Sinclair QL Adventures

A micro adventurer's handbook



Tony Bridge and Richard Williams



946408 66 1/Z1

Sinclair QL Adventures

A micro adventurer's handbook

Tony Bridge and Richard Williams

LONDON BOROUGH OF GREENWICH

GREENWICH LIBRARIES

RESERVE STOCK (P)

LSZ0016

GREENWICH LIBRARIES

3 8028 00852379 4

First published 1984 by:
Sunshine Books (an imprint of Scot Press Ltd.)
12–13 Little Newport Street
London WC2H 7PP

Copyright © Tony Bridge and Richard Williams, 1984

™ Sinclair QL, QL Microdrive and SuperBASIC are Trade Marks of Sinclair Research Ltd.
© The contents of the QL are the copyright of Sinclair Research Ltd.
™ Quill, Archive, Abacus and Easel are Trade Marks of Psion Software Ltd.

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 and/or otherwise, without the prior written permission of the Publishers.

British Library Cataloguing in Publication Data
Bridge, Tony
Sinclair QL Adventures.
1. Electronic games 2. Sinclair QL (Computer)
I. Title II. Williams, Richard
794.8'028'5404 GV1469.2

ISBN 0-946408-66-1

Cover design by Grad Graphics Design Ltd.
Illustration by Stuart Hughes.
Typeset by Paragon Photoset, Aylesbury.
Printed in England by Short Run Press Ltd, Exeter.

Contents

	Page
Introduction	1
PART 1: A HISTORY OF ADVENTURE	
1 Origins	7
2 The Background	11
3 Further Adventures	37
4 Adventure Generators	53
PART 2: THE ADVENTURE	
5 How to Use the Programs	57
6 The Adventure Generator — QLAG	67
7 The QL Adventure — QAD	99
8 QL Graphics	153
Index	157

SBN 1 | 2 | 3 LOC. COPY
946408 66 1 Z
EDITION/DATE PBK 1984
CLASS NO. 794
INVOICE JM 21 MAR 1986 £5.95

Contents in detail

Introduction
What makes a good adventure game — how to play adventures.

PART 1: A HISTORY OF ADVENTURE

CHAPTER 1
Origins
The origins of computer-assisted adventures — wargames — Dungeons & Dragons — adventures.

CHAPTER 2
The Background
Types of adventure problems you might come across — ZX80 and ZX81, adventure games for early Sinclair machines — Spectrum, colour and sound — text adventures, puppet method and first person — a typical adventure — Level 9 — Channel 8 — basics of adventures, words, a potted biography, monsters, spells.

CHAPTER 3
Further Adventures
Some more UK and US greats — Scott Adams — Infocom — *The Hobbit* — *Valhalla* — *Lords of Midnight* — QL adventures.

CHAPTER 4
Adventure Generators
Graphics and text generation — *Dungeon Builder* — *The Quill*.

PART 2: THE ADVENTURE

CHAPTER 5
How to Use the Programs
General information about the two programs — the QL Adventure Generator and how to use it — the QL Adventure and how to use it — Banes, Curse, Combat — notes on using the Generator — exits, identifying exits, bitwise logic, marking doors in QLAG, drawing doors in QAD — main variables in QLAG.

CHAPTER 6
The Adventure Generator — QLAG
Allows you to set up your own data, for as many games as you like, to be used in the QL Adventure.

CHAPTER 7
The QL Adventure — QAD
By loading the data already saved in QLAG, you can play your own text or graphics adventure.

CHAPTER 8
QL Graphics
A program to calculate the graphics data.

Thank you
To the Post Office for not losing any of our precious cartridges.
Mobil for not putting up the petrol prices while we wrote this book and:
Fiat Motors (UK) for stylish transport.
Sascha Kerry & Lucy for more cups of tea and Jilly for solo decorating. . . .


Digitized by the Internet Archive
in 2022 with funding from
Kahle/Austin Foundation

https://archive.org/details/sinclairqladvent0000brid

Introduction

Adventuring has become more than a pastime for bored computer programmers to play illicitly after hours, using million-dollar mainframes and very expensive computer time to fight dragons and snakes. It has become big business and, with the present explosion of home computer sales, has won the hearts of millions of computer hobbyists around the world.

Indeed, over the last few months, while writing this book, we have noticed a distinct turn away from arcade games toward adventure games.

Why is this? We believe that a plateau has been reached with arcade games. These started, back in the PET/Nascom days, with very expensive hardware running *Space Invaders*. The Space Invaders were Xs and Ws, but the lack of graphics didn't detract from the game: it was enough to see even this pale imitation of the pub classic (as well as not having to find 10p each time we wanted to play — conveniently forgetting about the £800 to buy the machine!).

This was the start. *Space Invaders* held sway for years, until eventually one or two intrepid souls, such as Clive Sinclair in the UK, finally brought the computer within the reach of teenagers. This led to a surge of new games for this dedicated audience.

First were the Space Invader clones such as *Galaxians*, *Gorf* and so on — then came *Pacman* and a million versions and variants. Each new level of expertise reached by the pioneers made it easier for the swelling numbers of games programmers who came after. The Pacman, Kong and Invader games remain an invaluable aid to budding programmers and there are also some new and highly original games which have made their appearance on the scene.

But we have still reached a plateau, though, however high in the sky it may be. At the present level of computer technology, it is not possible to do more than invent new ways of zapping aliens and running round mazes, or climbing ladders. As each new machine is announced, it introduces new ways of making it easier for us to program effects into our arcade games, but those effects remain broadly the same. Several attempts have been made at new techniques — at the time of writing, there is a certain interest in 3D games, some of which are more successful than others, and there is, from the late-lamented Imagine outfit, a hint of megagames, whatever they might be — but, in the end, arcade games can only take a leap forward with more realism.

Realism is what players are looking for in an arcade shoot 'em up game. The danger, no matter that it be vicarious, must appear to be real; the adrenalin must flow. And the hardware, at present levels of development, cannot present the player with more than a token involvement with the action on screen.

There is a great difference between playing a game in an arcade, with the great thumps, roars and whistles, and playing the same game on a micro. But even arcade games have an average life of only a few months — the smash hits, of course, survive much longer — but the game business is a hotbed of invention. Machines are with us now which use video discs to create a life-like environment for the player, who may be sitting in a dark cockpit in front of the screen, which might wrap completely round the player, drawing closer and closer to the action. Networks are also starting to make an appearance, allowing two or more players, at separate machines, to play against each other. In the near future, we can expect holography to be used, to give the player an illusion of 3D reality. After that, who knows? Maybe there will one day be some kind of tactile sensation from a game — smell, perhaps, or touch: after all, sight and sound both make a huge impact in the present arcade games, so what is left?

Most of these techniques — with the possible exception of networking, which would seem to be made for microcomputers — will be far too expensive and cumbersome to make it worthwhile implementing them on the home computer.

The next step towards realism in games for home computers can only be some sort of mental link between the machine and the human mind. That is obviously far in the future and, thrilling though it may be to imagine now, brain-to-brain contact must bring with it other things too dark to contemplate.

Is there another area of computer games in which the imagination can be allowed to run riot and create another world for the player? There is, and you will no doubt have guessed what it is. Adventures, of course!

Adventure games have progressed in much the same way as arcade games — they were around for a long time until the present generation of home micros dragged them into the light. Since then, because their scenarios exist largely in the collective imaginations of author and player, adventure programs have not yet reached that plateau on which arcade games now reside.

What makes a good adventure game?

The original adventure, *Colossal Cave*, was written for a mainframe computer. These computers did not, in those days, have VDUs (visual display units); instead hardcopy was taken on a printer linked to the

computer. So the game took the format of text dumped to the printer. The mythology has thus grown up among computer adventurers, that text is best. This is not so! At least, not necessarily.

High resolution graphics *can* add an awful lot of atmosphere, and occasionally the odd clue, but the price to pay is a dramatic erosion of available memory. Unfortunately, memory is at a premium in most home micros — as a rough guide, an adventure written in machine code might reasonably hold up to one hundred locations, with a couple of lines of text, and all the objects and puzzles associated with those locations. Less locations will allow more detailed descriptions, or more objects and puzzles. Adding graphics may well cut that figure by two-thirds.

A compromise is to have the adventure resident on disk, from which data may be called as the player comes across each particular location. This is the way most American adventures work, although cut-down versions are usually available on tape. Another way of creating a larger adventure than might otherwise be possible using tape, is to have two or more parts or 'chapters' to a game, each being loaded in from tape as required. This is rather inflexible, however, and not as good a solution as the disk.

Fortunately, memory is becoming cheaper and cheaper as the months go by, and now, with the QL, we have enough RAM (random access memory) to create a pretty large adventure — although there exists a kind of Parkinson's Law of Computer Memory, which is that the current requirements always expand to fill all the available memory! When the 1K ZX81 was released, wonders were done with the meagre RAM, and everyone waited expectantly for the 16K expansion, saying: 'That is all we will need, who could possibly use more than 16K?'. Of course, that was soon filled up and 32K packs began to appear. Then the Spectrum was released, and the 48K version outsold the 16K model many times over. 128K, though it seems more than enough for our purposes now, will certainly seem cramped in a few months' time as more and more complex programs are released. The possibility exists, too, of accessing the micro-drives of the QL in order to simulate a disk-based adventure. Thus the code for the program would be loaded into the QL, and a second car-tridge then used to load in data on locations as required..

So, we've seen that a good adventure can be *either* text only, *or* text-with-graphics, and is not dependent on either format for success. It is also possible to have graphics-only adventures, and these we call 'Arcventures'. Well-known programs that would qualify for this title, we believe (though many people would regard them as just more arcade games), are *Jet Set Willy* and *Manic Miner*, *Miner 2049'er* and *Jumpman*, among many other similar examples. While these have many of the features to be found in regular arcade games, they nevertheless have an element (indeed, it is the main element), of puzzle-solving. Space doesn't

permit us to go into details here, but Arcventures are covered in more depth in *Atari Adventures* by Tony Bridge (also published by Sunshine Books).

How to play adventures

How are adventures played? Let's have a quick look at a typical text adventure.

There are two ways in which the vast majority of adventures are conducted. The first way is for the player to instruct the computer in what to do. This is called the 'puppet method', in which the computer acts as the player's puppet. Thus, a typical exchange would be:

I am in a Graveyard. All around me are old Tombstones, partly obscured by a low creeping mist. I can see an eerie figure in the shadows. Its eyes glow a deep red, and it is now moving towards me. Obvious exits are North and West, and beside me is an open grave. What do I do now?

The required answer may be, in the first instance, GO NORTH, although this may usually, in any program worth its salt, be shortened to just NORTH, or even N. This is the most basic response — there are many others that will be required, depending on the circumstances prevailing at the time. The player may, for example, have picked up a sword at an earlier stage in the game, and may now instruct the computer to use it and attack the figure. Or, indeed, by SEARCHing the graveyard at this point some usable object may be found behind a tombstone. Also, no self-respecting adventurer would leave that open grave unexplored!

The second method of computer adventuring is the 'first person'. Thus:

You are in a Graveyard . . .

While there is not a lot of difference between the two, we tend to prefer the puppet method, as a more personalised story-line is built up over the course of the game. We will cover this in more detail in Chapter 2.

As we have come to expect from America's Infocom, they have come up with the world's first 'sixth' person adventure, in *Suspended* — see later for a discussion.

PART I

A History of Adventure

CHAPTER 1
Origins

Switch the light on, quick! There, that's better, now we can see a little further ahead into the gloom. The strange noises went away when the light went on, and now all is quiet. Ahead, an opening in the cave wall — let's go through and see what lies ahead for us.

Aha! What's this? A black rod lying on the damp floor of the cave. Go on, pick it up. Anything else around? No, that's it, so let's go further into the catacombs. In the next cave, an empty wicker cage lies discarded in one corner. Is it a trap? Try poking it with the rod — wave the rod at it — well, nothing happens, so we might as well pick it up.

Now the faint sound of a bird singing comes from the next cave. Quietly moving into the cave, we can see a bird happily singing on a rock not far away. That must be what the wicker cage is for! Right, let's catch the little blighter. But it flies away — something is frightening it. Well the black rod looks a bit menacing, so let's drop it. That's better, the bird has settled again and is merrily singing, oblivious to us creeping towards it with our cage. The bird is caught!

On again, pausing to pick up the rod (no doubt it will come in handy later), to the next cave, with lamp held high before us. Suddenly a vast shadow rears up before us from the gloom! A huge green snake fixes us with its eye and sways before us. There is no other exit apparent, so we must get past this beast in order to progress on our quest. Let's stop and think for a moment.

Waving the rod might work this time — but no, the snake isn't worried by that! How about using the rod as a pole vault? The snake is too big to vault over. The little bird is still singing away merrily — now, wait a moment here, maybe we can feed the bird to the snake. After all, the snake is probably hungry and this might keep it busy long enough for us to rush past and away. So let's free the bird and see what happens. Now would you believe it? The snake, frightened by the fluttering of wings, hisses violently and slithers away into the blackness, leaving us free to continue our quest.

This is a typical scenario from a computer-assisted adventure in which overt violence does not play a large part, but there are variations in which your sword-arm and fighting abilities take precedence over your reason-

ing powers. We'll look at some of these variations later, but for now let's examine what we mean by adventure.

To find the origins of computer-assisted adventure, we have to look first at another pastime from the pre-home computer age. (Remember those days, back before you spent your evenings glued to the latest version of *Zaxxon* or *Qix*?) A pastime as old as Chess itself, and nearly as old as another pastime!

Wargames

Wargames have been fought since chieftains had more than half-a-dozen men to their armies — after all, even the most basic of fighting manoeuvres need practising and what better way of doing that than making a game of it, thus giving some incentive to the proceedings.

Some very complex versions of wargames have been mounted, but the one that most people will recognise as such is the table-top war. This has been a popular form of recreation and a military training technique since the seventeenth century.

Wargamers love complex rules, and H. G. Wells, an avid wargamer himself, published many years ago a slim book called *Little Wars*, covering certain aspects of table-top battles. It soon became the bible for any serious devotee of the hobby.

During the 1960s, however, far more intricate books of rules to use with wargames made a commercial appearance and became very popular. These rulebooks covered the smallest details of the period concerned, such as uniforms, weapons and logistics — the main periods were ancient, medieval, Napoleonic and modern.

Dungeons & Dragons

There were within these categories many sub-categories, and one of these was medieval fantasy. Dave Arneson, of the Castle and Crusade Society in America, began a vast campaign and expanded the original rules to offer a complete environment for the players. These rules evolved, with the help of Gary Gygax, into one of the most successful games of the century, *Dungeons & Dragons*, published by Tactical Studies Rules. D&D presents the player with a highly stylised system of play, where nothing is left to chance and everything is played according to tables.

After the success of *Dungeons & Dragons*, it was only natural that many imitators should spring up, some making more impact on the scene than others. Probably the most enduring of these has been Ken Andre's game *Tunnels & Trolls*. It simplified the rules of D&D to a great extent, but featured much of what had become so popular with the earlier game.

As in D&D, a 'dungeon master' sets up a complex of caves (or in fact

whatever sort of scenario he desires) and then the party of fellow-players is let loose to fare as they will. Combat is moderated by the throw of dice, but T&T has no percentile dice. Spells are also inherent in T&T's system, progressing in hard-earned levels from the lowly 'knock-knock' (which opens locked doors) and 'Take that you fiend' (which uses IQ as a weapon), through such quaintly-named spells as 'Zombie Zonk' and 'Mutatum Mutandorum' to the ultimate, 'Born Again', which pretty well speaks for itself.

However the main development of T&T, which has endeared it to many thousands of fantasy game-players throughout the world and which makes it particularly interesting to us computer users, is the unique system of 'solo dungeons' which has taken T&T as its game system.

These dungeons take the form of illustrated books, each containing a ready-made adventure which can be played, according to the T&T rules, by one person. Really a series of multiple-choice actions in which the text acts as dungeon master, the books are a boon to the player who is unable to get together with other adventurers.

The great success enjoyed by these slim volumes is indicative of the great number of people who are unable to play full-blown RPG's (role-playing games), and who now look toward the computer as mediator, referee and dungeon master.

Because of the relative simplicity and clarity of the rules, T&T has found itself in great favour with the writers of combat-orientated computer adventures.

Adventures

In the mid-70s two enterprising chaps called Willie Crowther and Don Woods, whilst hunched over their huge mainframe computer, devised a game that they called *Colossal Cave*, or *Adventures*. They were very likely D&D fans, as the scenario for their game included a complex of caves, peopled with assorted strange beings, and liberally scattered with treasure of all kinds.

Like D&D, the player makes his way slowly through unknown territory, receiving information about his surroundings — but this time from the computer. A tireless referee and dungeon master, the computer is the ideal medium for the fantasy game.

Other games came along in the wake of *Colossal Cave*, probably the most successful being *Zork* which is the forerunner of many of the adventure games implemented for the home micros of the late 70s. The PET, Apple and Tandy machines were well-served with these for several years, while the original was passed in disk form around the circle of computer professionals. This free market was, however, rather black — the companies who owned the big computers that were used by their

employees for playing these illicit games were naturally rather upset at this use of expensive computer time!

The obvious attraction of these computer-moderated adventures to the home enthusiast is that they can be played at any time, and alone if need be (it's quite often illuminating to play these games with companions, each putting in their own contribution). Whilst the game can often take weeks or months to complete, the state of play can be saved to disk or tape and resumed at any moment.

The obvious drawback for the home enthusiast is that, in its original form, adventures can only be run on a mainframe computer costing a couple of hundred thousand pounds — not the usual living-room furniture!

With the advent of the cheap microcomputer, programs of the adventure genre came within reach of the home user. Now solo play became possible to the enthusiast not lucky enough to possess an IBM mainframe.

CHAPTER 2
The Background

At the time of writing, a new microcomputer is announced every other week. Some of these have turned out to be what has been called 'vapourware', that is, they never materialise. While this is a distressingly common phenomenon in the computer business, there are still machines about and very good machines at that. The initial release of software that eventually arrives from the manufacturer of the computer invariably includes an adventure program.

ZX80 and ZX81

Back in the prehistoric era of microcomputing, about five or six years ago, three machines dominated the scene. All were American, and all were very expensive — and, incidentally, still are! And all had implementations of adventures and *Zork* written for them. The Apple I, the Commodore PET, and the TRS-80 are still with us in one form or another, but the cost of these machines in those far-off days served to keep the adventure club rather exclusive.

And then along came Sir Clive Sinclair. After single-handedly transforming the digital watch market with the Black Watch, and the pocket calculator market with the Executive (though only by virtue of their low cost and attractive styling, not long-term reliability), it seems with hindsight inevitable that he should go on to concentrate on the computer market.

The ZX80 was, however, still aimed at the hobbyist and commercial software was practically non-existent. It was not until the launch of the ZX81 with its attendant sales pitch at the larger consumer market that home microcomputer sales took off and, with that, the creation of worthwhile software. Incidentally, this seems to be the exception that proves the rule that software helps sell hardware!

Adventure programs for the ZX81 are abundant and take several forms — some are purely text, while others are graphic in nature, with every shade in between. This follows the general trend in this area of software, with some authors staying with the traditional approach, and others treading a more innovative path. We've seen the beginnings of computer adventuring in the Crowther/Woods original and *Zork*, but

several other popular programs were written in the States in the mid-70s. As programmers became more efficient on their new toys, the ZX machines, so these older programs were adapted.

Many of the programs taken for conversion were originally published in David Ahl's *Creative Computing*, an American magazine. The most popular games included *Hammurabi*, often mis-spelt nowadays as Hamurabi. In the original you, the player, have to direct the eponymous administrator of Sumeria in managing the city. Given so many acres of fertile land, so many bushels of grain in storage, and so many people in the city, you have to balance all the ponderables in order to last a certain number of years. This type of game has been much adapted since its original appearance in *Creative Computing*, but the general term for the genre is 'management games'.

Many of the adventure programs around today are actually descended from these management games, in their careful balancing of several ponderables. *Star Trek*, first written in the late sixties in the flush of enthusiasm for the TV series, is a kind of *Hammurabi*, involving as it does a delicate juggling between weapons control and ammunition levels, with damage controls and repairs all taking their toll. This sort of game is ideally suited to playing on a computer, leaving the machine to take care of all the details of galaxy scanning and status reporting, allowing you to get on with zapping Klingons! The day may come, though, when someone writes a Star Trek program in which the aim is to make friends with the aliens . . .

Wumpus was also written many years ago and has survived, in fact flourished, to this day. The original game involved a search for the mythical Wumpus through a complex of squares. Using clues given to you by the computer, you eventually narrow down the choices, triangulating on to the final location of the beast. We'll meet descendants of all these programs, *Adventures*, *Zork*, *Hammurabi* and *Wumpus*, in future pages. The early Sinclair machines, the ZX80 and 81, had many versions of the games written for them and made commercially available on tape, or as listings in one of the many books written for the machines.

In particular, Artic Computing produced an extremely worthwhile series of adventures imaginatively called Adventures A, B and C (well, they did have the alternative titles of *Planet of Death*, *Inca Treasure* and *Ship of Doom*). They have survived since the early days of ZX-mania, and are very good adventures in the Crowther/Woods vein. There are two reasons for their deserved success. First, the programs are without exception fiendishly difficult, but ultimately rewarding. Second, they are written in machine code (and this at a time when most software authors were still struggling with Sinclair BASIC), their responses to player's input being correspondingly rapid.

Representative of the graphic approach to adventure is a personal

favourite of ours (and we make no apology for mentioning it!), J. K. Greye's *Catacombs*. The game draws heavily on both *Dungeons & Dragons* and *Wumpus* in its game mechanics. As the player explores an unmapped underground complex, various monsters are met. If the threshold of a room containing a monster is crossed, information on the beast is given by the computer. Then battle is joined, and blows exchanged until the player or the monster is finally defeated. As higher levels are reached, the battles become harder. Still available, *Catacombs* was a very good introduction for the ZX81-er, to this kind of adventuring.

Another, rather novel, approach to graphics is seen in Foilkade's *Fantastic Voyage*. Based on the science-fiction novel of the same name, the action takes place in, of all places, the human body! I suppose this is not strictly an adventure game, but it still possesses the same kind of game mechanics as many more traditional programs in its search for the ultimate — in this case, the blood clot in the brain that has to be destroyed.

These are only two of many adventure games for the early Sinclair machines, but all of these games have one disadvantage: being written for the ZX80 and 81, they are all, necessarily, in black and white — and mute. This is, of course, rather like saying that any black and white film is not worth watching, or any mono record is not good music. The classic programs will remain as classics, even in silent monochrome, and the lack of sound and colour may be, albeit rarely, a virtue.

Spectrum

The arrival of the Spectrum brought a veritable flood of software, 95% of it games-orientated. A sizeable portion of this is adventure in one form or another. Artic Computing have stuck with the same adventures that were written for the ZX81, merely transcribing them complete with black text on white paper.

Following tradition, two new adventures are called E and F (and subtitled *Golden Apples* and *Ground Zero*). With these programs, Artic have taken advantage of the Spectrum's features and present the text in multi-hued ink on black paper — very colourful!

Adventure E, *Golden Apples* (like A, B, C and D before it), is purely text, but the new adventure seems at first playing rather more traditional than the previous games. The program is written for the larger memory, so more detailed exploration of locations may be undertaken. It's a pity that 48K versions of the previous adventures could not be released by Artic — there would certainly be a large market for such an undertaking. Adventure F, *Ground Zero*, written with the aid of *The Quill* from Gilsoft, is a charming tale of nuclear destruction. You, as the protagonist, have to survive the final conflagration.

Other software houses have followed Artic in staying with the traditional approach, notable among these being Abersoft, Foilkade and Level 9. Whilst both Artic and Foilkade (with *Adventure 200*, a reference to the number of locations in the game) have written their own adventure scenarios, both Abersoft and Level 9 have taken Willie Crowther and Don Woods' classic and translated it to the Spectrum. Both have, however, added touches of their own.

Level 9, for example, in their *Colossal Adventure* extend the original to some 70 more locations. And two sequels, *Dungeon Adventure* and *Adventure Quest*, follow on from locations in *Colossal*. The three together make an interesting suite of games, extremely tough, but always logical and amusing.

There are, of course, other programs that rely on text, yet which like Artic and Foilkade do not follow the original ideas — their distant relatives are usually *Wumpus* and *Dungeons & Dragons* (and sometimes — as in Mikrogen's *Sorcerer's Castle* — Snakes and Ladders!). These include such programs as *The Orb*, from Quest Software, *Velnor's Lair*, from Neptune, and *Volcanic Dungeon* from Carnell. Although these last programs may contain limited graphics, such as floor plans or weapon lists, to enhance the text, they can still be thought of as text adventures.

The Spectrum, however, cries out for its colour and sound capabilities to be used, and many authors have indeed taken advantage of these features in producing their adventure programs.

Text adventures

Originally, as we have seen, adventure was played on the big mainframes. These machines did not have the luxury of a monitor screen, instead printing out their output (the reason for several terms in home computing today, such as PRINT and TAB). Thus graphics were out. Instead, the whole game was played out using reams and reams of printout paper.

There exist nowadays bands of reactionary adventurers, each defending their own favoured type of adventure. The devotees of the text variety believe that, without pictures making a preconceived notion of the scene in their mind, they can enjoy the adventure more, with their imagination the only limit to that enjoyment. Adherents to the graphic adventure believe that the artist's representation of the scene enhances the game. We love adventures of any kind — while enjoying watching a good artist at work, we find that each text adventure takes on a personality of its own as the imagination works to flesh out the bare bones of the descriptions (you'll find, however, that Infocom and one or two others are so verbose in their descriptions as to leave very little to the imagination!)

The typical text adventure begins with the description of a location. In

the original, this was outside a small wooden hut in the middle of a forest. This description may be rather terse: '*You are standing outside a small wooden hut.*' If memory is not constricted, the locations may have correspondingly more description, although most authors would then decide to include more locations. If the adventure is supplied on a disk, then each location may be stored and read by the program as required, and the description may be much more detailed.

After the description, be it long or short (and see the section dealing with Infocom adventures in Chapter 3), the computer asks the player what he wishes to do. There are two ways in which adventures may be conducted. After describing the location, the computer may then say: '*I see nothing of interest. What do I do now?*'. The adventure will then be played out from the point of view of the computer, and you, the player, will direct. The computer, in this case, becomes the adventurer's puppet. Again, see the Infocom section — as befits the most original adventure software authors around at the moment, they approach this kind of third person adventure in a completely fresh way, in their program *Suspended.*

First person adventures are subtly different however. In this variety, you will be the protagonist, and the computer will say: '*You see nothing. What will you do now?*'. This makes the whole thing much more personal and immediate, and we would normally prefer to play this kind of adventure.

The difficulty in describing the text adventure is in attempting to convey the essential flavour of the game without giving away the answer to a problem that may have been intriguing a reader for many weeks — and much of the fun in solving text adventures is working at a tough problem until you crack it.

A typical adventure

What follows now is a hypothetical situation, dredged up from my imagination, that might serve as an example of the way a typical situation might develop. In most adventures, the underlying objective is very often the acquisition of treasure in one form or another. I suppose that any object that can be picked up and carried around may be called treasure, but some objects, like keys or cans of oil (very good for de-rusting door hinges!) are of practical use in an adventure, while others, like gold coins, can be regarded as 'real' treasure, and add to the player's final score. In this extract, our imaginary player is having a good time trying to get all this treasure!

Take a look at this situation. You are in a long corridor, in a complex of caves. You have arrived here after collecting treasures and other objects. To find out what you are presently carrying, type INVENTORY (or usually just INV). The computer replies:

YOU ARE CARRYING:
THE MAGIC HELM
THE GOLDEN RING
THE BRASS SHIELD
THE COMIC
THE FIRE OPAL
THE BROWN GLOVES

YOU ARE IN A LONG CORRIDOR. TO THE WEST YOU SEE A DARK CAVERN.
WHAT NOW?

Well, we might as well go in — this can often prove dangerous, as the author will probably be testing us and will set all sorts of traps about his complex of caves. The next message is often:

YOU HAVE SLIPPED IN THE DARK AND BROKEN YOUR SKULL. DO YOU WANT ANOTHER GAME Y/N?

But we have been warned in advance of the lack of light in the cave, and we have not been told of any light source in the corridor, so it is a reasonable assumption that there will be light of some kind in the cave itself. Having first SAVEd our position (any reasonable program will allow you to do this before any risky steps are taken, thus making it easy, if killed, to resume quickly), we decide to take the risk of looking into the cave. We type:

W(est)

and the computer answers:

YOU ARE NOW IN THE DARK CAVE. A FAINT LIGHT GLOWS FROM THE ENTRANCE BEHIND YOU. BETTER GET SOME LIGHT QUICK.
WHAT NOW?

Try this:

READ COMIC (for some light relief, geddit?)

Ingenious, but no good, I'm afraid! The computer merely repeats its messages:

IT'S DARK IN HERE. BETTER GET SOME LIGHT.

This is not too good! Adventure programs often let you stumble about in the dark for only a short time or a couple of turns before deciding that you haven't the faintest idea of how to proceed and then dumping you unceremoniously in the nearest ice-cold underground stream, or causing you to bang into an overhanging rock. Whatever method it chooses, the program will often bring the game to a sudden conclusion.

But what's this — something else comes up on the screen:

YOU FEEL SLIMY TENTACLES WRAP AROUND YOUR LEGS.
IT IS A SMALL OCTOPUS.

Now the octopus may come in handy here. So:

GET OCTOPUS

And the computer comes back with:

UGH — NO THANKS — ITS ALL SLIMY.
WHAT NOW?
WEAR GLOVES
O.K. I AM NOW WEARING THE BROWN GLOVES.
WHAT NOW?
GET OCTOPUS
I HAVE THE OCTOPUS. MANY HANDS MAKE LIGHT WORK. A SOFT LIGHT EMANATES FROM AN INVISIBLE SOURCE.
YOU ARE IN A SMALL CAVE. THERE IS A SMALL WOODEN BOX HERE. TO THE NORTH YOU SEE AN UNDERGROUND STREAM FLOWING THROUGH A SMALL CRACK IN THE ROCK.
THROUGH THE CRACK YOU CAN JUST MAKE OUT SOMETHING GLITTERING IN THE DARK BEYOND.
WHAT NOW?

He! He! Some more treasure! Must get it! Flushed with justifiable pride at our success at using the octopus, we attack the next problem with gusto. How are we going to get at that gold in the next cave?

We can try the direct route, by typing:

N (for NORTH)
YOU CAN'T GO THAT WAY.
WHAT NOW?

That's a bit of a blow. The computer, darn it, doesn't seem very impressed with our previous brilliant deductions and is already putting more

problems our way! There must be a way through the crack. Maybe we are carrying too many goodies, and we can't squeeze through.

DROP ALL
O.K.
WHAT NOW?
N
YOU CAN'T GO THAT WAY.
WHAT NOW?

How about jumping in the stream, we might be able to swim or float through?

JUMP IN STREAM
YOU MUST BE JOKING — I CAN'T BREATHE UNDERWATER!
WHAT NOW?

Hmm, this is getting silly. Is there anything to help us in the cave? There's a little pile of treasure on the floor which we dropped a few moments ago, but none of that is useful to us now. Wait, there's a box, isn't there. How could we have forgotten that?

EXAMINE BOX
YOU SEE A SMALL WOODEN BOX.
THERE IS NOTHING UNUSUAL.

Computers can be very pedantic!

OPEN BOX
O.K. THE BOX IS OPEN. INSIDE YOU SEE AN AQUALUNG.

Aqualung? Underground stream?

GET AQUALUNG
O.K. YOU HAVE THE AQUALUNG.
WHAT NOW?

Well, what now? We have the aqualung and it's a fair chance that we will now be able to negotiate the stream.

But we want to take all our lovely treasure with us, don't we, and water has an annoying habit of flowing one way only. You can bet your last golden ring that we won't be able to get back once in that steam. So we'll have to go through the long chore of retrieving everything we want to take with us. The computer, though, has other ideas!

YOU ARE CARRYING THE AQUALUNG. YOU CAN ONLY CARRY ONE MORE THING.

Obviously, the aqualung is too bulky or heavy. Let's just recap on what is currently in the cave.

LOOK
YOU SEE:
THE MAGIC HELM
THE GOLDEN RING
THE BRASS SHIELD
THE COMIC
THE FIRE OPAL
THE OCTOPUS WHICH IS WATCHING YOU
YOU ARE CARRYING:
THE AQUALUNG
THE BROWN GLOVES WHICH YOU ARE WEARING
THERE IS ALSO A SMALL BOX WHICH IS OPEN

That's a lot of stuff to carry! Wait a moment, though, the box might be useful. We're told it is small, but maybe it will carry everything we want to take with us and maybe it will float!

