# commodore 64 adventures

A guide to playing and writing adventures

mike grace

NL. 33.90

# commodore 64 adventures

## A guide to playing and writing adventures

mike grace

First published 1983 by:
Sunshine Books (an imprint of Scot Press Ltd.)
12–13 Little Newport Street,
London WC2R 3LD

Copyright © Mike Grace
Reprinted 1983

ISBN 0 946408 11 4

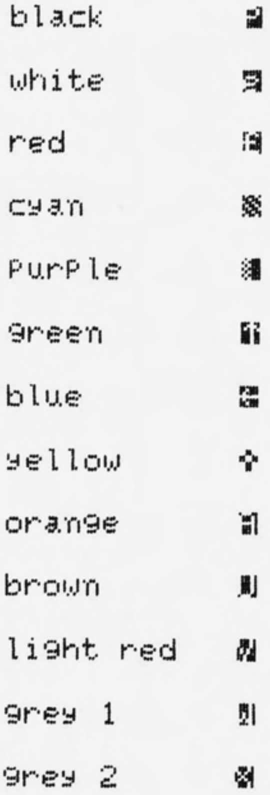
*All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording and/or otherwise, without the prior written permission of the Publishers.*

Cover design by Graphic Design Ltd.
Illustration by Ian Craig.
Typeset and printed in England by Commercial Colour Press, London E7.

# PROGRAM NOTES

A number of functions on the Commodore 64, as with other Commodore machines, are dictated by 'control characters' which are contained in ordinary strings and take effect when the string is printed. Control characters can normally be recognised by the fact that they are inverse characters (the colours of the background and foreground are reversed in the character position). The functions under the control of such characters include cursor position, print colour, inverse (RVS) on and off, cursor home and clear screen.

The following table shows the control characters as they appear in the programs in this book:



```
black
white
red
cyan
purple
green
blue
yellow
orange
brown
light red
grey 1
grey 2
```

| | |
|---|---|
| light green | ▩ |
| light blue | ▩ |
| gray 3 | ▩ |
| rvs on | ▩ |
| rvs off | ▩ |
| up | ▩ |
| down | ▩ |
| right | ▩ |
| left | ▩ |

# CONTENTS

*Page*

Foreword 11

Part 1

1 The Mystery and the Magic 15

2 Moving around an Adventure 25

3 Objects and Commands 43

4 Pitfalls and Perils 59

Part 2

5 Writing the Plot 71

6 Setting the Scene 85

7 Do you need help? 93

8 Sprites and Sound Effects 101

9 Where do we go from here? 119

10 Picking up nicely now 137

11 Not that way 149

12 Nice day for a dip 161

13 To talk of many things 177

14 A kiss in time 189

15 Looking to the future 201

Afterword 203

Appendices

A Table of Variables for Nightmare Planet 207

B The Line Numbers 209

C Overall Flow Chart for Nightmare Planet 215

# Contents in detail

CHAPTER 1
The Mystery and the Magic
The beginnings of Adventure, what is an Adventure game, Dungeons and Dragons, tips on playing Adventures, thinking about writing your own.

CHAPTER 2
Moving around an Adventure
Drawing and interpreting maps, converting your map to a grid, some programming hints, entering your locations into your program.

CHAPTER 3
Objects and Commands
Using arrays, entering objects into the grid, basic string handling, inventory, get and drop routines.

CHAPTER 4
Pitfalls and Perils
Adding excitement into your Adventure, programming perils, using variables to indicate hazards, solving your problems.

CHAPTER 5
Writing the Plot
Beginning the story, selecting an environment, choosing your quest, deciding on the hero, selecting the main characters, writing a synopsis, a simplified map, how to storyboard your plot.

CHAPTER 6
Setting the Scene
The initial concept, designing a title, starting to program, general guides on programming text.

CHAPTER 7
Do you need help?
Providing instructions for your games, LOADing and SAVEing to tape.

CHAPTER 8
Sprites and Sound Effects
Designing and creating sprites, moving sprites around, Victor and the Mealy Bug, selecting sound effects, adding sound to your responses.

CHAPTER 9
Where do we go from here?
Creating the map for Nightmare Planet, entering the locations, moving around the Adventure.

CHAPTER 10
Picking up nicely now
Airlocks and things, manipulating strings, adding to the inventory, getting and dropping things.

CHAPTER 11
Not that way
The six basic perils, displaying perils, preventing escape from the locations.

CHAPTER 12
Nice day for a dip
Killing routines, the eel peril, using variables to prevent movement, a general structure for Adventure perils.

CHAPTER 13
To talk of many things
Now you see Aurora — now you don't, dealing with quicksand, a dinosaur and a snake, climbing trees, adding time loops.

CHAPTER 14
A Kiss in Time
Help routines, shooting and firing, coping with the command eat, the ultimate kiss.

CHAPTER 15
Looking to the future
Extending your game, it's really up to you.

# FOREWORD

This book is about Adventure games and how to write your own. Although primarily a 'how to do it' book I have also tried to capture the excitement of playing this type of game and sprinkled quite a few hints and tips which should help Adventurers (novice and expert alike) to enjoy indulging in their passion.

When I started I was a fairly inexperienced programmer (I suppose I still am to an extent) and the thought of writing a game which contained any depth and mystery seemed formidable. I am glad to report that despite the apparent complexity and length of the final program it was quite easy — once I had solved the basic mechanics of constructing a framework for my program. My thanks go to David Lawrence who nudged me in the right direction on occasions when everything seemed to be getting too much. Thus, even if you are a newcomer to programming, this book is well within your capabilities.

Whilst the book will obviously enable you to enter and then play my own game — *Nightmare Planet* — it will also (and of more value in the end) give you a blueprint for creating your own Adventure right from scratch. It is not just a 'copy me and play it' book at all — and I suspect that for many Adventurers the thrill of creating your own masterpiece (to baffle and amaze your friends) is the ultimate satisfaction in programming and playing.

In the process of writing I have learnt about the Commodore 64 — a superb machine for Adventurers. The amount of memory gives plenty of scope for a complex and extensive plot and the graphics capabilities (whilst not really utilised in a text Adventure) will add that touch of professionalism to your game which make a program look and feel good.

The book is divided into two sections. The first covers the mechanics of creating a framework for your Adventure — how to construct locations and move between them, how to pick up objects and move them around your framework, and how to create hazards and perils for your player to encounter. This section contains a sample program called ADVENTURE GAME which is purely a simplified framework to illustrate the concepts (which can be used again and again).

The second section deals with creating the plot, writing the story, and programming into your 64 an Adventure — from conception to completion. The program *Nightmare Planet* (which will appear at first to be frighteningly long) is set out throughout the various chapters in modular form to help you when you start creating your own. Whilst it is possible to sit down and key in the whole thing in one marathon session (!!) if you work through chapter by chapter entering it a stage at a time it will help in planning your own program later.

Whilst this book can be read on the train or by the fireside it is really a book to be used beside the 64 or with a pencil and several reams of paper to play with. Certain pages contain charts or diagrams that I have found invaluable in my own planning — and if you have access to a photocopier I would suggest you take your own copies to help you. Whatever else though — have fun!

When I started (as I said before) I was really just an enthusiastic player of Adventures. By the time I'd finished I had become a better programmer and without doubt a better player as well. In fact my enthusiasm seemed to increase with each chapter I wrote.

My thanks go to Commodore for making a 64 available, to David Lawrence for his guidance and help, and to my family for their patience and their helpful suggestions whilst they were testing my Adventure.

# Part 1

# CHAPTER 1
# The Mystery and Magic

Chest heaving, sweat dripping down his bronzed body, muscles aching as if he had been stretched on the rack for a week, Karon teetered on the edge of a vast precipice that had appeared under his feet as if by evil design rather than a trick of nature. His eyes flickered behind him, hearing the dogs. Surely they would reach him in mere moments?

But which way to turn? How to escape? Behind him the armed might of the Emperor Draloor and his thrice accursed Zamrian hunting dogs (who were able, it was rumoured, to track a man through the very ocean itself) were closing in. To his left and right the walls of the mountains, a mere hundred paces or more, might well have been just feet away for all the cover they afforded. And ahead yawned a chasm — a natural cleft in the valley floor too wide to jump for even his mighty thews.

Trapped. Too late now to regret his indiscretions in the courtyard with Darleen (and how was he to know the wench was the emperor's daughter if she did not tell him?). He hefted his mighty broadsword into his right hand, and turned to face his pursuers.

At his feet a small gold ring sparkled in the evening sunlight, catching his attention. He stooped to pick it up — then with a suddenness the dogs were upon him.....

The above passage could well be from a book of heroic fantasy (another name for sword and sorcery — a form of fantasy writing popularised in the 1930s in the pulp magazines of that era), or part of the screenplay for a film. It could be the breakdown for a comic-book, or even the start of a game called *Dungeons and Dragons*. Or, of course, it could be a scene from an Adventure game.

Chances are if you're reading this book you already know about Adventure games and have probably played a few — but as the choice of this type of software is still pretty sparse for the Commodore 64 you may not if the 64 is your first computer.

Then again you may have heard or read about Adventures — but not really quite managed to take the time to key one in from a magazine (quite a marathon task) or better still succeed in playing a game through to the bitter end. Yet the concept of Adventure games has pretty much a universal appeal and Adventurers are on the increase everywhere.

## Dungeons and Dragons

If you are a newcomer to the role-playing concept I will spend just a few moments of introduction. Adventure games are so-called because they involve YOU — the player — in an adventure of one sort or another. Instead of just watching as a spectator (as in the film or book) you actually become the hero or heroine and to a certain extent you control your own destiny within the game. At no time can you be sure of a happy ending, more likely the opposite.

The concept was originally spawned from another type of role-playing game devised in the early 1970s called *Dungeons and Dragons*. Unlike most board and card games the players use their minds and imagination to move around a fantasy world created by the Dungeonmaster (someone in control of the game rather like the Banker in Monopoly) and also assume the *role* of various characters to add to the authenticity of the game. At first this use of the imagination of the mind was all that was needed, but later versions have tended to pander to a larger potential market by bringing in a board to give the players a tangible focus for their attention. This has the effect (in my opinion) of stifling the creative powers of the participants unlike the microcomputer. Perhaps this is one reason why adventures are gaining in popularity today.

An example of a game might well take the following pattern:

The players select various roles: an elf, a wizard, a warrior, a princess, a jester, etc. (Most of the original games centred around the fantasy world of fairy-tales and folklore — with a strong bias towards Tolkien's *Lord of the Rings*).

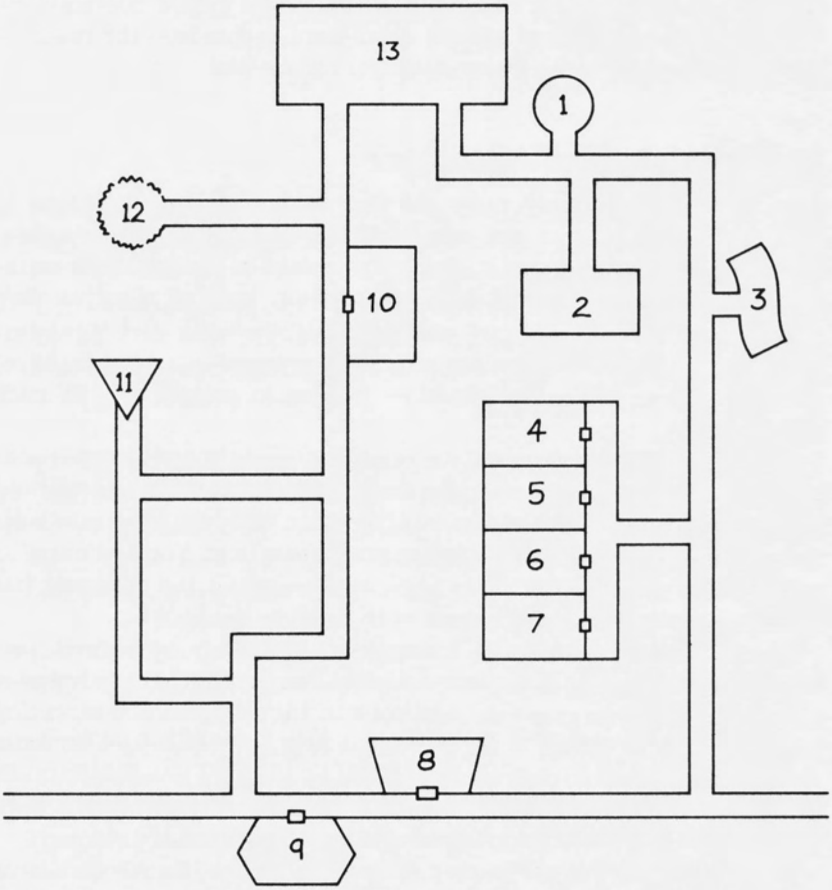
The Dungeonmaster creates the main plot of the game (usually some time before the players arrive). He will have drawn an intricate map of rooms (these are the dungeons) around which the players will move. He will also have created various treasures for them to collect and perils that he will spring upon them once they arrive in a particular room in his map. (See **Figure 1.1**).

The game begins. Each player takes a turn at moving — usually either east, west, north, south, up or down. As the player moves into a room so the Dungeonmaster will report on the type of room, the objects to be found there, and if applicable, the peril.

The player must then use his wits and imagination (as well as his skill in playing the game) to either battle and defeat the peril or escape (preferably with the object).

The appeal of the game is a combination of the mystery and magic of the fantasy world plus the tremendous scope of the human imagination. No

**Figure 1.1 A map from a Dungeon and Dragons game**



1. Entrance in a well in the garden.
2. Room with a map, sword and knife.
3. Space warp to level 6.
4. Cell with a giant spider guarding treasure.
5. Cell with six vicious goblins.
6. Empty cell.
7. Empty cell — door springs shut behind you.
8. Wizard's room. You need magic (level 4) to escape.
9. Shaft leading down to level 2. Slippery ladder.
10. Room with a wizard's hat. Only controllable with powerful magic (level 8).
11. Dungeon. No escape from here.
12. Dead-end passage.
13. Vast throne-room with giant statue with ruby eyes. On removal of a ruby the statue comes to life.

danger of ever playing the same game twice — for the variety of possible plots is extensive. It is rumoured that some games continue for weeks on end as hardened players delve even deeper into the maze of tunnels and traps that the Dungeonmaster has devised.

## The beginnings of Adventure

Also back in those early years the idea of introducing the game to computers took root in the minds of the mainframe programmers. There is a legend of computer operators trapped in front of their mainframes — eyes red-rimmed and bleary from lack of sleep as they wander lost around the vast complexes of passages and locations. Whatever the truth the game has not only continued under the takeover of the micro — it has blossomed — gaining in popularity with each passing year.

The main difference between the original *Dungeons and Dragons* concept and the *Adventure game* is that the latter becomes more personal and (in our time-conscious society) more acceptable. It is possible to save most games to tape part way through and return to them later. You don't need to collect a few people together to play. And somehow the computer has become a worthy opponent to beat — by hook or by crook!

The first variations tended to concentrate on text-only Adventures, perhaps appearing a trifle dull to the optically-battered fan of arcade games with an emphasis on graphics. And yet it is this very lack of images that gives the genre its appeal — for words can help us conjure up a far more vivid scene in our mind than graphics ever can. Until the programmers can produce pictures that look as good as a Walt Disney cartoon (and it's not too far off) I still prefer text Adventures to their graphic counterparts.

Don't let me put you off graphic Adventures though, for there are some around now which take on a completely different slant from the text-based versions, as long as they are not confused with a hybrid version where a character is manoeuvred around a 'graphic' number of locations with a joystick). My main objection is the use of valuable memory (that could be used to create a longer adventure or more hazards and objects along the way), and the graphics still tend to be a poor second to my own visualisation of the fantasy world that most Adventurers favour. But then I haven't played *The Hobbit* yet!

Because of my preference I have concentrated on the text-only type in this book. Having said that there are many good graphic versions around which are fun to play — I just put them in a different category.

I can't really leave this extremely brief introduction without mentioning Scott Adams, probably the best known name in the world of Adventures for the micro, who originally wrote his game for the TRS-80 and has subsequently adapted them for several popular machines (including the VIC 20 so it is possible the 64 will follow). These games have set the standard which many others have followed — and I have unashamedly copied the style with my own Adventure.

## Hints on playing Adventures

Although most Adventurers learn the tricks of the trade the hard way — by experience — there are quite a few standard ways of helping the novice (and let's face it — the experienced player as well at times) to improve his game. Like every hobby, occupation or sport, a bit of practice and a smidgeon of knowledge tends to increase enjoyment rather than spoil it.

However if you like to save yourself and make your own discoveries — skip this section altogether.

## Maps

I will be spending some time on maps in Chapters 2 and 9, so I will just add here that maps aren't essential but definitely help you to understand where you are in the Adventure. I always draw them on a large piece of paper and I tend to move from location to location at the start just planning where everything is.

It is worth noting the position of objects, where doors are, any special conditions (such as a light coming from a dark tunnel that you haven't explored yet) and any other clue. As almost *everything* you see on the screen has some relevance add it to your map. Try to use a piece of paper ten times larger than you'd think. You'll find it's too small.

In the really complex maze type of game with corridors, turning, levels, up and down, etc., it can get confusing. One way of trying to plot your progress is to carry various objects into the maze and drop them at strategic locations. When you return to the same location the object should be there and you can clarify exactly where you are.

## Objects

There is a golden rule in Adventures — always pick up everything you can. It's a good idea, as most games tend to have an object there for a reason (however obscure) and you can be sure you'll need it sooner or later. The real problem comes with a limit on your inventory as it needs skill to select the best objects to carry into the appropriate locations.

An added problem is the sneaky practice of needing certain objects two or three times — for example a gun or knife. Another little tip here — in some games you may need to repeat an action. For example in *Adventure Land* (The first of Scott Adams' games) you find a lamp and on the command RUB LAMP a genie appears with one set of helpful comments. If you rub the lamp a second time then a second and more helpful response is displayed. But beware a third...

A favourite trick in games is to create a place where it is too dark to see. You will usually find a lamp, a torch, matches or some other object with which to illuminate the scene — often somewhere else in the Adventure. But again there may be a snag — such as a time limit on the light, a draught that blows your match out, batteries that expire. . . .

Most games include an inventory — and a simple hint here is to check your inventory when you start the game as you may find you're already carrying something. So don't wait until you pick up objects before checking.

## Looking around

Some Adventures will let you LOOK around, and this is often necessary to discover objects or clues. Always try a LOOK if your game lets you on arriving at a new location. Other times the command LOOK will just display your location (as in my own game — *Nightmare Planet*).

With experience the computer response often gives you a clue as to what you can expect. Various games contain individual responses as a standard reply, for example I DON'T KNOW HOW TO for a verb that isn't in the vocabulary. You can use this to your advantage when playing. For example if you should come to a door and type OPEN DOOR and get the response I DON'T KNOW HOW TO OPEN then it's obvious the keyword is something else, perhaps UNLOCK or just GO DOOR. However if the response is YOU CAN'T DO THAT . . . . YET then you can be pretty sure OPEN is a recognised word but you need something else before you will be allowed to open the door — perhaps a key.

## Thinking of writing your own Adventure

I suppose we should start really with *why* people want to play Adventure games before thinking about writing one. I have already hinted at some of the reasons — the magic of our fairy-tale youth when fairy queens could grant three wishes and everything became all-right or fearsome giants could strike terror into our vivid nightmares as we lay in the dark. There's something primaeval about the world of fantasy and horror that has a universal appeal to people the world over — after all who doesn't prick up their ears when a good ghost story is being told (especially one reputed to be true).

So Adventure games (rooted in the dark and mysterious) carry this attraction in their very bones and we like to feel a part of it. And herein lies the second (and more important I suspect) fascination — the element of playing a part in the actual story. Role-playing is in its infancy (several books which draw heavily on the *Dungeons and Dragons* theme are already available). But we are still a long way from the science-fiction concept of plugging our minds into a 'dream-recorder' so that we can dream an Adventure of our choice where we become the hero or villain. But we are getting there.

We may not have total involvement yet but the micro-computer has definitely helped the move along in the right direction. With careful programming the player appears to have a variety of decisions to make which will direct the progress of the story — and as it unfolds so the *feeling* of taking part in a scenario is increased.

We all like to be in control if we can — and this is where writing your own game takes on an attractive perspective. YOU can control the story — YOU can decide the environment — YOU create the characters — YOU can make up puzzles and riddles to infuriate your hapless player — YOU are in control. And without a doubt the feeling of achievement in producing a work of art (I may be putting my head in the proverbial noose by calling a computer game a work of art — but it is a creative act which causes enjoyment so I feel justified!) is worth all the effort.

## Making a start

Let's end this chapter on a more practical note. You have decided to have a go at writing your own Adventure — but where do you begin?

If you want to start planning your story straight away I suggest you turn to Chapter 5 as the next section of the book deals with programming concepts. But before you do — it's not a bad idea to have a general plan clear in your mind first.

**Flowchart 1.1** is an imaginary flow chart describing the basics of creating an Adventure.

Block 1 starts by setting various variables (such as the position of the player) and dimensioning the arrays. (If you are a little unclear about arrays then it's not worth worrying at this stage. Any explanation I make such as thinking of them as the pages of a book or a railway timetable will probably only confuse you — and to be honest it's not necessary to understand them to write an Adventure.)

Block 2 is really just a method of printing a display on the screen which will contain the necessary information. This usually consists of a short description of where you are followed by any objects or perils that may be around. Also this stage of the program will check to see if you are in certain locations where special conditions apply (for example you may find when you enter a particular room the door is locked behind you and you cannot escape that way).

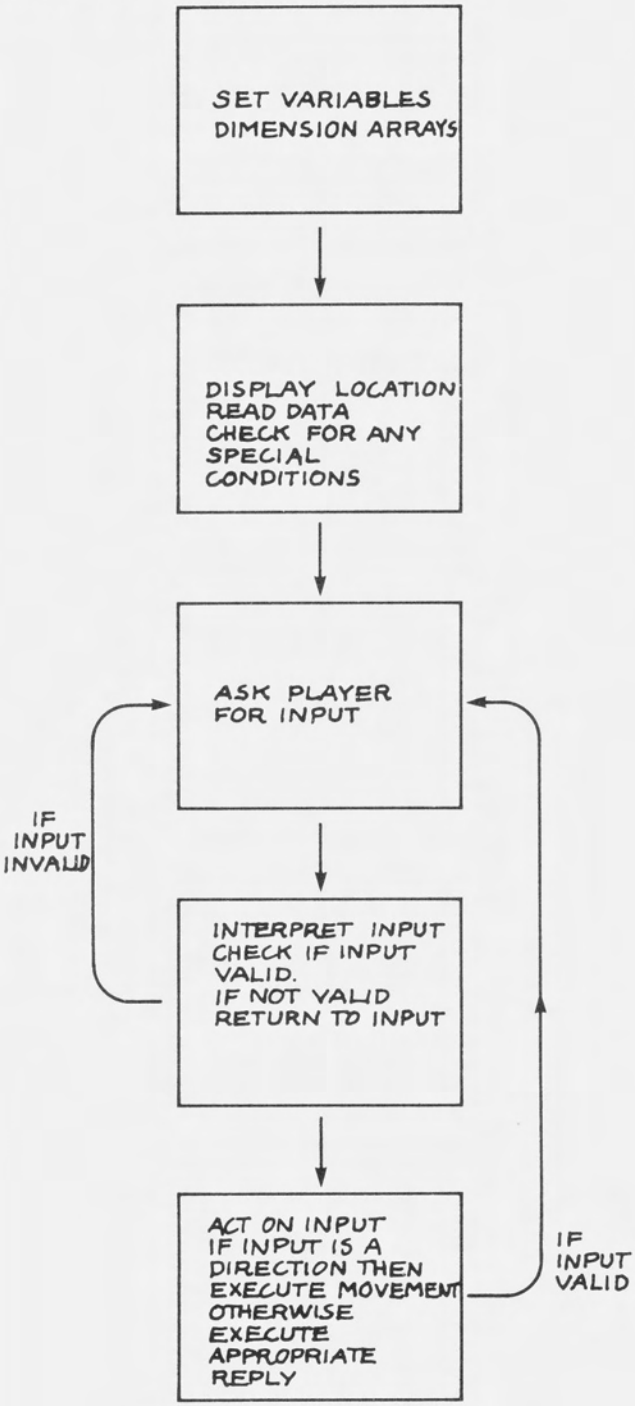
Block 3 asks the player for an input (usually by displaying the message WHAT SHALL I DO NOW?) and then waits for an answer.

Block 4 is a simple check to see if the input is valid. If not the program will give a standard reply (such as I DON'T KNOW HOW TO DO THAT!) and then repeat the input message in Block 3.

Block 5 is the real meat of the program where all the different responses (such as HELP or GO NORTH) are interpreted and then acted upon. The program then loops back to Block 3 to await the next input.

Of course there is a little more to writing a complete game than this —

**Flowchart 1.1 A basic Adventure**



but the essence is there. Now we can get on with the business of serious programming.

Although I have mentioned moving to Chapter 5 for a few ideas on creating your own story — unless you are an experienced programmer the little extra time it will take working through Chapters 2–4 will be well worth it as they contain a straightforward explanation of the basic mechanics of structuring your program. I suspect that Adventures need pre-planning more than most.

# CHAPTER 2
# Moving around an Adventure

We need to understand the basic mechanics of an Adventure Game before starting to look at creating our own masterpiece containing many locations, plenty of perils, objects galore scattered around — and all the other goodies that create the fun (and the frustration) of programming and playing. Before starting to switch on the 64 it's important to have an idea of what the game is to be about. This applies equally well to the simple example I have created to illustrate various procedures as to the marathon game in the second section of the book.

Probably the first most important feature of an interesting game is that you have a variety of locations — and that you can move about them in a consistent manner. To do this you need a map.

## Maps — how to draw and interpret them

Whilst it *is* possible to play the whole of an Adventure without a map almost every game will be more rewarding and easier if you can chart your progress as you go along. In fact one of the easiest ways of starting any game is to move from location to location noting down where you are and which exits lead to new areas. A LOCATION in Adventures can mean any area, for example a room, the top of a tree, inside a lift, outside a window, etc.

In one of Scott Adams' Adventures *The Count*, you wake up in bed. In your room is a window. If you type GO WINDOW (and it is already open) then you move to a new LOCATION on the window ledge. If you then go DOWN (using a method I will not divulge) you move to another LOCATION in a window box which in turn leads into another room in the castle. In each case the window ledge and the window box count as separate LOCATIONS.

Doubtless there are as many different ways of actually drawing a map as there are people to draw them — but the method I have found easiest seems pretty simple so I will use it to illustrate the maps throughout the book. Whenever I am starting a new Adventure I draw my first LOCATION in the centre of the page as a box containing the appropriate word. I always use the centre of the page at this stage as I do not know if I am going to be travelling east, north, south or west yet.

Having drawn the box I note the exits I am allowed (most Adventures do help by telling you which exits are available — but some wait for you to type

**GO NORTH**

before replying

**YOU CANNOT GO THAT WAY**

and then wait for you to try the next direction). Personally I find this latter approach a time-waster (as it serves no useful purpose to keep trying different directions until you hit the lucky one). I then mark arrows (one way only at this stage) going from my LOCATION in the appropriate direction.

**Figure 2.1 Drawing your map 1**

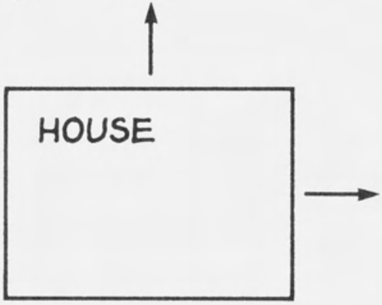


**Figure 2.1** shows the start of an Adventure where I have found myself in a house and have been told I can go NORTH or EAST.

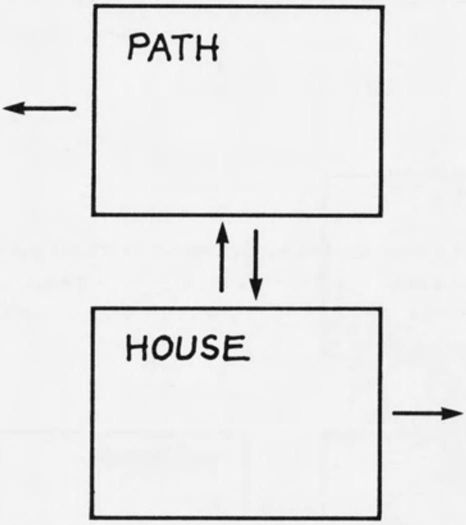
The next stage is to go in one of the possible directions — and then note the new LOCATION. In the imaginary example having keyed in NORTH or just N (as most Adventures will allow you to use only the first letter of each direction) we find ourselves on a path and now the possible exits are WEST or SOUTH (see **Figure 2.2**).

At this stage I can add an arrow going back from the path to the house (in most Adventures if you can go one direction into a new location you can usually move back again — but not always) and a new arrow moving WEST.

This process continues until eventually I will have mapped out most of the obvious LOCATIONS and can now start to move around them.

Not all Adventures have obvious LOCATIONS and in most of them the directions UP and DOWN are added to the points of the compass, making the initial map quite difficult to design as it may have several levels as well. An example of this must be one of the best known Adventures of all — *Adventure Land* by Scott Adams — where the initial map contains only a handful of LOCATIONS. It is only after discovering a certain 'tree' which,

**Figure 2.2 Drawing your map 2**



once climbed, reveals a way DOWN into a maze of passages and caverns that the full extent of the map can be drawn.

As I mentioned earlier one snag with this is that when you start you do not know where the possible LOCATIONS are — so it is possible to find yourself going off the page (see **Figure 2.3**). A little redrawing of the map is all that is required.

## Converting the map to a grid

The essence of all Adventure games is the ability to move around from place to place, and there are several ways of achieving this on a computer. Probably the simplest is to convert your map into a grid and then give the computer the information to allow it to accept certain areas of the grid (the actual locations) as places it can move into. If these areas have a reference number then it is quite easy to persuade the computer to 'move' into that area and display the location.

This is probably easier to understand with a diagram, (see **Figure 2.4**):

In **Figure 2.4(a)** our map is ready for conversion. There are six locations (numbered 1 to 6 for convenience at this stage) starting with a field, then moving along a path to a house. To the east of the house is a forest, then we move south to a plain and finally a lake.

**Figure 2.4(b)** has now placed that map onto a 5 × 5 grid (I could have used 4 × 4 for the purpose of this example, but as I wanted to allow room for you to experiment by adding your own locations I decided to use the 5 × 5). The conversion is a straight placement of the squares — leaving 19 squares blank for the moment. One of the advantages of this method is that it is easy to expand or change your map as you go along.

**Figure 2.3 Drawing your map 3**

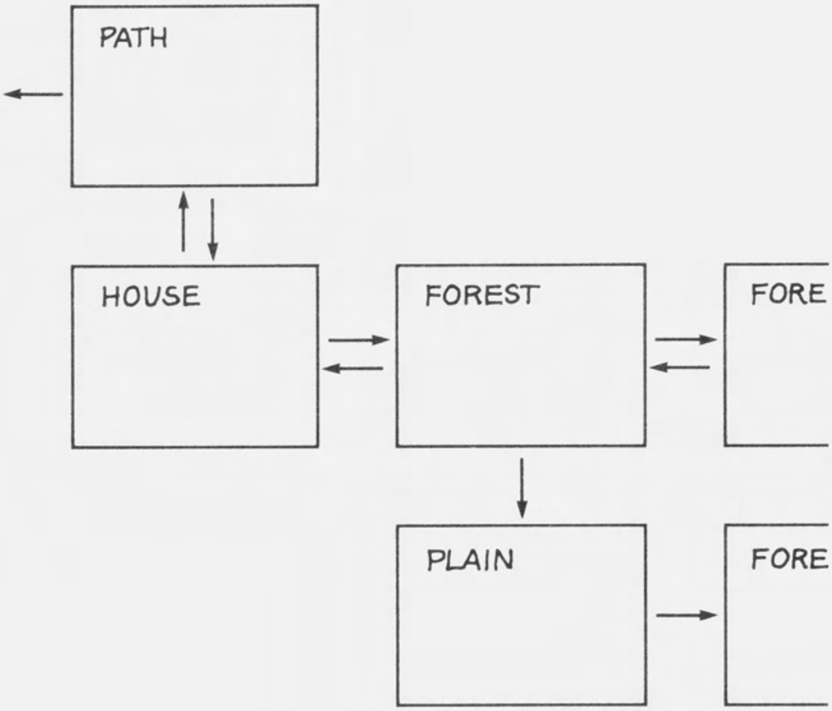


**Figure 2.4a Converting your map to a grid**

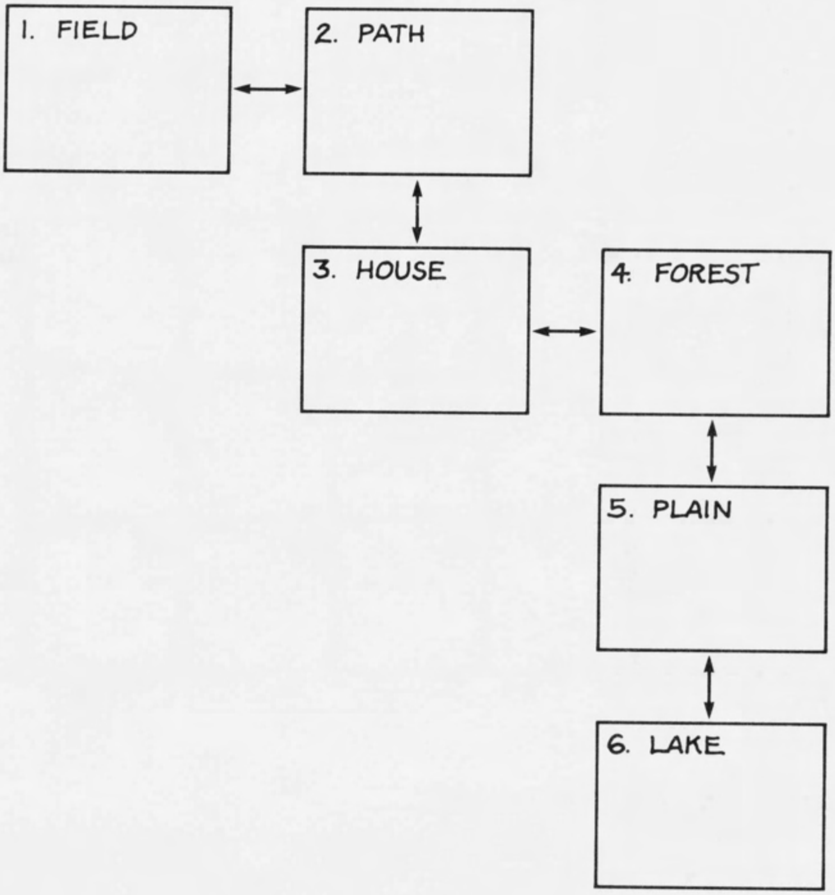
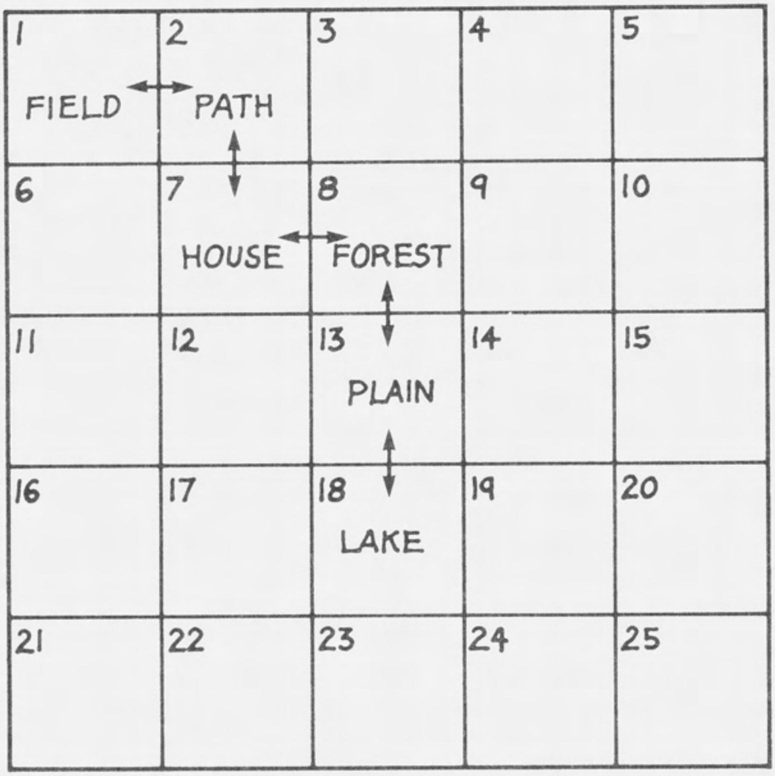


**Figure 2.4b**



## Moving about in the grid

You should have noticed in **Figure 2.4(b)** that the various locations now have a different reference number. For example the house (number 3 in **Figure 2.4(a)**) is now number 7. This allows us to give the computer a simple formula to 'persuade' it to allow us to 'move'.

If we are in Location 7 and we wish to move south then we need to move into Location 13. Similarly if we are in Location 13 south takes us to Location 18. The formula $x = x + 5$ will take us south — if $x = 8$ then south will be $8 + 5$ which $= 13$. Using the same idea going north will be a subtraction – $x = x - 5$ will move from 18 back to 13. East will be $x = x + 1$ whilst west is the reverse, $x = x - 1$.

In a $4 \times 4$ grid the formula would be $x = x + 4$ for south and $x = x - 4$ for north. Obviously east and west remain the same regardless of the grid size.

The last consideration for movement is how to prevent the computer moving into a location that isn't there — for example east or west from the plain (LOCATION 13). The solution is to fool the computer by telling it that LOCATION 14 doesn't exist — by giving that square a value of 0.

This will become clearer once we start the actual programming — but for now **Figure 2.5** should demonstrate how we know exactly what to tell the computer to achieve the process of movement from one location to another.

**Figure 2.5 The basic grid for Adventure Game**

| | | | | |
|---|---|---|---|---|
| 1.<br>FIELD | 2.<br>PATH | O | O | O |
| O | 7.<br>HOUSE | 8.<br>FOREST | O | O |
| O | O | 13.<br>PLAIN | O | O |
| O | O | 18.<br>LAKE | O | O |
| O | O | O | O | O |

## Starting to program your Adventure

Now — at last — we can start programming. Before describing the first module I would like to pass on a few hints about programming in general that I feel are worth mentioning at this stage. Experienced programmers can skip this bit as there is nothing revolutionary here — but I know from my own experience that some of the simplest tricks of the trade are often not mentioned as it may be assumed they are *so* simple that everyone must know about them.

### 1. *Keep saving the program*

This is such old hat that I hesitate to mention it — except that everyone is caught out and when you're in the thick of a program it's only too easy to forget. At one stage in *Nightmare Planet* (the Adventure game in the second section of this book) a thunderstorm crashed my 64 and about an hour's hard work was lost. Another time the disk drive suddenly corrupted my program and also (neat trick this) my backup as well — because I had automatically made the backup without checking. Now I keep a backup of the previous version as a spare to fall back on should this happen.

If the thought of all that SAVEing and VERIFYing irks you — look at Hint 2.

### 2. *A neat time-saver*

If you have a disk drive repeated SAVEing and VERIFYing of your program becomes tedious if you don't know this little gem of a trick. Before I read about this routine I was unaware that you could overwrite your program on disk (as happens automatically on tape) and so I was SAVEing my new version under a different name and then changing the name back after SCRATCHING the original version.

At the start of every program type this little module:

```
1. GOTO 3
2. SAVE "@0:NAME",8:VERIFY"NAME",8:STOP
3. REM
```

If you type the name of your program between the quotes then every time you wish to SAVE the version you just type GOTO 2 and the 64 does the rest. Of all the tips I've picked up this is one of the best and I'm indebted to David Lawrence.

### 3. *Test your modules*

When I first began to copy programs from books and magazines I would sit down in front of my computer and bash away — trying to get as much Basic into the memory as I could in one go. It was rather as though I had an imaginary deadline to meet and couldn't waste a second.

Naturally I made mistakes and then tracking them down became a time-waster par excellence! However if you check each section of the program as often as you can (having saved it to tape or disk) to see if it is working you will pick up these mistakes along the way (which is much less frustrating and saves on time).

4. *Use the screen for editing and copying*

When you start programming from books and magazines it's often easier to just copy line by line and the idea of screen use to save time can be neglected. To understand how best to use this idea look at the beginning of Module 1.1. and you will see that I have used a lot of REM statements and ':' statements to space out the program and give it a polished appearance. If you just type line 100 once and line 101 once then it's easy to change the 101 to 99 (remember to add or delete a '*' if the line number changes from 2 figures to 3 or from 3 to 4) and press RETURN. The new line is placed into memory without retyping all those stars or ':' statements.

This system will also work well in lines 800 to 830 in Module 1.3.

My sample program contains only a few variables — outlined below:

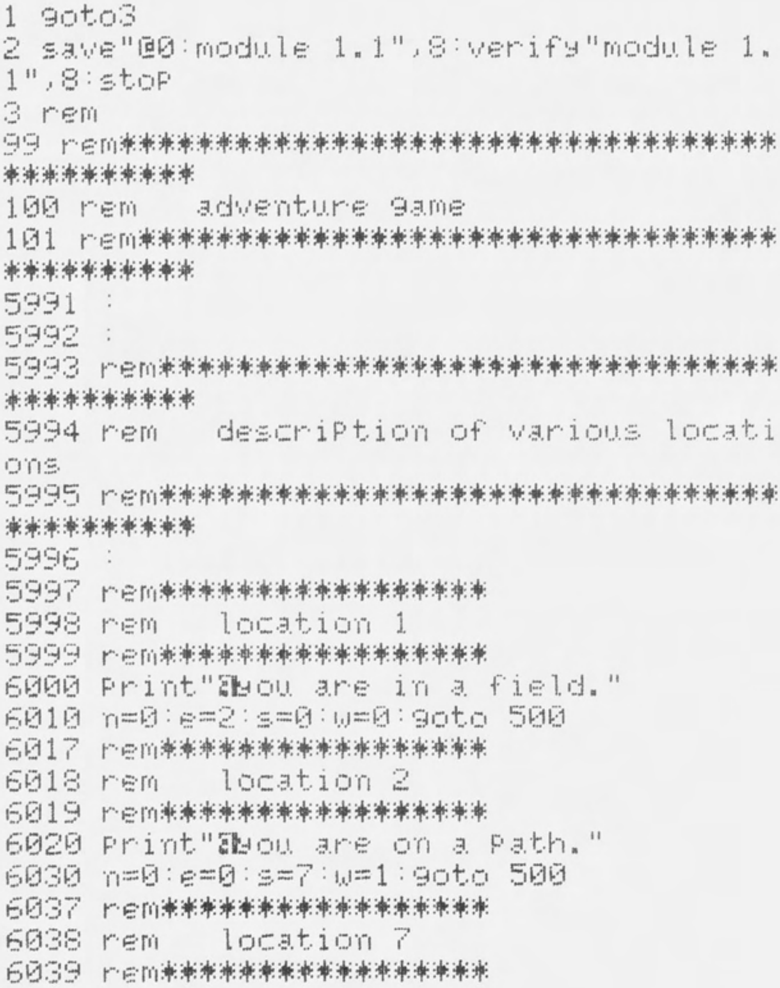
**Table 2.1 Variables used in Adventure Game**

| | |
|---|---|
| P | The position of the current location |
| P2 | The position of the new location |
| OB% | Integer value of Object in Object Array |
| OB$ | Single word description of Object |
| SI$ | Long description of Object for Location |
| I$ | Command expected from Player |
| VE$ | Verb expected in a 2 word command from Player |
| NO$ | Noun expected in a 2 word command from Player |
| SP | Local variable used to split I$ |
| I | Local variable to count objects in the Object Array |
| N | The direction NORTH |
| E | The direction SOUTH |
| S | The direction SOUTH |
| W | The direction WEST |
| SW% | Set at 2 when the swamp has been negotiated |
| MW% | Set at 2 when the monster has been killed |
| IV | Inventory counter |

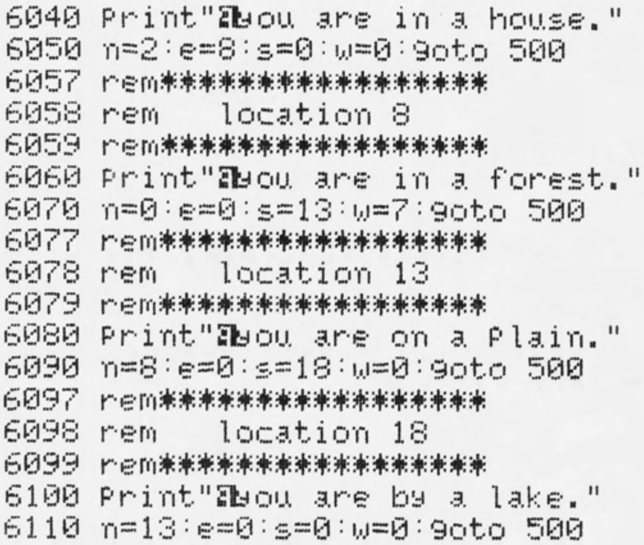
## The location descriptions

Having drawn up the locations on our map we need a display to come up on the screen when we move into each place so that the player knows where he is. Again for this simple example I have used the shortest description possible just to enable you to see how the game works. In the second section of the book this part of the program is perhaps the greatest byte-eater of all as the various descriptive passages used to give an atmosphere to the whole story take up quite a large chunk of memory. However as the feel of your game will rely on your ability to make the player become involved in each location this is memory well spent — and in my opinion a good Adventure game will trade plenty of memory to keep that atmosphere at the expense of items such as extra objects or graphics.

