 PRINT"Missed him!":OB%(40)=CP
 6012 OB%(4)=CP:ZZ=ZZ-1:ENDPROC
```

Explanation

Line 5999: start of procedure, and check for a mischievous gargoyle!

Line 6000: print out a hostile message, and if the random number generator gives a number greater than 85 on a scale of 1 to 100 then he throws a knife at you, otherwise just end the procedure.

Line 6002: print out message, and if the random number generated is greater than 99, in a range of 1 to 100, then you're dead, so go to the routine at line 612 onwards.

Line 6006: print out simple message that he missed, and end the procedure.

Line 6010: if the random number generated is greater than 1, on a scale of 1 to 10, then you've killed him, so jump to line 6012, set the relevant flags and remove the gargoyle (OB%(40) = 0).

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

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

The Mischievous Gargoyle

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

```
 6020 PRINT"OH NO!  A MISCHIEVIOUS GARGOYLE!":GS=0
 6022 FORI=11TOLO
 6024 IFOB%(I)=-1THENOB%(I)=RND(14)+31:GS=GS+1:PRI
NT"Har Har!"
 6026 NEXTI
 6028 IFGS=0 THENPRINT"Oooh,lucky!"
 6030 ENDPROC
```

Explanation

Line 6020: print simple message.

Line 6022: start of subroutine to check what you're carrying. Note that the gargoyle can't take any of the first ten objects, so you can always recover them when they get taken. It just makes life awkward for you.

Line 6024: if you're carrying the object then place it somewhere between rooms 32 and 45, and print a jovial message.

Line 6026: continue the loop to check what you're carrying.

Line 6028: if the counter GS hasn't been set, then nothing has been stolen, so print a simple message on the screen.

Line 6030: return from the mischievous procedure.

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.

```
 5003 IFCP=42ANDTB=1ANDP%(42,1)=0THENPRINT"The bea
r scares the panther.":P%(42,1)=43:OB%(11)=0:P$(42
)="in a comfortable lair."
```

Explanation

You've seen this one already, but basically we print an appropriate message, clear south path from room 42, remove the panther (OB%(11)), and change the room message.

```
 6300 OC=0:FORI=1TOLO:IFOB%(I)=-1THENOC=OC+1
 6302 NEXT
 6304 IFOC>1OROB%(37)<>-1THENPRINT"You can't get t
hrough.":GOTO 210
 6308 CP=100:PRINT"The stone glows eerily.":GOTO21
0
```

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, or you aren't carrying a particular object, then print a suitable message and go to line 210

Line 6308: puts you in room 100, prints message, and goes back to line 210 again.

May I Introduce You?

Owing to the prodigious demands on memory, we haven't even given Underground Adventure a title page. Not even a simple printing of the name of the game as we did for the other two listings.

However, you'll most probably want to give your games some sort of title, so it's always worth trying to leave a little space somewhere in the program! But we haven't, so let's move smartly on and look at the input routine.

Input Procedure

This all-important routine governs what can and what can't be typed in, and is also a way of stopping anyone using the arrow and copy keys to foul up the inputs.

It will allow you to delete characters only up to the input prompt, and won't allow you to press RETURN on a null prompt. If the ESCAPE key was disabled, it would prevent crashing out of the program as well.

```
30000DEFPROCINFO:CM$=""
30004PRINT"*";C$;
30006Z$=GET$
30008Z=ASC(Z$):IFZ>95ANDZ<>127THEN30006
30010ZL=LEN(CM$):IFZL>28THEN30016
30012IFZ=127THEN30018
30014IFZ>31THENCM$=CM$+Z$:PRINTZ$;:GOTO30004
30016IFZ=13ANDZL>0THENPRINT" ":ENDPROC
30018IFZ=127ANDZL>0THENCM$=LEFT$(CM$,ZL-1):PRINT"
";C$;C$;
30020GOTO 30004
```

Explanation

Line 30000: start procedure and set input CM$ to zero string.

Line 30004: print up prompt by showing a '*', and use the string C$ (defined earlier to equal CHR$(8)) to backspace over it.

Line 30006: get a character, and do nothing until something is pressed.

Line 30008: check the ASC value of the key being pressed, and if it's greater than 95 and it doesn't equal 127 (the delete key) go back for another character. This stops any unwanted characters being accepted by the computer.

Line 30010: take the length of the input string, and if it's greater than 28 then GOTO 30016, because we've had enough!

Line 30012: if the ASC value equals 127, i.e. the delete key has been pressed, then GOTO line 30018.

Line 30014: if the ASC value is greater than 31, then it's a legitimate entry. Add it to our input string, and echo it back to the screen. Then go back for another character.

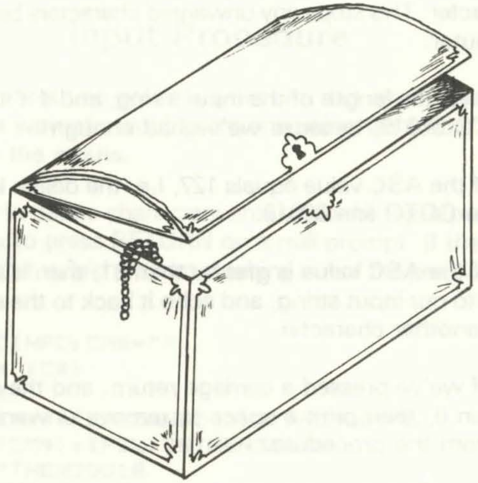
Line 30016: if we've pressed a carriage return, and the string length is greater than 0, then print a space to remove unwanted asterisks and return from the procedure.

Line 30018: if we've pressed the delete key and the string is greater than 0, then the input string becomes the left side of the string, by taking the ZL-1 first characters. Echo the character to the screen by printing a space to remove unwanted asterisks, followed by 2 CHR$(8)s.

Line 30020: go back to 30004 and start off again with the next character.

A powerful routine that could easily be adapted to trap even more characters if necessary.

Here it forms the backbone of all our input handling, and is called by the program every time some data has to be entered. So type it in correctly!



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!

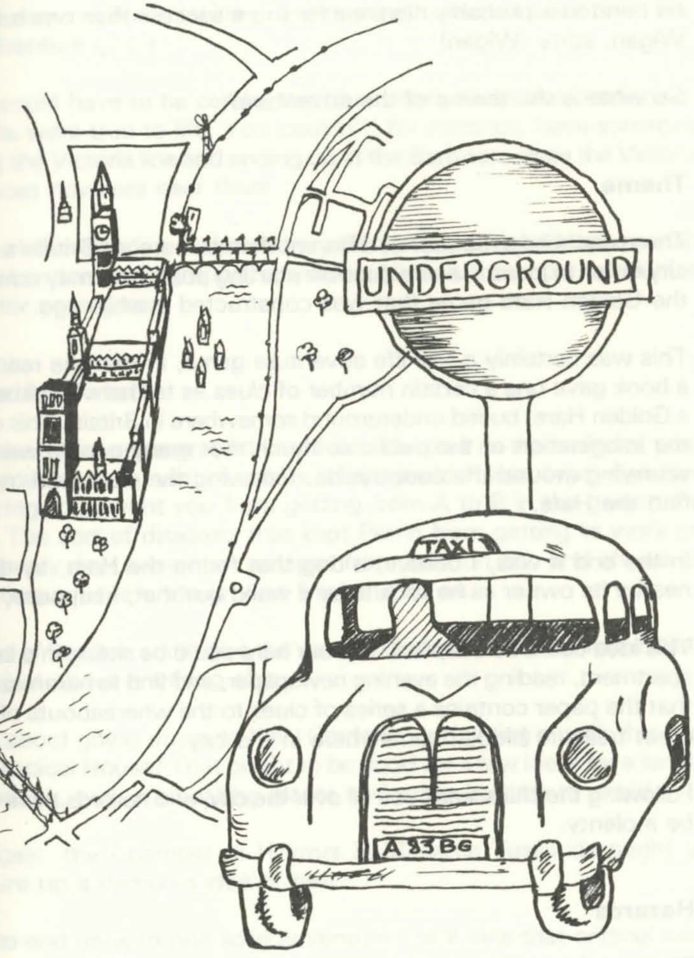
The five areas that we'll look at are all individual in their way, and none of them cross over into any of the others: they are five unique scenarios that could easily be built up into complete and enjoyable games.

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.

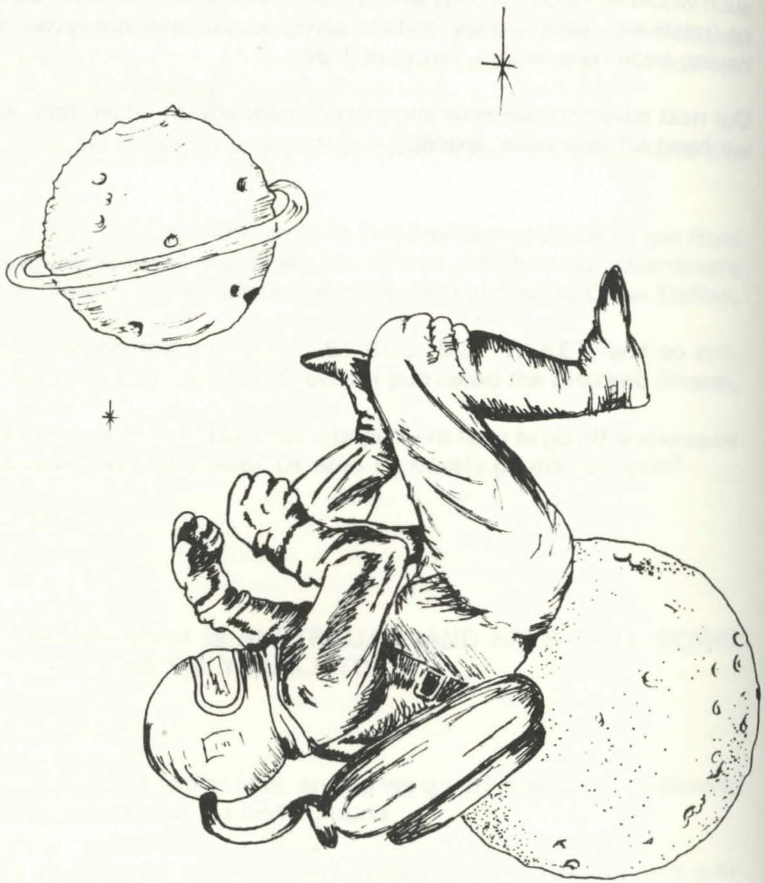
The game could encompass many famous London landmarks, each holding a clue on the trail, and each presenting its own particular problems. Big Ben would presumably feature somewhere, and, as in the famous scene in the re-make of the Thirty Nine Steps, a hazardous climb out onto the clock face could be 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

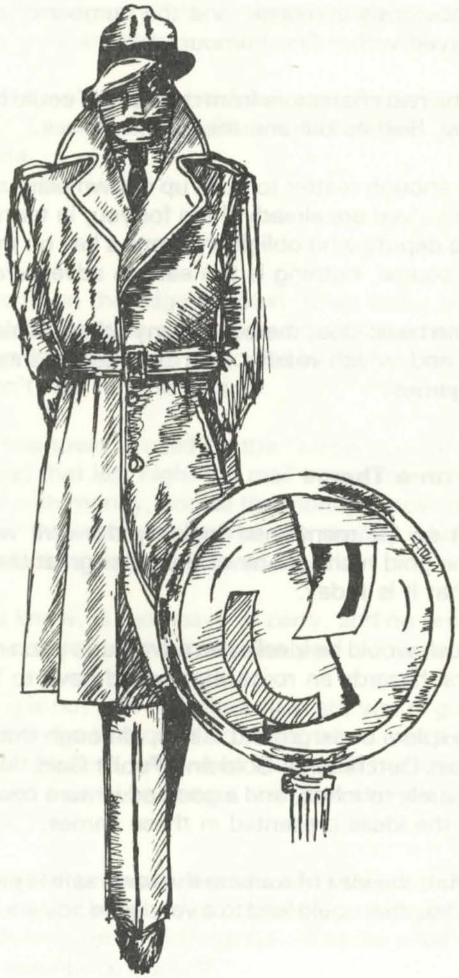
We haven't so far mentioned Indians, the civil war, the railroad pioneers, the gold rush, or any of the other great themes that made America what it is today.

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.

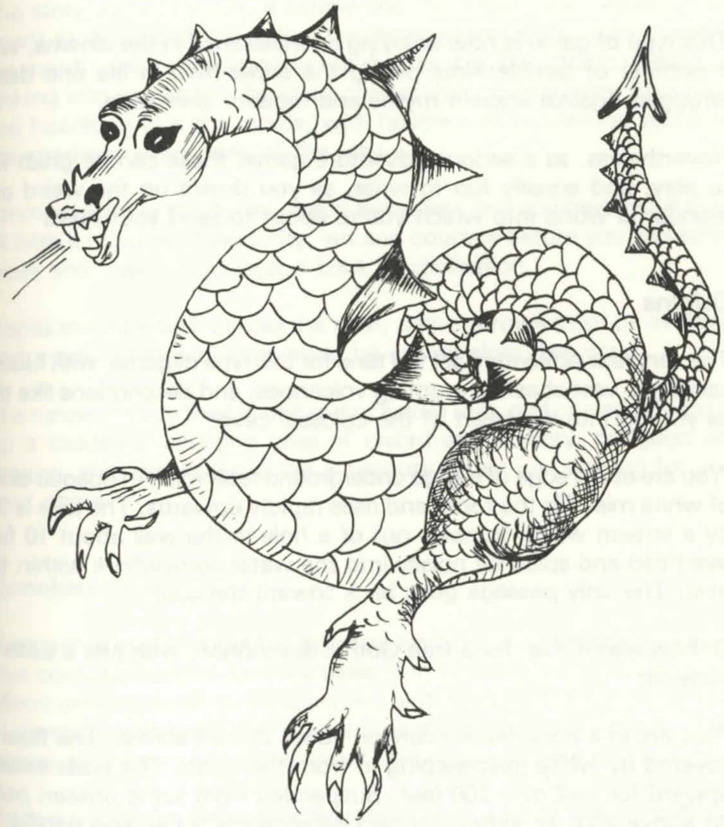
Conclusion

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.

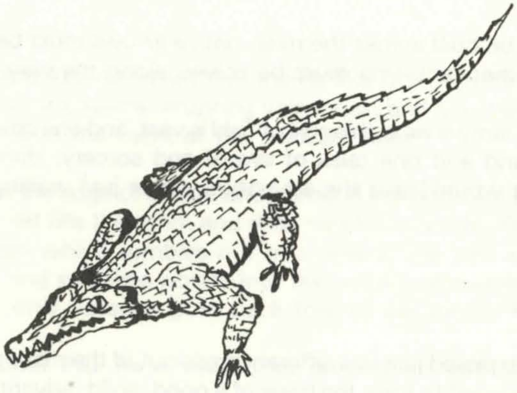
So too are the problems that must be solved, and the ease or difficulty with which the player can progress to other levels in the game, but none the less it is the story line that will initially attract a player, and start him playing your game rather than any other.

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 acquaint 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 5000 onwards, e.g.