Now we can try:

GET RING
O.K. YOU HAVE THE RING.
WHAT NOW?
PUT RING IN BOX
O.K.
WHAT NOW?
LOOK

and we get the same list as before, but with a difference at the end:

AND A SMALL BOX WHICH IS OPEN. IT CONTAINS THE GOLDEN RING

It works! Now we'll have to repeat the process with the rest of the treasure. After several moments typing, another LOOK informs us that the box holds all our hard-won treasure. The octopus we keep for the moment. Now let's try:

THROW BOX
WHERE?

IN STREAM
O.K. THE BOX IS CARRIED SWIFTLY FROM VIEW INTO THE CRACK IN THE ROCKS

Now let's do the same with the octopus. The computer tells us that the octopus has also disappeared from view. Now we'll type:

JUMP IN STREAM

and the computer tells us:

YOU MUST BE JOKING — I CAN'T BREATHE UNDERWATER!

So boring! We'll have to go through every motion for the computer's benefit.

WEAR AQUALUNG
O.K. YOU ARE WEARING THE AQUALUNG.
YOU ARE SUFFOCATING. QUICK DO SOMETHING!

Whoops! Forgotten something!

TURN ON AIR VALVE
O.K. THE AIR VALVE IS NOW ON.
WHAT NOW?
JUMP IN STREAM
O.K. YOU ARE CARRIED ON THE SWIFT CURRENT OF THE STREAM THROUGH THE CRACK IN THE ROCK, AND INTO A CAVE. THE OCTOPUS HAS GONE BEFORE. MANY HANDS HAVE MADE THE LIGHT WORK. ABOVE YOU IS THE BANK OF THE STREAM.

We breathe a sigh of relief. Now we must get up on to the bank of the stream.

U (for UP)
O.K.
YOU ARE IN A LARGE CAVE.
YOU SEE A SMALL BOX. IT IS OPEN AND CONTAINS MUCH TREASURE.

(But wait a moment before going to retrieve the treasure!)

YOU ALSO SEE:

A DRAGON ASLEEP. HIS SCALES GLISTEN IN THE LIGHT.
ALL THAT GLISTENS IS NOT GOLD! HE IS STIRRING.
WHAT NOW?

What a question! Well, what do we do now?

This is an example of a typical problem to be met in a traditional adventure. (I won't tell you which one, in case you are currently playing it) — a series of interlocking problems that eventually yield a result. That result may, as in our example, lead you into a deal of trouble, but of course the possibility is that you will end up with treasure.

Getting the treasure is not, however, the final story, as you have to return to a certain location to deposit the treasure and collect your points.

Level 9

We'll look now at some Level 9 adventures and, in particular, *Dungeon Adventure*. Any reader who finds himself currently in this particular adventure should skip the rest of this section! Those of you who may be thinking of buying this program should not despair — nothing crucial will be given away!

Dungeon Adventure is part of an excellent series written by Level 9 of High Wycombe, which starts with *Colossal Adventure*, based closely on the Crowther/Woods original *Colossal Cave* program, but adding some 70 locations to the end game. The first program in this series, called The Middle Earth Adventures by Level 9, is Adventure Quest. All the Middle Earth adventures inter-relate and form a large fantasy world which can be explored and enjoyed.

Snowball, the first of another Level 9 series, Silicon Dream, is set on a huge, five-mile long starship. With over 7000 locations (so Level 9 claim), and rampant, man-eating nightingales patrolling the corridors of the spaceship, the title probably refers to your chances of surviving! The Silicon Dream series promises to be the equal of Middle Earth.

Yet another title from Level 9 is *Lords of Time*, the first in another series — written by Sue Gazzard, it is a marvellous romp among nine time sectors, with the player witnessing a fascinating panoply of scenarios.

The opening scene of *Dungeon Adventure* (and the puzzles associated with it) is a very pure example of the classic adventure program, and will serve to lay some ground rules for adventuring in general.

The booklet which accompanies the cassette of *Dungeon Adventure* is typical of the quality of Level 9's output. A page of scene-setting, with a brief snatch of story concerning the siege of Minas Tirith, the death of the Demon Lord, and the protagonist's decision to rush off and search for the Lord's treasure in the Black Tower, prepares you for the start of the game. The rest of the booklet outlines the various commands you may

give to the computer, along with several hints for the adventurer. Finally, a unique touch — an envelope with which you may, if you get dreadfully stuck, beg Level 9 for a clue.

At the start of the Adventure you, the hero, awake cold and weaponless on a mudbank beside a large packing case, open at one end. There is also a piece of driftwood lying nearby. You see a stone bridge across a river, reaching from the granite cliffs above to the flat lands of the far bank. Now you may, of course, go charging off up on to the bridge, eager to get on with exploring the caves which you secretly know are up there. And here we get to our First Golden Rule of adventuring:

Look around every location in as much detail as possible

Adventures of the traditional kind, which we are presently engaged in, almost invariably started off with the player alone and defenceless outside some sort of building.

In this present instance, there are some 170 locations to be explored within the cavern complex that you will find over that bridge. Your chances of surviving to see more than about half a dozen of them are, however, depressingly remote! Very soon after starting off, you will find certain objects become necessary — objects which can only be obtained from the outside locations. That is, the locations which you will find on this side of the bridge.

You will find out all this after a couple of tries at the cavern complex — and now you can start a search for these vital objects. We know that there is nothing useful — yet! — to be found by going north across the bridge, so let's try going south.

But first of all, we must not forget our first rule of adventuring, and *look around as much as possible*! Now, if you remember, we are on a mudbank and we can see some driftwood and a large packing case. Let's pick up the wood. Different programs recognise different commands for this action, and we can try PICK, TAKE or GET, the usual ones. In fact, *Dungeon Adventure* will accept TAKE, so now we are the proud possessors of a lump of wood.

There doesn't seem to be much to do with it, so let's concentrate on something else. The packing case, if you remember, was lying with one end open, so it would be a good idea to go inside, wouldn't it? No, it wouldn't! Not yet, anyway! Writers of adventure programs are a sadistic lot, and would like nothing better than for you to walk wide-eyed and innocent into a trap — maybe just like this one. So let's try another keyword which you will often come across in traditional adventures such as this one — EXAMINE N (where N is the object to be EXAMINED).

So let's type in EXAMINE CASE and see what the computer has to tell

us. The case is enchanted! Not only that, but there is room for us to crawl in. So we know now that we will not be eaten or stepped on by some passing monster, and we can proceed unscathed for the moment.

Let's leave the case here, though. There are secrets to be learnt inside, of course, not least being the fact that the case is where you must deposit your treasure in order to gain points and win the game. The riddles to be solved in order to gain this information are fun to work out, and we don't want to spoil your fun at such an early stage.

Let's assume, then, that we've thoroughly explored the case and are back outside on the mudbank, with our lump of wood. We have decided, maybe after several painful attempts, that we are on a sticky wicket in trying to negotiate the dark caverns to the north of the bridge. So let's try going to the south from the bridge. We are playing the adventure as we write, so this is a voyage of discovery for all of us!

We are now on an east-west road south of the river across which runs the bridge. A gigantic orc's head is carved into the cliff north of the river, its tongue forming the bridge. A ruined tower stands on top of the cliff. Exits are EAST, WEST and SOUTH. This is the standard formula for text adventures — enter a location and a brief description is given with a list of possible exits. Actually, the descriptions of the various locations are rather detailed in Level 9's adventures.

Usually, another list is also given — that of the objects and/or monsters or other entities to be seen, and the things that you may be carrying. There is nothing to be seen lying around at this location, however, so we have to make up our minds which direction to take now. We have no clues here, so let's toss a coin and take the east road. The next location is again the east-west road, but now we are further along and we can see to the south a flat grassy plain, stretching as far as the eye can see.

This is one of those phrases that you will come to treat with respect. It usually means that if you once set foot in the location to which it refers, you will be lost for evermore — or at least until you give up in sheer despair!

So we will give that grassy plain a miss for the moment. However we can also see, so the computer informs us, a line of stepping-stones which leads to a small island in the water. A young girl with flowing locks sits on the island — how charming! So let's type GO NORTH. The computer informs us that we are now on the stepping-stones, and the girl is still sitting there on the island. We type GO NORTH again. Oh dear! The girl was a siren, and sings her siren song — we flounder into the treacherous waters of the river. The computer informs us that we have managed to get ourselves killed.

The program now dumps us back on the mudbank, and we have to retrace our steps back to the east-west road, just south of the bridge. This time, let's explore to the east. We type GO EAST, and the computer duly

tells us that we are now further east along the road, north of a steep treeless hill.

This treeless hill seems to be worth investigating: going south, we come to the side of the steep hill, which rises into the clouds. Rumblings emanate from above! We can go UP here, so let's do so. We are now in a circle of distorted monoliths, etched into grotesque figures by the acid rain. Lightning arcs overhead, revealing the stark horror of this haunted place, while thunder echoes from every crag. Sounds very cosy, doesn't it? If we instruct the computer to LOOK around, a host of fierce flames will spring forth — they are Rakshasa! Their leader floats forward and challenges us to a game — if you win the throw of the dice he'll make our spiritual strength greater in some way. Of course, we may lose . . .

We have to play and, after all, it sounds quite a bargain, doesn't it? Unfortunately, we need a way of loading the dice (these Rakshasa are pretty cunning), and we haven't found it yet, although I might say that it is to be found somewhere in this opening sequence.

As you may expect, we lose this particular game and once again find ourselves, after being asked if we want to play again, back on the mudbank — we're learning each time, though.

Let's get back to that stretch of road just north of the steep hill where we recently met the Rakshasa. We won't go back there again until we find the method of winning the dice game. So we continue west to see what awaits us there.

And now we see a vast field of poppies which stretches west as far as the eye can see — and where have we seen that phrase before? We must tread very carefully here! A somnolent perfume hangs heavily on the breeze. Nearby we can see a dry poppy seed lying on the ground. Let's pick it up, by typing TAKE SEED. And here we come to our Second Golden Rule of adventuring:

Everything has a purpose

The typical adventure program is so packed full that the programmer cannot afford to leave too many useless items lying around. Like all rules, however, this one can be broken, and you will come across the occasional red herring, which will have you rushing about trying to find some way to use it. And of course there will sometimes be a really horrid author who will delight in handing the unwary adventurer a bomb in disguise!

So although we can act on this golden rule, we must add a rider:

Exercise extreme caution at all times

Now, let's retrace our steps — we have a strange feeling about that field of poppies!

We can work our way back along the road, past the steep hill and the bridge, back to those stepping-stones. As we have a poppy seed with us now, we might be able to use it to fight that siren (we don't know how yet, but it'll be worth a try!). At the stepping-stones, we type GO NORTH, which get us on to the stones. Last time we were here, we got unceremoniously dumped in the water when we tried to proceed further north past the siren, so let's be a little more cautious this time!

We might as well try the poppy seeds now — we'll type THROW SEEDS and see what that does (we're condensing a lot of heartache here, and a lot of effort in finding the correct solution). The seeds, in falling, burst with loud explosions. Did they frighten the siren off?

To find this out, we have to give a command to the computer. Type LOOK, and the computer will obligingly inform you of what is at the location. Quite often, of course, the answer you get is 'I see nothing special', but the question is, like EXAMINE, well worth asking. LOOK should not be confused with REDESCRIBE (REDES or R for short), which will instruct the computer to do just that with the present location.

Back to our present problems. Having tried LOOK, we find that the siren is still there, sitting by the river! Now we'll introduce yet another useful word — LISTEN. Unpleasant things have a nasty habit of hiding behind trees or rocks out of sight of the computer, but can't help making slimy, squidgy noises which will give them away!

Ah! Now we're getting somewhere! The computer tells us that we have been temporarily deafened, and can't hear anything. Although she must still be singing away as a good siren should, we won't hear her, so it's probably safe now to creep past her . . . but she's got us again! The temporary effect of deafness must be very short-lived.

Here we are back again at the mudbank. Now we can waste no time in getting back to that pesky siren, not forgetting to collect the poppy seed!

Another word worth remembering, and which would help us in the future, is SAVE. Most good adventure programs will allow this word which means that you can save your present stage of play to a separate tape or disk. You should if possible do this before taking any risky initiative. We should have been more prudent and done this ourselves before attempting to get past the siren — we would now simply be able to type RESTORE and, once our old position was re-established, make a fresh attempt to cross the stepping-stones. As it is, we have to do a lot of typing to return, poppy seed at the ready, to that siren.

So, let's make this our Third Golden Rule of Adventuring:

Always save your position if in doubt

But here we are again, and we've dropped the seed, which duly explodes. We'll SAVE our position in case we are discombobulated yet again, and

this time let's get on with it and rush past her. After all, we know now what effect the explosions will have. And, it works! The siren flees in panic, seeing that her song has no effect on us. In fact, the program will at this point allow you up to four commands before the deafness wears off.

And this is another ploy that the better adventure program will use. Approach a problem in one way, and the result may be completely different from the outcome of another attempt.

We're now at the southern end of a small island — the far end is occupied by a viscious-looking willow tree with six long rubbery branches. A silver mirror lies nearby. We can TAKE the mirror, but if we try and get past the tree we will be killed. Why *six* branches, there must be a clue there!

Let's leave *Dungeon Adventure* now. This particular game is extremely rich — we've only covered a few locations and we haven't mentioned the resurrection procedure which you will need before playing the game for real, and several other important details which can be found in the opening scenario.

Actually this whole sequence, which consists of some 30 rooms, is only an introduction to the main adventure which takes place in the cave complex that we saw at the start. The original *Adventures* program started with the player standing outside a small building in a forest. This building contained several articles required for the adventure, such as keys, food and water — any treasure had to be brought back to the building. The very large network of locations at the start of *Dungeon Adventure* serves the same purpose as that small building in the original — but now in a far more complex way, which in fact is an intriguing mini-adventure in its own right.

After our first fumbling attempts at *Dungeon Adventure*, you should have gained an idea of what it feels like to play a text adventure (and feel motivated to play Level 9's adventures yourself).

There is little other British-written software to match Level 9's special blend of humour, complexity, literacy, puzzle-setting and sheer bravado. Their programs will have you alternately shaking with frustration and curled up with uncontrollable laughter. Sample, for instance, the moment in one of their adventures when the player is confronted with a tiny plant which whispers in a meek, pitiful voice, 'Water . . . Water'. Taking pity on the poor thing, the player waters it and then can but watch as the plant rears up to eight feet, and menaces the player, demanding, 'WATER, WATER!!'.

Channel 8

One other British software firm which *can* challenge Level 9 is Channel 8 Software, of Preston. Channel 8 has about ten adventures for the Atari,

Spectrum, BBC, Commodore 64, Dragon 32 and other machines. The catalogue has a long and honourable pedigree, having been originally written by Brian Howarth for the TRS-80 and Nascom machines way back in 1980, and marketed through Molimerx. A list of the intriguingly-named adventures follows:

The Golden Baton: Ever since the Golden Baton was stolen from the kingdom of King Ferrenuil, brave warriors and hard knights were sent far and wide through the world in search of this artefact. None ever returned . . . and now it is up to you!

The Time Machine: Alone in the fog on the moors, you, the news reporter for the Tulkingham & Dunsby Gazette, have set out to investigate the strange goings-on around the old house.

Arrow of Death: You have at last regained the Golden Baton, and returned it to King Ferrenuil. Recently, however, the Golden Baton has become a symbol of hatred and evil. With a sense of foreboding, you are now travelling to the palace. Surely a mere mortal, such as yourself, cannot stand up against the evil power that threatens the future of your land.

Arrow of Death Pt. 2: Having successfully completed Part 1, you will now need the Arrow in order to defeat Xerdon the Evil!

Escape from Pulsar 7: On board Pulsar 7, you are alone and being hunted by a savage creature, which has killed and eaten the rest of the crew.

Circus: Out of petrol on a dark, deserted country lane, you come upon a Circus in the middle of a field. The raucous sounds of laughter and merriment stop as soon as you approach . . . Darkness is falling, and it's going to be a long, long night!

Feasibility Experiment: You awake from a troubled sleep to find yourself the subject of an experiment by strange, disembodied aliens from an artificial world far beyond the rim of the Galaxy (where else?). They wish to test the human race, to find a heroic strain for their gene pool: they will test your heroism in an arena.

The Wizard Akryz: The evil Sorcerer has had enough of the pesky mortal who has foiled him at every move in his attempt to snaffle the Golden Baton. He has now dreamed up the ultimate plan — you are that mortal, and this looks like Golden Baton Pt. 3!

Perseus and Andromeda: You are Perseus, and your task is to bring back the head of Medusa the Gorgon, whose very glance will turn grown men to stone!

Ten Little Indians: The treasure of Major Johnston-Smythe has been much publicised in the press, and it is hidden somewhere in his creepy Manor. You cannot resist the challenge of finding it and thus find yourself on a train, rattling through the countryside towards your destiny.

Waxworks: This is the most recent mysterious adventure — having spent the day on a hot beach, you decide to rest and take in the cool atmosphere of the Waxworks. But you know, don't you, that no waxworks are really what they seem?

By the time you read this, one or two more will have been released, although too late for me to have seen copies. Again, they have intriguing titles like *Midwinter* and *After the Fire* (a post-nuclear adventure).

The versions for the Commodore and Spectrum contain high-resolution graphics, and these are toggled off as required — when the graphics are off, then a list of relevant information is shown on-screen in place of the graphics, including a list of useful objects. In the main, the programs follow the usual pattern of the two-word input and thus hold no surprises for the adventurer weaned on Scott Adams. Howarth even follows Adams in dedicating each adventure (sometimes obscurely, true, but that is what dedications are for, surely).

The plots and scenarios, as we've seen, are unusual and make a welcome change from the troll-bashing that is present in many other adventures. Many of the puzzles are also extremely hard to crack (of course, once unravelled, that's it — there is only one way to solve these adventures) and will keep you guessing for a long while. The adventures themselves, and the attractive packaging which would look good on any shelf, make for an interesting series that will reward a closer look.

Basics of adventures

In the next few pages, we will look at some general areas of adventures, to give you some idea of the many problems you will come across.

The golden rules

We've mentioned three in the section on *Dungeon Adventure*, but of course you will be able to think of others!

Words

As the argument over text or graphic adventures rages, so there is a similar argument over vocabulary. To reveal or not to reveal, that is the question. Some adventures disclose all in the documentation, in which case all the words recognised by the computer will be set down in black and white — there will be no confusion when it comes to playing. Most adventures, however, deem it necessary to add the Great Word Hunt to the puzzles inherent in the game, in which case only a smattering of the recognised words will be divulged in the documentation, or possibly none at all. In the former case, the player is given just enough to start him off, while the latter case assumes that the player is a hardened adventurer and will know the conventions.

There are two opinions on this — if the game were a real-life situation, players would of course be on their own, and would have to react in whichever way they saw fit, in which case their lack of vocabulary may very well influence the outcome of the game. But then many games are so tough that the player will certainly not need to be confounded from the very start, and have to work out the vocabulary before even getting to the nitty-gritty of problem-solving.

So if the authors are kind enough to give us guidance with vocabulary, we deem it a bonus and accept it gratefully: if nothing is given away, then we shrug and get on with it. It can be particularly galling, however, when this little bit of bad programming is encountered (and it is encountered in almost every program, unfortunately — with a few notable exceptions). You come across a location which is described thus: '*You are in a clearing. Before you is a hut*' — and then the program refuses to recognise the word HUT when you answer, 'Enter hut', saying instead, '*I see no hut here*'. This happens quite often: all that is happening is that the object is included, rather unfairly, in the location description as mere colour.

A potted vocabulary

This is a short list of those words that you will most often meet in adventures. The list isn't exhaustive so, if these don't do the trick for you, try a Thesaurus!

GO: This may be used when in a location with only one exit. However, most adventures are waiting for at least a two-word input, and you will then be asked, 'GO where?', or, 'Try a direction!'. If an open window is included in the initial description of the room, you can usually obtain a view from the window by typing 'Go WINDOW'.

DIRECTION: You can of course type 'GO NORTH', 'GO WEST', 'GO UP' and so on, but most programs will allow just SOUTH, EAST — or even S, E, which saves a lot of typing and makes for a much faster game.

Lateral thinking will often work wonders when you are deciding which way to go. Many adventures describe the location with '*Some of the exits are . . .*', and you may then try directions that are not mentioned. And, of course, the cardinal compass points are not the only ones you can use — try the odd SW or NE. If a ledge is at the location, you may often jump up or climb up, and you'll usually be well rewarded for doing so (although you could easily find yourself in a man-eating tiger's den).

Whichever way you find yourself going, *make a map*! If you get lost, you will be able to retrace your steps, or start the whole adventure again, and quickly get back to where you were (see SAVE, later).

QUIT: This is the word to use if you're ready to give up. Very occasionally, an author will regard this as the worst Californianese, and require you to type STOP, but such pedantry is mercifully rare. Most adventures will then go on to ask if you *really* want to QUIT, so giving you a chance to change your mind.

Then you are given your score, if a score feature is included. Scott Adams Adventures will inform you of the number of treasures you have so far collected, and give you a rating based on this number, compared with the total treasures available.

Channel 8 Software, in their Mysterious Adventures, seem content to stop the game as soon as you type QUIT, and you'll get no second chance! You'll not find a SCORE routine, either, in Mysterious Adventures — if you ask for your score, you'll be told, '*This is not a football match*!'

HELP: You'll get a lot of funny answers to this question — be prepared to be well and truly insulted for admitting to the computer that you haven't the faintest idea of how to progress. Many programs will merely say: '*Look around and try rephrasing your command*', which means that you will get absolutely no help whatsoever throughout the course of the game. Others will sometimes give a cryptic clue based on what is currently at the location but no adventure can give you a specific clue.

INVENTORY: Typing this, at any time, will give you a list of items that you are currently carrying. The word can be abbreviated to IN (don't be surprised if the computer misunderstands you, and takes you into a room full of bloodthirsty Orcs!), or I, which is probably safer.

LIST: Often accomplishes the same as INVENTORY.

LOOK: Will give you another chance to see the room or location that you are in.

REDESCRIBE: Has the same effect as LOOK.

EXAMINE: When you come across the body of a dead tramp, or a pile of leaves — in other words, some fairly inanimate object — then EXAMINE

it! Don't forget our golden rule: *Everything has a purpose.* Adventure programs haven't the memory space to include non-essential items, so assume that anything you see will have some purpose in the overall plan. Thus a pile of leaves will often conceal a gleaming sword, or a dead body yield up a diamond bracelet.

SAVE/RESTORE: Judicious use of this command will save you a lot of time. Before embarking on a hazardous mission, like going into a dark cave or opening a large wooden door, first of all SAVE your current position. Then if you are killed off, as you usually will be, you won't have to retrace your steps — steps which have no doubt been very hard-won. RESTORE is the command usually required in order to reload your position from disk or tape.

PICK/TAKE/GET: One of these will be your program's required command to pick up an object. To find if you have indeed acquired the object, type INVENTORY, when you should see the object added to the list of things carried by you.

DROP: Conversely, you may need to leave an object — most programs have a weight limit which, realistically, you cannot exceed. So a careful compromise is required, between an object's usefulness, and the player's acquisitiveness.

OPEN: This command is most often used when the player is confronted by a door or receptacle. Most programs are very pedantic in this regard — you will usually be required to unlock the door or chest before being able to open it. Sometimes, the player will have to PICK or TAKE the chest before being able to open it.

KILL: This is one for the old D&Ders! Most adventures will recognise the word, or indeed any synonym like STAB, KICK, THROTTLE, BREAK (that can also be used against the stubbornly-closed door) and so on. Finding the one that will elicit the response you are hoping for is really left to your imagination — don't forget to SAVE your position first.

D---: Along with all the other words that will be thought of at three a.m., when you are about to throw the d---ed computer out of the window, some form of swear word is almost always understood by the program. If they are not, it does not say very much for the programmer's sense of humour — on the other hand, you can't really expect the program to recognise every variation that you might dredge up.

This is necessarily a very short list of the words you will come across — you may be expected to DIG, SWIM, FLY, SHOUT, WHISPER, CRAWL, MOVE, SPIT, DRAW and COMMIT SUICIDE among many

other possibilities. The only way to find out which words a particular program accepts, is to try everything that comes to mind.

Monsters

We can have some fun now, and look at some of the monsters that might be met in the typical adventure.

For our present purpose, the term 'monsters' can be applied to any character in the program that is out to do us harm — and they are not always recognisable as monsters.

Like the program themselves, we can see distinct categories here. The classic adventure, descended from the original mainframe *Adventures*, contains fairly passive monsters which tend to sit there, waiting for some brilliant strategem from the player to scare him away, as happened with the snake in our scene at the beginning of this book. Or we may have to avoid the monster by finding a way around him.

The evil dwarves in the *Colossal Cavern* adventures are a notable exception. They appear occasionally to throw a weapon at the adventurer. The first dwarf throws an axe. This must be picked up by the player and then thrown at the succeeding dwarves. They are all hurling knives after the initial axe but, if you remember to keep retrieving the axe, you should have no trouble in surviving their attacks.

In Level 9's version of the classic, an endgame is included that extends the original in some 70 new locations. And here you can really get your own back on those little dwarves! By dropping dynamite near a crowd of them, you can score many points. Incidentally, during this finishing sequence you can also score points by saving a number of elves from death.

As played originally on mainframe computers after hours, the printer was often the only means of seeing what was happening — so blow-by-blow combat in D&D style was not really feasible.

This leads us to the next monster category, which we find in the action, or arcade games. In this category, the monsters are extremely active and definitely out to get you! The combat system from the role-playing games like *Dungeons & Dragons* is often used in these games, and the system is also very popular in many adventure games. And the monsters themselves, as befits the ancestry of this type of game, are of the leg-ripping, skull-crushing, heart-stopping type commonly met in RPGs. There follows a list of some of these lovely things, along with brief details of their attributes, and origins. I've put them in a very subjective order of fearsomeness — so, if you meet a Balrog, you would usually treat it with rather more respect than, say, an Orc.

DOGS: Very lowly, but also very vicious, monsters which usually attack in pairs. Only the most rudimentary of weapons is needed to fight them.

WOLVES: Even more vicious than dogs, and often more cunning.

DWARVES: Appear in the original *Colossal Caves*, as well as all the programs that have since based themselves on this classic. They usually jump out of the shadows, throw an axe or dagger, and disappear again — usually of nuisance value only!

ORCS: The Jack-of-all-trades monster — appears in most Tolkienesque adventures. Extremely unlikeable creatures, very spiteful and ugly. They hunt in groups and wield spears or scimitars. There is an orcish Archer in one of Level 9's games.

WAUG: This lumpish creature appears in the book *The Hobbit*, and also in the adventure of the same name from Melbourne House.

SNAKE: Most programs featuring a snake do so to create a puzzle rather than to pose any physical danger — see Chapter 1.

BIRDS: Often to be found at the top of mountains, sitting on golden eggs. Find a way to frighten them off before attempting to purloin the treasure! Birds also come in handy against snakes — see Chapter 1.

ELEMENTALS: Come in four types — Air, Fire, Earth and Water. You will need magic of a sort relating to the elemental before attempting to fight. Electronic Arts, in their program *Archon*, use Elementals to great effect.

VAMPIRES: No need to tell you how to fight these! Before coming across them, you should have picked up any requisites at other locations. If you come across some garlic, don't turn your nose up at it — you can be certain that there will be a vampire not too far away.

MIND VAMPIRES: A special sort — they're not after your blood . . .

GHOSTS: You won't need a sword or spear to battle with these!

WEREWOLF: Only magical weapons will be of any use against these, and preferably made of silver.

GOBLIN: Small, ugly creatures which delight in tormenting their victims by prodding their legs with sharp sticks.

HOBGOBLIN: Larger, more dangerous and more cunning than their animal-like half-brothers.

HARPY: Winged creature of amazing strength and agility.

SIREN: A sea-faring harpy, usually found basking on rocks near the sea, singing. (Rock music was never like this.)

TROLL: Devious, greedy, oafish creatures. Can absorb a lot of punishment, but they quite often don't take too kindly to light.

BARROW WIGHT: Ghostly apparitions that populate the Middle Earth wilderness.

CENTAUR: Half horse and half man, armed with bows and arrows, of which they are masters.

FIRE IMP: Little, agile flames of nuisance value.

FIRE GIANT: Large dangerous flame.

THUNDER LIZARD: One of the most potent of monsters. Only tackle if you are well-armed with conventional weapons, and have a fair amount of dexterity.

SAND WORM/PURPLE WORM: Extremely dangerous! The sand worm is basically a mouth with a 60-foot stomach behind it. The purple worm is similar, but with eyes and not confined to the sand.

MINOTAUR: The well-known bull-like creature. As dangerous as you would expect a highly-intelligent bull on the rampage to be.

WYVERN: Another winged creature, this one comes equipped with vicious fangs and razor-sharp claws.

BALROG: One of the most dangerous of monsters, much beloved of adventure writers.

DRAGON: Probably *the* most dangerous of all monsters: only the strongest and bravest of adventurers should approach this monster, or the most devious — dragons don't always want to roast you!

This is, of course, only a partial list of the monsters you may expect to meet. Most adventure programs will feature some of these and more of their own. In writing your own adventure, your imagination can be the only restriction. Just about anything can be pressed into service in a game, and I have seen everything from snowmen to London double-decker buses being used as death-dealing enemies.

Most programs that rely on a D&D-style combat system will keep you informed of your current physical status. This may take the form of physical points, combat points, or food points. You may, indeed, be given a combination of all these but it will be clear to you when playing that a decision on whether or not to fight with a particular monster must be made by you, taking into account your own strength, and using

whatever system the program adheres to together with your knowledge of the monster's rating.

Of course, you may be given no forewarning of a monster's presence, and thus have to fight whether you want to or not. This sort of program would be a very unfair one, and they're not common — most games will give you both a certain pre-knowledge of the monster's capabilities and a method of escape should you wish to decline the challenge. You may not, though, make good your escape if the monster is a particularly speedy one!

Spells

The sort of combat we've spoken about so far in this chapter with the system of strength or combat points is of course based on the physical side of combat — that is, a few well-aimed chops at the monster's head with your broad-sword or morningstar. Many programs, however, give you the option of employing magical powers and casting simple spells at a monster. This idea is very well-entrenched in D&D, T&T, and other RPGs, where many pages of the rulebooks are devoted to complex spells, which become ever more complex the longer you survive as a wizard.

The basic idea of a spell remains the same, however — to ZAP the monster with a well-aimed spell. This can take the form of a simple SLEEP spell which lays out the monster for a certain amount of time, to the more complex spells seen in Avalon Hill's *Telengard*, for a variety of American machines including the Atari, Commodore and Apple, and Quicksilva's *Velnor's Lair*.

Velnor's Lair concerns the black wizard, Velnor, who has gone into hiding in the goblin labyrinth of Mt. Elk. Your task as the adventurer is to prevent Velnor from metamorphosing into a demon on Earth, which event will not be good for mankind! You must explore the labyrinth, facing his ghastly guards, and finally come face-to-face with Velnor himself. You start the quest armed with a torch, a tinderbox and a club.

At the start, you are asked whether you want to be:

A WARRIOR: Which will of course be your choice if you fancy a bit of monster-bashing. The warrior has no spell-casting powers, but may use any magical items found in the labyrinths.

A WIZARD: Who is a bit of an easy target, having very limited physical prowess. His strength lies in his spells, of which he has three.

1. *Polymorph:* This costs one spell point (starting with 10 in total), and changes any non-magical living creature into a harmless frog. The success rate is normally 50%.

2. *Teleport:* Which costs three points, and which will set the wizard, with whatever he may be carrying, back at the entrance to the labyrinth.

3. *Fireball:* The most potent, but also the most costly, at five points. This spell kills any creatures without magical protection at the location.

A PRIEST: Who is somewhere between the wizard and the warrior in ability. Like the wizard, he has three spells to choose from, and an initial 10 spell points. He has three spells available to him.

1. *Shield:* Which costs him three points, and which makes him less vulnerable to attack — it can, however, only be used once.

2. *Dispel undead:* Which does just that, and also costs three points.

3. *Heal:* Which costs four points and heals all the priest's injuries.

After all this preamble, the rest of the game follows our *Wumpus* model fairly closely. A maze of tunnels is explored by the player. Many problems have to be solved and, of course, there are lots of monsters to battle with. Apart from the Spell routines, which are complex and worth obtaining the program for (although other more recent programs have even more complex routines), *Velnor's Lair* is a run-of-the-mill adventure, but one which should give a lot of pleasure to the D&D enthusiast. At the time of writing, it still features in the adventure Help columns of all the various magazines, so it has obviously stood the test of time.