MODULE 1.1



```
1 goto3
2 save"@0:module 1.1",8:verify"module 1.
1",8:stop
3 rem
99 rem**********************************
**********
100 rem   adventure game
101 rem*********************************
**********
5991 :
5992 :
5993 rem********************************
**********
5994 rem   description of various locati
ons
5995 rem********************************
**********
5996 :
5997 rem******************
5998 rem   location 1
5999 rem******************
6000 print"▮you are in a field."
6010 n=0:e=2:s=0:w=0:goto 500
6017 rem******************
6018 rem   location 2
6019 rem******************
6020 print"▮you are on a path."
6030 n=0:e=0:s=7:w=1:goto 500
6037 rem******************
6038 rem   location 7
6039 rem******************
```



```
6040 Print"♥you are in a house."
6050 n=2:e=8:s=0:w=0:goto 500
6057 rem******************
6058 rem   location 8
6059 rem******************
6060 Print"♥you are in a forest."
6070 n=0:e=0:s=13:w=7:goto 500
6077 rem******************
6078 rem   location 13
6079 rem******************
6080 Print"♥you are on a Plain."
6090 n=8:e=0:s=18:w=0:goto 500
6097 rem******************
6098 rem   location 18
6099 rem******************
6100 Print"♥you are by a lake."
6110 n=13:e=0:s=0:w=0:goto 500
```

Lines 99–101: These supply the titles for the program. Please do look at Number 4 in my section on hints before programming this (and all the other REM lines) to save time and effort.

Lines 5991–5992: The colon sign acts as a REM statement and is purely a method of separating the areas in the printout for neatness.

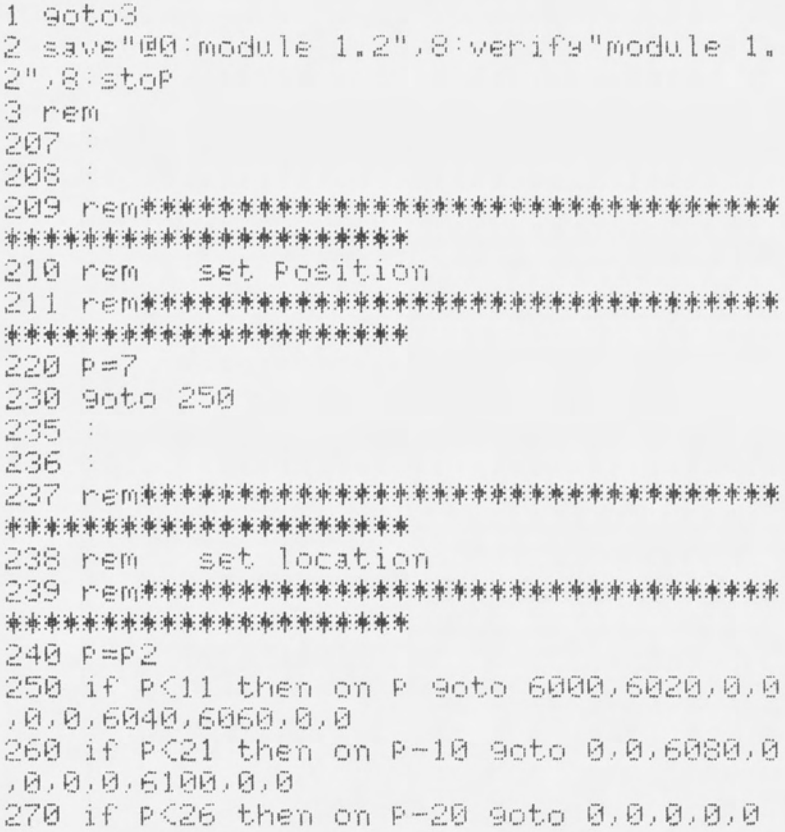
Lines 6000–6010: The first line prints out the description of the location.

Line 6010 is the key to correct movement so I will clarify what is happening. If you look back at **Figure 2.5** you will see that Location 1 is the field. The only exit possible is east to Location 2 (the path). Thus line 6010 states that N = 0 (you cannot go north) that E = 2 (you can go east to Location 2) and both South and West = 0 (you cannot go south or west).

Finally this line tells you to GOTO 500 which will display the possible exits from this location (see Module 1.3).

Lines 6020–6110: The rest of the program completes the different PRINT statements for the different locations and the appropriate instructions for allowing movement into the locations in a north, east, south and west direction.

MODULE 1.2



```
1 goto3
2 save"@0:module 1.2",8:verify"module 1.
2",8:stop
3 rem
207 :
208 :
209 rem*********************************
*********************
210 rem   set position
211 rem*********************************
*********************
220 p=7
230 goto 250
235 :
236 :
237 rem*********************************
*********************
238 rem   set location
239 rem*********************************
*********************
240 p=p2
250 if p<11 then on p goto 6000,6020,0,0
,0,0,6040,6060,0,0
260 if p<21 then on p-10 goto 0,0,6080,0
,0,0,0,6100,0,0
270 if p<26 then on p-20 goto 0,0,0,0,0
```

Lines 220–230: This sets the position that we will arrive in upon RUNning the program. P is the variable POSITION and if we allocate a value of 7 then the first message to appear on the screen will relate to Location 7 — the house.

Of course we could decide to start anywhere in the Adventure — not just in the house. After you type in Lines 240–270 you can experiment by changing P = 7 to P = 13 and then you will start on the plain instead of the house.

Line 240: This line is really a little difficult to explain at this stage — so just type it in and forget about it for the moment. All you need notice is that it introduces another variable — P2 — and then immediately makes it equal to P.

Lines 250–270: These are the lines which tell the 64 how to display the appropriate message on the screen. Using the ON. . GOTO statement as a method of selecting the correct line number the program takes the value of

P (which we have set at 7 in line 220) and starts in line 250. If P is less than 11 (which it is) then it stays in this line and works along until it reaches the 7th number (which happens to be 6040). You then drop right through the program until you reach 6040 and find the line: PRINT "YOU ARE IN A HOUSE". Thus the correct display for P = 7 appears on the screen.

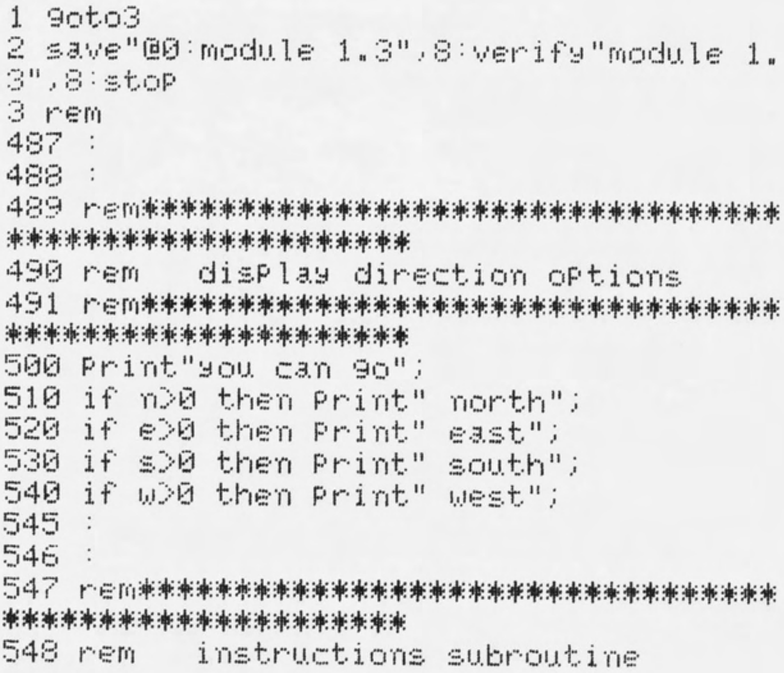
If P had been a number greater than 11 (say 13) then the program drops into line 260 and as P is less than 21 it takes the statement ON P – 10 (13 – 10 which equals 3) GOTO the 3rd number in the line (which happens to be 6080).

If you enter Modules 1.1 and 1.2 and then RUN them you should get the display "YOU ARE IN A HOUSE." on the screen followed by the error message UNDEF'D STATEMENT ERROR IN 6050. This is because you have told the computer to GOTO line 500 which doesn't exist yet.

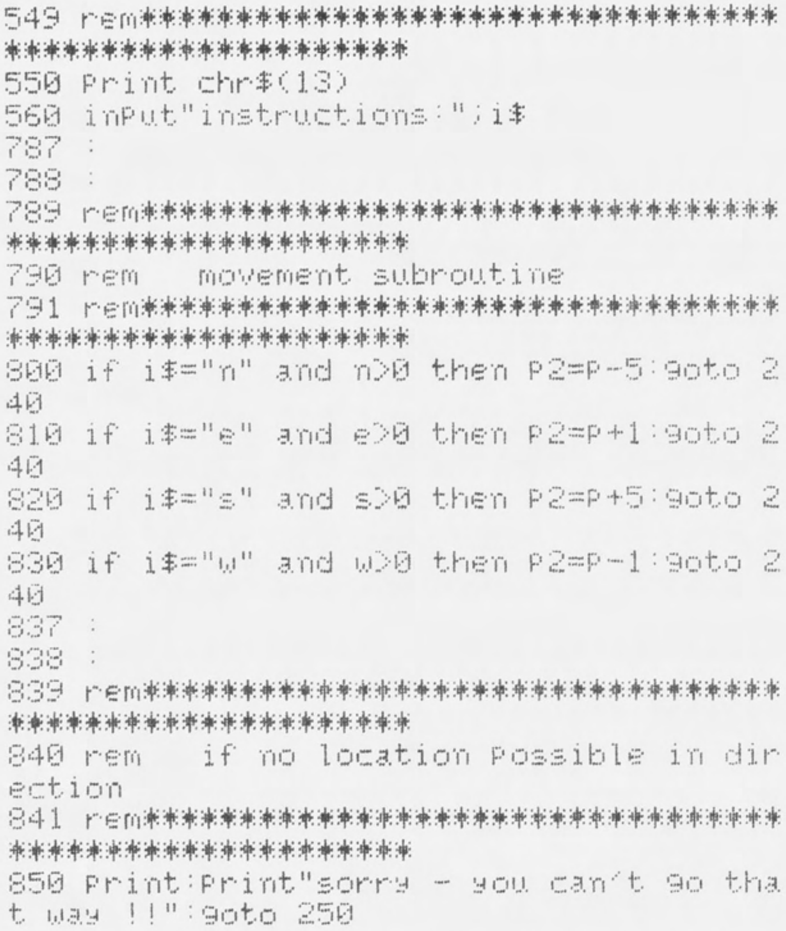
LIST the program again and change line 220 to read P = 18 — then RUN it again. This time you should get the display "YOU ARE BY A LAKE." and then the error message. Before entering the next module change the value of P back to 7.

Having created the locations and told the computer how to get to them we now need to add the element of movement and also give the player an INPUT so that he can tell the 64 which direction to move in.

MODULE 1.3



```
1 goto3
2 save"@0:module 1.3",8:verify"module 1.
3",8:stop
3 rem
487 :
488 :
489 rem*********************************
*********************
490 rem   display direction options
491 rem*********************************
*********************
500 print"you can go";
510 if n>0 then print" north";
520 if e>0 then print" east";
530 if s>0 then print" south";
540 if w>0 then print" west";
545 :
546 :
547 rem*********************************
*********************
548 rem   instructions subroutine
```



```
549 rem*********************************
*********************
550 print chr$(13)
560 input"instructions:";i$
787 :
788 :
789 rem*********************************
*********************
790 rem   movement subroutine
791 rem*********************************
*********************
800 if i$="n" and n>0 then p2=p-5:goto 2
40
810 if i$="e" and e>0 then p2=p+1:goto 2
40
820 if i$="s" and s>0 then p2=p+5:goto 2
40
830 if i$="w" and w>0 then p2=p-1:goto 2
40
837 :
838 :
839 rem*********************************
*********************
840 rem   if no location possible in dir
ection
841 rem*********************************
*********************
850 print:print"sorry - you can't go tha
t way !!":goto 250
```

Lines 500–540: These lines display the possible exits. Thus if N is greater than 0 the 64 will print NORTH and so on. Note the use of the semi-colon at the end of lines 510–540 which prevents the directions being printed on separate lines — and the space at the beginning of each statement in quotes which allows each word to have a space between it.

Lines 550–560: Line 560 is a straightforward INPUT. However if we did not put line 550 first which prints a carriage return then the word ''INSTRUCTIONS ?'' would follow on the end of the line containing the exits (because of the semi-colon in lines 510–540). If you want to see this happen type in line 560 and leave out 550 first time you try it.

Lines 800–830: Here is the real kernel of the whole program — the actual movement. Using the formula I described earlier these lines check I$ and then check if the move is a legal one (in other words if you type in ''N'' and

N = 0 in that location then the computer will reject this line). Providing the move is legal there is a swift calculation changing the value of P to P2 — the new location.

Let's follow this through. If P = 7 (Location 7 — the house) and we type N for north, then in line 800 the 64 checks if N = 0. If you glance at Module 1.1 you will see the variable N has been set to 2 in line 6050 — so the move is a legal one. Now the formula gives the variable P2 the value of P – 5 (7 – 5) which equals 2. Finally the line gets a GOTO instruction to line 240.

Now we can understand why line 240 is here, for before we can move we must change P2 back to P so that the following lines can understand where to 'move' to. Now we have the variable P equalling 2 — and by looking at line 250 we can see that we now GOTO line 6020 — which happens to be the display for Location 2 — the path.

Line 850: The final line in the module. Suppose we were in Location 7 (the house) and we asked the computer to go west. This is an illegal move as in line 830 W = 0. The computer drops to the next line which is a standard response YOU CAN'T GO THAT WAY followed by a return to the "INSTRUCTIONS" input to repeat the request.

If you have typed in the whole program correctly you should find you can move around with complete freedom. It's not exactly an exciting procedure — but it does illustrate the principle of getting from one place to another in your Adventure game.

The flow chart for this part of the program is illustrated in **Flowchart 2.1.**

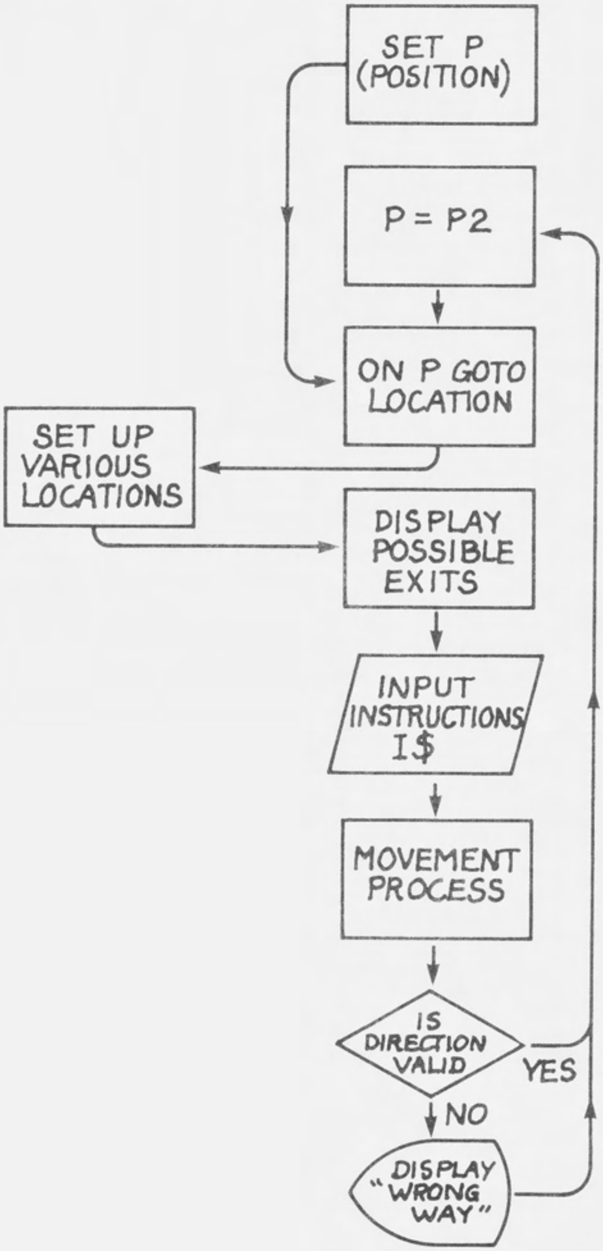
Other methods of movement that can be used involve data statements and arrays. Whilst saving on memory I prefer to use the method I've outlined because of the ease of alteration in a large program. In fact when I was writing *Nightmare Planet*, the Adventure in the second section of the book, I changed several locations and also the direction between locations to improve the plot.

A second consideration with this method is the ability to prevent movement easily — something I shall refer to in detail later. For the moment consider **Figure 2.5** again and imagine that we want to be able to move from Location 2 to Location 1 in one direction only — from east to west. It is a simple matter to alter lines 6010 and 6030 as follows:

```
6010  N=0:E=0:S=0:W=0:GOTO 50
0 (note E=2 in the original program)
6030  N=0:E=0:S=7:W=1:GOTO 500
```

Although this action would trap the player in Location 1 (every exit is now 0) the point is that in a situation where there was another exit it would be possible to pass from the path into the field — but not back out again.

**Flowchart 2.1 Movement in the locations**



This flexibility *is* possible using DATA statements — but quite a lot harder to implement.

The next stage in our Adventure is to start to place some objects into the various locations — and then be able to move them around. As a final note — don't forget to SAVE to tape or disk the program so far as you will be building on this framework in the next two chapters.

# CHAPTER 3
# Objects and Commands

In the last chapter we created five Locations for our Adventure and discovered how to move around between them — and you must have noticed that after entering and using this program you soon find it a little boring.

So let's liven it up a little by placing a few objects into the locations, and then learn how to pick them up, move them around, and drop them again. I will also describe how to collect an Inventory (a record of what objects you are carrying) and start to analyse the various commands a player may input.

## Objects in arrays

As I said before arrays can cause confusion and an understanding of what is going on is not essential to programming your Adventure game — but it helps. The way I visualise arrays is to think of a variable (let us say X) which we know is going to change its value during the course of the program. However we want to assign certain values to X in a sequence rather than in a random fashion — and also we want to know at any time what each value represents.

To do this we can write X as X(1) where 1 is the first value of X, X(2) where 2 is the second value, and so on. The figure in brackets is known as the subscript of X.

**Figure 3.1 A simple array**

| | |
|---|---|
| X(1) | |
| X(2) | |
| X(3) | |
| X(4) | |
| X(5) | |

**Figure 3.1** clarifies this. We can think of the array as consisting of a number of boxes under the same main heading of X — where X(1) is the first box, X(2) is the second, and so on. (A point to note here is that the 64 will number

the first box 0 if you don't instruct it otherwise — eg Box 1 is X(0), Box 2 is X(1) and so on. I find this confusing so I tend to start my own arrays with X(1) for simplicity).

Having created our array we can then assign values to our variable X and the result can be seen in Figure 3.2.

**Figure 3.2 A simple array containing values for X**

| | |
|---|---|
| X(1) | 3 |
| X(2) | 14 |
| X(3) | 22 |
| X(4) | 38 |
| X(5) | 73 |

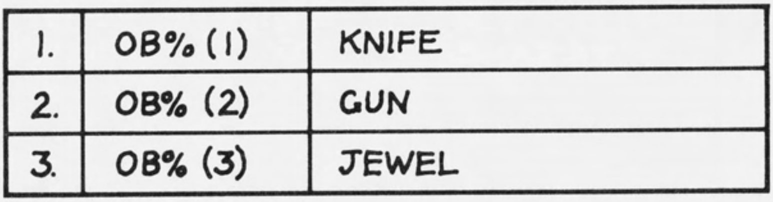
Now let us turn to our program Adventure Game. We need to decide what objects to place into the game — and in which Locations to place them. **Figure 3.3** is the grid again that we used for our map and I have put a "KNIFE" in Location 1 (the field): a "GUN" in Location 7 (the house) and a "JEWEL" in Location 13 (the plain).

**Figure 3.3 The grid for Adventure Game containing objects**

| | | | | |
|---|---|---|---|---|
| 1.<br>FIELD<br>KNIFE | 2.<br>PATH | O | O | O |
| O | 7.<br>HOUSE<br>GUN | 8.<br>FOREST | O | O |
| O | O | 13.<br>PLAIN<br>JEWEL | O | O |
| O | O | 18.<br>LAKE | O | O |
| O | O | O | O | O |

We can set up the array for these three objects as in **Figure 3.4**.

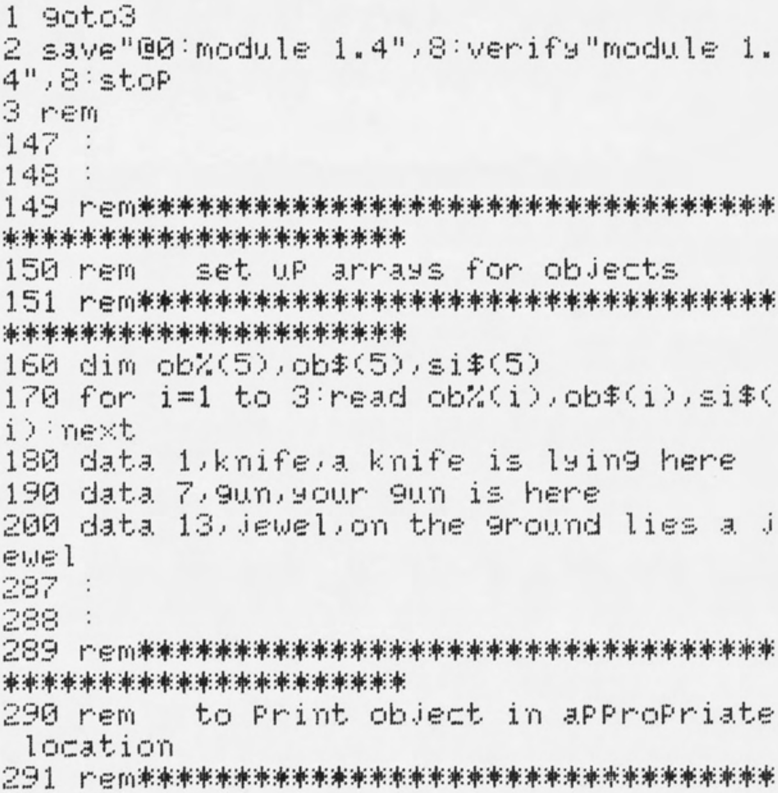
**Figure 3.4 The array of objects in Adventure Game**



| | | |
|---|---|---|
| 1. | OB% (1) | KNIFE |
| 2. | OB% (2) | GUN |
| 3. | OB% (3) | JEWEL |

Note that I have assigned the objects the variable name OB% (the % signifies an integer value — ie a whole number). Thus when we talk about OB%(1) we are talking about the KNIFE whilst OB%(2) is the GUN and OB%(3) is the JEWEL. Now we can 'talk' to the 64 about these objects and manipulate them around our locations — so when we type in the word KNIFE it is understood by the computer as OB%(1).

MODULE 1.4



```
1 goto3
2 save"@0:module 1.4",8:verify"module 1.
4",8:stop
3 rem
147 :
148 :
149 rem*********************************
**********************
150 rem   set up arrays for objects
151 rem*********************************
**********************
160 dim ob%(5),ob$(5),si$(5)
170 for i=1 to 3:read ob%(i),ob$(i),si$(
i):next
180 data 1,knife,a knife is lying here
190 data 7,gun,your gun is here
200 data 13,jewel,on the ground lies a j
ewel
287 :
288 :
289 rem*********************************
**********************
290 rem   to print object in appropriate
 location
291 rem*********************************
```

```
**********************
300 for i=1 to 3:if ob%(i)=p then print
si$(i)
310 next
```

Before starting to program again please note that the modules in this chapter are supposed to be added to the modules in the last chapter and will not work if keyed in on their own.

Module 1.4 sets up the arrays for our example and then prints the appropriate display on the screen.

Line 160: This is the line which sets up the arrays in this program. DIM stands for DIMension and is the instruction to the computer telling it you are going to set up your arrays. Strictly speaking you don't need to DIM in this example as the 64 can handle arrays up to 11 (that means a DIM of (10) as you start with a 0 remember) so you would only need to use it for DIM X(100) say or DIM X(30) — but I might well be adding some objects later on in the larger Adventure so it's best to put a DIM statement in.

OB%(5) gives us five possible values for the objects — OB$(5) gives us five names — and SI$(5) gives us five descriptions. We will only be using three for the moment and lines 180–200 are the DATA statements that go with this line.

Line 170: A standard line telling the program to READ the data one item at a time, and linking each string with the integer value. So when the 64 encounters OB%(1) it will either print out "KNIFE" or "A KNIFE IS LYING HERE".

Lines 170–200: Often data is placed at the beginning or end of the program but for clarity I have placed it here. Two points to note are:

1. The integer value is the same number as the Location Number.
2. The actual words in SI$ are quite important and should be used with care.

I can illustrate this last point by changing the value of SI$ for OB%(2) — the 'gun' — from "YOUR GUN IS HERE" to "YOUR GUN IS LYING ON THE TABLE". Now this second phrase is fine on first playing the game — for on entering Location 7 you will see the following display on the screen:

**YOU ARE IN A HOUSE**
**YOUR GUN IS LYING ON THE TABLE**

But suppose you pick up the gun and take it to the forest — then have to drop it there for some reason. On returning to the forest at a later date you would see the display:

**YOU ARE IN A FOREST**
**YOUR GUN IS LYING ON THE TABLE**

This is obviously ridiculous and will make your game look amateurish — which is why we have to write the rather more bland but acceptable "YOUR GUN IS HERE".

Lines 300–310: These lines will actually display the string SI$ under the Location description providing the object is in that Location. When an object is in a particular Location then OB%(I) is given the value of that Location. If you look at Lines 180–200 you will see this.

Thus (every time) the Location changes the 64 will scan the array OB% starting at OB%(1) and working through to OB%(3) and if any of these numbers are the same as the Location number (which we set as P) then it will PRINT SI$ for that number. This gives the appearance that the object is in that Location.

The NEXT in line 310 sends the program back to look for OB%(2) and OB%(3) and if they are in the Location as well they will also be displayed.

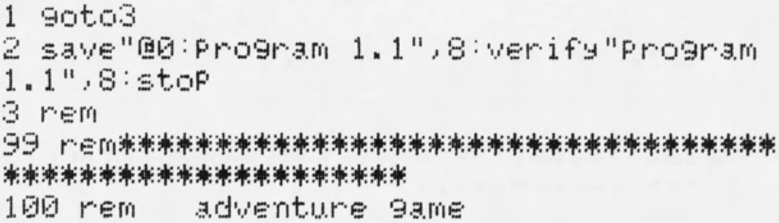
**Important note**

When you first keyed in the game in Chapter 2 all the lines describing the exits for the Locations (Lines 6010, 6030, 6050, 6070, 6090 and 6110) ended in GOTO 500. You must now change these from GOTO 500 to GOTO 300 as we are directing the program to this Section of Module 1.4 to allow the appropriate object string to be displayed on the screen. In the last chapter we had not yet written this section and so had to direct the program to Line 500 instead.

Please alter the value after the GOTO from 500 to 300 (now) to allow the program to work when you test it.

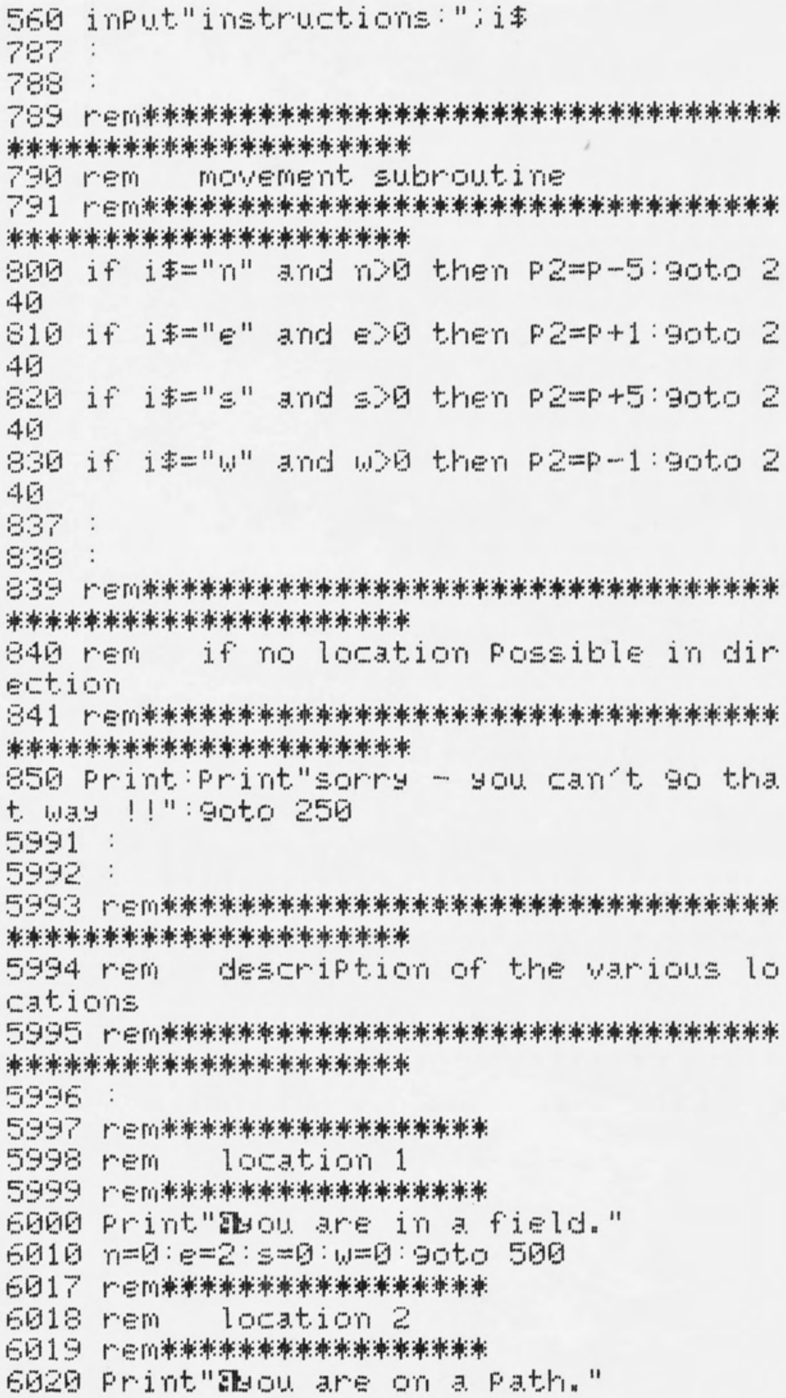
If you have keyed in your program correctly so far it should look like **Program 1.1** and if you RUN it now you should be able to move around your Locations and see the knife, gun and jewel displayed in the appropriate sites.

**Program 1.1**



```
1 goto3
2 save"@0:program 1.1",8:verify"program
1.1",8:stop
3 rem
99 rem**********************************
*********************
100 rem   adventure game
```

```
101 rem*********************************
*********************
207 :
208 :
209 rem*********************************
*********************
210 rem   set position
211 rem*********************************
*********************
220 p=7
230 goto 250
235 :
236 :
237 rem*********************************
*********************
238 rem   set location
239 rem*********************************
*********************
240 p=p2
250 if p<11 then on p goto 6000,6020,0,0
,0,0,6040,6060,0,0
260 if p<21 then on p-10 goto 0,0,6080,0
,0,0,0,6100,0,0
270 if p<26 then on p-20 goto 0,0,0,0,0
487 :
488 :
489 rem*********************************
*********************
490 rem   display direction options
491 rem*********************************
*********************
500 print"you can go";
510 if n>0 then print" north";
520 if e>0 then print" east";
530 if s>0 then print" south";
540 if w>0 then print" west";
545 :
546 :
547 rem*********************************
*********************
548 rem   instructions subroutine
549 rem*********************************
*********************
550 print chr$(13)
```



```
560 input"instructions:";i$
787 :
788 :
789 rem*********************************
*********************
790 rem   movement subroutine
791 rem*********************************
*********************
800 if i$="n" and n>0 then p2=p-5:goto 2
40
810 if i$="e" and e>0 then p2=p+1:goto 2
40
820 if i$="s" and s>0 then p2=p+5:goto 2
40
830 if i$="w" and w>0 then p2=p-1:goto 2
40
837 :
838 :
839 rem*********************************
*********************
840 rem   if no location possible in dir
ection
841 rem*********************************
*********************
850 print:print"sorry - you can't go tha
t way !!":goto 250
5991 :
5992 :
5993 rem********************************
*********************
5994 rem   description of the various lo
cations
5995 rem********************************
*********************
5996 :
5997 rem*****************
5998 rem   location 1
5999 rem*****************
6000 print"█you are in a field."
6010 n=0:e=2:s=0:w=0:goto 500
6017 rem*****************
6018 rem   location 2
6019 rem*****************
6020 print"█you are on a path."
```

```
6030 n=0:e=0:s=7:w=1:goto 500
6037 rem******************
6038 rem   location 7
6039 rem******************
6040 print"♥you are in a house."
6050 n=2:e=8:s=0:w=0:goto 500
6057 rem******************
6058 rem   location 8
6059 rem******************
6060 print"♥you are in a forest."
6070 n=0:e=0:s=13:w=7:goto 500
6077 rem******************
6078 rem   location 13
6079 rem******************
6080 print"♥you are on a plain."
6090 n=8:e=0:s=18:w=0:goto 500
6097 rem******************
6098 rem   location 18
6099 rem******************
6100 print"♥you are by a lake."
6110 n=13:e=0:s=0:w=0:goto 500
```

## Basic string handling

At this stage we need to begin to understand a little of how the 64 can manipulate strings of text because we shall be dealing with sentences and words — and most important of all we need to know when the player gives us a poor or invalid response.

String handling can be quite confusing to the beginner — so I will confine myself to the commands that relate specifically to the Adventure game. If you want to find out more then either Commodore's own *Introduction to BASIC* or one of the otl er books or articles around should help you expand on your knowledge.

MODULE 1.5

```
1 goto3
2 save"@0:module 1.5",8:verify"module 1.
5",8:stop
3 rem
687 :
688 :
689 rem************************************
```

```
**********************
690 rem   check single letter directions
 for i$
691 rem*********************************
**********************
700 if i$="n"ori$="e"ori$="s"ori$="w" th
en 800
715 goto 950
937 :
938 :
939 rem*********************************
**********************
940 rem   check single letter command fo
r i$
941 rem*********************************
**********************
950 if i$="i" then 2000
960 if i$="h" then 2100
987 :
988 :
989 rem*********************************
**********************
990 rem   subroutine to check for two wo
rds
991 rem*********************************
**********************
1000 for i=1 to len(i$)
1010 if mid$(i$,i,1)=" " then 1100
1020 next
1030 print"please can you use two words"
:goto 550
```

Lines 700–710: We have already established that the 64 will respond to the commands N, E, S or W (lines 800–830 in Module 1.3). These two lines will point Direction Commands (north, east, etc) to the Movement Routine in Lines 800–830 and all other commands to the next section in this module (lines 950–960) where they will be checked as valid commands.

Lines 950–960: These just take you to the INVENTORY or HELP sections of the program further down.

Lines 1000–1030: We have exhausted our possible single word or single letter commands — and so we must check if the player has input 2 words or not. This little routine checks the input for a space and if it doesn't find one it PRINTS the message "PLEASE USE TWO WORDS.".

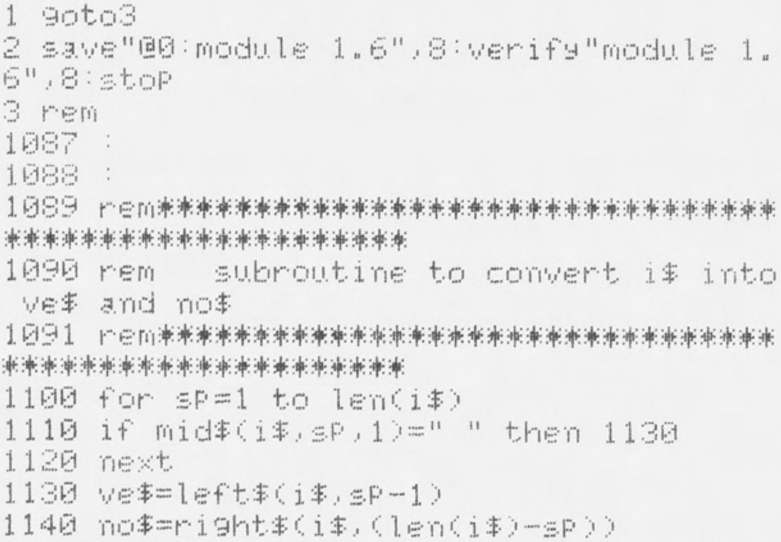
Line 1000 scans the whole of the input I$ as follows: LEN(I$) is the total length of the string given as a number (eg GO would be the number 2 whilst GO NORTH would be the number 8 as it counts the space as one number). Thus if the input is GO NORTH this line becomes FOR I = 1 TO 8 and the 64 will start at 1 (the first letter which is "G" in this case) and work through the string to 8 (the last letter which is "H").

Line 1010 looks for a " " (space) as the string is being scanned. MID$ means it looks in the 'middle' of the string — starting at position I (which starts at 1 and works through to 8) and then looks for a single character (because of the figure 1 in the command MID$(I$,SP,1)). If it finds a single space then it drops through the program to Line 1110. If it goes right through the string without finding a space then Line 1030 PRINTS out the appropriate message "PLEASE USE TWO WORDS" and the program loops back to Line 550 to wait for another input.

This can be illustrated by the simple flowchart in **Flowchart 3.1**.

The next module contains the most complex-looking commands concerned in string-handling — but as with other sections it isn't really as hard as it looks.

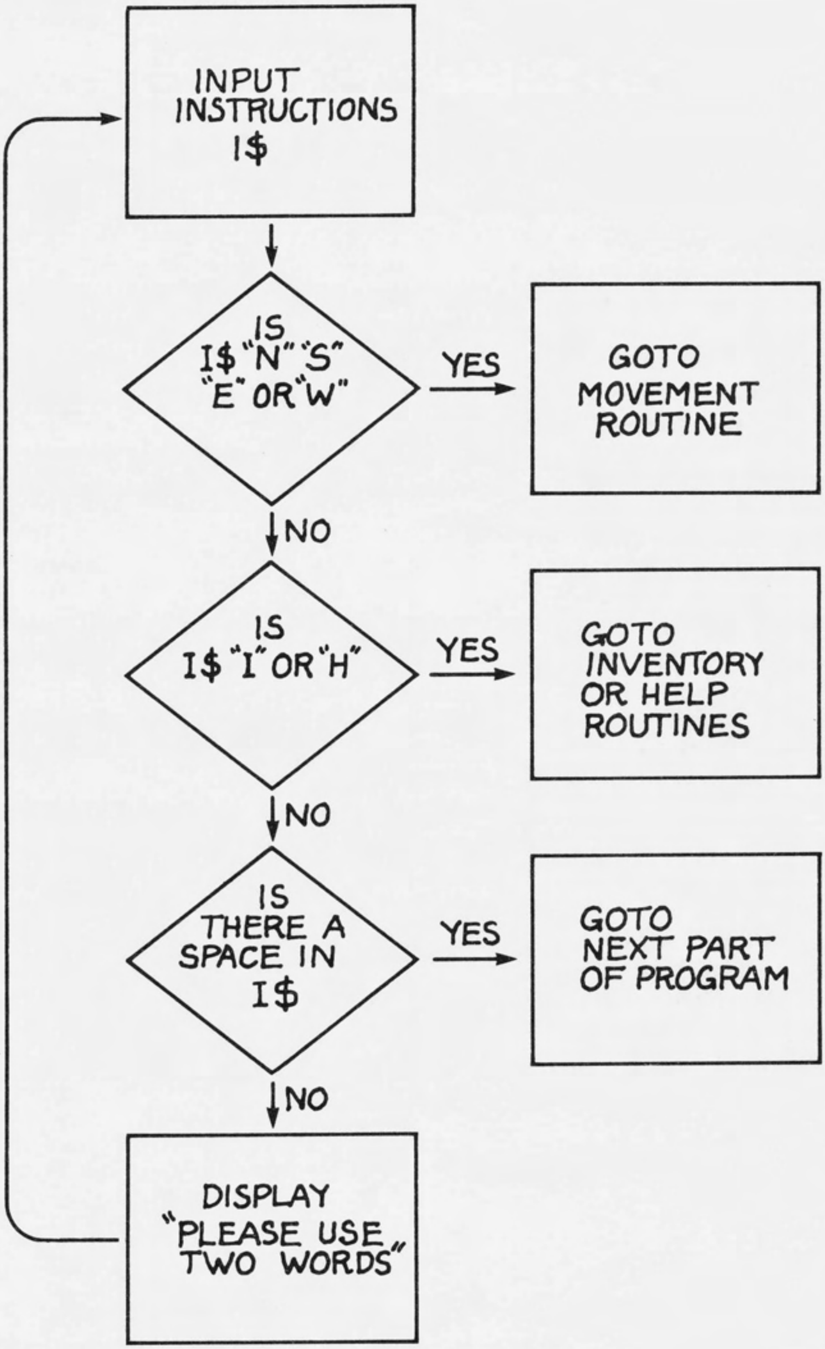
MODULE 1.6



```
1 goto3
2 save"@0:module 1.6",8:verify"module 1.
6",8:stop
3 rem
1087 :
1088 :
1089 rem********************************
*********************
1090 rem   subroutine to convert i$ into
 ve$ and no$
1091 rem********************************
*********************
1100 for sp=1 to len(i$)
1110 if mid$(i$,sp,1)=" " then 1130
1120 next
1130 ve$=left$(i$,sp-1)
1140 no$=right$(i$,(len(i$)-sp))
```

Lines 1100 – 1120: These lines scan the length of I$ again (as did Line 1000 in Module 1.5) only this time the variable SP (for SPACE) is used. We know there will be a space as the loop in Lines 1000 – 1020 either found one or sent

**Flowchart 3.1 Basic string handling**



the program back to the input INSTRUCTIONS again. Now we want to find out where that space is and call the part of I$ in front of the space VE$ (VERB) and the part of I$ behind the space NO$ (NOUN).

Line 1130: This is the line that identifies the part of I$ before the space. It is fairly easy to work out using an example. If the command I$ is GO NORTH then the space will be at number 3 (three units along). Thus VE$ is the LEFT part of I$ that comprises the first two letters (SP – 1 is 3 – 1 which = 2) which is GO.

Line 1140: Now for the rest of I$. We take the RIGHT$ function of I$ (the right section) and split it — but this time we use the full length of the string (LEN(I$)) and subtract the space. So using GO NORTH again LEN(I$) is 8 (seven letters plus one space) and NO$ is the right part of I$ starting at the 8–3) 5th letter — which is the N or NORTH.

Thus we have split GO NORTH into VE$ of GO and NO$ of NORTH. It may seem a little complicated — but we need to be able to recognise both VE$ and NO$ later in the program.

*Testing the program*

If you RUN the program now you should find it looks pretty much the same as it did after you had added the objects into their Locations at the beginning of this chapter.

We can test the string-handling modules by typing any instruction which the program will not recognise. Try typing in FLY or LEAP and you should get the response "PLEASE USE TWO WORDS".

To test Module 1.6 we need to type in any two words (try EAT CHEESE) and then come out of the program by pressing RUN/STOP and RESTORE. Now type in direct mode PRINT VE$ and the computer should print "EAT". Now, still in direct mode type in PRINT NO$ and this time "CHEESE" should appear on the screen. If it doesn't then check carefully to see if you have made any mistakes before moving on to the next module.

Try typing I or H and you'll get an error message, but we are about to correct all that by adding the INVENTORY and HELP routines. More exciting though are the GET and DROP routines which will finally enable us to pick up that knife or that gun and carry it around.