```
5011 IF CP=24 THEN 8000
.
.
.
8000 PRINT "IN A LONG DARK TUNNEL THAT HAS BEEN CARVED
OUT OF THE ROCKS."
8002 PRINT "THE ROCKS HAVE WEATHERED OVER THE YEARS
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 program 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.

```
  270IFNO$<>""ANDNO=0THENPRINT"I don't know that w
ord.":GOTO 210
  272IFNO>28ORNO<21THENPRINT"Que?":GOTO210
  274IFNO>24THENNO=NO-4
  276NO=NO-21:IFNO ANDCP=3ANDGF=1THEN5200
  278IFNO ANDPD THENPRINT"You're in a pit!":GOTO 6
12
  288IFP%(CP,NO)=0THENPRINT"Can't do that.":GOTO 2
10
  289IFCP=53ANDNO=1THEN6300
  290CP=P%(CP,NO):GOTO200
```

Explanation

Line 270 - 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 272 - 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 274 - just adjust NO, if it's greater than 24, to lie between 21 and 25.

Line 276 - adjust NO to lie between 1 and 4, and 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 278 - if we're moving but it's pitch dark (PD is set), then print message and GOTO death routine.

Line 288 - 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 289 - if we're in room 53, and we're trying to go south, then GOTO 6300

Line 290 - 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.

```
  300IFNO=0THEN1900ELSEGOSUB5300:GOSUB5400:IFOB=0T
HEN210
  304 IFOB%(NO)=-1THENPRINT"Try inventory!":GOTO21
0
  308 IF(NO=18OR NO=39)ANDOB%(19)<>-1THENPRINT"No
container.":GOTO 210
  312 IFNO=39ANDOB%(19)=-1THENOB%(19)=0:ZZ=ZZ-1:NO
=52:GOTO389
  314 IFNO=18ANDOB%(19)=-1THENOB%(19)=0:ZZ=ZZ-1:NO
=51:GOTO389
  315 IFNO=1ORNO=3ORNO=6ORNO=8ORNO=9ORNO=11ORNO=17
ORNO=20ORNO=29ORNO=30ORNO=31ORNO=32ORNO=35ORNO=36O
RNO=40ORNO=41ORNO=43ORNO=49ORNO=50THENPRINT"You ca
n't.":GOTO210
  324 IFNO=12ANDCP=10THENP%(10,3)=0:P$(10)="faced
by a vast chasm."
  326 IFNO=15ANDSC=0THENPRINT"Not yet!":GOTO210
  389IFZZ>3THENPRINT"Hands full!":GOTO210
  390 ZZ=ZZ+1:PRINT"OK.":OB%(NO)=-1:GOTO210
```

Explanation

Line 300 - if the noun number is zero, then we don't know what the noun is, so GOTO 1900 to print out message, then gosub to the two routines to check bottles and torches, and to check whether or not an object is being carried or is in the room. If it isn't being carried, then go back to line 210 again.

Line 304 - if the object number is set to minus 1, we're already carrying it!

Line 308 - if you're trying to get object 18, the pool of oil, or object 39, the whisky, but you're not carrying the empty bottle, object number 19, then you can't have it!

Line 312 - 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 314 - ditto for the pool of oil

Line 315 - 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 324 - if you pick the plank up from room 10, then you can't get past the chasm again, so adjust everything accordingly.

Line 326 - 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 389 - check to see how much is already being carried, and if the player's carrying more than four objects (we increase the counter AFTER this line!) then he can't have it.

Line 390 - everything's OK, increase the number of objects being carried counter, put the object in the player's 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.

```
  500 GS=0:ZZ=0:PRINT"You have :":FORI=1TOLO:IFOB%
(I)=-1THENPRINTOB$(I):GS=GS+1:ZZ=ZZ+1
  504NEXT:IFGS=0THENPRINT"Nothing."
  505GOTO210
```

Explanation

Line 500 - print out a simple message, and set the variable GS to zero, and also the number of objects being carried to zero. Then 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 object's description from the variable O$, and increment the two counters GS and ZZ.

Line 504 - NEXT step through the loop. If GS equals zero then you can't be carrying anything, so just print out the word NOTHING.

Line 505 - go back and get another input.

DROP

This verb is used to drop anything that you might be carrying.

Since some objects when dropped can cause us problems (bottles have a tendency to break!), we have to check for everything.

```
   560IFNO=0THEN1900ELSEGOSUB5300:IFOB%(NO)<>-1THEN
PRINT"LOOK!":GOTO210
   566 IF (NO=19ORNO=51ORNO=52)THENPRINT"Smash!":OB
%(NO)=0:OB%(50)=CP:GOTO586
   572 IFNO=16THENPRINT"It vanishes!":OB%(16)=0:ZZ=
ZZ-1:GOTO210
   574 IFNO=46THENOB%(NO)=0:OB%(45)=CP:GOTO586
   575 IFNO<>12ORCP<>10THEN580
   578 PRINT"Good idea!":OB%(12)=CP:P%(10,3)=14:P$(
10)="on the plank.":GOTO586
   580 PRINT"Okay.":ZZ=ZZ-1:OB%(NO)=CP:IFTB=1THEN58
4 ELSE210
   584 PRINT"You scared the bear!":TB=0:OB%(9)=RND(
41):GOTO210
   586 PRINT"Okay.":ZZ=ZZ-1:GOTO210
```

Explanation

Line 560 - noun not recognised, so GOTO 1900, GOSUB 5300 to check the bottle and torch situation, and if you're not carrying the object then you can't drop it, so print a message and GOTO 210.

Line 566 - if you drop the bottle (or the bottle of whisky or the bottle of oil), 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 572 - if you attempt to drop object number 16, the mirror, it vanishes, so print out a suitable message, remove mirror, and decrement object counter, then GOTO 210 for more input.

Line 574 - dropping the blazing torch causes the blazing torch to disappear, the old torch to appear in the room, and a jump to line 586.

Line 575 - if you're not trying to drop object 12, the plank, in room 10, the chasm, then off to line 580.

Line 578 - print out a message of congratulations at doing something right, put the plank in the room, enable you to go west from room 10, change the description for room 10, and GOTO 586.

Line 580 - 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 584, but otherwise it's back to 210 for more inputs.

Line 584 - 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 586 - 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.

```
1890PRINT"Save to tape?"
1892PR$=GET$
1893IFPR$="Y"THEN3000ELSE614
```

Explanation

Line 1890 - print out a message and give the player the chance of saving the game onto tape, so that he can start 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 they've pressed 'Y' then go to the routine at line 3000 onwards, otherwise GOTO 614, which gives you the chance of running through the game again before definitely finishing.

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 the player 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.

```
690 PRINT"Try a direction!":GOTO 210
```

Explanation

Line 690 - just print up a simple message, and go back to line 210 again.

OPEN

There are a number of things that can be OPENed in this game, or at least that the player can try to 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.

```
  780IFNO=0THEN1900ELSEIFCP<>60ANDCP<>3THENPRINT"N
othing to open.":GOTO 210
  782 IFNO<>32ANDNO<>35THENPRINT"Not necessary.":G
OTO 210
  784 IFCP=60THEN790
  786IFGF=1THEN794
  788 IFOB%(42)<>-1THENPRINT"You have no key.":GOT
O 210
  789PRINT"The gate opens!":GOTO2510
  790IFDF=1THEN794
  792PRINT"Not so fast!":GOTO210
  794PRINT"It's open!":GOTO210
```

Explanation

Line 780 - if you don't recognise the noun, ask them what they're trying to do, and GOTO210. Otherwise, if they're not in rooms 60 or 3 then there's nothing to open, so tell them so, and GOTO210

Line 782 - if they're not trying to open the door or the gate then tell them that it isn't necessary, and GOTO 210.

Line 784 - if they're in room 60 then it's off to 790.

Line 786 - if the gate flag is set then the gate is already open, so GOTO 794.

Line 788 - if they're not carrying object 42, the key, then they can't open it anyway, so tell them again, and GOTO210

Line 789 - they can open the gate, so print a suitable message. This signifies the end of the game, so GOTO2510 to print out a congratulatory message.

Line 790 - if the door flag is set then it's already open, so GOTO 794.

Line 792 - you don't open doors in this game merely by using the word OPEN, so print a message and go back to line 210 as usual.

Line 794 - standard response if something's open, and back to 210 again.

CLOSE

This is used whenever the player attempts to close something in the game. In Underground Adventure the only things that he can close are the gate or the door, so we check for that accordingly.

```
   880 IFNO=0THEN1900
   881 IFNO<>32ANDNO<>35THENPRINT"Huh?":GOTO 210
   882 IFCP=3ANDGF=0THENPRINT"It's closed already!"
ELSEPRINT"You can't.":GOTO210
   884 IFDF=0THENPRINT"It's closed!":GOTO210
   886 PRINT"Okay.":P%(60,1)=0:DF=0:P$(60)="faced w
ith a closed door.":GOTO210
```

Explanation

Line 880 - if you don't recognise the noun, GOTO 1900 to print out message.

Line 881 - 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 882 - if they're in room 3 and the gate flag is set to zero, then it's already closed, so tell them so. Otherwise just print up a short message telling them that they can't (that's the trouble with magic gates) and GOTO 210.

Line 884 - if the door flag is set to zero then the door is already closed, so tell them so and GOTO 210

Line 886 - print OK, close off the south exit from room 60, set DF equal to zero, change the message for room 60, and GOTO 210

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 IFNO=0THEN1900
  902 GOSUB5400:IFOB=0ORNO<>10THENPRINT"Remember y
our diet!":GOTO210
  910 PRINT"Delicious!":OB%(10)=0:ZZ=ZZ-1:GOTO 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 - GOSUB 5400 to check if the object's anywhere in sight. If it isn't, or they're trying to eat something other than object 10, the bun, then GOTO 210 after printing a suitably silly message.

Line 910 - 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 IFNO=0THEN1900
   952 GOSUB5400:IFOB=0ORNO<>9THENPRINT"Nothing doi
ng!":GOTO210
   954 IFOB%(10)=-1THEN1072ELSEPRINT"It's not hungr
y.":GOTO210
```

Explanation

Line 950 - if you don't recognise the noun, then go off to 1900 to print out the usual message.

Line 952 - GOSUB 5400 to see if the object's anywhere around. If it isn't, or you're not trying to feed the bear, then print a suitable message and go back to line 210 again.

Line 954 - if you're trying to give the bear the bun then go off to the routine at line 1072 (part of the OFFER routine), to save duplicating code, since the same thing is handled there as would be handled here. Otherwise, whatever you're trying to feed is suddenly not hungry, so print a message and go back to line 210 again.

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 IFNO=0THEN1900
 1010 GOSUB5400:IFOB=0ORNO<>52THENPRINT"Thirsty?":
GOTO210
 1020 PRINT"Glug Glug Glug ... HIC!!":OB%(52)=0:OB
%(19)=-1:GOTO210
```

Explanation

Line 1000 - if you don't understand the noun, then it's off to the subroutine at line 1900

Line 1010 - GOSUB 5400 to check if the object's anywhere around. If it isn't, or the player's trying to drink anything other than whisky (people do, people do!), then print a short message and dive back to line 210 as per usual.

Line 1020 - 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.

```
 1050IFNO=0THEN1900ELSEGOSUB5300:GOSUB5400:IFOB=0T
HEN210
 1054 IFNO=52ANDCP=50THENPRINT"The grateful spirit
 takes it.":OB%(52)=0:OB%(19)=-1:P%(50,3)=55:P$(50
)="walking past old spirits.":OB%(29)=0:GOTO210
 1060 IFNO=10ANDCP=27THEN1072ELSEPRINT"No dice!":G
OTO210
 1072 PRINT"The bear befriends you.":P%(27,0)=28:P
$(27)="in bear country.":OB%(10)=0:ZZ=ZZ-1:TB=1:GO
TO210
```

Explanation

Line 1050 - if you don't recognise the noun, then GOTO 1900 as usual. Otherwise, go to our usual procedures to check that we're not confusing the various bottles and torches, and to check whether the object in question is anywhere to be seen.

Line 1054 - if they're in room 50 and are trying to offer the bottle of whisky to the ancient denizen of the caves (referred to as a 'ghostie' in the data statements) then aha! the denizen of the caves gratefully accepts their kind present, so print out a suitable message. 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. Remove the ghostie and go back to line 210 again.

Line 1060 - if they're not in room 27 and they're not trying to offer the bun, then no one's interested, so print a message and go back to 210

Line 1072 - the bear eats the bun! Print out message. Allow them to go north from room 27, change the description of room 27, make the bun disappear (Paul Daniels would be proud of you), decrement the object counter ZZ and set the bear flag equal to 1. This means that you will now be followed by the bear until ... wait and see! Finally, back to 210 for the next input.

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 players' heads, and they then merrily wave anything they can get their hands on, so we have to check for all of that.

```
 1100IFNO=0THEN1900ELSEGOSUB5300:GOSUB5400:IFOB=0T
HEN210
 1104 IFCP<>15ORNO<>2THENPRINT"Nothing happens.":G
OTO210
 1106IFBR=1THENPRINT"Again?":GOTO210
 1108PRINT"Lucky you!":BR=1:OB%(6)=CP:P%(15,1)=17:
P%(15,2)=16:P$(15)="crossing the chasm.":GOTO210
```

Explanation

Line 1100 - if you don't recognise the noun, then GOTO 1900, otherwise it's our usual trip to lines 5300 and 5400 respectively. If the object in question is nowhere to be seen, then shoot off to 210 as usual.

Line 1104 - if they're not waving object number 2, the staff, and they're not in room 15, the chasm, then print out message and GOTO 210

Line 1106 - 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 1108 - 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. Allow them to go east from room 15 as well, change the description of room 15, and finally GOTO 210 as usual.

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.

```
 1200IFNO=0THEN1900ELSEGOSUB5300:GOSUB5400:IFOB=0T
HEN210
 1204 IFNO<>3ANDNO<>12THENPRINT"Ahem!":GOTO210
 1205 IFOB%(4)<>-1THENPRINT"You need an axe.":GOTO
210
 1206 IFNO=3THENPRINT"TIMBERRRR!":P%(21,2)=22:P$(2
1)="in tree country.":OB$(3)="an ex-tree.":GOTO210

 1212 PRINT"Well done!":OB%(12)=0:OB%(53)=-1:GOTO2
10
```

Explanation

Line 1200 - usual jaunt to line 1900 if the noun isn't recognised. Otherwise, it's off to the subroutines at 5300 and 5400 to check for bottles and torches, and to see if the player is carrying the object or if it's in the room.