CHAPTER 3
Further Adventures

In this chapter, we will look at some other great names in US and UK adventures, and at the currently available software for the QL.

Two of the American software houses stand far taller than the others producing text adventures — Scott Adams' Adventure International, and Infocom.

Scott Adams

The first name most people hear of when they start computer adventuring in earnest is Scott Adams. This is especially true if they own an American machine such as one of the Commodores, the Apple, or the Atari. Scott Adams, and his company Adventure International, have had almost a monopoly in American adventures, where games for British machines like the Spectrum, the Dragon and the BBC are very much in the minority. At the time of writing, versions for all these British machines are appearing — written, incidentally, by Brian Howarth of Channel 8 fame (and one suspects that a great deal of prodding went on to persuade Adventure International that there was indeed a market for games on these machines!).

It is fitting that Howarth should have a hand in implementing these adventures for home computers, for Channel 8 programs have a great deal in common with Adams' adventures even, as already mentioned, down to the dedications at the start of each game.

Scott Adams started his career as a computer programmer in Florida and, like many before and since, found Crowther and Woods original *Colossal Cave* on the company's mainframe. Entranced at the fantasy world conjured up, he wrote a version on his TRS-80, before releasing a commercial program, *Adventureland*, in 1978. The project very nearly died in the making, when his wife Alexis put the disk in the oven — luckily for Scott and a grateful world, the oven was not switched on!

Adams went on to found Adventure International, which nowadays has several arcade success stories including the great *Preppie*, as well as many business programs and utilities. AI are also promoting another adventure-writer, Jyym Pearson, who has written four excellent adventures.

The twelve programs written by Scott (and Alexis) Adams are recognised as classic text adventures.

There are twelve:

Adventureland: This is the one that started it all — there are 13 treasures in all to find, and some 29 locations to be mapped. All the other adventures follow the same pattern as this first one and, indeed, Adams wrote an Adventure Compiler, which acts as a framework on which can be hung any storyline.

Pirate's Cove: And this shows the technique in action — Alexis Adams actually wrote this one, using the Compiler. It is probably the easiest, but do not believe for one moment that any Scott Adams Adventure is not frustrating and devilishly difficult! A lot of humour is present in *Pirate's Cove*, and it has 25 locations, though only two treasures.

Mission Impossible: Is a race against time, with the player attempting to stop a saboteur from blowing up the nuclear reactor.

Voodoo Castle: Set in a dark castle with 24 rooms, this one leads on naturally to:

The Count: Dracula, in this case — although the adventure only has 19 locations, 'things change' throughout the game, as night succeeds day.

Strange Odyssey: Set in space on a damaged spaceship crash-landed on an alien planet, this adventure has 22 locations, and only five treasures — but the last one will prove extremely difficult to find!

Mystery Fun House, *Pyramid of Doom*, and *Ghost Town*: A trio of adventures, the scenarios of which should be fairly obvious from the titles. *Fun House* has some 37 locations and is many people's favourite of the 12. The second can be rather tricky, while the third is extremely difficult. It has nearly 40 locations, and can be solved in more than one way. It also has a couple of objects carried over from *Pyramid*.

Savage Island Pts. I & II: These two are parts of a whole — the first has to be solved before the second can be tackled. The goal of the first is a password to the second. They are the toughest of the Scott Adams adventures and not for the faint-hearted.

Golden Voyage: Less difficult than the *Islands*, this concerns your race to bring back the elixir of youth for the king.

These adventures may appear at first glance to be just like all the other programs around, but they were the first — in fact, most of the other text adventures merely imitate the Scott Adams series. No news yet of more adventures in the series, but the existing programs will keep many players happy for hours and hours in their attempts to solve the Adams' conundrums.

Infocom

If Scott Adams is the American Channel 8, then Infocom is certainly the American Level 9. Infocom display a lot of panache in their programs and the packaging of those programs. Just look at the paraphernalia which is supplied with *Deadline*. This adventure is a sort of sophisticated 'Cluedo', with you as the Inspector attempting to solve the murder mystery. You are given 12 hours to accomplish the task, and a lot of paper. This paper includes lab reports and so on. You'll also be given a page from a newspaper, a scrawled note, and a book of matches with a telephone number on the back. It all adds to the atmosphere surrounding the adventure, and it is typical of the panache with which Infocom endow their programs.

But to begin at the beginning. Infocom is the first of a new breed of companies revolutionising the adventure game market. It is a young company, made up of young computer professionals who cut their teeth working within the hallowed walls of MIT.

Since going it alone, Infocom has played a large part in the development of adventure games. *Zork*, their first program, has become a classic equalled only by the original *Adventure* and the Scott Adams series. If this had been their only program, they would have a secure place in the Adventure Hall of Fame, but they have since released *Zork II* and *III*, each of which is more than a match for the first.

Zork I was originally written in a LISP-inspired language, MDL, on mainframe computers. In the early 1980s, Infocom used their expertise to implement the program on micros. It borrows heavily from Crowther and Woods original *Colossal Caves*.

The games are now available for a very wide range of computers including Apples, Commodores, CP/M systems, Tandys, Osbornes and DECs.

One of the reasons for Infocom's success is the language used in the programs. The descriptions are no mere one-liners, as in other adventures — instead, they are extremely witty and rich in detail. Whereas the Scott Adams adventures are rather spare in their scene-setting and recognise only short commands from the player, being content to give the player more and more difficult puzzles, Infocom has developed an extremely sophisticated 'parser', a routine that allows the player to communicate with the computer in a natural language.

Thus a Scott Adams adventure will say: '*I'm in a flat in London. Visible items are . . .*', and the player must reply with one or two words, for example: '*Take sneakers*'. An Infocom adventure, on the other hand, will say (and notice the shift in emphasis here to a first person game): '*You are standing before a chasm . . .*' (the description may well go on to fill up a whole screen!), to which the player can reply, for example: '*Throw the newspaper, the red book, and the magazine in the chasm*', or similarly: '*Take book. Go north. Drop bell, book and candle*'. Other characters may be asked to carry out various actions, or divulge certain information — for instance, the whereabouts of the gold. The program will even try to figure out what you mean when you don't give enough information.

While LOOK is recognised by every adventure game, although the word can take on several different meanings, think about the possibilities in *Zork*, where LOOK INSIDE and LOOK UNDER are equally valid!

Several commands can be used when travelling about the world of *Zork*. For example, VERBOSE will ensure that you are given a very detailed description of each room, every time you enter. BRIEF will give you a description of only newly-encountered rooms, while SUPER-BRIEF will give only the name of each room.

Although each of the three Zorks may be played and enjoyed separately, there are glimpses of the others in each one, and the storylines do in fact follow on. In *Zork I*, the Great Underground Empire confronts the player with predicaments ranging from the mystical to the macabre, with 20 treasures to be found before escape has to be made. *Zork II* takes the player to new depths of this subterranean realm where the Wizard of Frobozz will be met. And in the final game of the trilogy players encounter the Dungeon Master himself, who holds their destiny in the balance.

The *Zork* adventures, unrivalled though they are, are only the tip of the Infocom iceberg.

Deadline we have seen, and this is complemented by *The Witness*, which is a whodunnit rooted in the popular crime novels of the 1930s and 40s — it too boasts a dossier stuffed with clues in the form of matchbooks, newspapers and so on.

The science fiction adventure *Starcross* takes place in the year 2186. The player has to rendezvous with a starship from the outer reaches of the galaxy and enter its mysterious interior. Once inside, other-worldly beings are met, some helpful and some harmful. Another science fiction adventure is *Planetfall*, though this one is rather more light-hearted than the others. It takes place on a distant planet where the player has been shipwrecked, armed with only a patrol-issue, multi-purpose scrub-brush and aided by an impish robot companion. The player is challenged with saving the doomed and plague-ridden world while trying to keep a straight face.

```
>ALL ROBOTS,REPORT LOCATION
FC: Cryolink established to Whiz.
WHIZ: In the Weather Monitors.

FC: Cryolink established to Poet.
POET: In the Weather Monitors.

FC: Cryolink established to Auda.
AUDA: In the Sloping Corridor.

IRIS INTERRUPT: You may not be aware
of this, but I'm not working. I need
to get repaired.

FC: Cryolink established to Sensa.
SENSA: In the Weather Monitors.

FC: Cryolink established to Waldo.
WALDO: In the Weather Monitors.

FC: Cryolink established to Iris.
IRIS: In the Weather Monitors.
```

Figure 3.1(a): Suspended (Infocom).
Your six robotic extensions report in with their initial starting places — Iris interrupts with a plaintive cry for help.

```
>SENSA,REPORT
FC: Cryolink established to Sensa.
FC: Full report from SENSA
SENSA: Internal map reference --
Central Chamber
All around me charges flow, shaped by
the very nature of this room. The
electrons are being channeled into an
electrical column, central to this
environment.
In the room with me is Poet.
SENSA: I cannot sense a thing in my
grasping extensions.

>POET,REPORT
FC: Cryolink established to Poet.
FC: Full report from POET
POET: Internal map reference --
Central Chamber
It hops and skips and leaves a bit,
and can't decide if it should quit. It
tells the world what it should know,
but doesn't know when it's been shown.
In the room with me is Sensa.
POET: As far as I know, I'm Zen on
inventory.
POET: Sensory pads detect no abnormal
flow.
```

Figure 3.1(b): Suspended.
Sensa and Poet see the central chamber in very different ways.

The most original of the Infocom adventures is *Suspended*, subtitled *A Cryogenic Nightmare*. Written by staff writer Michael Burlyn, an established writer of SF novels, the adventure concerns cryogenic suspension — bodies in deep freeze. Having woken from a long frozen sleep, the player finds the planet in crisis and remembers that he has won, 500 years before, a lottery to serve as Contra's Central Mentality. Using a game board and game pieces, the player must manouevre six robots in an attempt to end the crisis.

Each of the robots has one sense, and thus the overall picture has to be built up from the input received piecemeal from each robot. Poet, as his name hints, is given to long-winded obscure passages, while others are more straightforward in their information. The player also has control of Waldo, which is able to manipulate objects, and Whiz, which can interface with the Central Computer's memory banks. Thus, it is not enough to say 'Get object' — in *Suspended*, the player will first have to ensure that Waldo is already at the location, after making sure that Iris the Eye robot has already seen the object. And, of course, robots can go wrong!

There is no need to map the 58 locations, as in the majority of other adventures — this is taken care of by the board and rubber pieces already mentioned. There are four levels of play, one of which will allow the player to configure the adventure to his own tastes — the robots may be placed in any starting position, and the various systems set.

The most original and challenging adventures that we know of!

The Hobbit

The Hobbit is probably the best known, at least by reputation if not by personal experience, of computer adventures,. It is a classic, and is one of the most talked-about and puzzled over! In the true tradition of software progress, other programs will arrive that will, no doubt, improve on *The Hobbit*, but it will retain its position in adventure history.

The book called *The Hobbit* by J. R. R. Tolkien is, if you like, the preface to the monumental *Lord of the Rings*, probably the richest source of material for the adventure-writer. This vast work with its three volumes of absolutely riveting story and the supporting historical documentation in *The Silmarillion* (together with encyclopaedic information from authors such as Tony Tyler with his *Tolkien Companion*), is an absolute goldmine of ideas. Melbourne House pulled off a fabulous coup in acquiring the rights to use *The Hobbit* from the Tolkien estate, and has produced a program which lives up to the novel. *The Lord of the Rings* is an obvious sequel to *The Hobbit* and, of course, is the one everyone is waiting for.

The basic concept had been used several times before *The Hobbit* was written — a text adventure in the good old style, supported by graphics.

Melbourne House has taken many scenes from the book, and illustrated them with beautiful high-resolution pictures, apparently working from an artist's impression.

This was a logical progression from the earlier programs from other authors, which featured block graphics. The other important details that set *The Hobbit* apart from previous text/graphics adventures is what Melbourne House calls animation and Inglish. In this context, animation means that the characters in the program, such as Thorin and Gandalf, the various monsters, like Gollum and the Butler (he didn't do it, by the way!), and the wood-elves, carry on their own lives while you, as Bilbo Baggins, get on with the adventure. In practice, this means that characters keep walking in and out, singing about gold and so on. Gandalf has even been known to walk off with the front door to the hall at Bag-End! All this is not necessarily useful, but it is entertaining all the same. Inglish (and that is no misprint) is Melbourne House's term for compound sentences.

Now let's imagine that you have reached the dragon's lair. He is, of course, guarding the gold which you have battled through many dangers to retrieve from his clutches. When you arrive, you want to draw your sword, fight the dragon, and then collect the gold. Most programs would require you to type in several commands before the final objective was achieved. So you might well have to type DRAW SWORD, KILL DRAGON, then finally (and if you were successful), GET GOLD.

The Hobbit, however, will accept commands such as: 'Draw sword and kill dragon, then get gold and leave', thus saving finger stress, but also having a more natural feel.

The adventure may be solved in many ways. Unlike the traditional programs, in which there is only one solution to the complete puzzle, *The Hobbit* may be completed by any one of several methods. So whichever way you choose to solve the game's ultimate puzzle, there is a different game awaiting you should you wish to play it another way. The aim remains the same, however — to rescue the gold from Smaug, the dragon, and return to the comfort of your own home under the Hill at Bag-End. A percentage SCORE is given throughout the game and

```
You are in a comfortable tunnel like hall
To the east there is the round green door
You see:
     the wooden chest.
     Gandalf.
     Thorin.
Gandalf opens the round green door.
Thorin waits.
```

Figure 3.2(a): The Hobbit (Melbourne House).
This is where it all begins.

```
You go east.
You are in a gloomy empty land with dreary
hills ahead
To the west there is the round green door
Visible exits are: east north northeast
You see:
     Nothing
Gandalf enters.
Thorin enters.

You go east.

You are in the trolls clearing
Visible exits are: southwest southeast
north
You see:
     the hideous troll. The troll is
     carrying
       the large key.
     the vicious troll.
Gandalf enters.
Thorin enters.
The hideous troll says " Blimey, looks at
this!! Can yer cook'em? ".
The vicious troll says " Yer can try, but
he wouldn't make above a mouthful ".
```

Figure 3.2(b): The Hobbit.
And this is where it could all finish!

```
Day dawns.

You go west.
You are in in a clearing with two stone
trolls
Visible exits are: southwest southeast
north
You see:
     the large key.
Thorin enters.

You take the large key.

You are carrying
     a curious map.
     the large key.
     some lunch.
Thorin sits down and starts singing about
gold.
```

Figure 3.2(c): The Hobbit.
. . . but read the book and you should be safely through. Thorin sings about gold . . . again!

people have finished the adventure with scores as low as 42% and as high as 210%!

As an aside, let me say that I feel that *The Hobbit* is a classic but, nevertheless, a flawed classic. Many people have, for example, reported that pressing CAPS SHIFT and 1 together (and this could well happen accidentally!) causes the program to crash. A similar fate awaits the person trying to enter certain locations without a particular item. Animation (Melbourne House's name, if you remember, for the character's self-motivation) occasionally manifests itself in strange ways — kill off Gandalf (surely you wouldn't be tempted to do that, would you?) and he may well turn up alive and well at some later date (and may also be carrying your front door . . .). These are little quirks that Hobbitists have become accustomed to, and in fact there is almost a fan club of Hobbit Bug Spotters!

The Hobbit, while being on the surface a traditional adventure, nevertheless plays in a unique way. The puzzles set by programs in the classic style (programs like Level 9's) have one solution (usually) which remains the same. Once you have completed the adventure, that's it: of course, the puzzles may take months to solve, and several programs have vast areas which act as a kind of divertissement — all those locations off the beaten track of the game, which play no part in the solution but which would serve to entertain you. *The Hobbit* itself has an intriguing location which is, the computer informs you, '*Too full to enter*'. Does this mean that the Spectrum's memory is too full for the programmers to write more locations? Almost certainly not, for they would surely be more inventive than this! Is there, more likely, a plan afoot at Melbourne House to launch a sequel carrying on from here into the uncharted area, à la Level 9?

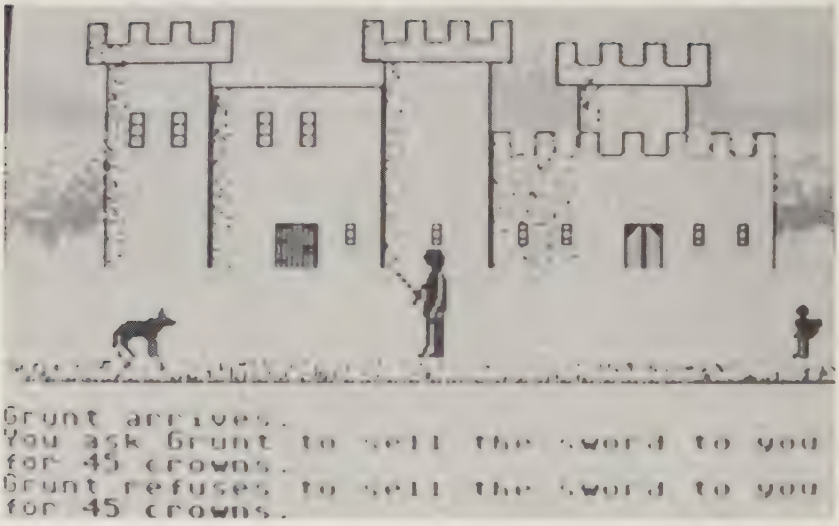
Most adventurers will be pleased to know that Melbourne House has obtained the rights to *The Lord of the Rings* — the resultant program will almost undoubtedly take the form of three parts, as the original book, although it is tempting to think that the QL with its microdrives might be a good machine to be blessed with the new program.

There has been, and there will I hope still be in the future, lengthy discussion on *The Hobbit* — if you have wondered whether to buy this most charming, frustrating, annoying, beautiful program, rest assured that you will enjoy many hours of good adventuring.

Valhalla

Though *The Hobbit* was the first of the *classic* Spectrum adventures, and has since been translated to several other micros including the BBC and Oric machines, a couple of more recent programs deserve mention.

Valhalla comes from Legend and is an animated adventure. The screen



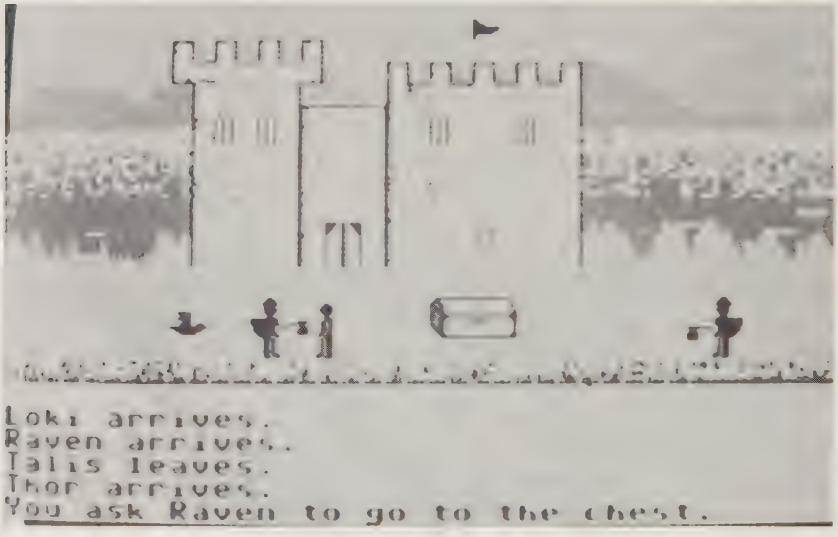


Figure 3.3: Valhalla (Legend).
Representative views of the scenery, with one or two inhabitants, from the Commodore 64 version.

consists of a large graphic screen, taking up two-thirds of the total screen area, and a smaller text window beneath. Every location has an associated picture made up from a number of standard features. There are many castles, with fluttering flags and icy wastes. Once the picture has been built up from the various 'blocks', then the characters are placed in the picture: 'you' are always there, of course.

Valhalla concerns the player's six quests, the search for six objects (Ofnir the Key, Drapnir the Ring, and so on). On completion of these quests, the player attains Valhalla, the home of the gods. On his way around the landscape, the player can equip himself with the basic items like wine, food, weapons and armour. As I've hinted, you are not alone — there are evil gods (like Loki and Hel), and good gods (like Thor and Saga), as well as various dwarves, giants, wolves and dragons, all with their various characteristics.

There are many of the usual adventure commands, such as LIST, GET, TAKE and HELP, and some rather more unusual ones, like WHO and WHERE — these reveal who is at the present location with you, and where exactly you might be.

Other unusual features are the method of travelling (usually by moving in one of the compass directions, but sometimes by Ringways which can only be used if you have a ring — and you don't always know where you will end up . . .), and the way in which you can influence characters to do your bidding, like fighting another character or giving up a weapon or food. Even better, each character acts according to his or her own qualities, but the player can make a character more cooperative by attacking another character of opposite qualities. Thus Odin, who is a goodie two-shoes, will be more inclined to help you if you attack Hel and Gripe, who are a couple of baddies.

The program is written mostly in BASIC, with some machine coding to help with drawing the pictures, and the action can be pretty slow at times (although the Commodore 64 version has been speeded up somewhat), but it is still a fascinating game. Actually, Legend prefer to call it a 'computer movie', as it is entirely possible to sit back and just watch things happening.

Lords of Midnight

Beyond Software released a couple of arcade games before *Psytron*, but that game was a complex piece of work — in fact, probably too complex for its own good. It did very well in sales, however, if the chart positions it attained are any indication; and one or two of the techniques introduced in the game were innovative, like the method of progressing from one level to the next, on a percentage rating based on an average of best scores. Their next game was a real corker!



Figure 3.4: Lords of Midnight (Beyond).
Nice holiday snaps of some of the lords.

Lords of Midnight is unlike any other adventure so far released for any machine. The beautifully-produced booklet with the program contains five chapters from the Book of Midnight. This extract tells you of the beginnings of the War of the Solstice, of Morkin and Luxor and their fight against the evil Doomdark.

There are no moving graphics in *Lords of Midnight*, but the scenes at each location are stunning. Like *Valhalla* they are built up from several standard blocks, but each one looks fresh. The back cover of the booklet contains a map of the Land of Midnight, and you can move freely around the landscape — as you arrive at each point, you may look around all the cardinal points and as you do so the view in that direction is presented on-screen. Move forward, and the features before you will grow in size

Your mission is simply to destroy Doomdark, whose part is taken by the trusty Spectrum, but victory can be won in two ways. First, a strategic victory in which the Keep of Ushgarak is stormed and overrun by the forces at your command; and second, Morkin may find the Ice Crown and destroy it in some way.

To accomplish all this, you can summon up four characters. The first and most important is Morkin, the son of Luxor. It is his task to capture the Ice Crown, and in order for him to do this he must be allowed to reach his objective: you can help him in this with diversionary feints with your armies. Luxor is almost as important, as he wears the Moon Ring — this is the only shield against the Ice Fear, which demoralises and saps the strength of any of the Free not protected by Luxor. Luxor is the leader of all the armies, but must first influence the commanders of those armies to take up the Cause. Corleth the Fey and Rorthron the Wise make up the remainder of the quartet that you control from the keyboard.

Choose one of the four characters, and the screen shows his point of view, as well as his name and coat-of-arms lovingly depicted in heraldic glory, and a brief description of where he is looking, such as: '*Rorthron the Wise. He stands in the Mountains of Ahimar, looking East*'. There is also a representative of the various nasties, such as wolves, dragons and the local nuisances, the Skulkrin, which may be present. You may turn the character through the eight compass directions, and the view will change accordingly. Then you may move the character in the direction he is facing or THINK, on which command the screen clears and information is given about the armies the character may control, the physical state of the character, and so on. The player may also CHOOSE, which presents him with a list of options: these options are affected by the basic personality of the character.

Each character may be moved up to eight times before night must fall. Under cover of darkness, the forces of Doomdark are moved and battle is joined across the map where the opposing forces meet.

Lords of Midnight is extremely complex and will take many hours to

play, one of the attractions being that each game will be different. It seems to me to be the ideal combination of strategy and adventure game, with the strategy element probably the more important, although it is more difficult to succeed with. Two sequels are planned — *Doomdark's Revenge*, which takes place in the land to the north-east of Midnight; and *The Eye of the Moon*, which will be set in the land to the south of Midnight. So the Midnight addict is assured of many happy hours ahead!

QL adventures

At the moment, of course, there is not much software at all for the QL, though by the time you read this I'm sure that there will be quite a selection from which to choose. The only software available at the time of writing are a couple of over-priced utilities (LOAD with one keypress, tabulated microcartridge directory and so on), a very expensive business package and, what interests us here, two adventures.

West and *Zkul* are both from the same software house, Talent Computer Systems of Glasgow, and together illustrate two facets of what will become the QL software scene.

When previous Sinclair machines were introduced, programmers started with a blank sheet of paper with which to experiment. The ZX80 had no pool from which to draw, and so users of that pioneering machine were able to bring fresh ideas and techniques to bear. The ZX81, with a more powerful BASIC language on board, offered more facilities but the commercially available software took some months to reach any great volume.

It was the Spectrum that changed the essentially 'hobbyist' tag that was applied to computing at this time. With the knowledge that had been gained of the Z80 instruction set, programmers were able to get to grips very quickly with the new machine, and the colour and high(er) resolution of the graphics made for ever-more-exciting programs. True, the first few months were devoted to versions of *Othello*, *Star Trek*, *Hammurabi* (most of them converted from the ZX81), and all the Space Invader, Scramble and Moon Lander clones which kept the public happy during this time. 1984 will be remembered, however, as the year in which arcade games for the home computer really took off and became big business. And the authors of the new games responded by taking the Spectrum to the outer limits of its capabilities.

Have they reached those limits? Well, it is dangerous to be firm about anything when we talk about home computers but we believe that, without pressing outboard devices into service, software writers can *probably* not attain much more, although there will of course be more arcade games, and ever more imaginative scenarios. It is for this reason that we are very optimistic for the future on the Spectrum of more

complex adventures, strategy/war games, management simulations and the like. So many of the machines have been sold that it is unlikely that it will be dropped in favour of the latest XYZ sub-£100 computer, and the future looks bright. Now we are in the same position with the QL as those early writers were with the Spectrum — a new machine, exciting capabilities, an exciting new language. The processor is different, so a new set of instructions will have to be learnt before machine code can be used to the full and a short period of adjustment can be predicted until really stunning programs are released.

What sort of programs will these be? Well, I'm sure that we shall *not* be seeing 'QLInvaders' or 'QLScramble'! Although there is already a horrendously overpriced (£12) version of *Connect Four*, the QL is obviously worthy of far greater things. We believe that we can look forward to complex strategy games, like the Spectrum's *Codename MAT*. This program from Mikrogen is a very sophisticated version of *Star Trek*, but this time you are all that stands between a free solar system and the dreaded alien enemy. Mixing superb graphics in the arcade sequences with detailed tactical deployment of the space fleets under your command, the game is a delight to play. Rather than being a quick bash to relieve those feelings of aggression, a more thoughtful approach is called for. Personally, we find the arcade passages rather hard to master and would prefer emphasis on the tactical. However, this is the kind of game that would translate well to the QL.

The two adventures from Talent do not really make full use of the QL's capabilities. *West* has been translated from the Commodore 64 version (and we believe that a Spectrum version is also now available) and, apparently, no further locations have been added. Only a fraction of the QL's available memory is used, but the adventure is still enjoyable for all that. As the title suggests, the game takes place in a Wild West town, complete with tumbling Tumbleweed, wheeling vultures and sneering bandits. Unlike many others, this adventure will allow you to be resurrected very easily and without penalty. The game parameters are not reset so that, after dying a few times, you will find yourself stumbling over piles of corpses.

Unlike *West*, Talent's other QL adventure *Zkul* is a brand-new invention (as far as we are aware). It follows the traditional adventure scenario, with the protagonist proceeding through dank caves with underground streams and lots of treasure to keep the player amused. Several features of the original mainframe *Colossal Cave* are present here — spend too long wandering around a problem, and the program will ask you if you require a hint! This, though, will cost you a few points, as will resurrection — you may be resurrected at any time, but your body materialises at another location outside the cave complex, into which you will have to regain entrance.

The text in *Zkul* is suitably florid and long-winded — but the available memory has still been sorely under-used. It seems only natural that, with two microdrives at hand, any adventure should use the first cartridge to load in the handling program, and then the second cartridge to feed in data as and when required. The drawback to that, of course, is the initial cost to the software house of the microcartridges, which eventually has to be passed on to the public. The end result will be pretty expensive adventures. This is not too bad if the game itself is good.

Some of the games already mentioned in this book would be welcome conversions. *Lords of Midnight* would be an excellent conversion from the Spectrum to the QL, expanding the map and graphics, along with the number of characters. The adventure any Sinclair fan will be waiting for, however, is *The Lord of the Rings* from Melbourne House, who have recently bought the computer game rights of the classic. It could be superb on the QL, with battle sequences, quests and so on, all going at once.

For now, that is the whole adventure scene for the QL — but it will not remain that way for long.

CHAPTER 4
Adventure Generators

One final type of adventure program needs mention here, and that is the 'adventure generator'.

There are two types, one of which grew from the other. Scott Adams and Infocom, names which should be familiar from previous chapters, create their programs with the aid of a master generator (which must be insured for many millions), as do most of the companies who specialise in adventures. The scenario of the adventure, with locations, objects and puzzles, is written in the form of a storyboard. Then the generator program is run and all the data written in. Voilà! another best-selling hit.

There is a problem here, of course, in that all the resulting adventures may then become too similar, and most of the games from the above-mentioned companies have a certain likeness to others from the same stable — so the player can often tell an Infocom game from a Scott Adams game and so on. The trick that these two in particular, and others to a lesser extent, employ to keep their games from becoming stale in format is to write an interesting scenario. This must come first and, just as important for Infocom, a witty and entertaining text must be employed to develop this scenario. As we've seen, Scott Adams approaches the problem in a rather different way to Infocom; descriptions are rather terse, but the puzzles are extremely difficult. Infocom's language is much more flowery (although this can be turned off if desired), and the scene-setting superb. Also, the puzzles contained in their adventures are difficult, but a premium is placed on logical deduction and novel situations.

While Infocom's adventures are text-only, Scott Adams and Adventure International now mix graphics with text in their games. The original Adams games were text-only, but then a graphic generator was written, which enabled the early stories to be converted to meet the current need for graphics. Now all Scott Adams adventures include graphics. The first of the graphics adventures for British machines has at last been released; this is *The Hulk*, which has met with mixed reactions. The graphic generator has also been released commercially for the Atari, under the name of SAGE (Scott Adams' Graphic Editor), so that graphics may be included in your own program — you'll have to wait a long time for Adams to release the text generator that he uses!

Meanwhile, several generators have come on to the market in Britain:

in fact 'games machines' have become something of a vogue this year, with titles like *Fifth* (from CRL), *Scope* (from ISP) and *White Lightning* (from Oasis). These are primarily for arcade games, but we have a number for adventures, too.

Dungeon Builder is from Dream Software and is a very friendly utility, enabling the user to design an adventure complete with graphics. It is a very easy program to use, although drawing the graphics can be rather fiddly. The excellent manual guides the budding author through the various stages. The results are pretty good, though there is not room for a lot of text and the graphics are rather slowly-drawn when play is under way. Don't be fooled by the dungeon of the title — it isn't at all necessary for the scenario to be set underground!

But the most successful and widely-used adventure utility must be *The Quill*, from Wales-based Gilsoft. Much has been written about this program, and much has been written *by* it, so we won't go into detail here. Suffice to say that the manual, while not quite as friendly as that of the *Dungeon Builder*, is certainly exhaustive, and should help even the novice author to write a professional-looking adventure. As even the biggest software houses find, great care has to be taken to prevent all the adventures written with the utility becoming clones of each other, and some of the adventures available commercially at present manage this better than others. While certain parameters can be changed (for example, the character set may be changed and graphics can be implemented if the user is experienced enough), in the main the writer is stuck with Gilsoft's own view of adventures. This isn't a bad thing, of course, but generally a Quilled game is easily recognisable — some of the system messages, such as '*Quit: are you sure you want to quit now?*', cannot be changed. Thus the scenario and novelty of the adventure becomes even more important. One or two adventures have been released that take advantage of unusual packaging (eg including a bottle of pills which might act as a clue), although most are still sold in the ubiquitous cassette box.

If you wanted to look at a Quilled adventure before buying the utility, which is currently available for the Spectrum and Commodore 64, Gilsoft have recently released their Gold Collection. The first part of the Collection comprises some six or seven programs, some from in-house authors and some from third-party authors who have sent in their adventures. Apparently, Gilsoft have mountains of tapes sent to them each month.