## Adding and taking away

MODULE 1.7

```
1 goto3
2 save"@0:module 1.7",8:verify"module 1.
7",8:stop
```

```
3 rem
1187 :
1188 :
1189 rem************************************
**********************
1190 rem   subroutine to scan for variou
s ve$ commands
1191 rem************************************
**********************
1200 ifve$="get"orve$="grab"orve$="take"
orve$="carry"then 2200
1210 ifve$="drop"orve$="lose"orve$="leav
e"then 2300
1220 ifve$="kill"then 2500
1230 print:print"i do not know how to ";
ve$:goto 550
1987 :
1988 :
1989 rem************************************
**********************
1990 rem    inventory section
1991 rem************************************
**********************
2000 print"your inventory is:":iv=0
2010 for i=1 to 3
2020 if ob%(i)=-1 then print ob$(i):iv=i
v+1
2030 next
2040 if iv=0 then print"nothing"
2050 goto 500
2087 :
2088 :
2089 rem************************************
**********************
2090 rem    help section
2091 rem************************************
**********************
2100 if p=18 then print"you could try ki
lling it":goto 550
2110 print"not much help here i´m afraid
":goto 550
2187 :
2188 :
2189 rem************************************
```

```
**********************
2190 rem     get subroutine
2191 rem********************************
**********************
2200 for i=1 to 3
2210 if ob$(i)=no$ then 2230
2220 next
2230 ifob%(i)=-1 then print"you've got i
t":goto 500
2250 ifob%(i)<>p thenprint"it isn't here
":goto 500
2260 print"ok":ob%(i)=-1
2270 goto 500
2287 :
2288 :
2289 rem********************************
**********************
2290 rem     drop subroutine
2291 rem********************************
**********************
2300 for i=1 to 3
2310 if ob$(i)=no$ then 2330
2320 next
2330 ifob%(i)<>-1 then print"you haven't
 got it":goto 500
2340 print"ok":ob%(i)=p
2350 goto 500
```

Lines 1200–1230: Having split I$ we now need to be able to recognise certain likely commands and act accordingly. For this example program I have just selected a very few but in *Nightmare Planet* the number of possible verbs became so large that I found I hadn't left enough space and had to do a little renumbering.

Line 1230 is just a standard response if the program encounters a command it doesn't recognise before returning to the Instructions input — so if you typed in CLIMB the response would be "I DO NOT KNOW HOW TO CLIMB".

Line 2000: If you had typed in I as a response then the program would go to this line and PRINT the display "YOUR INVENTORY IS ": then the variable IV is set to 0.

Lines 2010–2030: This small loop scans the array for the object data. If OB%(I) is true (in other words – 1) then the 64 will print OB$ (the name of the object). Every time a new object is added IV is increased by 1. This is of more value in a program where you might wish to restrict the number of objects the player can carry at any one time (I have written just such a routine in *Nightmare Planet*).

Lines 2040–2050: Line 2020 used the IF statement to check if OB% was true (= – 1). If not then the program drops to Line 2040 and prints "NOTHING" IF IV = 0. Then it loops back to the display for exits in Line 500.

Line 2110: At this stage there is no help required — so this line just gives the "H" command a place to go.

Lines 2200–2260: The GET routine follows the same procedure as the INVENTORY routine except that it checks if NO$ is the same word as the object in the Location (OB$) and then either informs you that you already have it if OB$ is already in the Inventory or that the object isn't there if OB%< > P. In Line 2250 the condition OB%(I) = – 1 puts the object into the inventory to be read in Line 2020.

Lines 2300–2350: Again the ˬame routine, except this time Line 2340 replaces the object into the new position P.

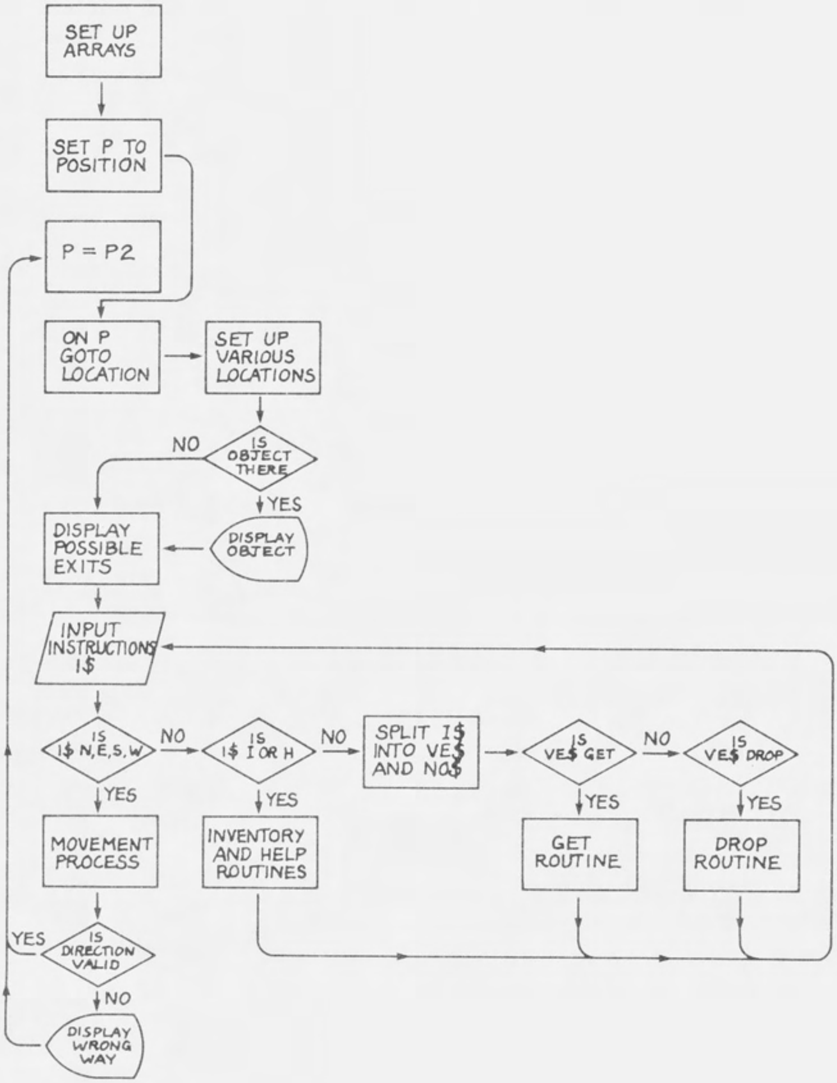
If you now RUN this program (don't forget to SAVE to tape first) you should find that you can 'get' the objects and if you then return to that location they will appear to have gone. If you now 'drop' them somewhere else then upon leaving the location and returning the object will be displayed. Finally if you check your inventory having picked something up it should be in there.

There's quite a sense of achievement having got this far — because now some of the possibilities of what you can do with your program should be opening up. I recommend you spend a short time playing around with the listing, adding other objects in other locations by changing the DATA statements in Lines 180–200. You can add more objects by changing the FOR I = 1 TO 3 and if you have more than five objects the DIM statements as well.

For those of you who like flow charts I have included mine at this stage in **Flowchart 3.2**. It looks a little complicated now, but is the same chart as in **Figure 2.6** with the contents of this chapter added.

In the next chapter we will start to add a new dimension by introducing a couple of 'perils' for our player.

**Flowchart 3.2 Adventure Game including objects**



# CHAPTER 4
# Pitfalls and Perils

The story of Romeo and Juliet is hardly new — boy meets girl, they fall in love, parents will not let them marry, they run off together, and then through misunderstandings tragedy strikes. What turns this story into one of universal appeal and has led to so many variations is the amount of obstacles and problems the hero and heroine encounter in their brief relationship.

It is the same in all of life — the spice we seek often comes when things start going wrong — especially in our fiction and fantasy. And Adventure games are no exception!

We need to start thinking about the various problems and perils that our Adventurer is going to meet on his travels around our Adventure. After all, wandering around a number of locations just picking things up and dropping them again is hardly awe-inspiring and it is the solving of riddles and escaping from perils that makes playing Adventures so appealing and frustrating.

Perils come in all shapes and sizes (and a variety of colours as well). Some are fairly straightforward — like a monster coming at you — a Sorcerer casting a spell on you — falling down a hole — and so on. Some are a little more subtle — a riddle you must answer to achieve your prize — a maze that has trapped you and cannot find the exit — and countless others.

Let's stick with the obvious and discover how to fit a few into our example program.

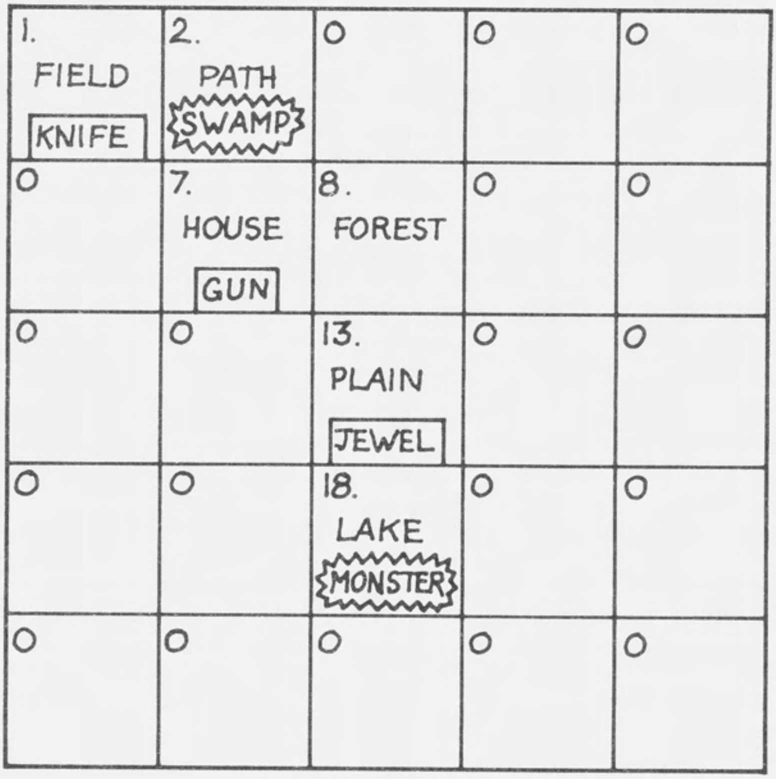
## Thinking of your problems

Like most programming the most difficult part is thinking problems up in the first place — which is why I am sticking with the obvious for the moment. When I was creating my example program I decided to liven it up by introducing a swamp (or quicksand) into Location 2 (the path) and a monster in Location 18 (the lake). These can be added to our map as in **Figure 4.1.**

Having got the perils into the plan I needed a solution as well and so I decided that to escape from the monster you would need to have 'picked up' the gun (in other words it would be in the INVENTORY) and if you had the jewel in Location 13 then its magic would protect you from the swamp. Pretty mundane stuff — but it is only an example to demonstrate the principle.

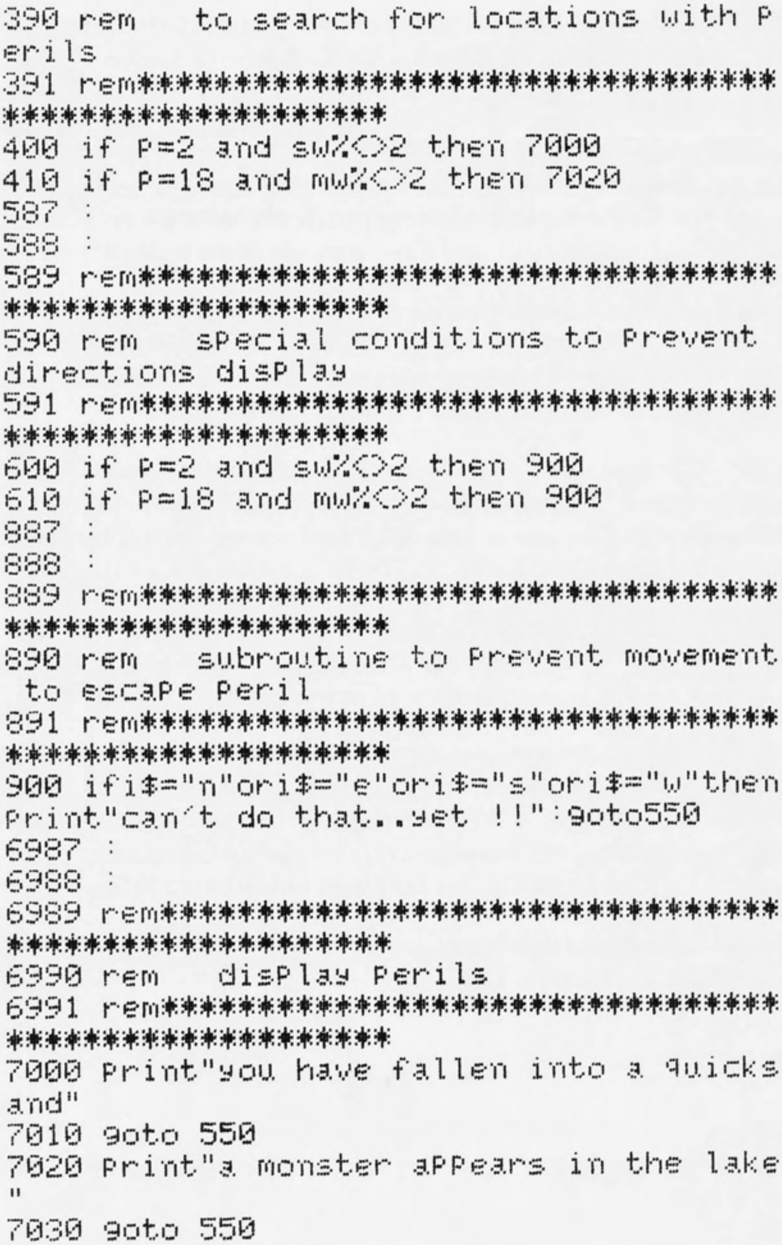
So let's go back to the keyboard to see how we deal with adding and escaping from perils.

**Figure 4.1 Grid for Adventure Game with perils**



MODULE 1.8

```
1 goto3
2 save"@0:module 1.8",8:verify"module 1.
8",8:stop
3 rem
107 :
108 :
109 rem*********************************
********************
110 rem   set up variables
111 rem*********************************
********************
120 sw%=0:mw%=0
387 :
388 :
389 rem*********************************
********************
```



```
390 rem    to search for locations with p
erils
391 rem*********************************
*********************
400 if p=2 and sw%<>2 then 7000
410 if p=18 and mw%<>2 then 7020
587 :
588 :
589 rem*********************************
*********************
590 rem    special conditions to prevent
directions display
591 rem*********************************
*********************
600 if p=2 and sw%<>2 then 900
610 if p=18 and mw%<>2 then 900
887 :
888 :
889 rem*********************************
*********************
890 rem    subroutine to prevent movement
 to escape peril
891 rem*********************************
*********************
900 ifi$="n"ori$="e"ori$="s"ori$="w"then
print"can't do that..yet !!":goto550
6987 :
6988 :
6989 rem********************************
*********************
6990 rem    display perils
6991 rem********************************
*********************
7000 print"you have fallen into a quicks
and"
7010 goto 550
7020 print"a monster appears in the lake
"
7030 goto 550
```

This module is really a number of tiny modules strung together which will fit in between the other parts of the program that you have already entered and saved to tape.

Line 120: These two variables are the key to telling the 64 what to do when it comes to a special condition (in other words a peril or similar unusual situation). By giving SW% and MW% (standing for Swamp and Monster) an arbitrary value of 2 once the peril has been defeated I can 'switch on' or 'switch off' the parts of the program that refer to these special conditions.

Thus we start by giving them a value of 0 (it could have been anything of course) so that if the computer meets the peril it will 'switch it on'. (If you find this a little confusing just read on — it should become clearer).

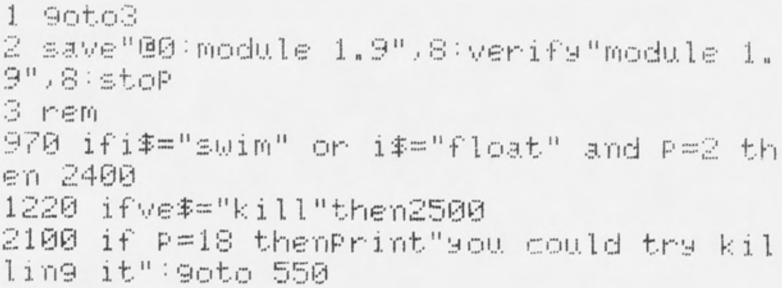
Lines 400–410: Now we actually meet the peril. If the program is in Location 2 or Location 18 then it looks to see if SW% in Location 2 and MW% in Location 18 are set at 2. However we have just set them to 0 in Line 120 — so the program jumps to Lines 7000 or 7020 respectively.

Lines 600–610: Now that we have placed the player into a hazard we must prevent him from moving out of the Location by just typing N or S (or else it would be rather easy to escape). Line 600 is for Location 2 and notes that if SW% does *not* equal 2 (and it will equal 0 until you have defeated the problem) then you bypass the movement routine and GOTO Line 900. Line 610 does exactly the same thing for Location 18. Once you *do* defeat the peril then the variable SW% will be set to 2 — resulting in this part of the program being ignored and the usual procedure of movement being allowed again.

Line 900: The program checks to see if the player did try and escape from the location by typing in N, S, E or W. If he did then this line will print the message "CAN'T DO THAT .. YET!!" and loop back to the INSTRUCTIONS? input message. No matter how many times the command for going etc. is given it will be impossible for the player to escape our little hazard.

Lines 7000–7030: These lines just display the peril on the screen then return to the instructions module in Line 550.

MODULE 1.9



```
1 goto3
2 save"@0:module 1.9",8:verify"module 1.
9",8:stop
3 rem
970 ifi$="swim" or i$="float" and p=2 th
en 2400
1220 ifve$="kill"then2500
2100 if p=18 thenprint"you could try kil
ling it":goto 550
```

This is a 'filler' module of three lines which will insert themselves into parts of the program but were not applicable earlier.

Line 970: One of the more difficult parts of Adventure programming is trying to think of all the various responses that the player might make to try and escape from his predicament once he has encountered a hazard. I will expand on this in great depth in the program *Nightmare Planet* in the second section of the book — but this line is put here to demonstrate the point.

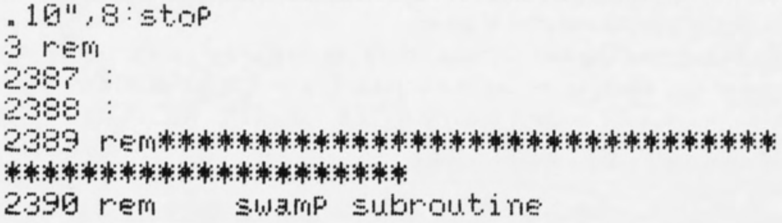
I assumed that, having fallen into a swamp, two of the most likely responses would be SWIM or FLOAT. So in anticipation this line states that if the response is either SWIM or FLOAT (and the Location is 2 as we do not want the escape response coming up in the wrong Location — it would look stupid!!) then the program is directed to Line 2400 which is a special section for the Swamp Peril (see Module 1.10).

Line 1220: This is the same principle as Line 970 but is in the section of the program looking at VERB (VE$) commands as you would almost certainly write KILL SOMETHING. In fact if you do write KILL on its own the program will respond with "PLEASE USE TWO WORDS". Try it and see. As before this line directs the program to Line 2500 if your response to the monster in the lake is to try and kill it.

Line 2100: This line joins the HELP section of the program. It seems sensible to try and help the player at times, so if he arrives in Location 18 and types H this line will display the message "YOU COULD TRY KILLING IT" before looping back to the "INSTRUCTIONS" command.

Traditionally help routines are vague and puzzling in Adventure games, but there are times when the player will be eternally grateful for a hint at a hard part of his struggle. Of course you can give false information or even deliberately lead the player towards his doom — it seems all is fair in Adventures!!

MODULE 1.10



```
1 goto3
2 save"@0:module 1.10",8:verify"module 1
.10",8:stop
3 rem
2387 :
2388 :
2389 rem********************************
*********************
2390 rem     swamp subroutine
```

```
2391 rem********************************
*********************
2400 if ob%(3)=-1 then print"okay - you
survived":sw%=2:goto 500
2410 print"you're too heavy without the
magic       jewel.  you've sunk !":stop
2487 :
2488 :
2489 rem********************************
*********************
2490 rem    kill subroutine
2491 rem********************************
*********************
2500 if p<>18 then 2540
2510 if ob%(2)=-1 then print"ok":mw%=2:g
oto 500
2520 if ob%(1)=-1 then print"with just a
 knife ? you must be joking !":goto550
2530 print"i haven't got anything to kil
l it with !":goto 550
2540 print"kill what ?":goto500
```

We have reached the final module of my example program — two little sections which deal with the two perils we have set up. If you look at lines 2400 and 2520 you will note that I have set the variables SW% and MW% to 2 once the peril has been successfully passed — which will allow the player to move out of this location and will ensure that when he visits it again the peril will not be displayed.

Lines 2400–2410: If you have entered the right response (having fallen in the swamp) in Line 2400 first the program checks if you have the magic jewel in your possession. OB%(3) is the jewel if you remember and when OB%(3) = 1 then we do have it in the Inventory. If this is so the program will display the message "OKAY — YOU SURVIVED" — set SW% to 2 — and loop back to Line 500 (to display possible exits now that we can move out of the Location). If we were writing a more complex program and wanted to check for further perils we would need to loop back to Line 400 to check for other special conditions at this point.

Line 2410 is the standard response if the player has *not* got the jewel in his Inventory and upon receiving the instruction to SWIM or FLOAT will display the gloomy reply "YOU'RE TOO HEAVY WITHOUT THE MAGIC JEWEL. YOU'VE SUNK!" and then END the program. This message will at least give the player an idea of what he needs to do next time around. Note the number of spaces between the word MAGIC and JEWEL (six). This is because it improves the display on the screen and does not break up the words so that part of JEWEL appears on one line and part on the next.

Lines 2500–2540: These deceptively simple lines hide quite a lot of thinking and most of the principles of peril programming that I have used in *Nightmare Planet*.

First we need to deal with a response of KILL that does not involve the monster in the lake. Line 2500 simply diverts the program past all the other responses to Line 2540 which is a standard reply for the KILL command in any other location.

Now we are left with the player commanding KILL MONSTER in Location 18. I have only worked out three possibilities in this program:

1. The player has the gun (and may have the knife).
2. The player has the knife only.
3. The player has neither.

In Possibility 1 (the player has the gun) Line 2510 notes that OB%(2) is in the INVENTORY and makes the response ''OKAY'' then sets MW% to 2 and returns the player to the game. Now he can escape from the location (note we have said GOTO 500 and not GOTO 550) and return safely at another stage. If the player had the knife as well as the gun it wouldn't have made any difference, of course. As long as OB%(2) is true this line is executed.

In Possibility 2 Line 2520 recognises that fact but will not let the player escape. So we PRINT an appropriate reply and then loop back to the INSTRUCTIONS command.

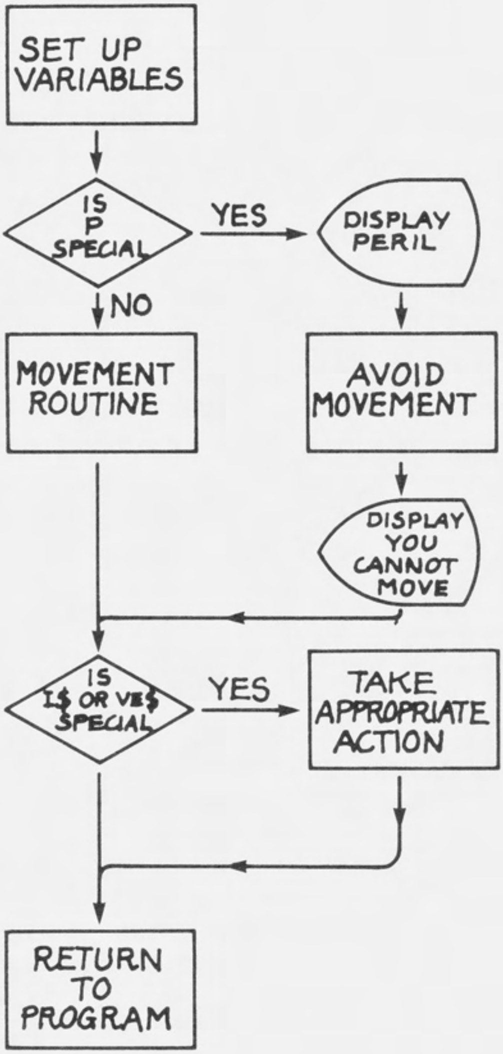
Finally in Possibility 3 (no gun or knife) there is a standard reply ''I HAVEN'T GOT ANYTHING TO KILL IT WITH'' before looping back.

It is important to have Line 2500 or else Line 2530 would be displayed if you wrote KILL in any other Location.

Some of the ideas and programming techniques I have introduced in this chapter merely scratch the surface of the construction of a good Adventure. In fact this part of your program will certainly take up the least space but take the most time to prepare and to debug afterwards. Thinking up a really intriguing and frustrating problem for your game and then seeing it work when someone else tries to fathom their way out of it is probably the most satisfying aspect of writing a good Adventure.

To help understand basic problem programming I have added a flow diagram which should apply to all cases in outline. (See **Flowchart 4.1**)

**Flowchart 4.1 Perils in Adventure Game**



If you have entered Adventure Game correctly along the way it should be possible to move freely from location to location until you encounter a peril — and then be stopped from moving out unless you have the appropriate response or appropriate object in your inventory. You should be able to pick up objects and when you return to that location the object will not be displayed. Alternatively if you drop an object then on returning to that location the object should now be displayed after the location description.

To test your understanding of the principles involved why not try to add a new location and a new object in it. You can try to add several locations and a couple of new perils as well.

Don't be afraid to experiment — both on paper and on your 64. Time spent on this now will greatly ease both your understanding of the second part of the book and your own ability to write your own original Adventure later.

# Part 2

# CHAPTER 5
# Writing the Plot

Most of the books and articles on programming will tell you that writing any type of software should follow certain well-defined paths and Adventure games should be no exception. I have found that despite the obvious desire to sit down at the keyboard and start programming right away this is one time when the advice — *think first* — is vital.

I know this is irksome — in fact the attraction of the hardware is a fatal flaw in my own programming — but unless you *do* force yourself to work out the bulk of your story on paper first all you'll achieve (besides creating problems for later) is an extra few weeks rewriting your program again and again.

The story (or perhaps a better word would be plot) is critical to both the success of the game and the structure of the program. In many ways the process of creating a suitable Adventure is similar to the methods that film-makers use when constructing a film (a concept I will return to later in the chapter), and I found this was the most exciting stage in my program. You need to both create the basic theme (write the story in other words) and then visualise it as though viewing it through the eyes of your audience.

Part of the thrill of most Adventure games I've played has been the feel of participation of actually taking part in the scenario. If it is to be successful this must be due to a combination of features which I will expand on in this chapter but just summarise at this stage to set your mind to thinking about them now — whilst starting to create the beginnings of your story.

The success and satisfaction of your game will depend on:

1. Your ability to use words to create images of your story.
2. The depth and plausibility of your plot.
3. The imagination of the person playing the game.

With regard to the last point you don't have any control over the skill and imagination of potential players, but as it appears that Adventure players are often fans of science fiction and fantasy then it seems a reasonable assumption that they will have a well-developed imagination — so I think we can take Point 3 for granted. The other two features now take on a more important role.

## Beginning the story

As I mentioned before Adventure games still tend in many cases to follow the style of the original versions created for mainframe computers or the well-known Scott Adams games — they have a bias towards the dungeons and dragons, science fiction, fantasy or horror theme. A quick scan of any magazine will throw up a variety of titles such as Mysterious Castle, Dracula's Lair, Island of Doom, Tale of the Dragon, and many more. Of course you don't have to follow this trend and there are several games with a totally different storyline, (escaping from an asylum, looking for the right husband/wife, attempting to slip out for a night on the town) which add a welcome touch of originality for the hardened player, but as the fantasy theme is such a popular one (and one I personally enjoy) I have based my own on the familiar space opera type of plot.

I read once there are no original plots for stories — only different variations. Of course it's true, but Star Wars is a perfect demonstration of the ability to take a simple plot and transform it into a smash hit! In all our stories we need some type of quest or goal to be achieved (find treasure, rescue a princess, escape from a dangerous situation, discover the meaning of a puzzle, etc.). We need a recognisable hero or heroine (in the Adventure game the player takes on that role) and usually either a villain or some other conflict for our main characters.

When I wrote my own story I used the following steps:

1. Select the environment (eg fantasy, horror, sf).
2. Choose a quest or goal (eg find treasure, escape from a wizard).
3. Decide on the role of the hero/heroine.
4. Select the main characters (eg wizard, vampire, countess).
5. Write a synopsis of the story.
6. Draw a simplified map with a few basic locations.
7. Storyboard the plot.

It may sound as though there is a lot of hard work before even touching the keyboard, but many of the steps in writing your story will follow so naturally that it becomes a fascinating and challenging goal in itself. And, as I have already mentioned at length, short-circuiting this step will either produce extra work later or result in an unsatisfactory product in the end.

## Select the environment

The traditional type of Adventure game which borrows heavily on the fantasy world of Tolkien and related writers, abounds with elves, dragons, sorcerors, castles dripping with magic and mystery and similar things. Perhaps it really is the nostalgia of the fairy-tales of our youth that partly explains this popularity — perhaps a deeper reason, but for the budding Adventurer the range of possibilities opened by selecting this environment adds a zest and originality less likely in the more mundane world. After all — anything is possible in your story!

Sword and sorcery is a branch of this type of environment which substitutes the more magical aspects for violence. In this genre (typified by the writings of Robert E Howard and his splendid hero Conan) the world is a dark and savage place where spells and sorcery are real and your prowess with the sword is your only real asset. Whilst being an excellent medium for adaptation to Adventure gaming the lack of enchantment found in the Tolkien world is my main reason for preferring fairytale fantasy Adventures.

Traditional horror themes are also extremely popular with Dracula playing a star role in many of these Adventures. Again the imagination of the programmer is unleashed with a variety of stimulating possibilities — people who suddenly become vampires, crypts full of dark and dank tunnels where the unexpected can leap out as you turn a corner, a time limit on escaping before you turn into a vampire etc. This environment lends itself to haunted houses with a number of rooms for you to move around, descents into Hell to confront the devil and reclaim your soul, escape from voodoo islands where zombies attack at every turn, werewolves, devil-worshippers, and countless more. To me this type of environment is the best for variety when thinking of a plot.

Another world packed with possibilities (and so far relatively ignored by writers) is the one of the comic strip hero. Superman, Batman and others have been demonstrating a variety of different themes over decades, (still the same basic plots but dressed differently) and a few quick ideas that come to mind are A-man (Adventure-man if you haven't already guessed) chasing super-villain Bugman all over Commodore City before Bugman reveals his secret identity, or A-man rescuing Lois Left$ from the evil Interface who is planning to take over the world. I'm sure this is one genre with a wealth of possibilities.

There are plenty of other situations you can use as I have mentioned, the detective solving the crime, the innocent caught up in the world of espionage, the castaway on a desert island. You only need to look at the programmes on the television, the books in the fiction section at your library or the films being shown at your local cinema to find immediate ideas for your story. And, of course, there is science fiction.

I've left science fiction until now because this is the environment I have chosen for my own adventure — *Nightmare Planet*. Purist SF fans would probably argue that *Nightmare Planet* is more space opera than true SF, but for the sake of simplicity I regard all stories with a background of time and space as science fiction. Again possibilities are pretty extensive, time travel to rescue the good Doctor who has been flung into the far future by his premature tamperings with a time machine, battles against the aliens planning to invade the Earth, searching a post-nuclear planet for life after the holocaust. It was into this type of general environment that I decided to place my story.

## Choose a quest or goal

It might seem strange to place this before any thought of the story itself, but as the whole idea of an Adventure game is to solve a puzzle, find an answer, achieve a goal — so the main consideration right from the start must be to decide what your own goal will be.

When I first began to work out *Nightmare Planet* I had only one goal — to rescue the Princess Aurora. As the story expanded during the programming stage I added a second goal — to find the energy crystal and bring it back to the spaceship. This added to the difficulty of the game and extended the scope of the Adventure quite considerably but was not really an essential part of the original story. Thus, despite the importance of choosing your goal, it is possible to amend it later or as I did — add to it.

Don't be tempted to start to work out fine detail at this stage. For example, suppose you have decided to make your goal FIND THE TREASURE inside the Castle of Doom. As your imagination begins to work out the story you also start thinking of whether to add a score to the game, adding 10 points for every item of treasure. Whilst your mind is thinking of this you *could* become diverted into adding the concept of subtracting points for various problems encountered which your player hasn't solved in a particular time limit. From this you *may* decide that you will need a display of the score on the screen all the time — so you sit down in front of the 64 and begin to work out the graphics of your scoreboard . . .

The essence of good storywriting at this stage is simplicity. The frills will come later once you begin to program.

## Decide on the role of the hero

This — quite simply — places the player into the scenario. As involvement in the adventure is the key to good playing you need to make your potential player feels as if he is the hero.

You have two main choices here:

1. Your player acts as himself thrown into the fantasy world.
2. Your player takes on the role of the fantasy hero.

I don't think it matters too much which you decide — as long as you make it clear right from the start of the game. In my own case I wanted the hero to be the pilot of a battered but reliable spaceship (shades of Han Solo) who made his living as a freight-operator.

## Select the other characters

Of course the other characters in the plot will depend on the story you are writing, so this stage should really be considered at the same time as creating the actual story itself. But as most of these stages are slightly artificial... (what really happens is that as you think of your story you will automatically be thinking of the hero, the location, the villain, etc.) it is easier to have some kind of structure which ensures that you don't leave anything out.

The main characters must be accomplices, people to rescue, villains, and assorted types to add local colour or act as red herrings or clue-givers. *Nightmare Planet* is fairly lacking in characters because of its location — an alien planet which contains various perils rather than villains — so the only other real character is the Princess Aurora.

One aspect of writing this type of game is the obvious problem of sexism. In any story where the player takes the role of a man trying to rescue a woman it could be argued that this game will only appeal to males, especially at certain stages of the plot, (this will be obvious to anyone who has played it already!). Whilst I have not done so for the purpose of this book — it would be a simple matter to include a prompt at the beginning of the game asking if the player is male or female. Upon receiving the appropriate response the game would then set various variables so that the Princess could become a Prince and the player become a female space pilot. To me this adds an element of flexibility to Adventure games lacking in other media such as books and films.

## Write a synopsis of the story

Ideas can occur at the strangest of times, often in the bath, late at night, or as in my case on the train. As I rattled down from Manchester to London one dull May morning the whole plot came to me quite unexpectedly and quickly, and I scribbled it down there and then. This formed the synopsis — which remained the same in basic content throughout the creation and programming of the whole game.

My synopsis was as follows:

> You are the pilot of a spaceship on a mission to deliver the beautiful Princess Aurora to the planet Zen where she is to be married to the tyrant Ruler. You have fallen in love with Aurora but dare not tell her.
>
> A sudden power failure or meteor storm causes you to crash on a strange, uncharted planet. You successfully land your ship but black out during the crash.
>
> When you awaken you discover the ship has been entered from outside and the Princess has been captured. You have to set out and rescue her.

> The atmosphere on the planet is poisonous so you are forced to wear your spacesuit at first, but along the way you chance upon an alien plant with strange fruit which (when eaten) allows you to breath the air safely. You cross a vast desert to a ruined city in your travels, but all you find there is a giant snake which attacks you.
>
> You eventually come to a huge forest and continue your search. You become lost, and after some time climb a tree to see where you are. To the south you see smoke curling in the air — life of some sort. On the way to the area you are attacked by a dinosaur, obviously the planet has not developed very far along the evolutionary trail yet, and eventually you find a village of mud huts belonging to the natives who inhabit this part of the land.
>
> You have found Aurora who is considered by the natives to be a Goddess. They threaten you when you try to take her away — and it is only when you kiss her and show them you are her intended mate that they will let you both escape.
>
> On the way back to your ship you discover that Aurora really loved you all along (there was obviously magic in your kiss!!) and you fly away together to a blissful future at the edge of the Galaxy.

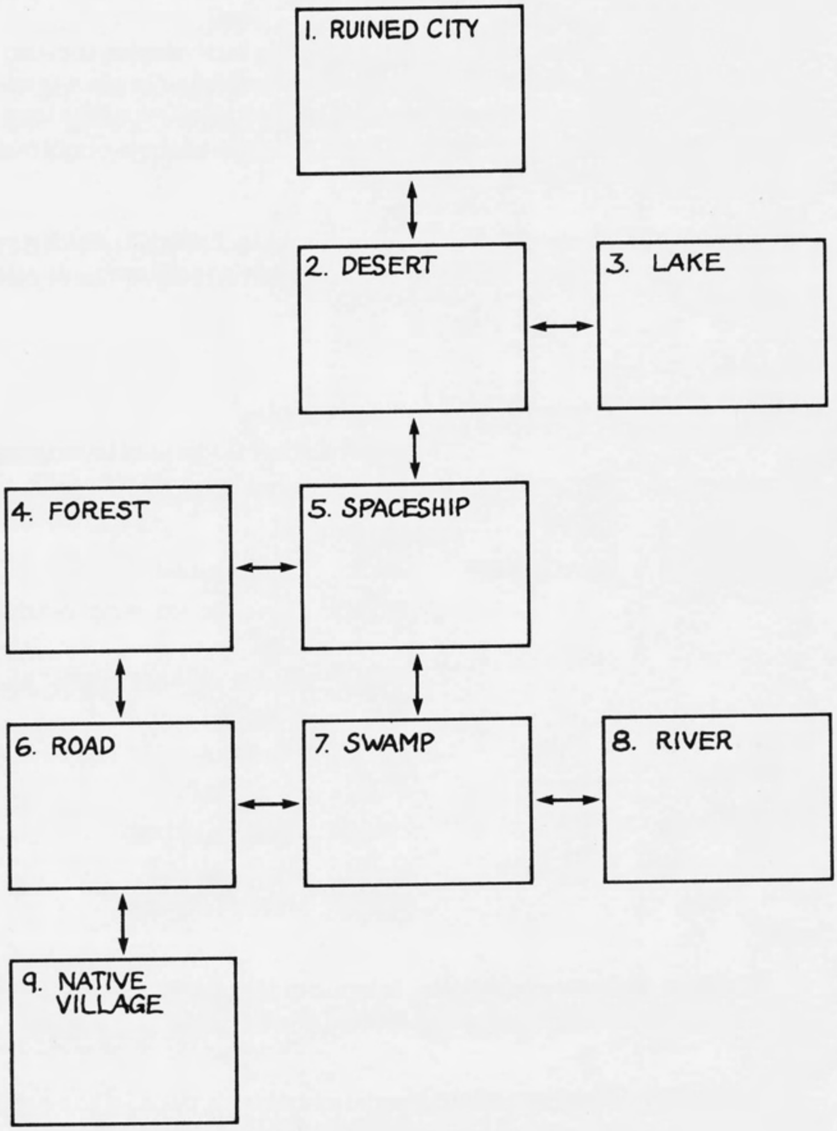
It may not be great literature — but the essence of my plot is all there. As I began developing the story so I added refinements and improvements along the way but surprisingly little. Much of the story is pretty obvious, but in some ways this adds to the feeling of satisfaction of the player. We often like to feel that the film we are watching or the book we are reading is 'right' — in other words we expect certain events to occur in a particular situation — and it adds to the satisfaction when we are indeed proved to be correct. So I didn't try to be too original or too way-out.

## Drawing the initial map

I have already covered the basics of maps in an earlier chapter — but at this stage in the creation of a real Adventure it is necessary to describe yet another type of map which will be converted later into the grid form outlined in Chapter 2.

You need some idea of the geographical relationship of the various locations in your adventure to help you avoid making mistakes in your planning later on. I found this was simpler if I just drew a very basic map (see **Figure 5.1**) which placed the various locations into 'real space' rather than attempting to fit them into a grid.

**Figure 5.1 Initial map for Nightmare Planet**



At this stage I kept the number of locations pretty small (in fact it varied from 7 to 9 at the beginning) because I knew I would be expanding certain sections quite considerably later. In the final version both the ruined city and the forest contained about 11 different locations each — allowing me freedom to place the objects and perils as I went along.

The discipline of drawing this map is a great help in developing the story, for as I began to think of the objects and perils I would start to place in specific locations so I found the skeleton of my original plot developing layers of clothing. **Figure 5.2** shows the start of this, and an analysis of my own story should illustrate how it helped me.

This map can now be expressed in the form of a table (**Table 5.1**) which is an excellent way of collecting your thoughts at this stage in the planning of your Adventure.

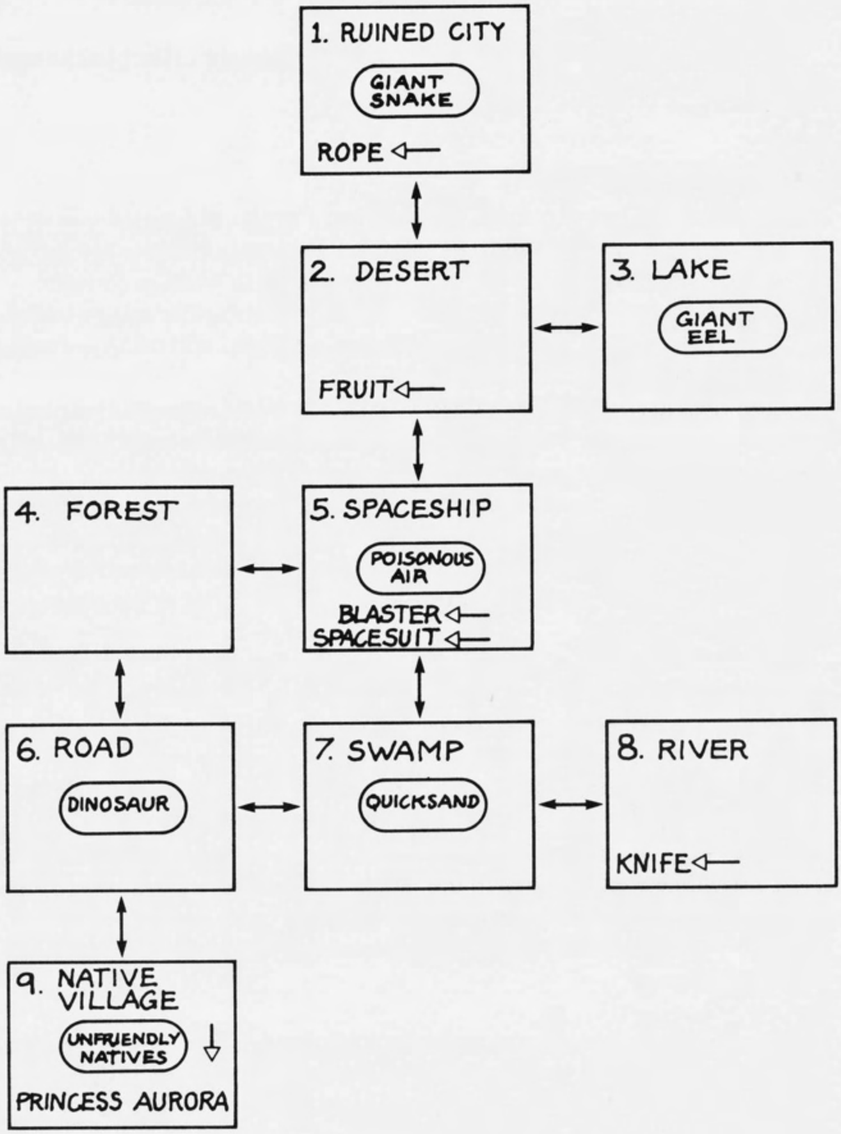
**Table 5.1**

| | | | |
|---|---|---|---|
| LOCATION 1 | RUINEDCITY | Peril: | Snake |
| | | Object: | Rope (to be used in swamp) |
| LOCATION 2 | DESERT | Object: | Fruit on bush (to be eaten) |
| LOCATION 3 | LAKE | Peril: | Eel |
| LOCATION 5 | SPACESHIP | Peril: | Poisonous air |
| | | Object: | Spacesuit (to wear outside ship) |
| | | Object: | Blaster (to kill snake and dinosaur) |
| LOCATION 6 | ROAD | Peril: | Dinosaur |
| LOCATION 7 | SWAMP | Peril: | Quicksand |
| LOCATION 8 | RIVER | Object: | Knife (to kill eel) |
| LOCATION 9 | VILLAGE | Peril: | Hostile natives |
| | | Object: | Princess Aurora |

This table needs some explanation, because in planning the initial map I was also planning the difficulty of the Adventure. Let's take a simple example:

In LOCATION 1 (the ruined city) lies a rope which is essential to escape the quicksand in the swamp (LOCATION 7) later on. But to GET the rope you first have to KILL the snake — and you cannot do that unless you remembered to GET the blaster before leaving LOCATION 5. Once you reach LOCATION 9 (the native village) you cannot rescue Aurora without KISSING her. But you cannot KISS her unless you remove your spacesuit

**Figure 5.2 Initial map for Nightmare Planet with objects and perils**



and if you REMOVE your spacesuit before EATING the fruit in LOCATION 2 (the desert) then you will die in the poisonous atmosphere.

(nb the words in capitals in these examples refer to commands that you will be giving the 64 in your game.)

Having drawn the initial map and given some thought to the plot the next stage is to start to storyboard your Adventure.

## Storyboard the plot

Essentially I'm a visual person. I love films, I prefer illustrated stories to prose (that means comics) and when it comes to writing I tend to see the finished product in my mind's eye before I put finger to word-processor!

So it was natural for me to begin to write my Adventure by using a technique of story-boarding similar to the process a director will often use when planning out a film in the early stages.

A storyboard is just a collection of visual images portraying the *story* on a collection of *boards* — a strip cartoon of the film in other words. Obviously writing a computer program relying heavily on text is not the same as shooting an epic, but if I was to use a similar principle then what I wanted was to 'see' first what the player would see on his computer screen. I needed to imagine the layout of text on the screen — and to picture some of the possible responses of my potential player so I could begin to think about the framework of my plot.