Line 1204 - if they're not trying to chop the tree, or the plank, then tell them that it can't be done, and GOTO 210

Line 1205 - 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 1206 - if it's the tree they're after, then print message, 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

Line 1212 - print a message about the plank, remove the plank, put the neatly sawn timber in their possession (a fine piece of axemanship!), 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.

```
 1250IFNO=0THEN1900ELSEGOSUB5300:GOSUB5400:IFOB=0T
HEN210
 1254 IFNO=3THENPRINT"In these shoes?":GOTO210
 1255 IFNO<>5ANDNO<>13THENPRINT"What?!":GOTO210
 1256 IFNO=5THEN1260
 1257 IFCP<>45ANDCP<>47THENPRINT"No good here.":GO
TO210
 1258 IFOB%(13)=-1THENCP=92-CP:OB%(13)=CP:GOTO200
 1259 PRINT"Try picking it up!":GOTO210
 1260 IFCP<>35ANDCP<>36THENPRINT"No good here.":GO
TO210
 1262 IFOB%(5)=-1THENCP=71-CP:OB%(5)=CP:GOTO200
 1263 GOTO1259
```

Explanation

Line 1250 - our usual run through lines 1900, 5300 and 5400. If the object isn't in the room or the player's possession (OB = 0), then GOTO 210.

Line 1254 - 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 1255 - if the player isn't trying to climb object 5 (the rope) or object 12 (the ladder), then print out message and GOTO 210.

Line 1256 - if the player is trying to climb object 5, the rope, then off to 1260

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 ladder's in the player's possession, and if the player's in room 45 or 47, then put him in room 47 or 45, put the ladder in room 47 or 45, then GOTO 200 for a LOOK.

Line 1259 - if the ladder or the rope's not being carried, then print a helpful suggestion and GOTO 210

Line 1260 - 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 1262 - if the rope is in the player's possession, if the player's in room 35 or 36, then put him and the rope in room 36 or 35, print a message and GOTO line 200

Line 1263 - otherwise, GOTO 1259 to print out suitable message.

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 IFNO=0THEN1900
 1301 IFOB%(44)<>-1THENPRINT"What with ?":GOTO210
 1302GOSUB5300:GOSUB5400:IFOB=0OR(NO<>45ANDNO<>46A
NDNO<>7)THENPRINT"You can't.":GOTO210
 1303 IFNO=7THEN1310
 1304 IFOB%(46)=-1THENPRINT"It's lit."ELSEPRINT"Ok
ay.":OB%(46)=-1:OB%(45)=0:PD=0:GOTO210
 1310 IFOB%(7)=-1THENPRINT"That was silly.":GOTO61
2
 1312 IFCP<>4THENPRINT"Wrong place.":ZZ=ZZ-1:OB%(7
)=0:GOTO210
 1314 PRINT"You're through.":ZZ=ZZ-1:OB%(7)=0:P%(4
,3)=5:P$(4)="walking through rubble.":GOTO210
```

Explanation

Line 1300 - unrecognised noun, so GOTO 1900

Line 1301 - check to see if the player is holding object 44, i.e. the matches, and if he isn't then he can't light anything, so print message and GOTO 210

Line 1302 - the usual check using the subroutines at 5300 and 5400, and if the object isn't being held or isn't in the room, or the player is trying to light something other than the torch or the dynamite, then print out a message and GOTO 210.

Line 1303 - if the object they're trying to light is the dynamite then GOTO 1310

Line 1304 - if they're carrying the blazing torch, object 46, then there's no point lighting the torch, so say so and GOTO 210. Otherwise, light the torch and put it in the player's possession, remove the old torch, reset the Pitch Dark counter and GOTO 210.

Line 1310 - if the player is holding the dynamite while trying to light it, this is understandably fatal, so GOTO 612 for the death routine.

Line 1312 - if the player isn't in room 4, then the dynamite blows up but nothing else happens, so make the dynamite disappear, decrement the object counter, print a message and GOTO 210.

Line 1314 - 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, print a reasonable message and GOTO 210

ATTACK, KILL and HIT

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.

Again, owing to the demands on the machine's memory, and since ATTACK, KILL or HIT don't actually achieve anything in this game, all three verbs are sent off to a common subroutine.

This not only saves space, but also manages to convey the impression that the adventure can handle a lot more verbs than it is really programmed for. They may all produce the same response, but it is far better to have something like this than a standard 'I don't know that word' routine built in. So, if all three verbs do the same thing, how does the program handle it?

```
 1400IFNO=0THEN1900ELSEGOSUB5300:GOSUB5400:IFOB=0T
HEN210ELSEPRINT"I think not.":GOTO210
```

Explanation

A single line of code to handle all three verbs!

Using the level of Basic that Acorn have kindly supplied us with, this is not too difficult, and although the line may be rather long, it manages to cope with everything that the player might type in using the verbs ATTACK, KILL or HIT.

In your own games these routines would probably be understandably longer, but since we aren't too bothered about killing things (at least, not with a simple word like one of the above three!), the single routine will suffice.

Line 1400 - first of all, check for a recognised noun. Then check that the thing to be attacked, killed or hit is in the room or in the player's possession by going to the subroutine at 5400 (via a visit to 5300 to avoid confusion over bottles and torches). If it isn't then GOTO 210. If it is, then gently remind the player that this is not a very good idea, and go back to 210 and try something else.

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 IFNO=0THEN1900
 1502 IFNO<>13THENPRINT"Sorry?!":GOTO210
 1504 IFOB%(53)<>-1OROB%(14)<>-1OROB%(4)<>-1THENPR
INT"Not yet.":GOTO210
 1506 PRINT"Done.":OB%(13)=-1:OB%(14)=0:OB%(53)=0:
ZZ=ZZ-1:GOTO210
```

Explanation

Line 1500 - if it's an unrecognised noun then print up a simple statement to that effect (via the subroutine at 1900) 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 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). Finally, back to 210 again for the next input.

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 IFNO=0THEN1900
 1552 IFNO<>47ORCP<>93THENPRINT"Nothing doing.":GO
TO210
 1553 IFOB%(16)<>-1THENPRINT"No can do.":GOTO210
 1554 IFSC=1THENPRINT"Again ?!":GOTO210
 1556 PRINT"The curtain falls!":P%(93,0)=95:OB%(15
)=CP:SC=1:P$(93)="in curtain land.":GOTO210
```

Explanation

Line 1550 - an unrecognised noun, so print an appropriate message via 1900 and GOTO 210 to try again.

Line 1552 - if the object that the player is trying to reflect isn't the LIGHT, and they're not in room 93 where the curtain lives, then print an 'I don't understand' message and try again.

Line 1553 - if the player isn't holding the mirror (object 16), then there isn't anything that can be done, so print a message and GOTO 210.

Line 1554 - 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 1556 - print the all important message, 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

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 IFNO=0THEN1900
 1602 IFOB%(51)<>-1THENPRINT"What with?":GOTO210
 1604 IFCP<>79ORNO<>17THENPRINT"What a waste!":GOT
O1608
 1606 PRINT"The track slides away.":P%(79,2)=80:P%
(79,3)=81:OB%(17)=0:P$(79)="walking by old tracks.
"
 1608 OB%(51)=0:OB%(19)=-1:GOTO210
```

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 and we're not trying to oil the track, then there isn't anything worth oiling, so we inform the player and then go to line 1608, which removes the bottle of oil, and returns an empty bottle to the player. This makes the game unsolvable, but if people will do these things ...

Line 1606 - print the message of success, then allow the player to go east from room 79 to room 80, and west from room 79 to room 81, remove the track, and finally change the description of room 79.

Line 1608 - remove the bottle of oil, put the empty bottle back in the player's possession, and go back to our old friend line 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.

```
 1650IFNO=0THEN1900ELSEGOSUB5300:GOSUB5400:IFOB=0T
HEN210
 1654IFOB%(38)<>-1THENPRINT"No sword.":GOTO210
 1656IFNO<>30THENPRINT"No can do.":GOTO210
 1658PRINT"Success. You got it!":P%(84,2)=86:P%(84
,3)=85:OB$(30)="a dead spider.":P$(84)="in ex-spid
er country.":GOTO210
```

Explanation

Line 1650 - usual check for an unrecognised noun, followed by the dive to lines 5300 and 5400 to check for bottles and torches, and to see if the object in question is either in the room or in the player's possession. If the player hasn't got the object then GOTO 210.

Line 1654 - 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 1656 - if the player isn't trying to stab the spider, then print a suitable message and GOTO 210

Line 1658 - print 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. Then 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 1656 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..

```
 1700IFNO=0THEN1900ELSEGOSUB5300:GOSUB5400:IFOB=0T
HEN210
 1703IFOB%(34)<>-1THENPRINT"You've no spray.":GOTO
210
 1705IFNO<>31THENPRINT"Cough-Cough!":GOTO210
 1707PRINT"You've killed it!":OB$(31)="a dead fly.
":P%(74,3)=75:P$(74)="in the spiders graveyard.":G
OTO210
```

Explanation

Line 1700 - you should be used to this by now, as we go to lines 1900, 5300 and 5400 as usual.

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 1705 - 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 1707 - print the message, 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 particular objects have two very important roles to play, as we shall see:

```
 1750IFNO=0THENPRINT"Throw what?":GOTO210
 1752IFNO<>33ANDNO<>4THEN560ELSEIFNO=4ANDOB%(4)=-1
ANDNP=1THENGA=1:PROCGARG
 1754IFNO=4THEN560
 1756IFOB%(33)<>-1THENPRINT"You've not got it.":GO
TO210
 1758IFCP<>60THENPRINT"Okay!":OB%(33)=14:ZZ=ZZ-1:G
OTO210
 1760PRINT"You've smashed the door!":P%(60,0)=61:P
%(60,3)=65:OB%(33)=0:ZZ=ZZ-1:DF=1:P$(60)="walking
by the door.":GOTO210
```

Explanation

Line 1750 - as usual, we don't recognise the noun, so print a message and GOTO 210.

Line 1752 - if the player isn't throwing the mortar or the axe then transfer program execution to the drop routine starting at line 560. However, if the player is trying to throw the axe, the axe is actually in his possession, and the native present counter (NP) is set, then assume the axe is being thrown at the gargoyle. Set the gargoyle counter GA and go to the PROCGARG procedure.

Line 1754 - if the relevant counters aren't set, but the player still wants to throw the axe, then trot off to line 560 and continue from there.

Line 1756 - check that the mortar is in the player's possession.

Line 1758 - if the lump of mortar is thrown anywhere other than in room 60, the room with the old door, then put it in room 14 (awkward to get to), nothing else happens, we decrement the object counter, and GOTO 210.

Line 1760 - it's been thrown in the right place, so print the appropriate message, 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. Then we 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:

```
 1800IFNO=0THEN1900ELSEGOSUB5300:GOSUB5400
 1802 IFOB=0THEN210ELSEPRINT"Hmmm, strange idea!":
GOTO 210
```

Explanation

Line 1800 - usual check for the presence of an unknown noun, followed by our check for torches and bottles, and the check for the object about to be rubbed being in the player's possession, or at least being in the room.

Line 1802 - if the object isn't around, then GOTO 210, otherwise 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!

```
 1850IFNO=0THEN1900ELSEGOSUB5400
 1854 IFOB=0THEN210ELSEIFNO<>48THENPRINT"This isn'
t the SUN!":GOTO210
 1856 PRINT"Try finding nails, planks and axes to
  make a ladder.":GOTO 210
```

Explanation

Line 1850 - usual check for an unrecognised word, followed by the check for something being a bottle or a torch, and then the check to see if the object is around anywhere.

Line 1854 - if they're not trying to read object 48, the old parchment, then print a suitable message, and go back to line 210

Line 1856 - print the message contained on the old parchment, and then GOTO 210.

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 IFNO=0THENPRINT"What's that mean ?":GOTO 210
 1901 IFNO=1ORNO=6ORNO=43THENPRINT"Nothing here.":
GOTO 210
 1903GOSUB5300:GOSUB5400:IFOB=0THEN210
 1905 IFNO=2ORNO=26ORNO=33ORNO=37ORNO=38THENPRINT"
This is useful!":GOTO210
 1906 PRINT"It's just ";OB$(NO):GOTO 210
```

Explanation

Line 1900 - usual check for an unrecognised noun, and then print a message before going back to 210 again.

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, and then 5400 to see if the object to be examined is anywhere around. If it isn't, then back to 210 as usual.

Line 1905 - if the player is examining the staff, the mirror, the brick, the 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 IFCP=15ORCP=45THENPRINT"Silly person!":PRINT
:PRINTD$:GOTO612
 1952 PRINT"Wheee!":GOTO 210
 1960PRINT"Try another word.":GOTO210
```

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 - tell the player that he can't break anything, and go back to line 210 again.

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.

```
1970GOTO1960
```

Explanation

Same response as to the request to BREAK something. You've just got to try using another word until you find the correct one.

Nevertheless, it still contains that all important feeling that the adventure game can understand a lot more words than it really does. Far better to have something like this, than a boring 'I don't understand' message every time.

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!

```
 3000PRINT"Insert tape.":M=OPENOUT "data":PRINT#M,
CP,TB,GF,PD,ZZ,SC,DF,BR,NP,GS,GA,P$(10),P$(60),P$(
50),P$(27),P$(15),P$(21),P$(4),P$(93),P$(79),P$(84
),P$(74),P$(69),P$(42),OB$(30),OB$(3),OB$(31)
 3002 PRINT#M,P%(45,1),P%(10,3),P%(3,0),P%(60,1),P
%(50,3),P%(27,0),P%(15,1),P%(15,2),P%(21,2),P%(4,3
),P%(93,0),P%(79,2),P%(84,2),P%(84,3),P%(74,3),P%(
60,0),P%(69,3),P%(42,1)
 3003 FORI=1TOLO:PRINT#M,OB%(I):NEXT
 3004 CLOSE#M:GOTO210
```

Explanation

Line 3000 - print a message to insert tape, and open a file for printing data onto tape. Since Acorn have built the ROM routines in, there is no need to tell the player to press key when ready. Those ROM routines will make the tape sit and wait until RETURN is pressed again.

The rest of line 3000 saves the various flags onto tape, followed by all the room descriptions that change during the course of the game, and the object descriptions. When devising your own save to tape routine, do make sure that you don't leave any of the variables out. Just missing a single one could easily lead to disaster!

Line 3002 - save the room direction data.