It was programs like *The Quill* that inspired us to write the programs for this book: rather than showing you how to write a text adventure, we thought it might be rather more entertaining to present you with an Adventure Generator, and a game to go with it. So let's get on to Part 2 of *QL Adventures* and start programming!

PART 2

The Adventure

CHAPTER 5
How to Use the Programs

We recommend that you read through this section before typing anything into the QL. When you have digested everything, then is the time to begin pounding the keys. As a direct command, type AUTO — this will ensure that the listing starts at 10, and will then increase in increments of 10 each time ENTER is pressed. In this way, you don't have to keep checking on line numbers.

QLAG and QAD

An adventure is a fantasy. As the author of this fantasy, you can draw from any period in history (or the future) and any location. Think of the exercise as writing a novel. Your fantasy can be located, for example, in a huge spaceship travelling the vast wastes of inter-galactic space, or a Viking longboat in the North Sea. Although the term adventure can be stretched pretty far in being applied to games which are less and less like the original adventures, nevertheless the basic program that we will be 'composing' here is of the traditional type.

So let's indulge a fantasy and decide that our adventure will take place in a dank, dark dungeon beneath some ancient castle. Inhabited by monsters of all kinds, it's the sort of place in which you need some companions, so you will have a party of three characters instead of just one.

Using QLAG

After you have entered the listing in Chapter 6, SAVE the program and then RUN it — in fact, *always* SAVE a program before running it.

After 20 seconds or so of initialising, you will be presented with the main menu. The options available are:

LOAD DATA FROM MICRODRIVE (1)
ALTER THE MAP (2)
ALTER THE OBJECTS (3)
ALTER THE MONSTERS (4)
ALTER THE CHARACTERS (5)
END THE PROGRAM (6)

The first time the program is run, there will of course be no data to load in (you will use this if you have saved data from a previous session). So your first choice will be the second, 'alter the map'. Press 2, the screen will clear and the menu will be replaced by a 12 × 24 grid in the top left of the screen. This is the adventure map. In the centre of the lefthand lowermost square, or 'cell', you'll see a small white dot, the cursor. By using the cursor keys, you can move this dot around the grid; as it moves to each location, information on that cell will be displayed top right.

To start with, all the cells have the description 'nondescript' and are empty. By selecting E (for Exits) or D (for Description), you can place exits in each cell as you like, and give a name to each cell. After typing in the description, you are asked if there is a monster or object here. There can be both, and more than one of each.

From options 3, 4 and 5, it is possible to alter the names of the objects (thus creating 'The Golden Sword' or 'King Tony's magic stone') and the strengths and weaknesses of monsters and characters.

Changing the strengths and weaknesses is achieved through the use of 'spreadsheets', which contain the necessary data to alter the 'stats' of characters and monsters — that is, information on their Greed, Spells, Attack and Defence capabilities, and so on. Initially, these are all set to 0 for each monster and character but, by moving the cursor around the spreadsheet, this null value may be changed.

Greed controls the willingness of a monster to accept a gift, while Anger controls its readiness to attack. Sometimes characters may run from a fight, and sometimes monsters may. Gift is the object number that the monster will accept automatically as a bribe. Monsters should generally have low attack and defence values early in the game so that, as the strength of the party increases after prolonged exploration, the monsters may be given greater and greater powers. The combat value of a monster is related to the square of the value you give to it in the spreadsheet, whereas the characters simply have the exact value you give to them. Characters, however, get bonuses for objects carried.

Finally, the data thus created may be SAVEd to cartridge for use in QAD, the second program, which will use all this data to run the adventure. By selecting option 6, End the Program, you will be asked to give a name to the set of data: the program actually saves two files, one of numbers and one of characters. When you run the main program, use this name to load in your data.

Using QAD

It is perfectly possible to sit down at the keyboard, with a blank map in front of you, and design a game from scratch. However, a better method, especially if a large adventure is foreseen, is to design the whole thing

beforehand. The numbers do not all have to be worked out at this stage, but you could draw up the map, the locations of the monsters and treasures, and the usefulness of those treasures. In particular, pay special attention to the exits and entrances. We have not included a routine here to create Time Traps or Exits to remote locations (all exits lead to a physically adjacent room), but this can be simulated by the use of cursed objects (see later) and imaginative descriptions. While on the subject of locations, two warnings:

1. *Don't* put a monster in the first location. Your party will be trapped!

2. *Do* ensure that each exit from a room has its mirror entrance in the adjoining room. You may, of course, wish to make a room into a trap in this way so that the party will eventually be able to leave only if in possession of a cursed object — but this should be a result of your careful planning, not haphazardness. It is also a good idea to be fairly sure of the descriptions of locations (so typing is kept to a minimum) and the objective of your game.

The structure of the adventure has been set up so that it can be played in several ways. The simple game is mere 'hack and slay' — players wander round the place finding monsters, killing them off and discovering treasures. The point of this kind of game is to get a large score and this is measured by the amount of treasure acquired, the number of monsters killed and the number of turns survived. A more rewarding type of game adds the idea of an ultimate goal, which is a single object which has to be found.

If the latter game is more your style, one object should be declared to be the ultimate goal. This is done by giving the value '2' to the object, in the LC column of the object data spreadsheet. You type 0 to make an object a unique specimen, or 1 to create many such objects.

Several other aspects of the game can be user-defined. The game revolves around combat, so most of the variables affect, or are affected by, combat in some way. In the object description section of the Generator (Procobdat), you can not only declare the location type (LC), but also whether an object has a curse and whether it is the Bane of a monster.

BANE: The parameter BN holds the number of the monster for which that object is the Bane. So if the characters try to fight the correct monster with this Bane, it will run away.

CURSE: The parameter CS is used to denote a cursed object — 0 for no curse, 1 for reducing the owner's attack in combat, 2 for causing injury to

the 'owner' of the object during combat and 3 for a 'spell' which will sometimes randomly transport the whole party to a new location. The adventurers can discover that they have a cursed object either by using it in combat (rather a drastic method) or by the fortuitous intervention of a mystic disembodied voice (**Procvoice**), which sometimes deigns to give information (not only about cursed objects, but also on the whereabouts of certain treasure) to the party when resting.

Resting, incidentally, has the benefit of randomly incrementing one variable for one character (**Procrest**) — but there is also the chance of a random routine being called, which may transport the party (if cursed) to a random location, create a magical pool (drinking from this may increase or decrease variables), or call up a surprise in the shape of a monster.

COMBAT: In combat, a furious attack will add Anger to Attack values, but subtract Anger from Defence. Caution, however, uses Nerves to supplement Defence but reduce Attack. So a Cautious attack reduces the enemy's chance of hitting, but also reduces yours. A Furious attack will have the opposite effect. Spells are missiles — they nearly always work, but may hit with no effect and anyway will never cause more than three points damage.

Monsters will die if *either* their Attack *or* Defence values fall to 0 or below, while characters die if *both* values fall to 0.

With careful design, an adventure can be created which mixes both the puzzle and combat elements. The 'dungeon' can be virtually a random maze, but with clever use of the descriptions you can make it more structured. For example, one room may have a Vulture and another a Vulture's Feather. This Feather could be either the Bane of, or a Gift for, that monster. In order to get the Feather, the characters might first have to defeat three Ogres. They may only be able to do this if they first find and drink a potion of Ogre strength (which gives a large Attack and Defence value, let us say). However, the potion could be cursed so that there is a risk that the party will be transported elsewhere in the dungeon, and waste their strength.

So the basic method is to make an object the key to passing the next monster, which just happens to be guarding the object which is the key to passing another monster — and so on. It is probably a good idea to use either the potion or the bottle as a general object which can be scattered all around the adventure and carried about until needed. Potions and bottles only confer their virtues (when drunk) on the user, as do all objects which have an LC value of 1, whereas other objects contribute Attack and Defence values every time there is a fight. On the other hand, all the other features of unique objects are passed to their owner the moment that the object is taken, so unique objects could be quite

powerful. A book of Spells, for example, could transfer nine spells to the character.

Try to give a rationale to some objects. You can give a clue in the name of the object, perhaps. For example, an orb of healing may confer Defence points, a bent sword may be cursed, a Giant axe could be a gift for an Ogre, a Red Feather could be a gift for a Flamingo. It is feasible to make the ultimate goal an object with some connection to a monster. In this case, a Wizard's Staff, for instance, may be guarded in surrounding cells by an army of Trolls, with the mighty Wizard himself at the heart of the maze.

Once you have finished the Herculean labour of typing in the listings (just think how long it took to write!), you will find it very rewarding to be able to create and play an infinite number of adventures with very little trouble. If you are particularly proud of an adventure you have created with the aid of QLAG and want to share it, we'd be very happy to see it.

Now, before you put on your typing fingers, here are some notes which you should find useful.

Notes on using QLAG

1. Declaring exits in the map routine (**Procmap**) automatically sets the equivalent exit in any adjacent room. To remove the exits in a room (for example, to set up one-way routes), select the exit option and press any key other than N, S, E or W. Removing exits, however, only removes them from the current room and *not* from an adjacent room. This could leave a room that can be entered but not left, so make sure that, if you intend to delete an exit completely, it is deleted from both cells.

2. To end the object description routine (**Procobject**), it is necessary to select an object and then press E to End. You cannot quit the routine without requesting an object.

3. If you want to change part of the description of a room (**Procdescribe**), the whole routine must be repeated — it is not possible to change the monsters in a cell without also retyping the description.

4. For this reason, *planning* is essential: before you start, map out the dungeon complex, decide what you want in each cell and stick to your plan. A few moments spent now saves a lot of time at the data input stage.

5. In the monster routine (**Procmonster**), Attack and Defence are used in combat, Anger decides whether a monster will attack or not, Spells are magical missiles, Greed determines whether a monster wants a treasure or not — and Nerve is a measure of Cowardice/Bravery.

6. The value given to each object (**Procobdat**) is a modifier applied to the character carrying the object, for Attack, Defence and Spells. Thus, **CS** is Curse. Curse type 1 reduces the chances of hitting an opponent in combat, while type 2 inclines a character towards hitting a friend rather than an enemy. Finally, type 3 is prone to transport a party into another, random, location.

LC specifies the location properties of an object. Thus, 0 denotes a unique object, one that only occurs once in the dungeon. The value 1 is applied to an object of which there can be many examples, but all have the same description. The value 2 is applied to the object for which all the characters are searching. When this object is found, the game is won.

BN is applied to the object which is the Bane of the monster at the location. This is obviously a pretty powerful object for, if one of the characters of a party is carrying it, the party will automatically vanquish the monster. However, only monsters 1 to 9 are affected in this way, so that the other monsters are potentially stronger — a Bane of 0 (BN = 0) means that this object affects no monster. If values greater than those allowed are entered, they will be recorded by QLAG but will have no effect on the game — that is, they will count as 0.

7. Character variables are roughly the same as the monster variables.

Exits

The following section may not mean very much to you at the moment, but should become clearer as you type in the program. Don't panic if you can't follow it here.

There are four possible exits from any room — N, S, E and W. In QLAG we need to be able to mark these exits on a map, and in QAD we need to draw the appropriate doors, excluding the door by which we enter the room.

This means, unfortunately, some elementary maths.

Identifying the exits

To identify the exits, we could use an array with three dimensions, 'number of cells horizontally', 'number of cells vertically' and 'number of possible exits'. This would give an array of $12 \times 24 \times 4 = 1152$ for each of the four possible exits. But this would be very wasteful when we consider that the information we want is binary — either an exit is available or it is not (ON or OFF, as it were). We could save a quarter of this space if we used a single bit to represent each possible exit, the last dimension in our array.

Suppose we use bit 0 (the rightmost bit) for S, bit 1 for W, bit 2 for N

and bit 3 for E. This means that South is represented by the decimal number 1, West by 2, North by 4 and East by 8. We will call these decimal numbers (1, 2, 4 and 8) the exit numbers, and the bit numbers (0–3) the exit codes. Our exits would look like:

```
        N(2)
(1)W            E(3)
        S(0)
```

If all four exits are present in one room, then the array will hold the number 1 + 2 + 4 + 8 = 15, which means that the decimal value of the cell's exits (which we will call the cell code) is 15.

In the programs, we will assign the exit code values by using the string 'SWNE' and subtracting 1 from the positions of the exits in that string. S is element 1 in the string and 1−1 gives us 0, the exit code for S; W is element 2 and 2−1 gives us 1, the exit code for W, and so on.

If we want to find the cell code, we use the basic formula:

cell code (decimal value of a cell's exits) = the sum of all exit numbers where each exit number is 2 ^ ((position in the string 'SWNE') − 1)

So if all four exits are present in one room, the decimal value of the cell's exits will be:

```
2^(1−1) + 2^(2−1) + 2^(3−1) + 2^(4−1)
= 2^0 + 2^1 + 2^2 + 2^3
= 1 + 2 + 4 + 8 = 15
```

Bitwise logic

To use this information in QLAG and QAD, we need to use 'bitwise logic'.

If you type

```
PRINT 4 && 12
```

you will get the result '4'. This is because the operator && causes two bytes to be compared so that, if the same bit of *both* of the two bytes is set, this bit remains set; but if a bit is set on *one* byte only, this bit will be reset to 0.

As decimal 12 sets bits 2 and 3 but decimal 4 sets only bit 2, then only bit 2 is set in both numbers. Bit 2 in decimal is 4, so 4 is returned by the bitwise AND function. If we compared decimal 12 with 1, where decimal

12 sets bits 2 and 3 and decimal 1 sets bit 0, the bitwise function will return 0.

The QL treats 0 as 'False' and any other number as 'True', and so will act on a condition such as:

```
IF 4 && 12 THEN PRINT "IT WORKS!!"
```

In other words, if we compare any number with 1, 2, 4 and 8 in turn by using &&, we will get 0 if the corresponding bit is not set, and either 1, 2, 4 or 8 if it *is* set.

Marking doors in QLAG

If we place the cursor in the centre of a cell in the map grid, to mark any given exit it has to move either a positive or a negative distance (a distance multiplied by +1 or −1) on either the vertical or horizontal axis. For example, to plot the North exit the cursor must move positively on the vertical axis, and to plot the West exit it must move negatively on the horizontal axis.

By using the bitwise AND function, we can translate the directions entered in Procentrance into the exit codes 0–3. We now have to translate these exit codes into either +1 or −1 (the direction to move on the axis) and also figure out which axis to apply them to.

If the exit code (referred to in the procedure as 'wall') is even (0 or 2), then the axis is vertical (S or N). If it is odd (1 or 3), then it is horizontal (W or E). To find out if a number is even, divide it by 2 using both real and integer division (/ and DIV). If both give the same result, the number is even; if not, it is odd.

The movement of each axis is held in Xw for horizontal and Yw for vertical. In the case of an even number, the movement is up/down so Xw is 0. Yw must therefore be either +1 or −1: we subtract 1 from our given number (0 or 2) to find out which. If the number is odd, then Yw is 0 and Xw will be either +1 or −1 (this is the ELSE case in QLAG **Procdoor_draw**). We subtract 2 if the number is odd (because the number will be either 1 or 3) to find out whether Xw is positive or negative.

The coordinates Xw and Yw are then simply added to the existing cursor coordinates and, hey presto!, the cursor is moved to the position for marking the door on the map.

Drawing doors in QAD

To draw doors in QAD itself, the position is not so simple. We must not only interpret the exit code but draw only the exits that the player can see (that is, excluding the one by which the party entered) and these exits

must be oriented correctly. This means that we need to know which exit was also the entrance, and the value of this is held in the variable 'entrance%'.

To draw the doors the player can see so that the exits are oriented correctly — ie the East door will be to the left of the North door, etc. — we can assign door codes for 'righthand door', 'lefthand door' and 'opposite door'.

If you move North, then you enter the next room from the South door (think about it — you will be facing North, so the door opposite you will be the North door, and so you have entered via the South door). Using the exit codes from our string 'SWNE'−1, the door we entered by, S, equals 0; W is on the left, and so the door on the left is code 1, 1 more than 0; N is ahead, and so the door opposite is code 2, 2 more than 0; E is to the right, and so the door on the right is code 3, 3 more than 0.

If we entered from the West door, the door we entered by would be 1, the exit code for W. The door on the left would be N=2, 1 more than 1, the entrance; the door opposite E=3, 2 more than 1. So far, this suggests that we can give door codes of 1 for 'lefthand door' and 2 for 'opposite door'. But, the righthand door is S=0, which is 1 less than 1, not 3 more as we would expect.

However, if we treat the room as a circle, going clockwise round the room we can number the exits from 0 to 3, and then continue to 4, 5, etc. Obviously, as we have only 4 exits, door 4 is the same as door 0 (and door 8, door 12 and so on). So, for the purposes of our door codes, we can treat door 0 as door 4. This is 3 more than 1, which is what we want.

So we can say that the door code for 'lefthand door' is 1, for 'opposite door' 2 and for 'righthand door' 3. We now have codes for the doors (left, opposite and right, 1–3) and for the exits (0–3). We need to have a code for the entrance.

We need to transform the value for the exit from one room, into the value for the entrance to the next. As we have already seen, if you leave a room via the North door, you enter the next room through the South door — so you enter through the 'opposite' door. Remembering that 2 is the door code for opposite, we need a string of entrance codes which is the inverse of the string of exits — this string of entrance codes is 'NESW'−1. So if we leave a room via the North door, we put the value of its entrance code (0) into entrance%, which means that, when we enter the next room via the South door, entrance% contains the value of the exit code for South (0, from the string 'SWNE'−1).

So in QAD **Procgraphic**, we take our cell code (contained in 'map%') and do a bitwise comparison with 2^0, 2^1, 2^2 and 2^3 (each, of course, in turn via a loop). If the test returns 'True', then the next step is to see whether this was the value of the door we entered by. If not, QAD **Procdoor_draw** is called (to draw the doors) with one of the numbers 0 to

3 as its parameter. This is repeated until all the possible exits have been checked.

For programming purposes, we can think of these door codes as the remainder from the calculation:

door code = (exit code + 4 − entrance code) MOD 4

(see **Procdoor_draw**). So if you enter a room via the East door, ie the last movement you made was West, and want to know where to draw the South exit, the procedure is:

1. The entrance code for W is 3 (position in 'NESW'−1).
2. The exit code for S is 0 (position in 'SWNE'−1).
3. 0 plus 4 equals 4.
4. 4 minus the entrance code (3) equals 1.
5. The remainder after dividing 1 by 4 is 1.
6. 1 is the door code for 'on the left'.
7. Therefore the S exit should be drawn on the lefthand wall (ie to the left of the East door).

Main variables in QLAG

map%	rooms across, rooms down, code for exits
	object number
	monster
	no. of monsters
map$	rooms across, rooms down, max string length
object$	number of objects, **pre** description
	object name
	post *description*
object%	objects, attack
	defence
	spell
	curse
	location
	bane
monster%	monsters, attack
	defence
	spell
	anger
	greed
	gift
character%	same parameters as monster%, *but* no gift

CHAPTER 6

The Adventure Generator — QLAG

This chapter and the next contain the program listings for QLAG and QAD, with commentaries as necessary. As far as possible, the procedures are discussed in the order in which they are called by the program, so that you can follow the sense of the programs. However, neither of the programs will actually do anything until the complete program has been entered.

Throughout the discussions of the two programs, we have occasionally referred to the QL manual. As with the operating system of the computer, the manual is continually being updated — the version we refer to is 6/84.

Program handler

```
100 REMark QL ADVENTURE GENERATOR
1984 N RICHARD WILLIAMS & TONY BRIDGE
110 RESTORE
120 init
130 REPeat supervisor
140 menu
150 get "1","6"
160 g=g$
170 SELect ON g
180 ON g = 1
190 infile
200 ON g=2
210 CLS
220 map
230 ON g=3
240 CLS
250 object
260 ON g=4
270 CLS
280 monster
290 ON g=5
```

```
300 CLS
310 character
320 ON g=6
330 EXIT supervisor
340 END SELect
350 END REPeat supervisor
360 file_out
370 STOP
```

Program handler commentary

This is the main handler for the whole program, and calls up each main procedure as required. The first procedure called is:

Procinit

```
12000 DEFine PROCedure init
12010 CLS
12020 AT 10,10:PRINT "Initialising...
."
12030 DIM map%(12,24,4)
12040 DIM map$(12,24,60)
12050 DIM object$(36,3,12)
12060 DIM object%(36,6)
12070 DIM monster$(16,8)
12080 DIM monster%(16,6)
12090 DIM character%(6,5)
12100 DIM character$(6,8)
12110 DIM category$(9,4)
12120 DIM axis$(6,2)
12130 FOR i = 1 TO 36
12140 READ object$(i,2)
12150 FOR j = 1 TO 6
12160 READ object%(i,j)
12170 NEXT j
12180 NEXT i
12190 FOR i=1 TO 12
12200 FOR j=1 TO 24
12210 FOR k= 1 TO 4
12240 map$(i,j)="nondescript"
12250 NEXT k:NEXT j:NEXT i
12255 monsterskilled=0:turns=0:h=1:v=
1
```

```
12256 char$="123"
12260 compass$="SWNE"
12270 FOR i=1 TO 16
12280 READ monster$(i)
12290 NEXT i
12300 FOR i=1 TO 9
12310 READ category$(i,1 TO 4)
12320 NEXT i
12330 FOR i = 1 TO 6
12340 READ character$(i)
12350 FOR j= 1 TO 5
12360 character%(i,j)=RND(1 TO 9)
12370 NEXT j
12380 NEXT i
12390 FOR i=1 TO 6
12400 READ axis$(i)
12410 NEXT i
12420 END DEFine init
```

Procinit commentary

The procedure dimensions several strings, each of which will hold information on a certain aspect of the map at any point. The map will take the form of a 12 × 24 matrix, each cell of which may (or may not) contain a number of monsters and objects. The map will be described by a short sentence, as dictated by the user (see **Procdescribe**). In **line 12260**, **compass$** contains the exit direction string discussed in the previous chapter.

Map% is an integer variable that holds, in its third element, information on the exits from each cell — there being four directions in which a possible exit may be made. **Map$** holds text messages in its third element, and will be used later for display on the map-altering screen. The integer variables **object%**, **monster%** and **character%** contain the values given to objects and so on, and **object$**, etc., contain the names and descriptions — these will be discussed later. The dimension statements show that we have 36 objects, 16 monsters and six characters. These are in the data statements, which are in line 12430 onwards, and the data is now read into the relevant strings and arrays. The data for **category$** is in line 12810 and the data for **axis$** in line 12830. The data lines are given at the end of this chapter.

Lines 12190–12250: When we start, all cells (or rooms) of the map will be empty, so these loops ensure this is so. **Line 12240** places a message to this effect in **map$**, and this will be printed later as required.

Procmenu

```
500 DEFine PROCedure menu
510 REMark OPEN A FULL SCREEN
520 OPEN #1, con_482x256a0X0
530 PAPER 2
540 CLS
550 AT 2,2:PRINT "Press the correct n
umber to:"
560 STRIP 0
570 AT 4,8:PRINT "load data from micr
odrive(1)"
580 AT 6,8:PRINT "alter the map
      (2)"
590 AT 8,8:PRINT "alter objects
      (3)"
600 AT 10,8:PRINT "alter the monsters
       (4)"
610 AT 12,8:PRINT "alter the characte
rs     (5)"
620 AT 14,8:PRINT "end the program
       (6)"
630 STRIP 2
640 END DEFine menu
```

Procmenu commentary

The main program handler, now control has returned from **Procinit**, enters a loop at line 130 (the supervisor loop). Initialising has been completed, and now we will always return to the menu.

Line 520: Opens a window which actually covers the whole screen (see Manual, Beginner's Guide). **Line 530** chooses a colour for the window — red here, but, of course, you may feel free to experiment with your favourite colours. **Line 560** chooses a Strip colour (see Manual, Keywords) on which to print the options which follow. Note in **line 520** that either upper *or* lower 'x' is valid.

Before we examine the items in the menu, we need to look at the next procedure called by the handler.

Procget

```
5370 DEFine PROCedure get(low$,hi$)
5380 REPeat kget
```

```
5390 g$=INKEY$
5400 IF g$>=low$ AND g$<=hi$ THEN EXI
T kget
5410 END REPeat kget
5420 END DEFine get
```

Procget commentary

This will be called several times throughout the Generator whenever a menu is printed, and sets low and high limits to numerical inputs, depending on the values of **low$** and **hi$** when Procget is called. Procget traps incorrect typing, repeating the process until a correct key is pressed — ie if a mistake is made the keypress is ignored. In the handler (at line 150), Procget sets a low limit of 1 and a high limit of 6 — so keypresses within this range are acted upon, and others are ignored.

The first selection on the menu that we shall discuss is 'alter the map', which we choose by pressing 2. This (lines 200–220 in the handler) means that our first task is to draw the map.

Procmap

```
1000 DEFine PROCedure map
1010 xpos=13.5:ypos=35
1020 CLS
1030 REMark CREATE THREE WINDOWS
1040 OPEN #10,scr_320x160a32x16
1050 PAPER #10,3
1060 OPEN #11,con_440x64a32x176
1070 PAPER #11,5
1080 OPEN #13,scr_120x160a352x16
1090 PAPER #13,0
1100 CSIZE #10,0,0
1110 CLS #10
1120 CLS #11
1130 CLS #13
1140 INK #10,0
1150 SCALE #10,160,0,0
1160 FOR i = 10 TO 130 STEP 10
1170 LINE #10, 10,i+20 TO 226,i+20
1180 NEXT i
1190 FOR i=9 TO 225 STEP 9
1200 LINE #10, i,30 TO i,150
```

```
1210 NEXT i
1220 INK #10,7
1230 FOR i = 97 TO 120
1240 AT #10,0,i-96 :PRINT #10,CHR$(i)
1250 NEXT i
1260 FOR i = 1 TO 12
1270 AT #10,i,0 :PRINT #10,CHR$(109-i
)
1280 NEXT i
1290 CSIZE #10,3,1
1300 PRINT #10, "     ADVENTURE MAP"
1310 FOR i = 1 TO 12
1320 FOR j= 1 TO 24
1330 IF map%(i,j,1)=0 THEN GO TO 1370
1340 FOR k=0 TO 3
1350 IF (map%(i,j,1) && 2^k) THEN doo
r_draw k,j,i
1360 NEXT k
1370 NEXT j
1380 NEXT i
1390 INK #11,0
1400 scr_mes
1410 vp=1:hp=1
1420 REMark THE CURSOR ROUTINE
1430 cur 0,0,9,10,220.5,13.5,145,35,-
2
1440 room
1450 REPeat control
1460 g$=INKEY$
1470 g=CODE(g$)
1480 SELect ON g
1490 ON g= 192
1500 cur -9,0,9,10,220.5,13.5,145,35,
-2
1510 room
1520 ON g= 200
1530 cur 9,0,9,10,220.5,13.5,145,35,-
2
1540 room
1550 ON g= 208
1560 cur 0,10,9,10,220.5,13.5,145,35,
-2
1570 room
1580 ON g=216
```

```
1590 cur 0,-10,9,10,220.5,13.5,145,35
,-2
1600 room
1610 ON g= 232
1620 CLOSE#10
1630 CLOSE#11
1640 CLOSE#13
1650 EXIT control
1660 ON g= 69,101
1670 entrance
1680 scr_mes
1690 ON g= 68,100
1700 describe
1710 scr_mes
1720 END SELect
1730 END REPeat control
1740 END DEFine map
```

Procmap commentary

xpos and **ypos** are the coordinates of the position where the cursor will eventually be placed when the grid is drawn, and the early appearance here of these two variables is really to get the matter out of the way!

Lines 1040–1100: The windows are defined and opened, and the paper colour set. Window #10 contains our map, and #11 a smaller display panel in which information is posted on what is in the current cell (information held in **map$**). Finally, window #13 is for system messages and prompts to the user.

Lines 1110–1140: The character size is set (see Manual, Keywords) the windows cleared, and ink colour set.

Lines 1150–1210: Now the scale of the x axis is defined (see Manual, Keywords) and the map is drawn. **Lines 1220–1280:** And now the letters (**lines 1230–1240**), in lower case, are printed along the x and y axes of the map-grid.

Lines 1290–1300: Give us the largest character size possible (see Manual, Keywords) and then print 'ADVENTURE MAP' beneath the map.

Lines 1310–1380: Draw the exits in each cell of the map, if they have been defined by the user — for now, of course, this part of the procedure will not find any exits and thus keeps repeating until all cells are checked. Later, as exits have been defined and then found, **Procdoor_draw** will be called, by **line 1350**, which uses a pretty neat bit of maths to find if an exit has been assigned to the cell currently under investigation. See the Exits discussion in the previous chapter for a deeper explanation of all this.

Remember that if you delete an exit in one room you must also delete that exit in the adjacent room.

Line 1390: Now window #11's ink is set to black and in **line 1400** **Procscr_mes** is called. This prints the instructions beneath the map (in window #11, as we shall see).

Line 1410: Sets the initial position of the cursor — the values of **vp** and **hp** (vertical position and horizontal position) will be used in **Proccur**, which is called in **line 1430** (which also sets the initial parameters in **Proccur**).

Line 1440: Calls **Procroom**. As we shall see, this procedure gives information, in window #13, on what has been assigned to the current cell in the way of monsters and objects, as well as the description that has been given to the location.

Lines 1450–1730: The control loop. The code of the cursor key which has been pressed is read. Look, for example, at **line 1490** — 192 (decimal) is the character code for cursor left and, if this key is pressed, the first parameter of the variable **cur** is set from 0 (set at **line 1430**) to −9, moving the cursor to the left. Turn to **Proccur**, line 1750, and you'll see that the first parameter is **xinc** — this is the one that is changed. On **line 1590**, however, the second parameter **yinc** at this point is set to −10, sending our cursor down. In **line 1610**, 232 is the decimal code for the F(unction) 1 key, and this ends the whole sequence and exits to the menu (that is, we go back to the supervisor loop at the start of the program). After each of the cursor changes, **Procroom** is called to redraw the map. After F1 is re-pressed, the windows are closed and all files closed down.

Line 1660: If, however, the key pressed is E (for Exit), **Procentrance** is called. Notice that both upper (E, code 69) and lower (e, code 101) cases are catered for. The user shouldn't have to worry whether or not capitals are required.

Line 1680: The main screen prompts are displayed once more (**Procscr_mes**). **Line 1690:** If D (for Description) is pressed, then **Procdescribe** is called.

And so we carry on, until we have visited each cell we wish to deal with. All the parameters we have changed while using the routine have been placed in the final element of **map%**, with the position of each cell being read into **vp** and **hp**. Thus, when the cursor is positioned at, say, 'ft' on the map, **vp** becomes 'f', and **hp** becomes 't'. Then the program looks to the final element of **map%** and finds there the contents and exits of cell ft.

Now let's look more closely at the various procedures called from Procmap.

Proccur

```
1750 DEFine PROCedure cur (xinc,yinc,
xint,yint,xr,xl,yu,yd,var)
1760 INK #10,3
1770 LINE #10,xpos,ypos-1 TO xpos,ypo
s+1
1780 xpos=xpos+xinc:ypos=ypos+yinc
1790 IF xpos>xr THEN xpos=xr
1800 IF xpos<xl THEN xpos=xl
1810 IF ypos<yd THEN ypos=yd
1820 IF ypos>yu THEN ypos=yu
1830 vp=(ypos DIV yint)+var:hp=xpos D
IV xint
1840 CURSOR #10,xpos,ypos
1850 REMark CURSOR POSITION
1860 CSIZE #10,0,0
1870 INK #10,7
1880 LINE #10,xpos,ypos TO xpos,ypos
1890 END DEFine cur
```

Proccur commentary

This procedure sets the position of the little cursor in each cell as the cursor keys are pressed.

Line 1750: The parameters in brackets here receive information passed to them from the variable **control** (lines 1450–1650 in **Procmap**).

Line 1770: This draws a line, but it is extremely short — one pixel in fact! So the cursor effectively becomes a dot. **Lines 1780–1820:** Starts the cursor in the bottom left cell of the map (a,a), and updates the cursor position using the first parameters in **line 1750**. **Lines 1790–1820** ensure that the cursor is kept within the confines of the map.

Line 1830: Matches the position of the cursor against the variable **map%**, enabling the program to keep track of what contents to print in window #10.

Line 1840: Overprints the old cursor with the same colour as the paper, thus effectively making the cursor disappear. **Line 1880** prints the cursor at the new position.