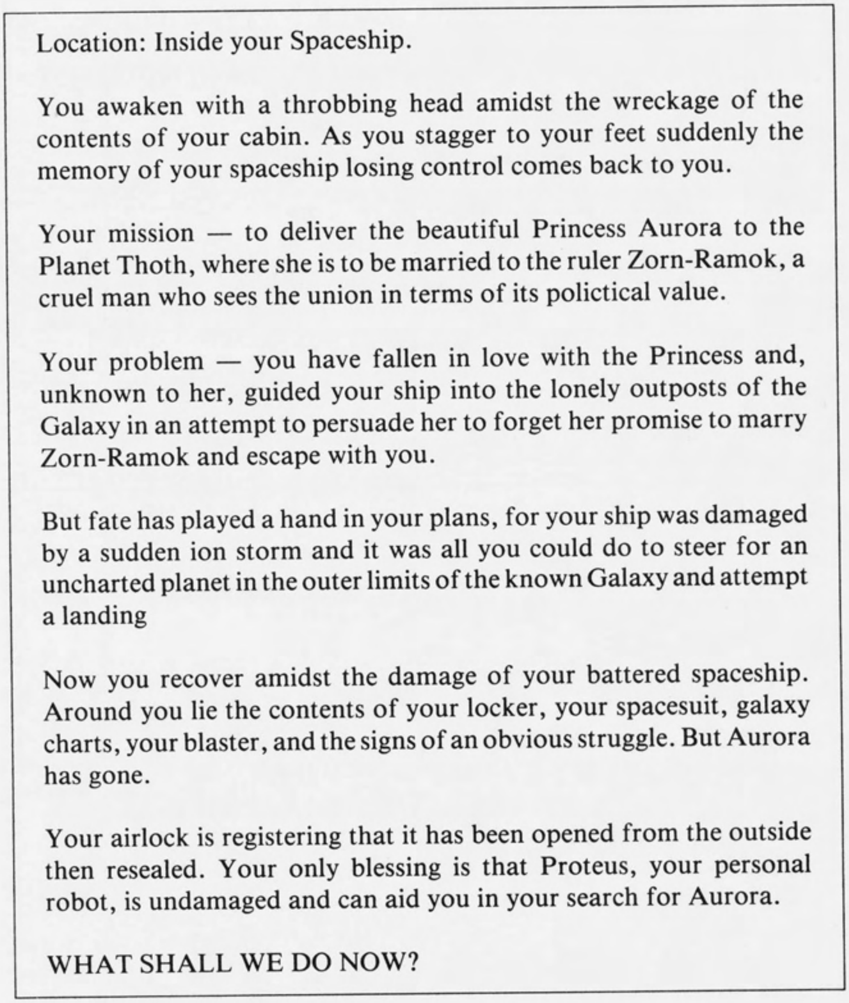
At first this technique may seem a bit of a time-waster — after all who wants to sit around writing imaginary responses to ''WHAT SHALL I DO NOW?'' on bits of paper when there's a keyboard to play with — but in my own case I found this part of my construction extremely valuable.

I suspect it was because I was able to begin to 'thought-launder' (a phrase I've borrowed from a good friend of mine because it so aptly describes the process). This means that I was forced (by the act of writing down various ideas and replies) to think much more deeply about both my plot and also my locations. I began to get ideas which would serve me in good stead later. In short — I was really thinking about the story. **Figure 5.3** might explain what I mean.

The wording on my very first storyboard is a little too long for the amount of memory in the 64, but it captures the feel of the game. And although the actual wording I will use later in the program is different, at this stage one or two ideas were emerging that I would use later — ideas I might have used had I not gone through this process first.

A simple example is the ''personal robot Proteus'' who has survived the crash and as it says on the screen ''. . . can aid you in your search for Aurora.''

**Figure 5.3 Inside the spaceship**



Location: Inside your Spaceship.

You awaken with a throbbing head amidst the wreckage of the contents of your cabin. As you stagger to your feet suddenly the memory of your spaceship losing control comes back to you.

Your mission — to deliver the beautiful Princess Aurora to the Planet Thoth, where she is to be married to the ruler Zorn-Ramok, a cruel man who sees the union in terms of its polictical value.

Your problem — you have fallen in love with the Princess and, unknown to her, guided your ship into the lonely outposts of the Galaxy in an attempt to persuade her to forget her promise to marry Zorn-Ramok and escape with you.

But fate has played a hand in your plans, for your ship was damaged by a sudden ion storm and it was all you could do to steer for an uncharted planet in the outer limits of the known Galaxy and attempt a landing

Now you recover amidst the damage of your battered spaceship. Around you lie the contents of your locker, your spacesuit, galaxy charts, your blaster, and the signs of an obvious struggle. But Aurora has gone.

Your airlock is registering that it has been opened from the outside then resealed. Your only blessing is that Proteus, your personal robot, is undamaged and can aid you in your search for Aurora.

WHAT SHALL WE DO NOW?

I first hatched the idea of having a little robot who would be around to come in with a comic comment or quip on this screen — my first storyboard. This idea would change and develop, to become Victor the robot (a sprite in the 64) who drops down from the top of the screen at various moments throughout the game. I wonder if I'd have thought of Victor if I hadn't spent the time thinking and writing up this first screen back at the beginning?

**Figure 5.4 Beside a huge river**

LOCATION: Beside a huge river.

The jungle clears and suddenly you stand on the banks of a huge river, a vast stretch of water almost like a sea, yet moving swiftly past your feet with the speed of a tumbling stream.

Too far to cross by swimming, you feel disheartened. To have come so far only to be thwarted now.

Around you lie the rocks and debris of an ancient age, looking as they have done for thousands of years. You sit for a spell, gazing across that vast swiftly-moving river, before deciding to make your way back.

**Figure 5.5 Inside the forest**

LOCATION: Forest

You wander into a vast forest, filled with trees as tall as skyscrapers with trunks as thick as houses.

Everywhere the air is thick with insects, darting and flashing in the sunlight filtering through from above. Occasionally a small animal will run from cover across your path and disappear in the forest.

As you walk on you realise that you are going ever deeper, for the air is becoming colder and the light fading as the trees become thicker and closer together. Soon any sign of a pathway has gone, and you are left to fight through bush and bramble.

You are lost. With a chill you realise there is no way you can remember the way forward or back. And behind you are strange noises, as if something huge is following you slowly and with certainty.

The two screens in **Figure 5.4** and **Figure 5.5** are just further locations, again to show how I was planning out my plot. I hope the atmosphere of the game can be felt in the wording of these screens, and although I wasn't able to go to the same lengths of descriptive phrase on the 64 I think I captured the essence of the 'feel' of *Nightmare Planet*.

**Figure 5.6 By the shores of a lake**

```
LOCATION: By the shores of a lake.

WHAT SHALL WE DO NOW?

Dive

CAN'T DO THAT......YET!

WHAT SHALL WE DO NOW

Swim

O.K

You bob silently on the top of the water, looking down at the glinting
metal object on the bed of the lake. It is another blaster from the
spaceship.

WHAT SHALL WE DO NOW?

Dive

O.K

A giant monster eel attacks you, appearing as if from nowhere

WHAT SHALL WE DO NOW?

Fire blaster.

IMPOSSIBLE — it is too damp

WHAT SHALL WE DO NOW?

Kill monster

I DON'T KNOW WHAT 'MONSTER' IS

WHAT SHALL WE DO NOW?

Kill eel
```

**Figure 5.6** is one of the many screens I made up as I went through my story — trying to imagine I was keying in the appropriate responses to the messages from the computer. When I came to the actual programming of this sequence (Chapter 12) I was already fairly certain of what I wanted — and although I changed a few ideas around slightly the essence remains in the final version of *Nightmare Planet*.

## Summary

This chapter should have been the hardest part of writing your own Adventure and perhaps the most frustrating for keen programmers — we still haven't switched the computer on.

But I have a feeling that Adventurers are imaginative and creative people who will find that once the idea of a plot-line has crept into their mind it'll be hard to shake it. Once this happens then the only way to escape is to write — to transfer the thoughts to paper and then to computer.

I hope my introduction to the actual act of transferring that idea, that concept that will make your Adventure unique and workable will save you time and heartache later. I have spent some time on it because all the books and articles I have read so far on programming Adventures concentrate on the technical expertise of programming and tend to skip the hard part — the creation.

One final tip before you turn to Chapter 6 and start hitting the keys — take your time getting your story right. There's no rush to finish. Time in thinking now will pay off in your final effort.

# CHAPTER 6
# Setting the Scene

Computer games, like everything else in life, can succeed or fail in the first few moments. We all make snap decisions about people, places, books, jobs, etc. based on first impressions — rightly or wrongly — and so those first few seconds of your game are going to be very important.

A good game will start with a good title — one that will grab the attention and hold it. When you put a cassette into your tape player, wait for interminable minutes while it loads, and then are greeted with a poorly presented title — you feel aggrieved. And if you don't — then you should! Good titles are well-designed (and correctly spelt) graphics and words which give a taste and an impression of the game to come.

There is no need to go wild about it, (I've seen some games which expend virtually all the memory, and most of the programmer's skill it seems, on the title to the detriment of the program itself). But I've also seen appalling titles with words off-centre, typed incorrectly, and no attempt to change colours or even try and place a graphic anywhere.

## The initial concept

In the last chapter I described the basic plot for my story — but before starting to develop the plot further on the 64 I wanted to get a good 'feel' into the game by designing and programming a title I felt did it justice.

As it's possible to create your own character set I toyed with the idea of designing my own letters for the words *Nightmare Planet* but time and memory eventually excluded that, although I did spend quite a bit of effort designing various attempts. In the end I decided I would like a simple title with a spaceship somewhere on the screen, and to enhance it I would move a sprite across the words, preferably with a whooshing noise.

I will deal with the sprite itself later in Chapter 8 — for now let's start with the title itself.

N.B. I have split the program into chunks called Modules in this section of the book similar to the Modules in Section 1. However there will be quite a few places where lines will be omitted in the early modules to make it possible for you to enter and run your program (to check it). Thus as these missing lines will be added later please ensure you follow the numbering carefully or you will overwrite lines later on — with a lot of problems in debugging.

I also had problems with memory, running up to the limit on several occasions. To retrieve memory I was forced to renumber the program and to cut down the asterisks in the REM statements to 2. I did contemplate removing the REM's altogether — but decided against it as it makes the program so much easier to follow — especially several months later.

MODULE 6.1

```
1 goto3
2 save"@0:module 6.1(a)",8:verify"module
 6.1(a)",8:stop
3 rem
100 rem**
110 rem pre-credits
120 rem**
130 print"█":poke53280,0:poke53281,0
140 print"█████in the far flung future .
....."
150 goto280
250 rem**
260 rem variables
270 rem**
280 clr:v=53248:sc=54272:bs=53280:bc=532
81
290 pr=0:cv=0:sv=0:ev=0:qv=0:dv=0:es=2:n
v=0:na=0:np=0:fv=0:rc=0:yy=0:hh=0:hr=0
300 vo=54296:w1=54276:a1=54277:s1=54278:
w2=54283:a2=54284:s2=54285
310 w3=54290:a3=54291:s3=54292
320 h1=54273:l1=54272:h2=54280:l2=54279:
h3=54287:l3=54286
970 rem**
980 rem title
990 rem**
1000 pokebs,0:poke bc,0:print"█":ss=1024
:cs=55296
1010 poke53272,21
1020 fori=1to23
1030 pokecs+40*i,4:pokess+40*i,102
1040 pokecs+40*i+38,4:pokess+40*i+38,102
1050 nexti
1060 print"██▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
▒▒▒▒▒▒▒▒▒▒▒"
1070 printtab(11)"████nightmare planet"
1080 print"█████████an adventure in time
 and space"
```

```
1090 Print"▓▓▮▮▮▮▮▮▮|"
1100 Print"▮▮▮▮▓▒ ▒"
1110 Print"▮▮▮▮▓B+B▓
   ▓Q"
1120 Print"▮▮▮▮▓▓BQB"
1130 Print"▮▮▮▮▓▓BQB"
1140 Print"▮▮▮▓▒BQB▒▓      ▒▓*"
1150 Print"▮▮▮▮▓ ▓B+B▮ "
1160 Print"▮▮▮▮ ▓B▒B▮ ▓
        ▓*"
1170 Print"▮▮▮▮▓ ▓B▒B▮ ▓"
1180 Print"▮▮▮▮▓      ▓ ▓"
1190 Print"▮▓▓▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
1210 form=1to1500
1220 nextm
```

Lines 100–150: The game takes a few seconds to read and set up all the variables (especially the setting up of the sprites) and this results in a pause after you have typed RUN. I always find these pauses irritating and wonder if everything is okay or if I have done something wrong — so I decided to use this interval to put a short "pre-credit" sequence in.

As soon as you type RUN the screen clears, goes totally black, and the words "IN THE FAR FLUNG FUTURE" appear. Line 150 then has a GOTO statement as a section will be placed between the pre-credit sequence and the variables later.

Lines 250–320: These lines define all the variables. There seem to be quite a lot, but for now let's just look at a few as most of them will be described more fully in the appropriate module. (There is a glossary containing all variables at the end of the book).

Line 280: V is the starting address of the Video Chip and allows us to create and move sprites around using the formula V + X (X being the appropriate number). SC is the starting address of the Sound Chip. BS is the starting address of the screen border colour and BC is the starting address of the centre colour. (The reason why these are NOT used in Line 130 are simply for convenience in keeping all the variables here in the program).

Line 290: This line contains a number of variables concerned with playing the game which I shall refer to later.

Lines 300–320: These variables are all sound values for all three voices on the 64, and will be explained more fully in the section on sound effects.

The next section is the title page itself. I often find working out graphics in a read-out tricky (especially counting up cursor right and space symbols) so I have included a second version of Module 6.1 which has been converted to CT Standards.

For those of you unfamiliar with these they are very easy once you get the hang of them (they just look awful!!).

Thus [CLS] stands for "CLEAR SCREEN" and [CU] and [CD] stand for "CURSOR UP" and "CURSOR DOWN" respectively. [G< ] and [G> ] refer to "GRAPHIC LEFT" and "GRAPHIC RIGHT" so [G< + ] means the " + " key with the Commodore key held down — resulting in the appropriate graphic symbol.

If you look at Line 1180 in Module 6.1(b) I will explain it as clearly as I can:

```
1180 PRINT"[3CR] – 3 cursor right –
              [GRN] – control key plus green colour key –
              [G< K] – graphic left key K –
              [3SPC] – 3 spaces –
              [REV] – control key plus reverse on –
              [G< K] – graphic left key K –
              [OFF] – control key plus reverse off"
```

Thus if you refer to Module 6.1(b) as well as Module 6.1(a) you should have no trouble keying in this part.

## MODULE 6.1(b)

```
970 REM**
980 REM TITLE
990 REM**
1000 POKEBS,0:POKE BC,0:PRINT"[CLS]":SS=1024:CS=55296
1010 POKE53272,21
1020 FORI=1TO23
1030 POKECS+40*I,4:POKESS+40*I,102
1040 POKECS+40*I+38,4:POKESS+40*I+38,102
1050 NEXTI
1060 PRINT"[CR][PUR][38G<+]"
1070 PRINTTAB(11)"[3CD][WHT]NIGHTMARE[SPC]PLANET"
1080 PRINT"[5CR][3CD][GRN]AN[SPC]ADVENTURE[SPC]IN[SPC]TIME[SPC]AND[SPC]SPACE"
1090 PRINT"[2CD][5CR][GRN][G>-]"
1100 PRINT"[4CR][REV][G>£][SPC][G<*]"
1110 PRINT"[4CR][REV][G>B][G>+][G>B][OFF][23SPC][RED][G>Q]"
1120 PRINT"[4CR][REV][GRN][G>B][G>Q][G>B]"
1130 PRINT"[4CR][REV][GRN][G>B][G>Q][G>B]"
1140 PRINT"[3CR][REV][G>£][G>B][G>Q][G>B][G<*][OFF][4SPC][PUR]*"
1150 PRINT"[3CR][GRN][G<K][REV][G>B][G>+][G>B][G<K]"
1160 PRINT"[3CR][G<K][REV][G>B][G<+][G>B][G<K][OFF][27SPC][YEL]*"
1170 PRINT"[3CR][GRN][G<K][REV][G>B][G<+][G>B][G<K][OFF]"
1180 PRINT"[3CR][GRN][G<K][3SPC][REV][G<K][OFF]"
1190 PRINT"[CR][CD][PUR][37G<+]"
1210 FORM=1TO1500
1220 NEXTM

THIS LISTING WAS PRODUCED USING THE 64 FORMAT CONVERTER
```

Lines 1000–1010: First we clear the screen again, and turn the border and centre to black with POKES BS and BC. The reason for doing it again is because at a stage later in the game you may wish to return to this portion and would not be re-entering the program right back at the beginning.

Two variables are set here, SS and CS. These concern the setting up of a pattern around the screen and as they refer just to this section of the title I have left them here for simplicity. SS is the starting address of the screen and CS of the colour memory.

Line 1010 converts the 64 to upper case — unnecessary on the first RUN but as the Adventure text is in lower case (I think it is much more attractive) you need this line to convert back should you be RUNning the program a second or third time.

Lines 1020–1050: This routine POKE's two vertical lines on the extreme left and right of the screen using the graphic symbol on the Commodore Key and " + " (POKE 102) in purple (POKE 4). I used POKEing here because it is possible to place the symbol in the last right position without causing a carriage return and causing the whole screen to move up one line.

Lines 1060 and 190: Simple PRINT statements which fill in the horizontal upper and lower borders of the screen with the same graphic symbol.

Lines 1070–1180: The picture of a spaceship in the left lower portion of the screen plus three stars and the title words will appear once you key in these lines.

Lines 1210–1220: Finally a short timing loop to hold this title for a few moments. This loop is a little short at this stage as once we added the sprite to this sequence it will prolong the time the title stays on the screen. Of course you can easily add to this timing loop (FOR M = 1 TO 3000 for example) if you want your title to stay on the screen longer.

*Testing Module 6.1*

If you type RUN the screen should black out all over, and after the pre-credit sequence a box should draw itself around the border followed by the words "NIGHTMARE PLANET — AN ADVENTURE IN TIME AND SPACE" in the centre of the screen in the upper half. Finally a green spaceship should be standing poised for take-off in the lower left corner.

## New readers start here

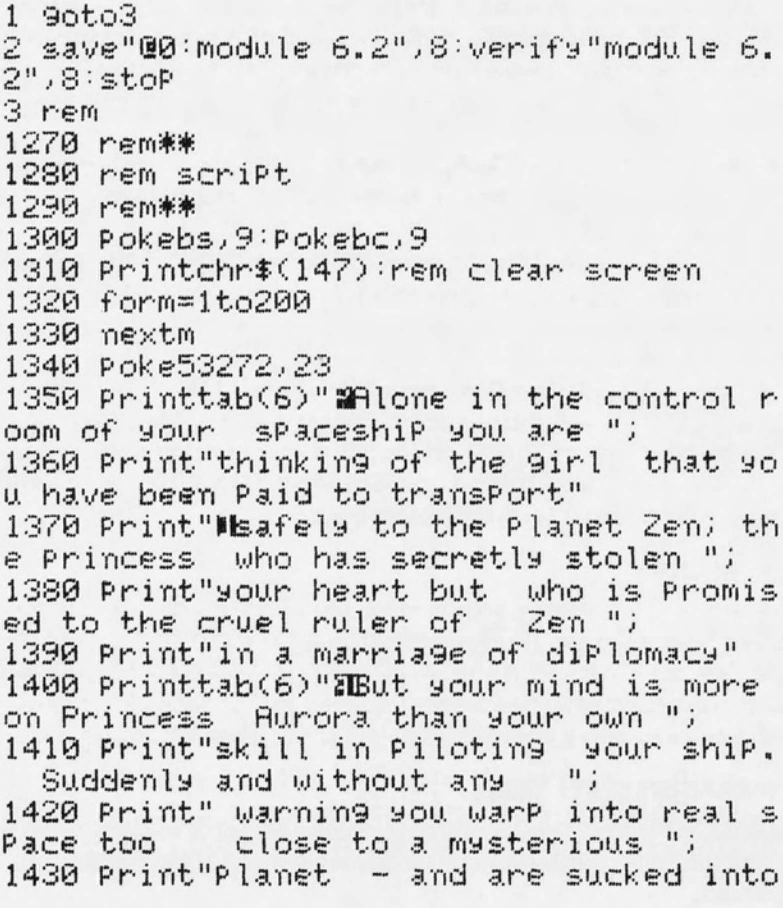
If you look at **Figure 5.3** in Chapter 5 you will see how I planned my opening screen in the storyboard section. This second Module is the result of that storyboard.

I have often felt that Adventures need a touch of drama to make you feel you really are entering into the imaginary world (the Scott Adams games I've seen are completely lacking in this) so having set up my title I wanted a page of text to appear which would set the scene for what is to follow.

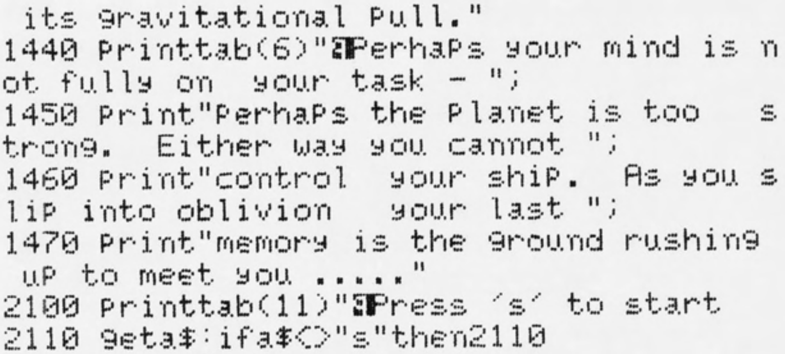
Writing text for computers isn't easy. I had to write it in direct mode first of all to get a good balance, changing words to fit the screen. Then I decided on a brown background with black lettering for dramatic effect. Finally I had to key in the text so that the words were positioned exactly as I wanted.

The problem of colours is one you will only solve by playing around with different borders and coloured letters, trying out various combinations until you are happy. Again I feel time spent here on adding that little extra is well spent in giving your game an overall feel of professionalism.

MODULE 6.2



```
1 goto3
2 save"@0:module 6.2",8:verify"module 6.
2",8:stop
3 rem
1270 rem**
1280 rem script
1290 rem**
1300 pokebs,9:pokebc,9
1310 printchr$(147):rem clear screen
1320 form=1to200
1330 nextm
1340 poke53272,23
1350 printtab(6)"▒Alone in the control r
oom of your  spaceship you are ";
1360 print"thinking of the girl  that yo
u have been paid to transport"
1370 print"▌safely to the Planet Zen; th
e Princess  who has secretly stolen ";
1380 print"your heart but  who is promis
ed to the cruel ruler of   Zen ";
1390 print"in a marriage of diplomacy"
1400 printtab(6)"▒But your mind is more
on Princess  Aurora than your own ";
1410 print"skill in piloting  your ship.
  Suddenly and without any   ";
1420 print" warning you warp into real s
pace too     close to a mysterious ";
1430 print"Planet  - and are sucked into
```



```
 its gravitational pull."
1440 printtab(6)"█Perhaps your mind is n
ot fully on  your task - ";
1450 print"perhaps the planet is too   s
trong.  Either way you cannot ";
1460 print"control  your ship.  As you s
lip into oblivion   your last ";
1470 print"memory is the ground rushing
 up to meet you ....."
2100 printtab(11)"█Press 's' to start
2110 geta$:ifa$<>"s"then2110
```

Lines 1300–1340: Before the text appears it is necessary to change the colour to brown, clear the screen (PRINTCHR$(147) is an alternative to PRINT[CLS]) and set up a tiny loop. Finally line 1340 sets the text to lower case.

Lines 1350–1470: Straightforward text lines — enter carefully with an eye for spacing. The annoying graphic symbols that appear are because of the upper case letters at the beginning of the sentence.

Lines 2100–2110: Having set up the text we need to keep it on the screen long enough for anyone to actually read it — so what better way than to have a holding line with the prompt "PRESS S TO START"? Line 2110 is a standard method of keeping a PRINT statement on the screen until the operator wishes to continue.

*Testing Module 6.2*

If you now RUN the two modules together you should find the screen will change to a lovely brown and the text will be evenly spaced out over the whole area. No words should be broken — if they are check the spacing in your own listing.

Now press S and the program will break out with the READY prompt.

## General hints on text writing

As much of the actual space of an Adventure game is taken up with text I will spend a few moments on tips I've developed in writing *Nightmare Planet*.

The biggest problem (as you may already have discovered) is spacing the words correctly on the screen. The way I tackle this is to write two or three PRINT statements at one time and then check their appearance on the screen.

For example I would write Lines 1350–1370 and then in direct mode type GOTO 1350. The few lines would appear giving me an idea of where I needed to add or subtract spaces to make the text look neat. By then adding LIST 1350–1370 (again in direct mode) I can play around with the lines whilst the display is still on the screen. In fact if you type your GOTO statement in the very top line all you have to do is HOME the cursor after every change you make and then press RETURN to give you an immediate display.

Later on in the program you will find PRINT statements with GOSUB and GOTO statements in the same line, and I found it easiest to add a syntax error here

```
eg . . . HOME.";GOTO 3300 (note the semi-colon)
```

instead of

```
. . . HOME.":GOTO 3300
```

By using a semicolon instead of a colon I would cause the 64 to break out of the program with the SYNTAX ERROR message and not execute the GOTO or GOSUB — which was exactly what I wanted to allow me to edit the text.

## Summary

This chapter has set us on the way into the program. It is fairly easy so far, but I hope you'll agree that already the atmosphere of the story can be felt in the title page and the text.

Of course your own Adventures can be as simple or complex as you like. Some might foresake any title altogether to allow more memory for the program itself. Others might indulge in better graphics. Every programmer is different. My intention in this chapter is to introduce you to text programming and give you some ideas of presentation. The rest is, as always, up to you.

# CHAPTER 7
# Do you need help?

Just to complete some of the essential parts of your game before we move into the actual storyline I'm going to deal with ''General Instructions'' and the modules for SAVEing to tape and LOADing from tape in this chapter, and the addition of sprites and sound in the next chapter.

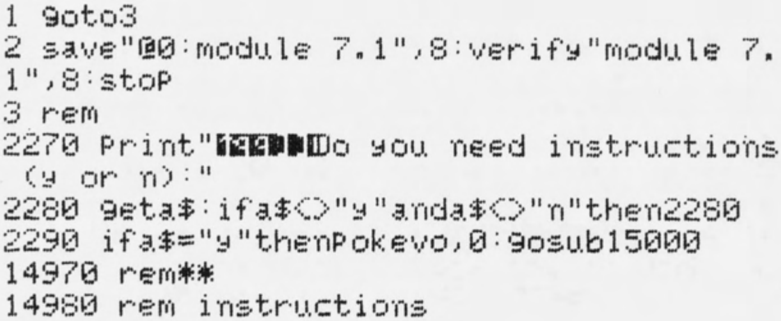
The reason for doing this first is because these modules will form valuable subroutines that you will want to be able to put into your program as you go along — which is much faster than having to go back after programming the main story to keep adding them in the appropriate places.

Also the ability to save and load will make life much easier for you once you begin the main story, as you can save your own game and then reload it time and time again as you test out particular lines. This is much quicker than having to play the game up to each location every time you want to test out a particular routine.

## If you really need help!

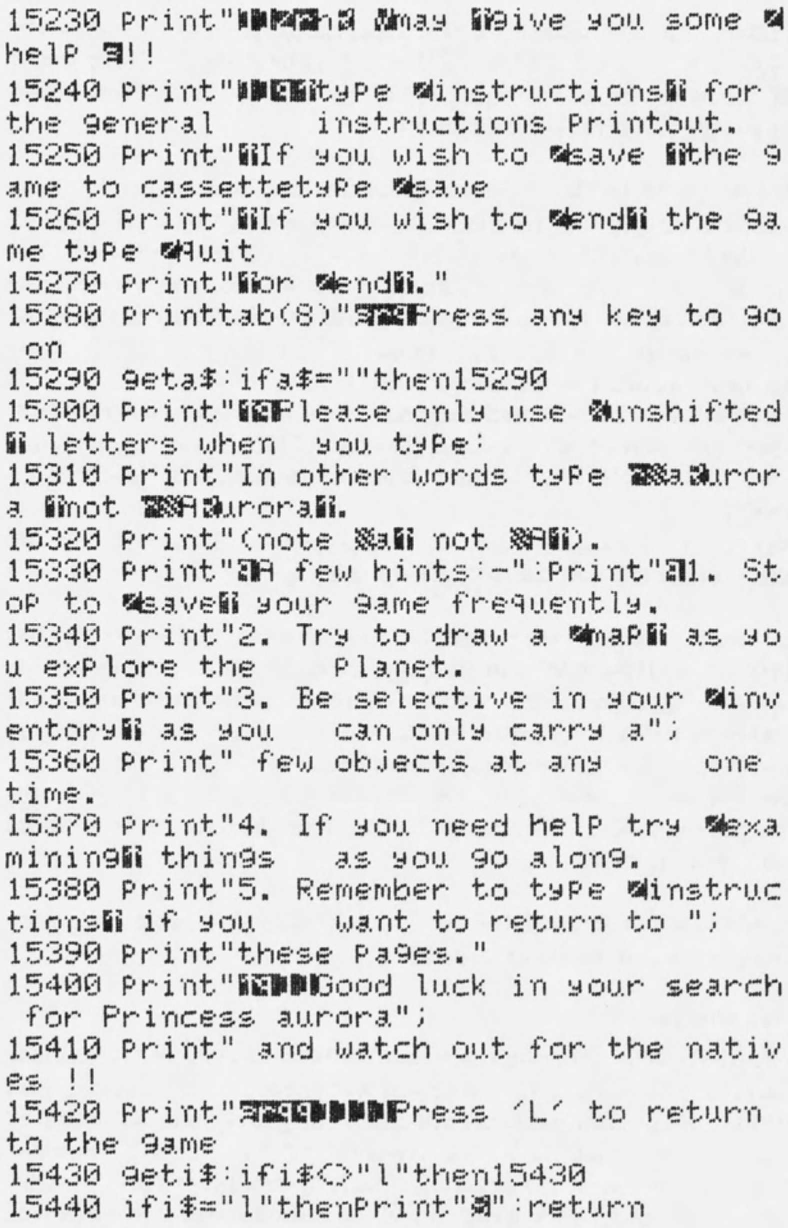
When I had been programming *Nightmare Planet* for about a month I asked a few friends round to have a go — the best way to discover your mistakes! I just sat them down in front of the computer and watched, pen and paper at the ready to jot down my mistakes. What surprised me most, however, was their elementary lack of knowledge about just playing an Adventure game. Now whilst you and I know the rules and traditions of Adventure playing — not everyone does. So the good programmer will provide a page or two of instructions to help the novice to Adventure games.

MODULE 7.1



```
1 goto3
2 save"@0:module 7.1",8:verify"module 7.
1",8:stop
3 rem
2270 print"▮▮▮▮▮▯Do you need instructions
 (y or n):"
2280 geta$:ifa$<>"y"anda$<>"n"then2280
2290 ifa$="y"thenpokevo,0:gosub15000
14970 rem**
14980 rem instructions
```

```
14990 rem**
15000 Print"▨":Pokebs,11:Pokebc,11:Print
tab(13)"▨▨Instructions▨
15010 Print"▨▶▶▶▶▨You are enterin9 a wor
ld of fantasy and adventure";
15020 Print" with your comPuter as your9
uide.  To solve this";
15030 Print" adventure you must9ive the
comPuter commands:
15040 Print"always a ▨verb▨ and in some
cases a ▨noun▨.
15050 Print"for examPle:- ▨9et blaster ▨
or ▨climb tree
15060 Print"▨▶▨You will be 9iven a descr
iPtion of the location you are ";
15070 Print"in and also the direct-ions
in which you may move. You can only";
15080 Print"9o ▨north▨, ▨south▨, ▨east▨
or ▨west▨.
15090 Print"▨▶At times you may chance uP
on an object which could be of use ";
15100 Print"to you later in   the 9ame a
nd you should Pick it uP (by  usin9 ";
15110 Print"the word ▨'9et'▨ or ▨'take'▨
).  This  will also aPPly to ";
15120 Print"▨'9ettin9'▨ your sPace-suit
or ▨'9ettin9'▨ the Princess back.
15130 Printtab(8)"▨▨Press any key to 9o
 on
15140 9eta$:ifa$=""then15140
15150 Print"▨▨▶▶You may also ▨'droP'▨ or
 ▨'remove'▨ an    object at any time ";
15160 Print"and that object will remain
in that location for you to Pick ";
15170 Print"uP should you need it a9ain.
15180 Print"▨▨SPecial key letters will s
ave you time: for examPle:-
15190 Print"▨▨▶▨n▨ ▨e▨ ▨s▨ ▨or ▨▨w▨ ▨wi
ll take you ";
15200 Print"▨north▨, ▨east▨, ▨south ▨or
▨west▨.
15210 Print"▶▶▨▨i▨ ▨will disPlay an ▨inv
entory ▨of objectsyou are carryin9.
15220 Print"▶▶▨▨l▨ ▨will let you ▨look▨
around."
```



```
15230 print"▒▒▒▒n▒ ▒may ▒give you some ▒
help ▒!!
15240 print"▒▒▒▒type ▒instructions▒ for
the general       instructions printout.
15250 print"▒If you wish to ▒save ▒the g
ame to cassettetype ▒save
15260 print"▒If you wish to ▒end▒ the ga
me type ▒quit
15270 print"▒or ▒end▒."
15280 printtab(8)"▒▒▒Press any key to go
 on
15290 geta$:ifa$=""then15290
15300 print"▒▒Please only use ▒unshifted
▒ letters when  you type:
15310 print"In other words type ▒▒a▒uror
a ▒not ▒▒A▒urora▒.
15320 print"(note ▒a▒ not ▒A▒).
15330 print"▒A few hints:-":print"▒1. St
op to ▒save▒ your game frequently.
15340 print"2. Try to draw a ▒map▒ as yo
u explore the     planet.
15350 print"3. Be selective in your ▒inv
entory▒ as you    can only carry a";
15360 print" few objects at any      one
time.
15370 print"4. If you need help try ▒exa
mining▒ things    as you go along.
15380 print"5. Remember to type ▒instruc
tions▒ if you     want to return to ";
15390 print"these pages."
15400 print"▒▒▒▒Good luck in your search
 for Princess aurora";
15410 print" and watch out for the nativ
es !!
15420 print"▒▒▒▒▒▒▒Press 'L' to return
to the game
15430 geti$:ifi$<>"l"then15430
15440 ifi$="l"thenprint"▒":return
```

Lines 2270–2290: Once the title pages are finished and the player has pressed S to start the game 2 more prompts should follow before getting into the game proper. The first of these is whether to load from tape and the second is displayed on the screen by Lines 2270–2290. Basically the player is asked if he wishes to have "instructions".

Line 2290 acts if the response is Y by sending the program to the subroutine at 15000. (The instruction POKE VO,0 in this line switches off the sound which I'll be covering in Chapter 8). If the response is N the program automatically drops to the next section.

Lines 14970–15410: This is the whole subroutine. On screen it looks like three separate pages of instructions, each one scrolling up on pressing any key on the 64 (Lines 15140 and 15280). There's not much to say here. The only problem you might have is interpreting the various colours and reverse on/reverse off keys as I have highlighted words and phrases by changing the colour all over the text. This also enhances the appearance of the pages, giving them an attractive presentation and layout.

The basic colours I have used are green for the main text, blue (the Commodore key plus key 7 not the Control key plus 7) for keywords, white for the "PRESS A KEY" lines and black for the heading "INSTRUCTIONS".

Once again when keying in text only write a couple of lines at a time and then GOTO the line number to check the spacing.

Lines 15420–15440: Notice that this time I've stated PRESS 'L' TO RETURN TO THE GAME" instead of just PRESS A KEY. The reason for this is that if the player is in the middle of the plot and calls up this subroutine then when he RETURNS to the game he needs to have the program display his current position to remind him where he is.

Later on we will be programming "L" or "LOOK" which will be the correct prompt to allow the current Location to be displayed. Thus we need the same prompt here.

(If you don't understand this fully yet — don't worry. It'll be clearer later on when you program the I$ commands).

*Testing Module 7.1*

If you now RUN the program you should find that after the introduction to the story on a brown screen — once you press S the screen changes to grey and the first page of instructions will appear. If you do not press S to start but any other key then the brown screen should stay. (Try it to check — don't just take my word!). You should be able to scan three pages of instructions before the program will end. If you run through these three pages and then press another key at the very end the program will loop back to the beginning of the three pages and go through them again. If you continue to press a key on the second run the program will abort with the error message RETURN WITHOUT GOSUB — which might confuse you. The explanation is that on the first run the program will perform the RETURN in Line

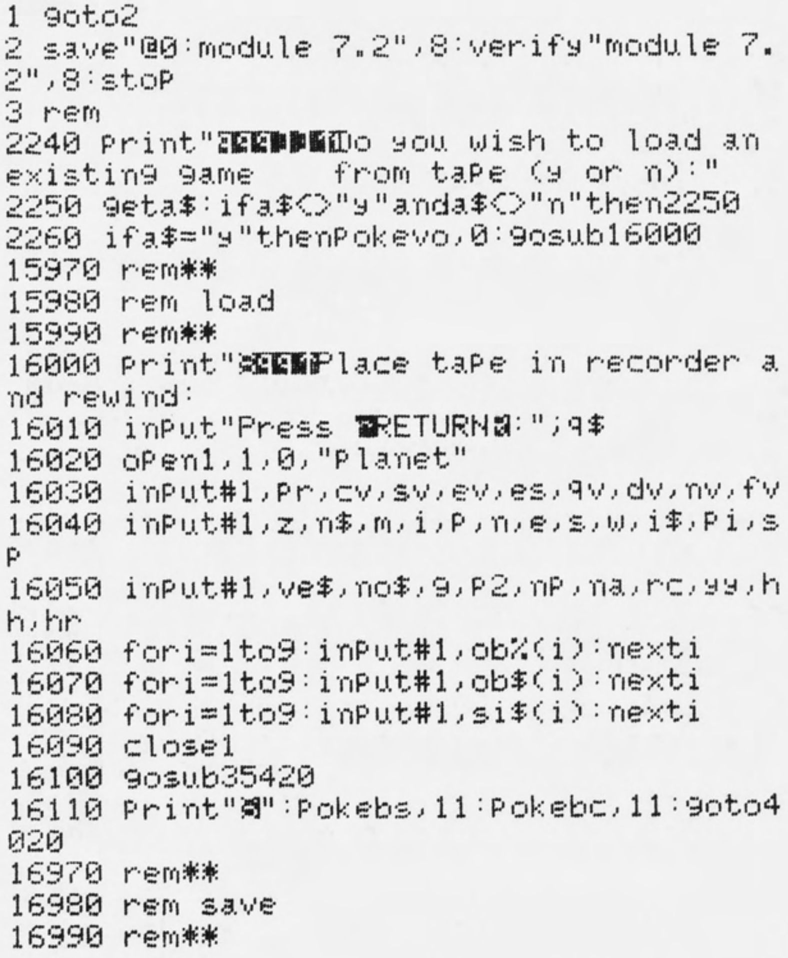
15440 and loop back to 14970 because this is where it left. On the second run though it has not had the GOSUB command in line 2290 so it aborts.

## Loading and Saving

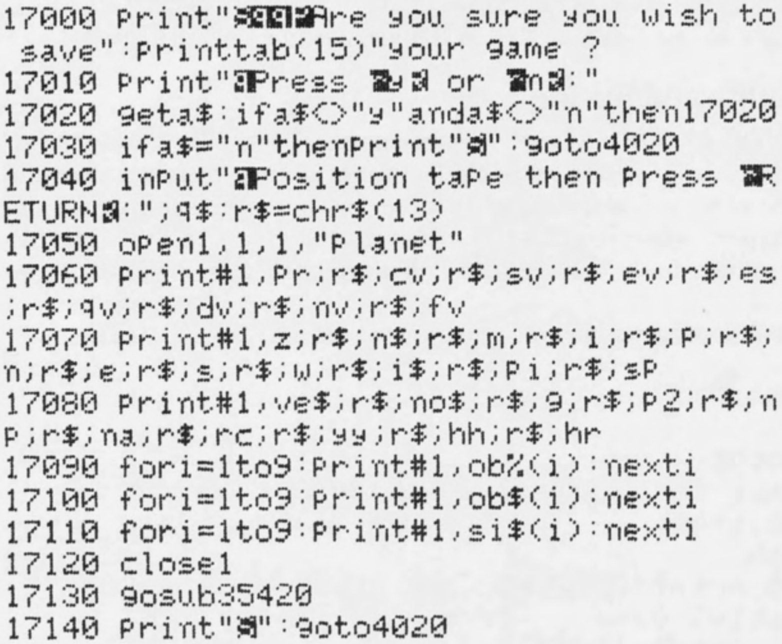
There's no doubt that Adventure games must have the facility for SAVEing a game to tape. Apart from the obvious use of storing your game once you run out of time it's also very handy for saving your game when you reach a critical point where you may be about to lose a life — and SAVEing at this stage will enable you to start again without going right through the Adventure.

The following module will allow you to both SAVE and LOAD.

MODULE 7.2



```
1 goto2
2 save"@0:module 7.2",8:verify"module 7.
2",8:stoP
3 rem
2240 Print"██████Do you wish to load an
existing game     from taPe (y or n):"
2250 geta$:ifa$<>"y"anda$<>"n"then2250
2260 ifa$="y"thenPokevo,0:gosub16000
15970 rem**
15980 rem load
15990 rem**
16000 Print"████Place taPe in recorder a
nd rewind:
16010 inPut"Press █RETURN█:";q$
16020 oPen1,1,0,"Planet"
16030 inPut#1,Pr,cv,sv,ev,es,qv,dv,nv,fv
16040 inPut#1,z,n$,m,i,P,n,e,s,w,i$,Pi,s
P
16050 inPut#1,ve$,no$,g,P2,nP,na,rc,yy,h
h,hr
16060 fori=1to9:inPut#1,ob%(i):nexti
16070 fori=1to9:inPut#1,ob$(i):nexti
16080 fori=1to9:inPut#1,si$(i):nexti
16090 close1
16100 gosub35420
16110 Print"█":Pokebs,11:Pokebc,11:goto4
020
16970 rem**
16980 rem save
16990 rem**
```



```
17000 print"■■■■Are you sure you wish to
 save":printtab(15)"your game ?
17010 print"■Press ■y■ or ■n■:"
17020 geta$:ifa$<>"y"anda$<>"n"then17020
17030 ifa$="n"thenprint"■":goto4020
17040 input"■Position tape then press ■R
ETURN■:";q$:r$=chr$(13)
17050 open1,1,1,"planet"
17060 print#1,pr;r$;cv;r$;sv;r$;ev;r$;es
;r$;qv;r$;dv;r$;nv;r$;fv
17070 print#1,z;r$;n$;r$;m;r$;i;r$;p;r$;
n;r$;e;r$;s;r$;w;r$;i$;r$;pi;r$;sp
17080 print#1,ve$;r$;no$;r$;g;r$;p2;r$;n
p;r$;na;r$;rc;r$;yy;r$;hh;r$;hr
17090 fori=1to9:print#1,ob%(i):nexti
17100 fori=1to9:print#1,ob$(i):nexti
17110 fori=1to9:print#1,si$(i):nexti
17120 close1
17130 gosub35420
17140 print"■":goto4020
```

Lines 2240–2260: A straightforward routine to allow the player to go to the subroutine for loading.

Lines 1600–1610: If you press Y to enter this subroutine then the screen will blank and the statement "PLACE TAPE IN RECORDER AND REWIND:" followed by "PRESS RETURN". These two lines do that. The strange hieroglyphics in Line 16010 is RETURN in shifted mode.

Line 16020: This line OPENs a file to the cassette tape (that means it will allow the data you have created in your program to be recorded on the cassette tape).

The third figure (0) in this line is called the "secondary address" and all this does is tell the cassette to 'write' data. Finally we will give the file the name of "PLANET".

When I started playing Adventure games I thought the cassette would store my game in a sequential fashion, as though the computer had kept a record of my journey and would then write all this information to the cassette as though writing the story so far. Thus if I had spent two hours playing it would have to store twice as much information as if I had been playing for one hour.

What actually happens is much neater — and much more obvious. All the 64 needs to record is the value of all the variables in the program, then on reloading this data it will appear to be in the same position as when you left.

Thus you will be recording a ''file'' of data made up of values of all the variables that we will be discussing later in the book.

Lines 16030–16050: These three lines just store all the variables that will change whilst you are playing onto the tape with the INPUT$ command. It is very important to keep them in the same order as I have, as if we do not SAVE data and then READ it again in exactly the same order the 64 will abort the process.

Lines 16060–16090: Finally we will write all the data from the arrays set up for objects (as in the program Adventure Game in the first section of the book). The last line CLOSES the file.

Line 16100: This line refers to the sprites I will be setting up in the next chapter, but as the principle is important whenever you set up a load and save routine I will mention it here. When you store data for sprites the cassette buffer is used (in other words the chunk of memory that the 64 uses as a temporary store whilst the cassette is recording or playing back data). Thus after performing a SAVE or LOAD routine we must recreate the sprite data in subroutine 35420 before returning to the main program.