Line 3003 - save the position of every object in the game.

Line 3004 - close the file, and then go back to our usual prompt of WHAT NOW at line 210.

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.

```
 3200 PRINT"Insert tape and press PLAY.":PRINT"Pre
ss space when ready."
 3201ZX$=GET$
 3203 M= "data":INPUT#M,CP,TB,GF,PD,ZZ,SC,DF,BR,NP
,GS,GA,P$(10),P$(60),P$(50),P$(27),P$(15),P$(21),P
$(4),P$(93),P$(79),P$(84),P$(74),P$(69),P$(42),OB$
(30),OB$(3),OB$(31)
 3204 INPUT#M,P%(45,1),P%(10,3),P%(3,0),P%(60,1),P
%(50,3),P%(27,0),P%(15,1),P%(15,2),P%(21,2),P%(4,3
),P%(93,0),P%(79,2),P%(84,2),P%(84,3),P%(74,3),P%(
60,0),P%(69,3),P%(42,1)
 3205FORI=1TOLO:INPUT#M,OB%(I):NEXT
 3206CLOSE#M:GOTO210
```

Explanation

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 3203 onwards - load in data in order saved, and carry on till line 3205 has been reached.

Line 3206 - close the file, and then GOTO our usual routine at line 210.

The Rest of the Verbs

Just four more to go now, and they all perform fairly minor functions.

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.

SCORE

One line only, and this is:

```
540PRINTRND(100)" points.":GOTO210
```

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, and so the program merely gives you a random score!

In adventures in which 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, giving vital clues while taking points away, but in Underground Adventure you get no help at all, like this:

```
650 PRINT"No chance!":GOTO210
```

Only a simple message that tells you absolutely nothing!

TAKE

In this game, TAKE functions in exactly the same way as GET, so program execution is just transferred to line 300 and everything is dealt with in the usual way.

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 Underground 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; the only section that we haven't yet seen is the data, which 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.

```
 2001 P$(1)="on an old path heading towards   the
caves."
 2002 P$(2)="getting nearer and nearer to the eeri
e caves."
 2003 P$(3)="at the entrance of the caves,withpath
s leading off in all directions."
 2004 P$(4)="in front of a solid wall of rock and
your progress is halted."
 2005 DATA 0,2,0,0,1,3,0,0,2,15,20,4,0,0,3,0,6,13,
4,9,0,5,0,7,0,9,6,8,0,10,7,0,7,12,5,10,8,11,9,0,10
,0,12,0,9,0,13,11,5,0,0,12,0,0,10,0,3,0,0,0,0,18,0
,15,15,33,18,19,16,34,0,17,0,32,17,0,0,0,21,3,0,0,
0,20,23,0,0,21
 2008 P$(5)="in a subterranean tomb, dotted   abou
t with cracks and crevices."
 2010 P$(6)="walking around the side of the   crev
ice room."
 2012 P$(7)="surrounded by old walls that havelong
 since been bricked up."
 2014 P$(8)="near a great chasm in the rock,  whic
h plunges down hundreds of feet."
 2016 P$(9)="in the heart of the crevice room,with
 tunnels leading off everywhere."
 2018 P$(10)="in front of a chasm which is muchtoo
 deep and wide to cross."
 2020 P$(11)="on the southern rim of the cavernin
a jumble of rocks and boulders."
 2022 P$(12)="hopelessly lost in chasm country!"
```

```
 2024 P$(13)="in a room full of rocks, rocks,  roc
ks and rocks!"
 2026 P$(14)="on the west side of the chasm,   but
 there's not much to look at."
 2028 P$(15)="faced with a crack in the floor  tha
t is too wide to jump."
 2030 P$(16)="in an east-side chamber."
 2032 P$(17)="on the main track through the    cav
es, with paths leading off all over."
 2034 P$(18)="away from the centre track with acho
ice of routes as the tunnels widen   out into reas
onably well worked paths."
 2037 P$(19)="on a sharply twisting corridor   and
 you keep banging your elbow on the  wall (shame!)
"
 2038 P$(20)="on a long east-west track into   the
 heart of the mines."
 2040 DATA25,22,24,0,26,0,0,23,0,23,26,0,27,24,0,2
5,0,26,0,0,0,27,30,29,0,0,28,0,31,0,0,28,0,30,0,0,
19,42,33,41,17,0,34,32,18,0,35,33,0,0,0,34,0,0,38,
0,0,38,39,0,37,0,0,36,0,0,40,37,0,0,0,39
 2041 P$(21)="forced to a halt by an extremely lar
ge underground tree."
 2042 P$(22)="heading down a twisty path into  an
old animals lair.  You can see "
 2043 P$(22)=P$(22)+"from  the old paw prints that
 something big   must live here."
 2044 P$(23)="surrounded by rock in a mixture  of
corridors."
 2045 P$(24)="walking along an old tunnel that has
 been carved out of the rocks."
 2046 P$(25)="forced to turn sharply."
 2048 P$(26)="walking along a fairly large     cor
ridor."
 2050 P$(27)="face to chest with a large and   not
 very friendly looking bear."
 2052 P$(28)="at a T-junction behind the old   bea
r's lair."
 2054 P$(29)="halted by a total dead end."
 2056 P$(30)="near the heart of the bear's     hid
ing place."
 2058 P$(31)="in an old cave used as a resting pla
ce by the bear."
 2059 P$(32)="heading down an off-shoot from   the
 main mine."
 2060 P$(33)="faced with a choice of routes as you
 stand here surrounded by propped up timbers and w
alls."
 2062 P$(34)="heading down a long corridor, andare
 forced to climb over stones and      rubble."
 2064 P$(35)="faced with a very deep drop that is
too deep to jump or climb."
```

```
 2066 P$(36)="at the foot of the drop, with    cor
ridors stretching out before you."
 2068 P$(37)="at the foot of the drop, with a  cor
ridor heading east."
 2070 P$(38)="forced into a turn as the path   tws
ists and turns."
 2072 P$(39)="in a long,low east-west corridor."
 2074 P$(40)="in a dead end, and you can go no fur
ther down this route."
 2075 DATA 0,0,32,0,32,0,0,0,42,0,44,46,0,45,0,43,
44,0,0,0,0,0,43,0,0,52,49,48,0,51,47,50,0,53,54,47
,0,0,48,0,48,66,0,0,47,77,0,0,49,100,0,0,0,0,88,49
,56,57,50,58,0,55,0,0,55,0,0,0,0,0,55,59,60,0,58,0
,0,59,0,0
 2076 P$(41)="at the west of the caves, and    wel
l and truly stopped."
 2078 P$(42)="face to face with an extremely   ang
ry looking panther!"
 2080 P$(43)="at an underground T-junction."
 2082 P$(44)="near a great incline."
 2084 P$(45)="at the foot of a great incline   tha
t is much too steep to climb."
 2086 P$(46)="stuck in another dead end."
 2088 P$(47)="at the top of the great incline  (ph
ew!)"
 2090 P$(48)="near an old scary part of the    cav
es that is reputedly haunted."
 2092 P$(49)="near a reputedly magical part of the
 caves."
 2094 P$(50)="ground to a halt by the ghostly  spi
rit of the caves."
 2096 P$(51)="on an off-shoot from the main    tra
ck."
 2098 P$(52)="on an old path heading north-    sou
th."
 2099 P$(53)="stopped by an extremely narrow   squ
eeze that you can't get through."
 2100 P$(54)="near the magical caverns."
 2102 P$(55)="in an open corridor, with exits  lea
ding off everywhere."
 2104 P$(56)="stuck in an old, stuffy dead end."
 2106 P$(57)="stuck in an old, musty dead end."
 2108 P$(58)="on a well trodden old path."
 2110 P$(59)="forced to turn as the path bobs  and
 weaves amongst the rocks."
 2112 P$(60)="faced by a door marked with the  wor
ds 'BEGONE STRANGER'!"
 2114 DATA61,61,61,62,61,61,61,63,61,64,61,61,61,6
1,65,61,61,61,60,61,51,67,0,68,66,0,0,69,0,69,66,0
,68,0,67,0,0,71,69,0,70,72,0,74,71,0,0,73,74,0,72,
0,0,73,71,0,0,0,74,76,0,0,75,0
 2116 FORI=61 TO 65:P$(I)="in a maze of twisting p
```

```
assages,  all alike!":NEXT
 2122 P$(66)="walking along an indistinct path,and
 the walls look rather damp."
 2124 P$(67)="in a low, damp corridor."
 2126 P$(68)="in a low corridor.  It all seems to
be a little damp here."
 2128 P$(69)="stopped by a wall of hazy mist   tha
t obscures all light."
 2130 P$(70)="on the south side of the mist.   The
 air seems clearer here."
 2132 P$(71)="heading along a clear path cut   fro
m the living rock."
 2134 P$(72)="in a sharply twisting corridor."
 2136 P$(73)="twisting and turning amongst the roc
ks in the FLY room!"
 2138 P$(74)="face to face with a giant fly    tha
t completely blocks your path."
 2140 P$(75)="in a low east-west corridor that is
totally free of insects."
 2142 P$(76)="in a complete dead end and can   go
no further."
 2144 P$(77)="still heading north-south (phew!)"
 2146 DATA 52,78,0,0,77,79,0,0,78,0,0,0,0,83,0,79,
0,82,79,0,81,0,0,0,80,84,0,0,83,0,0,0,0,0,84,0,0,8
7,0,84,86,0,0,0,89,90,92,54,0,88,91,0,88,94,0,0,0,
92,93,89,91,0,97,88,0,97,0,91
 2148 P$(78)="at the bottom of a long, low     nor
th-south passage."
 2152 P$(79)="prevented from going further by  an
old seized up mining track."
 2154 P$(80)="weaving around, with old, dusty  cob
webs hanging everywhere."
 2156 P$(81)="on the west side of the track."
 2158 P$(82)="in the salvage room, but can go  no
further."
 2160 P$(83)="near to the SPIDER room!"
 2162 P$(84)="in spider country and the large  spi
der here won't let you pass!"
 2164 P$(85)="in an old chamber known as the   spi
ders graveyard."
 2166 P$(86)="near to the SPIDER room!"
 2168 P$(87)="in a total dead end!"
 2170 P$(88)="in the heart of the magical      cav
erns, with paths going everywhere."
 2172 P$(89)="in a northern off-shoot from the mai
n path."
 2174 P$(90)="walking along a magical corridor."
 2176 P$(91)="in a dimly lit corridor."
 2178 P$(92)="near the source of the magic!"
 2180 P$(93)="halted by a magical shimmering   cur
tain that will not let you pass."
 2182 P$(94)="in another dead end."
 2183 DATA 90,0,0,0,0,93,96,0,99,0,98,95,93,0,0,92
,0,0,0,96,0,96,0,0,53,0,0,0,15,20,21,34,24,0,40,0,
27,7,42,46,0,14,93,67,79,48,98,69
 2184 P$(95)="on the northern side of the      shi
mmering curtain."
 2186 P$(96)="in a low, low corridor."
 2188 P$(97)="treading over dimly lit rocks andrub
ble."
 2189 P$(98)="in a dead end.  The wall is      bri
cked up and you can go no further."
 2190 P$(99)="in a northern off-shoot from the mai
n path."
 2192 P$(100)="in an old warehouse, last used   by
 the miners many years ago."
 2200 FORI=1TOP:FORJ=0TO3:READP%(I,J):NEXTJ,I
 2212 DATAa chasm,a staff,a tree,an axe,some rope,
a bridge,dynamite!,some rubble,a bear,a bun,a pant
her,a plank,a ladder,some nails,the curtain,a mirr
or,a blocked track,an oil pool,an empty bottle,a m
isty wall
 2222 FORI=1TO20:READOB%(I):NEXT:FORI=1TO20:READOB
$(I):NEXT
 2224 DATA 50,84,74,60,76,87,3,53,63,31,73,0,0,100
,0,3,1,0,0,39,0,0,0,0,0
 2226 DATAa ghostie,a spider,a fly,an old door,som
e mortar,a fly spray,a gate,a narrow crack,a stone
,a sword,some whisky,a gargoyle,a knife,a key,a wa
ll,some matches,an old torch,a lit torch,a glowing
 light,a parchment,program,broken glass
 2237 DATAa bottle of oil,a bottle of whisky,some
sawn timber
 2238 FORI=29TOLO:READOB%(I):NEXT:FORI=29TOLO:READ
OB$(I):NEXT
 2250 DATACHA,STA,TRE,AXE,ROP,BRI,DYN,RUB,BEA,BUN,
PAN,PLA,LAD,NAI,CUR,MIR,TRA,OIL,BOT,MIS,NOR,SOU,EA
S,WES,N,S,E,W,GHO,SPI,FLY,DOO,MOR,SPR,GAT,CRA,STO,
SWO,WHI,GAR,KNI,KEY,WAL,MAT,TOR,TOR,LIG,PAR,PRO,GL
A,BOT,BOT,TIM
 2256 DATA GO,GET,LOO,INV,SCO,DRO,HEL,QUI,CRO,TAK,
OPE,CLO,EAT,FEE,DRI,OFF,WAV,CUT,CHO,CLI,LIG,ATT,KI
L,HIT,MAK,REF,OIL,STA,SPR,THR,RUB,REA,EXA,JUM,BRE,
PUS,SAV,LOA
 2262FORI=1TONN:READNO$(I):NEXT:FORI=1TONV:READVB$
(I):NEXT:DIMD$(3):FORI=0TO3:READD$(I):NEXT:ENDPROC
 2267 DATA North,South,East,West
```

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. Although Acorn allows us to have strings defined to a reasonable length, it does take up a lot of memory space. The trade-off between room descriptions and number of verbs, objects and problems is up to you!

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 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 line 2238 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 1999.

The shortened forms for the nouns, i.e. the words that we use when analysing any data that has been typed in, are stored in line 2250, and are read in as NO$(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 line 2256, and is read into the variable VB$(I) in line 2262.