Procscr_mes

```
2830 DEFine PROCedure scr_mes
2840 CLS#11
2850 PRINT #11,"Cursor keys move curs
```

```
or, E selects exits, D sets descripti
ons, F1 to end"
2860 END DEFine scr_mes
```

Procscr_mes commentary
Clears window #11 and prints the option menu.

Procroom

```
2750 DEFine PROCedure room
2760 CLS#13
2770 PRINT #13;"Cell ";CHR$(vp+96);",
";CHR$(hp+96)
2780 IF map$(vp,hp,1)<>"" THEN PRINT
#13;map$(vp,hp,1 TO 60)
2781 FOR k=0 TO 3
2782 IF (map%(vp,hp,1)&& (2^k)) THEN
PRINT #13,compass$(k+1);" ";
2783 NEXT k
2784 PRINT #13," "
2790 IF map%(vp,hp,2)<>0 THEN PRINT #
13; object$(map%(vp,hp,2))
2800 IF map%(vp,hp,3)<>0 THEN PRINT #
13;map%(vp,hp,4);" ";monster$(map%(vp
,hp,3));
2810 IF map%(vp,hp,4)>1 THEN PRINT #1
3;"s"
2820 END DEFine room
```

Procroom commentary
This procedure clears window #13, the window reserved for descriptions of the room (or cell) that we are inspecting, and then examines **map$** and **map%**. Depending on the results, the information is then printed to the window. Note the bitwise AND function in **line 2782**, which checks on the exits present.

Procentrance

```
2100 DEFine PROCedure entrance
2110 CLS #11
2120 PRINT #11;"Type in all the direc
tions"!"then press enter"
```

```
2130 INPUT #11, direction$
2135 map%(vp,hp,1)=0
2140 FOR i = 1 TO LEN(direction$)
2150 d$=direction$(i)
2160 z=d$ INSTR compass$
2170 IF z<>0 THEN exproc
2180 NEXT i
2190 END DEFine entrance
```

Procentrance commentary

If the player has chosen the Exit option from the **Procscr_mes** menu, this procedure clears that menu from window #11 and prints a new message. Once a direction or directions (first letters only are needed) are typed in, they are placed in **direction$**. **Compass$** is checked and the variable **z** will be given the value of each direction chosen (**lines 2140–2150**) — the values are S=1, W=2, N=3 and E=4 (see line 12260 in **Procinit** and also the discussion of Exits in the previous chapter). The value of **z** is placed in **map%**. **Line 2170** calls:

Procexproc

```
11250 DEFine PROCedure exproc
11260 IF z=1 AND vp=1 THEN RETurn
11270 IF z=2 AND hp=1 THEN RETurn
11280 IF z=3 AND vp=12 THEN RETurn
11290 IF z=4 AND hp=24 THEN RETurn
11300 SELect ON z
11310 ON z=1
11320 map%(vp-1,hp,1)=map%(vp-1,hp,1)
!!4
11330 ON z=2
11340 map%(vp,hp-1,1)=map%(vp,hp-1,1)
!!8
11350 ON z=3
11360 map%(vp+1,hp,1)=map%(vp+1,hp,1)
!!1
11370 ON z=4
11380 map%(vp,hp+1,1)=map%(vp,hp+1,1)
!!2
11390 END SELect
11400 map%(vp,hp,1)=map%(vp,hp,1)!!(2
^(z-1))
11410 door_draw z-1,hp,vp
11420 END DEFine exproc
```

Procexproc commentary

This procedure defines the exits. As the user types in the choice of exits, the information is placed in the variable **map%** and the parameters are passed to **Procdoor_draw**. **Lines 11320**, **11340**, **11360** and **11380** use the bitwise OR function (||), which effectively adds the number after the symbol to **map%** (unless the relevant bit is already set). These lines mark exits in the adjoining cells, as relevant. **Line 11400** deals with exits in the current room.

Lines 11260–11290 prevent exits being drawn on the borders of the map.

Procdoor_draw

```
2200 DEFine PROCedure door_draw(wall,
cem,brik)
2210 LOCal c:LOCal b:LOCal w
2220 c=cem:b=brik
2230 w=wall
2240 IF w DIV 2=w/2 THEN
2250 w=w-1
2260 Yw=w
2270 Xw=0
2280 ELSE
2290 w=w-2
2300 Xw=w
2310 Yw=0
2320 END IF
2330 LINE #10,(4.5+(c*9)+(Xw*4.5)),(2
5+(b*10)+(Yw*5))
2340 LINE_R #10 TO 1,1
2350 END DEFine door_draw
```

Procdoor_draw commentary

This procedure translates the direction values into movement on the horizontal or vertical axis. **Line 2240** checks if the value is even, in which case movement is on the horizontal axis (Yw=0 and Xw=+1 or −1). If it is not, the ELSE statement in **line 2280** comes into effect and movement is on the vertical axis (Xw=0 and Yw=+1 or −1). See the Exits section in the previous chapter for more details. **Lines 2330–2340** contain the calculations to move the cursor a positive or negative distance on the horizontal or vertical axis and to draw the exit.

Procdescribe

```
1900 DEFine PROCedure describe
1910 CLS #11
1920 PRINT #11;"TYPE IN THE CELL DESC
RIPTION (max 60 characters)"
1930 INPUT #11;map$(vp,hp)
1940 CLS#11
1950 PRINT #11; "Any monster here?(Y/
N)"
1960 REPeat get_loop
1970 g$=INKEY$
1980 IF g$<>"" THEN EXIT get_loop
1990 END REPeat get_loop
2000 IF g$="y" OR g$="Y" THEN assign_
mon
2010 CLS #11
2020 PRINT #11; "Any object here? (Y/
N)"
2030 REPeat get_loop
2040 g$=INKEY$
2050 IF g$<>"" THEN EXIT get_loop
2060 END REPeat get_loop
2070 IF g$="y" OR g$="Y" THEN assign_
object
2080 CLS#11
2090 END DEFine describe
```

Procdescribe commentary

This procedure processes the user's input, in answer to the prompts, calling a couple of other procedures, **Procassign_mon** and **Procassign_object**. The description of each cell ('a cold, dark tunnel', 'a throne-room be-draped with velvet curtains', 'a cold, airless crater on the moon', or whatever takes your fancy) is put into **map$ (line 1930)**. The variables **vp** and **hp** ensure that the program keeps track of which cell each description refers to. If you want to change any part of the description of a cell, you must repeat all the information for that cell.

Procassign_mon

```
2360 DEFine PROCedure assign_mon
2370 FOR i= 1 TO 16
2380 PRINT #13,i;" ";monster$(i)
```

```
2390 NEXT i
2400 CLS#11
2410 PRINT #11;"Type the number of th
e monster"
2420 INPUT #11, map%(vp,hp,3)
2430 CLS #11
2440 PRINT #11;"How many ";monster$(m
ap%(vp,hp,3));"s in this room? (maxim
um 3)"
2450 INPUT #11,map%(vp,hp,4)
2455 IF map%(vp,hp,4)>3 THEN map%(vp,
hp,4)=3
2460 END DEFine assign_mon
```

Procassign_mon commentary

This procedure prints, in window #13, a list of the monsters available (16 of them — data lines 12790–12800). In the message/prompt window, #11, it asks the user to type in the number of the monster (**lines 2370–2380**) and places this information in **map%** (in the third element of the array), and then how many of each monster are required — this information goes in the fourth element of **map%**. You can only have one type of monster per cell. Don't forget that **vp** and **hp** are the coordinates of the current cell.

Procassign_object

```
2470 DEFine PROCedure assign_object
2480 CLS#11
2490 CLS#13
2500 PRINT #11;"Which object in this
room?"
2510 PRINT #11;"(Type name OR number)
"
2520 INPUT#11; temp$
2530 IF LEN(temp$)>2 THEN
2540 name_dec
2550 ELSE
2560 num_dec
2570 END IF
2580 IF t%=0 THEN
2590 PRINT #11;"No such object"
2600 PAUSE 60
2610 ELSE
```

```
2620 map%(vp,hp,2)=t%
2630 END IF
2640 END DEFine assign_object
```

Procassign_object commentary

This does more or less the same as **Procassign_mon**, but for the objects. There is no list shown this time, as we have some 36 objects (data lines 12430–12780): if we wished to display all of these at one time, we would have to go into mode 8, so the program here accepts either a number *or* a name. You can only have one object per cell.

Lines 2520–2620: Place the input in a temporary string, **temp$** — if the length of this string is *greater* than 2 (**line 2530**), then we assume that it is a name, and **Procname_dec** is called. Otherwise, **Procnum_dec** is called. The result of calling either of these procedures (which will be discussed in their turn), is put into **map%**, in **line 2620**.

Procnum_dec

```
2650 DEFine PROCedure num_dec
2660 t%=temp$
2670 IF t%<1 OR t%>36 THEN t%=0
2680 END DEFine num_dec
```

Procnum_dec commentary

Assigns the number of the monster or object input by the user to **t%**.

Procname_dec

```
2690 DEFine PROCedure name_dec
2700 t%=0
2710 FOR j=1 TO 36
2720 IF temp$ INSTR object$(j,2) THEN
 t%=j
2730 NEXT j
2740 END DEFine name_dec
```

Procname_dec commentary

Checks for the existence of the object name in **object$** — if valid, line 2620 in **Procassign_object** places the value in **map%**. If *not* valid, the INSTR

function in **line 2720** (see Manual, Keywords) gives the value 0 to **t%**, and this is acted on in line 2580, in **Procassign_object**.

Now we have completed the initial set-up of the adventure map, and we have a 12 × 24 grid of cells, or rooms, each with a selection of exits, monsters and objects, all according to the user's wishes. So we go back to lines 230–250 in the supervisor loop (in the handler). These lines are activated by a keypress of '3' (option 3, line 590 in **Procmenu**) which takes us to the routine which will alter data on the objects. **Procobject** is now called.

Procobject

```
4000 DEFine PROCedure object
4010 REPeat choose
4020 CLS
4030 AT 5,4:PRINT"Do you want to:"
4040 PRINT "       change object data
[1-18]  (1)"
4050 PRINT "           or object data
[19-36] (2)"
4060 PRINT"           or descriptions
       (3)"
4070 PRINT "           or end routine
         (4)"
4080 get "1","4"
4090 g=g$
4100 SELect ON g
4110 ON g=1,2
4120 obdat
4130 ON g=3
4140 obdesc
4150 ON g=4
4160 EXIT choose
4170 END SELect
4180 END REPeat choose
4190 END DEFine object
```

Procobject commentary

This procedure starts by printing a mini-menu — note here that it is important to get lines of print, like these, away from the edge of the screen, where they can so easily be lost.

Line 4080: Procget is called again to define limits of 1–4 and then, depending on the number chosen, **Procobdat** (1 and 2) or **Procobdesc** (3) are chosen, or exit (4). The 'change object data' routine is spread over two screens, as there are so many to deal with.

Procobdat

```
4200 DEFine PROCedure obdat
4210 kon=g
4220 map_screen kon,3,1,6,18,2
4230 vp=2:hp=9
4240 ov=vp:oh=hp
4250 REPeat control
4260 g$=INKEY$
4270 g=CODE(g$)
4280 SELect ON g
4290 ON g= 48 TO 57
4300 object%(vp-1+((kon-1)*18),(hp/3)
-2)=g-48
4310 ON g= 192
4320 hp=hp-3
4330 IF hp<9 THEN hp=9
4340 ON g= 200
4350 hp=hp+3
4360 IF hp>24 THEN hp=24
4370 ON g=208
4380 vp=vp-1
4390 IF vp<2 THEN vp=2
4400 ON g=216
4410 vp=vp+1
4420 IF vp>19 THEN vp=19
4430 ON g=232
4440 EXIT control
4450 END SELect
4460 AT #10,ov,oh:INK #10,0:PRINT #10
,object%(ov-1+((kon-1)*18),(oh DIV 3)
-2)
4470 AT #10,vp,hp:INK #10,7:PRINT #10
;object%(vp-1+((kon-1)*18),(hp DIV 3)
-2):ov=vp:oh=hp
4480 END REPeat control
4490 END DEFine obdat
```

Procobdat commentary

Line 4210 sets up a local variable, **kon**, then calls up **Procmap_screen**. We haven't seen this procedure yet, but you will find that it changes the values on the spreadsheet (the object spreadsheet in this case) to the user's requirements. **Lines 4280–4430** read the keypresses. On the spreadsheet set up and displayed in **Procmap_screen**, the initial position of the cursor is top left. It takes the form of a flashing number — ie the number in the top left position flashes. If a cursor key is read from the keyboard, then the cursor moves in that direction — the number in the new position flashes. Type a number between 0 and 9 (only one digit is permitted — the code numbers are 48 to 57, see **line 4290**), and that number is substituted in the spreadsheet, whose original values have been read in from the data statements at the end of the program (**Procmap_screen** will finally put these values into **object%**).

Procobdesc

```
4500 DEFine PROCedure obdesc
4510 disob
4520 messob
4530 comob
4540 REPeat loop
4550 CLS#12
4560 INK #12,2
4570 PRINT #12,"Type number and press
 <ENTER>":INPUT#12; number
4580 IF number<1 OR number>36 THEN EN
D REPeat loop
4590 x=(number-1) MOD 12:y= ((number-
1) DIV 12)*12
4600 AT #10,x+1,y+3:INK #10,1:PRINT#1
0;object$(number,2)
4610 CLS#11:PRINT#11,object$(number,1
);" ";object$(number,2);" ";object$(n
umber,3)
4620 REPeat comloop
4630 PRINT#12,"B = change text Before
 keyword"
4640 PRINT#12,"A = change text After
keyword"
4650 PRINT #12,"N = Next keyword (sto
res current text)"
4660 PRINT #12,"E = End choice"
```

```
4670 get "A","n"
4680 g= CODE(g$)
4690 SELect ON g
4700 PRINT #11,g
4710 ON g=65,97
4720 after
4730 ON g=66,98
4740 before
4750 ON g=69,101
4760 EXIT loop
4770 ON g=110,78
4780 EXIT comloop
4790 END SELect
4800 END REPeat comloop
4810 CLS#11:AT #10,x+1,y+3:INK #10,2:
PRINT#10;object$(number,2)
4820 END REPeat loop
4830 END DEFine obdesc
```

Procobdesc commentary

Not only can we change the various properties of the objects, but we can also change the descriptions of those objects. This is handled by this procedure. In its turn, it calls three more procedures, **Procdisob**, **Procmessob** and **Proccomob**. In window #12 is printed the name of the object and then a mini-menu (**lines 4630–4660**), giving the user four choices — note that to quit the routine you need to select an object and then press E to end (you cannot just press E). Our old friend **Procget** is called yet again for the occasion, and a couple more procedures are called, **Procbefore** and **Procafter** (**lines 4710–4740**).

Procdisob

```
4840 DEFine PROCedure disob
4850 CLS
4860 OPEN #10,scr_464x134a16x16
4870 PAPER#10,3
4880 INK #10,0
4890 CLS#10
4900 BORDER #10,2,1
4910 PRINT #10,"   Name          Name
      Name"
5010 FOR i = 1 TO 12
```

```
5020 AT#10,i,0:IF i<10 THEN PRINT#10,
"0";:AT #10,i,1:PRINT #10!i!object$(i
,2)!
5030 IF i>9 THEN AT#10,i,0:PRINT#10!i
!object$(i,2)!
5040 AT#10,i,12:PRINT#10!i+12!object$
(i+12,2)!
5050 AT#10,i,24:PRINT#10!i+24!object$
(i+24,2)!
5060 NEXT i
5070 END DEFine disob
```

Procdisob commentary

This procedure displays a list of all the objects in window #10.

Procmessob

```
5080 DEFine PROCedure messob
5090 OPEN#11,scr_464x32a16x150
5100 PAPER#11,0
5110 INK #11,7
5120 CLS#11
5130 BORDER#11,3,5
5140 END DEFine messob
```

Procmessob commentary

Opens a window, #11, for text input. You should notice the small difference between the first parameters of each window definition in this procedure and the next, **Proccomob**. In **line 5090**, the first parameter is 'scr' (screen), while the first parameter in line 5160 of **Proccomob** is 'con' (console). The first is for display only, while the other is for your keyboard input.

Proccomob

```
5150 DEFine PROCedure comob
5160 OPEN #12,con_464x64a16x182
5170 PAPER#12,7
5180 CLS#12
5190 BORDER#12,3,1
5200 END DEFine comob
```

Proccomob commentary
This procedure sets up a window, #12, and allows input to the window (notice the 'con' statement in **line 5160**).

Procbefore

```
5210 DEFine PROCedure before
5220 in_text
5230 object$(number,1)=text$
5240 CLS#11
5250 PRINT #11,object$(number,1);" ";
object$(number,2);" ";object$(number,
3)
5260 END DEFine before
```

Procbefore commentary
Takes the input (the description) and places it in **object$** (number,1), so that it appears *before* the object name.

Procafter

```
5270 DEFine PROCedure after
5280 in_text
5290 object$(number,3)=text$
5300 CLS#11
5310 PRINT #11,object$(number,1);" ";
object$(number,2);" ";object$(number,
3)
5320 END DEFine after
```

Procafter commentary
Does the same as **Procbefore**, to put the description after the object name.

Procin_text

```
5330 DEFine PROCedure in_text
5340 CLS#12
5350 AT #12,2,24:PRINT #12,"_________
___"
```

```
5355 AT #12,2,0:INPUT #12;"Type the d
escription ",text$
5360 END DEFine in_text
```

Procin_text commentary

Each of the previous procedures calls this little procedure, which prompts the user to input his text.

The program returns to the main menu, should the F1 key (g=232) be pressed at line 4430 in **Procobdat**. Now the user can go on to alter the monster data, or the character data.

Procmonster

```
6000 DEFine PROCedure monster
6010 map_screen 3,3,1,6,16,4
6020 vp=4:hp=9:ov=vp:oh=hp
6030 ov=vp:oh=hp
6040 REPeat control
6050 g$=INKEY$
6060 g=CODE(g$)
6070 SELect ON g
6080 ON g= 48 TO 57
6090 monster%(vp-3,(hp/3)-2)=g-48
6100 ON g= 192
6110 hp=hp-3
6120 IF hp<9 THEN hp=9
6130 ON g= 200
6140 hp=hp+3
6150 IF hp>24 THEN hp=24
6160 ON g=208
6170 vp=vp-1
6180 IF vp<4 THEN vp=4
6190 ON g=216
6200 vp=vp+1
6210 IF vp>19 THEN vp=19
6220 ON g=232
6230 EXIT control
6240 END SELect
6250 AT #10,ov,oh:INK #10,0:PRINT #10
,monster%(ov-3,(oh DIV 3)-2)
6260 AT #10,vp,hp:INK #10,7:PRINT #10
```

```
;monster%(vp-3,(hp DIV 3)-2):ov=vp:oh
=hp
6270 END REPeat control
6280 END DEFine monster
```

Procmonster commentary

This procedure is going to change the stats for our monsters, and is called from the opening supervisor loop and opening menu (option 4), at line 280 in the handler. Like **Procobdat**, the present procedure first of all passes its own values to **Procmap_screen** (at **line 6010**). The remaining lines are pretty much the same as **Procobdat**, except that the values now assigned by the user are put into **monster%** (at **lines 6250–6260**).

Proccharacter

```
7000 DEFine PROCedure character
7010 map_screen 4,3,1,5,6,4
7020 vp=6:hp=9
7030 ov=vp:oh=hp
7040 REPeat control
7050 g$=INKEY$
7060 g=CODE(g$)
7070 SELect ON g
7080 ON g=78,110
7090 name vp
7100 ON g= 48 TO 57
7110 character%(vp-5,(hp/3)-2)=g-48
7120 ON g= 192
7130 hp=hp-3
7140 IF hp<9 THEN hp=9
7150 ON g= 200
7160 hp=hp+3
7170 IF hp>21 THEN hp=21
7180 ON g=208
7190 vp=vp-1
7200 IF vp<6 THEN vp=6
7210 ON g=216
7220 vp=vp+1
7230 IF vp>11 THEN vp=11
7240 ON g=232
7250 EXIT control
7260 END SELect
```

```
7270 AT #10,ov,oh:INK #10,0:PRINT #10
,character%(ov-5,(oh DIV 3)-2)
7280 AT #10,vp,hp:INK #10,7:PRINT #10
;character%(vp-5,(hp DIV 3)-2):ov=vp:
oh=hp
7290 END REPeat control
7300 END DEFine character
```

Proccharacter commentary

If the user opts to change the data of a character, by pressing key 5 from the opening menu, **line 310** in the handler calls this procedure. Again, it is very similar to **Procobdat**, passing its own values to **Procmap_screen** — the changed values are then placed in **character%** (at **lines 7270–7280**). The names of the characters may also be changed (**lines 7080–7090** and **Procname**).

Procname

```
11000 DEFine PROCedure name(pos)
11010 AT#10,pos,0:PRINT #10,"________
"
11020 initial$=""
11030 FOR i=1 TO 6
11040 initial$=character$(i,1)&initia
l$
11050 END FOR i
11060 count=1
11070 name$=""
11080 REPeat loop
11090 REPeat gloop
11100 g$=INKEY$
11110 IF g$<>"" THEN EXIT gloop
11120 END REPeat gloop
11130 IF CODE(g$)>90 THEN g$=CHR$(COD
E(g$)-32)
11140 IF count =1 AND (g$ INSTR initi
al$<>0) THEN AT #10,18,0:PRINT#10, "I
llegal character":EXIT loop
11150 AT #10,pos,count-1:PRINT#10,g$
11160 name$=name$ & g$
11170 IF g$=CHR$(10) OR count>7 THEN
EXIT loop
```

```
11180 count=count+1
11190 END REPeat loop
11200 IF count<8 THEN
11210 AT#10,pos,count-1:PRINT #10;FIL
L$(" ",(9-count))
11220 END IF
11230 character$(pos-5)=name$
11240 END DEFine name
```

Procname commentary

Now presented with the character spreadsheet, the user is able to change the names of the characters, initially set as per the DATA statements at the end of the program. By pressing N (for Name) (see **Procmap_screen**), the name presently displayed at the position of the cursor is blanked out, and spaces made available for a new name to be typed in (**line 11010**).

Procmap_screen

```
10000 DEFine PROCedure map_screen(con
trol,cellh,cellv,toth,totv,totcat)
10010 OPEN #10,scr_432x216a32x8
10020 BORDER #10,1,0
10030 PAPER #10,3
10040 INK#10,0
10050 CLS#10
10060 CSIZE #10,0,0
10070 OPEN #11,SCR_432X32A32X224
10080 CLS #11
10090 PRINT #11;"Cursor keys move cur
sor."!"Change a value by overtyping i
t."!"F1 to end routine.";
10095 IF control=4 THEN PRINT #11," P
ress N to change name"
10100 FOR i= 1 TO totv
10110 SELect ON control
10120 ON control = 1
10130 AT #10,(i*cellv)+1,0;:PRINT #10
;object$(i,2)
10140 ON control = 2
10150 AT #10,(i*cellv)+1,0;:PRINT #10
;object$(i+18,2)
10160 ON control = 3
```

```
10170 AT #10,(i*cellv)+3,0;:PRINT #10
,monster$(i)
10180 ON control = 4
10190 AT #10,i*cellv+5,0;:PRINT #10;c
haracter$(i)
10200 END SELect
10210 NEXT i
10220 FOR i = 1 TO toth
10230 FOR j=1 TO totcat
10240 SELect ON control
10250 ON control=3,4
10260 AT #10,j-1,(i*cellh)+6;:PRINT #
10,category$(i,j)
10270 ON control=1,2
10280 AT #10,j-1,(i*cellh)+6;:PRINT #
10,axis$(i,j)
10290 END SELect
10300 NEXT j
10310 NEXT i
10320 FOR i=1 TO totv
10330 FOR j=1 TO toth
10340 SELect ON control
10350 ON control=1
10360 AT #10,(i*cellv)+1,(j*cellh)+6;
: PRINT #10;object%(i,j)
10370 ON control=2
10380 AT #10,(i*cellv)+1,(j*cellh)+6;
: PRINT #10;object%(i+18,j)
10390 ON control=3
10400 AT #10,(i*cellv)+3,(j*cellh)+6;
: PRINT #10;monster%(i,j)
10410 ON control=4
10420 AT #10,(i*cellv)+5,(j*cellh)+6;
: PRINT #10;character%(i,j)
10430 END SELect
10440 NEXT j
10450 NEXT i
10460 END DEFine map_screen
```

Procmap_screen commentary

This is the procedure that is used by each of the procedures **Procobject**, **Procmonster** and **Proccharacter**: the first parameter in **line 10000** receives

its initial value from line 4220 in **Procobdat**, line 6010 in **Procmonster** and line 7010 in **Proccharacter**. The other parameters set the horizontal and vertical spacing of the grid ('cellh' and 'cellv'), the horizontal and vertical increments ('toth' and 'totv'), and displays the nine categories along the top of the spreadsheet (these are in data line 12810). Then a window #10 is defined, in which the spreadsheet will be displayed, and a window #11 in which the prompt menu will be printed (**line 10090**).

Depending on the procedure from which Procmap_screen has been called, the program now prints out the contents of the various strings as required, in the form of a grid. Down the lefthand side will be printed the names of the monsters, objects or characters, while along the top will be shown the categories (referring to Attack, Defence and Spell points, and so on).

So now we have covered all the choices available to the user from the main menu, except for two important facilities, options 1 and 6. Option 1 is to load in data from a microdrive, from a previous session with the Generator.

But it would make more sense for the moment to look at option 6, which is to end the program. Before signing off completely, the program allows the user to save all that hard-won data.

Procfile_out

```
9000 DEFine PROCedure file_out
9010 PRINT "Please type name of game"
9020 INPUT game$
9030 PRINT"Place the cartridge into M
icrodrive 1"
9040 PRINT "And press <Enter>"
9050 REPeat get_loop
9060 in$=INKEY$
9070 IF in$=CHR$(10) THEN EXIT get_lo
op
9080 END REPeat get_loop
9090 gn$="MDV1_"& game$ & "numbers"
9100 OPEN_NEW #9, gn$
9110 FOR i=1 TO 36
9120 FOR j=1 TO 6
9130 PRINT #9,object%(i,j)
9140 NEXT j
9150 NEXT i
9160 FOR i=1 TO 12
```

```
9170 FOR j=1 TO 24
9180 FOR k=1 TO 4
9190 PRINT #9,map%(i,j,k)
9200 NEXT k
9210 NEXT j
9220 NEXT i
9230 FOR i=1 TO 16
9240 FOR j=1 TO 6
9250 PRINT #9,monster%(i,j)
9260 NEXT j
9270 NEXT i
9280 FOR i=1 TO 6
9290 FOR j=1 TO 5
9300 PRINT #9,character%(i,j)
9310 NEXT j
9320 NEXT i
9322 PRINT#9,turns
9323 PRINT#9,monsterskilled
9324 PRINT#9,v
9325 PRINT#9,h
9330 CLOSE #9
9340 gm$="mdv1_"&game$&"names"
9350 OPEN_NEW #9, gm$
9360 FOR i= 1 TO 12
9370 FOR j=1 TO 24
9380 PRINT #9,map$(i,j)
9390 NEXT j
9400 NEXT i
9410 FOR i= 1 TO 36
9420 FOR j=1 TO 3
9430 PRINT #9,object$(i,j)
9440 NEXT j
9450 NEXT i
9460 FOR i =1 TO 6
9470 PRINT #9,character$(i)
9480 NEXT i
9485 PRINT #9,char$
9490 CLOSE #9
9500 END DEFine file_out
```

Procfile_out commentary

This is a long procedure to type in, but all it does is to open a channel, at **lines 9100** and **9350**, and then PRINT all the data through this channel to

the cartridge. Each string and variable has to be read in turn — the program supplies the filename automatically.

Back at the main menu the user may choose option 1 which will load in data from a previous session with QLAG.

Procinfile

```
8000 DEFine PROCedure infile
8010 CLS
8020 PRINT "Load which game?"
8030 INPUT game$
8040 PRINT"Place the cartridge into M
icrodrive 1"
8050 PRINT "And press <Enter>"
8060 REPeat get_loop
8070 in$=INKEY$
8080 IF in$=CHR$(10) THEN EXIT get_lo
op
8090 END REPeat get_loop
8100 gn$="mdv1_"&game$&"numbers"
8110 gm$="mdv1_"&game$&"names"
8120 OPEN #9, gn$
8130 FOR i=1 TO 36
8140 FOR j=1 TO 6
8150 INPUT #9,object%(i,j)
8160 NEXT j
8170 NEXT i
8180 FOR i=1 TO 12
8190 FOR j=1 TO 24
8200 FOR k=1 TO 4
8210 INPUT #9,map%(i,j,k)
8220 NEXT k
8230 NEXT j
8240 NEXT i
8250 FOR i=1 TO 16
8260 FOR j=1 TO 6
8270 INPUT #9,monster%(i,j)
8280 NEXT j
8290 NEXT i
8300 FOR i=1 TO 6
8310 FOR j=1 TO 5
8320 INPUT #9,character%(i,j)
```

```
8330 NEXT j
8340 NEXT i
8342 INPUT#9,turns
8343 INPUT#9,monsterskilled
8344 INPUT#9,v
8345 INPUT #9,h
8350 CLOSE #9
8360 OPEN #9, gm$
8370 FOR i= 1 TO 12
8380 FOR j=1 TO 24
8390 INPUT #9,map$(i,j)
8400 NEXT j
8410 NEXT i
8420 FOR i= 1 TO 36
8430 FOR j=1 TO 3
8440 INPUT #9,object$(i,j)
8450 NEXT j
8460 NEXT i
8470 FOR i =1 TO 6
8480 INPUT #9,character$(i)
8490 NEXT i
8495 INPUT#9,char$
8500 CLOSE #9
8510 END DEFine infile
```

Procinfile commentary

A mirror image of **Procfile_out** in which, of course, the information travels the other way. Data is saved and loaded in two parts — 'numbers', or the variables holding numerical information, and 'names', the strings holding the various text.

Data statements

```
12430 DATA "sword",1,1,1,1,1,1
12440 DATA "torch",2,2,2,2,2,2
12450 DATA "club",3,3,3,3,3,3
12460 DATA "axe",2,2,2,2,2,2
12470 DATA "knife",2,5,5,5,5,5
12480 DATA "staff",3,3,3,3,3,3
12490 DATA "book",4,4,4,4,4,4
12500 DATA "coin",7,7,7,7,7,7
12510 DATA "orb",3,4,3,4,3,4
```

```
12520 DATA "flower",5,5,5,5,5,8
12530 DATA "wand",6,6,6,6,6,6
12535 DATA "gem",4,5,6,8,6,5
12540 DATA "ring",5,5,5,7,7,7
12550 DATA "cloak",7,7,7,7,7,7
12560 DATA "torque",8,8,8,8,8,8
12570 DATA "helmet",7,6,7,8,7,3
12580 DATA "scroll",9,9,9,9,9,9
12590 DATA "rug",7,6,7,6,5,6
12600 DATA "shield",7,6,5,4,3,2
12610 DATA "stone",8,7,6,5,4,3
12620 DATA "bone",4,5,6,7,8,9
12630 DATA "boots",9,8,7,6,5,4
12640 DATA "bow",6,5,4,7,6,5
12650 DATA "box",3,3,3,4,4,4
12660 DATA "sack",5,6,7,8,9,0
12670 DATA "hat",5,4,3,6,5,4
12680 DATA "cross",5,4,6,7,8,9
12690 DATA "goblet",4,5,6,7,8,9
12700 DATA "crown",5,4,3,6,5,4
12710 DATA "sceptre",2,4,6,8,9,7
12730 DATA "belt",4,5,6,8,6,5
12740 DATA "jar",4,3,6,5,1,6
12750 DATA "bracelet",7,4,6,3,5,2
12760 DATA "feather",7,5,3,6,4,2
12770 DATA "bottle",5,4,3,7,5,7
12780 DATA "potion",7,5,3,4,6,8
12790 DATA "snake","ghost","wild dog"
,"roper","gnome","dragon","spider","g
oblin","troll"
12800 DATA "wizard","hulk","zombie","
ostrich","vulture","demon","lionman"
12810 DATA "ATTK","DFNC","SPEL","ANGR
","NERV","GRED","CURS","LOCN","BANE"
12820 DATA "GAEYLOR","ARAGAUNT","GRUM
P","STUGGLE","ENDAR","SHELL"
12830 DATA "AT","DF","SP","CS","LC","
BN"
```

Data statements

Now finally we come to the data to be used in the program. Compared with QAD, the next program, there are not very many lines. The names in **line 12820** are not important, in that they can be changed at will —

whatever you type in can be changed during the course of the program when run. Other data information can only be changed when you type in the data lines. The numbers in **lines 12430–12780** appear in the object spreadsheet.