Line 16110: If the player has loaded in a program from an earlier game he will need to enter the game so that the screen will display his current position (which will not be back at the beginning). Thus the subroutine needs to have a GOTO not a RETURN or else it will just go back to the start of the story. This line enables the correct procedure although at this stage in your program it won't operate as you haven't entered Line 4020 yet. It also sets the correct screen colour.

Lines 17000–17040: Before allowing the SAVE subroutine to start there is a safety message ''ARE YOU SURE'' in Line 17000. If the player presses Y then the display allows time to place the tape in position and then press ''RETURN'' (note the same hieroglyphics as Line 16010).

Lines 17050–17120: These are virtually the same as in the LOAD routine, except that the secondary address is 1 in Line 17050, PRINT # is used instead of INPUT #, and (most important this) R$ (which equals a carriage return and is set up in Line 17040) is placed between each variable. The reason for this is that the 64 is a little temperamental in saving data and needs this subdivision between all the variables.

Note that the order of variables is the same as Lines 16030–16080.

Lines 17130–17140: Finally we clear the cassette buffer and redefine the sprite data as before, clear the screen, and then GOTO the area in the program where we decided to save. This is necessary in case you want to SAVE the program then continue to play.

*Testing Module 7.2*

Of course at this stage in your program there is nothing to load and nothing to save — but if you just RUN it you should find that after setting the "PRESS S TO START" message you will get another prompt — "DO YOU WISH TO LOAD AN EXISTING GAME FROM TAPE (Y OR N)". Upon pressing "N" you should now move on to the "INSTRUCTIONS" prompt in Module 7.1 whereas pressing Y will take you into the load routine.

The screen will go black, and you will be asked to load a tape into the cassette and press RETURN. On doing this (obviously you don't need to bother with the tape) you should get the PRESS PLAY ON TAPE response. Run Stop will take you out of the program. If this does not happen — something has gone wrong and I suggest you check carefully that you've typed the right numbers after the command OPEN in Lines 16020 and 17050.

When you do actually LOAD and SAVE you will see the screen flashing on and off and the words you typed appearing for a brief moment. This is an idiosyncracy of the 64 and it's best to ignore it. Whatever — don't worry about it.

## Summary

The careful and adequate provision of instructions is an important element of any software — often lacking in even the more expensive packages available. The difficulty is writing enough without taking up too much memory. In *Nightmare Planet* I have provided extra because of the reactions of people who didn't know a thing about Adventures — an excellent audience for your game. You may wish to be more sparing, or even to omit this area if only you and a friend or two will be playing your game.

However you might find that, on returning to your game in a year or more you cannot remember so easily facts that seemed obvious beyond any shadow of a doubt at the time — memory is elusive. Some type of guide is invaluable in these circumstances.

The next stage is to provide sound and graphics — to add the final touches before starting on the story itself.

# CHAPTER 8
# Sprites and Sound Effects

Although I've said that I prefer playing pure text Adventures it was pretty obvious that if I added graphics and sound effects there would be quite a subtle improvement. Obviously there isn't enough memory in the 64 to enable me to create a full Adventure plus pictures and soundtrack but a little programming can go a long long way due to the provision of two features:

1. Sprites.
2. A sound chip with three 'voices'.

There isn't space in a book this size to dwell at length on sprite and sound programming, so I intended describing how I achieved the effects I have used in *Nightmare Planet* and will assume that you can do as I did — read up how to utilise these facilities in the Reference Guide if you aren't already familiar with them.

## Creating your sprites

What exactly are sprites? I think of them as chunks of programmable pixels rather like a large user-defined character. In other words by using a few data statements you can create an object in high resolution that will move around the screen (without all the hassle ϼf POKEing and PEEKing that tends to accompany graphics and without POKEing blank spaces if you want your sprite to move).

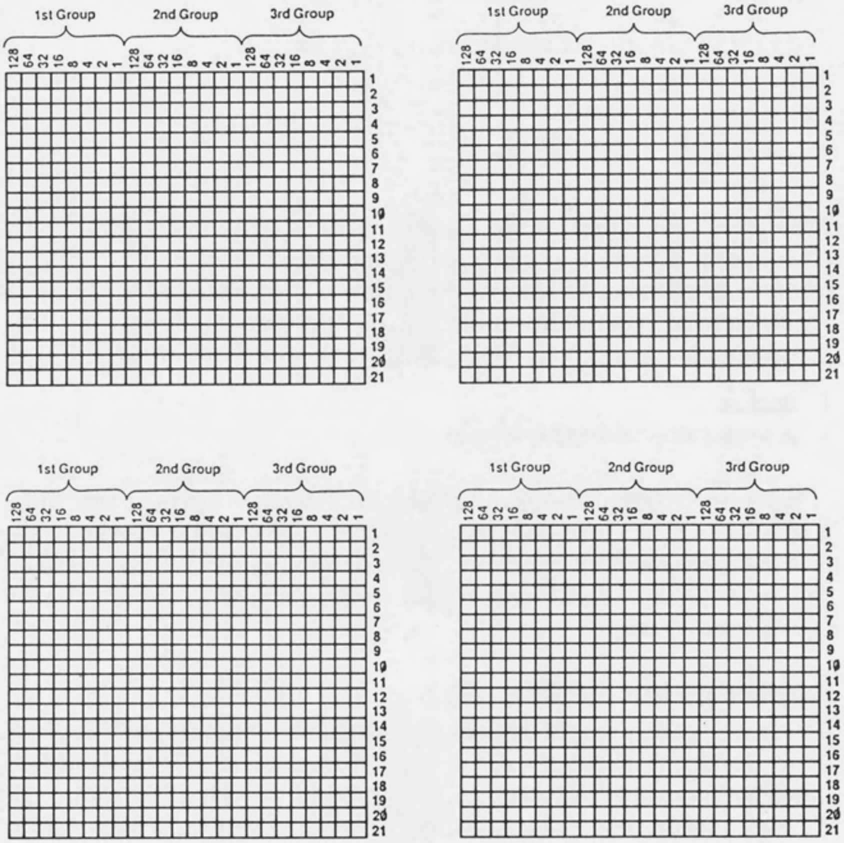
The Commodore Reference Guide sums it up when it says .. (and I quote):

". . . all you have to do is tell a sprite 'what to look like', 'what colour to be' and 'where to appear'. The Vic-II chip will do the rest!"

To recreate a sprite first you need to know what you want it to look like. There are various ways of achieving this but the easiest is to draw it on a 21 × 24 grid as in the User Manual. In **Figure 8.1** I have taken a copy of such a grid and duplicated it four times to allow me to play around with different designs.

Let's look at the development of one of my sprites, Victor, from conception to completion.

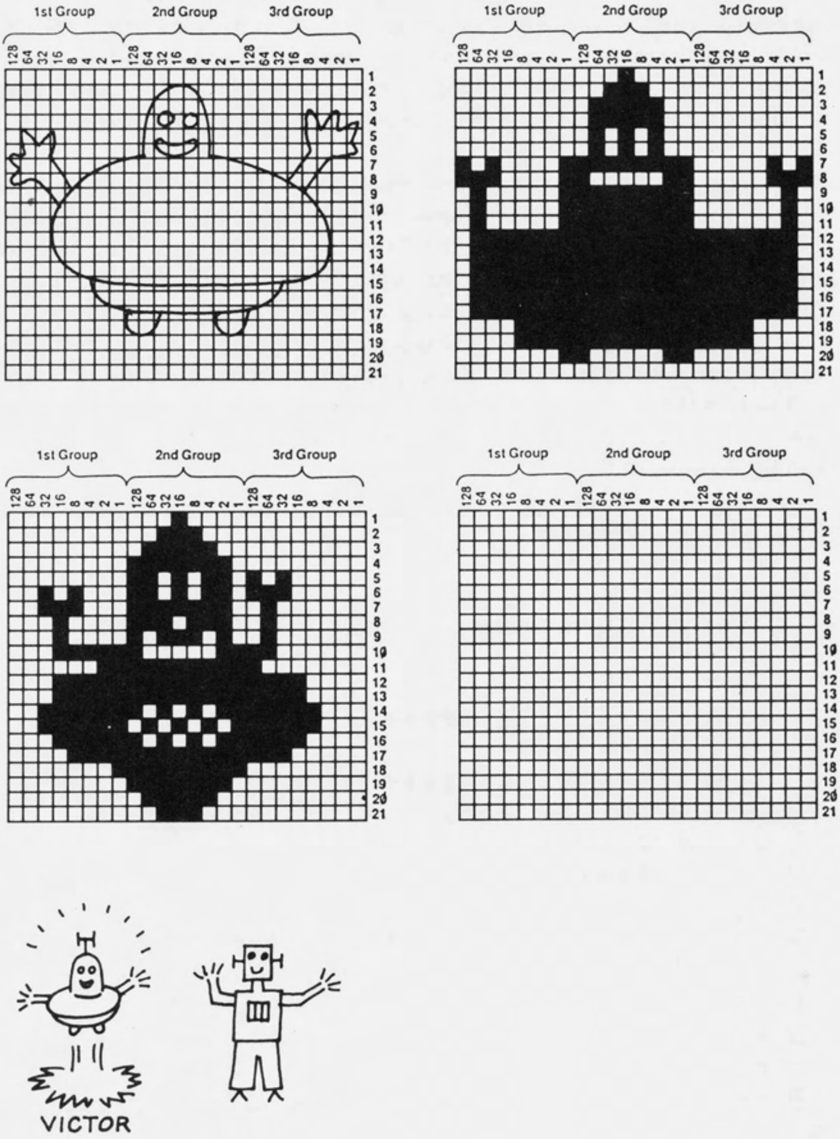
**Figure 8.1 Grids for designing sprites**



If you remember I wanted to have a Personal Robot (who started life named Proteus but somewhere along the way became Victor) appear at times in the story with a message for the player. A sprite seemed the ideal solution — but first I had to design him. **Figure 8.2** shows the various designs, starting with a couple of rough sketches which I then tried to adapt into my 24 × 21 grid. This turned out to be harder than I first thought — as I'd ideas of attempting curves and such — but by the third attempt he was beginning to shape up.

The next stage is to convert your grid into data statements. You can do this manually (this somewhat horrendous task is adequately described in the Reference manual) but I decided it would be better to find a program that would do this for me and ended up using a Sprite Editor program written by A. R. Bennet which I found in a recent copy of *Personal Computer World*. You can also buy ready-made Sprite Editors (Simon's BASIC also contains one) but be sure to have one that will allow you to print out your data once you've drawn your sprite on the screen.

**Figure 8.2 Designing Victor**



I'm pretty sure they all work the same way, by letting you block in your pixels on the screen and then delete as well, so you can play around with your design until you're happy. The one I used also had the facility to expand the sprite horizontally and vertically before you saved the data so you could really see what it would look like.

## Moving your sprite around

Having drawn Victor and found the appropriate data to create him on the screen I decided to run a little test program to set him moving about before entering all the lines into the Adventure program. This has the advantage of seeing whether background colours, sprite colours and size need adjusting before entering the full program, and also gives the feel of sprite manipulation.

Perhaps the biggest problem I found with sprites was their attraction. Having found I could create a character with relative ease and then started to move him all over the place (with 3-D effects, collisions, etc.) a vast range of ideas swept over me, and it was exceedingly hard to stifle my imagination and concentrate on sticking to just one image — the helpful robot sidekick who appears at times of stress and humour throughout the program. The 64 certainly makes life easy when it comes to professional- looking effects.

The first thing you need to do is 'turn the sprite on' or 'enable' it, otherwise although the sprite may be in memory you can't see it. **Program 8.1** is a simple example of this.

**Program 8.1 Spriteman**

```
1 goto3
2 save"@0:spriteman",8:verify"spriteman"
,8:stop
3 rem
7 rem****************************
8 rem initialise
9 rem****************************
10 vc=53248:rem location of video chip
15 print"█"
17 rem****************************
18 rem creation of sprite
19 rem****************************
20 poke vc+21,128:rem lowest level prior
ity (range = 0 (top) - 128 (lowest))
30 poke 2047,13:rem correct location for
 sprite
40 for n=0 to 62:read d:poke 832+n,d:nex
t:rem sprite now created in memory
```

```
45 poke vc+23,128:poke vc+29,128:rem dou
ble x & y size
47 rem****************************
48 rem sprite movement
49 rem****************************
50 poke vc+15,100:rem ´y´ co-ordinate
57 rem****************************
58 rem x movement in 2 stages
59 rem****************************
60 for x=0 to 255:poke vc+14,x:for j=1 t
o 25:next:next
70 poke vc+16,128:for x=0 to 64:poke vc+
14,x:for j=1 to 25:next:next
77 rem****************************
78 rem rem reset sprite
79 rem****************************
80 poke vc+16,0:poke vc+21,0
97 rem****************************
98 rem rem data
99 rem****************************
100 data 15, 129, 240, 0, 129, 0, 0, 129
, 0, 255, 255
110 data 255, 255, 255, 255, 192, 231, 3
, 192, 231, 3, 255
120 data 231, 255, 255, 231, 255, 255, 2
31, 255, 255, 231, 255
130 data 192, 0, 3, 192, 0, 3, 192, 0, 3
, 255, 255
140 data 255, 255, 255, 255, 1, 128, 192
, 1, 128, 192, 1
150 data 128, 192, 1, 128, 192, 7, 128,
240
```

This program creates a little man (a little bit like Victor) with the DATA statements in Lines 100–150 and then turns him on (Line 20) and prepares a block of memory (Line 30). Line 40 reads the DATA and puts it into the memory (remember this is the cassette buffer) and Line 45 doubles the sprite in size both vertically and horizontally.

Line 50 places our spriteman in a position down the screen and then Lines 60 and 70 move him across the screen. Finally Line 80 turns him off again.

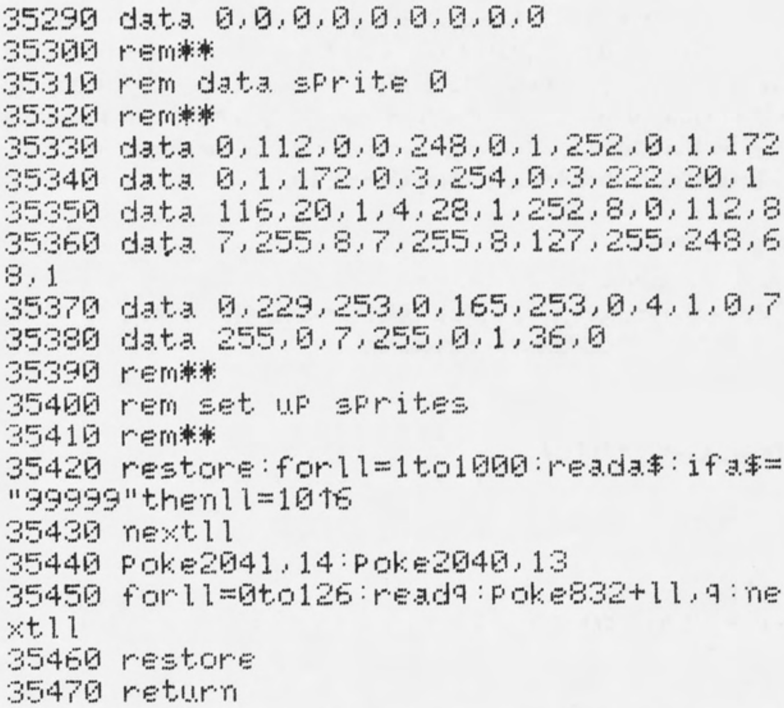
I have been brief to the extreme in this explanation because it's basically a straight crib from either the User Manual or Reference Manual and you can read it up yourself if you wish.

## Victor and the Mealy Bug

I mentioned in Chapter 6 that I wanted a sprite to move across the title *Nightmare Planet* as well as having Victor (who would appear at various times during the program). Having drawn Victor on my grid I just played around with my Sprite Editor until I came up with a slightly evil insect-like object that looked ideal for moving across the credits — which I named a Mealy Bug. The time had come to program these two into our Adventure.

MODULE 8.1

```
1 goto3
2 save"@0:module 8.1",8:verify"module 8.
1",8:stoP
3 rem
34970 rem**
34980 rem mealy bug
34990 rem**
35000 Pokev+39,2:Pokev+27,1:Pokev+23,1:P
okev+29,1
35010 Pokev+1,100:Pokev,0:Pokev+21,1
35020 forx=0to255:Pokev,x:nextx:Pokev+16
,1
35030 forx=0to95:Pokev,x:nextx:Pokev+16,
0:Pokev+21,0
35040 return
35070 rem**
35080 rem victor
35090 rem**
35140 Pokev+40,4:Pokev+29,2:Pokev+23,2
35150 Pokev+2,80:Pokev+3,0:Pokev+21,2
35160 fory=0to190:Pokev+3,y:nexty
35180 return
35200 data 99999
35210 rem**
35220 rem data sPrite 1
35230 rem**
35240 data 255,128,0,255,128,0,1,128,0,1
,128
35250 data 0,3,252,0,31,255,224,255,252,
32,63
35260 data 252,63,63,255,255,224,0,1,63,
255,240
35270 data 63,255,240,3,1,0,3,25,0,3,31
35280 data 0,227,0,0,255,0,0,0,0,0,0
```



```
35290 data 0,0,0,0,0,0,0,0,0
35300 rem**
35310 rem data sPrite 0
35320 rem**
35330 data 0,112,0,0,248,0,1,252,0,1,172
35340 data 0,1,172,0,3,254,0,3,222,20,1
35350 data 116,20,1,4,28,1,252,8,0,112,8
35360 data 7,255,8,7,255,8,127,255,248,6
8,1
35370 data 0,229,253,0,165,253,0,4,1,0,7
35380 data 255,0,7,255,0,1,36,0
35390 rem**
35400 rem set uP sPrites
35410 rem**
35420 restore:forll=1to1000:reada$:ifa$=
"99999"thenll=10↑6
35430 nextll
35440 Poke2041,14:Poke2040,13
35450 forll=0to126:readq:Poke832+ll,q:ne
xtll
35460 restore
35470 return
```

Lines 35000–35040: The variable V in all the sprite programming refers to the starting address of the Video Chip (53248) and was defined in Line 280 in Module 6.1. Thus these lines 'enable' the sprite, define his colour and size, and move him across the screen from left to right whereupon he disappears. It is this subroutine that keeps the initial title page on the screen for a little longer.

Lines 35140–35180: These lines do exactly the same thing for Victor, except that he moves in a horizontal instead of a vertical direction so that he appears to drop from above into the screen and stops about two thirds of the way down on the left hand side.

Line 35200: This line creates the DATA figure of "99999" to allow the program to distinguish between the sprite data and the data we will be programming later with regard to the objects that will be scattered around the Adventure.

Lines 35240–35380: The DATA for the two sprites

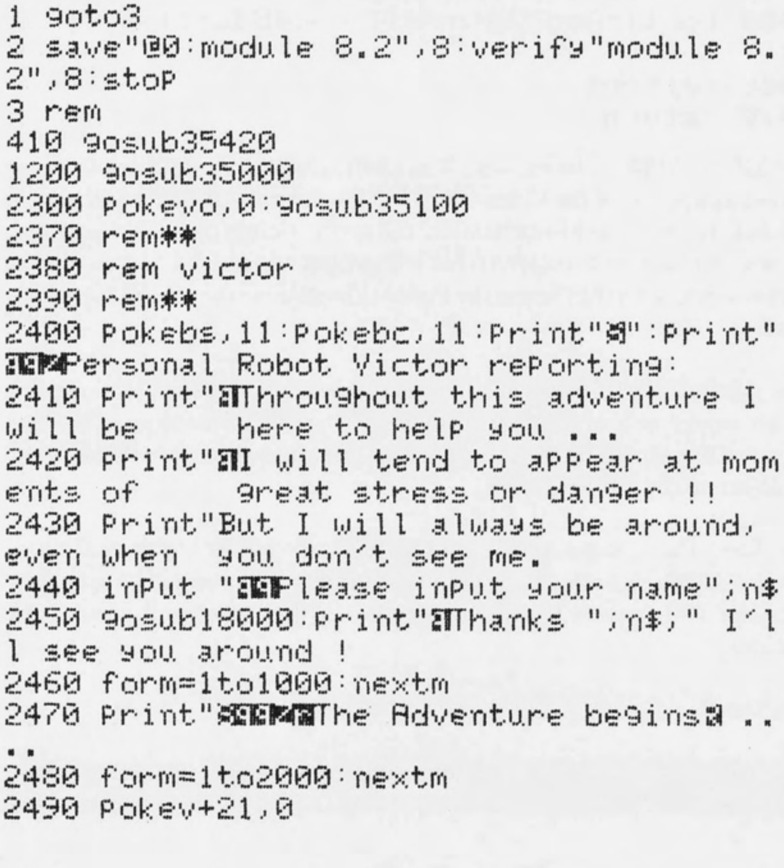
Lines 35420-35470: The section is the initial subroutine to set up the sprites. First the 64 RESTORES itself to start reading DATA afresh. Then we check for the figure 99999 (in Line 35200 remember) which prevents the string data in the other part of the program from being READ in error and creating a syntax error message. Finally the DATA statements for the sprites are READ and placed into the cassette buffer memory. This is the subroutine that causes the computer to appear to pause right at the beginning of the program and also after LOADing and SAVEing.

*Testing Module 8.1*

If you do try to test this module nothing will appear to happen because although we have created our sprites there is nothing in the program to call them up. The next module will correct that.

## Introducing Victor

MODULE 8.2



```
1 goto3
2 save"@0:module 8.2",8:verify"module 8.
2",8:stop
3 rem
410 gosub35420
1200 gosub35000
2300 pokevo,0:gosub35100
2370 rem**
2380 rem victor
2390 rem**
2400 pokebs,11:pokebc,11:print"▨":print"
▨▨▨Personal Robot Victor reporting:
2410 print"▨Throughout this adventure I
will be      here to help you ...
2420 print"▨I will tend to appear at mom
ents of      great stress or danger !!
2430 print"But I will always be around,
even when  you don't see me.
2440 input "▨▨Please input your name";n$
2450 gosub18000:print"▨Thanks ";n$;" I'l
l see you around !
2460 form=1to1000:nextm
2470 print"▨▨▨▨▨The Adventure begins▨ ..
..
2480 form=1to2000:nextm
2490 pokev+21,0
```

Line 410: This is the initial GOSUB which sets up the sprites at the very beginning of the program. It is the same as the GOSUBs in Lines 16100 and 17130 in Module 7.2.

Line 1200: We are calling up the Mealy Bug in the main title with this line.

Line 2300: This line turns off the sound first (as Victor can be called at any time in the program I wanted to be sure any existing sound was turned off before calling him) with the instruction POKE VO,0. Then GOSUB 35100 will call Victor.

Lines 2400–2440: As Victor arrives on the scene this message appears with him. Following his message you are asked to input your name (given the variable N$).

Lines 2450–2490: The first GOSUB in Line 2450 is covered in Module 8.4 (in a moment) and refers to a sound routine. I have tried to add a beep or a burp noise whenever the 64 responds to the input. The beep is for an okay reply and the burp for a not okay reply.

Line 2460 is a short pause loop, followed by the screen clearing and the words "THE ADVENTURE BEGINS . . ." appearing at the top of the screen. There is another short pause (Line 2480) then Victor disappears (Line 2490 which turns him off).

*Testing Module 8.2*

The title is beginning to shape up at last. If you run it now there is a significant pause whilst the pre-credit sequence is on the screen (whilst the sprites are set up) then the Mealy Bug will travel sedately across the screen behind the purple border and the words "NIGHTMARE PLANET".

However if you move through the program you won't actually get to Victor yet — just the message UNDEF'D STATEMENT ERROR IN 2300. This is because the Victor subroutine is about to gain sound, and Line 35100 does not exist yet.

## Now you hear it . . .

The addition of sound to a program is like watching a colour film in black and white first before buying your colour television — in other words fantastic. Of course after a while the sound begins to be taken for granted, but after keying in the following modules I promise you that you'll notice a terrific difference between the program before and after this section.

I'll start by stating that I am in no way going to get involved in an explanation of sound and the sound chip. Again your Manual or Reference Guide both contain plenty of information — so all I'll do is run through how I achieved the effects I used.

I did not attempt any music itself — partly my own musical inability and partly a lack of time. If you are musical I would imagine a short introductory tune would be even better than my own soundtrack.

I began to play around with sound by using some of the example programs in the User Manual but very soon came to the conclusion that I needed a simple program of my own which would allow me to substitute different values for the various waveforms, frequencies, etc. of all three voices of the 64. So it was necessary to convert the various values of the different variables in the sound chip to simple figures, and I made my own rather impressive-looking table of variables (See **Table 8.1**).

**Table 8.1: The Sound Variables**

| | |
|---|---|
| VO | Volume of the sound (VO = 54296) |
| W1 | Waveform of Voice 1 (W1 = 54276) |
| W2 | Waveform of Voice 2 (W2 = 54283) |
| W3 | Waveform of Voice 3 (W3 = 54290) |
| A1 | Attack/Decay for Voice 1 (A1 = 54277) |
| A2 | Attack/Decay for Voice 2 (A2 = 54284) |
| A3 | Attack/Decay for Voice 3 (A3 = 54291) |
| S1 | Sustain/Release for Voice 1 (S1 = 54278) |
| S2 | Sustain/Release for Voice 2 (S2 = 54285) |
| S3 | Sustain/Release for Voice 3 (S1 = 54292) |
| H1 | High Frequency for Voice 1 (H1 = 54273) |
| H2 | High Frequency for Voice 2 (H2 = 54280) |
| H3 | High Frequency for Voice 3 (H1 = 54287) |
| L1 | Low Frequency for Voice 1 (L1 = 54272) |
| L2 | Low Frequency for Voice 2 (L2 = 54279) |
| L3 | Low Frequency for Voice 3 (L3 = 54286) |

Before returning to *Nightmare Planet* I'll just briefly describe my program because this should not only help you to understand some of the sounds I've found, but also improve on them in your own programs.

**Program 8.2 Sound Effect**

```
1 goto3
2 save"@0:sound effect",8:verify"sound e
ffect",8:stop
3 rem
5 rem***********************************
6 rem sound effect program
7 rem mike grace 24.7.83
8 rem***********************************
10 sc=54272
20 w1=54276:rem waveform for voice 1
30 w2=54283:rem waveform for voice 2
```

```
40 w3=54290:rem waveform for voice 3
50 a1=54277:rem attack/decay for voice 1
60 a2=54284:rem attack/decay for voice 2
70 a3=54291:rem attack/decay for voice 3
80 s1=54278:rem sustain/release for voic
e 1
90 s2=54285:rem sustain/release for voic
e 2
100 s3=54292:rem sustain/release for voi
ce 3
110 h1=54273:rem high frequency for voic
e 1
120 h2=54280:rem high frequency for voic
e 2
130 h3=54287:rem high frequency for voic
e 3
140 l1=54272:rem low frequency for voice
 1
150 l2=54279:rem low frequency for voice
 2
160 l3=54286:rem low frequency for voice
 3
170 vo=54296:rem volume
200 print"hit any key"
210 geta$:ifa$=""then210
300 forl=0to24:pokesc+l,0:next:rem clear
 all variables
320 pokevo,15:rem set volume
330 pokea1,17:pokea2,64:pokea3,17:rem se
t attack/decay
340 pokes1,128:pokes2,128:pokes3,128:rem
 set sustain/release
350 pokew1,33:pokew2,33:pokew3,129:rem s
et wavelengths
360 pokeh1,7:pokel1,0:rem frequencies fo
r voice 1
370 pokeh2,3:pokel2,0:rem frequencies fo
r voice 2
380 pokeh3,1:pokel3,0:rem frequencies fo
r voice 3
400 fort=1to800:next:pokevo,0
450 print"space bar twice to continue"
455 print"'p' to print - 'e' to end"
```

```
460 geta$:ifa$<>"p"anda$<>"e"anda$<>" "t
hen460
470 ifa$="p"then600
480 ifa$="e"thenend
490 ifa$=" "then210
600 input"title";t$
610 open4,4
620 print#4,chr$(14):rem double width co
mmand
630 print#4,t$:rem print title
640 print#4,chr$(15):rem return to stand
ard width
650 cmd4:list 330-380
700 print"remember to type print#4:close
4"
900 end
```

Lines 10–170: Setting up the sound variables.

Lines 200–210: This little couplet just prints the message "HIT ANY KEY" and waits for you to hit that key whereupon the sound will be played. It is purely a device for allowing you to sound the note when you wish.

Line 300: Before making any noise this line sets up a loop to go right through the range of variables in the sound chip turning them all to 0. It is simply a way of starting afresh — and I use this line several times in *Nightmare Planet*.

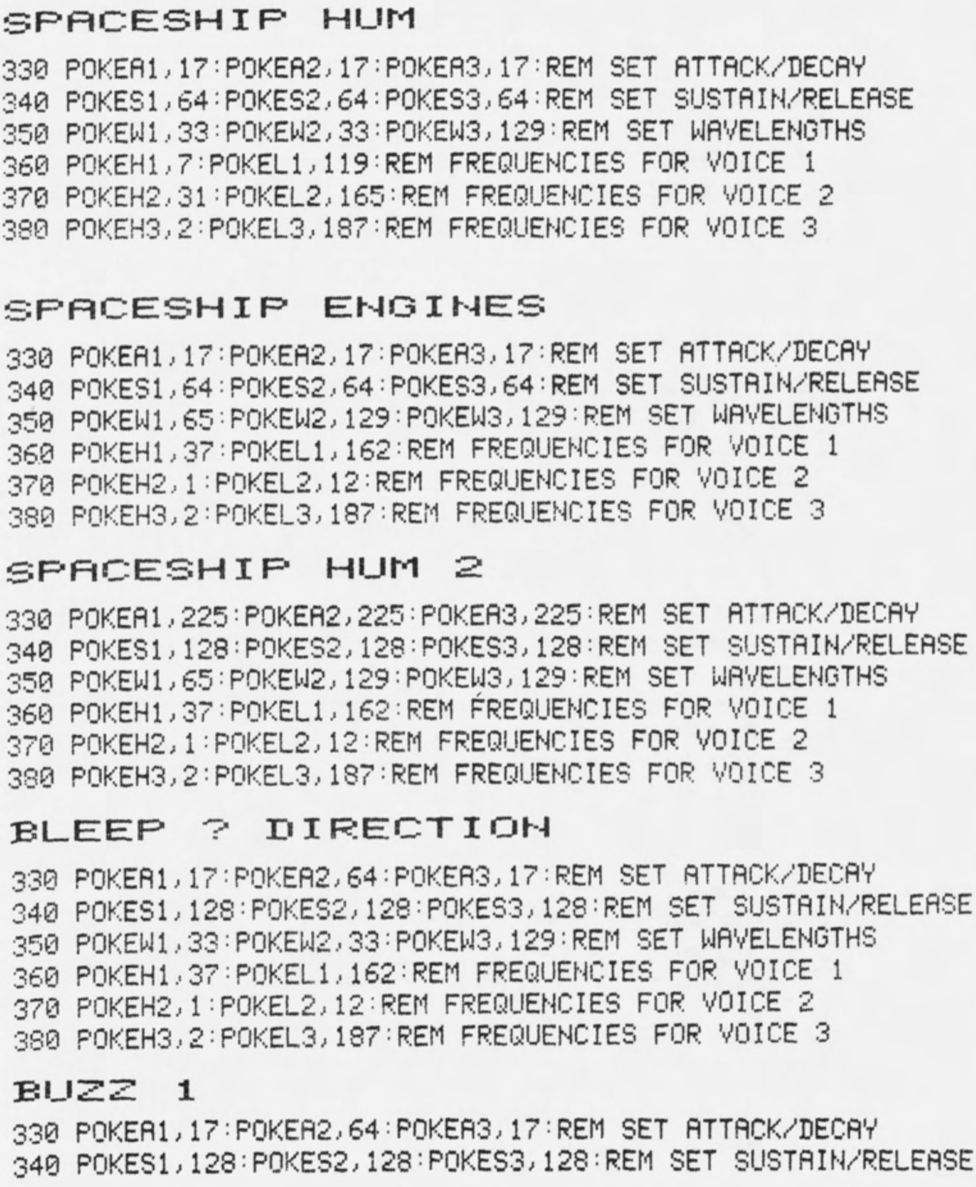
Lines 320–380: As the REM statements say in the lines themselves these set the various values. All you have to do is LIST the program and change the values, then RUN it again to listen to the different noise. By a system of trial and error you will eventually come upon a sound that gives you the effect you want.

Line 400–900: Line 400 gives the length of the note. Lines 450–460 are a very simple menu to allow you to repeat the sound or end the program or print out a LISTING which will enable you to keep a record of the values. Lines 600–650 are instructions to the printer (in my case the 1515) and Line 700 is not actually of any use apart from reminding me that I must type "PRINT #4:CLOSE4" at the end of the LISTING.

This program isn't particularly brilliant and I wrote it in about 15 minutes. But it does the job that I wanted it to — and whilst I could have spent time and effort making it a superb piece of programming with fancy displays and better menus, etc., I really only wanted it for a quick bit of help. As such it serves me well. The only reason I put it here is to give you an idea of how I achieved my own sound.

The printout for some of the effects I played with can be seen in **Figure 8.3**.

**Figure 8.3 Some sample sound effects**



SPACESHIP HUM

```
330 POKEA1,17:POKEA2,17:POKEA3,17:REM SET ATTACK/DECAY
340 POKES1,64:POKES2,64:POKES3,64:REM SET SUSTAIN/RELEASE
350 POKEW1,33:POKEW2,33:POKEW3,129:REM SET WAVELENGTHS
360 POKEH1,7:POKEL1,119:REM FREQUENCIES FOR VOICE 1
370 POKEH2,31:POKEL2,165:REM FREQUENCIES FOR VOICE 2
380 POKEH3,2:POKEL3,187:REM FREQUENCIES FOR VOICE 3
```

SPACESHIP ENGINES

```
330 POKEA1,17:POKEA2,17:POKEA3,17:REM SET ATTACK/DECAY
340 POKES1,64:POKES2,64:POKES3,64:REM SET SUSTAIN/RELEASE
350 POKEW1,65:POKEW2,129:POKEW3,129:REM SET WAVELENGTHS
360 POKEH1,37:POKEL1,162:REM FREQUENCIES FOR VOICE 1
370 POKEH2,1:POKEL2,12:REM FREQUENCIES FOR VOICE 2
380 POKEH3,2:POKEL3,187:REM FREQUENCIES FOR VOICE 3
```

SPACESHIP HUM 2

```
330 POKEA1,225:POKEA2,225:POKEA3,225:REM SET ATTACK/DECAY
340 POKES1,128:POKES2,128:POKES3,128:REM SET SUSTAIN/RELEASE
350 POKEW1,65:POKEW2,129:POKEW3,129:REM SET WAVELENGTHS
360 POKEH1,37:POKEL1,162:REM FREQUENCIES FOR VOICE 1
370 POKEH2,1:POKEL2,12:REM FREQUENCIES FOR VOICE 2
380 POKEH3,2:POKEL3,187:REM FREQUENCIES FOR VOICE 3
```

BLEEP ? DIRECTION

```
330 POKEA1,17:POKEA2,64:POKEA3,17:REM SET ATTACK/DECAY
340 POKES1,128:POKES2,128:POKES3,128:REM SET SUSTAIN/RELEASE
350 POKEW1,33:POKEW2,33:POKEW3,129:REM SET WAVELENGTHS
360 POKEH1,37:POKEL1,162:REM FREQUENCIES FOR VOICE 1
370 POKEH2,1:POKEL2,12:REM FREQUENCIES FOR VOICE 2
380 POKEH3,2:POKEL3,187:REM FREQUENCIES FOR VOICE 3
```

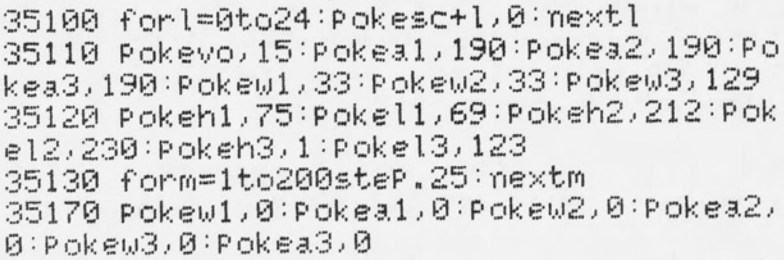
BUZZ 1

```
330 POKEA1,17:POKEA2,64:POKEA3,17:REM SET ATTACK/DECAY
340 POKES1,128:POKES2,128:POKES3,128:REM SET SUSTAIN/RELEASE
```

```
350 POKEW1,33:POKEW2,33:POKEW3,129:REM SET WAVELENGTHS
360 POKEH1,7:POKEL1,0:REM FREQUENCIES FOR VOICE 1
370 POKEH2,3:POKEL2,0:REM FREQUENCIES FOR VOICE 2
380 POKEH3,1:POKEL3,0:REM FREQUENCIES FOR VOICE 3
```

MODULE 8.3

```
1 goto3
2 save"@0:module 8.3",8:verify"module 8.
3",8:stop
3 rem
330 rem**
340 rem sound effect
350 rem**
360 forl=0to24:pokesc+l,0:nextl
370 pokevo,15:pokea1,255:pokea2,255:poke
a3,255
380 pokes1,255:pokes2,255:pokes3,255
390 pokew1,17:pokew2,33:pokew3,17
400 pokeh1,37:pokel1,162:pokeh2,1:pokel2
,12:pokeh3,2:pokel3,187
1230 pokevo,0
2000 rem**
2010 rem pregame
2020 rem**
2030 forl=0to24:pokesc+l,0:nextl
2040 pokevo,15:pokea1,255:pokea2,255:pok
ea3,255
2050 pokes1,128:pokes2,128:pokes3,128:po
kew1,65:pokew2,129:pokew3,129
2060 pokeh1,37:pokel1,162:pokeh2,1:pokel
2,12:pokeh3,2:pokel3,187
2120 ifa$="s"thenpokevo,0
2130 form=1to150:nextm
2160 rem crash
2170 forl=0to24:pokesc+l,0:nextl
2180 pokevo,15:pokea1,140:pokea2,140:pok
ea3,140:pokes1,0:pokes2,0:pokes3,0
2190 pokew1,33:pokew2,129:pokew3,129
2200 pokeh1,37:pokel1,162:pokeh2,1:pokel
2,12:pokeh3,2:pokel3,187
2210 print"▮":pokebs,0:pokebc,0
2220 print"▮▮▮▮▮▮▮I think you may have c
rashed !!!!"
2230 form=1to1000:nextm
```



```
35100 forl=0to24:Pokesc+l,0:nextl
35110 Pokevo,15:Pokea1,190:Pokea2,190:Po
kea3,190:Pokew1,33:Pokew2,33:Pokew3,129
35120 Pokeh1,75:Pokel1,69:Pokeh2,212:Pok
el2,230:Pokeh3,1:Pokel3,123
35130 form=1to200steP.25:nextm
35170 Pokew1,0:Pokea1,0:Pokew2,0:Pokea2,
0:Pokew3,0:Pokea3,0
```

Line 360: As I've already said this line clears all the variables in the sound chip.

Lines 370–400: This is the first sound you hear upon RUNning the program, and I wanted something slightly eerie and futuristic — rather similar to the theme that began that superb SF film *Alien* a few years back. My efforts fall rather short of this, but I like to imagine there is an element of slight unease in the noise that accompanies the opening title.

By giving the value 255 to all three voices for the Attack/Decay and Sustain/Release variables (see Lines 370 and 380) I allow the sound to rise slowly from silence towards its peak. If the title lasted long enough the sound would decay again to silence, but as the second screen appears before this there is a cut-off as the screen switches from black to brown and the first text comes up.

Line 1230: This is the line that cuts the opening noise as the screen switches to the first text by POKEing VO (the volume) to 0.

Lines 2030–2060: Once the text comes up I wanted a sound like the thrum of a spaceship engine which would cut to silence as you press S to start the program. These lines provide that noise. Again note the slow rise towards a peak (in Line 2040 the Attack variables A1, A2 and A3 are set at 255 again) but this time I keep the volume by setting the Sustain Level high with very little Release (the value 128 for the variables S1, S2 and S3 in Line 2050).

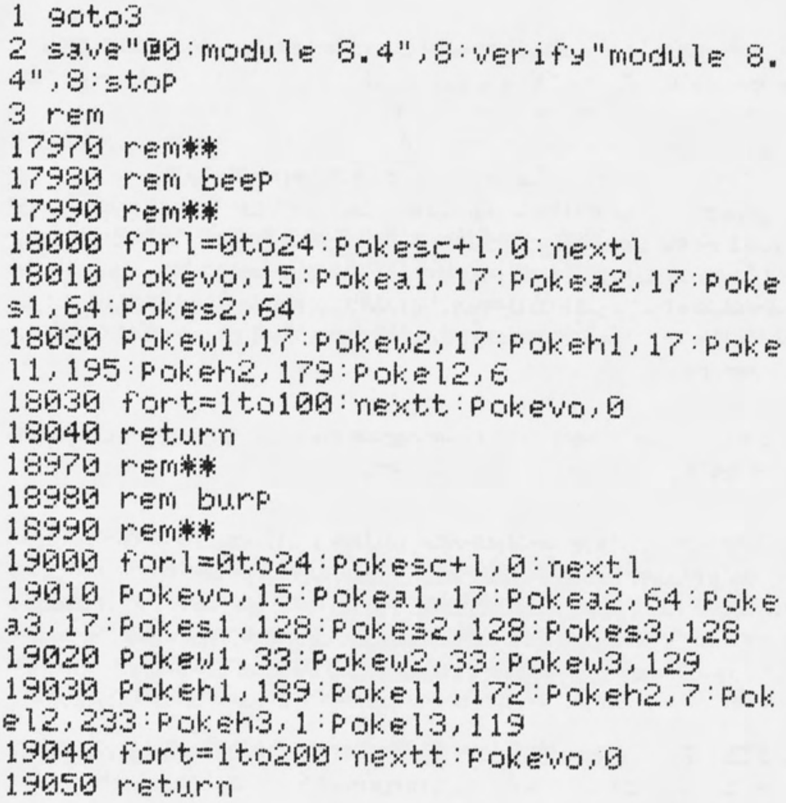
Lines 2120–2130: These two lines fill in after the "PRESS S TO START" prompt, by cutting the sound after you press "S" and then entering a tiny pause loop.

Lines 2170–2230: Once the player presses S the screen turns black again, there is the noise of a crash, and the words "I THINK YOU MAY HAVE CRASHED" come up. They last for a few seconds before the prompts about loading to tape and going for instructions follow (which we covered in Modules 7.1 and 7.2). These lines provide that little scenario.

Lines 35100–35170: The last lines in this module concern Victor. I wanted him to appear with a sound effect rather like a robot dropping down from above. By chance I think I've hit upon an ideal sound (it was more luck than anything else). The main difference between this sound routine and the others so far is in Line 35130 where there is a little FOR. . NEXT loop which lowers the sound as Victor drops down.

Before testing this module let's just finish programming as the final module in this chapter puts the finishing touches on the opening sequence.

MODULE 8.4



```
1 goto3
2 save"@0:module 8.4",8:verify"module 8.
4",8:stop
3 rem
17970 rem**
17980 rem beep
17990 rem**
18000 forl=0to24:pokesc+l,0:nextl
18010 pokevo,15:pokea1,17:pokea2,17:poke
s1,64:pokes2,64
18020 pokew1,17:pokew2,17:pokeh1,17:poke
l1,195:pokeh2,179:pokel2,6
18030 fort=1to100:nextt:pokevo,0
18040 return
18970 rem**
18980 rem burp
18990 rem**
19000 forl=0to24:pokesc+l,0:nextl
19010 pokevo,15:pokea1,17:pokea2,64:poke
a3,17:pokes1,128:pokes2,128:pokes3,128
19020 pokew1,33:pokew2,33:pokew3,129
19030 pokeh1,189:pokel1,172:pokeh2,7:pok
el2,233:pokeh3,1:pokel3,119
19040 fort=1to200:nextt:pokevo,0
19050 return
```

Lines 18000–18040: This sound effect is a pip — or beep noise. Note the different values for the A variables (Attack/Decay) and S variables (Sustain/Release) which are concentrating on a quick rise and sharp fall type of note rather than the slow rise and fall of the earlier sounds. I also only used two voices for this note, and kept it short (FOR T = 1 TO 100). The purpose of this subroutine is to sound that quick, slightly pleasant noise to let the player know the 64 has responded.

Lines 19000–19050: This subroutine provides the slightly longer burp noise (it's more of a musical burp actually) that tells the player he has made a boob! I reserve this note for error messages or when you've done something stupid. This time I used all three voices again and included the white noise waveform (POKE W3,129 in Line 19020) to add to the effect.

*Testing Modules 8.3 and 8.4*

This is the moment.

Lean back, take in the screen, and type RUN.

You should find a vast difference in your program now. First the slightly uneasy noise as the titles flash up and the Mealy Bug flies quietly across your screen. Then the deep rumble of your engines as your spaceship wings its way through space. The sudden cut-off as you start the game — followed by the crash as your ship pancakes onto the surface of Nightmare Planet. A pause, then you respond with N to the prompts for tape and instructions — and Victor appears (the first time you will have seen him) with a noise you'll come to recognise. He drops into place, the sound cuts off, and after reading his message you input your name. There is a pleasant beep noise — Victor replies and the screen clears with the message THE ADVENTURE BEGINS .... Then Victor disappears.

With the program in its present state you should then get the Instructions Pages again as the program will naturally follow to this part. But I hope you are beginning to feel some of the effort I've put into building up a little suspense into the beginning of the game.