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 2267, 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 2262, 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, for example 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!

```
   5 GOSUB1460:DF=0:KN=0:CF=0:SP=0
  10 GB$="A gold bar falls out!"
  15 D1$="Gulp-Gulp-Gulp.  You are shrinking!"
  20 DI$="You haven’t got it."
  25 CM$=CHR$(44)
  30 X=RND(-5):X=0:ZZ=1:PIT=12
  35 VF=0:PF=0:WF=0:SH=0:KN=0:SP=0
  40 QT$=CHR$(34)
  45 VT$="Behind the sign is a vault in the wall.
 The vault is locked."
  50 CLS:PRINT:PRINT:PRINT"       ****Castlemaze A
dventure****"
  55 DB$="You must supply a direct object."
  60 DEFFNR(Q)=INT(RND(1)*Q+1
  65 CP=49:S1$="I don’t see it here.":S2$="Don’t
be ridiculous.":PROCSETUP
  70 PROCMOVE
  75 IFCP=52 AND KN=0 THEN750
  80 IFOB(2,0)=-1 AND PIT=CP THEN790
  85 IFCP=29ANDSP=0THEN800
  90 T=T+1:PROCINPUT:IFVB$="3.1"THENP(30,2)=31:GO
TO70
  95 *FX15,0
 100 IFVB=-1AND(NO>21ANDNO<30)THENVB=1
 105 IFVB$="CRO"THENIF(CP=52ORCP=53)THENCP=105-CP
:GOTO70
 110 IFVB<>30AND(VB>10ORVB=20ORVB=6)ANDNO$=""THEN
PRINTDB$:GOTO90
 115 IFVB=30THEN695
 120 IFVB=-1ANDNO<>0AND(NO<22ORNO>29)THENPRINT"Yo
u must supply a verb.":GOTO90
 125 IFVB<1ANDNO=0THENPRINT"I don’t know how to d
o that.":GOTO90
 130 IFNO=0ANDVB>10THENPRINT"What’s a ";N1$;" ?":
GOTO90
 135 ON VB GOTO 140,180,70,215,230,250,470,295,42
5,840,310,315,330,180,375,380,405,375,430,430,330,
250,270,510,515,560,580,250,650,695
 140 IF (NO<22 OR NO>29) AND NO$<>"" THEN PRINT"I
’m afraid I can’t do that.":GOTO 90
 145 IF NO$="" THEN PRINT "Go where ?":GOTO 90
 150 IF NO>25 THEN NO=NO-4
 155 NO=NO -22:IF P(CP,NO)=0 THEN PRINT"You can’t
 go that way yet.":GOTO 90
 160 IF CP=1 AND NO=1 AND DF=0 THEN PRINT"The cas
tle door is locked I’m afraid.":GOTO 90
 165 IF CP=17 AND NO=1 AND CF=0 THENPRINT"The cra
ck is much too small for you!":GOTO 90
 170 IF CP=18 AND NO=0 AND OB(9,0)=-1 THEN PRINT"
The painting is too large for the crack.":GOTO 90
 175 CP=P(CP,NO):GOTO 70
 180 IFOB(NO,0)=-1 THEN PRINT"You’ve already got
```

```
it!":GOTO 90
 185 IF NO=0 THEN PRINT"What’s a ";N1$;" ?":GOTO
90
 190 IF OB(NO,0)<>CP THEN PRINTS1$:GOTO90
 195 IFNO=17 OR NO=21 OR NO=20 OR NO=16 THEN PRIN
TS2$:GOTO 90
 200 IF ZZ>4 THEN PRINT"You’re carrying too much
already.":GOTO 90
 205 IF NO=19 AND PF=0 THEN PRINTVT$:PF=1:OB(16,0
)=CP:OB(19,0)=-1:ZZ=ZZ+1:GOTO 90
 210 PRINT"Okay.":ZZ=ZZ+1:OB(NO,0)=-1:GOTO 90
 215 IFZZ=0 THEN PRINT"You’re not carrying anythi
ng at present.":GOTO 90
 220 PRINT"You are carrying the following :=":FOR
I=1TOLO:IFOB(I,0)=-1THENPRINTOB$(I)
 225 NEXTI:PRINT:GOTO90
 230 GOSUB235:GOTO90
 235 J=0:FORI=1TOLO:IFOB(I,0)=1THENJ=J+OB(I,1)
 240 NEXT :PRINT"You’ve scored ";J;" points out o
f 100":IFJ<100 THEN RETURN
 245 CLS:PRINT:PRINT:PRINT"Well done!":END
 250 IFNO<>0 AND OB(NO,0)<>-1THEN560
 255 IFNO=0 THENPRINT"I’ve never heard of a ";N1$
;"!":GOTO 90
 260 IFNO=18 AND OB(13,0)<>CP THEN OB$(18)="A Sha
ttered Vase":OB(18,1)=0
 265 OB(NO,0)=CP:PRINT"Okay then.":ZZ=ZZ-1:GOTO 9
0
 270 IF NO=8 OR NO=14 THEN PRINT"Try ’SWING’":GOT
O 90
 275 IF NO=1 OR NO=4 THEN PRINT"Try ’SHOOT’":GOTO
 90
 280 IF NO=10 THEN PRINT"Try ’SHARPEN’":GOTO 90
 285 IF NO=31 THEN PRINT"Try ’JUMP’":GOTO 90
 290 IF NO=13 THEN PRINT"Just ’DROP’ it where you
 need it.":GOTO 90
 295 IFCP<49 AND CP>44 THEN CP=CP-25:GOTO 70
 300 IFCP<24 AND CP>19 THEN CP=CP+25:GOTO 70
 305 PRINT"That is not possible.":GOTO 90
 310 PRINT"All right then :":PRINTN1$:GOTO 90
 315 IFOB(NO,0)<>-1 THEN PRINTDI$:GOTO 90
 320 IFNO<>7 THEN PRINTS2$:GOTO 90
 325 PRINTD1$:ZZ=ZZ-1:OB(7,0)=0:CF=1:GOTO 90
 330 IF NO<>31 AND NO<>16 AND NO<>30 THEN PRINT"H
ow do you expect to open that ?!":GOTO 90
 335 IF NO=16 AND OB(16,0)<>CP THEN PRINT"What va
ult?":GOTO 90
 340 IF NO=16 AND OB(2,0)<>-1 THEN PRINT"You have
n’t got a key.":GOTO 90
 345 IFNO=16 THEN PRINT"The vault is open.":VF=1:
IFOB(15,0)=0 THEN PRINTGB$:OB(15,0)=CP
 350 IF NO=16 THEN 90
```

```
355 IF NO=31 THEN 730
360 IF CP<>1 THEN PRINT"What door?":GOTO 90
365 IF OB(2,0)<>-1 THEN PRINT"You don't appear to have the key.":GOTO 90
370 PRINT"The door is open.":DF=1:GOTO 90
375 PRINT"How?":GOTO 90
380 IF OB(NO,0)<>-1 THEN PRINT"You don't have it.":GOTO 90
385 IFNO<>3 THEN PRINT"How am I supposed to read that then ?":GOTO 90
390 PRINT"It says 'A SECRET PASSAGE LIES NEARBY"
395 PRINT"WHICH OPENS IF YOU ENTER PI'."
400 GOTO 90
405 IF OB(NO,0)<>-1 AND OB(NO,0)<>CP THEN PRINT"I can't see it here.":GOTO 90
410 IF OB(1,0)<>-1 THEN PRINT"You haven't got a bow!":GOTO 90
415 IF OB(4,0)<>-1 THEN PRINT"You haven't got any arrows!":GOTO 90
420 ZZ=ZZ-1:OB(4,0)=CP:GOTO 90
425 PRINT"You need a tool.":GOTO 90
430 IF NO=16 OR NO=30 OR NO=31 THEN 445
435 IFOB(NO,0)<>-1 THEN 425
440 PRINT"I don't know how to close such a thing.":GOTO 90
445 IF NO=16 AND OB(16,0)<>CP THEN PRINT"What vault ?":GOTO 90
450 IF NO=16 THEN PRINT"The vault is closed and locked":VF=0:GOTO 90
455 IF NO=31 THEN 715
460 IF CP<>1 THEN PRINT"What door?":GOTO 90
465 PRINT"The door is closed and locked.":DF=0:GOTO 90
470 IFCP<8 THEN PRINT"Be persistent.":GOTO 90
475 IFCP<20 THEN PRINT"Examine things.":GOTO 90
480 IFCP<24 THEN PRINT"What goes up, must come down!":GOTO 90
485 IFCP<34 THEN PRINT"Value things.":GOTO 90
490 IF CP<41 THEN PRINT"Do as Hansel and Gretel did.":GOTO 90
495 IFCP<45 THEN PRINT"Think.":GOTO 90
500 IF CP<51 THEN PRINT"This Adventure has a violent beginning.":GOTO 90
505 PRINT"Cross the Bridge.":GOTO 90
510 PRINT"It's value is ";OB(NO,1);" points.":GOTO 90
515 IFOB(NO,0)<>-1 THEN PRINT "You don't seem to have it.":GOTO 90
520 IF NO<>14 THEN 535
525 FORPP=1TO19:IF OB(PP,0)=-1 THEN OB(PP,0)=CP
530 NEXTPP:ZZ=0:CP=23:GOTO 70
535 IF NO<>8 THEN PRINT"Wow, this sure is fun!":
```

```
GOTO 90
540 IF OB(20,0)<>CP THEN PRINT"Whooosshhhh!!":GOTO 90
545 IF SH=0 THEN PRINT"The sword bounces off the Sorceror and  hits you!":GOTO 785
550 PRINT"The sharp sword slices the sorceror.":SH=SH+1:IFSH<4 THEN 75
555 OB(20,0)=0:OB(14,0)=CP:PRINT"The Sorceror disappears.":GOTO 75
560 IF OB(NO,0)<>-1 THEN PRINT"You don't have it.":GOTO 90
565 IF NO<>8 THEN PRINTS2$: GOTO 90
570 IF OB(10,0)<>-1 THEN 425
575 PRINT"The sword is now RAZOR sharp!":SH=1:GOTO 90
580 IF (CP=1 AND NO=30) OR (CP=44 AND NO=31) THEN 620
585 IF OB(NO,0)<>-1 AND OB(NO,0)<>CP THEN PRINTS1$:GOTO 90
590 IF NO=17 AND OB(2,0)=0 THEN OB(2,0)=CP:PRINT"There's something in his pocket!":GOTO 90
595 IF NO=21 AND OB(6,0)=0 THEN OB(6,0)=CP:PRINT"There's something in its stomach!":GOTO 90
600 IF NO=8 AND SH=0 THEN PRINT"It's blunt.":GOTO 90
605 IF NO=8 THEN PRINT"It's very sharp.":GOTO 90
610 IF NO=7 THEN PRINT"On the bottom it says 'Drink me'.":GOTO 90
615 IF NO=18 THEN PRINT"It's very fragile.":GOTO 90
620 IF NO=31 THEN PRINT"It's big enough to jump out of.":GOTO 90
625 IF NO=1 OR NO=30 THEN PRINT"It's made of wood.":GOTO 90
630 IF NO=20 THEN PRINT"He's preparing to cast a spell on you!":GOTO 90
635 IF NO=13 THEN PRINT"It's soft.":GOTO 90
640 IF NO=10 THEN PRINT"It's grey and gritty.":GOTO 90
645 PRINT"It's nothing special.":GOTO 90
650 IF NO=4 AND OB(4,0)=22 THEN PRINT"Look for it in the Forest.":GOTO 90
655 IF NO=2 AND OB(2,0)=0 THEN PRINT"Try examining things.":GOTO 90
660 IF OB(NO,0)=-1 THENPRINT"You're holding it, stupid!":GOTO 90
665 IF OB(NO,0)=CP THEN PRINT"It's right in front of you, stupid!":GOTO 90
670 IF NO<>20 THEN PRINT"Just get on with it and look for it!":GOTO 90
675 PRINT"You're in the Sorceror's Torture Chamberand he was a white hot POKER in his      hands!
```

```
He’s coming towards you !!"
  680 FOR J=0 TO 3: PROCINPUT
  685 IF VB=25 AND NO=14 AND OB(14,0)=-1THEN 515
  690 PRINT"The Sorceror attacks you with the poke
r!":NEXT:GOTO 785
  695 IF (CP>19 AND CP<24) OR CP=24 THEN PRINT"Dow
n":PRINT"Down":PRINT"Down":PRINT"Down":PRINT"Down.
...":GOTO 785
  700 IF CP<>44 THEN PRINT"Wheee!!":GOTO 90
  705 IF WF=0 THEN CP=43 : GOTO 700
  710 PRINT"You land safely in the branches of the
 tree - lucky old you.":CP=21:GOTO 90
  715 IF CP<>44 THEN PRINT"I see no windows.":GOTO
 90
  720 IF WF=0 THEN PRINT"It’s already closed.":GOT
O 90
  725 PRINT"It’s stuck.":GOTO 90
  730 IF CP<>44 THEN 715
  735 IF WF=1 THEN PRINT"It’s already open.":GOTO
90
  740 PRINT"It’s not easy, but you manage to get t
hewindow open (phew!).  You see a big    leafy tr
ee about two metres below the    window."
  745 WF=1: GOTO 90
  750 PRINT"There’s a Black Knight riding across t
hebridge towards you!":PROCINPUT
  755 IF VB<>17 OR NO<>17 THEN 780
  760 IF OB(1,0)<>-1 THEN PRINT"You have no bow (O
H NO!).":GOTO 90
  765 IF OB(4,0)<>-1 THEN PRINT"You have no arrow
(OOOPS!).":GOTO 90
  770 PRINT"The arrow finds a chink in the Knight’
s armour and he plunges to his death."
  775 KN=1:ZZ=ZZ-1:OB(4,0)=52:OB(17,0)=52:GOTO 90
  780 PRINT"The Knight skewers you with his lance.
"
  785 FORI=1TO2500:NEXT:PRINT"You’re dead!":FORI=1
TO2500:NEXT:GOTO 840
  790 PRINT"A Pirate sneaks up on you and steals t
hekey.  ’HAR HAR HAR’ he chortles, as he  scurries
 off.  ’I’LL HIDE THIS IN ME   MAZE!’"
  795 OB(2,0)=34:ZZ=ZZ-1:GOTO 90
  800 PRINT"A Giant Spider drops from the ceiling!
"
  805 PRINT"It is moving towards you.":PROCINPUT
  810 IF VB<>17 OR NO<>21 THEN 835
  815 IF OB(1,0)<>-1 THEN PRINT"Whose lost his bow
, then?":GOTO 835
  820 IF OB(4,0)<>-1 THEN PRINT"Why did you leave
that arrow behind ?":GOTO 835
  825 PRINT"The arrow rips into the Spider!":SP=1:
ZZ=ZZ-1
```

```
  830 OB(21,0)=29:OB(4,0)=0:OB(5,0)=29:GOTO 90
  835 PRINT"The Spider pounces on you and sinks it
s fangs into your neck (oh dear).":GOTO 785
  840 CLS:PRINT:PRINT:PRINT"Byeeeee..!!":END
  845 DEFPROCINPUT
  850 PRINT:PRINT"What now ? ";:PROCINFO
  855 PRINT:N1$="":V1$="":NO=0:VB=0:NO$="":VB$="":
H=0
  860 CM=LEN(CM$):FORI=1TOCM:IFMID$(CM$,I,1)=" "TH
EN H=I-1
  865 NEXT
  870 IFH=0THENH=LEN(CM$)
  875 IFH=1 THEN V1$=CM$:GOTO 885
  880 V1$=LEFT$(CM$,H)
  885 VB$=LEFT$(V1$,3):FORJ=1TONV:IFVB$(J)=VB$THEN
VB=J
  890 NEXT J
  895 IFVB>0THEN905
  900 VB=-1:N1$=V1$:GOTO 915
  905 IFLEN(V1$)+1>LEN(CM$)THENNO=0:ENDPROC
  910 N1$=RIGHT$(CM$,LEN(CM$)-1-LEN(V1$))
  915 NO$=LEFT$(N1$,3):FORI=1TONN:IFNO$(I)=NO$THEN
NO=I
  920 NEXTI
  925 ENDPROC
  930 DEFPROCMOVE
  935SS$="You can see :="
  940 CLS:PRINT:PRINT:PRINT"You’re ";P$(CP):PRINT
  945 FORK=1TOLO
  950 IFOB(K,0)=CP THEN PRINTSS$:PRINTOB$(K):SS$=C
HR$(11)
  955 NEXTK
  960 IF CP=1 AND DF=0 THENPRINT"The door is locke
d."
  965 IFCP=18 AND VF=0 AND OB(16,0)=18 THENPRINT"T
he vault is locked."
  970 IFCP=17 AND CF=0 THENPRINT"A narrow crack le
ads southwards."
  975 IFCP=1 AND DF=1 THENPRINT"The door is open."
  980 IFCP=35 AND VF=1 AND OB(16,0)=35 THENPRINT"T
he vault is open."
  985 IFCP=17 AND CF=1 THENPRINT"A wide crack lead
s to the south."
  990 IFCF=0 THEN P(17,1)=0
  995 IFCP=44 AND WF=1 THENPRINT"The window is ope
n.  A tree lies two      meters below you."
 1000 K=0:PRINT:PRINT"You can go ";:FORL=0TO3:IFP(
CP,L)=0 THEN 1015
 1005 IFK=1 THENPRINT", ";
 1010 PRINTD$(L);:K=1
 1015 NEXTL:IFK=0 THENPRINT"Nowhere!"
 1020 IFK=1 THENPRINT
 1025 PRINT:P(17,1)=18
 1030 ENDPROC
 1035 DEFPROCSETUP
 1040 NP=53:LO=35:NN=31:NV=30:DIMP(NP,3),P$(NP),VB
$(NV),NO$(NN),OB(LO,1),OB$(LO)
 1045 DIMD$(3)
 1050 P$(1)="outside a medieval castle.  The  pave
ment has an inscription:"
 1055 P$(1)=P$(1)+CHR$(13)+CHR$(10)+"It reads 'LEA
VE ALL TREASURE HERE'"
 1060 DATA0,8,4,0,53,7,3,6,0,0,3,2
 1065 P$(2)="standing at a great crossroads,  with
 roads leading off everywhere."
 1070 P$(3)="walking along the great east     road
.  Your feet ache."
 1075 P$(4)="walking along the great west     road
.  Isn't it long ?!"
 1080 DATA 0,0,2,1,0,0,2,4,0,0,2,5,2,7,0,0
 1085 P$(5)=P$(4):P$(6)=P$(4)
 1090 P$(7)="walking along the great south    road
.  Your feet ache."
 1095 P$(8)="standing in a splendid chamber   with
 walls thirty feet high."
 1100 DATA 1,11,0,10,0,0,11,0,0,9,8,0
 1105 P$(9)="in a comfortable old sitting roomthat
 looks somewhat neglected."
 1110 P$(10)="in the master bedroom, that looksver
y grand."
 1115 P$="in a vast corridor stretching outof sigh
t to the ":P$(11)=P$+"south."
 1120 DATA 8,12,14,9
 1125 P$(12)=P$+"north and south."
 1130 P$(13)=P$+"north."
 1135 P$="in a bedroom with a ":P$(14)=P$+"stone f
loor.":P$(15)=P$+"wooden floor."
 1140 P$(16)=P$+"dirt floor."
 1145 P$(17)="in a dusty old pantry, that looksdis
tinctly messy and untidy."
 1150 DATA 11,13,15,25,12,0,16,17,0,0,0,11,0,0,0,1
2,0,33,0,13
 1155 DATA0,18,13,24
 1160 P$(18)="in a private art gallery, with   pai
ntings hanging all around you."
 1165 DATA 17,26,0,19
 1170 P$(19)="in a store room full of musty oldcob
webs."
 1175 P$(20)="at the top of a big,bushy tree."
 1180 P$(21)=P$(20)
 1185 P$(22)=P$(20)
 1190 P$(23)=P$(20)
 1195 DATA 0,0,18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0
```

```
 1200 P$(24)="standing in the old kitchen."
 1205 DATA 25,0,17,0,0,24,12,0,18,28,29,0
 1210 P$(25)="now in the old dining room, once fil
led with old diners but now old and  neglected."
 1215 P$(26)="in a creepy,eerie,shadowy alcove."
 1220 P$(27)="in an austere office.  It ain't  hal
f posh, mum!"
 1225 P$(28)="in the drawing room."
 1230 P$(29)="in the old parlour.":P$(30)="standin
g in what used to be the  study."
 1235 P$(31)="in a damp,stone passage."
 1240 P$(32)="in the old,old dungeon.  Brrr!"
 1245 P$(33)="in an ancient conference room."
 1250 DATA 0,0,28,0,26,0,0,27,0,0,30,26,0,0,0,29,0
,0,32,30,41,0,0,31,16,0,35,0
 1255 P$(34)="in a tower which overlooks a hugekin
gdom down a monsterous mountain.  Thegrass is gree
ner than green itself."
 1260 DATA 40,0,0,0
 1265 P$(35)="in a maze of twisty little pass- age
s, all alike (remember these ?)"
 1270 FORI=36TO40:P$(I)=P$(35):NEXTI
 1275 DATA 36,36,36,33,37,35,35,35,36,36,38,36
 1280 DATA 36,36,36,39,40,36,36,36,36,34,36,36,42,
32,0,0
 1285 P$(41)="on a long flight of stairs       lea
ding further and further down."
 1290 P$(42)="in a mile-long passageway that iscol
d, wet and miserable. Stagnant water laps about yo
ur feet - urgh!"
 1295 P$(43)="on a long flight of stairs       hea
ding up and up."
 1300 P$(44)="at the end of the castle.  Out ofthe
 window you can see a forest in the  distance."
 1305 DATA 43,41,0,0,44,42,0,0,0,43,0,0
 1310 P$(45)="in a dense, dark, creepy forest  wit
h nothing but the trees for company."
 1315 FORI=46TO48:P$(I)=P$(45):NEXTI
 1320 DATA 45,49,46,48,45,47,46,49,50,51,46,48,45,
47,49,48
 1325 P$(49)="walking along an old path made byhor
ses many years ago."
 1330 P$(51)="in the middle of a clearing.  To the
 south you can see an old bridge."
 1335 P$(52)="on the north side of the bridge."
 1340 P$(53)="on the south side of the bridge."
 1345 DATA 45,50,46,48,49,47,46,48,47,52,0,0,51,0,
0,0,0,2,0,0
 1350 DATA BOW,KEY,BOO,ARR,BRO,SAP,LIQ,SWO,PAI,WHE
,SIL,PEN,PIL,SCE
 1355 DATA BAR,VAU,KNI,VAS,SIG,SOR,SPI,NOR,SOU,EAS
,WES,N,S,E,W,DOO,WIN
```

```
 1360 DATA GO,GET,LOO,INV,SCO,DRO,HEL,CLI,DIG,QUI,
SAY,DRI,OPE,TAK,KIL,REA,SHO,ATT
 1365 DATA CLO,LOC,UNL,GIV,USE,VAL,SWI,SHA,EXA,THR
,FIN,JUM
 1370FORI=1TONP:FORJ=0TO3:READP(I,J):NEXTJ:NEXTI
 1375 FORI=1TONN:READNO$(I):NEXTI:FORI=1TONV:READV
B$(I):NEXTI
 1380 DATA A Long Bow,-1,0,A Bronze Key,0,0,A Leat
her-bound book,30,0
 1385 DATA A Silver Arrow,22,10,A Broken Arrow,0,1
0,A Gigantic Sapphire,0,10
 1390 DATA A Vial of Amber Liquid,24,0,A Golden Sw
ord (ooh!),34,10
 1395 DATA A large Rembrandt Painting,18,20
 1400 DATA A Whetstone,19,0,A Set of Silverware,25
,10,A Platinum Pen,27,10
 1405 DATA A Velvet Pillow,44,0,The Sorceror's Sce
ptre,0,10,A Gold Bar,0,10
 1410 DATA A Vault in the Wall,0,0,A Dead Knight (
boo),0,0,A Ming Vase,9,10
 1415 DATA A sign saying DIABOLICAL MAZE,35,0,A Wi
cked Sorceror,32,0
 1420 DATA A Dead Spider,0,0
 1425 FORI=1TO21:READOB$(I),OB(I,0),OB(I,1):NEXTI
 1430 DATA North,South,East,West
 1435 FORI=0TO3:READD$(I):NEXTI
 1440 P$(50)="at the end of a path, with forestsur
rounding you in all directions.  Whatis a soul to
do ?"
 1445 P$(47)=P$(47)+CHR$(13)+CHR$(10)+"To the sout
h there is a dim light."
 1450 ENDPROC
 1455 END
 1460 MODE7
 1465 *FX4,1
 1470 RETURN
 1475 DEFPROCINFO
 1480 CM$=""
 1485 PRINT"*";CHR$(8);
 1490 Z$=GET$
 1495 IFZ$=""THEN1490
 1500 Z=ASC(Z$):IFZ>95ANDZ<>127THEN1490
 1505 ZL=LEN(CM$):IFZL>27THEN1515
 1510 IFZ>31ANDZ<96THENCM$=CM$+Z$:PRINTZ$;:GOTO148
5
 1515 IFZ=13ANDZL>0THENPRINT" ":GOTO1530
 1520 IFZ=127ANDZL>0THENCM$=LEFT$(CM$,ZL-1):PRINT"
";Z$;Z$;
 1525 GOTO 1485
 1530 ENDPROC
```