CHAPTER 7
The QL Adventure — QAD

Having hacked in all the lines for the generating program we save it to microdrive. We can now type in the program to run all the data generated by QLAG — this will be our adventure program, which we will call QAD (QL ADventure).

At this point, a decision must be made — do you want a graphic adventure or a text-only adventure? If graphic (and the graphic adventure will certainly be very rewarding), then include all the graphic procedures, **Procmonster_show**, **Procobj_show**, **Procdoor_draw** and **Procgraphic**. You will then have the daunting task of typing in all those data lines at the end of the program, although to start with you will find that you needn't type them all in — just a couple will suffice to get a flavour of the game. And in the next chapter we will give you a little program to define your own graphics, so that the burden may be eased somewhat. If you decide on a text-only adventure, you will have to ensure that you don't call the graphics procedures from the other procedures.

Let's get on now with QAD, the program which will run using the data set up by us in the first program, QLAG.

Setup and main loop

```
100 REMark QL ADVENTURE  1984
 N. RICHARD WILLIAMS & TONY BRIDGE
110 CLEAR
120 RESTORE
130 init
140 infile
150 char_select
160 cell_display
170 char_check char$
180 REPeat main_loop
190 CLS#14
200 PRINT #14,"B=battle, C=chat, D=dr
ink, G=give, I=investigate, L=leave,
M=move, Q=quit, R=rest, S=score, T=ta
ke"
```

```
210 REPeat getloop
220 keypress$=INKEY$
230 keypress=CODE(keypress$)
240 IF keypress<123 AND keypress>64 T
HEN EXIT getloop
250 END REPeat getloop
260 turns=turns+1
270 qr=RND(1 TO (LEN(char$))):s=RND(1
 TO 2)
280 character%(char$(qr),s)=character
%(char$(qr),s)-1
290 SELect ON keypress
300 ON keypress=77,109
310 char_move
320 ON keypress=76,108
330 drop char$(RND(1 TO 3) )
340 ON keypress=73,105
350 cell_display:char_check char$
360 ON keypress=66,98
370 combat
380 ON keypress=84,116
390 take
400 ON keypress=68,100
410 drink
420 ON keypress=83,115
430 score
440 ON keypress=82,114
450 rest
460 ON keypress=67,99
470 talk
480 ON keypress=71,103
490 give
500 ON keypress=81,113
510 EXIT main_loop
520 END SELect
530 FOR k=1 TO LEN(char$)
535 FOR m=6 TO 8
540 IF object%(character%(char$(k),m)
,5) = 2 THEN victory=1
545 PRINT object%(character%(char$(k)
,m),5)
550 NEXT m
555 NEXT k
560 IF LEN(char$)=0 OR victory=1 THEN
```

```
 EXIT main_loop
570 END REPeat main_loop
580 IF victory=1 THEN celebrate:STOP
590 IF LEN(char$)=0 THEN cry:STOP
600 PRINT #14,"Do you want to save da
ta? (Y/N)"
610 REPeat loop
620 g$=INKEY$(0)
630 IF g$="Y" OR g$="N" THEN EXIT loo
p
640 END REPeat loop
650 IF g$="Y" THEN file_out
660 PRINT #14,"Farewell"
670 STOP
```

Setup and main loop commentary

This first part of the program controls the whole proceedings, calling several procedures at the start. **Procinit** and **Procinfile** will be familiar from QLAG.

The structure of the main loop (**lines 180–510**) is pretty easy to follow. In window #14 a menu is printed, giving the player several options and calling one of several procedures, depending on the choice (note that both upper and lower case input is catered for here, with the code of each being checked for).

Battle	– calls **Proccombat**
Chat	– calls **Proctalk**
Drink	– calls **Procdrink**
Give	– calls **Procgive**
Investigate	– calls **Proccell_display**
Leave	– calls **Procdrop**
Move	– calls **Procchar_move**
Quit	– exits the main loop
Rest	– calls **Procrest**
Score	– calls **Procscore**
Take	– calls **Proctake**

On leaving the main loop, the program is really finished, although there is a bit of housekeeping to be done. The player is asked if he wishes to save data on this game — if yes, then **Procfile_out** is called, and everything is saved.

We'll tackle the procedures in the order in which the program calls them.

Procinit

```
6650 DEFine PROCedure init
6660 RESTORE
6670 base=138898
6680 OPEN #5,scr_153x88a32x16
6690 OPEN #6,scr_153x88a185x16
6700 OPEN #7,scr_153x88a337x16
6710 OPEN #8,scr_180x88a32x16
6720 OPEN #9,scr_280x52a212x16
6730 OPEN #10,scr_280x36a212x68
6740 OPEN #11,scr_153x110a32x104
6750 OPEN #12,scr_153x110a185x104
6760 OPEN #13,scr_153x110a337x104
6770 OPEN #14,con_460x42a32x214
6780 FOR i=1 TO 7
6790 INK #(i+7),0
6800 PAPER#(i+7),i-1
6810 CLS #(i+7)
6820 BORDER #(i+7),1,0
6830 NEXT i
6840 INK#8,7
6850 INK#7,0
6860 INK#6,0
6870 INK #5,0
6880 PAPER #5,1
6890 PAPER #6,2
6900 AT #8,3,5:PRINT #8,"ROOM"
6910 PAPER #7,3
6920 PRINT #10,"CHARACTER MESSAGES"
6930 FOR i=1 TO 3
6940 PRINT #(i+10),"CHARACTER"!" STAT
S"
6950 NEXT i
6960 PRINT #14,"YOUR INPUT"
6970 PRINT #9,"SYSTEM OUTPUT"
6980 h=1:v=1:beforeh=1:beforev=1
6990 dor$="NESW":rdor$="SWNE"
7000 entrance%=4:bargain=0
7010 victory=0
7020 char$="123"
7030 pl=0
7040 comb=0
7050 sc=0
```

```
7060 monsterskilled=0:no_of_events=4
7070 turns=0
7080 AT #9,3,2:PRINT #9,"Initialising
...."
7090 DIM map%(12,24,4)
7100 DIM map$(12,24,60)
7110 DIM object$(36,3,12)
7120 DIM object%(36,6)
7130 DIM monster$(16,8)
7140 DIM monster%(16,6)
7150 DIM character%(6,8)
7160 DIM character$(6,8)
7170 DIM category$(9,8)
7180 DIM tempmon%(3,6)
7190 DIM first$(6,20),second$(6,20),n
asty$(6,16)
7200 FOR i=1 TO 6
7210 READ first$(i),second$(i),nasty$
(i)
7220 NEXT i
7230 FOR i=1 TO 16
7240 READ monster$(i)
7250 NEXT i
7260 FOR i=1 TO 6
7270 READ category$(i)
7280 NEXT i
7290 END DEFine init
```

Procinit commentary

The base address at **line 6670** is the bottom left address of the screen memory, from which all graphics will be computed. Then 10 windows are defined (**lines 6680–6770**) one of which, window #14, will accept input typed in from the keyboard. The borders are drawn, and the various ink and paper colours set. Notice here that each window has a different colour (set in **line 6790**). In each of these windows is printed information on what will be displayed in that particular window as the game progresses. This gives the player something to look at while initialising is in progress.

Then the variables are set to their initial starting values and several strings are dimensioned. Note in **lines 6990–7000** that the two strings of entrance codes ('NESW') and exit codes ('SWNE') from our Exits section in Chapter 5 are placed in **dor$** and **rdor$** respectively, and that

entrance% is set to 4 (as this is not one of the values 0–3). Finally the data (lines 9500–11760) is read in to several strings.

Having set up the main screen display, the program now calls **Procinfile**.

Procinfile

```
7300 DEFine PROCedure infile
7310 CLS#14
7320 PRINT #14,"Load which game?"
7330 INPUT #14,game$
7340 PRINT#14,"Place the cartridge in
to Microdrive 1"
7350 PRINT#14, "And press <Enter>"
7360 REPeat get_loop
7370 in$=INKEY$
7380 IF in$=CHR$(10) THEN EXIT get_lo
op
7390 END REPeat get_loop
7400 gn$="mdv1_"&game$&"numbers"
7410 gm$="mdv1_"&game$&"names"
7420 OPEN #15, gn$
7430 FOR i=1 TO 36
7440 FOR j=1 TO 6
7450 INPUT #15,object%(i,j)
7460 NEXT j
7470 NEXT i
7480 FOR i=1 TO 12
7490 FOR j=1 TO 24
7500 FOR k=1 TO 4
7510 INPUT #15,map%(i,j,k)
7520 NEXT k
7530 NEXT j
7540 NEXT i
7550 FOR i=1 TO 16
7560 FOR j=1 TO 6
7570 INPUT #15,monster%(i,j)
7580 NEXT j
7590 NEXT i
7600 FOR i=1 TO 6
7610 FOR j=1 TO 5
7620 INPUT #15,character%(i,j)
7630 NEXT j
7640 NEXT i
```

```
7650 INPUT#15,turns
7660 INPUT#15,monsterskilled
7670 INPUT#15,v
7680 INPUT#15,h
7690 CLOSE #15
7700 OPEN #15, gm$
7710 FOR i= 1 TO 12
7720 FOR j=1 TO 24
7730 INPUT #15,map$(i,j)
7740 NEXT j
7750 NEXT i
7760 FOR i= 1 TO 36
7770 FOR j=1 TO 3
7780 INPUT #15,object$(i,j)
7790 NEXT j
7800 NEXT i
7810 FOR i =1 TO 6
7820 INPUT #15,character$(i)
7830 NEXT i
7840 INPUT#15,char$
7850 CLOSE #15
7860 END DEFine infile
```

Procinfile commentary

This procedure will always be called, as the game cannot be played without a set of previously-defined data, transported from QLAG — **lines 7320–7350** instruct you to load the data for your game. The procedure follows the same pattern as **Procinfile** from QLAG (QLAG lines 8000–8510) — in fact, we used the same procedure, changing one or two details, such as the window number (#15 here, rather than the #9 of the QLAG procedure).

The best way of achieving this is to SAVE what you already have of QAD, then NEW, then LOAD QLAG again. Now, just SAVE lines 8000–8510 of QLAG. Reload QAD, MERGE the procedure, and then change as necessary.

Procchar_select

```
680 DEFine PROCedure char_select
690 char$=""
700 CLS#5
710 CLS#6
```

```
720 CLS#7
730 BORDER#5,1,0
740 BORDER #6,1,0
750 BORDER #7,1,0
760 FOR i=1 TO 3
770 FOR j= 1 TO 6
780 PRINT #(i+4),category$(j);" ";cha
racter%(i,j)
790 PRINT #(i+10),category$(j);" ";ch
aracter%(i+3,j)
800 NEXT j
810 AT #(i+4),7,5:PRINT #(i+4),i
820 AT #(i+10),9,5:PRINT #(i+10),i+3
830 NEXT i
840 PRINT #14,"Choose three character
s by pressing     three of the number
s 1 to 6"
850 REPeat count
860 REPeat check
870 g$=INKEY$
880 IF g$ INSTR char$=0 AND g$ INSTR
"123456"<> 0 THEN EXIT check
890 END REPeat check
900 char$=char$ & g$
910 IF LEN(char$)=3 THEN EXIT count
920 END REPeat count
930 END DEFine char_select
```

Procchar_select commentary

Once the selected data is loaded, the characters are displayed in six windows (numbers 5, 6, 7, 11, 12 and 13 — **lines 780–790**), complete with the various parameters as set by the user in QLAG. Then the player is prompted in window #14 to choose three characters to set forth on the adventure (**line 840**).

Lines 850–920 check the validity of the input, and exit the procedure once a valid choice has been made (for more information on the INSTR operator, see Manual, Keywords).

Proccell_display

```
940 DEFine PROCedure cell_display
950 CLS#9:BORDER #9,1,0
```

```
960 CLS#10:BORDER #10,1,0
970 tell v,h
980 graphic v,h
990 IF map%(v,h,4)<>0 THEN monster_sh
ow
1000 IF map%(v,h,2)<>0 THEN obj_show
1010 END DEFine cell_display
```

Proccell_display commentary

Clears windows #9 and #10, defines the borders and then calls **Proctell** and **Procgraphic**. A check is made to see if there is a monster or object present in this cell (from information in **map%**). If there is something present, the respective procedures are called — **Procmonster_show** or **Procobj_show**. Proccell_display is also called by lines 340–350 in the main loop.

Procgraphic

```
1020 DEFine PROCedure graphic(v,h)
1030 CLS#8:BORDER#8,1,0
1040 SCALE #8,180,0,0
1050 REMark h gives horizontal and v
vertical map reference
1060 LINE #8,0,0 TO 60,60 TO 60,140 T
O 0,180,270,0 TO 200,60 TO 200,140 TO
 270,180,60,140 TO 200,140,60,60 TO 2
00,60
1070 REMark entrance%=exit entered by
 (n=0,W=3)
1080 REMark if there is a code for th
e directon then draw the door
1090 PRINT #10,"Exits :";
1100 FOR k=0 TO 3
1110 IF (map%(v,h,1) && 2^k) THEN PRI
NT #10," ";rdor$(k+1);:IF k<>entrance
% THEN door_draw k
1120 NEXT k
1130 PRINT #9," "
1140 END DEFine graphic
```

Procgraphic commentary

A graphic procedure. Using the two parameters **v** and **h**, this procedure draws in window #8 a little '3D' picture of the room. **Line 1110** checks

map% to see if there is a doorway in one of the three walls, using the bitwise AND function and a loop (k =0 TO 3). As long as the value of **k** is not the value of **entrance%**, then **Procdoor_draw** is called to draw the door, using the parameter **k**.

Procmonster_show

```
1310 DEFine PROCedure monster_show
1320 PAPER #8,0
1330 FOR mon=0 TO (map%(v,h,4)-1) MOD
 3
1340 REMark number of monsters
1350 RESTORE (9700+(map%(v,h,3)*300))
1360 READ vert_it
1370 READ hor_it
1380 startadd=(mon)*(4+hor_it)
1390 FOR upward=0 TO vert_it
1400 FOR along=0 TO hor_it   STEP 2
1410 READ dat
1420 POKE_W base+2560-(vert_it*128)+s
tartadd+along+(upward*128),dat
1430 END FOR along
1440 END FOR upward
1450 END FOR mon
1460 END DEFine monster_show
```

Procmonster_show commentary

A graphics procedure. The data for the graphic of each monster present is now read in, in the foreground of the room.

Procobj_show

```
1470 DEFine PROCedure obj_show
1480 REMark draw here obj graphic for
 map%(h,v,2)
1490 RESTORE (14660+(map%(v,h,2)*10))
1500 READ down,along
1510 FOR j=0 TO down
1520 FOR k=0 TO (along) STEP 2
1530 READ a
1540 POKE_W (base+4100+k+(j*128)),a
1550 NEXT k
```

```
1560 NEXT j
1570 END DEFine obj_show
```

Procobj_show commentary

A graphics procedure. And the data for the graphic of each object present is read into the same window.

Proctell

```
1580 DEFine PROCedure tell
1590 CLS#9
1600 CLS#10
1610 PRINT #9,map$(v,h)
1620 IF map%(v,h,4) <>0 THEN PRINT#9,
 "Ahead you see ";map%(v,h,4);" ";:pl
ural
1630 IF map%(v,h,2)<>0 THEN PRINT#9,
"There is a ";object$(map%(v,h,2));"
nearby."
1640 END DEFine tell
```

Proctell commentary

This procedure simply reads **map%**, **map$** and **object$** and prints the information for the player.

Procdoor_draw

```
1150 DEFine PROCedure door_draw (k)
1160 REMark move the position code
around according to the entrance dire
ction
1170 door_code=(k+4-entrance%) MOD 4
1180 SELect ON door_code
1190 REMark door_code 1= left wall, 2
 = mid wall, 3 = right wall
1200 ON door_code=1
1210 REMark draw left_door
1220 LINE#8,20,20 TO 20,130 TO 40,116
 TO 40,40
1230 ON door_code=2
```

```
1240 REMark draw mid_door
1250 LINE #8,110,60 TO 110,115 TO 150
,115 TO 150,60
1260 ON door_code=3
1270 REMark draw right_door
1280 LINE#8,250,20 TO 250,130 TO 230,
116 TO 230,35
1290 END SELect
1300 END DEFine door_draw
```

Procdoor_draw commentary

A graphic procedure. The parameter **k** is the result from the **k** loop in **Procgraphic**, lines 1100–1120. Using this, **line 1170** assigns a code to the variable **door_code**, and this code is used to decide which wall to draw the door graphic on, should one be called for (see the Exits discussion in Chapter 5).

Now we return to the very start of the program, where **Procchar_check** is called (line 350).

Procchar_check

```
1650 DEFine PROCedure char_check(r$)
1660 FOR i=1 TO LEN(r$)
1670 REMark print (window of current
character)
1680 CLS#(i+10)
1690 FOR j=1 TO 5
1700 IF character%(r$(i),j)<1 THEN ch
aracter%(r$(i),j)=0
1710 PRINT #(i+10), category$(j);" ";
character%(r$(i),j)
1720 NEXT j
1730 PRINT #(i+10);"Carrying:"
1740 j=6
1750 jt=0
1760 REPeat loop
1770 IF character%(r$(i),j)<>0 THEN
1780 PRINT#(i+10)," ";object$(charact
er%(r$(i),j),2)
1790 FOR k=3 TO 5
1800 IF object%(character%(r$(i),j),5
```

```
)<>1 AND object%(character%(r$(i),j),
5)<>2 THEN character%(r$(i),k)=charac
ter%(r$(i),k)+object%(character%(r$(i
),j),k):object%(character%(r$(i),j),k
)=0
1810 NEXT k
1820 ELSE
1830 jt=jt+1
1840 END IF
1850 j=j+1
1860 IF j=8 THEN EXIT loop
1870 END REPeat loop
1880 IF jt=2 THEN PRINT#(i+10);" Noth
ing"
1890 AT #(i+10),9,(11-(LEN(character$
(r$(i)))))DIV 2:PRINT #(i+10),charact
er$(r$(i))
1900 NEXT i
1910 IF turns>3 AND comb <>1 THEN my_
word
1920 END DEFine char_check
```

Procchar_check commentary

This procedure prints, in three windows below the playing area, the stats of each of the three characters selected earlier in **Procchar_select**, along with any objects carried, **lines 1660–1810** —at first, of course, no objects will be carried. Finally, after a certain number of turns (three here, but this can be altered at will), **Procmy_word** is called (see later).

So now the initial setup has been carried out, and the player is presented with the main display area, showing his party of three characters, a window showing the current cell with exits, monsters and objects. Alongside is the information window and, beneath that, the text input window. Now we return to the main program loop, from which most of the work will be carried out.

Procchar_move

```
1930 DEFine PROCedure char_move
1940 ex%=(entrance%+2)MOD 4
1950 CLS#14
1960 PRINT #14,"Which way?"
1970 REPeat which
```

```
1980 REPeat way
1990 g$=INKEY$ (#14)
2000 g$=change$(g$)
2010 entrance%= (g$ INSTR dor$)-1
2020 IF entrance% <>-1 AND g$<>"" THE
N EXIT way
2030 END REPeat way
2040 it= (2^((((g$ INSTR "SsWwNnEe")+
1)DIV 2)-1))
2050 IF (map%(v,h,1)&&it) THEN
2060 EXIT which
2070 ELSE
2080 PRINT #14,"No door that way."
2090 END IF
2100 END REPeat which
2110 IF map%(v,h,4)<>0 AND ex%<>entra
nce% THEN PRINT #9,"You can only go "
;dor$(ex%+1):entrance%=(ex%+2)MOD 4:R
ETurn
2120 beforeh=h:beforev=v
2130 g=CODE(g$)
2140 SELect ON g
2150 ON g=78,110
2160 v=v+1
2170 ON g=83,115
2180 v=v-1
2190 ON g=69,101
2200 h=h+1
2210 ON g=87,119
2220 h=h-1
2230 END SELect
2240 IF h<1 THEN h=1
2250 IF h>12 THEN h=12
2260 IF v<1 THEN v=1
2270 IF v>24 THEN v=24
2280 cell_display
2290 END DEFine char_move
```

Procchar_move commentary

Called by lines 300–310 in the main loop. This procedure is called if the Move option is selected.

Line 1940: **ex%** is set to a code for the door opposite to the current entrance — if the player enters through the East door, **ex%** is set to the value of West. The player then types in the desired exit, which is held in

entrance% (line 2010). Line 2110: However, if there is a monster in the room, the player can only leave by the door entered by (ie can only retreat) and **entrance%** is reset to the value it had on entry (its value in **line 1940**).

Lines 1980–2030: The 'way' loop simply repeats until a valid input is made — N, S, E or W in upper or lower case characters. **Lines 1970–2100:** 'Way' is part of a larger loop, 'which' — this checks the input from 'way' and compares it to **map%**. If indeed there is an exit in the requested direction, then 'which' is left, and the program goes on to:

Lines 2110–2230, which take the code of the input character (N, S, E or W) and update the coordinates variables, **v** (vertical) and **h** (horizontal), as necessary.

Lines 2240–2270: Ensure that **v** and **h** are kept within the limits of 1 and 20. Then **Proccell_display** is called again in **line 2280** for an update of the graphic display.

Change$

```
6480 DEFine FuNction change$(g$)
6490 LOCal f$
6500 IF CODE(g$)>90 THEN
6510 f$=CHR$(CODE(g$)-32)
6520 ELSE
6530 f$=g$
6540 END IF
6550 RETurn f$
6560 END DEFine change$
```

Change$ commentary

This function is called several times in QAD, the first being the occurrence in **Procchar_move**. We use a function in order to return a value to the procedure from which it is called — other procedures using **change$** are **Procdrink** (line 2410) and **Procwho** (line 3490).

The routine reads the keyboard, and *changes* the lower case response to an upper case response, by changing the code of the keypress. Lower case keys are codes 97–122, and upper case are codes 65–90. For more details of key codes, see Manual, Concepts.

Procdrop

```
8780 DEFine PROCedure drop(dc)
8785 LOCal dp
8790 dp=9
8800 REPeat obloop
8810 dp=dp-1
```

```
8820 IF dp=6 OR character%(dc,dp)<>0
 THEN EXIT obloop
8830 END REPeat obloop
8840 IF character%(dc,dp)=0 THEN RETu
rn
8850 IF map%(v,h,2)<>0 THEN PRINT#9,"
The ";object$(map%(v,h,2));" disappea
rs"
8860 map%(v,h,2)=character%(dc,k):PRI
NT #9,map%(v,h,2)
8870 character%(dc,dp)=0
8880 char_check char$:cell_display
8900 END DEFine drop
```

Procdrop commentary

Called by lines 320–330 in the main loop. This routine allows the player to leave an object, should they wish to do so. A random number is generated in line 330 in the main loop, and this number is passed along to the procedure, which removes the **object$** from **map%** and **character%**.

Proccombat

```
3550 DEFine PROCedure combat
3560 CLS#14
3570 IF map%(v,h,4)<1 THEN PRINT #9,"
No opponents":RETurn
3575 FOR i=1 TO 3:tempmon%(i,1)=0:tem
pmon%(i,2)=0
3580 FOR i=1 TO map%(v,h,4)
3590 FOR j=1 TO 2
3600 tempmon%(i,j)=RND(monster%(map%(
v,h,3),j))+(monster%(map%(v,h,3),j)^2
)
3610 NEXT j
3612 FOR j=3 TO 6
3614 tempmon%(i,j)=monster%(map%(v,h,
3),j)
3616 NEXT j
3620 NEXT i
3630 END FOR i
3640 tchar$=char$ :REMark Create temp
orary character string
3650 REMark Check for bane
3660 banei=0:banej=0
3670 FOR i=1 TO LEN(char$)
```

```
3680 FOR j=6 TO 8
3690 IF object%(character%(char$(i),j
),6)=map%(v,h,3) THEN banei=i:banej=j
3700 NEXT j
3710 NEXT i
3720 IF banei<>0 THEN PRINT #9,"The "
;object$(object%(char%(char$(banei),b
anej),6));" scares them away":map%(v,
h,3)=0:map%(v,h,4)=0:cell_display:RET
urn
3730 REMark If monsters are greedy an
d bargain flag is not set then monste
rs accept gift
3740 greed=0
3750 FOR i=1 TO map%(v,h,4):greed= te
mpmon%(i,5)+greed:NEXT i
3760 IF (greed DIV map%(v,h,4))> RND(
1 TO 9) AND bargain=0 THEN CLS#9:PRIN
T#9,"For a gift you can go past":barg
ain=1:PAUSE 100:RETurn
3770 bargain=0 :REMark Reset bargain
flag
3780 REMark If monsters are very stro
ng or outnumber the party then test e
ach character.
3790 FOR i=1 TO LEN(tchar$)
3800 IF (map%(v,h,4) > LEN(tchar$)) O
R (tempmon%(1,1)>(character%(tchar$(i
),1)) AND RND(1 TO character%(tchar$(
i),5)))>3 THEN
3810 PRINT #9,character$(tchar$(i));"
 runs and hides":CLS #(10+i)
3811 z$=""
3812 FOR k=1 TO LEN(tchar$)
3814 IF tchar$(k)<>ch THEN z$=z$ & tc
har$(k)
3816 END FOR k
3818 tchar$=z$
3820 END IF
3830 NEXT i
3840 REMark If all characters run the
n put them in the previous room and r
eturn
3850 IF LEN(tchar$)=0 THEN v=beforev:
```

```
h=beforeh:RETurn
3860 REMark If either characters or m
onsters are angry then call attack ot
herwise just chat.
3870 angst=0
3880 FOR i= 1 TO LEN(tchar$)
3890 angst = angst + character%(tchar
$(i),4)
3900 NEXT i
3910 angst=angst DIV LEN(tchar$) + RN
D(2)
3920 IF angst>3 THEN
3930 attack
3940 ELSE
3950 PRINT #9,character$(tchar$(1));"
 says: Do we really want to fight?":P
AUSE 100
3960 FOR i=1 TO map%(v,h,4):angmn=ang
mn+tempmon%(i,4):NEXT i
3970 IF angmn>3 THEN
3980 attack
3990 ELSE
4000 PRINT#9,"The ";plural;"replies:
Seems silly to me":PAUSE 100
4010 END IF
4020 END IF
4030 RETurn
4040 END DEFine combat
```

Proccombat commentary

Called by lines 360–370 in the main loop. This was the first part of the program to be designed, and everything else really grew from this kernel. It is a fairly lengthy routine, and one that calls many more procedures.

Lines 3560–3570: Clear the window, #14, and then check in **map%** to ensure that there is a monster here — after all, the player would look pretty silly trying to fight thin air! If no opponent, then we RETURN to the main menu.

Line 3575–3630: These lines clear three windows along the top of the screen. The lower three, with our character names and stats displayed, remain throughout this routine. The three new windows will contain information on the monsters — there could be up to three of the wee beasties. Then random values are given to each of the five categories in the stat list (Attack, Defence and so on), and this information is printed.

Lines 3640–3720: Now the program checks the presence of an object which is Bane to a monster. If it scares the monster or monsters away, then the program returns to the main menu.

Lines 3730–3770: The monster's Greed factor is initially set at 0, but **line 3760** compares the variable **greed** against a random number, and allows the player past if he has been lucky. In this case, **bargain** is set to 1, where it remains until this procedure is called again. If he has not been so lucky, **bargain** is set to 0, and the procedure carries on in:

Lines 3790–3810: Test the relative strength of the monsters and the player's party. If that of the monster or monsters is higher than that of the exploring party, they all become demoralised and run from the room, to return to the previous room. Should, however, either the monsters or the party be angry, then **Procattack** is called (**line 3930**) if **angst** is low — otherwise, the monster calls the whole thing off, and the main menu is returned to.

Procattack

```
4090 DEFine PROCedure attack
4100 comb=1:dead=0:retr=0
4110 REMark Whilst a monster exists a
nd a character is in the room
4120 FOR i=1 TO 3:CLS#(i+4):BORDER#(i
+4),1,0:NEXT i
4130 FOR i=1 TO map%(v,h,4):monup(i):
NEXT i
4140 REPeat while
4150 REPeat choose
4160 mon=RND(1 TO 3)
4170 IF tempmon%(mon,2)<>0 AND tempmo
n%(mon,1) <>0 THEN EXIT choose
4180 END REPeat choose
4190 REPeat chloop
4200 PRINT #14,"Who will attack ";mon
ster$(map%(v,h,3));" ";mon;"?"
4210 who
4220 IF ch INSTR tchar$ <>0 THEN EXIT
 chloop
4230 PRINT#14, "Not here":PAUSE 50
4240 END REPeat chloop
4250 charatt= character%(ch,1)
4260 chardef=character%(ch,2)
4270 charang=character%(ch,4)
```

```
4280 FOR j=6 TO 8
4290 IF character%(ch,j)<>0 THEN char
att = charatt + object%(character%(ch
,j),1):chardef=chardef+object%(charac
ter%(ch,j),2)
4300 NEXT j
4310 monatt= tempmon%(mon,1)
4320 mondef=tempmon%(mon,2)
4330 PRINT#14, "Will you :cast a spel
l (S), attack furiously (F), attack c
autiously (C)or retreat (R)?"
4340 REMark Get tactic
4350 tac$=""
4360 REPeat loop
4370 tac$=INKEY$:IF tac$="" THEN GO T
O 4370
4380 IF tac$ INSTR "FfSsCcRr" <>0 THE
N EXIT loop
4390 END REPeat loop
4400 tac=CODE(tac$):hit=0
4410 SELect ON tac
4412 =82,114
4414 retr=1
4416 EXIT while
4420 =83,115
4430 IF character%(ch,3)>0
4440 PRINT #14,character$(ch);" ";
4450 spell character%(ch,3)
4460 character%(ch,3)=character%(ch,3
)-1
4465 char_check tchar$
4470 ELSE
4480 PRINT#14,"No spells left":PAUSE
100
4490 END IF
4500 =70,102 :frenzy
4510 =67,99 :caution
4520 END SELect
4530 REMark Check for cursed object
4540 chekcurs
4550 IF tac<>83 AND tac<>115 THEN str
ike charatt,mondef
4560 IF kflag=0 AND retr<>1 THEN mons
terhurt(hit)
```

```
4565 ELSE PRINT #14,character$(ch);"
is cursed. He hurts himself":PAUSE 10
0
4566 character_hurt hit-1
4567 END IF
4570 REMark Monster chooses tactic
4580 tic=RND(1 TO 3)
4590 hit=0
4600 SELect ON tic
4610 =1
4620 IF tempmon%(mon,3)>0 THEN
4630 PRINT #14,"The ";monster$(map%(v
,h,3));" ";
4640 spell(tempmon%(mon,3))
4650 tempmon%(mon,3)=tempmon%(mon,3)-
1
4660 ELSE
4670 PRINT #14,"The monster wants to
cast a spell but has run out":PAUSE 1
00
4680 END IF
4690 =2
4700 monatt=monatt+(tempmon%(mon,4) D
IV 2)
4710 mondef=mondef-(tempmon%(mon,4) D
IV 2)
4720 PRINT #14,"The ";monster$(map%(v
,h,3));" is furious":PAUSE 50
4730 =3
4740 monatt=monatt-(tempmon%(mon,6) D
IV 2)
4750 mondef=mondef+(tempmon%(mon,6) D
IV 2)
4760 PRINT #14,"The ";monster$(map%(v
,h,3));" is cautious":PAUSE 50
4770 END SELect
4780 REMark If NOT SPELL then  CALL S
TRIKE for monster
4790 IF tic<>1 THEN strike monatt,cha
rdef
4800 IF hit <>0 THEN characterhurt(hi
t)
4810 REMark If monster is dead then r
emove him from inventory and array
```

```
4820 IF tempmon%(mon,2)<1 OR tempmon%
(mon,1)<1 THEN map%(v,h,4)=map%(v,h,4
)-1:CLS#(mon+4):monkil=monkil+1
4830 REMark If character is dead then
 remove him from both temporary and p
ermanent strings
4840 IF character%(ch,1)=0 AND charac
ter%(ch,2)=0 THEN
4850 z$="":x$=""
4860 FOR k= 1 TO LEN(tchar$)
4870 IF tchar$(k)<>ch THEN z$=z$ & tc
har$(k)
4882 NEXT k
4884 tchar$=z$:char_check tchar$:dead
=1:CLS#(11+LEN(char$))
4890 FOR k=1 TO LEN(char$)
4900 IF char$(k)<>ch THEN x$=x$ & cha
r$(k)
4912 NEXT k
4914 char$=x$
4920 END IF
4930 IF dead=1 OR map%(v,h,4)<1 THEN
EXIT while
4940 END REPeat while
4950 IF map%(v,h,4)=0 THEN
4955 PRINT#14, "All monsters dead":PA
USE 100
4960 ELSE
4962 IF dead=1 THEN
4964 PRINT #14,"A character has died,
 so you run away"
4966 h=beforeh:v=beforev
4967 CLS#(11 + LEN(char$))
4968 ELSE
4970 PRINT #14,"You try to retreat"
4971 IF h<>1 OR v<>1 THEN
4972 h=beforeh:v=beforev
4973 ELSE PRINT #14,"You are trapped"
4974 END IF
4976 END IF
4977 END IF
4978 comb=0:PAUSE 100
4979 cell_displa
4980 END DEFine attack
```

Procattack commentary

Along with **Proccombat**, this procedure is the kernel of the present program and is pretty long. However, most of it is straightforward and won't need very much explanation. It is certainly an improvement on the usual 'bash-bash', featuring as it does a certain amount of intelligence (the dreaded 'artificial intelligence') — this means that although the results of combat are really random this randomness can be influenced by several factors. After the initial setting of the variables **Comb**, **Dead** and **Retr** (**line 4100**), the monster(s) currently present in the cell or room are displayed. There could be up to three, so three windows are set up (**line 4120**), and the stats for each monster displayed. **Procmonup** is called for this purpose. After asking which character is to fight the monster(s), the program calls **Procwho** (which does guess what!).