## Summary

I have tended to be very brief in my description of the mechanics of programming this chapter, mainly through lack of space. Using sprites and sound isn't as difficult as the Reference Guide makes it look I found and I encourage those of you who don't take naturally to PEEKing and POKEing to sit down for a couple of hours with your Guide and your 64 and just work through it — it will come. The capabilities of the computer really are extended once you've conquered the first problem of learning any new skill — taking that first step.

The titles are finished and you are set to go — to get into the program. Chapter 9 will turn out to be the marathon of the book — the Locations. But take heart ... the best is yet to come.

# CHAPTER 9
# Where do we go from here?

This chapter is probably the most exciting and the most annoying part of the book at the same time. Exciting — because after you have finished programming Modules 9.1 and 9.2 you will be able to see the entire scope of the Adventure and be able to move around all the locations. Annoying because there's a lot of typing as we are now into the text itself.

There's no easy way to avoid all this programming — but I find that keying in all the lines of text isn't as bad as it looks as writing words is much easier than writing BASIC. Still, that's my problem!

## Creating the map

I've already covered the principles of maps in Chapter 2 and drawn the basic map for *Nightmare Planet* in **Figure 5.2** in Chapter 5. But the real Adventure needs a better map than this — one with plenty of Locations to add a breadth to the story and to make the player feel he really is moving around a large space.

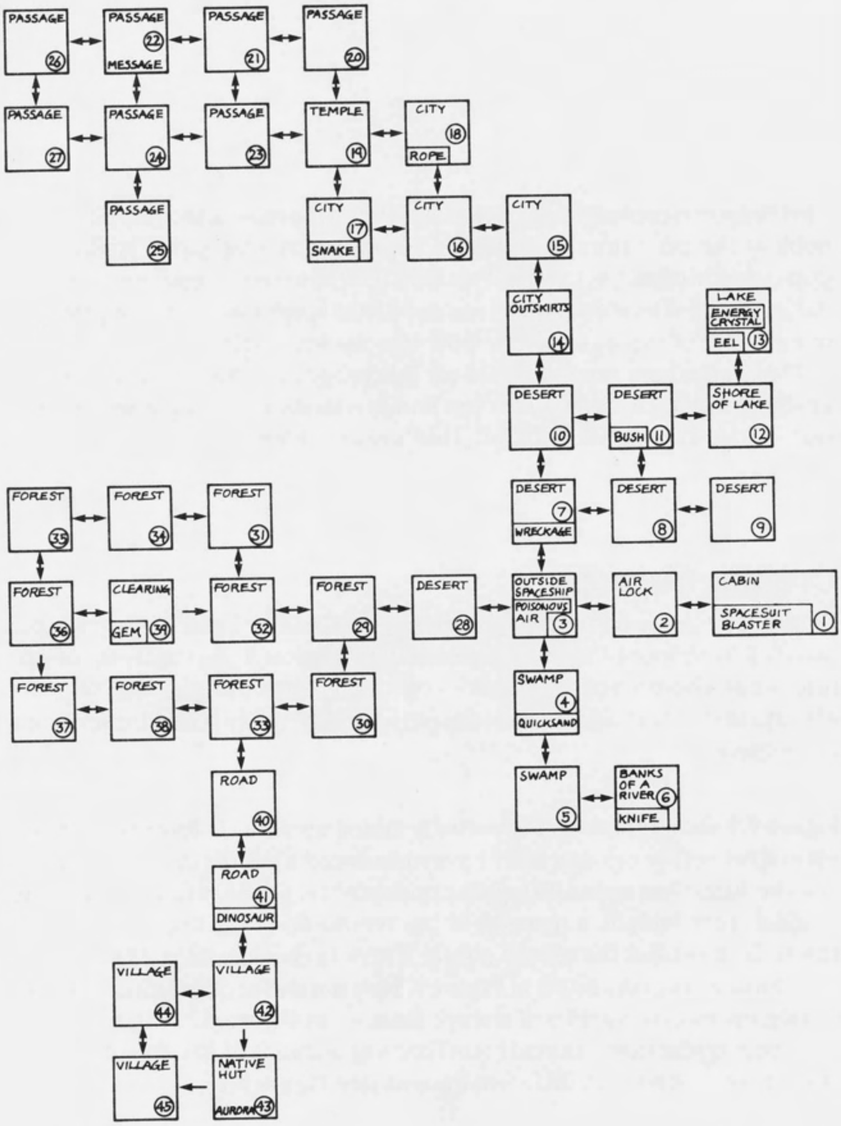
**Figure 9.1** shows the map I eventually ended up with. It follows the basic plan of my earlier attempt but I have introduced a temple in the ruined city and the forest has expanded quite considerably. Other refinements to the original story include a message in the temple (Box 22) and a gem in the forest (Box 38) and the energy crystal is now in the lake (Box 13).

The numbers in the boxes in **Figure 9.1** are purely for convenience as converting my map to a grid will change these — as **Figure 9.3** will show.

To convert the map to a grid form I drew up a standard 10x10 grid to allow me to play around with different designs (see **Figure 9.2**).

Transferring the original map to the grid was virtually like lifting the map and placing it straight onto the grid — I only made a few changes. Once the final map was ready then it was just a question of placing the appropriate objects into their Locations and I was ready to begin programming again.

**Figure 9.1 The final map for Nightmare Planet**

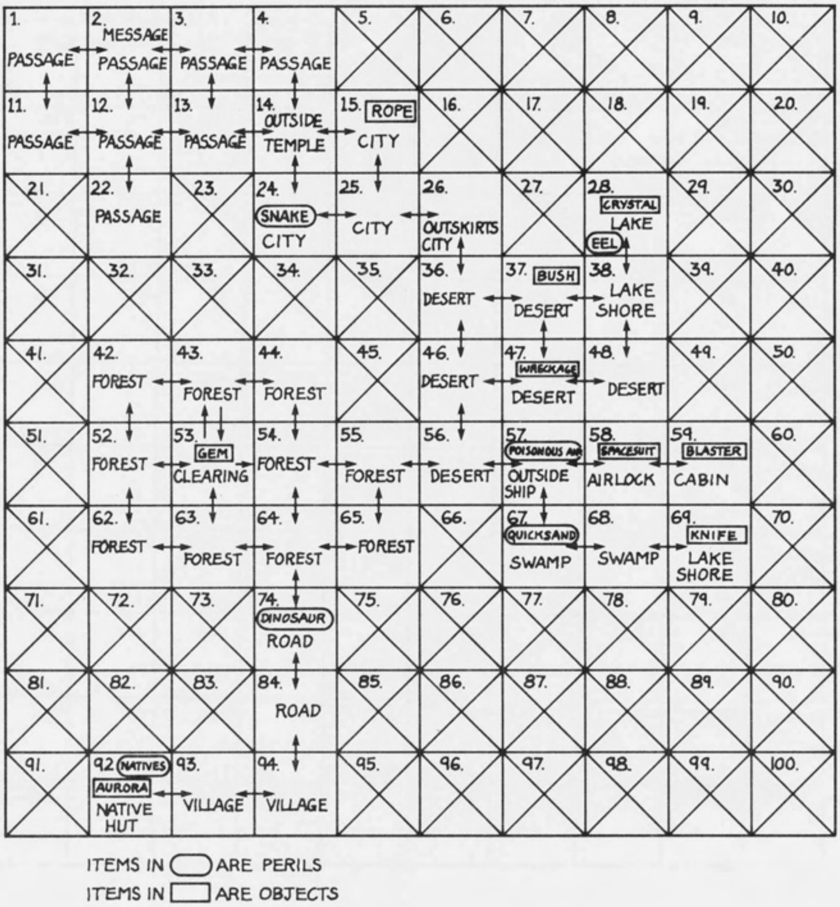


## Figure 9.2 A 10 x 10 grid

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1. | 2. | 3. | 4. | 5. | 6. | 7. | 8. | 9. | 10. |
| 11. | 12. | 13. | 14. | 15. | 16. | 17. | 18. | 19. | 20. |
| 21. | 22. | 23. | 24. | 25. | 26. | 27. | 28. | 29. | 30. |
| 31. | 32. | 33. | 34. | 35. | 36. | 37. | 38. | 39. | 40. |
| 41. | 42. | 43. | 44. | 45. | 46. | 47. | 48. | 49. | 50. |
| 51. | 52. | 53. | 54. | 55. | 56. | 57. | 58. | 59. | 60. |
| 61. | 62. | 63. | 64. | 65. | 66. | 67. | 68. | 69. | 70. |
| 71. | 72. | 73. | 74. | 75. | 76. | 77. | 78. | 79. | 80. |
| 81. | 82. | 83. | 84. | 85. | 86. | 87. | 88. | 89. | 90. |
| 91. | 92. | 93. | 94. | 95. | 96. | 97. | 98. | 99. | 100. |

ADVENTURE GRID FOR ............................................................

DATE ...................... AUTHOR ................................



## Tips for map-making

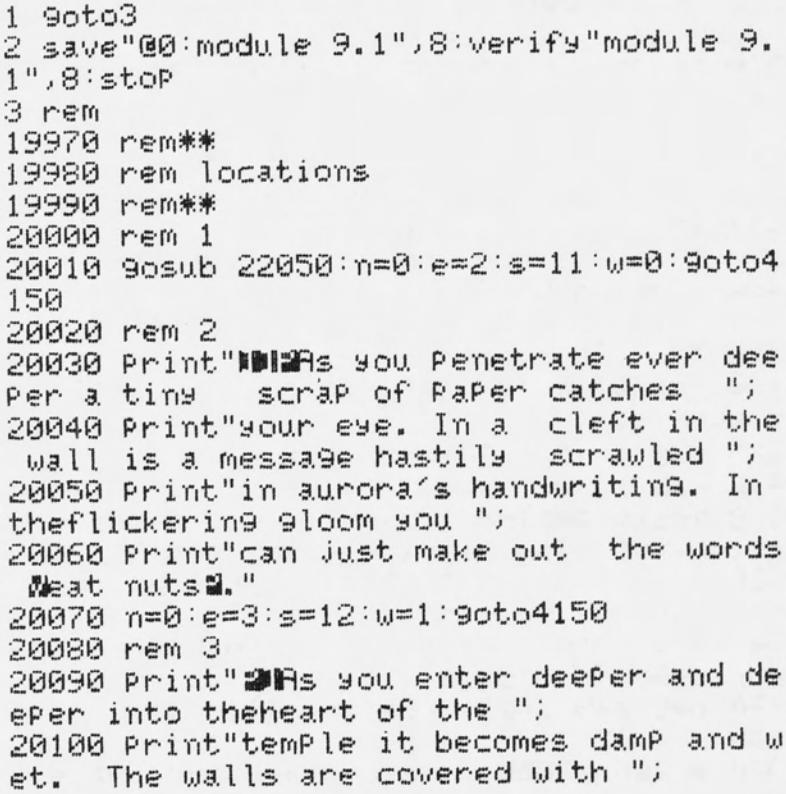
When you begin to develop your own Adventure one or two of the tips I found useful were:

1. Always draw in pencil — you'll be changing things quite a lot as you develop ideas and change locations.
2. Try to incorporate a 'maze' area where there are several locations and also plenty of exits to allow your player to get thoroughly lost. The temple and the forest are examples of this in *Nightmare Planet*.

3. Some Locations have one way direction only. If you look at the "clearing" in Location 53 in **Figure 3.3** you will notice I have arrows going out of the clearing in all directions, but you can only enter from the west or north. A good way of using this facility is to have a Location of, say, a cave or a room which allows you to enter but not to leave until a certain task has been performed. I'll be demonstrating this in Location 92 later in the book.
4. Don't be too ambitious with objects and perils. The amount of memory to enable you to deal with these is quite substantial (as you will see later) and it's best to start small and build if you can.

## The Locations

MODULE 9.1



```
1 goto3
2 save"@0:module 9.1",8:verify"module 9.
1",8:stop
3 rem
19970 rem**
19980 rem locations
19990 rem**
20000 rem 1
20010 gosub 22050:n=0:e=2:s=11:w=0:goto4
150
20020 rem 2
20030 print"███As you penetrate ever dee
per a tiny   scrap of paper catches  ";
20040 print"your eye. In a  cleft in the
 wall is a message hastily  scrawled ";
20050 print"in aurora´s handwriting. In
theflickering gloom you ";
20060 print"can just make out  the words
 █eat nuts█."
20070 n=0:e=3:s=12:w=1:goto4150
20080 rem 3
20090 print"██As you enter deeper and de
eper into theheart of the ";
20100 print"temple it becomes damp and w
et.  The walls are covered with ";
```

```
20110 print"an evil-smelling slime that
seems to pulse witha malignant";
20120 print" light ...":n=0:e=4:s=13:w=2
:goto4150
20130 rem 4
20140 print"■■It's very dark inside the
 temple. the ground is uneven ";
20150 print"and several times you  stumb
le and fall.  A passage appears ";
20160 print"to the west.":n=0:e=0:s=0:w=
3:goto4150
20170 rem 11
20180 gosub 22050:n=1:e=12:s=0:w=0:goto4
150
20190 rem 12
20200 gosub 22050:n=2:e=13:s=22:w=11:got
o4150
20210 rem 13
20220 print"■■Inside the temple it is t
oo dark to   see much. Passages ";
20230 print"seem to stretch in   several
 directions.  Which way is best ?"
20240 n=3:e=14:s=0:w=12:goto4150
20250 rem 14
20260 print"■  You are outside a buildin
g shaped likea temple. ";
20270 print"On the ground outside the do
orare some marks ";
20280 print"scratched in the dirt of the
 pathway.  It looks like an ";
20290 print"arrow    pointing west into
the temple."
20300 n=0:e=15:s=24:w=13:goto4150
20310 rem 15
20320 print"■■The scrabbling of taloned
 feet makes  you swirl - ready for ";
20330 print"anything. But onlythe stones
 stand impassive - oblivious  ";
20340 print"to your fear.  Perhaps they
are not deadbut only sleeping .."
20350 n=0:e=0:s=25:w=14:goto4150
20360 rem 22
20370 gosub 22050:n=12:e=0:s=0:w=0:goto4
150
```

```
20380 rem 24
20390 print"■■You have the impression s
omething is  watching you from the ";
20400 print"shadows.   In the silence yo
u want to shout, to call out, to ";
20410 print"do anything to make you feel
 there issomething alive in ";
20420 print"this dead place.":n=14:e=25:
s=0:w=0:goto4150
20430 rem 25
20440 print"■The wind whistles a mockin
g, eerie tunebetween the stone ";
20450 print"walls as you move pastempty
doors and gaping windows.  Is this";
20460 print"all that remains of civilisa
tion here ?"
20470 n=15:e=26:s=0:w=24:goto4150
20480 rem 26
20490 print"■■Inside the city you wonde
r where to   look ! If only ";
20500 print"you could find a clue butthe
 crumbling walls of huge houses ";
20510 print"blockany exploration apart f
rom a hive of    alleys leading west."
20520 n=0:e=0:s=36:w=25:goto4150
20530 rem 28
20540 print"■You are at the edge of the
 lake.  The  stones in the water ";
20550 print"seem to be sparklingin the s
unlight as if with a magic ";
20560 print"powerof their own.":n=0:e=0:
s=38:w=0:goto4150
20570 rem 36
20580 print"■■From here you can see qui
te clearly   that the towers and ";
20590 print"spires are the       jagged r
uins of a once-proud city now ";
20600 print"  deserted by its former inh
abitants."
20610 n=26:e=37:s=46:w=0:goto4150
20620 rem 37
20630 print"■At last the sand is giving
 way to more pebbles and stones -";
```

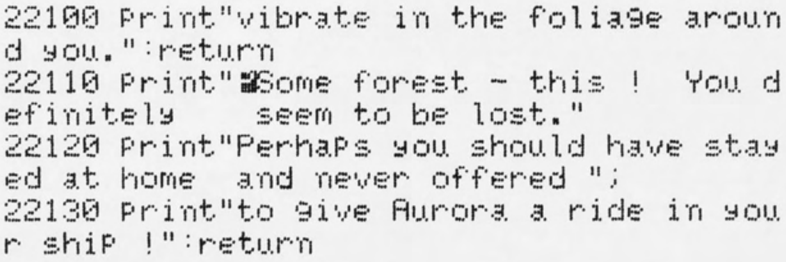
```
20640 print" it must be the out-skirts o
f the desert.  There are ";
20650 print"some   scrub-like bushes wit
h fruit like coco- nuts hanging ";
20660 print"from them.":n=0:e=38:s=47:w=
36:goto4150
20670 rem 38
20680 print"▓▓You are still in the dese
rt.  To the  north you can see ";
20690 print"a lake.  To the east  the de
sert stretches to the horizon.
20700 n=28:e=0:s=38:w=37:goto4150
20710 rem 42
20720 gosub 22110:n=0:e=43:s=52:w=0:goto
4150
20730 rem 43
20740 gosub 22070:n=0:e=44:s=53:w=42:got
o4150
20750 rem 44
20760 gosub22110:n=0:e=0:s=54:w=43:goto4
150
20770 rem 46
20780 print"▓Even though the sun is sta
rting to burna haze of ";
20790 print"confusion into your brain -
  on the north horizon gleam the ";
20800 print"spires   and towers of a cit
y. Or is it only a  mirage ";
20810 print"to torment you ??":n=36:e=37
:s=56:w=0:goto4150
20820 rem 47
20830 print"▓As you trudge wearily acros
s the burningsand you can";
20840 print" see score marks where your
ship crashed on landing.  You ";
20850 print"wonder howyou managed to hol
d the ship together.
20860 n=37:e=46:s=57:w=46:goto4150
20870 rem 48
20880 print"▓You have been walking for
hours now andcan see";
20890 print" no end to the vast expanse
of   sand. In fact you know that ";
```

```
20900 print"if you don´tturn back soon y
ou will probably die    from ";
20910 print"exposure under the sun which
 seems to hang motionless in the sky."
20920 n=38:e=0:s=0:w=47:goto4150
20930 rem 52
20940 gosub 22070:n=42:e=53:s=62:w=0:got
o4150
20950 rem 53
20960 print"■▌You are in a small clearin
g amidst the trees.  Soft moss-";
20970 print"covered banks surroundan azu
re pool of silent, still ";
20980 print"water andyou rest for a mome
nt before continuing your ";
20990 print"quest.":n=43:e=54:s=63:w=52:
goto4150
21000 rem 54
21010 print"■▌As you fight through bramb
les and thickfoliage you catch ";
21020 print"tantalising glimpses  of a c
learing to the west - but you can-";
21030 print"not seem to be able to get t
o it !!":n=44:e=55:s=64:w=0:goto4150
21040 rem 55
21050 print"■■▌You are just in the fores
t now, with  trees as thick as houses ";
21060 print"towering into  the sky and b
lotting out the sun.  Small";
21070 print"animals skitter and run arou
nd and aboveyou.
21080 print"Two paths lead deeper - one
to the southand one to the west."
21090 n=0:e=56:s=65:w=54:goto4150
21100 rem 56
21110 print"■■▌You are at the edge of th
e desert  -  for the sand soon ";
21120 print"changes to scrub-like grass.
  Towards the west ";
21130 print"a huge forest  fills the sky
line like a massive carpet of ";
21140 print"green.":n=46:e=57:s=0:w=55:g
oto4150
```

```
21150 rem 57
21160 print"■■■You are now outside your
 spaceship.   Around you lie the ";
21170 print"scorch marks and     scarred
 charred fragments of your ship. You ";
21180 print"have crashed in an arid dese
rt of   purple sand.
21190 n=47:e=58:s=67:w=56:goto4150
21200 rem 58
21270 print"■■■You are in the airlock.
21290 print"■■■One of the spacesuits is
missing. ";
21300 print"On  the floor you see a smal
l shred of clothwhich you ";
21310 print"recognise as a piece of the
  royal gown Princess Aurora was ";
21320 print"wearing  when you last saw h
er....":goto21380
21380 n=0:e=59:s=0:w=57:goto4150
21390 rem 59
21430 print"■■■You are in the cabin of y
our spaceship and the crash has ";
21440 print"emptied the contents  of you
r locker all over the floor.
21460 print"Even so you realise that the
re are the  signs of a struggle.
21470 print"■■■■***The Princess has gone
!***"
21480 print"■■■The airlock dial register
s that the airlock has been ";
21490 print"opened from the outside   an
d then resealed ..."
21500 n=0:e=0:s=0:w=58:goto4150
21510 rem 62
21520 gosub 22070:n=52:e=63:s=0:w=0:goto
4150
21530 rem 63
21540 gosub 22110:n=0:e=64:s=0:w=62:goto
4150
21550 rem 64
21560 print"■■The forest is definitely t
oo large for you to ever find ";
21570 print"your way out again, andyou a
```

```
re beginning to worry as well as ";
21580 print"  beginning to wonder how yo
u will ever   succeed.
21590 n=54:e=65:s=74:w=63:goto4150
21600 rem 65
21610 print"■■This path seems to be hea
ding away    from the sun and into a ";
21620 print"a gloom you      would never
have thought possible in    the ";
21630 print"daylight.  Behind you are he
avy      rustlings, as though some ";
21640 print"massive beast is following y
ou.":n=55:e=0:s=0:w=64:goto4150
21650 rem 67
21660 print"■The desert soon becomes lu
sh with thickmauve vegetation";
21670 print" that seems to thicken  with
 every step.  Better watch yourself ";
21680 print"around here - the ground is
starting to get very boggy."
21690 n=57:e=68:s=0:w=0:goto4150
21700 rem 68
21710 print"■■You are in swampland, hot
, sticky and unpleasant. ";
21720 print"You can hear the roar of a
river, seemingly huge and rushing.";
21730 print"  To  the east lies the rive
r -  to the west  the swamp.
21740 n=0:e=69:s=0:w=67:goto4150
21750 rem 69
21760 print"■■It's so refreshing to sit
 on the bank of the river and ";
21770 print"dangle your steaming   feet
in the ice-cold spray.  But the ";
21780 print"   river is vast, like a chu
rning sea to   the horizon.  It would ";
21790 print"be impossible to cross.":n=0
:e=0:s=0:w=68:goto4150
21800 rem 74
21810 print"■■You are on a road to the
south of the forest.  The smooth ";
21820 print"polished surface    suggests
 it was made by a civilisation  far ";
```

```
21830 print"superior to any you have kno
wn.  Butis this road still used ....";
21840 print" or is it   just a relic of
past glory ?"
21850 n=64:e=0:s=84:w=0:goto4150
21860 rem 84
21870 print"■The road winds on across t
he grasslands - stretching north ";
21880 print"and south like some huge rib
bon tossed down by a giant ";
21890 print"hand.On either side the vege
tation closes in at times ...
21900 n=74:e=0:s=94:w=0:goto4150
21910 rem 92
21920 print"■■You are inside the mud hu
t.  Several  items of primitive ";
21930 print"crockery crunch underyour fe
et.":n=0:e=93:s=0:w=0:goto4150
21940 rem 93
21950 print"■■You make you way through
the village  hoping the natives ";
21960 print"remain friendly.":n=0:e=94:s
=0:w=92:goto4150
21970 rem 94
21980 print"■■You are in the outskirts
of a native  village.  Mud huts ";
21990 print"are scattered around in a ha
phazard fashion."
22020 print"The village seems deserted,
but you can hear a woman ";
22030 print"singing sadly to herself ina
 hut.
22040 n=84:e=0:s=0:w=93:goto4150
22050 print"■There's such a labyrinth o
f twists and turns that you are";
22060 print" getting just a littleafraid
 you are lost..... ":return
22070 print"■You must be lost in the fo
rest.  You   keep catching glimpses of";
22080 print" the clearing, sometimes to
the east, sometimes the    south...";
22090 print"and all the time the noises
of  some fearsome beast echo and ";
```



```
22100 print"vibrate in the foliage aroun
d you.":return
22110 print"▓Some forest - this !  You d
efinitely    seem to be lost."
22120 print"Perhaps you should have stay
ed at home  and never offered ";
22130 print"to give Aurora a ride in you
r ship !":return
```

This module is straightforward programming of the Locations. At the end of the module are three subroutines (Lines 22050, 22070 and 22110) which I have inserted in the temple and the forest to save a little space and confuse the player — as he will get the same response on the screen in several places. Apart from that it's just typing until you're done.

In Location 2 (Line 20060) there is a rather enigmatic message scrawled on the wall — EAT NUTS — which is a slightly obscure clue to help a frustrated player who may have not worked out that 'eating' the nuts in Location 37 will allow him to remove his spacesuit — a necessary requisite if he is to rescue Aurora. I like the idea of leaving obscure clues around, and in fact this temple contains a couple of clues which should keep a player guessing. Apart from the message there is an arrow scrawled in the dust pointing into the temple in Location 14. Was it left by Aurora? Is she still in the temple somewhere?

In fact I have tried in various places to build in an element of mystery and slightly uneasy tension. Apart from the clues that lead you to try and search (fruitlessly) for Aurora in the temple — presumably she was there but was removed to the native village before you arrived on the scene — I have added bits about mysterious beasts in the forest (which never materialise) and the total red herring of a gem in the forest clearing which has no useful role at all.

To a certain extent this Adventure may not be hard enough. But then it is intended as an example program — not a full-scale cannot-be-solved Adventure. Why not try to make it harder if you think of ways as you go along?

Remember the tips for text typing I described in Chapter 6 about testing a few lines as you go. It's fairly easy in this module as they all end in GOTO 4150 which you haven't entered yet — so on giving the direct instruction GOTO 20080 (for Location 3 for example) you will get a printout of the Location then the UNDEF'D STATEMENT ERROR message and the program will break, allowing you to LIST the relevant lines and alter them if you wish.

Also don't forget to start each new Location with the CONTROL key and 1 key to colour the text "black".

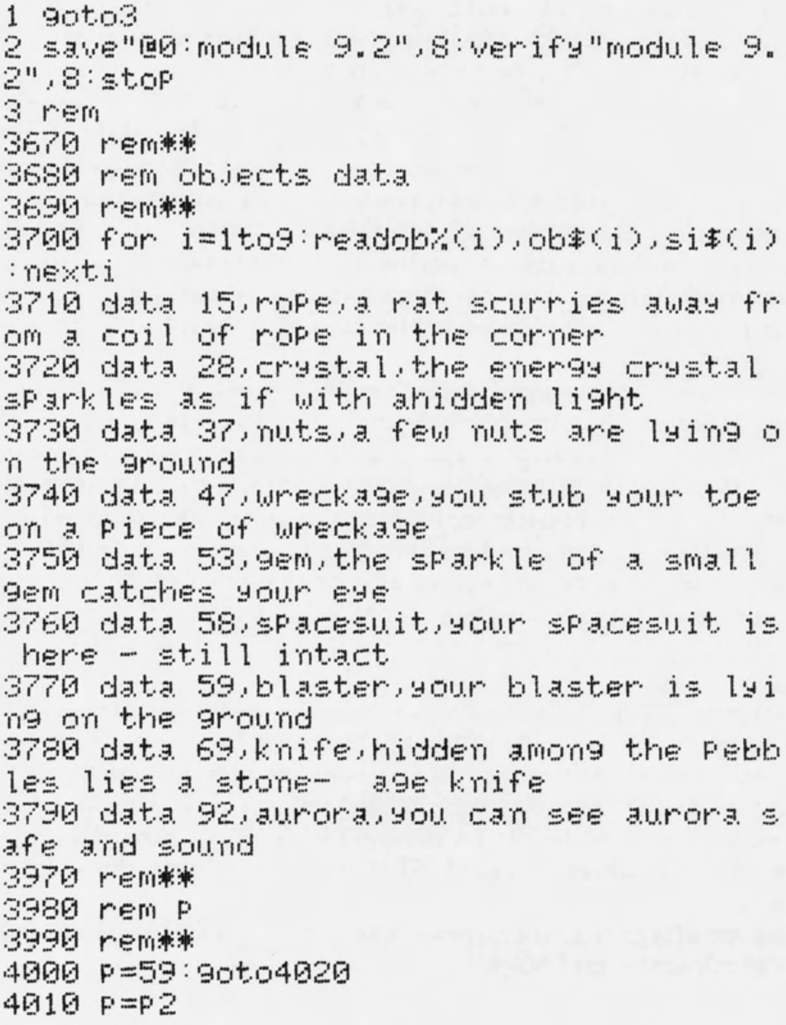
## Moving around the adventure

### Variables in Module 9.2

OB% Integer value of Object in Object Array.
OB$ Single word description of Object.
SI$ Text description of Object for Location.

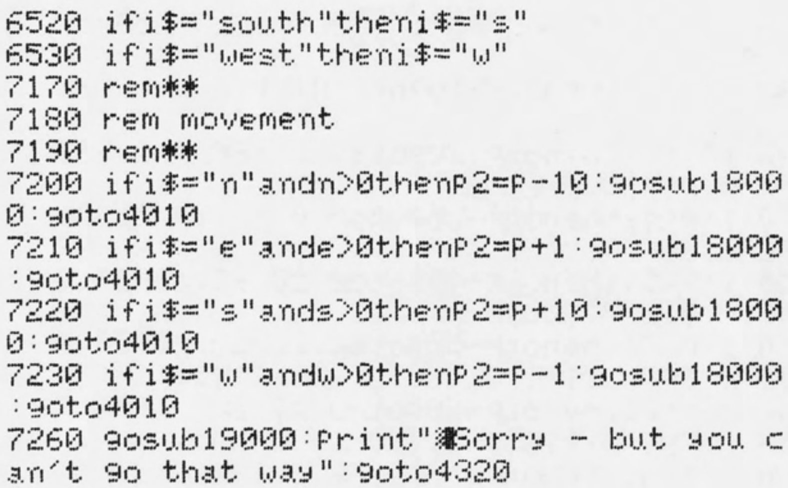
This module involves the same principles as in Chapter 2 on my sample program, so I don't intend going into much detail. The lines in many cases are exactly the same.

MODULE 9.2



```
1 goto3
2 save"@0:module 9.2",8:verify"module 9.
2",8:stop
3 rem
3670 rem**
3680 rem objects data
3690 rem**
3700 for i=1to9:readob%(i),ob$(i),si$(i)
:nexti
3710 data 15,rope,a rat scurries away fr
om a coil of rope in the corner
3720 data 28,crystal,the energy crystal
sparkles as if with ahidden light
3730 data 37,nuts,a few nuts are lying o
n the ground
3740 data 47,wreckage,you stub your toe
on a piece of wreckage
3750 data 53,gem,the sparkle of a small
gem catches your eye
3760 data 58,spacesuit,your spacesuit is
 here - still intact
3770 data 59,blaster,your blaster is lyi
ng on the ground
3780 data 69,knife,hidden among the pebb
les lies a stone-  age knife
3790 data 92,aurora,you can see aurora s
afe and sound
3970 rem**
3980 rem p
3990 rem**
4000 p=59:goto4020
4010 p=p2
```

```
4020 ifp<11thenonpgoto20000,20020,20080,
20130
4030 ifp<21thenonp-10goto20170,20190,202
10,20250,20310
4040 ifp<31thenonp-20goto0,20360,0,20380
,20430,20480,0,20530
4050 ifp<41thenonp-30goto0,0,0,0,0,20570
,20620,20670
4060 ifp<51thenonp-40goto0,20710,20730,2
0750,0,20770,20820,20870
4070 ifp<61thenonp-50goto0,20930,20950,2
1000,21040,21100,21150,21200,21390
4080 ifp<71thenonp-60goto0,21510,21530,2
1550,21600,0,21650,21700,21750
4090 ifp<81thenonp-70goto0,0,0,21800
4100 ifp<91thenonp-80goto0,0,0,21860
4110 ifp<101thenonp-90goto0,21910,21940,
21970
4120 rem**
4130 rem see object
4140 rem**
4150 fori=1to9:ifob%(i)=pthenprintsi$(i)
4160 nexti
4290 rem**
4300 rem exits
4310 rem**
4320 print"▨▩You can go";
4330 ifn>0thenprint" north";
4340 ife>0thenprint" east";
4350 ifs>0thenprint" south";
4360 ifw>0thenprint" west";
4970 rem**
4980 rem instruction
4990 rem**
5000 printchr$(13):x=fre(0)
5010 input"▨Instructions:▐▐▐██▐";i$
5020 pokev+21,0
5030 ifi$="*"thenprint"▩▩":goto5010
6470 rem**
6480 rem i$
6490 rem**
6500 ifi$="north"theni$="n"
6510 ifi$="east"theni$="e"
```



```
6520 ifi$="south"theni$="s"
6530 ifi$="west"theni$="w"
7170 rem**
7180 rem movement
7190 rem**
7200 ifi$="n"andn>0thenP2=P-10:gosub1800
0:goto4010
7210 ifi$="e"ande>0thenP2=P+1:gosub18000
:goto4010
7220 ifi$="s"ands>0thenP2=P+10:gosub1800
0:goto4010
7230 ifi$="w"andw>0thenP2=P-1:gosub18000
:goto4010
7260 gosub19000:Print"█Sorry - but you c
an´t go that way":goto4320
```

Lines 3700–3790: I have nine objects in *Nightmare Planet* as below in **Table 9.1**.

***Table 9.1 Objects in Nightmare Planet***

| | |
|---|---|
| 1. ROPE | LOCATION 15 |
| 2. CRYSTAL | LOCATION 28 |
| 3. NUTS | LOCATION 37 |
| 4. WRECKAGE | LOCATION 47 |
| 5. GEM | LOCATION 53 |
| 6. SPACESUIT | LOCATION 58 |
| 7. BLASTER | LOCATION 59 |
| 8. KNIFE | LOCATION 69 |
| 9. AURORA | LOCATION 92 |

These lines place these objects into DATA statements.

Lines 4000–4110: Using the same principle as in my sample program these lines just set the position (P = 59 which is the cabin of the spaceship — see Grid in **Figure 9.3**) then direct the 64 to the appropriate location once you decide to move with the ON. . . GOTO facility.

Lines 4150–4160: A loop to scan the object data and if it is in the location then SI$ will appear under the text you typed in Module 9.1. Once again a point to make is the importance of having SI$ look applicable wherever the object happens to be. Thus if we had said "THE CRYSTAL SPARKLES IN THE DEPTHS OF THE LAKE" for Object 2 (the crystal) it would look fine when first you came upon it but suppose you drop the crystal later in Location 59 then that line looks pretty silly in a spaceship.

Lines 4320–5030: First the display "YOU CAN GO" followed by the appropriate direction, then the "INSTRUCTIONS ?" input that comes up after every move by the 64. I chose the input "INSTRUCTIONS" rather than "WHAT SHALL I DO NOW?" because I felt it was more applicable to Victor and the SF image of the game.

Note the [2 spaces right] and then the "*" symbol followed by [3 spaces left]. On screen this puts an asterisk under the flashing cursor — which looks better than the straightforward cursor. Line 5030 prevents any action if the user types the "*" symbol — as it puts the cursor back over your own asterisk.

Line 5020 POKEs Victor off again.

N.B. In Line 5000 X = FRE(0) is a way of doing a quick 'garbage collection' to clear out all the string spaces to prevent an OUT OF MEMORY error.

Lines 6500–7260: Again standard movement lines. Note that we have P – 10 and P + 10 for north and south movement as we have a 10x10 grid, and also note the beep and burp subroutines (GOSUB 18000 and GOSUB 19000) in these lines.

### *Testing Modules 9.1 and 9.2*

You'll find this as good (if not better) as testing Modules 8.3 and 8.4. Now you can actually move right around your map and see the objects displayed in the appropriate locations.

Spend some time just exploring, imagining if you are playing the game for the first time. There's something very satisfying about actually being able to move around your Adventure.

A note here for when you design your own game. I first wrote my Locations as though I was leaving the ship and truly exploring — for example I would say "You are leaving the desert and entering a ruined city. . ." This looked fine on the way into the city — but when I was 'leaving' the city again later on it looked pretty odd to see the line "You are leaving the desert. . ." because I wasn't! I was leaving the city!

So always try and write your description as though you can go either way into or out of the Location and it sounds all right. Time spent in sorting this out early on is well-spent, and in fact I was changing some of the wording of my Locations right up to the last minute as I spotted yet another mistake!!

This chapter has been a pretty exhausting programming bout — but the result should look very impressive. Next we'll start picking up and dropping your objects.

## Summary

For me the addition of the locations is probably the most exciting part of the whole game. At last I can begin to feel the scope of my imaginary world, I can see myself in the different scenarios, feel the hot sand under my feet and the sun on my neck, or picture the rushing river stretching like some pulsating sea to the far horizon. The forest with its tiny alien inhabitants, the city stark and wind-sculptured, the dank depths of gloom inside the alien temple — it's all there.

I love science-fiction, I love the lurid covers adorning many books and magazines, my mind soars out of its daily existence into a world of excitement and fantasy where I can safely shake my preconditions and my necessities and just fly on my own.

If nothing else writing this program has let me live a little more over the past few months in a world I see too little of and will never lose.

# CHAPTER 10
# Picking up nicely now

As with all good Adventures the time has come to start to manipulate our objects as well as our hero around our imaginative world in deep space. The art of "PICKING" or "GETTING" objects and "DROPPING" them again was explained in Chapter 3 so again I will tend to skim quite quickly through some of the programming techniques.

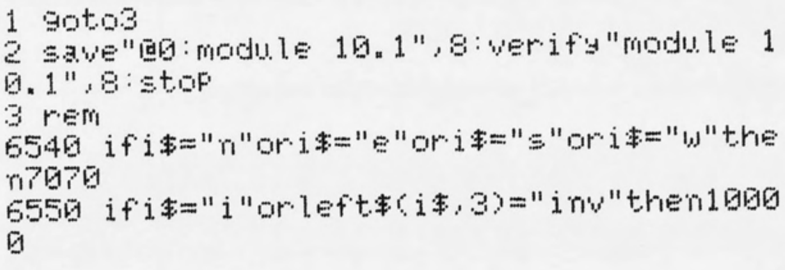
One golden rule for Adventuring is to always "GET" an object as soon as you see it. Thus you start to load yourself up with various assorted paraphernalia as you delve ever deeper into the plot. I wanted to make life a little difficult by making some objects inaccessible without having first acquired some other object — for example the knife in Location 69 cannot be reached without first having "got" the rope in Location 15 otherwise you cannot escape from the quicksand in Location 67.

Another way of making life difficult is to set a limit on how many objects you can carry at one time, which I will cover in Module 10.3. But let's get back to the keyboard.

## Airlocks and things

I wanted to add a little extra touch as we move out of the airlock, and by using sound plus visuals give the impression of the airlock actually opening. Purists might say that the airlock should open twice to be a true airlock (true) but as in the use of noise in space to enhance battle sequences my airlock only opens once (to save on time).

MODULE 10.1



```
1 goto3
2 save"@0:module 10.1",8:verify"module 1
0.1",8:stop
3 rem
6540 ifi$="n"ori$="e"ori$="s"ori$="w"the
n7070
6550 ifi$="i"orleft$(i$,3)="inv"then1000
0
```

```
6570 ifi$="instructions"thengosub15000
6580 ifi$="run"ori$="walk"thengosub18000
:print"██Which way ?":goto4320
6690 ifi$="open airlock"andp=57then7100
6700 ifi$="open airlock"andp=58then7100
6840 ifi$="save"thengoto17000
6850 ifi$="quit"ori$="end"thengoto17500
7000 ifi$<>"n"andi$<>"e"andi$<>"s"andi$<
>"w"then7300
7040 rem**
7050 rem airlock
7060 rem**
7070 ifp=58andi$="w"then7100
7080 ifp=57andi$="e"then7100
7090 goto7200
7100 print"██airlock opening ...."
7110 forl=0to24:pokesc+l,0:nextl
7120 pokevo,15:pokea1,17:pokea2,17:pokes
1,130:pokes2,130
7130 pokew1,17:pokew2,17:pokeh1,13:pokel
1,78:pokeh2,253:pokel2,46
7140 fort=1to1500:nexttt:pokevo,0:print"█
                    ":rem 20 spaces
7150 fort=1to700:nexttt:print"█airlock op
en ......"
7160 fort=1to700:nexttt:goto7200
7240 ifi$="open airlock"andp=57thenp2=p+
1:gosub18000:goto4010
7250 ifi$="open airlock"andp=58thenp2=p-
1:gosub18000:goto4010
```

Line 6540: This line is a bypass for all the I$ commands that we will be programming later. What it does is send the program through to the movement section thereby avoiding any 'error' messages, etc that lie within an I$ reply. It is more obvious later in the programming (or you can turn to the finished program at the end to see what I mean).

Lines 6550–6580: A few I$ commands I have put in here.

Lines 6690–6700: Although you can move quite easily into and out of the spaceship by giving an EAST or WEST command some people could well type in OPEN AIRLOCK and it seemed sensible to cater for them. Thus these two lines direct the program to Line 7100 which is the routine opening the airlock (note, providing you are in Locations 57 or 58 — the relevant Locations). If you were to type "OPEN AIRLOCK" in any other location a standard response would come up.

Lines 6840–6850: The SAVE and QUIT commands are catered for.

Line 7000: This line sends the program past the movement area if no direction has been typed in. It saves having to worry about various responses such as "YOU CANNOT MOVE IN THAT DIRECTION" within the movement area.

If you are becoming a little confused about the various sections of the program so far **Flowchart 10.1** should help clarify where we are at this stage.

Lines 7070–7250: This section is a visual and audible simulation of the airlock opening. As soon as you type W or OPEN AIRLOCK then the 64 first works out which way you're going (Lines 7070 and 7080) and PRINTS the words "airlock opening . . .".

At the same moment a hum sounds (Lines 7110–7130). As the hum ceases the words "airlock opening . . ." are blanked out (Line 7140 — note the 20 spaces to cover the words). There is a short pause, then "airlock open . . ." is PRINTed to replace the original message (Lines 7150 and 7160).

Finally the next location appears (Lines 7240 and 7250). These last two lines act instead of the ON . . . GOTO commands in Module 9.2.

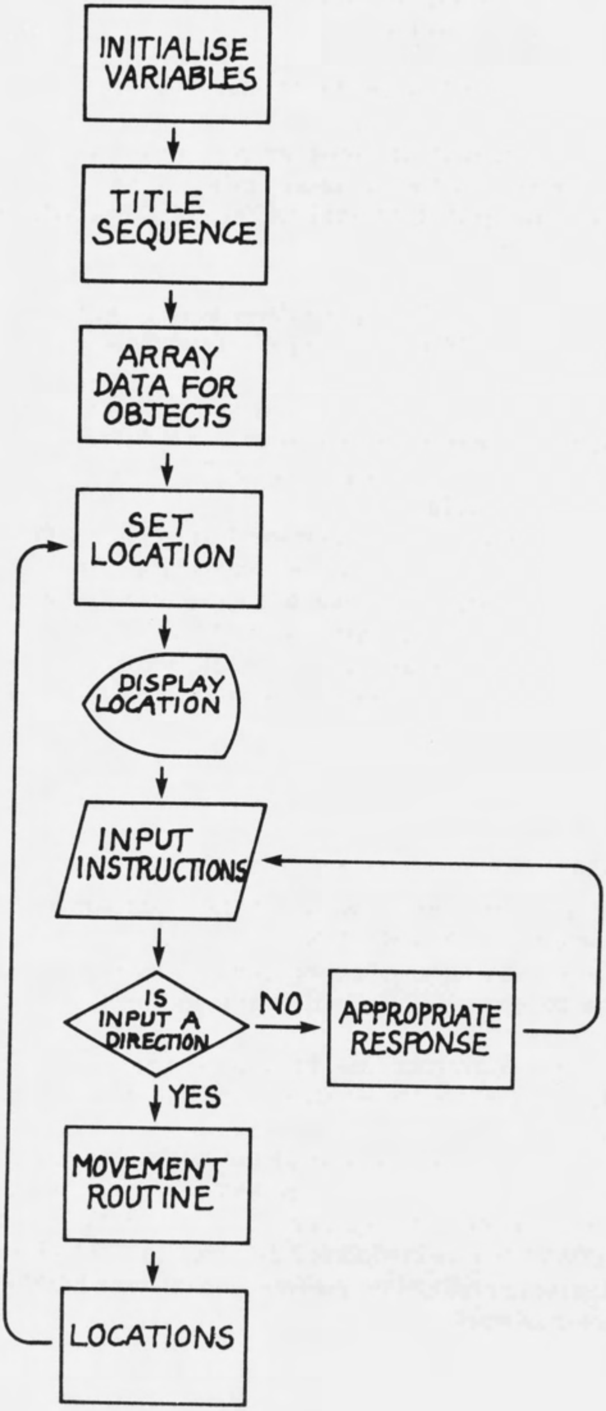
*Testing Module 10.1*

RUN the program, and on reaching the first Location type W to get into the airlock. Now type OPEN AIRLOCK and you should see the words appear and hear the delicious hum as the door opens. I must confess it took several hours of trial and error to get that particular sound, but the result was worth it.

You can now return back into the ship by either typing E or OPEN AIRLOCK again. Again the words and the hum will accompany your movement.

There's not much else you can do with this module yet, except type RUN or WALK to get the response. If you type SAVE the screen should blank and an "ARE YOU SURE" message appear. If you want to test if you have entered the SAVE routine in Module 7.2 correctly then SAVE at this stage. To complete the test reRUN the program and this time LOAD from your tape. It should all work.