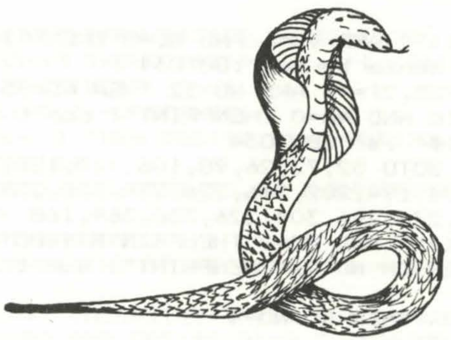
8

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!



```
    0 PROCSTART
    2 DIMD$(3)
    3 BI=0:TG=0:MF=0:M2=0:GF=0:J=0:ZZ=0:MI=0
    4 W1$="The panther sees the snake and flees.":
S4$="You're not holding it."
    6 W2$="You are out of matches.":DR$="It's very
 draughty here."
    8 GB$="A bird swoops down out of the sky and
 lands in front of you."
   10 I1$="You need a direct object."
   12 W5$="The turtle eats the carrot and rubs you
rleg affectionately."
   14 WA$="No Way! The boss says I have to pay for
 anything you break!"
   16 WB$="It doesn't burn.":SP$="You have to disc
over a secret passage."
   18KN$="There's something in his pocket.":IM$="T
hat is not possible."
   20 WD$="It's pitch dark.":CR$=CHR$(13)+CHR$(10)
:PRINT""
   21JA$="The javelin glides through the air as if
pulled by magic."
   22 S1$="I don't see it here.":S2$="Don't be rid
iculous.":CP=39
   24 S3$="I don't know that word."
   25 PROCVARIABLES
   26 GOSUB414
   28 IFTG>0 THEN OB%(29,0)=CP:IFCP=36 THEN TG=0
   30 IFTG THEN PRINT"The turtle is following you.
"
   32 IFTG>0 AND CP=11 THEN GOSUB148:PRINT"CAVE-IN
 !":P%(13,3)=0:P%(9,1)=0:CP=13
   34 PROCINPUT
   36 IFMF=1 AND M2=0 THEN M2=1
   38 IFVB>9 AND VB<>20 AND NO$="" THENPRINTI1$:GO
TO34
   40 IFNO$<>"" AND VB=1 AND NO=0 THENPRINT"That d
oesn't make sense to me.":GOTO34
   42 IFOB%(35,0)=-1 AND NO=32 THEN NO=35
   44 IFVB>10 AND NO=0 THENPRINT"I don't know what
 a ":PRINTN1$" is!":GOTO34
   46 ON VB GOTO 52,72,26,98,106,110,128,376,136,7
2,154,172,174,194,202,216,226,234,236,238,246,248,
246,262,272,274,296,304,226,356,364,168
   52 IFNO$<>"" AND NO=0 THENPRINTS3$:GOTO34
   54 IFNO>28 OR NO<21 THENPRINT"I don't understan
d.":GOTO34
   56 IFNO>24 THEN NO=NO-4
   58 NO=NO-21
   60 IFNO>0 AND PD>0 THENPRINT"You have fallen in
to a pit.":GOTO612
   62 IFNO>0 AND OB%(30,0)=CP THEN142
```

```
   64 IFGF=0 AND CP=18 AND NO=1 THENPRINT"The gate
 is locked!.":GOTO34
   66 IFP%(CP,NO)=0 AND CP<>1 THENPRINTIM$:GOTO34
   68 IFP%(CP,NO)=0 THENPRINT"You can't go that wa
y  .":GOTO34
   70CP=P%(CP,NO):GOTO26
   72 IFNO$="" THENPRINTI1$:GOTO34
   74 IFOB%(NO,0)=-1 THENPRINT"You've already got
it.":GOTO34
   76 IFNO=0 THENPRINTS3$:GOTO34
   78 IFCP=18 AND NO=31 THEN88
   80 IFNO<>37 OR CP<>29 THEN86
   82 IFOB%(17,0)<>-1 THENPRINT"You need a contain
er.":GOTO34
   84 OB%(17,0)=0:GOTO96
   86 IFOB%(NO,0)<>CP THENPRINTS1$:GOTO34
   88 IF(NO>18 AND NO<32) OR NO>49 THENPRINT"It's
too heavy.":GOTO34
   90 IFNO=12 THENGOSUB 140
   92 IFZZ>3 THENPRINT"Your hands are full.":GOTO3
4
   94 ZZ=ZZ+1
   96 PRINT"OK":OB%(NO,0)=-1  :GOTO34
   98 PRINT"You are carrying:":ZZ=0
  100 FORI=1TOLO:IFOB%(I,0)=-1 THENPRINT OB$(I):ZZ
=ZZ+1
  102 NEXTI:IFZZ=0THENPRINT"Nothing"
  104 GOTO34
  106 PRINT"Points are scored by leaving valuables
 at the mouth of the tunnel."
  108 GOSUB378:GOTO34
  110 IFNO=0 AND NO$<>"" THENPRINT"What's a :=":PR
INTN1$"?":GOTO34
  112 IFNO=0 THENPRINT"Huh?":GOTO34
  114 IFOB%(NO,0)<>-1 THENPRINT"You have no ";N1$:
GOTO 34
  116 IFNO=35 THEN OB%(35,0)=0:NO=32
  118 IFNO=15 THENPRINT"You can't. It's stuck to y
our hand.":GOTO34
  120 OB%(NO,0)=CP:ZZ=ZZ-1
  122 IFNO=17 THEN OB%(17,0)=0:OB%(38,0)=CP:PRINT"
Crash!":GOTO34
  124 IFOB%(12,0)=OB%(30,0) THENPRINTW1$:OB%(30,0)
=0:GOTO34
  126PRINT"OK":GOTO34
  128 IFCP=31 THENPRINT"Read the medallion.":GOTO3
4
  130 IFCP=7 THENPRINT"Try prime numbers.":GOTO34
  132 IFJ=90 AND OB%(41,0)=0 THENPRINT"Some music
would be nice.":GOTO34
  134 PRINT"Try examining things.":GOTO34
  136 IFCP=43ORCP=44THENCP=87-CP:GOTO26
  138 PRINTIM$:GOTO34
  140 PRINT"The snake bites you.":BI=BI+8:RETURN
  142PRINT"The panther pounces on you.":GOTO612
  144 IFOB%(32,0)THENRETURN
  146 OB%(32,0)=OB%(35,0):OB%(35,0)=0:RETURN
  148FORI=1TOOB%(31,1):NEXT:GOSUB144:OB%(31,1)=100
:RETURN
  150 FORI=1TO54:IFOB%(I,0)=13THENOB%(I,0)=10
  152 NEXT:RETURN
  154 IFNO=31ANDCP=18THEN162
  156 IFNO=31THENPRINT"I see no gate here.":GOTO34
  158 IFOB%(NO,0)<>-1THENPRINTS4$:GOTO34
  160 PRINT"That's not necessary.":GOTO34
  162 IFGF=1THENPRINT"It's already open.":GOTO34
  164 IFOB%(4,0)=-1THENGF=1:PRINT"The gate swings
open.":GOTO34
  166 PRINT"You need a key to open the locked gate
.":GOTO34
  168 IFNO<>46OROB%(46,0)<>-1THEN110
  170 OB%(46,0)=17:ZZ=ZZ-1:PRINTJA$:GOTO34
  172 PRINT"Try 'Push'.":GOTO34
  174 IFOB%(NO,0)<>-1THENPRINTS4$:GOTO34
  176 IFNO<>2ANDNO<>16ANDNO<>18ANDNO<>5THENPRINT"T
here's no writing.":GOTO34
  178 PRINT"It says:":PRINT:IFNO<>2THEN186
  180 PRINT"At half time, Manchester United were"
  182 PRINT"leading Duckworth Rovers by an easy"
  184 PRINT"three goal margin. Ray Davies,leading
  scorer for Duckworth this season, was .."
  185 PRINT:PRINT"The rest of the paper is tattere
d and  torn, and can no longer be read.":GOTO 34
  186 IFNO=18THENPRINT"Felines ":PRINT"Have":PRINT
"enemies":GOTO34
  188 IFNO=16THENPRINT"Fermented juice is alexipha
rmic":GOTO34
  190PRINT"Take the first six letters":PRINT"Throw
 away the left half"
  192PRINT"Double the middle":PRINT"Turn it round"
:GOTO34
  194 IFOB%(NO,0)<>-1THENPRINTS4$:GOTO34
  196 IFNO<>13THENPRINTS2$:GOTO34
  198 IFOB%(13,0)<>-1THEN210
  200 ZZ=ZZ-1:PRINT"Yuk!It tastes terrible":OB%(13
,0)=0:GOTO34
  202IFOB%(NO,0)<>-1ANDOB%(NO,0)<>CP THENPRINTS1$:
GOTO34
  204 IFNO<>12ANDNO<>29ANDNO<>30ANDNO<>41THEN218
  206 IFNO=30THEN142
  208 IFNO=12THENGOSUB140:GOTO34
  210 IFOB%(13,0)<>-1THENPRINT"You have no food":G
OTO34
  212 IFOB%(29,0)<>CP THENPRINT"What turtle?":GOTO
```

```
34
  214 ZZ=ZZ-1:PRINTW5$:OB%(13,0)=0:TG=1:GOTO34
  216 IFOB%(NO,0)<>-1ANDOB%(NO,0)<>CP THENPRINTS1$
:GOTO34
  218 IFNO<>12ANDNO<>41ANDNO<>29ANDNO<>30THENPRINT
"It isn't alive.":GOTO34
  220 IFNO=12THENGOSUB140:GOTO34
  222 IFNO=30THEN142
  224 PRINT"It's immortal.":GOTO34
  226 IFNO<>22ANDNO<>11THEN230
  228 IFCP=21ANDP%(21,1)=0THENP%(21,1)=9:PRINT"You
've broken through!":GOTO34
  230 IFOB%(12,0)=CP THENGOSUB140:GOTO34
  232 PRINT"Nothing happens.":GOTO34
  234 PRINT"You don't have enough charisma.":GOTO3
4
  236 PRINT"Try 'open'":GOTO34
  238 IFNO<>51THENPRINT"No effect.":GOTO34
  240 IFCP<>22THENPRINT"What mirror?":GOTO34
  242 IFMI THEN P%(22,1)=4-P%(22,1):PRINT"It rolls
 easily.":GOTO34
  244 PRINT"It's stuck.":GOTO34
  246 PRINT"Try 'Use'":GOTO34
  248 IFNO<>39ANDNO<>37THENPRINT"Express that anot
her way.":GOTO34
  250 IFOB%(NO,0)<>-1THENPRINTS4$:GOTO34
  252 IFNO=39THEN258
  254 IFCP<>22THENPRINT"There's no use for oil her
e.":GOTO34
  256 MI=1:PRINT"The rollers are now oiled.":GOTO3
4
  258 IFOB%(15,0)+1THENPRINT"Your nails are nice a
nd clean now.":GOTO34
  260 OB%(15,0)=CP:ZZ=ZZ-1:PRINT"The statuette sli
ps from your grasp.":GOTO34
  262 IFNO<>36THENPRINTS2$:GOTO34
  264 IFOB%(36,0)<>-1THENPRINT"You have no wine.":
GOTO34
  266 PRINT"Glug-glug-glug":OB%(36,0)=0:OB%(17,0)=
-1
  268 IFBI>0THENPRINT"Aahh...it cures the snakebit
e.":BI=0
  270 GOTO34
  272 PRINTWA$:GOTO34
  274 IFNO<32ORNO>35THENPRINTIM$:GOTO34
  276 IFOB%(33,0)<>-1THENPRINT"You don't have a ma
tch.":GOTO34
  278 IFNO=33THENPRINT"The matches burn brightly."
:ZZ=ZZ-1:OB%(33,0)=0:GOTO34
  280 IFNO<>34THEN290
  282 IFOB%(34,0)=-1THENGOSUB148:PRINT"You are blo
wn to bits!":GOTO612
```

```
  284 IFOB%(34,0)<>CP THENPRINTS1$:GOTO34
  286 IFCP=13THENP%(13,2)=24:P$(13)=P$(9):CP=11:GO
SUB150
  287 PRINT:PRINT"Booommmm!!!!"
  288 OB%(34,0)=0:GOSUB148:GOTO34
  290 IFOB%(35,0) THENPRINT"It's already lit.":GOT
O34
  292 IF OB%(32,0)=-1 THEN OB%(32,0)=0:OB%(35,0)=-
1:PD=0:GOTO26
  294 PRINT"You haven't got a torch with you.":GOT
O34
  296 IF NO<>48 THENPRINT"What ?":GOTO34
  298 IF OB%(48,0)<>-1 THENPRINTS4$:GOTO34
  300 IF CP>35 THENPRINTGB$:OB%(41,0)=CP:GOTO34
  302 GOSUB 148 : CLS:PRINT:PRINT:PRINT"Cave-in!!!
!":GOTO612
  304 IF NO<>1 THEN314
  306 IF CP=21 THENPRINT"The South wall is badly e
roded.":GOTO34
  308 IF CP=17 AND P%(17,3)=0 THEN P%(17,3)=34: PR
INTSP$:GOTO34
  310IFCP=34 AND P%(34,3)=0 THEN P%(34,3)=35:P%(35
,2)=34:PRINTSP$:GOTO34
  312 PRINT"You find nothing special.":GOTO34
  314 IFNO=31 AND CP=18 THEN320
  316 IFNO=37 AND CP=29 THENPRINT"It's just oil.":
GOTO34
  318 IFOB%(NO,0)<>CP AND OB%(NO,0)<>-1 THENPRINTS
1$:GOTO34
  320 IFNO=2 OR NO=16 OR NO=18 THEN174
  322 IFNO=13 THENPRINT"It's not fit for human con
sumption."
  324 IFNO=40 THENPRINT"It's Topaz.":GOTO34
  326 IFNO=9THENPRINT"It's malachite.":GOTO34
  328 IFNO=41THENPRINT"It's made of gold.":GOTO34
  330 IFNO=10THENPRINT"It's lapis lazuli.":GOTO34