Lines 4220–4230: Trap any attempt to nominate a member of the party who is not present, and:

Lines 4250–4290: Assess the character's Anger, Defence and Attack capabilities as held in **character%**. After printing up the battle menu, the program waits for the tactical input from the player. As you can see, there are several options open to the player. If they decide to Retreat, then the loop 'while' is left, which forces the program to jump to **line 4950**. The consequences of this action are not always advantageous to the player!

Should they decide, however, to stay and fight, they may use a Spell (by pressing S), or attack, either cautiously or furiously:

Lines 4430–4465: These lines call **Procspell** and update a character's window after using a spell, while **line 4480** traps an attempt to use a Spell when the character has none, and causes the character to choose again.

Lines 4500–4510: Call **Procfrenzy** and **Proccaution**, which speak for themselves. **Line 4540:** Calls **Procchekcurs**, which checks for a cursed object in the character's possession, which could prove to be an embarrassment to him.

Lines 4550–4560: Call **Procstrike** and **Procmonsterhurt**, which we will come to in a moment. Then it is the monster's turn to deal out the blows. A random number is chosen (**line 4580**) and assigned to the variable **tic** (short for tactic), and then one of several options is chosen by the program, depending on this number.

Lines 4620–4770: The monster can do one of three things, just like the character before him. He can choose to cast a spell, in which case **Procspell** is called again, or decide to attack Furiously or Cautiously.

Lines 4790–4968: The program now checks to see who has perished after the round of combat. If no-one (or no-thing) is yet dead, the program carries on with **Procstrike**, updating the windows of both characters and monsters. If all monsters in the room are now dead after combat, then we go back to the main menu — but if a character has died, then the party must retreat (**line 4964**) and the position parameters are

reset to the previous position. The party may find, of course, that it is trapped (**line 4973**) and the room is again displayed.

Procmonup

```
9080 DEFine PROCedure monup(mon)
9090 CLS#(4+mon)
9100 FOR j=1 TO 5
9110 PRINT #(mon+4),category$(j);" ";
9120 PRINT #(4+mon),tempmon%(mon,j)
9130 END FOR j
9140 AT #(mon+4),7,0:PRINT #(mon+4),m
onster$(map%(v,h,3));" ";mon
9150 END DEFine monup
```

Procmonup commentary

As we've already seen, this procedure clears a window or windows in which to display the monster's stats.

Procwho

```
3430 DEFine PROCedure who
3440 REPeat inloop
3450 REPeat gloop
3460 g$=INKEY$
3470 IF CODE(g$)>32 THEN EXIT gloop
3480 END REPeat gloop
3490 g$=change$(g$)
3500 FOR j=1 TO LEN(char$)
3510 IF g$=character$(char$(j),1) THE
N ch=char$(j):EXIT inloop
3520 END FOR j
3530 END REPeat inloop
3540 END DEFine who
```

Procwho commentary

Called from various procedures. Takes the input to the question 'who?' and assigns the value found in **character$** to the variable **ch**. Look at line 4220 in **Procattack** for an example of this at work — **ch** in this line now holds a value which the procedure can work with.

Procspell

```
4990 DEFine PROCedure spell(spels)
5000 PRINT #14,"tries to cast a spell
":PAUSE 100
5010 IF spels=0 THEN PRINT #14,"But h
as no magic":PAUSE 100:RETurn
5020 hit=RND(9)
5030 IF hit=1 THEN PRINT #14,"a miss
":PAUSE 100:RETurn
5040   hit = RND(1 TO 6) MOD 4
5050 PRINT #14,"A hit but only of pow
er ";hit:PAUSE 100
5060 END DEFine spell
```

Procspell commentary

Taking the value passed along to the parameter **spels** from line 4450 in **Procattack**, which reads the third element of **character%**. This is the element holding information on the Spells each character holds. If the value is 0, of course, then no magic is available to the character. Otherwise a random value is given to **hit**, and the outcome of the cast printed.

Procstrike

```
5400 DEFine PROCedure strike(ak,dc)
5410 IF ak>0 THEN
5420 av=RND(1 TO ak)
5430 ELSE
5440 av=0
5450 END IF
5460 IF dc>0 THEN
5470 dv=RND(1 TO dc)
5480 ELSE
5490 dv=0
5500 END IF
5510 IF av>dv THEN
5520 PRINT #14,"A hit":PAUSE 100
5530 hit =1+((RND(av-dv))DIV 2)
5540 RETurn
5550 ELSE
```

```
5560 PRINT#14, "The blow misses":PAUS
E 100
5570 END IF
5580 END DEFine strike
```

Procstrike commentary

This little routine is the heart of the combat routine, and decides whether a blow is a Hit or a Miss. This depends a little on luck, but largely on the attack and defence capabilities of the monsters and characters, as passed along to the procedure from **Procattack**, line 4790.

Procfrenzy

```
5070 DEFine PROCedure frenzy
5080 charatt=charatt + (character%(ch
,4) DIV 2)
5090 chardef=chardef-(character%(ch,4
) DIV 3)
5100 END DEFine frenzy
```

Proccaution

```
5110 DEFine PROCedure caution
5120 charatt=charatt-(character%(ch,5
) DIV 3)
5130 chardef=chardef+(character%(ch,5
) DIV 2)
5140 END DEFine caution
```

Procfrenzy and *Proccaution commentaries*

These are essentially two sides of the same coin. Back in **Procattack** (lines 4250–4260), values are given to the two variables **charatt** and **chardef**, and these values are used in the two procedures to work out the result of the character's Furious or Cautious attack.

Procchekcurs

```
5200 DEFine PROCedure chekcurs
5210 kflag=0
5220 FOR kur=6 TO 8
```

```
5230 IF character%(ch,kur)<>0 THEN IF
 object%(character%(ch,kur),4)=1 THEN
 charatt=charatt-5
5240 k1= object%(character%(ch,kur),4
)
5250 SELect ON k1
5260 ON k1=1
5270 charatt=charatt-5
5280 ON k1=2
5290 kflag=1
5300 END SELect
5310 END IF
5320 NEXT kur
5330 END DEFine chekcurs
```

Procchekcurs commentary

This procedure is called at line 4540 of **Procattack** if an object carried by the attacking character is cursed, and the result, in the values of the flag **kflag** and the variable **charatt**, is passed back to **Procattack**.

Procmonsterhurt

```
5150 DEFine PROCedure monsterhurt(val
)
5160 r=RND(1 TO 2)
5170 tempmon%(mon,r)=tempmon%(mon,r)-
hit
5180 monup mon
5190 END DEFine monsterhurt
```

Procmonsterhurt commentary

Called at various times to update the stats of a monster after a hit has been registered.

Proccharacterhurt

```
5340 DEFine PROCedure characterhurt(v
al)
5350 r=RND(1 TO 3)
5370 character%(ch,r)=character%(ch,r
)-hit
```

```
5380 char_check tchar$
5390 END DEFine characterhurt
```

Proccharacterhurt

Similar to **Procmonsterhurt**, but updates the stats of a character.

Proctake

```
5840 DEFine PROCedure take
5850 CLS#9
5860 IF map%(v,h,2)=0 THEN PRINT#9,"N
othing to take":RETurn
5870 IF map%(v,h,4)<>0 THEN PRINT #9,
"First you'll have to get around the
";:plural:RETurn
5880 PRINT #9,"Who wants to take it?"
5890 who
5900 FOR j=8 TO 6 STEP -1
5910 IF character%(ch,j)=0 THEN takep
=ch:takeh=j
5920 END FOR j
5930 IF takep=0 THEN PRINT#9,characte
r$(takep);" cannot carry any more":RE
Turn
5940 character%(takep,takeh)=map%(v,h
,2)
5950 map%(v,h,2)=0
5960 cell_display
5970 char_check char$
5980 END DEFine take
```

Proctake commentary

Called by lines 380–390 in the main loop. Should the player decide to take an object at the location, this procedure is called. First of all a check is made to ensure that there is, indeed, something to take here. If not, then the program returns to the main menu. But if an object *is* present, then **line 5870** checks for the presence of a monster — if there's one here, then only combat will enable the player to take possession of the object, and we return to the main menu where we came from. Note, at the end of this line, the calling of **Procplural**, which we will look at later.

Lines 5880–5980: These lines run through the three character's stats,

with particular reference to the objects carried at the time, and distributes the taken object or objects among the characters, assuming that one or more of the characters is able to carry more objects (if not, again we return to the main menu).

Procdrink

```
2300 DEFine PROCedure drink
2310 CLS#9
2315 CLS#10
2320 sflag=0
2330 FOR i= 1 TO LEN(char$)
2340 char_search char$(i),35
2350 char_search char$(i),36
2360 NEXT i
2370 IF pl<>1 AND sflag=0 THEN PRINT
#9,"There's no drink here":RETurn
2380 PRINT#9, "Who wants a drink?"
2390 REPeat inloop
2400 person$=INKEY$
2410 person$=change$(person$)
2430 FOR k=1 TO LEN(char$)
2440 IF person$=character$(char$(k),1
) THEN drinker= char$(k):EXIT inloop
2450 NEXT k
2460 END REPeat inloop
2470 IF pl=1 THEN
2480 pl=0
2490 rp=RND(1 TO 2):sp=RND(1 TO 5)
2500 PRINT#9,character$(drinker);"'s
";category$(sp);" is ";
2510 IF rp=1 THEN
2520 PRINT #9,"increased"
2530 character%(drinker,sp)=character
%(drinker,sp)+1
2540 ELSE
2550 PRINT #9,"decreased"
2560 character%(drinker,sp)=character
%(drinker,sp)-1
2570 END IF
2575 char_check char$
2580 RETurn
2590 ELSE
2600 sflag=0
```

```
2610 char_search drinker,35
2620 char_search drinker,36
2630 IF sflag<>0 THEN
2640 FOR i=1 TO 5
2650 character%(drinker,i)=character%
(drinker,i)+object%(sflag,i)
2660 END FOR i
2665 PRINT #9,"That feels better"
2680 character%(drinker,sflag)=0
2690 char_check char$
2700 RETurn
2710 ELSE
2720 FOR i= 1 TO 3
2730 sflag=0
2740 char_search i,35
2750 char_search i,36
2760 IF sflag<>0 THEN
2762 owner=char$(i)
2763 obn=sflag
2765 END IF
2770 END FOR i
2780 rp=RND(10-(character%(owner,5) M
OD 10))
2790 IF rp<2 THEN
2800 PRINT #9,character$(owner);" say
s 'You will have to give me something
 first"
2810 IF character%(drinker,6)=0 THEN
PRINT #9,"but you have nothing I want
":RETurn
2820 ELSE
2830 obmove=character%(owner,obn)
2840 character%(owner,obn)=character%
(drinker,6)
2850 character%(drinker,6)=obmove
2920 PRINT #10,"That'll do nicely"
2930 FOR i=1 TO 5
2940 character%(drinker,i)=character%
(drinker,i)+object%(character%(drinke
r,6),i)
2950 NEXT i
2960 character%(drinker,6)=0
2970 END IF
2980 END IF
```

```
2990 char_check char$
3000 CLS#9
3010 END DEFine drink
```

Procdrink commentary

Called by lines 400–410 in the main loop. This procedure allows drinking from a bottle (carried), a potion (also carried by one of the characters in the party), or from a Magic Pool. A flag, **sflag**, is first of all set to zero and this value is passed along to **Procchar_search**, which looks at each character to see if they are at present carrying either a bottle which can then be filled at the pool, or a potion which can be drunk. If the flag is set to zero by this procedure, then this means that the character has no potion or bottle, and therefore can't drink.

In **Procpool**, pl is set to 1 as long as a Magic Pool is present, and **line 2370** ensures that there is, indeed, a pool here. If not, control is returned to the main menu.

Lines 2380–2460: Take the player's choice of drinker and updates character$ accordingly, so that:

Lines 2470–2580: First of all set the Magic Pool variable **pl** to zero, and then go on to increase or decrease a randomly-chosen attribute of the character. **Lines 2590–2710:** Allow a character to drink from his own bottle, updating **character%** as necessary.

Lines 2720–2980: The remaining lines cover the situation where a character wishing to drink has to accept a drink from another member of the party — in which case he must give something to that member in return. He may not always be lucky! Whatever happens, **character%** is always kept up to date. Finally, **Procchar_check** is called, and updates the stat windows.

Procchar_search

```
5790 DEFine PROCedure char_search(c%,
value)
5800 FOR j=6 TO 8
5810 IF character%(c%,j)=value THEN s
flag=j
5820 NEXT j
5830 END DEFine char_search
```

Procchar_search commentary

Actually searches **character%** for the value passed to it from the procedure, in this case a number of lines in **Procdrink**.

Procplural

```
5660 DEFine PROCedure plural
5670 PRINT #9,monster$(map%(v,h,3));
5680 IF map%(v,h,4)>1 THEN
5690 PRINT#9, "s"
5700 RETurn
5710 ELSE
5720 PRINT#9, " "
5730 END DEFine plural
```

Procplural commentary

If the number of monsters at the location is greater than 1, then an 's' is added to the printed description.

Procpool

```
5740 DEFine PROCedure pool
5750 CLS#9
5760 PRINT#9,"here is a magical pool"
5770 pl=1
5780 END DEFine pool
```

Procpool commentary

Prints the information that here is a magic pool, and sets the flag **pl** to 1, for use in other routines.

Procscore

```
6360 DEFine PROCedure score
6370 sc=0
6380 FOR i=1 TO LEN(char$)
6390 FOR j=6 TO 8
6400 IF character%(char$(i),j)<>0 THE
N sc=sc+1
6410 END FOR j
6420 END FOR i
6430 sc=(monsterskilled*(turns DIV 6)
)+sc
6440 CLS#9
6450 PRINT #9;"Score so far :";sc
```

```
6460 my_word
6470 END DEFine score
```

Procscore commentary

Called by lines 420–430 in the main loop. Here the score is computed, based on the number of turns the player has survived, monsters killed, and objects found (information on this last item being carried in **character%**). At the end of this procedure, **Procmy_word** is called.

Procmy_word

```
8430 DEFine PROCedure my_word
8440 spflag=2
8450 r=RND(6)
8460 IF r INSTR char$=0 THEN RETurn
8470 CLS#10
8480 PRINT #10,character$(r);" says:
";
8490 s=RND(3)+1
8500 SELect ON s
8510 =1
8520 grab r
8530 =2
8540 PRINT #10,"I want to rest"
8550 rest
8560 RETurn
8570 =3
8580 PRINT #10,"I'm thirsty"
8590 drink
8600 RETurn
8610 ON s =REMAINDER
8620 IF character%(r,6)=0 THEN grab r
:RETurn
8630 PRINT #10,"I'm fed up with carry
ing this"
8640 drop r
8650 RETurn
8660 END SELect
8670 END DEFine my_word
```

Procmy_word commentary

This procedure is called occasionally throughout the program, and gives an impression of characters talking among themselves. Actually, a random selection is made from a number of possibilities (including no action at all, in which case nothing will appear to happen).

If the number 1 is generated, then **Procgrab** is called, and the character takes an object or says something. If 2 is generated, then **Procrest** is called. If 3 is returned, then **Procdrink** is called. On other numbers, the character's Nerves are checked — if 0, then something is taken from him or something done to him, and if he still has nerves then he gets fed up with carrying something and drops it.

Procgrab

```
8680 DEFine PROCedure grab(a)
8690 IF character%(a,8)<>0 THEN ransa
y:RETurn
8700 b=0
8710 FOR kt= 1 TO LEN(char$)
8720 IF character%(char$(kt),6)<>0 AN
D k<>r THEN b=char$(kt)
8730 NEXT kt
8740 IF b=0 THEN ransay:RETurn
8750 character%(a,8)=character%(b,8):
character%(b,6)=0
8760 PRINT #10,"I'm taking the ";obje
ct$(character%(b,6))
8770 END DEFine grab
```

Procgrab commentary

This allows a character to 'nick' another's object!

Procrest

```
3020 DEFine PROCedure rest
3030 CLS#9
3040 p1=0
3050 PRINT #9,"You sit down to rest"
3060 s=RND (1 TO LEN(char$)):r=RND(1
TO 2)
3070 PRINT #9;character$(char$(s));"
feels stronger":PAUSE 100
```

```
3080 rt=RND(1 TO 3):character%(char$(
s),r)=character%(char$(s),r)+rt
3090 IF map%(v,h,4)<>0 THEN
3100 PRINT #9,"but you have forgotten
 the ";:plural
3110 PAUSE 100:combat
3120 RETurn
3130 ELSE
3140 re=RND(1 TO no_of_events)
3150 SELect ON re
3160 =1:cursmove
3170 =2:pool
3180 =3:voice
3190 =4:randomonster
3200 END SELect
3205 END IF
3210 char_check char$
3220 END DEFine rest
```

Procrest commentary

Called by lines 450–460 in the main loop. Checks for the presence of a monster in the room, and calls **Procplural** if necessary — that is, if there is more than one monster here. Then the program goes to **Proccombat**, and you know what that procedure will do! If there's no monster, the procedure is not finished. A random number is generated (using the variable **no-of-events**, the value of which is initially set at 4 in line 7060 (**Procinit**) upon which several procedures are called. Finally, **Procchar_check** updates the display of each character.

The four possibilities here are a little bit of fun and, although the magic pool has proved useful before in this program, the others were results of afterthoughts while designing the program. The procedures are quite simple to follow, and show how a little extra thought and imagination can add an awful lot of atmosphere.

Proccursmove

```
3300 DEFine PROCedure cursmove
3310 FOR j=1 TO LEN(char$)
3320 FOR k= 6 TO 8
3330 IF object%(character%(char$(j),k
),4)=3 THEN
3340 REPeat hloop
```

```
3350 s=RND(1 TO 24)
3360 r=RND(1 TO 12)
3370 IF map%(r,s,1)<>0 THEN v=r:h=s:E
XIT hloop
3380 END REPeat hloop
3390 END IF
3400 NEXT k
3410 NEXT j
3420 END DEFine cursmove
```

Proccursmove commentary

This is the first of the new procedures and effectively moves the whole party, if one of the characters is carrying a cursed object (**line 3330**), to a random location — a very nasty one! Among other things, this makes it a little harder for the player to map the complex and will spoil any planned strategy.

Procvoice

```
9160 DEFine PROCedure voice
9165 CLS#10
9170 CLS#9
9180 PRINT#9;"A distant voice says: "
;
9190 qr=RND(1 TO 4)
9200 SELect ON qr
9210 =1:ransay
9220 =2:distance 2
9230 =3:distance 3
9240 =4:cursay
9250 END SELect
9260 END DEFine voice
```

Procvoice commentary

This is the second of the new procedures and gives back something of what we took away with **Proccursmove**. The effect is of a disembodied, distant voice giving some advice to the party. There are four possibilities here (in three procedures — **Procdistance**, **Proccursay** and **Procrandomonster**), depending on the random value given to **gr** at **line 9190**.

Procdistance

```
9340 DEFine PROCedure distance(item)
9350 find=0:k=1
9360 REPeat bloop
9365 j=1
9370 REPeat cloop
9380 IF map%(j,k,item)<>0 AND (j<>v 0
R k<>h) THEN find=ABS((ABS(v-j))-(ABS
(h-k)))
9390 IF j>11 OR find<>0 THEN EXIT clo
op
9400 j=j+1
9410 END REPeat cloop
9420 IF k>23 OR find <>0 THEN EXIT bl
oop
9430 k=k+1
9440 END REPeat bloop
9450 IF find=0 THEN RETurn
9460 SELect ON item
9470 =3 :PRINT #9,"A monster is ";fin
d;" rooms away"
9480 =2 :PRINT #9,"A ";object$(map%(j
,k,2));" is ";find;" rooms away"
9485 END SELect
9490 END DEFine distance
```

Procdistance commentary

If the value given to **gr** is 2 or 3, then this procedure is called and gives the party the whereabouts of monsters or objects. The value of **gr** decides whether we are dealing with a monster or an object.

Proccursay

```
9270 DEFine PROCedure cursay
9280 FOR i=1 TO LEN(char$)
9290 FOR j=6 TO 8
9300 IF object%(character%(char$(i),j
),6)<>0 THEN PRINT #9,character$(char
$(i));" is cursed"
9310 END FOR j
9320 END FOR i
```

```
9325 PRINT #9,"Be careful"
9330 END DEFine cursay
```

Proccursay commentary
The voice, if this procedure is called, warns the party if a cursed object is being carried by one of the characters.

Back at **Procrest**, we come to the fourth random event, at line 3190. This is another nasty one to keep the party on their toes.

Procrandomonster

```
3230 DEFine PROCedure randomonster
3240 s=RND(1 TO 16)
3250 map%(v,h,4)=1:map%(v,h,3)=s
3260 CLS#14:PRINT #14," Suddenly a ";
monster$(s);" appears":PAUSE 100
3270 cell_display
3280 combat
3290 END DEFine randomonster
```

Procrandomonster commentary
This procedure calls up a random monster, updating the cell display and entering the Battle sequence.

Proctalk

```
5590 DEFine PROCedure talk
5600 CLS#9
5610 IF map%(v,h,4)=0 THEN PRINT #9;"
Your chatter might attract a monster"
:RETurn
5620 spflag=1
5630 PRINT#9,"A ";monster$(map%(v,h,3
));" says: ";
5640 ransay
5650 END DEFine talk
```

Proctalk commentary

Called by lines 460–470 in main loop. This is a very short procedure, but quite important. It is called when the player wants to talk to a monster and calls a larger procedure, **Procransay**. This routine is also used by the characters, so some of the utterances are a little inappropriate! If there is no monster here, then the program simply returns to the main menu.

Procransay

```
8910 DEFine PROCedure ransay
8920 r=RND(3)
8930 SELect ON r
8940 =1
8950 bita=RND(5)+1:bitb=RND(5)+1
8960 PRINT #(8+spflag);first$(bita);"
 ";second$(bitb)
8970 ON r=REMAINDER
8980 PRINT #(8+spflag),"I'd like to "
;nasty$(RND(5)+1);
8990 IF spflag=2 THEN
9000 PRINT #(8+spflag)," a ";monster$
(RND(15)+1)
9010 ELSE
9020 PRINT #(8+spflag)," an adventure
r"
9030 END IF
9040 =2
9050 PRINT #(8+spflag),"Where's the "
;object$(RND(35)+1);" then?"
9060 END SELect
9070 END DEFine ransay
```

Procransay commentary

This procedure (RANdom SAYings) is called from a couple of places in the program. To ensure that what is said is not *too* unusual, and that monsters say more or less monstrous things while characters say more or less human things, the flag **spflag** is set to 1 when the procedure is called from **Procvoice**, and 2 when called from **Proctalk**.

Procgive

```
5990 DEFine PROCedure give
6000 CLS#14
6010 IF map%(v,h,4)=0 THEN CLS#9:PRIN
T #9,"There's no-one here to take it"
:RETurn
6020 PRINT #14,"What are you going to
 give?"
6030 INPUT#14, gift$
6040 FOR i=1 TO LEN(gift$)
6050 IF CODE(gift$(i))<97 THEN gift$(
i)=CHR$(CODE(gift$(i))+32)
6060 NEXT i
6070 gno=0
6080 FOR i=1 TO 36
6090 IF object$(i,2)=gift$ THEN gno=i
6100 END FOR i
6110 IF gno=0 THEN PRINT#9,"No such o
bject":RETurn
6115 okfl=0
6120 FOR i=1 TO LEN(char$)
6130 FOR j= 6 TO 8
6140 IF character%(char$(i),j)=gno TH
EN okfl=1
6145 END FOR j
6150 END FOR i
6155 IF okfl=0 THEN RETurn
6160 IF bargain=1 THEN bargain=0:acce
pt
6170 IF RND(1 TO 9)<(monster%(map%(v,
h,3),5)) THEN accept
6180 char_check char$
6190 RETurn
6200 END IF
6230 PRINT #9;"No-one has the";object
$(gno)
6240 END DEFine give
```

Procgive commentary

Called by lines 480–490 in main loop. Allows the player to make a gift to a monster. After checking to ensure that there is a recipient at the present

location, the procedure checks that the gift the player wishes to give does in fact exist and then checks that one of the party currently carries it. If the variable **bargain** is set at 1, then the monster will accept the gift (see **Procaccept**). If it is at 0 (it is initially set to 0 at line 7000 in **Procinit**) then there is still a random chance that the monster may accept anyway. Finally, **Procchar_check** is called in **line 6180** for an update of the display.

Procaccept

```
6250 DEFine PROCedure accept
6260 map%(v,h,3)=0:map%(v,h,4)=0
6261 FOR i=1 TO LEN(char$)
6262 FOR j=6 TO 8
6263 IF character%(char$(i),j)=gno TH
EN
6264 character%(char$(i),j)=0
6265 END IF
6266 END FOR j
6267 END FOR i
6270 PRINT #9;"Thanks for the gift, y
ou can pass"
6275 cell_display
6280 END DEFine accept
```

Procaccept commentary

Updates **map%** and prints the monster's acceptance of the gift.

Back at the main loop, line 500 initiates the exit procedures, which includes the option of saving the game along with all the data.

Before this, however, the program checks to see if the various victory conditions have been met. Whatever happens, we now exit the main loop. Lines 530 and 540 read **object%** to see if any character has the quest object (as defined by the user in QLAG) — if so, then **vic** is set to 1, and **Proccelebrate** is called. If all the characters are dead (that is, if **char$** = 0) then **Proccry** is called.

Proccry

```
6290 DEFine PROCedure cry
6295 FOR j= 1 TO 8
6300 FOR i =8 TO 13
```

```
6310 PAPER#(i),0
6315 INK #(i),i-7
6320 CLS#(i)
6325 PAUSE ((8-j)*2)
6330 PRINT #(i),"SHAME!!"
6340 NEXT i
6345 NEXT j
6350 END DEFine cry
```

Proccry commentary
A colourful display heralds the unfortunate demise of your party! And the end of the program.

Proccelebrate

```
6570 DEFine PROCedure celebrate
6580 FOR j=1 TO 10
6585 FOR i=8 TO 13
6590 CLS#(i)
6595 PAPER #(i),i-7
6600 PAUSE 10
6610 PRINT #(i),"Congratulations"
6620 NEXT i
6630 NEXT j
6640 END DEFine celebrate
```

Proccelebrate commentary
Again, a colourful celebration of your victory.

The program may find, however, that neither of these conditions has been met, and then the player is given the chance of saving all the data, and the position of the party.

Procfile_out

```
7870 DEFine PROCedure file_out
7880 PRINT #14,"Please type name of g
ame"
7890 INPUT #14,game$
7900 PRINT #14,"Place the cartridge i
```

```
nto Microdrive 1"
7910 PRINT #14,"And press <Enter>"
7920 REPeat get_loop
7930 in$=INKEY$
7940 IF in$=CHR$(10) THEN EXIT get_lo
op
7950 END REPeat get_loop
7960 gn$="MDV1_"& game$ & "numbers"
7970 OPEN_NEW #15, gn$
7980 FOR i=1 TO 36
7990 FOR j=1 TO 6
8000 PRINT #15,object%(i,j)
8010 NEXT j
8020 NEXT i
8030 FOR i=1 TO 12
8040 FOR j=1 TO 24
8050 FOR k=1 TO 4
8060 PRINT #15,map%(i,j,k)
8070 NEXT k
8080 NEXT j
8090 NEXT i
8100 FOR i=1 TO 16
8110 FOR j=1 TO 6
8120 PRINT #15,monster%(i,j)
8130 NEXT j
8140 NEXT i
8150 FOR i=1 TO 6
8160 FOR j=1 TO 5
8170 PRINT #15,character%(i,j)
8180 NEXT j
8190 NEXT i
8200 PRINT#15,turns
8210 PRINT#15,monsterskilled
8220 PRINT#15,v
8230 PRINT#15,h
8240 CLOSE #15
8250 gm$="mdv1_"&game$&"names"
8260 OPEN_NEW #15, gm$
8270 FOR i= 1 TO 12
8280 FOR j=1 TO 24
8290 PRINT #15,map$(i,j)
8300 NEXT j
8310 NEXT i
8320 FOR i= 1 TO 36
```

```
8330 FOR j=1 TO 3
8340 PRINT #15,object$(i,j)
8350 NEXT j
8360 NEXT i
8370 FOR i =1 TO 6
8380 PRINT #15,character$(i)
8390 NEXT i
8400 PRINT#15,char$
8410 CLOSE #15
8420 END DEFine file_out
```

Procfile_out commentary

As we saw some time back when discussing **Procinfile**, this whole procedure can be transported from the previously-typed QLAG (our Generator) and so, following the brief instructions in **Procinfile**, a lot of time can be saved by saving and merging the similar routine from QLAG — don't forget, however, to change the window numbers as necessary. The name of the procedure in QLAG is the same and can be found at QLAG lines 9000–9500.

We've arrived finally at the end of the program. But while typing in the program has been, up till now at any rate, a rather pleasant and (we hope!) therapeutic pastime, we come now to a spell of drudgery. The data to be used by the graphic routines in the adventure are, unfortunately, very long and the sheer hard work will prove an arduous task. As we've already hinted at the start of the program, it is not absolutely necessary to go through the ordeal — you may quite understandably decide to create a text adventure. The framework of QLAG can be used just as well for this purpose, in which case these lines of data will not be needed. Should you decide, though, to go the graphic route, then you will need to get your head down, your fingers flexed, and bash away at the keyboard.

There are two ways to alleviate the sheer terror when faced with all these lines — the first is to read the next chapter, where you will meet a program which we've written to enable you to design your own graphics. The data you need will be given to you, and typing in your own figures must be more rewarding in the long run. The second way means that you don't need to hack in every single line of our data to get going: if you just want to see how the adventure plays, or if you want to decide whether you want a text or graphic adventure, all you have to do is use QLAG to describe a couple of monsters and type in the data required. If this is the case, then don't forget to DIMension **monster%** in QLAG line 12080 to a figure of say (3,6), if this is enough for you — and, of course, re-

DIMension it afterwards! Save this data, run QAD and play it through. If you like the way it's going, then you can type in the rest of the data.

If you don't want all the graphics, and that includes the graphic representation of each room, then you must delete **Procmon_show**, **Procobj_show**, **Procdoor_draw** and **Procgraphic** and calls to these procedures. They are the graphic routines, and you won't need them in a text adventure.