**Flowchart 10.1 The basic structure of Nightmare Planet**



## Strings and things

MODULE 10.2

```
1 goto3
2 save"@0:module 10.2",8:verify"module 1
0.2",8:stop
3 rem
7270 rem**
7280 rem check i$ for space
7290 rem**
7300 fori=1tolen(i$)
7310 ifmid$(i$,i,1)=" "then7340
7320 nexti
7330 print"█please can you use two words
":gosub19000:goto5000
7340 fori=1tolen(i$)
7350 ifmid$(i$,i,2)="  "then7380
7360 nexti
7370 goto7420
7380 print"█please leave █one█ space bet
ween words":gosub19000:goto5000
7390 rem**
7400 rem i$-ve$&no$
7410 rem**
7420 forsp=pi+1tolen(i$)
7430 ifmid$(i$,sp,1)=" "then7450
7440 nextsp
7450 ve$=left$(i$,sp-1)
7460 no$=right$(i$,(len(i$)-sp))
```

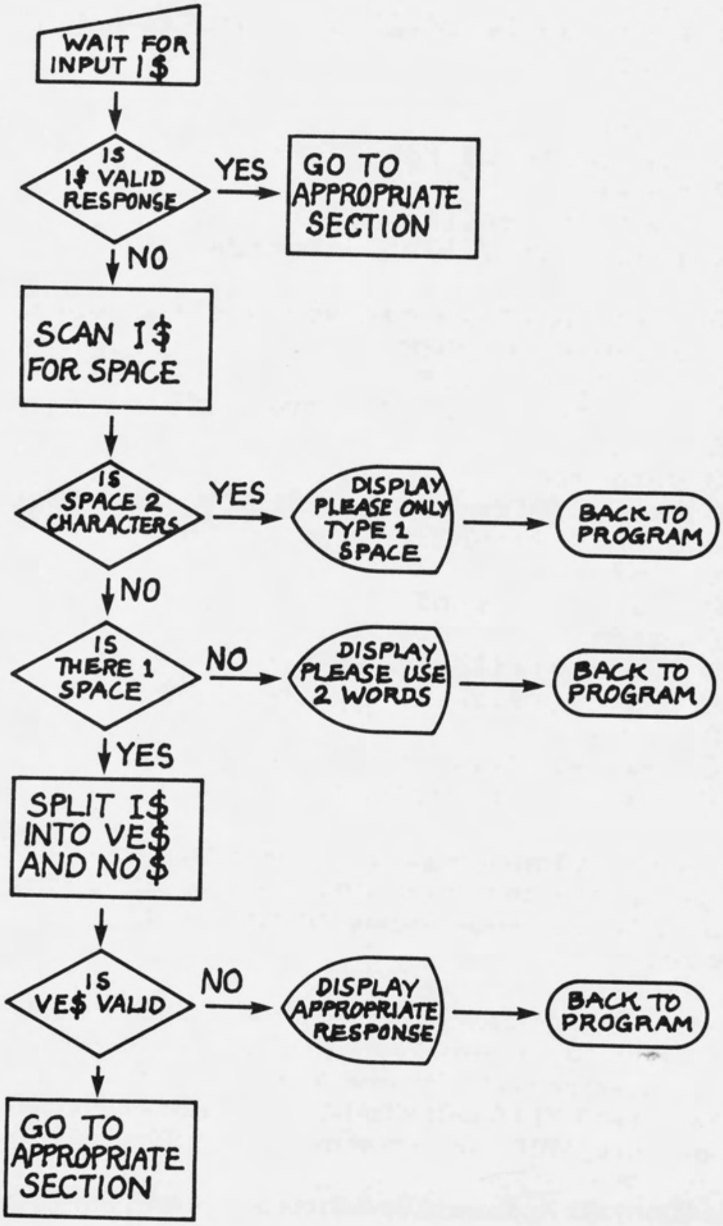
Lines 7300–7330: These lines are an exact copy of Module 1.5 in Chapter 3 so I'll just repeat they are here to test if the player has typed two words and if he has not then the message comes up "PLEASE CAN YOU USE TWO WORDS".

Lines 7340–7380: Extending that concept this sequence checks to see if there are two spaces between the words (as this prevents the 64 from 'recognising' appropriate words. This time the message is "PLEASE LEAVE < ONE> SPACE BETWEEN WORDS". The principle of recognition of two spaces using MID$ is the same as that for recognising one space.

Lines 7420–7460: And once again this routine is an exact copy of Module

1.6 in Chapter 6. It is splitting I$ into VE$ and NO$ (verb and noun).

To help illustrate the principles of string handling I suggest you look at **Flowchart 10.2**

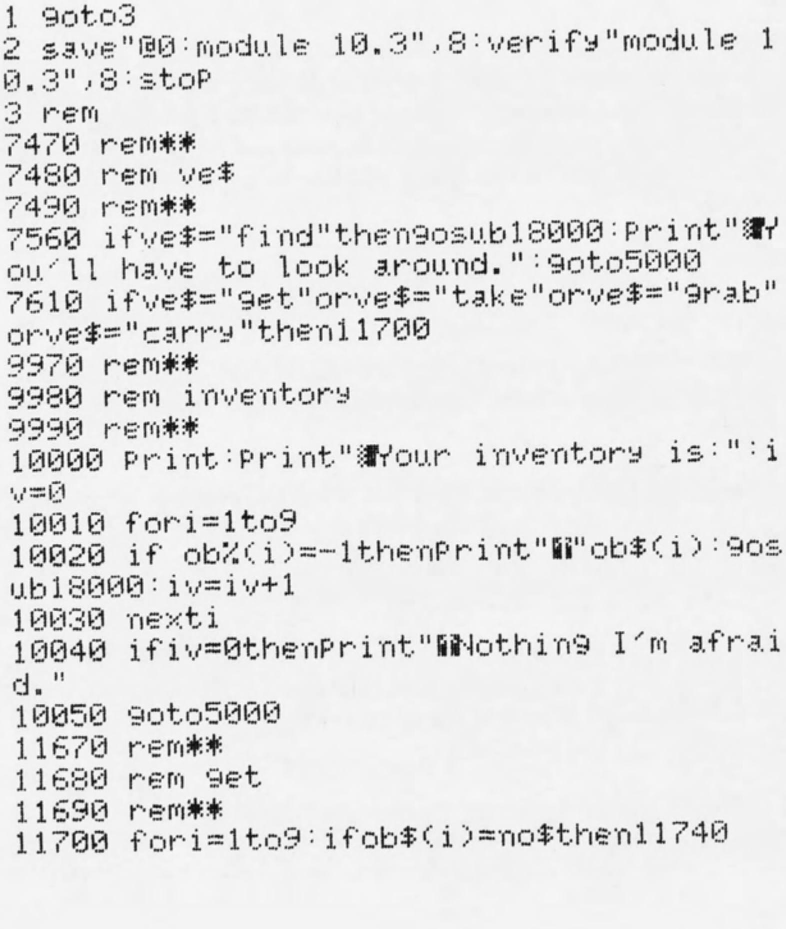


*Testing Module 10.2*

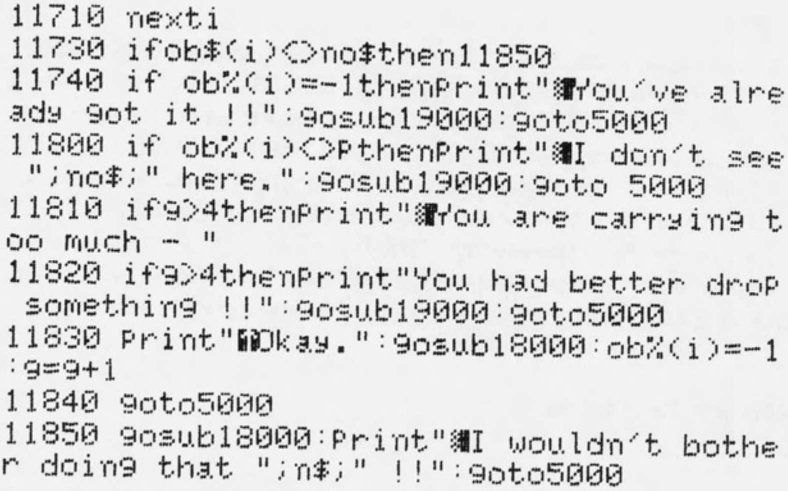
This is pretty easy to test. Once you are in any Location type "GOHORSE" say, and the message "PLEASE CAN YOU USE TWO WORDS" should appear. Now type "GO HORSE" (with two spaces) and the appropriate response should also appear. Finally type "GO HORSE" (the < correct> command) and as this is recognised as valid the program will drop into the Instructions routine. Once you have gone through that you will break with an error message. Now you can type "PRINT VE$" and "PRINT NO$" in direct mode and in each case you should see "GO" and "HORSE". If this doesn't happen go back and check your lines as you have possibly made an error in the brackets or the syntax.

## Getting a bit much

MODULE 10.3



```
1 goto3
2 save"@0:module 10.3",8:verify"module 1
0.3",8:stop
3 rem
7470 rem**
7480 rem ve$
7490 rem**
7560 ifve$="find"thengosub18000:print"█Y
ou´ll have to look around.":goto5000
7610 ifve$="get"orve$="take"orve$="grab"
orve$="carry"then11700
9970 rem**
9980 rem inventory
9990 rem**
10000 print:print"█Your inventory is:":i
v=0
10010 fori=1to9
10020 if ob%(i)=-1thenprint"██"ob$(i):gos
ub18000:iv=iv+1
10030 nexti
10040 ifiv=0thenprint"██Nothing I´m afrai
d."
10050 goto5000
11670 rem**
11680 rem get
11690 rem**
11700 fori=1to9:ifob$(i)=no$then11740
```



```
11710 nexti
11730 ifob$(i)<>no$then11850
11740 if ob%(i)=-1thenprint"▒You´ve alre
ady got it !!":gosub19000:goto5000
11800 if ob%(i)<>pthenprint"▒I don´t see
 ";no$;" here.":gosub19000:goto 5000
11810 ifg>4thenprint"▒You are carrying t
oo much - "
11820 ifg>4thenprint"You had better drop
 something !!":gosub19000:goto5000
11830 print"▒Okay.":gosub18000:ob%(i)=-1
:g=g+1
11840 goto5000
11850 gosub18000:print"▒I wouldn´t bothe
r doing that ";n$;" !!":goto5000
```

Lines 7560: One of the problems with an Adventure game is trying to anticipate the different commands people might use. FIND seemed a fairly popular one (in fact I even had one player type "FIND PRINCESS" as the first command right at the beginning of the game as if the computer would then tell where she was!!!) so I added this line to cope with all potential 'finders'.

Line 7610: This is the command which will enable us to 'get' our objects. Note I have also added 'take', 'grab' and 'carry' to this command.

Lines 10000–10050: A standard Inventory routine as in Chapter 3 (Module 1.7). The only addition is that a beep noise accompanies each object as it is printed out.

Also note that I have not put a noise in when the printout is "NOTHING". This is quite effective as the player expects the noises that accompany every response and looks up when nothing happens to see this is why. There are times when silence can be more effective than a noise.

Lines 11700–11840: The 'Get' routine, again as in Chapter 3. However note the addition of a couple of lines (11810 and 11820) which allow the variable G to reach five objects in the inventory before it complains "YOU ARE CARRYING TOO MUCH".

Line 11850: The standard response if you try to "GET" an object that isn't in the data and thus is considered invalid.

*Testing Module 10.3*

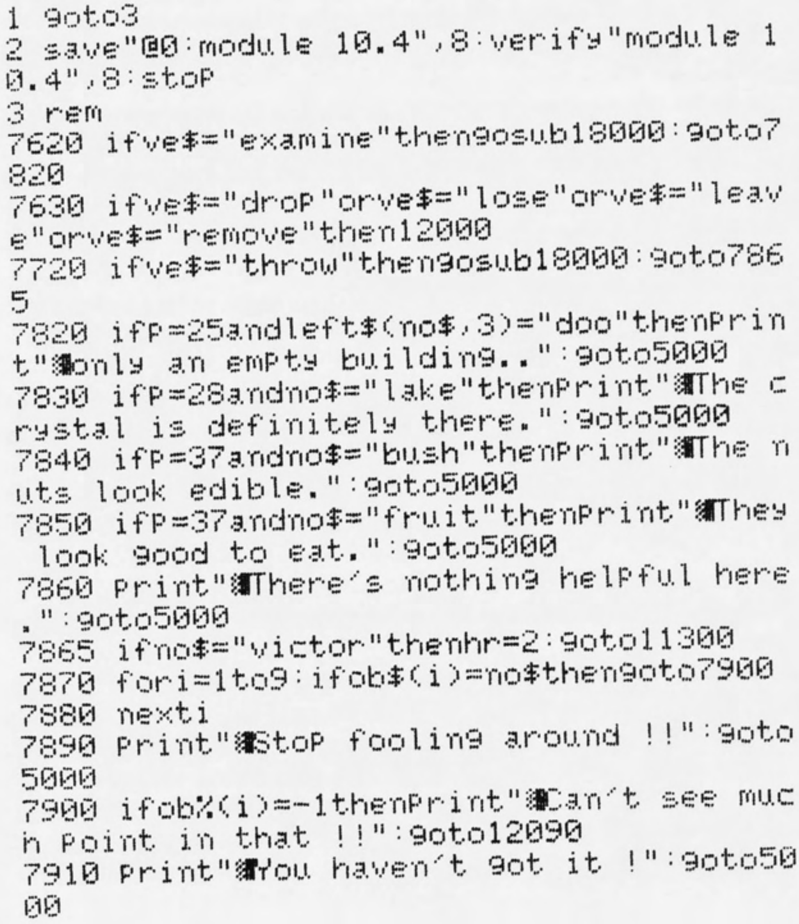
Try typing I once you are into the program to get the response "NOTHING". Then if you start to move around 'getting' objects you should be able to see them in the Inventory whenever you ask. Eventually you will run out of space — and with a nasty burp you will not be allowed to 'get' any more.

Now that you are starting to have to move into the Adventure to test each module you may find it easier to SAVE your program to tape and then just LOAD it back at the beginning of each test.

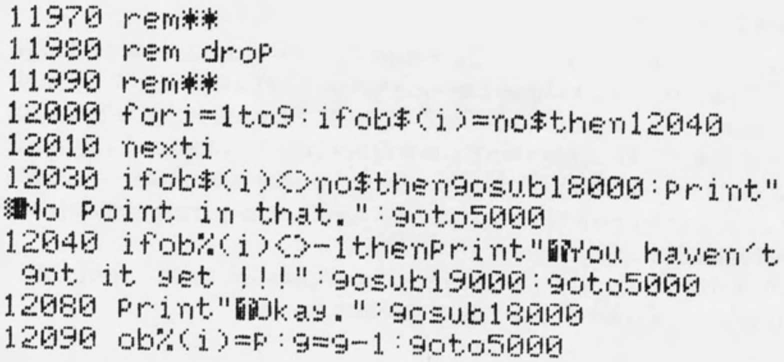
Having collected an armful of objects, we need to be able to 'drop' them again and the next Module will describe how we do it.

## Drop in again sometime

MODULE 10.4



```
1 goto3
2 save"@0:module 10.4",8:verify"module 1
0.4",8:stop
3 rem
7620 ifve$="examine"thengosub18000:goto7
820
7630 ifve$="drop"orve$="lose"orve$="leav
e"orve$="remove"then12000
7720 ifve$="throw"thengosub18000:goto786
5
7820 ifp=25andleft$(no$,3)="doo"thenprin
t"▓only an empty building..":goto5000
7830 ifp=28andno$="lake"thenprint"▓The c
rystal is definitely there.":goto5000
7840 ifp=37andno$="bush"thenprint"▓The n
uts look edible.":goto5000
7850 ifp=37andno$="fruit"thenprint"▓They
 look good to eat.":goto5000
7860 print"▓There's nothing helpful here
.":goto5000
7865 ifno$="victor"thenhr=2:goto11300
7870 fori=1to9:ifob$(i)=no$thengoto7900
7880 nexti
7890 print"▓Stop fooling around !!":goto
5000
7900 ifob%(i)=-1thenprint"▓Can't see muc
h point in that !!":goto12090
7910 print"▓You haven't got it !":goto50
00
```



```
11970 rem**
11980 rem drop
11990 rem**
12000 fori=1to9:ifob$(i)=no$then12040
12010 nexti
12030 ifob$(i)<>no$thengosub18000:print"
■No point in that.":goto5000
12040 ifob%(i)<>-1thenprint"■You haven't
 got it yet !!!":gosub19000:goto5000
12080 print"■Okay.":gosub18000
12090 ob%(i)=p:g=g-1:goto5000
```

Line 7620 and 7820–7860: One of the commands people often use in Adventures is EXAMINE. Line 7620 accepts that command and directs the program to Line 7820 where a number of responses are given. I had decided not to get involved in 'examining' objects too much because I had enough to cope with — but this area is a good one for all sorts of mysterious hints and cl ıes.

Lines 7820–7850 are concerned with certain locations (where I imagined someone might try to "EXAMINE" something) and give responses to the doors in the city (note LEFT$(NO$,3) = "DOO" here), the lake, and the fruit and nuts on the bush in the desert. I could have expanded this section still further had I wanted but I'll leave you to do that when you've finished and you know how much memory is left.

Line 7860 is the final line in this series which gives the response "NOTHING HELPFUL HERE" if the player tries to examine anything else.

Line 7630: The DROP, LOSE, LEAVE or REMOVE command will activate the 'drop' routine. The ability to have several choices for these commands is worthwhile, because whilst someone might easily DROP a knife, they would be more likely to REMOVE a spacesuit or LEAVE Aurora, and so allowing these extra words adds to the versatility of your game.

Lines 7720 and 7870–7910: These lines form a little routine to cope with the command THROW.

I decided I had to deal with this as though it was a 'drop' routine, because if you did throw away an object then you'd expect it to leave the inventory. Thus it becomes necessary to check if the object being thrown is a valid one (Lines 7870–7880) and if the object is indeed in the Inventory (Line 7870). If it is the program moves to the appropriate line (Line 7900) and PRINTs out the message "CAN'T SEE MUCH POINT IN THAT" so that the player knows he has not succeeded in anything much.

At this stage we GOTO 12090 which allows the object to move out of the Inventory and into the Location (see the 'drop' routine in a moment).

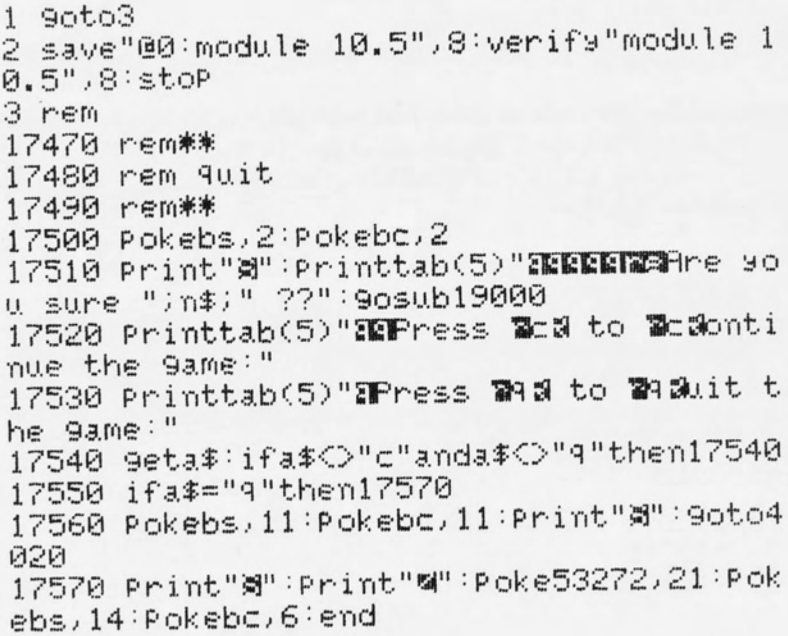
IF the object is not in the Inventory then the Line 7910 PRINTs "YOU HAVEN'T GOT IT". Line 7890 is a basic line for any other object that might be thrown, eg "throw horse". People can write the craziest things in an Adventure game (as I'm sure you know) so we need a response that caters both for crazy comments plus genuine attempts at throwing an object that is mentioned in the description but is not a valid object (for example the crockery in Location 92). Thus a response "I DON'T SEE IT HERE" would be inaccurate as the crockery obviously is there. I settled on "STOP FOOLING AROUND" as a good compromise.

Finally Line 7865 is one I added later to cope with the possibility that someone might try and 'throw' Victor. If you did you would go to Module 14.1 (Line 11300) where you'd be in for a nasty surprise.

Lines 12000 – 12090: The standard 'drop' section, as in Chapter 3. Nothing to add here, except we must remember to take the object away (G = G – 1) in Line 12090 to allow the player to pick up other objects if he is near his limit.

## So you want to quit?

MODULE 10.5



```
1 goto3
2 save"@0:module 10.5",8:verify"module 1
0.5",8:stop
3 rem
17470 rem**
17480 rem quit
17490 rem**
17500 pokebs,2:pokebc,2
17510 print"▨":printtab(5)"▨▨▨▨▨▨▨Are yo
u sure ";n$;" ??":gosub19000
17520 printtab(5)"▨▨Press ▨c▨ to ▨c▨onti
nue the game:"
17530 printtab(5)"▨Press ▨q▨ to ▨q▨uit t
he game:"
17540 geta$:ifa$<>"c"anda$<>"q"then17540
17550 ifa$="q"then17570
17560 pokebs,11:pokebc,11:print"▨":goto4
020
17570 print"▨":print"▨":poke53272,21:pok
ebs,14:pokebc,6:end
```

Line 17500: The screen changes to red for dramatic impact.

Lines 17510–17550: This section checks you really want to quit. If so the program will drop to Line 17570.

Line 17560: If you press C then the screen reverts to the familiar grey and clears, and you go back to the game.

Line 17570: This is your QUIT response and does three things. First the screen clears. Then POKE 53272,21 converts back to upper case. Then the screen will return to the familiar dark and light blue colours of the 64. The program ends with the computer looking like it should when you come out of a game.

*Testing Modules 10.4 and 10.5*

To test these modules you need to go through all the possible combinations of commands that you can think of. After getting into the game you need to:

1. Examine something (in any location).
2. Throw away an object you have picked up earlier.
3. Move out of the location and back into it to see if the object is displayed on the screen.
4. Throw away a valid object you have not picked up.
5. Throw away an invalid object (like a rock).
6. Check your inventory at each stage.
7. Try to QUIT or END, and then try to continue (C) or really quit (Q).

You should be able to do all these things and initiate the correct response with either a beep or a burp. The standard structure of the game is nearly complete now, and it's time to think about adding a bit of danger and excitement into our story.

## Summary

Much of this chapter has been a summary of original programming in Part 1 of the book, and as such I have tended to skim. It's quite interesting to note how I developed several of the modules to add a more sophisticated veneer to *Nightmare Planet* — I hope it's giving you ideas of your own to help you adapt the Adventure as you go.

# CHAPTER 11
# Not that way

*Nightmare Planet* contains only six basic perils — not many you might suppose. But programming just those six proved for me to be the most time-consuming and frustrating part of the whole game. So when you key in the appropriate lines over the next few chapters give a little thought to the time and effort I expended covering what will seem to be a relatively small part of the whole program. When you start thinking of your own perils it's best to be cautious at first because you can always add more at a later date if time, memory and sanity will allow.

The six basic perils are:

1. A snake in Location 24 which will crush you to death.
2. An underwater eel in Location 28 which will do the same.
3. The poisonous atmosphere which will kill you without a spacesuit.
4. A quicksand in the swamp in Location 67 which will suck you down until you die.
5. A dinosaur in Location 74 which will prevent you from moving south until you have solved how to get past.
6. Hostile natives in Location 92 who prevent you from taking Aurora away.

As well as these six obvious perils there are certain other hazards which are more concerned with the ultimate goal of winning the game. These are:

1. The fact that you cannot escape from the planet without Aurora.
2. You can only escape with Aurora if you kiss her (once you've found her).
3. You cannot kiss Aurora without removing your spacesuit — despite the poisonous atmosphere.
4. You need to eat the nuts in Location 37 to enable you to remove your spacesuit safely — note the clue in Location 2.
5. The fact that the energy crystal of the ship has been lost and you must retrieve it to escape from the planet.
6. You cannot sink in the water of the lake (to get the crystal) unless you

have something 'heavy' to let you sink (this is the wreckage in Location 47).

7. If you are not wearing your spacesuit when you sink in the depths you drown.
8. As your blaster will not work underwater you can only kill the eel with the knife (found in Location 69).
9. To acquire the knife you must pass the quicksand in Location 67 — and you will not escape this unless you have the rope (found in Location 15).
10. You can only kill the snake with your blaster.
11. You cannot kill the dinosaur and must climb a tree to get past it by swinging from tree to tree.

As you will see from this second list the original six perils have created quite a lot of programming. To enable you to understand how I developed the ideas and to help you with your own, I have tried in the Adventure to link various objects with various perils and make it quite difficult to find your way around.

## The use of variables

One of the first things to consider is to program the 64 so that the problem appears when you first encounter it — but you need to be able to switch it off should you return at a later date to the same location. I did this by setting a number of variables to 0 (Line 290 in the program — Module 6.1) and then once the peril had been defeated setting the variable to 2. This principle was explained in Chapter 4 — the only difference here is that there are rather more variables involved.

**Table of Variables for Perils**

| | |
|---|---|
| PR | Set at 2 when the Princess is rescued. |
| CV | Set at 2 when the Crystal is found. |
| SV | Set at 2 when the Snake has been killed. |
| EV | Set at 2 when the Eel has been killed. |
| QV | Set at 2 when the Quicksand has been negotiated. |
| DV | Set at 2 when the Dinosaur has been passed. |
| NV | Set at 2 when the Natives are friendly. |
| FV | Set at 2 when the Nuts have been eaten. |
| ES | Set at 2 when the Eel has been killed. |
| | Set at 1 when the Eel has got you to prevent movement. |
| NP | Set at 1 when you have been to the hut once. |
| NA | Set at 1 to allow movement from the hut. |
| Z | Set at 1 when the Nuts have been eaten. |

| | |
|---|---|
| RC | Set at 2 to stop movement from the lake. |
| HH | Set at 2 when Aurora is in the ship. |
| YY | Set at 1 when in the grip of the eel to stop 'look'. |
| HR | Set at 2 when Victor cannot help you. |

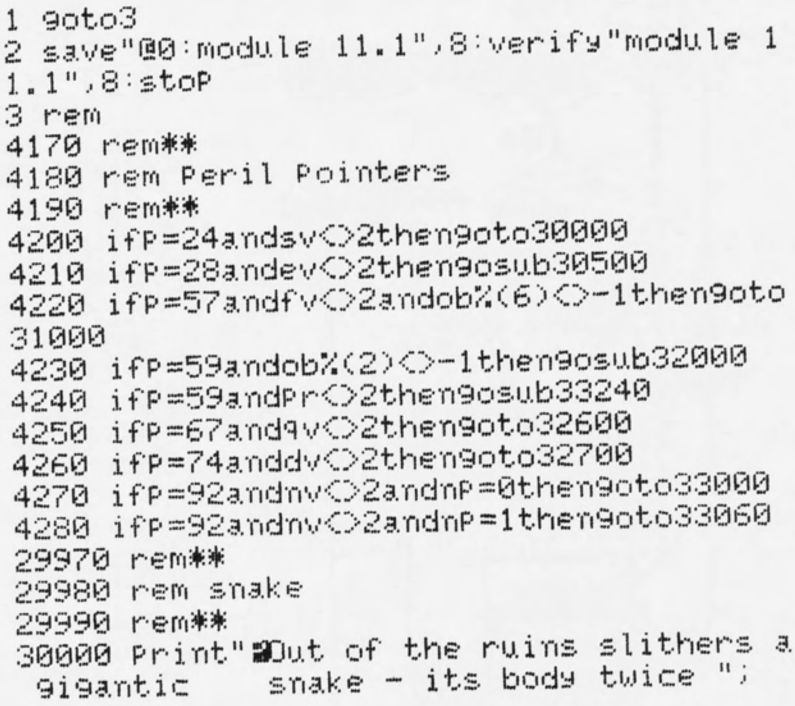
The values I have set for these variables are rather arbitrary because I tended to create them as I went along, and sometimes I chose 1 and sometimes 2. Most variables are set to 0 at the beginning of the program (Line 290 again) which means on the first time round the peril will be present.

## We seem to have a problem here

We can now return to the keyboard and start to program again. The first objective is to make the peril appear on the screen when the player enters the appropriate location.

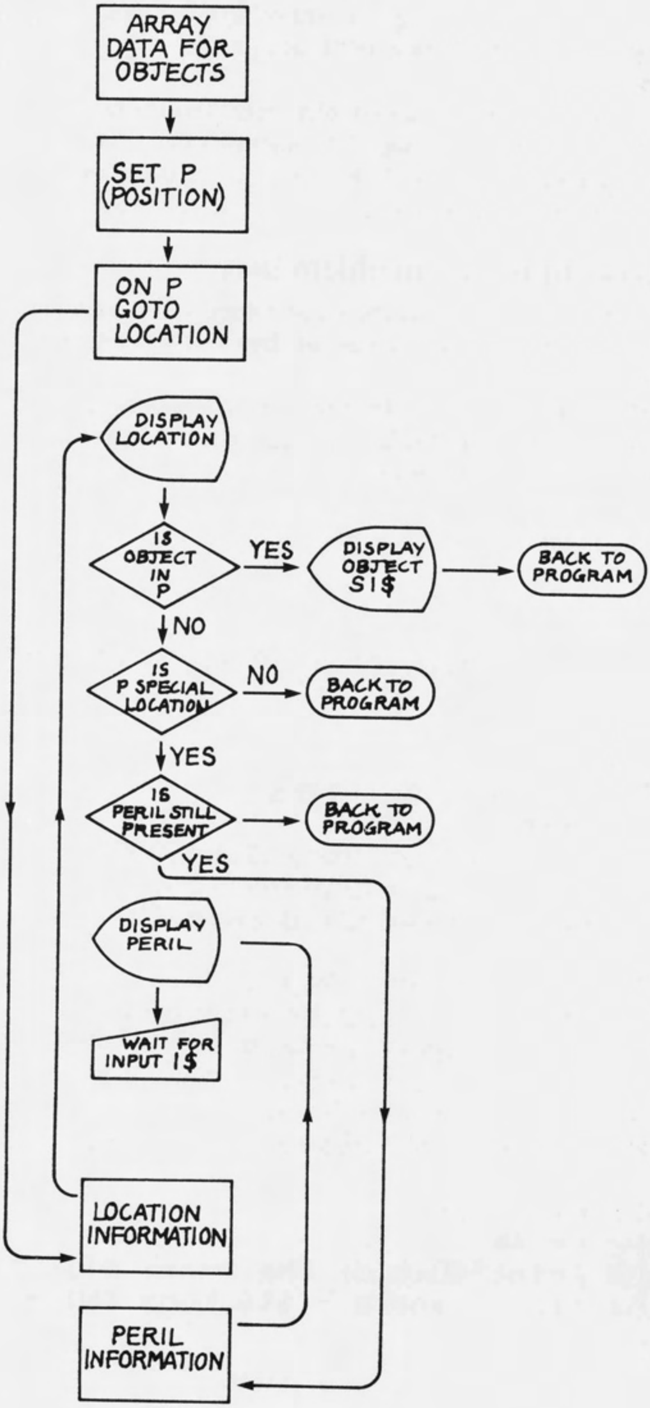
**Flowchart 11.1** illustrates the principles of Modules 9.2 and 11.1 — how you should see the right display on screen when you encounter the various objects and perils in a location.

MODULE 11.1



```
1 goto3
2 save"@0:module 11.1",8:verify"module 1
1.1",8:stop
3 rem
4170 rem**
4180 rem peril pointers
4190 rem**
4200 ifp=24andsv<>2thengoto30000
4210 ifp=28andev<>2thengosub30500
4220 ifp=57andfv<>2andob%(6)<>-1thengoto
31000
4230 ifp=59andob%(2)<>-1thengosub32000
4240 ifp=59andpr<>2thengosub33240
4250 ifp=67andqv<>2thengoto32600
4260 ifp=74anddv<>2thengoto32700
4270 ifp=92andnv<>2andnp=0thengoto33000
4280 ifp=92andnv<>2andnp=1thengoto33060
29970 rem**
29980 rem snake
29990 rem**
30000 print"■Out of the ruins slithers a
 gigantic     snake - its body twice ";
```

## Flowchart 11.1 Object and peril displays



```
30010 print"the thickness of your own. I
t blocks your only exit.":goto5000
30470 rem**
30480 rem eel
30490 rem**
30500 print"▘It is fairly deep in the wa
ter.":return
30970 rem**
30980 rem poisonous air
30990 rem**
31000 print"█▌Suddenly you become aware
that you are choking.  The ";
31010 print"atmosphere is poisonous.  Un
less you do something ";
31020 print"very quickly youwill die..."
31030 form=1to5000:nextm
31040 print"▓Start thinking ";n$:print"▓
*";
31050 forl=0to24:pokesc+l,0:nextl
31060 fori=1to30:print"*";
31070 pokevo,15:pokea1,41:pokes1,89:poke
w1,17:pokeh1,56:pokel1,99
31080 fort=1to200:nextt:pokevo,0:pokea1,
0:pokew1,0:pokes1,0
31090 nexti
31100 printchr$(13):print"▌█Too late !!"
:print"Too late !!":goto190
31970 rem**
31980 rem crystal
31990 rem**
32000 print"▌█You notice that the energy
 crystal is missing from the ";
32010 print"energy pod.  Without ityou c
annot fly the ship !! It must have ";
32020 print"been taken by some-one! You
have to findit before you can escape.
32030 return
32570 rem**
32580 rem quicksand
32590 rem**
32600 print"█▌You are sinking into quic
ksand.        In seconds it is about ";
32610 print"your knees.Betterthink of so
```

```
mething !!":q=0:goto5000
32670 rem**
32680 rem dinosaur
32690 rem**
32700 print"■From the trees at the side
of the road amassive ";
32710 print"dinosaur moves into your pat
h.  You cannot get past.":goto5000
32970 rem**
32980 rem natives
32990 rem**
33000 print"■A bunch of hostile natives
appear at theentrance to the ";
33010 print"hut - blocking the way. They
 advance - spears pointing at ";
33020 print"your  throat.  They mean bus
iness ";n$:q=0:goto5000
33030 rem**
33040 rem back to hut again
33050 rem**
33060 print"■Back again !! I hope you've
 worked out  how to give aurora ";
33070 print"that little kiss thatwill he
lp get her out of there !!"
33080 na=0:q=0:goto5000
33210 rem**
33220 rem no aurora
33230 rem**
33240 print"■With Aurora missing you ca
nnot completeyour mission.":return
```

Lines 4200–4280: This section of the program 'points' to the part which will display the appropriate error message. Thus in Line 4200 if you are in Location 24 (the city) and SV does not equal 2 (which it won't first time as we set it to 0 at the beginning of the program) then you GOTO 3000 (which will display the message "A SNAKE COMES TO GET YOU" or something similar.

Once you defeat the snake the variable SV is set to 2 — so whenever you return to Location 24 the snake will not reappear. The other conditions, of course, refer to the other locations.

Line 4220 is an example of the AND condition. If you leave the spaceship AND FV is not 2 (in other words you haven't eaten the nuts) AND

OB(6)< > − 1 (in other words you don't have your spacesuit in the inventory) then GOTO the part of the program that will say "POISONOUS ATMOSPHERE".

Lines 4230 and 4240 refer to both Aurora and the crystal — quite a complicated bit of programming which I will return to in Module 13.1.

In Lines 4270 and 4280 if you are in the native hut for the first time then NV = 0 so the natives will menace you and prevent you from removing Aurora. After 10 tries you are allowed to leave (perhaps to find the nuts) but NP is now set to 1 so that when you return the message will be "BACK AGAIN" instead of the same message on first entering.

This last may seem a refinement — but it looks a lot better when you're actually playing. In fact when you make up your own game often you need to actually play it to find out these little touches that will make it sound better.

For example I originally had the response "I DON'T SEE IT HERE" for a command such as "THROW" which did not involve one of the objects in the object array. Thus if a player typed "THROW HORSE" back would come the response "I DON'T SEE IT HERE". This was fine as long as the player attempted to throw things like horses or Ford Cortinas (in other words crazy objects) but in some of the Locations we talk about 'stones' or 'crockery' — which are not valid objects to throw. So if you type THROW CROCKERY and get the reply "I DON'T SEE IT HERE" when it obviously is here — it looks bad. The answer was to change "I DON'T SEE IT HERE" to "STOP FOOLING AROUND" which catered for objects in both situations.

I feel at this stage that I must make an apology for the fact that however I try and write this part of the book it might seem confusing to some people. This area of the program requires a knowledge of the Adventure which is easier for me (the author of the plot) as I've been living and breathing the story for a month or so by the time I get to this part of the program. If you are a little confused by the complexity of the story and the related modules I will be dealing with each Peril as a separate module — and this should clarify the situation. For now please hang in there and mainly stick with the fact that this area of the program is going to create the PRINT statements that will appear on the screen and then prevent the player from escaping by simply moving out of the location.

When you come to construct your own it will be much easier as you will have conceived and created your own plots and perils — and will be more familiar with them.

Lines 30000–30010: The display for the snake.

Line 30500: This line refers to the crystal only. For the moment I will leave the eel as it needs a whole chapter for itself.

Lines 31000–31100: To liven the poisonous atmosphere peril up a bit I added a little routine which simulates time ticking away as you are presumably choking to death. First the message "YOU ARE CHOKING..." comes up and the program pauses (Line 31030) to give you time to read it. Then another message in purple comes up "START THINKING..." and an asterisk is printed on the line below (Line 31040).

Line 31050 nullifies all sound variables, and we start a 30 times loop in Line 31060 which contains a sound effect (Lines 31070 and 31080) within it.

The effect is a high-pitched beep as each '*' appears on screen and the stars begin to track across adding another '*' until 30 have been printed. Then the message "TOO LATE — TOO LATE" appears and the program goes to the YOU ARE DEAD section (Module 11.3 — still to be added).

There is no way to escape this peril. Once you have stepped into the poisonous atmosphere you are doomed. And, of course, if you should "get" your spacesuit when you first leave but "remove" it later (to kiss Aurora say) without first eating the nuts this routine will still come up. But we'll add that to the program later.

Lines 32000–32020: Standard display for inside the ship if you have not found the crystal.

Lines 32600–332610: The quicksand display. Note Q = 0 here, which is setting up a 'counter' system. Briefly this will allow a certain number of attempts to escape before ending the game with a "YOU HAVE SUNK" message.

Lines 32700–32710: The dinosaur display. The difference between this section and the others is that all the dinosaur does is block your way south — it doesn't actually kill you.

Lines 33000–33080: These two sections deal with the natives — first when you enter Location 92 initially and secondly if you fail to rescue Aurora on the first attempt and return. The variable NA prevents escape from the hut until you have tried 10 times to rescue Aurora (note Q = 0 again).

Line 33240: The final line in the module is a simple display for Location 59 (inside your spaceship) if you have not rescued Aurora.

*Testing Module 11.1*

To test this module you need to RUN the game and start to move around the various locations. The first thing you'll notice is that on entering the game again in the first location (the cabin of your spaceship) you will find a full page of text as we've added the sections about the crystal and Aurora. Do NOT pick up the blaster or the spacesuit but just go WEST and as you leave the ship you should find the poisonous atmosphere choking you... up come the asterisks and you watch helplessly as your fate ticks away. As you end this section the error message UNDEF'D STATEMENT ERROR IN 31100 should appear.

Now reRUN and this time GET your spacesuit and then wander around the planet. As you come across each appropriate location so the peril should be displayed. At the moment you can escape by typing EAST or WEST (or whatever) to move to the next section. However we need to prevent that in the game proper — and that will be the role of the module module.

## No exit that way

You noticed how easy it was to escape from the peril when you were testing Module 11.1 — so how can we prevent this in our game? Quite simply we need a routine between the INSTRUCTIONS input and the MOVEMENT ROUTINE and Module 11.2 does just that.

MODULE 11.2

```
1 goto3
2 save"@0:module 11.2",8:verify"module 1
1.2",8:stop
3 rem
5970 rem**
5980 rem no exits
5990 rem**
6000 ifp=24 andsv<>2then6110
6010 ifp=28andes=2then6500
6020 ifp=28andev<>2then6110
6030 ifp=67andqv<>2then6110
6040 ifp=74anddv<>2then6170
6050 ifp=92andnv<>2andna=1then6500
6060 ifp=92andnv<>2andna=0then6110
6070 goto6500
6110 ifi$="n"ori$="e"ori$="s"ori$="w"the
ngosub19000:goto6130
6120 goto6150
6130 print"█You can´t escape that way !"
```

```
:Print"█You'll have to think of ";
6140 Print"█something else !":goto5000
6150 if i$="run"thengosub19000:Print"█The
re's no escape that way.":goto5000
6160 ifi$="walk"thengosub19000:Print"█Th
ere's no escape that way.":goto5000
6170 ifi$="south"ori$="s"thengosub19000:
Print"█No dice ";n$
6180 ifi$="south"ori$="s"thenPrint"█Ther
e's a dinosaur in the way !":goto5000
```

Lines 6000–6070: These lines test to see if you are in one of the special locations — and if you are the program will direct you to the PREVENT EXITS routine. If you are not then Line 6070 bypasses the 'prevent movement' routine.

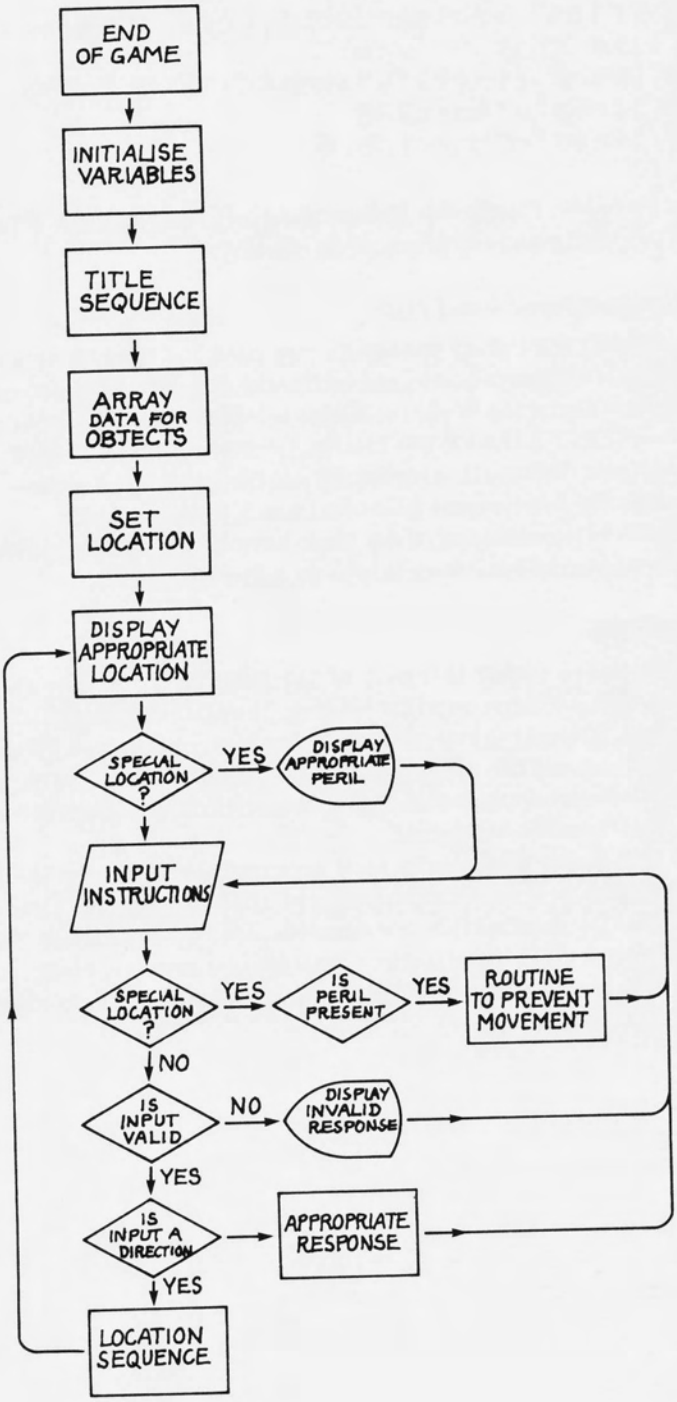
Lines 6010 and 6050 are slightly different in that these also pass the NO MOVEMENT area as well. In the case of Line 6010 the reason is that ES = 2 (which means the eel has been killed or has not yet been encountered). This should become more obvious in the section on the eel peril. Line 6050 is different in that the variable NA has been set to 1 (which means it is the second time you will have visited this Location). I thought it better not to limit movement on a return visit to the native hut to 'get' Aurora, especially as this may be due to the fact that you have 'forgotten' (literally) to type in "GET AURORA" after succeeding in rescuing her and have to go back just to 'get' her.

Lines 6110–6180: These are simply the different responses to various attempts to escape from the location by typing RUN or WALK. Note also that Lines 6170–6180 only stop a southwards move to try and get past the dinosaur — it is still possible to go north again.