  332 IFNO=42THENPRINT"It's pyrite.":GOTO34
  334 IFNO=12THENGOSUB140:GOTO34
  336 IFNO=30THEN142
  338 IFNO=1THENPRINT"It's embroidered with gold t
hread.":GOTO34
  340 IFNO=15THENPRINT"It glistens.":GOTO34
  342 IFNO=46ANDOB%(46,0)=17THENPRINT"It's pointin
g towards the West.":GOTO34
  344 IFNO=50THENPRINT"It contains sacred oil.":GO
TO34
  346 IFNO=51THENPRINT"It's on rollers.":GOTO34
  348 IFNO=52THEN366
  350 IFNO=20ANDOB%(33,0)=0THENPRINTKN$:OB%(33,0)=
43:GOTO34
  352 IFNO=20 AND OB%(4,0)=0 THENPRINTKN$:OB%(4,0)
=43:GOTO34
```

```
  354 PRINT"It's just:=":PRINT:PRINTOB$(NO)".":GOT
O34
  356 IFNO<>34THENPRINT"I don't know how to do tha
t.":GOTO34
  358 IFOB%(3,0)=-1ANDOB%(6,0)=-1ANDOB%(14,0)=-1TH
EN362
  360 PRINT"You aren't holding all the ingredients
.":GOTO34
  362 OB%(3,0)=0:OB%(6,0)=0:OB%(14,0)=0:OB%(34,0)=
-1:ZZ=ZZ-2:PRINT"Done.":GOTO34
  364 PRINT"Try 'make' ":GOTO34
  366 PRINT"Which compartment number?";:PROCINFO:N
O=VAL(CM$):IF NO=0 OR NO >99 THEN34
  368 IFNO=13ANDOB%(49,0)=0THENOB%(49,0)=7:GOTO374
  370 IFNO=71ANDOB%(8,0)=0THENOB%(8,0)=7:GOTO374
  372 PRINT"That compartment is empty.":GOTO34
  374 PRINT"Something fell out.":GOTO34
  376 GOSUB378:GOTO388
  378 J=0:FOR I = 1 TO LO:IFOB%(I,0)=36 THEN J=J+O
B%(I,1)
  382NEXTI
  383 PRINT
  384PRINT"You have scored ";J;" points out of 100
.":IFJ<100THENRETURN
  386 CLS:PRINT:PRINT:PRINT"Well done!!"
  388 END
  390 DEF PROCINPUT
  391PRINT:PRINT"What now ? ";:PROCINFO:PRINT:IF B
I>0 THEN BI=BI+1
  392 NO$="":VB$="":VB=0:NO=0:H=0
  394 CM=LEN(CM$):FORI=1TOCM:IFMID$(CM$,I,1)=" "TH
EN H=I-1
  395 NEXT
  396 IFH=0THENH=LEN(CM$)
  397 IFH=1 THEN V1$=CM$:GOTO 399
  398V1$=LEFT$(CM$,H)
  399VB$=LEFT$(V1$,3):FORJ=1TONV:IFVB$(J)=VB$THENV
B=J
  400 NEXTJ
  402 IFVB>0THEN406
  404VB=+1:N1$=V1$:GOTO 410
  406 IFLEN(V1$)+1>LEN(CM$)THENNO=0:ENDPROC
  408 N1$=RIGHT$(CM$,LEN(CM$)-1-LEN(V1$))
  410 NO$=LEFT$(N1$,3):FORI=1TONN:IFNO$(I)=NO$THEN
NO=I
  411 NEXTI:IFNO=35 THEN NO=32
  412 ENDPROC
  414 CLS:PRINT:PRINT::IFCP=16 THENT=T+1:IFT>2 THE
NPRINTDR$:IFT>3 AND RND(1)<T*.1 THENGOSUB144:T=0
  416IFOB%(35,0)+1>0 AND CP<35 THENPRINTWD$:PD=1:R
ETURN
  418 PRINT"You're "P$(CP):PD=0
  419 PRINT
  420 VB$="You can see :"
  422 FOR I = LO TO 1 STEP-1:IF OB%(I,0) =CP THENP
RINTVB$:PRINTOB$(I):VB$=CHR$(11)
  424NEXTI
  426 FL=0
  427 PRINT
  428 PRINT"You can go:    ":FORI=0TO3:IFP%(CP,I)<>
0THENPRINTD$(I);"   ";:FL=1
  430NEXTI
  432 PRINT:PRINT
  434 IFBI>12 THENPRINT"  >The bite's throbbing<"
  436 IFBI>23 THENPRINT"  >You're getting dizzy<"
  438 IFBI>34 THENPRINT"  >It's hard to breathe<":
IFBI>42THEN612
  440 IFCP<>18 THENRETURN
  442 IFGF=1THENPRINT"The gate is open.":RETURN
  444 PRINT"The gate in the grill is locked.":RETU
RN
  445 DEFPROCVARIABLES
  446 NN=54:NV=32:P=51:LO=54:DIMP$(P),P%(P,3),OB$(
LO),OB%(LO,1),VB$(NV),NO$(NN)
  448 P$(1)="in a storeroom."+CR$+"The walls are m
ade of concrete."
  450 DATA18,0,5,0,25,33,8,12,0,7,31,0
  452P$(2)="in a dusty passageway."
  454 P$(3)="in the quarters of Princess Anka."
  456 P$(4)="in the King's harem (lucky old   you)
."
  457DATA22,0,0,0,0,0,21,1,6,15,6,19
  458 P$(5)=P$(1)
  460P$(6)="in a twisty little tunnel (again)"
  462 P$(7)="in the jewelry niche."
  463DATA3,0,0,0
  464 P$(8)="in an artist's studio."
  465DATA0,0,0,2
  466 P$(9)="crawling over a jumble of broken rock
."
  467 DATA21,10,0,0
  468 P$(10)=P$(6):
  469 DATA9,6,6,6
  470 P$(11)="in an old,old tunnel, feeling alllos
t and neglected."
  471 DATA0,20,13,0
  472 P$(12)="in an ancient library."
  473 DATA0,0,2,0
  474 P$(13)=P$(11)+CR$+"A thick brick wall blocks
 the way."
  475 DATA0,0,0,11
  476 P$(14)=P$(6):
  477 DATA19,19,20,19
  478 P$(15)=P$(6):
```

```
  479 DATA6,6,6,10
  480 P$(16)=P$(6):
  481 DATA14,6,6,6
  482 P$(17)="in a wine closet."
  483 DATA0,0,30,0
  484 P$(18)=P$(11)+CR$+"A metal grill blocks the
way."
  485 DATA35,1,0,0
  486 P$(19)=P$(6)
  487 DATA6,16,6,6
  488 P$(20)=P$(6):
  489 DATA11,16,16,16
  490 P$(21)=P$(1):
  491 DATA0,0,0,5
  492 P$(22)="King Kaleb's bedroom."
  493 DATA0,0,0,31
  494 P$(23)="in the slaves' quarters.  The    sla
ves' halves are somewhere else."
  495 DATA26,27,0,0,28,0,25,13
  496 P$(24)="at the West end of the Temple."+CR$+
"An ugly hole is in the West wall."
  498 P$(25)="at the East end of the Temple."
  499 DATA29,2,0,24
  500 P$(26)=" in the warriors' quarters."
  501 DATA0,23,0,33
  502 P$(27)=" in a stable that, well, smells  a l
ittle bit."
  503 DATA23,0,0,0
  504 P$(28)="in the high priest's vestry."
  505 DATA0,24,0,0
  506 P$(29)="in the shrine of Isis."
  507 DATA0,25,0,0
  508 P$(30)="in what was once used as the      kit
chen in older days."
  509 DATA0,0,33,17
  510 P$(31)="in an antechamber."
  511 DATA33,32,22,3
  512 P$(32)="in the throne room."
  513 DATA31,0,0,0
  514 P$(33)=P$(2)
  515 DATA2,31,26,30
  516 P$(34)="in a secret compartment (oooh)."
  517 DATA0,0,17,0
  518 P$(35)=P$(11)+CR$+"You see daylight to the N
orth."
  519 DATA36,18,0,0
  520 P$(36)="at the mouth of the tunnel."
  521 DATA37,35,0,0,51,36,0,0
  522 P$(37)="at the road's end."+CR$+"A mountain
is to the south."
  524 P$(38)="in a dense, gloomy, dark forest."
  525 DATA38,39,38,38
  526 P$(39)="on a familiar old path made by   hor
ses many, many years ago."
  527 DATA38,40,38,38,39,41,38,38
  528 P$(40)="at the end of a path with forest sur
rounding you in all directions."
  530 P$(41)=P$(38)+CR$+"To the south there seems
to be light."
  531 DATA40,42,38,38
  532P$(42)="in the middle of a clearing, witha fa
miliar bridge to the south."
  534 DATA41,43,0,0,42,0,0,0:
  535 P$(43)="on the north side of the bridge."
  536 P$(44)="on the south side of the bridge."
  537 DATA0,45,0,0
  538 P$(45)="at a great crossroads."
  539 DATA44,48,47,46
  540 P$(46)="on the great west road."
  541 DATA0,0,45,46
  542 P$(47)="on the great east road."
  543 DATA0,0,47,45
  544 FORI=48TO51:P$(I)="walking on the great sout
h road.":NEXT
  546 DATA45,49,0,0,45,50,0,0,45,51,0,0,45,37,0,0
  548 FORI=1TOP:FORJ=0TO3:READP%(I,J):NEXTJ,I
  550 DATA28,10,37,0,1,0,0,10,28,0,5,0,4,10,0,10,1
3,0,18,10,0,0,36,0,30,0,21,0
  552 DATA8,10,12,0,0,0,34,10,32,0,43,0
  554 DATAAn Ephod,A Scrap of Newspaper,A Keg of C
harcoal
  556 DATAA Silver Key,A Parchment Scroll,A Keg of
 Saltpetre
  558 DATAA Platinum Chastity Belt,A Ruby Earring
  560 DATAA Green Pebble,A Blue Stone,,A Vicious C
obra,A Shrivelled Carrot
  562 DATAA keg of Sulphur,A Jade Statuette,An Old
 Medical Book,An Empty Bottle
  564 DATAA Gold Medallion,A Golden Throne,A Dead
Knight (remember me ?),27,10
  566 FORI=1TO20:READOB%(I,0):READOB%(I,1):NEXT
  568 FORI=1TO20:READOB$(I):NEXT
  570 DATA31,0,0,20,35,0,0,0,0,0,0,0,17,0,0,0,0,0,
3,0,11,10,0,10,13,0,30,0,8,0
  572 DATA23,0,26,0,27,0,32,0,0,0,29,0,22,0,7,0,22
,0,3,0
  574 DATAA Giant Turtle
  576 FORI=29TO54:READOB%(I,0):READOB%(I,1):NEXT
  578 DATAA Hungry Panther,A Gate,An Old Torch
  580 DATASome Matches,Three Kegs of Gunpowder,A S
hining Torch,A Bottle of Wine
  582 DATAA Bottle of Oil,Some Broken Glass,A Jar
of Nail-Polish Remover
  584 DATASome Gravel,A Bird,A Gold Nugget
```

```
  586 DATAA Wooden Spoon,A Block of Marble,A set o
f manacles,A Rusty Javelin
  588 DATAStraw and Dung,A Brass Clarion,A Satin R
ibbon,A Marble Font
  590 DATAA Huge Mirror,100 Small Compartments
  592 DATAA King-size Bed,A Triclinium
  594 FORI=29TO54:READOB$(I):NEXTI
  596 DATAEPH,NEW,CHA,KEY,SCR,SAL,BEL,EAR,PEB,STO,
WAL,COB,CAR,SUL,STA,BOO,BOT
  598 DATAMED,THR,KNI,NOR,SOU,EAS,WES,N,S,E,W,TUR,
PAN,GAT,TOR,MAT,GUN,TOR,WIN
  600 DATAOIL,GLA,REM,GRA,BIR,NUG,SPO,BLO,MAN,JAV,
STR,CLA,RIB,FON,MIR,COM,BED
  602 DATATRI,GO,GET,LOO,INV,SCO,DRO,HEL,QUI,CRO,T
AK,OPE,MOV,REA,EAT,FEE,KIL,HIT
  604 DATA CHA,UNL,PUS,REM,USE,OIL,DRI,BRE,LIG,PLA
,EXA,KIC,MAK,MIX,THR
  606 FORI=1TONN:READNO$(I):NEXT:FORI=1TONV:READVB
$(I):NEXT
  608 DATA North,South,East,West
  610 FORI=0TO3:READD$(I):NEXT
  611 ENDPROC
  612 FORI=1TO2000:NEXT:PRINT:PRINT"You're dead!!"
:GOTO 376
  613 DEF PROCINFO
  614 CM$=""
  615 PRINT "*";CHR$(8);
  616 Z$=GET$
  617 Z=ASC(Z$):IFZ>95ANDZ<>127THEN616
  618 ZL=LEN(CM$):IFZL>28THEN622
  619 IFZ=127THEN624
  620 IFZ>31THENCM$=CM$+Z$:PRINTZ$;:GOTO 615
  622 IFZ=13ANDZL>0THENPRINT" ":ENDPROC
  624 IFZ=127ANDZL>0THENCM$=LEFT$(CM$,ZL-1):PRINT"
 ";CHR$(8);CHR$(8);
  626 GOTO 615
  699 DEFPROCSTART
  700 CLS
  705 PRINT:PRINT:PRINT"            ****Tunnel Advent
ure***"
  710 *FX4,1
  799 ENDPROC
```