Data statements

```
9500 DATA "I'd like to","go home","st
rangle","Let's ","kill something","an
nihilate","Why don't we","have a drin
k","torment"
9510 DATA "I think you should","have
a rest","make a stew from","Its time
to","steal some treasure","beat up","
I want to","retire","roast"
9520 DATA "snake","ghost","wild dog",
"roper","gnome","dragon","spider","tr
oll","ogre","wizard","hulk","ghoul","
ostrich","vulture","demon","lionman"
9530 DATA "Attack ","Defence","Spells
 ","Anger  ","Nerves","Greed  "
10000 REMark monster graphic data
10010 REMark snake
10020 DATA 18,2
10030 DATA 42,128
10040 DATA 554,168
10050 DATA 170,0
10060 DATA 162,168
10070 DATA 160,0
10080 DATA 168,0
10090 DATA 40,0
10100 DATA 40,0
10110 DATA 42,0
10120 DATA 10,128
10130 DATA 10,160
10140 DATA 0,160
10150 DATA 32,168
10160 DATA 128,40
10170 DATA 128,40
10180 DATA 160,40
```

```
10190 DATA 170,168
10200 DATA 42,160
10210 DATA 2,128
10320 REMark ghost
10330 DATA 18,4
10340 DATA 0,10815,0
10350 DATA 43260,43775,35535
10360 DATA 43260,35054,35335
10370 DATA 43260,35054,35335
10380 DATA 10300,43775,35335
10390 DATA 10300,43775,35335
10400 DATA 10300,43775,35335
10410 DATA 2575,43775,43775
10420 DATA 2575,43775,43775
10430 DATA 515,43775,41200
10440 DATA 0,43775,32960
10450 DATA 0,43775,32960
10460 DATA 0,43775,32960
10470 DATA 515,43775,41200
10480 DATA 515,43775,41200
10490 DATA 2575,43775,43260
10500 DATA 2575,43775,43260
10510 DATA 10815,43775,43775
10520 DATA 10815,43775,43775
10610 REMark wild dog
10620 DATA 7,4
10630 DATA 10815,33475,0
10640 DATA 8240,515,41200
10650 DATA 10815,43775,10943
10660 DATA 10815,43775,41200
10670 DATA 10300,10300,10815
10680 DATA 43260,2575,0
10690 DATA 32960,0,32960
10700 DATA 32960,0,32960
10910 REMark roper
10920 DATA 9,4
10930 DATA 40960,0,40960
10940 DATA 2048,2560,0
10950 DATA 35344,2048,8192
10960 DATA 33280,33280,10752
10970 DATA 40960,33280,512
10980 DATA 10752,43520,41476
10990 DATA 2050,10370,8836
11000 DATA 2560,170,43520
```

```
11010 DATA 10752,33320,40960
11020 DATA 43520,43520,43520
11210 REMark gnome
11220 DATA 10,4
11230 DATA 0,0,2575
11240 DATA 10943,43775,10943
11250 DATA 192,3,2767
11260 DATA 202,168,192
11270 DATA 714,8360,192
11280 DATA 2254,2220,192
11290 DATA 2815,43263,192
11300 DATA 2815,10431,192
11310 DATA 575,41215,0
11320 DATA 10752,2800,0
11330 DATA 43520,2560,32768
11500 REMark dragon
11520 DATA 28,6
11530 DATA 0,0,0,2
11540 DATA 0,0,12,136
11550 DATA 0,0,575,160
11560 DATA 0,0,60,168
11570 DATA 0,0,60,2060
11580 DATA 0,3,255,252
11590 DATA 0,63,191,240
11600 DATA 0,3,255,0
11610 DATA 0,255,240,32896
11620 DATA 2056,3,240,2248
11630 DATA 117,15,192,240
11640 DATA 562,15,192,192
11650 DATA 60,63,3,192
11660 DATA 15,63,15,0
11670 DATA 15,255,255,0
11680 DATA 3,255,240,0
11690 DATA 0,63,0,0
11700 DATA 0,15,0,0
11710 DATA 0,15,192,0
11720 DATA 0,195,240,0
11730 DATA 3,3,252,0
11740 DATA 12,0,255,0
11750 DATA 15,0,63,192
11760 DATA 3,240,15,192
11770 DATA 0,252,3,192
11780 DATA 0,63,3,192
11790 DATA 0,15,255,192,0,3,255,0,0,0
```

```
,252,0
11800 REMark spider
11820 DATA 10,4
11830 DATA 10815,0,43260
11840 DATA 8759,0,35036
11850 DATA 10815,0,43260
11860 DATA 2056,0,8224
11870 DATA 2056,10280,8224
11880 DATA 10794,43690,43176
11890 DATA 43690,170,43690
11900 DATA 43690,43690,43690
11910 DATA 8738,8738,8224
11920 DATA 8738,8738,8224
11930 DATA 8738,8738,8224
12110 REMark troll
12120 DATA 9,4
12130 DATA 0,43008,0
12140 DATA 0,35360,0
12150 DATA 20,43008,0
12160 DATA 21,80,43260
12170 DATA 85,64,43260
12180 DATA 84,32768,32960
12190 DATA 84,10752,32768
12200 DATA 35328,512,32768
12210 DATA 33280,0,32960
12220 DATA 41715,32960,0
12410 REMark ogre
12420 DATA 27,6
12430 DATA 0,10752,32768,0
12440 DATA 512,43520,43008,0
12450 DATA 512,2560,2048,0
12460 DATA 512,43520,43008,0
12470 DATA 2560,32810,10880,0
12480 DATA 2560,10880,35360,0
12490 DATA 2560,43520,43520,0
12500 DATA 40965,598,85,40960
12510 DATA 40965,598,85,40960
12520 DATA 40965,598,85,40960
12530 DATA 40965,598,85,40960
12540 DATA 40965,2650,32917,40960
12550 DATA 40965,43690,43177,40960
12560 DATA 43520,43520,43520,40960
12570 DATA 43520,43520,43520,40960
12580 DATA 43520,43520,43520,40960
```

```
12590 DATA 1,2650,32917,0
12600 DATA 1,2650,32917,0
12610 DATA 1,2650,32917,0
12620 DATA 5,2650,32917,64
12630 DATA 5,2650,32917,64
12640 DATA 5,2650,32917,64
12650 DATA 5,2634,32901,64
12660 DATA 21,578,32901,80
12670 DATA 21,514,32897,80
12680 DATA 21,514,1,80
12690 DATA 43520,514,512,43008
12695 DATA 43520,514,512,43008
12700 REMark wizard
12720 DATA 17,4
12730 DATA 0,16,0
12740 DATA 0,64,2060
12750 DATA 1,64,2060
12760 DATA 1,64,10815
12770 DATA 5,80,10815
12780 DATA 21,84,2060
12790 DATA 522,160,2060
12800 DATA 10,128,2060
12810 DATA 2,128,2060
12820 DATA 1,85,104
12830 DATA 1,69,2124
12840 DATA 5,64,2124
12850 DATA 5,80,2060
12860 DATA 21,80,2060
12870 DATA 21,84,2060
12880 DATA 85,84,2060
12890 DATA 85,85,2060
12900 DATA 85,85,2060
13010 REMark hulk
13020 DATA 12,6
13030 DATA 10240,2560,32768,40960
13040 DATA 43520,10752,41472,43008
13050 DATA 43520,8708,8768,43008
13060 DATA 43520,43520,43520,43008
13070 DATA 43520,41480,10880,43008
13080 DATA 43520,40970,10880,43008
13090 DATA 43008,43520,43008,43008
13100 DATA 10240,10752,40960,40960
13110 DATA 10240,255,252,40960
13120 DATA 10240,252,252,40960
```

```
13130 DATA 43520,252,764,43008
13140 DATA 43520,43176,43688,43008
13150 DATA 43520,43176,43688,43008
13310 REMark ghoul
13320 DATA 25,8
13330 DATA 0,43775,43775,32960,0
13340 DATA 0,33475,41200,32960,0
13350 DATA 0,33475,41200,32960,0
13360 DATA 0,43775,43775,32960,0
13370 DATA 0,10815,43775,0,0
13380 DATA 0,8755,8755,0,0
13390 DATA 0,8755,8755,0,0
13400 DATA 0,10815,43775,0,0
13410 DATA 0,515,41200,0,0
13420 DATA 43260,515,41200,2575,32960
13430 DATA 43775,43775,43775,43775,32
960
13440 DATA 43260,0,32960,2575,32960
13450 DATA 8240,43775,43775,33475,0
13460 DATA 8240,0,32960,515,0
13470 DATA 8240,10815,43775,515,0
13480 DATA 8240,0,32960,515,0
13490 DATA 8240,2575,43260,515,0
13500 DATA 41715,41200,33475,41715,32
960
13510 DATA 41715,43775,43775,41715,32
960
13520 DATA 515,41715,41715,41200,0
13530 DATA 0,32960,32960,32960,0
13540 DATA 0,32960,0,32960,0
13550 DATA 0,32960,0,32960,0
13560 DATA 0,32960,0,32960,0
13570 DATA 0,32960,0,32960,0
13580 DATA 10815,41200,515,43775,0
13590 REMark ostrich
13600 DATA 16,4
13610 DATA 168,0,0,154,128,0
13620 DATA 170,130,160,40,2,106
13630 DATA 8,2,170,8,2,128
13640 DATA 10,10,0,2,8,0
13650 DATA 2,8,0,2,168,0
13660 DATA 10,170,0,10,170,0
13670 DATA 2,168,0,2,8,0
13680 DATA 2,8,0,2,8,0,10,10,0
```

```
13890 REMark vulture
13900 DATA 7,6
13910 DATA 0,10773,41040,0,0,8793,834
0,0
13920 DATA 1,2645,32853,0,5,534,81,64
13930 DATA 20,518,64,80,80,21,80,20
13940 DATA 64,16,16,4,64,68,68,4
14190 REMark demon
14200 DATA 18,6
14210 DATA 64,0,0,16,64,32768,8192,16
14220 DATA 80,32768,8192,80,80,8192,3
2768,80
14230 DATA 84,8197,32769,80,20,21,65,
64
14240 DATA 21,37,133,64,21,21,69,64
14250 DATA 5,5,5,0,5,85,85,0
14260 DATA 1,85,84,0,5,85,84,0
14270 DATA 5,85,85,0,5,21,69,0
14280 DATA 20,21,65,64,20,80,81,64
14290 DATA 80,64,16,80,576,40960,4300
8,16,576,8192,34816,16
14490 REMark lionman
14500 DATA 21,6
14510 DATA 2,682,32896,0,2,10922,4112
0,0
14520 DATA 2,10409,64,0,10,10922,4112
8,0
14530 DATA 10,43690,43176,0,10,43176,
43192,0
14540 DATA 2,43176,8240,0,0,43176,0,0
14550 DATA 0,43690,0,41120,160,10794,
43690,43196
14560 DATA 160,10794,41120,8224,8224,
10794,0,0
14570 DATA 8224,10794,0,0,8224,43690,
32896,0
14580 DATA 33410,10794,32896,0,33410,
10794,32896,0
14590 DATA 10280,2570,32896,0,0,2570,
32896,0
14600 DATA 0,2570,32896,0,0,2570,0,0
14610 DATA 0,2056,0,0,0,10794,41120,0
14660 REMark Object data
14670 DATA 5,2,0,8240,0,32960,515,0,3
```

```
5020,0,8240,0,35020,0
14680 DATA 8,2,0,8224,0,32896,2570,0,
10794,0,43176,0,84,0,84,0,16,0,16,0
14690 DATA 5,2,2,160,10,160,10,160,42
,128,168,0,160,0
14700 DATA 3,2,10260,0,85,85,10260,0,
43605,0
14710 DATA 5,0,2060,10815,2060,2060,2
060,2060
14720 DATA 7,0,10280,33410,33410,514,
514,514,514,514
14730 DATA 6,0,43775,43523,43523,4352
3,43523,43523,43523
14740 DATA 1,0,2575,2575
14750 DATA 6,2,2060,0,10815,0,2060,0,
42,0,170,128,170,128,42,0
14760 DATA 5,0,34884,8208,34884,8192,
8192,8192
14770 DATA 7,0,40,43690,43690,40,40,4
0,40,40
14780 DATA 3,0,33345,10260,10260,3334
5
14790 DATA 3,0,10280,33410,33410,1028
0
14800 DATA 5,2,32768,0,40960,0,10240,
0,10752,0,2560,40960,2560,43008
14810 DATA 4,2,8224,32896,32896,8224,
32896,8224,8224,32896,2570,0
14820 DATA 5,2,10815,0,43775,32960,32
960,32960,41715,32960,41715,32960,417
15,32960
14830 DATA 3,2,43775,43260,515,43775,
33475,43775,10815,43260
14840 DATA 3,0,34986,8874,34986,8874
14850 DATA 5,2,43775,41200,43775,4120
0,10815,32960,10815,32960,2575,0,2575
,0
14860 DATA 2,0,32,168,40
14870 DATA 2,0,65,85,65
14880 DATA 4,0,80,80,80,85,85
14890 DATA 6,2,43775,35016,32960,8224
,8240,34956,2574,2060,515,2060,0,3502
0,0,10300
14900 DATA 2,0,43690,43710,43690
```

```
14910 DATA 6,0,10,8,40,168,168,170,17
0
14920 DATA 2,2,5,0,5,0,85,80
14930 DATA 3,0,8240,43260,8240,8240
14940 DATA 4,0,43260,43260,8240,8240,
43260
14950 DATA 2,2,34952,32896,41642,3289
6,43690,32896
14960 DATA 5,0,8224,43192,8224,8224,8
224,8224,8224
14970 DATA 2,4,0,41200,0,170,41210,17
0,0,41200,0
14980 DATA 4,2,43605,41044,10773,3283
2,10773,32832,10773,32832
14990 DATA 4,2,10,0,8240,32960,32960,
8240,32960,8240,8240,32960
15000 DATA 5,0,515,2575,10300,41200,3
2960,32960
15010 DATA 5,0,8224,43176,43092,43176
,43176,43176
15020 DATA 2,0,10260,43605,43605
```

At last!

Whether you've typed in the data or decided against it, we have reached the end of the listing.

Now all that needs to be done is to play the game — we believe that this game is not only a great utility for anyone interested in this sector of home computer programs, but the best adventure to appear for the QL. . . .

CHAPTER 8
QL Graphics

The QL has no 'user-defined graphics', as the Spectrum and other popular micros have. Therefore, if we want to create monsters by colouring pixels on the screen rather than by drawing lines and filling in shapes, we have to send values directly to the screen memory itself.

The QL has a memory-mapped screen. This means that there is at least one memory location corresponding to each 'location' on the screen. So if we know which bit or byte in RAM controls which piece of the screen, we can alter the values in that portion of RAM and so alter the screen display.

The start of the screen memory for the QL is 131072. If you POKE a value to this location, some arrangement of dots will appear at the upper righthand corner of your screen (provided that all the dots the QL can address can actually be displayed on your TV or monitor). In low resolution mode (the mode used by both the programs in this book, QLAG and QAD), the screen is arranged in units of four pixels. You cannot address *less* than four pixels at a time. Each set of four is controlled by a pair of bytes, called a 'word'. The word is divided into eight pairs of bits and one pair in each byte holds the information for one pixel on the screen. The arrangement looks like this:

BYTES	high byte				low byte			
BITS	7/6	5/4	3/2	1/0	7/6	5/4	3/2	1/0
PIXEL	1	2	3	4	1	2	3	4
COLOUR	F/G	F/G	F/G	F/G	R/B	R/B	R/B	R/B

(F=Flash, G=Green, R=Red, B=Blue)

Now, if we want to set the fourth pixel on the screen to red, we must set bit 1 of the low byte. If we want the second and third pixels to be green, we must set bits 4 and 2 of the high byte. If we want a colour which is not one of these primary colours, then we must set two or more bits which control the same pixel. To produce magenta in pixel 3, we set the red bit and the blue bit for pixel 3, which is bit 2 and bit 3 of the low byte. If we want white in pixel 1, we set the red, blue and green bits for that pixel, ie, low byte bits 6 and 7, high byte bit 6.

As any combination of set and unset bits in a word represents a number

between 0 and 65535, there are 65535 possible colour patterns (including Flashing) in a group of four pixels. It is quite tedious to work out which bits should be set for which pixel on the screen (especially as the pixels are counted from left to right, the bits from right to left): and so, to calculate the decimal number coded by a particular combination of bits, we have included a short and crude program to do the job for you — it did the job for us, when we designed the graphics for the adventure program in this book!

Simply draw your graphic on graph paper and divide it into horizontal units of four dots. Then for each unit run one cycle of the program (it is an endless REPEAT loop). The program asks for the initial letter of the colour of each of the four pixels, referring to them by the righthand bit of the bytes of each unit (black is x because it is no colour, and we do not want to confuse it with blue). The program then asks for 'Bit 1', and you should respond by typing the initial letter of the colour of the lefthand pixel in your unit. When all four have been typed in, the program will give you a decimal value. Now if you POKE this value to a screen address you will see your pattern of coloured dots magically appear. As it stands, the program does not cater for flashing dots because making one dot flash makes the rest of the line flash. It does cater for blue, red, magenta, black, cyan, yellow, white and green.

All the monsters and objects in QAD are drawn using codes derived from this program. A pointer is restored to the first line for the block of DATA for the required graphic. The number of horizontal and vertical iterations are read in from this data. Then, as those horizontal groups of four are multiplied by the height of the graphic (ie, the number of pixels vertically), data is read in and POKEd to the correct screen addresses.

Graphic Set-up

```
100 REMark GRAPHIC SET-UP PROGRAM
110 CLS
120 REPeat main
130 s=0:t=0
140 FOR i=1 TO 7 STEP 2
150 fact=0
160 PRINT "Bit ";i
170 col$=" "
180 REPeat loop
190 cpos=0
200 col$=INKEY$
210 IF col$="" THEN GO TO 200
220 cpos =col$ INSTR "brmxcywg"
```

```
230 IF cpos <>0 THEN EXIT loop
240 END REPeat loop
250 lsb=cpos MOD 4
260 IF lsb =1 OR lsb=3 THEN fact=fact
 +(2^(i-1))
270 IF lsb=2 OR lsb=3 THEN fact=fact+
(2^i)
280 IF cpos>4 THEN
290 msb=2^(i+8)
300 ELSE
310 msb=0
320 END IF
330  t= msb+fact
340 s=s+t
350 NEXT i
360 PRINT "Code= ";s
370 END REPeat main
```


GREENWICH LIBRARIES

Index 1

GENERAL

A

Adams, Scott 28, 37ff, 53
Adams, Alexis 37, 38
Adventure 200 14
Adventure International 37
Adventure Quest 14
Adventure, history of 5ff
Adventureland 37, 38
Adventures, one of the original programs 9, 12, 32
After the Fire 28
Ahl, David 12
Andre, Ken 8
Apple 9, 11, 35, 37, 39
Archon 33
Arrow of Death 27
Artic 12
Atari Adventures 4
Atari 26, 35, 37, 53
Avalon Hill 35

B

BBC 27, 37, 45
Bitwise logic 63
Black Rod 7
Burlyn, Michael 42

C

Catacombs 13
Channel 8 26, 37
Chess 8
Circus 27
Codename MAT 51
Colossal Adventure 14, 51
Colossal Cave 2, 9, 21, 32, 33, 37, 39
Commodore 27, 35, 39, 47, 51, 54
Count, The 38
Creative Computing 12
Crowther, Willy and Woods, Don 9, 11, 21, 37, 39

D

Deadline 39
DEC 39
Doomdark's Revenge 50
Dragon (computer) 27, 37
Dragon (creature) 21, 34, 43
Dungeon Adventure 14, 21–26, 28
Dungeon Builder 54
Dungeons & Dragons (D&D) 8, 13, 32, 34, 35

E

Electronic Arts 33
Escape from Pulsar 7 27
Eye of the Moon 50

F

Fantastic Voyage 13
Feasibility Experiment 27
Fifth 54
Foilkade 13, 14

G

Galaxians 1
Gazzard, Sue 21
Ghost Town 38

Gilsoft 13
Golden Apples 13
Golden Baton, The 27
Golden Ring 19
Golden Voyage 38
Gorf 1
Ground Zero 13
Gygax, Gary 8

H
Hammurabi 12, 50
Hobbit, The 33, 43–45
Howarth, Brian 27, 37
Hulk 53

I
IBM 10
Inca Treasure 12
Infocom 15, 39–42, 53

J
Jet Set Willy 3

K
Kong 1

L
Level 9 14, 21, 26, 32
LISP 39
Little Wars 8
Lord of the Rings 42, 52
Lords of Midnight 47–50, 52
Lords of Time 21

M
Manic Miner 3
Melbourne House 33, 42–43, 52
Middle Earth series 21
Midwinter 28
Miner 2049'er 3
Mission Impossible 38
Monsters 32–35
Mystery Fun House 38

N
Nascom 27

O
Orb, The 14, 54
Oric 45
Osborne 39
Othello 50

P
Pacman 1
Pearson, Jyym 37
PET 9, 11
Pirate's Cove 38
Planet of Death 12
Planetfall 40
Preppie 37
Procedures, *see* separate index
Psytron 47
Pyramid of Doom 38

Q
QAD, *see* separate index
Qix 8
QL 3, 37, 45, 50–52
QLAG, *see* separate index
Quicksilva 14, 35
Quill, The 13, 54

R
RPG (role-playing games) 9, 32, 35

S
Savage Island, Pts 1 & 2 38
Scope 54
Ship of Doom 12
Silicon Dream Series 21
Silmarillion 42
Sinclair, Sir Clive 1, 11
Snakes and Ladders 14
Snowball 21
Sorcerer's Castle 14
Space Invaders 1
Star Trek 12, 50, 51

Starcross 40
Strange Odyssey 38
Suspended 4, 41, 42

T
Tandy TRS-80 9, 11, 27, 37, 39
Telengard 35
Ten Little Indians 28
Time Machine, The 27
Tolkien Companion 42
Tunnels & Trolls 8, 35

V
Valhalla 45–47
Velnor's Lair 14, 35–36
Vocabulary 29–32
Volcanic Dungeon 14
Voodoo Castle 38

W
Wargames 8
Waxworks 28
Wells, H. G. 8
West 50
White Lightning 54
Witness 40
Wizard Akryz, The 27
Woods, Don *see* Crowther, Willy
Wumpus 12, 13, 14

Z
Zaxxon 8
Zkul 50
Zork 9, 11, 12, 39–40
ZX80 11, 12, 50
ZX81 11, 12, 50

Index 2

PROGRAMS

QLAG 57ff, 67ff, 106
DATA 85, 87
Exits 62
Main Variables 66
Marking the doors 64
Procafter 85, 87
Procassign_mon 79–80
Procassign_object 80–81, 82
Procbefore 85, 87
Proccharacter 89–90, 92, 93
Proccomob 85, 86
Proccur 74, 75
Procdescribe 61, 69, 74, 79
Procdisob 85–86
Procdoor_draw 73, 78
Procentrance 74, 76–77
Procexproc 77–78
Procfile_out 93–95, 96
Procget 70–71, 83, 85
Procinfile 95–96
Procinit 68–69, 70, 77
Procin_text 87–88
Procmap 61, 71–74, 75
Procmap_screen 84, 91–93
Procmenu 70
Procmessob 85, 86
Procmonster 61, 88–89, 92, 93
Procname 90–91
Procname_dec 81–82
Procnum_dec 81

Procobdat 62, 83–84, 88, 93
Procobdesc 83, 84–85
Procobject 61, 82–83
Procroom 74, 76
Procscr_mes 74, 75–76, 77
Program Handler 67
Setting up and using QLAG 57–66

QAD 57, 99ff
DATA 143–151
Drawing doors 64
Exits 62
Graphic set-up program 154–155
Change$ 113
Procaccept 139
Procattack 117–122, 123ff
Proccaution 121, 124
Proccelebrate 139, 140
Proccell_display 101, 106–107, 113
Proccharacterhurt 125–126
Procchar_check 110–111, 129, 133, 139
Procchar_move 101, 111–112, 113
Procchar_search 129
Procchar_select 105–106, 111
Procchekcurs 121, 124–125
Proccombat 101, 114–117, 121, 133
Proccry 139–140
Proccursay 134, 135–136
Proccursmove 133–134
Procdistance 134, 135
Procdoor_draw 64, 65, 66, 99, 109–110, 143
Procdrink 101, 113, 127–129, 132
Procdrop 101, 113–114
Procfile_out 101, 140–142
Procfrenzy 121, 124
Procgive 101, 138–139
Procgrab 132
Procgraphic 65, 99, 106–107, 110, 143
Procinfile 101, 104–105, 142
Procinit 101, 102–104, 133, 139
Procmonsterhurt 121, 125, 126
Procmonster_show 99, 107, 108, 143
Procmonup 121, 122
Procmy_word 111, 131–132
Procobj_show 99, 107, 108–109, 143
Procplural 126, 130, 133
Procpool 130
Procrandomonster 134, 136
Procransay 137
Procrest 60, 101, 132–133, 136
Procscore 101, 130–131
Procspell 121, 123
Procstrike 121, 123–124
Proctake 101, 126–127
Proctalk 101, 136, 137
Proctell 107, 109
Procvoice 60, 134, 137
Procwho 113, 122

Other titles from Sunshine

SPECTRUM BOOKS

Artificial Intelligence on the Spectrum Computer
Keith & Steven Brain — ISBN 0 946408 37 8 — **£6.95**

Spectrum Adventures
Tony Bridge & Roy Carnell — ISBN 0 946408 07 6 — **£5.95**

Machine Code Sprites and Graphics for the ZX Spectrum
John Durst — ISBN 0 946408 51 3 — **£6.95**

ZX Spectrum Astronomy
Maurice Gavin — ISBN 0 946408 24 6 — **£6.95**

Spectrum Machine Code Applications
David Laine — ISBN 0 946408 17 3 — **£6.95**

The Working Spectrum
David Lawrence — ISBN 0 946408 00 9 — **£5.95**

Inside Your Spectrum
Jeff Naylor & Diane Rogers — ISBN 0 946408 35 1 — **£6.95**

Master your ZX Microdrive
Andrew Pennell — ISBN 0 946408 19 X — **£6.95**

COMMODORE 64 BOOKS

Graphic Art for the Commodore 64
Boris Allan — ISBN 0 946408 15 7 — **£5.95**

DIY Robotics and Sensors on the Commodore Computer
John Billingsley — ISBN 0 946408 30 0 — **£6.95**

Artificial Intelligence on the Commodore 64
Keith & Steven Brain — ISBN 0 946408 29 7 — **£6.95**

Simulation Techniques on the Commodore 64
John Cochrane — ISBN 0 946408 58 0 — **£6.95**

Machine Code Graphics and Sound for the Commodore 64
Mark England & David Lawrence — ISBN 0 946408 28 9 — **£6.95**

Commodore 64 Adventures
Mike Grace — ISBN 0 946408 11 4 — **£5.95**

Business Applications for the Commodore 64
James Hall — ISBN 0 946408 12 2 — **£5.95**

Mathematics on the Commodore 64
Czes Kosniowski — ISBN 0 946408 14 9 — **£5.95**

Advanced Programming Techniques on the Commodore 64
David Lawrence — ISBN 0 946408 23 8 — **£5.95**

Commodore 64 Disk Companion
David Lawrence & Mark England ISBN 0 946408 49 1 **£7.95**

The Working Commodore 64
David Lawrence ISBN 0 946408 02 5 **£5.95**

Commodore 64 Machine Code Master
David Lawrence & Mark England ISBN 0 946408 05 X **£6.95**

Machine Code Games Routines for the Commodore 64
Paul Roper ISBN 0 946408 47 5 **£6.95**

Programming for Education on the Commodore 64
John Scriven & Patrick Hall ISBN 0 946408 27 0 **£5.95**

Writing Strategy Games on your Commodore 64
John White ISBN 0 946408 54 8 **£6.95**

COMMODORE 16/PLUS 4 BOOKS

The Working Commodore C16
David Lawrence ISBN 0 946408 62 9 **£6.95**

The Commodore C16/Plus 4 Companion
Brian Lloyd ISBN 0 946408 64 5 **£5.95**

ELECTRON BOOKS

Graphic Art for the Electron Computer
Boris Allan ISBN 0 946408 20 3 **£5.95**

The Working Electron
John Scriven ISBN 0 946408 52 1 **£5.95**

Programming for Education on the Electron Computer
John Scriven & Patrick Hall ISBN 0 946408 21 1 **£5.95**

BBC COMPUTER BOOKS

Functional Forth for the BBC Computer
Boris Allan ISBN 0 946408 04 1 **£5.95**

Graphic Art for the BBC Computer
Boris Allan ISBN 0 946408 08 4 **£5.95**

DIY Robotics and Sensors for the BBC Computer
John Billingsley ISBN 0 946408 13 0 **£6.95**

Artificial Intelligence on the BBC/Electron
Keith & Steven Brain ISBN 0 946408 36 X **£6.95**

Essential Maths on the BBC and Electron Computer
Czes Kosniowski ISBN 0 946408 34 3 **£5.95**

Programming for Education on the BBC Computer
John Scriven & Patrick Hall ISBN 0 946408 10 6 **£5.95**

Making Music on the BBC Computer
Ian Waugh ISBN 0 946408 26 2 **£5.95**

DRAGON BOOKS

Advanced Sound & Graphics for the Dragon
Keith & Steven Brain ISBN 0 946408 06 8 **£5.95**

Artificial Intelligence on the Dragon Computer
Keith & Steven Brain ISBN 0 946408 33 5 **£6.95**

Dragon 32 Games Master
Keith & Steven Brain ISBN 0 946408 03 3 **£5.95**

The Working Dragon
David Lawrence ISBN 0 946408 01 7 **£5.95**

The Dragon Trainer
Brian Lloyd ISBN 0 946408 09 2 **£5.95**

ATARI BOOKS

Atari Adventures
Tony Bridge ISBN 0 946408 18 1 **£5.95**

Writing Strategy Games on your Atari Computer
John White ISBN 0 946408 22 X **£5.95**

SINCLAIR QL BOOKS

Artificial Intelligence on the Sinclair QL
Keith & Steven Brain ISBN 0 946408 41 6 **£6.95**

Introduction to Simulation Techniques on the Sinclair QL
John Cochrane ISBN 0 946408 45 9 **£6.95**

Developing Applications on the Sinclair QL
Mike Grace ISBN 0 946408 63 7 **£6.95**

Mathematics on the Sinclair QL
Czes Kosniowski ISBN 0 946408 43 2 **£6.95**

The Working Sinclair QL
David Lawrence ISBN 0 946408 46 7 **£6.95**

Quill, Easel, Archive & Abacus on the Sinclair QL
Alison McCallum-Varey ISBN 0 946408 55 6 **£6.95**

Assembly Language Programming on the Sinclair QL
Andrew Pennell ISBN 0 946408 42 4 **£7.95**

GENERAL BOOKS

Home Applications on your Micro
Mike Grace ISBN 0 946408 50 5 **£6.95**

Sunshine also publishes

POPULAR COMPUTING WEEKLY

The first weekly magazine for home computer users. Each copy contains Top 10 charts of the best-selling software and books and up-to-the-minute details of the latest games. Other features in the magazine include regular hardware and software reviews, programming hints, computer swap, adventure corner and pages of listings for the Spectrum, Dragon, BBC, VIC 20 and 64, ZX 81 and other popular micros. Only 40p a week, a year's subscription costs £19.95 (£9.98 for six months) in the UK and £37.40 (£18.70 for six months) overseas.

DRAGON USER

The monthly magazine for all users of Dragon microcomputers. Each issue contains reviews of software and peripherals, programming advice for beginners and advanced users, program listings, a technical advisory service and all the latest news related to the Dragon. A year's subscription (12 issues) costs £10 in the UK and £16 overseas.

MICRO ADVENTURER

The monthly magazine for everyone interested in Adventure games, war gaming and simulation/role-playing games. Includes reviews of all the latest software, lists of all the software available and programming advice. A year's subscription (12 issues) costs £10 in the UK and £16 overseas.

COMMODORE HORIZONS

The monthly magazine for all users of Commodore computers. Each issue contains reviews of software and peripherals, programming advice for beginners and advanced users, program listings, a technical advisory service and all the latest news. A year's subscription costs £10 in the UK and £16 overseas.

For further information contact:
Sunshine
12–13 Little Newport Street
London WC2H 7PP
01-437 4343

Telex: 296275

NOTES

NOTES

NOTES

The Sinclair QL offers micro-adventurers unparalleled opportunities for creating exciting and demanding adventures. The enormous memory and superb graphics facilities of the machine are exploited in this book, as Tony Bridge and Richard Williams show you how to write your own adventure using their specially-devised Generator program.

The Generator is nothing less than an adventure-maker, which can be used to create unique adventures to your own design, taking full advantage of all the outstanding features of the Sinclair QL. The Generator is followed by a database program which can be used in conjunction with the Generator to provide the details of the adventure itself.

Sinclair QL Adventures is an invaluable guide both for adventure enthusiasts, and for those keen to explore this exciting area of microcomputing, using one of the most powerful home micros.

Tony Bridge is the Adventure correspondent for Popular Computing Weekly and MicroAdventurer. In his other life he sits in a dark underground cavern, making records with many of today's top recording artists.

Dr Richard Williams is a lecturer on computer courses and artificial intelligence. He is the author of numerous articles and books on computer-related subjects. He is currently working on a major series of software titles for the Sinclair QL.

SUNSHINE
ISBN 0 946408 66 1

GB £ NET +005.95
ISBN 0-946408-66-1
00595
9 780946 408665

£5.95 net

KP-645-430