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.

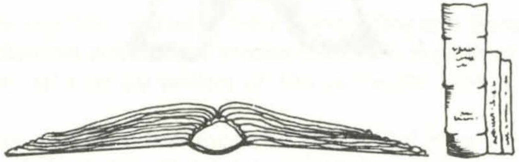
The December 1980 issue of *Byte* magazine, the one Daley Thompson does weight training with, is mainly devoted to adventuring, and features a whole host of excellent articles by many of the top authors around at the time, including Scott Adams, P. Lebling, Bob Liddell, and many more. A great issue, 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

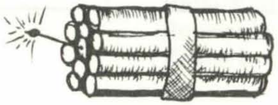
or
Beebug
Sheridan Williams/David Graham
P.O. Box 50
St. Albans
Hertfordshire

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.



EXPLORING ADVENTURES ON THE BBC MODEL B

The three adventures in this
book are available on a
cassette at £7.95, from
all good computer stores
and bookshops, or in case
of difficulty, direct
from the publisher.

Send your cheque/
postal order to:
Gerald Duckworth & Co Ltd
The Old Piano Factory
43 Gloucester Crescent
London NW1

and they will be sent to you
post-free

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.

Adams, Scott : 4, 5, 6, 28, 29, 30
Adventure : 1, 4, 21
Asc command : 65
Attack verb : 170
Bears : 102
Bottles : 107
Break verb : 190
Butterfield, Jim : 4, 9
Castlemaze adventure : 211
Chop verb : 164
Chr$ command : 65
Climb verb : 166
Close verb : 152
Contents : v
Creating adventures : 115
Cross verb : 148
Crowther, Willie : 3, 8, 10, 21
Cut verb : 164
Data command : 54
Data validation : 104
Death! : 106
Dialogue : 44, 45
Dim command : 73
Drink verb : 158
Drop verb : 144
Dungeons and Dragons : 35
Eat verb : 154
Else command : 56
Examine verb : 188
Feed verb : 156
For command : 66
Gargoyle : 108, 110
Get command : 56
Get verb : 140
Go verb : 138
Gosub command : 69
Goto command : 69
Hassett, Greg : 6
Hazards : 85, 86
Help verb : 199
Hit verb : 170
If command : 56
Input command : 51, 53
Input subroutines : 112, 113
Int command : 72
Introduction : vii
Inventories : 142
Jump verb : 190
Kill verb : 170
Left$ command : 61
Len command : 59
Light verb : 168
Load verb : 196
Logical Operators : 58
London adventures : 117
Look verb : 198
Lord of the Rings : 7, 10
Make verb : 172
Map drawing : 17, 20, 87, 89
Mazes : 95
Mid$ command : 60
Movement : 76, 78
Murder adventures : 128
Next command : 66
Noun data : 208
Objects : 42
Obstacles : 85
Offer verb : 160
Oil verb : 176
On command : 71
Open verb : 150
Panthers : 111
Personal Computer News : 236
Philosopher's Quest : 14
Pirate Adventure : 13, 28
Popular Computing Weekly : 2
Problem solving : 85, 100
Procedures : 70
Push verb : 192
Quit verb : 146
Read verb : 186
Reflect verb : 174
Repeat-Until command : 68
Restore command : 55

Return command : 69
Right$ command : 62
Rnd command : 72
Rub verb : 184
Save verb : 194
Score verb : 198
Screen Responses : 97
Solving adventures : 16
Space adventures : 122
Spray verb : 180
Stab verb : 178
Storylines : 82, 83, 84
Str$ command : 63
Subroutines : 58
Take verb : 200
Temple of Apshai : 11
Then command : 56
Throw verb : 182
Torches : 107
Traditional adventures : 131
Tunnel adventure : 221
Underground Adventure : 36, 135
Underground variables : 75
Underground verbs : 102, 137
Underground data : 202
User Groups : 237
Val command : 63
Variables : 52
Verb data : 208
Verbs : 42, 94
Vocabulary : 34
Wave verb : 162
Western adventures : 125
Woods, Don : 3, 8, 10, 21
Zork : 3, 9, 31, 32

DUCKWORTH
HOME COMPUTING

A POCKET HANDBOOK FOR THE BBC
by Peter Gerrard and Danny Doyle

The topics covered here include: ASCII tables, Assembler/Disassembler, Basic keywords, Basic error messages, Centronics standards, Conversion tables, FX calls, Flow charting, Hex/Dec/Binary conversions, Hyperbolic functions, Input/Output, Memory maps, Memory architecture, Machine Code interfacing, Machine Code instruction set, Musical note values, System calls. In short, everything you need to know about your machine.

Peter Gerrard, former editor of *Commodore Computing International*, is the author of two top-selling adventure games for the Commodore 64 and a regular contributor to *Personal Computer News, Which Micro? and Software Review* and *Commodore Horizons*.

Danny Doyle is Systems Performance Consultant for Sperry Ltd., and a regular contributor to *Commodore Computing International*. *£2.95*

BBC PROGRAMS 1
Edited by Nick Hampshire

This book provides you with a range of useful and exciting programs for the BBC Micro. Games, utilities, graphics and functional programs are covered. The games include an exciting version of Star Trek, a full length adventure game, Space Invaders, Battleships, Space Blaster, Brick Basher, and many others. Among the functional programs is a personal information retrieval package which enables you to create and manipulate up to 365 records. This is a basic book for every user of the BBC Micro.

Written by Carl Graham and edited by Nick Hampshire, publisher of *Commodore Computing International.* *£6.95*

Write in for a catalogue.

DUCKWORTH
The Old Piano Factory, 43 Gloucester Crescent, London NW1
Tel: 01-485 3484

Duckworth Home Computing

EXPLORING ADVENTURES ON THE BBC MODEL B

by Peter Gerrard

This is a complete look at the fabulous world of Adventure Games for the BBC Micro. Starting with an introduction to adventures, and their early history, it takes you gently through the basic programming necessary on the BBC 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 BBC, 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.

Peter Gerrard, former editor of *Commodore Computing International*, is the author of two top-selling adventure games for the Commodore 64 and a regular contributor to *Personal Computer News, Which Micro? and Software Review* and *Commodore Horizons.*

ISBN 0-7156-1795-8

9 780715 617953

Duckworth
The Old Piano Factory
43 Gloucester Crescent, London NW1

ISBN 0 7156 1795 8
IN UK ONLY £6.95 NET