MODULE 11.3

```
1 goto3
2 save"@0:module 11.3",8:verify"module 1
1.3",8:stop
3 rem
160 rem**
170 rem you're dead
180 rem**
190 Pokev+21,0:Print"████Well you blew i
t that time, didn't you ";n$
```

**Flowchart 11.2 Basic Adventure Game**



```
200 print"███Luckily I have the power to
 revive you and you can have";
210 print" another go.":print"███Would y
ou like that ? (y/n)"
220 getr$:ifr$<>"y"andr$<>"n"then220
230 ifr$="y"then280
240 ifr$="n"then17570
```

Lines 190–240: Finally this little module is the end message if you should come to a sticky end anywhere along the way.

*Testing Modules 11.2 and 11.3*

If you repeat the testing procedure you used for Module 11.1 then you should find that you can no longer escape should you wander into a location with a peril. On typing W say the response ‘‘YOU CAN’T ESCAPE THAT WAY — YOU’LL HAVE TO THINK OF SOME-THING ELSE’’ should come up (plus the burp!). Also should you forget your spacesuit — not only do you die but now you are given the chance to play again.

As most of the elements of the game have now been covered **Flowchart 11.2** illustrates the basic structure of an Adventure game.

## Summary

As I mentioned earlier this part of the book will probably seem more complex than it really is on a first reading. I have tried to simplify it as much as possible, but really a knowledge of what is to come helps a great deal, so it’s best to program this section in and move ahead — returning to the explanation of what is happening line by line when you have a better idea of how I coped with the various hazards.

It is also about this time in your programming that you should first become aware of the tremendous power you hold (as the programmer). Perhaps I shouldn’t dwell on this, but there’s no denying the fact that YOU control the game, YOU control just how your player can move, where he can go, and what he can do. No matter how you feel about this — there’s something slightly attractive about it.

# CHAPTER 12
# Nice day for a dip

In any Adventure program worth its salt you are dealing with a pretty large chunk of programming — and if you are using BASIC then as the plot thickens so you may find you are running out of space between modules (because unforeseen problems will arise). This calls for a bit of judicious renumbering, but even if you have a Programmers Aid you can still run into trouble. I found this starting to happen to me, so I devised a special chart (see **Figure 12.1**) which enables me to estimate how far ahead I need to start my numbering for the appropriate routine or module.

It is also essential when creating a program with a large number of GOTOs and GOSUBs to be able to put line numbers in as you go along, otherwise you are sure to forget some later — with a heap of extra work tracking down your error.

No matter how much space you leave there's bound to be too little in some areas of the program, so I do advise you to overestimate (by far more than you'd imagine) when you plan out your numbering. To give you an example of how I used the chart **Figure 12.2** shows where we are in the program so far.

## Killing me softly

One of the commands in any Adventure must be KILL (let's ignore the psychological implications...) so before we enter one of the hardest sequences in the game — the eel peril in the lake — we need to enter the KILL module.

MODULE 12.1

```
1 goto3
2 save"@0:module 12.1",8:verify"module 1
2.1",8:stop
3 rem
12270 rem**
12280 rem kill
12290 rem**
12300 ifno$="aurora"thengosub19000:print
```

**Figure 12.1 Module-planning chart**

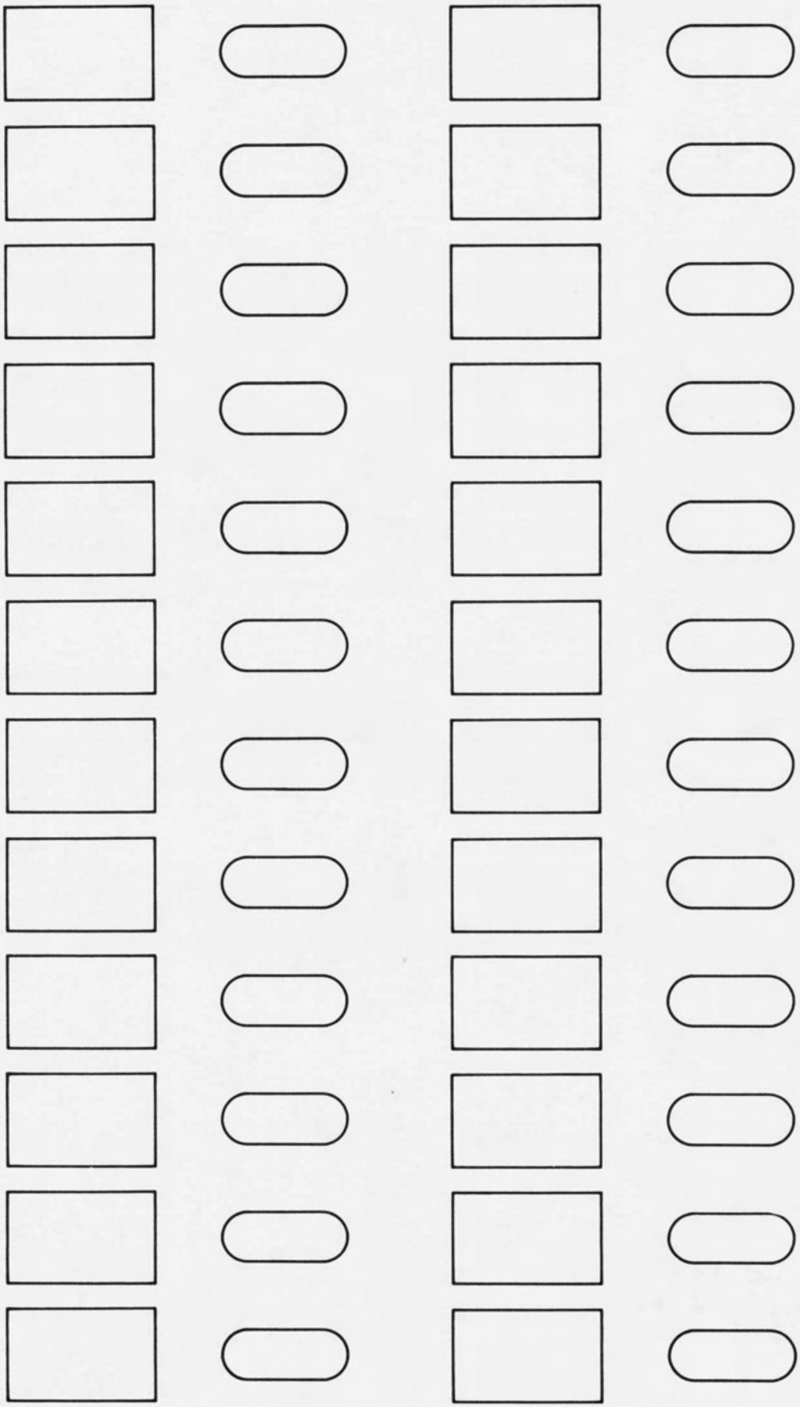
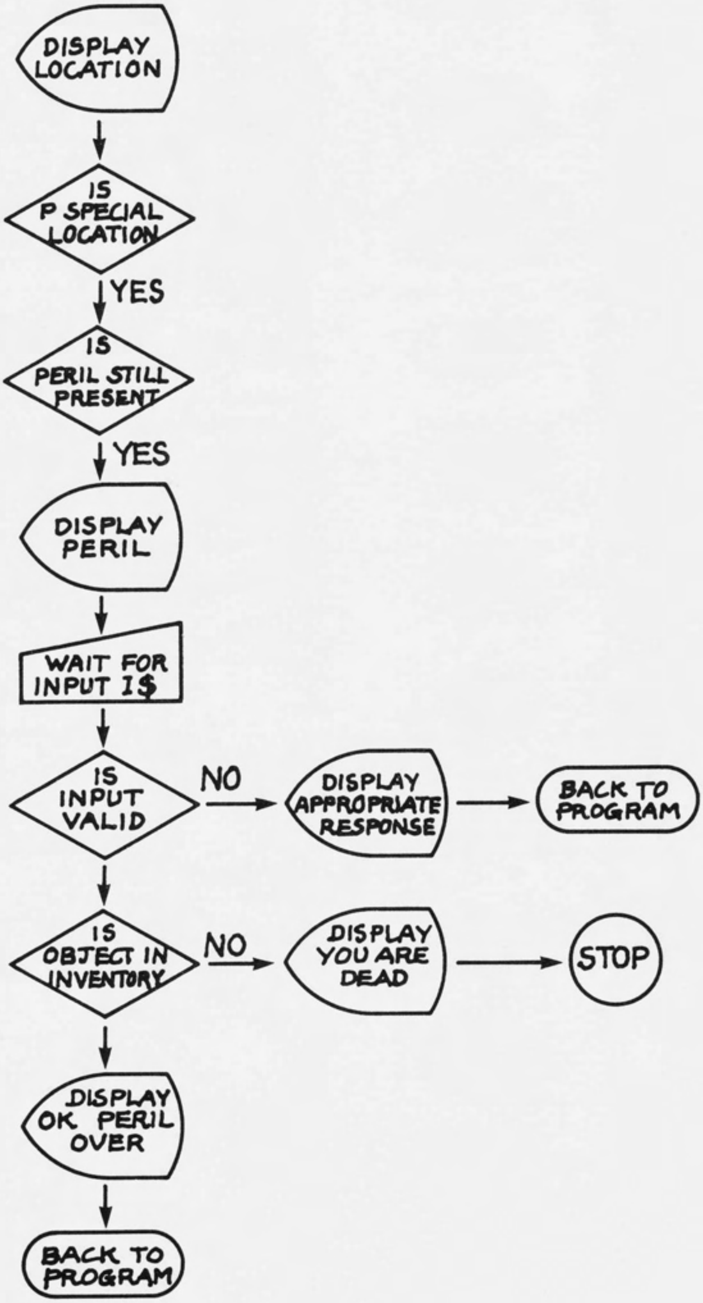
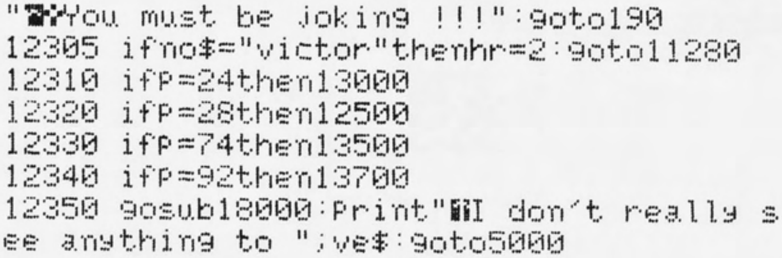


**Figure 12.2 Subroutine start numbers for Nightmare Planet**

| | | | |
|---|---|---|---|
| PRE-CREDITS | 130 | PERIL SECTION | 12500 |
| TITLES PAGES | 1000 | GENERAL INSTRUCTIONS | 15000 |
| OBJECTS AND MOVEMENT | 3700 | LOAD SAVE+QUIT | 16000 |
| INPUT "INSTRUCTIONS" | 5000 | LOCATIONS | 20000 |
| PREVENT MOVEMENT | 6000 | PERIL DESCRIPTIONS | 30000 |
| I$ COMMANDS | 6500 | SPRITES | 35000 |
| MOVEMENT ROUTINES | 7200 | | |
| VE$ COMMANDS | 7500 | | |
| INVENTORY | 10000 | | |
| GET | 11700 | | |
| DROP | 12000 | | |
| | | | |

**Flowchart 12.1 Coping with a peril**





```
"▚▚You must be joking !!!":goto190
12305 ifno$="victor"thenhr=2:goto11280
12310 ifp=24then13000
12320 ifp=28then12500
12330 ifp=74then13500
12340 ifp=92then13700
12350 gosub18000:print"▚I don't really s
ee anything to ";ve$:goto5000
```

Line 12300: It occurred to me that some bright spark could write ''KILL AURORA'' just to see what would happen so I added this line to discourage such 'heresy'. After the response ''YOU MUST BE JOKING'' the program goes to the subroutine at 190 (which is the 'You're dead!' section) and ends the game.

Line 12305: This line is in case someone types ''KILL VICTOR'' (as if anyone could!) and will send the program to the HELP section where a nasty surprise waits for the player (see Module 14.1).

Lines 12310–12340: These lines refer to the only locations where you might try to kill something, and refer the program on to the appropriate section (which we'll be adding later).

Line 12350: A standard response should you try and KILL anything anywhere else along the way.

## Eel take the low road

Sorry about the title to this section of the chapter — poetic licence I suppose.

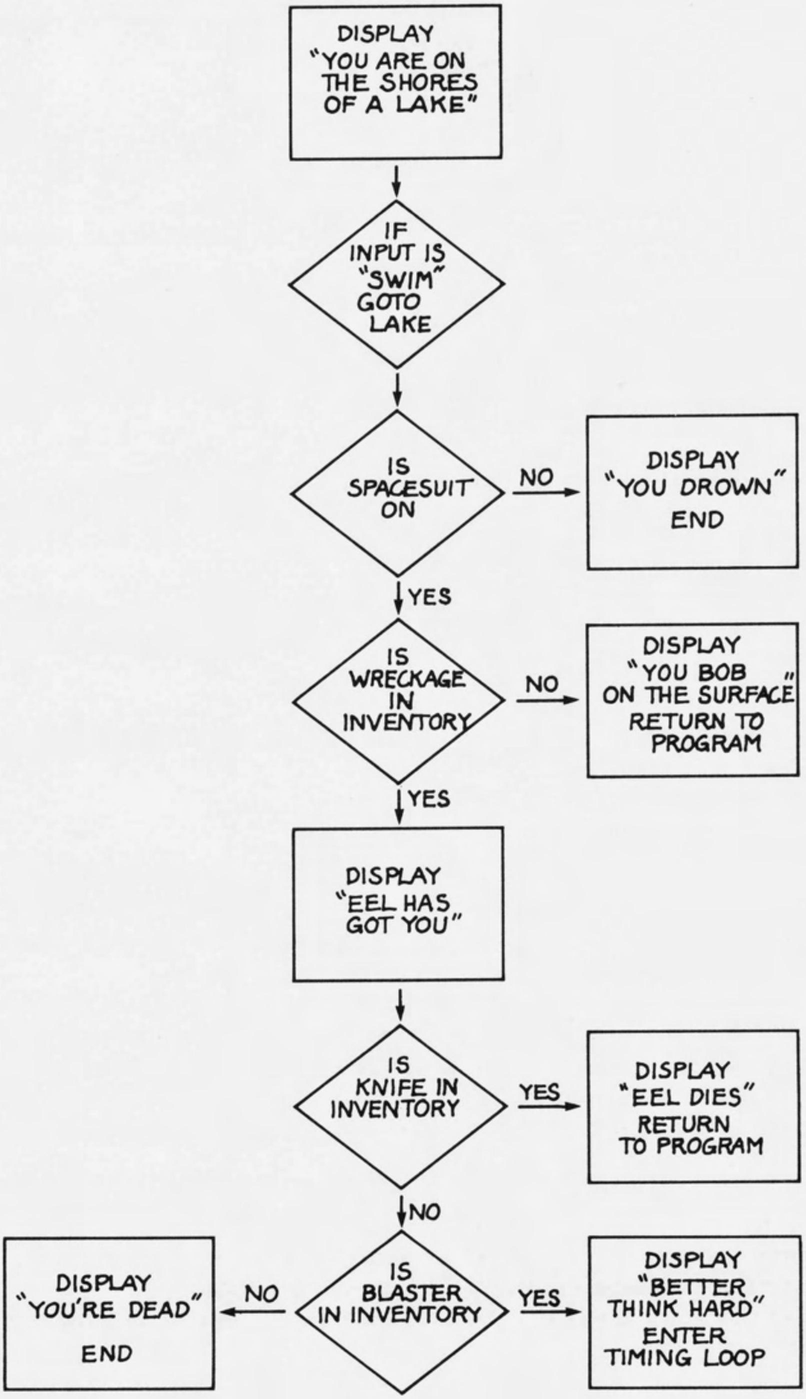
Now we can begin to add all those nasties that will make life more interesting for our player. I've started with perhaps the hardest from a programming point of view — so that it's all downhill from here on.

When I say the hardest — the actual copying of lines for you isn't hard at all. But thinking up and planning this section was the most convoluted for me — and may be the hardest for you to understand because of the number of possible variations in reply that the player could make (in an effort to escape the hazard we have set for him) and the resulting number of possibilities we have to program.

In fact it is trying to anticipate all the words that the Adventurer will use in his efforts to escape that may give you most aggravation — and trying to fit them all into your Adventure. Whatever happens, you'll never cope with all eventualities so its best to accept the fact right at the start.

Let's look at the way I've designed coping with any peril in **Flowchart 12.1**.

**Flowchart 12.2 The eel peril**


DISPLAY "YOU ARE ON THE SHORES OF A LAKE"
IF INPUT IS "SWIM" GOTO LAKE
IS SPACESUIT ON
NO
DISPLAY "YOU DROWN" END
YES
IS WRECKAGE IN INVENTORY
NO
DISPLAY "YOU BOB ON THE SURFACE" RETURN TO PROGRAM
YES
DISPLAY "EEL HAS GOT YOU"
IS KNIFE IN INVENTORY
YES
DISPLAY "EEL DIES" RETURN TO PROGRAM
NO
IS BLASTER IN INVENTORY
NO
DISPLAY "YOU'RE DEAD" END
YES
DISPLAY "BETTER THINK HARD" ENTER TIMING LOOP

Probably the easiest way to understand this is to go through the chart as if asking the computer questions or telling it commands — as follows:

QUESTION 1: Is the location special?
This refers to locations 24,28,57,67,74 and 92 and is covered by Module 11.1.

QUESTION 2: Is the peril still present?
The variables SV,EV,DV etc (Module 11.1 again) control this, by their value. If set to 0 then the peril is present. If set to 2 then the peril is not. At this stage the peril will be displayed on the screen.

QUESTION 3: Is the input valid?
The program now checks to see if the input I$ is a valid one. If not then the appropriate response is given. This really covers mistakes, typing without a space, etc dealt with in Module 10.2.

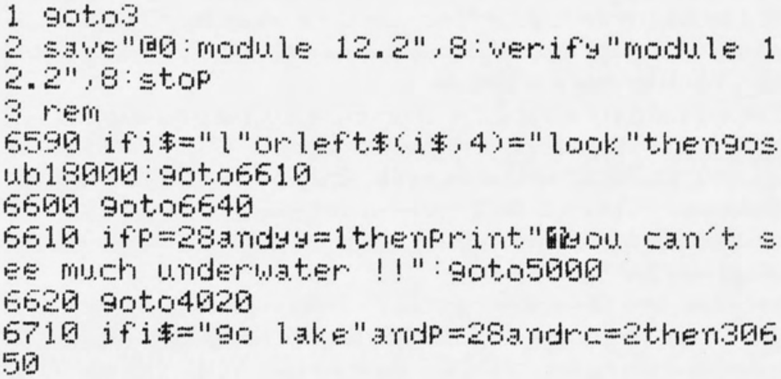
QUESTION 4: Is the object in the inventory?
This is not quite the whole truth, as you can escape some perils without an object in the inventory (eg the dinosaur). However in the main the problem is overcome by "KILLING" the monster and to do this you need a blaster or a knife, so the first thing the program does is check to see if you have the appropriate object. If you do have the right weapon then the program says "OKAY" and you are saved (and the variable is set to 2 to make it safe next time you wander into that location). If you do not have the right object you may find you're dead!

Of course this flowchart is oversimplified, but it forms the basic strategy I have used for dealing with most of the perils.

**Flowchart 12.2** illustrates these principles in Location 28.

## Into the swim

MODULE 12.2



```
1 goto3
2 save"@0:module 12.2",8:verify"module 1
2.2",8:stop
3 rem
6590 ifi$="l"orleft$(i$,4)="look"thengos
ub18000:goto6610
6600 goto6640
6610 ifp=28andyy=1thenprint"█you can't s
ee much underwater !!":goto5000
6620 goto4020
6710 ifi$="go lake"andp=28andrc=2then306
50
```

```
6720 ifleft$(i$,4)="swim"andp=28andrc=2t
hen30650
6730 ifleft$(i$,4)="dive"andp=28andrc=2t
hen30650
6740 ifi$="go lake"andp=28then30510
6750 ifi$="go shore"andp<>28then7810
6760 ifi$="go shore"andp=28then6780
6770 goto6800
6780 ifes=2thengosub18000:goto20540
6790 ifes<>2thengosub19000:goto6130
6800 ifleft$(i$,4)="swim"andp=28then3051
0
6810 ifleft$(i$,4)="dive"andp=28then3051
0
6820 ifleft$(i$,2)="go"thengosub18000:pr
int"█Just type direction ?":goto4320
6830 ifi$="get knotted"thengosub19000:pr
int"█And you ";n$:goto5000
```

When we first arrive in Location 28 we aren't actually *in* the lake at the start. Thus we need some way of actually getting into the water. I've tried several different people on what they would write, and the usual replies were SWIM, GO LAKE, or DIVE IN. This module caters for all these commands.

Lines 6590–6620: A little confusing these four lines because they're taken slightly out of context here. They refer to the command LOOK. First Line 6590 is the I$ command. Then Line 6600 takes you past this section of the program (to yet another area we haven't put in yet).

Line 6610 refers to the response "LOOK" if you are in Location 28 and *under* the water (hence the variable YY = 1 which is set once you enter the water). Line 6620 is the standard response to the command "LOOK" and just takes the program to the movement routine so that "YOU ARE IN A DESERT" or whatever is displayed.

Let me expand on this slightly. At times in the program you may find you are typing in quite a number of commands (such as "CLIMB TREE" or "EXAMINE STONES") and as the replies scroll up the screen so your location disappears. When you decide to move you may have forgotten where you are or which direction you can take, so typing "LOOK" will restore the text telling you this.

Now in Location 28 we have a special problem — part of this location is the lake. Thus if we are under the lake (in the grips of the eel) it would look pretty stupid if on typing "LOOK" the message "YOU ARE BY THE

SIDE OF A LAKE'' came up again. Lines 6610 and 6620 prevent this.

Before moving onto the next section let's just look again at the variables involved in Location 28 and the Eel peril.

**Table 12.1: Variables for the eel peril in Location 28**

CV  Set at 2 when the Crystal is found.
EV  Set at 2 when the Eel has been killed.
ES  Set at 2 when the Eel has been killed.
    Set at 1 when the Eel has got you to prevent movement.
RC  Set at 2 to stop movement from Location 28 by swimming.
YY  Set at 1 when the eel has you to stop a 'look' response.

Lines 6710–6730: These lines apply when RC = 2 (in other words the eel has got you) and you try to escape by typing ''SWIM'' or ''DIVE''. There will be a response ''THE BEAST HAS YOU GRIPPED TOO TIGHTLY TO ESCAPE'' (in Module 12.4 — still to come).

Lines 6740, 6800 and 6810: These are the lines that take you into the lake. So if you are in Location 28 and you type one of those three commands (SWIM, DIVE OR GO LAKE) you will go to Module 12.4.

Line 6750: This line is for the people who type ''GO SHORE'' when they're not in Location 28 (and someone is sure to do it!) It directs the program to a response ''YOU'RE ACTING A BIT STRANGE. . .''.

Line 6760–6770: If you are in the lake and you type ''GO SHORE'' then we need to check various points before allowing you to escape — otherwise someone in the grip of the eel might be able to escape by just saying ''GO SHORE'' and we can't have that! Line 6770 is for commands which do not refer to the lake and the eel and bypasses the next few lines.

Lines 6780–6790: Okay, you're in the lake and you type ''GO SHORE''. First we check to see if ES = 2 (ES is set to 2 after the eel has been successfully killed). If ES is set to 2 then we go to Line 20540 which is the start of Location 28 description — in other words the display ''YOU ARE AT THE EDGE OF THE LAKE'' comes up again.

If ES is not 2 then you are still in the grips of the eel — so the program loops back to 6130 (which is the routine to prevent movement in Module 11.2).

Line 6820: This line is not really concerned with this location, but I put it here as it might have been left out otherwise. It is a simple routine in case the player types GO at any time. Note how I have used LEFT$(I$,2) = ''GO'' because this will also cover commands like ''GO WEST'' or ''GO NOW''.

Line 6830: If you're feeling particularly frustrated you might be tempted to write "GET KNOTTED" and this line adds a touch of humour.

MODULE 12.3

```
1 goto3
2 save"@0:module 12.3",8:verify"module 1
2.3",8:stop
3 rem
7520 ifve$="get"andp=28then32070
7530 ifve$="take"andp=28then32070
7550 ifve$="grab"andp=28then32070
7640 ifve$="kill"orve$="smash"orve$="des
troy"orve$="waste"then12300
32040 rem**
32050 rem cannot get crystal
32060 rem**
32070 ifno$="crystal"then32090
32080 goto11700
32090 ifev<>2thengosub19000:print"▒▌You c
an't .....";:form=1to500:nextm
32100 ifev<>2thenprint"....yet !!":print
"It's ▓under▒ the water.":goto4320
32110 ifev=2then11700
```

Suppose you arrive in Location 28, see the crystal, and type "GET CRYSTAL". At the moment nothing would stop the program saying "OKAY" and neat as you like the crystal is in your inventory. This module prevents that.

Lines 7520–7550: Straightforward commands concerning the "GET" command when you are in Location 28. To ensure that you don't get an incorrect response the 64 will bypass the usual GET Module and drop through the program to Line 32070.

Line 7640: This is the line for the command KILL etc, which will take you to Module 12.1 (which we entered at the start of this chapter).

Lines 32070–32080: Line 32070 checks to see if NO$ is "CRYSTAL". If you were in Location 28 and typed "GET" but didn't type "CRYSTAL" (for example you might well have dropped something at a previous visit and now wish to retrieve it) then Line 32080 will direct the 64 to the appropriate section of the program.

Lines 32090–32100: These two lines just prevent you from "GETTING" the crystal if EV is not set to 2 (the eel is still alive).

Line 32110: If the eel is dead (EV = 2) then you are directed back to the GET routine in Module 10.3 and can successfully "TAKE" the crystal.

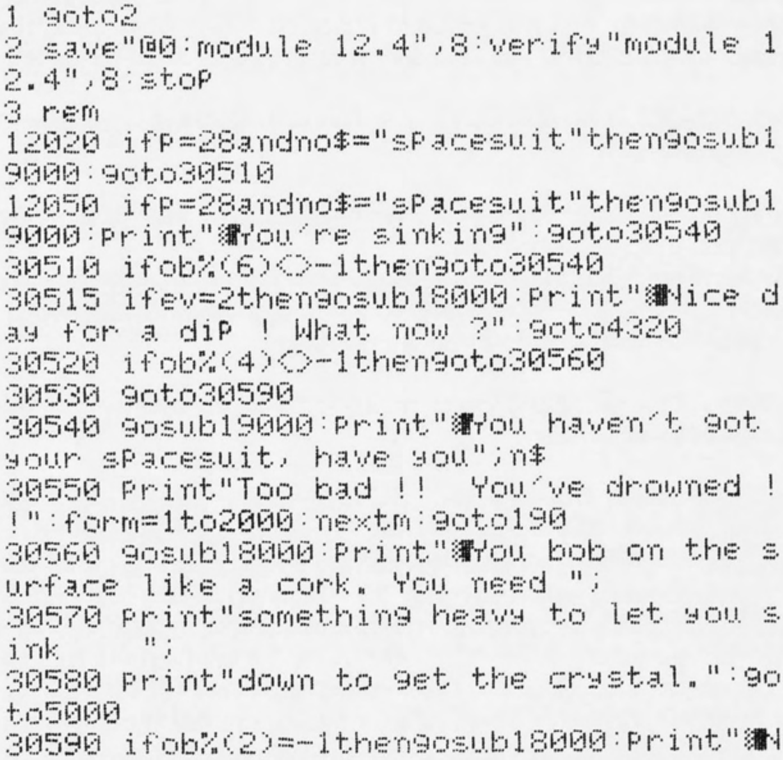
## The eel has got you

We can assume that our player has worked out how to get into the lake to try and retrieve the crystal. There are several possibilities that affect how we treat the next stage at this moment which need more thought for the next module.

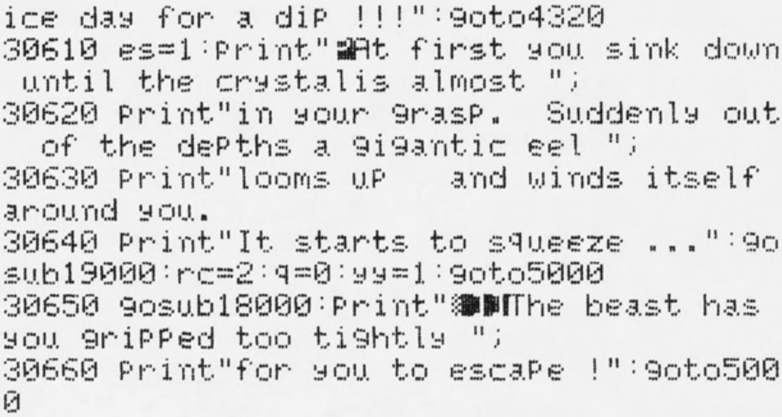
The possibilities are:

1. You are not wearing your spacesuit.
2. You are wearing your spacesuit but you have not yet picked up the wreckage.
3. You are wearing the spacesuit and you have the wreckage.

MODULE 12.4



```
1 goto2
2 save"@0:module 12.4",8:verify"module 1
2.4",8:stop
3 rem
12020 ifp=28andno$="spacesuit"thengosub1
9000:goto30510
12050 ifp=28andno$="spacesuit"thengosub1
9000:print"█You're sinking":goto30540
30510 ifob%(6)<>-1thengoto30540
30515 ifev=2thengosub18000:print"█Nice d
ay for a dip ! What now ?":goto4320
30520 ifob%(4)<>-1thengoto30560
30530 goto30590
30540 gosub19000:print"█You haven't got
your spacesuit, have you";n$
30550 print"Too bad !!  You've drowned !
!":form=1to2000:nextm:goto190
30560 gosub18000:print"█You bob on the s
urface like a cork. You need ";
30570 print"something heavy to let you s
ink     ";
30580 print"down to get the crystal.":go
to5000
30590 ifob%(2)=-1thengosub18000:print"█N
```



```
ice day for a dip !!!":goto4320
30610 es=1:print"■At first you sink down
 until the crystalis almost ";
30620 print"in your grasp.  Suddenly out
  of the depths a gigantic eel ";
30630 print"looms up   and winds itself
around you.
30640 print"It starts to squeeze ...":go
sub19000:rc=2:q=0:yy=1:goto5000
30650 gosub18000:print"▓■▌The beast has
you gripped too tightly ";
30660 print"for you to escape !":goto500
0
```

Lines 12020–12050: These two lines refer to the unlikely event of you removing your spacesuit whilst in the lake. Both refer you to Line 35040 ultimately. They will fit into the DROP section of the program.

Lines 30510–30530: These lines check the inventory to see if you are wearing your spacesuit, if you've defeated the eel, or if you're carrying the wreckage. Line 30530 will take you past the next few lines to the eel section.

Lines 30540–30550: No spacesuit — you drown. There's a short pause and you exit to Line 190.

Lines 30560–30580: If you haven't the wreckage then you 'bob on the surface' and wait for instructions.

The relevance of RC becomes applicable here. At this stage you haven't actually met the eel yet — so RC is still set to 0 and you can escape by typing "GO SHORE" to try and find something heavy.

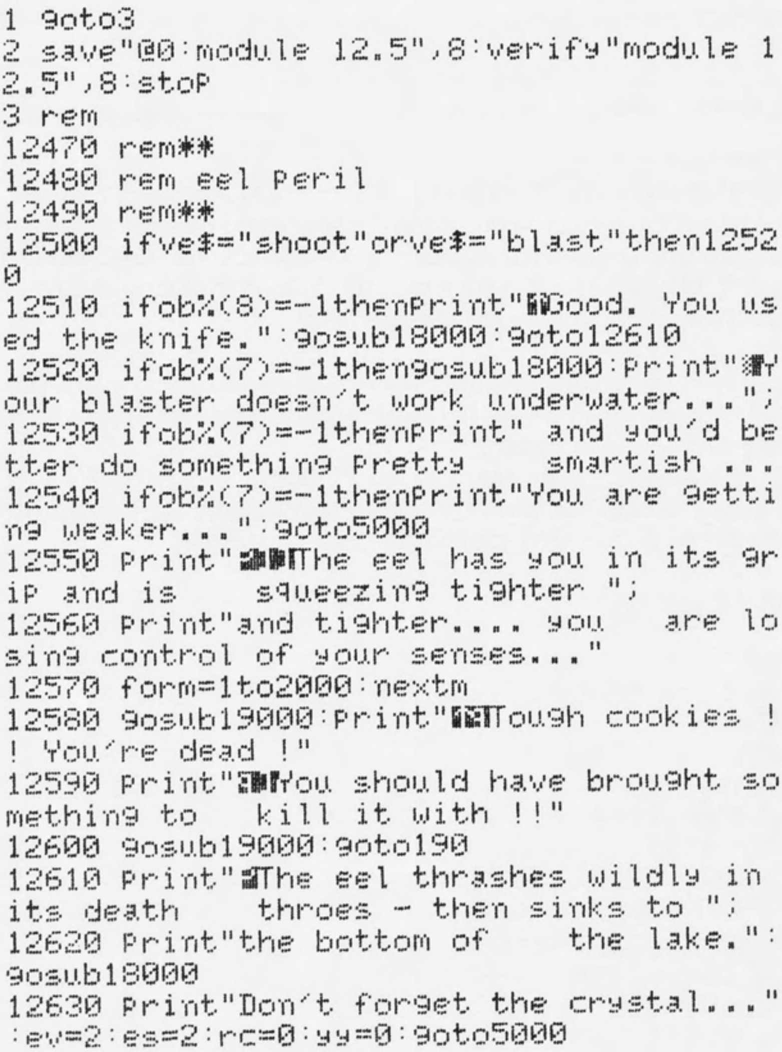
Line 30590: This line allows you to return to the lake after you have the crystal should you so desire.

Lines 30610–30640: This is the main peril — the eel itself. As you sink down into the lake the eel comes up and gets you. The use of variables needs a little explanation here. Before we start we set ES to 1 — which prevents you from moving out of the lake again by typing "S" (Module 11.2, Line 6010). Then we set RC to 2 (which also prevents you from escaping by typing "SWIM" or "DIVE" instead of "N" or "S"). The reason for needing both RC and ES is because of the fact that you could escape with either a direction or the word "SWIM". Q is set to 0 because we are going to count the number of tries the player has to escape before squeezing him to death. Finally YY is set to 1 (this variable prevents the command "LOOK" from eliciting the usual response — Module 12.2).

Lines 30650–30660: These lines refer to the attempts to escape by typing "SWIM" etc. that I mentioned in Module 12.2.

## Success or failure

MODULE 12.5



```
1 goto3
2 save"@0:module 12.5",8:verify"module 1
2.5",8:stop
3 rem
12470 rem**
12480 rem eel peril
12490 rem**
12500 ifve$="shoot"orve$="blast"then1252
0
12510 ifob%(8)=-1thenprint"█Good. You us
ed the knife.":gosub18000:goto12610
12520 ifob%(7)=-1thengosub18000:print"█Y
our blaster doesn't work underwater...";
12530 ifob%(7)=-1thenprint" and you'd be
tter do something pretty    smartish ...
12540 ifob%(7)=-1thenprint"You are getti
ng weaker...":goto5000
12550 print"██The eel has you in its gr
ip and is    squeezing tighter ";
12560 print"and tighter.... you   are lo
sing control of your senses..."
12570 form=1to2000:nextm
12580 gosub19000:print"██Tough cookies !
! You're dead !"
12590 print"██You should have brought so
mething to   kill it with !!"
12600 gosub19000:goto190
12610 print"█The eel thrashes wildly in
its death    throes - then sinks to ";
12620 print"the bottom of    the lake.":
gosub18000
12630 print"Don't forget the crystal..."
:ev=2:es=2:rc=0:yy=0:goto5000
```

Line 12500: This line relates to a command regarding shooting the eel and ensure that you go to the appropriate line. In the case of this peril the blaster will not get you out of trouble — as it won't work underwater.

Line 12510: If you have typed KILL or STAB then this line checks if the knife is in the inventory. If it is then success — the program drops to Line 12610.

Lines 12520 – 12540: These lines are the response if you have no knife but do have your blaster. Having checked the inventory for the blaster you then get the response "YOUR BLASTER DOESN'T WORK UNDERWATER". You are sent back to the Instructions input to try and think of something else.

Lines 12550 – 12600: A cheery section that comes into force in 2 situations:

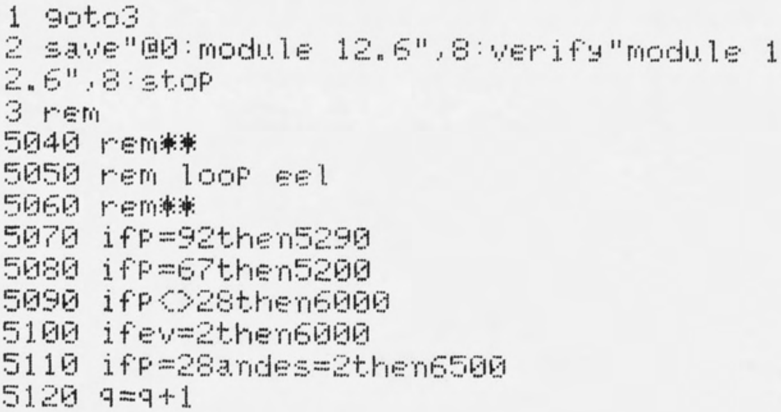
1. If you have no knife or blaster.
2. If you have only your blaster and after 10 attempts you haven't escaped.

Of course there's no way you can escape this little hazard without your knife and as the program stands you cannot escape to get the knife — so it's a bit of a con trick. But giving the player a feeling that he has a chance here helps add to the mystery of the Adventure. (Sneaky, eh?)

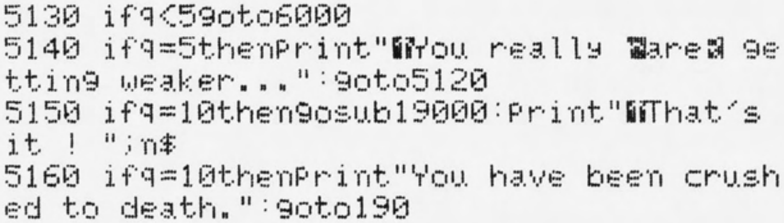
Lines 12610 – 12630: Success at last. The eel dies gloriously and with a helpful hint not to forget the crystal the program releases you from the location and sets all the variables again.

You could if you wanted omit that line about "DON'T FORGET THE CRYSTAL" and let the player get all the way back to the ship only to discover he forgot to "GET CRYSTAL" back at the lake.

MODULE 12.6



```
1 goto3
2 save"@0:module 12.6",8:verify"module 1
2.6",8:stop
3 rem
5040 rem**
5050 rem loop eel
5060 rem**
5070 ifp=92then5290
5080 ifp=67then5200
5090 ifp<>28then6000
5100 ifev=2then6000
5110 ifp=28andes=2then6500
5120 q=q+1
```



```
5130 ifq<5goto6000
5140 ifq=5thenprint"█You really █are█ ge
tting weaker...":goto5120
5150 ifq=10thengosub19000:print"█That's
it ! ";n$
5160 ifq=10thenprint"You have been crush
ed to death.":goto190
```

Lines 5070–5110: These lines ensure that the program bypasses this little loop if you are not in Location 28 and you are not in the grip of the eel. As there are other loops to consider the program passes you to the appropriate area. Best just program it in for the moment and come back later to see exactly where all these lines go.

Lines 5120–5160: This is the actual loop itself. Each time the player gives the 64 a command (such as KILL EEL or whatever) then Q increments by one. When Q =5 there is a little reminder that time may be running out... "YOU REALLY ARE GETTING WEAKER". If you don't get out by the 10th attempt then that's it — folks! Back to square one.

*Testing Modules 12.1–12.6*

Unfortunately you cannot test these modules as some of the lines contain GOTOs which direct the program to a section we will be adding in the next chapter.

## Summary

This chapter is a tour de force of programming in Adventure games, containing a baffling blend of ideas, variables and situations. Don't let it put you off if you're not quite sure what it's all about though — because I did the worst first. I found solving the programming extremely challenging (once I'd worked out what I was doing) and by the time I'd finished the whole Adventure I was quite sorry that the amount of text I'd chosen to add atmosphere ended up by limiting my possibilities in the perils.

Still, I'm already thinking of how to change that for my next Adventure.

# CHAPTER 13
# To talk of many things

The story so far....

Our intrepid hero has managed to pilot his spaceship down to a pancake landing in the vast desert of some distant planet — out of the regular spacelanes. He awakes to find his ship in tatters, his precious energy crystal missing, and worst of all, the love of his life kidnapped by persons unknown. Rescue seems unlikely and now he must try to find both the Princess and his energy crystal if he is ever to see the blue skies and green hills of home.

He sets out across the alien terrain, encountering hazards and mystery in his quest, until at last after some time he discovers his crystal in the depths of a lake, and succeeds in surviving the hazard of a giant eel to retrive it. He takes it back to the ship.

Now read on....

When I had reached this stage in my programming I ran up against a slight problem — when I programmed the locations I had initially forgotten about returning with either the crystal or Aurora (or both) to Locations 58 and 59 (the spaceship). Thus the first time'I returned to the ship I entered (supposedly with Princess, crystal or both) only to be told on the screen that both were still missing.

In other words I needed to alter the display on the screen if I came back with OB%(2) — the crystal — or OB%(9) — Aurora — in the inventory. Now is the time to remedy that deficiency.

## A happy ending

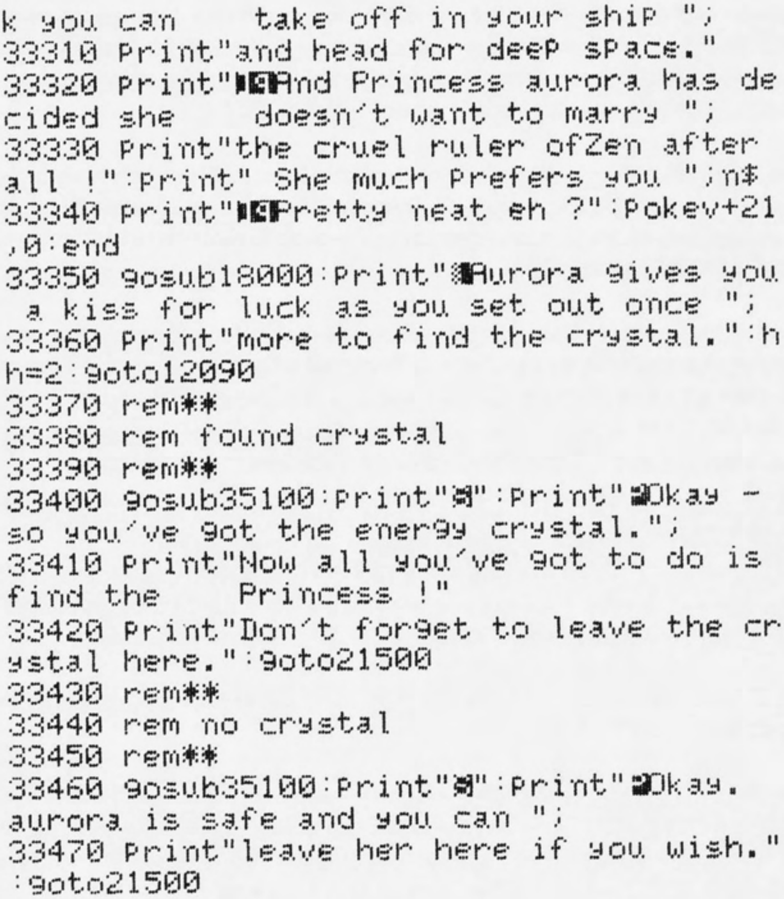
The next Module contains lines missing from the Locations section (Module 9.1) as well as the appropriate lines for 'finding' both Aurora and the crystal. I have based it on the following possible combinations:

1. You return with the Princess but not the crystal.
2. You return with the crystal but not the Princess.
3. You return with both.
4. You rescue the Princess but forget to "GET" her.

I needed this last option because it was possible to beat the natives peril (of which more in Module 13.4) but simply forget to type "GET AURORA". I felt it wasn't necessary to also allow for not "GETTING" the crystal as this was more obvious (and I'd put a reminder in the text).

MODULE 13.1

```
1 goto3
2 save"@0:module 13.1",8:verify"module 1
3.1",8:stop
3 rem
12070 ifp=59andno$="aurora"then33350
21205 ifob%(2)=-1andob%(9)=-1then33280
21210 ifob%(2)<>-1then21240
21220 ifob%(2)=-1thenprint"█Great ! You´
ve got the crystal.":cv=2
21230 print"█Better just slip it into th
e pod in the main cabin.":goto21380
21240 ifpr=2andob%(9)=-1then21330
21250 ifhh=2then21270
21260 ifpr=2andob%(9)<>-1then21350
21280 ifpr=2then21380
21330 print"█Back with aurora - I should
 take her     into the main cabin ";
21340 print"and give her a nice cup of t
ea !!":goto21380
21350 gosub19000:print"█You forgot to █g
et█ aurora back in the     ";
21360 print"native village ";n$
21370 print"Bit stupid - that !! You´ll
have to go  back for her.
21400 ifcv=2andpr=2then33280
21410 ifcv=2andpr<>2then33400
21420 ifcv=0andpr=2then33460
21450 ifpr=2then21500
22000 ifpr<>2then22020
22010 goto22040
33250 rem**
33260 rem found aurora
33270 rem**
33280 gosub35100:print"█"
33290 print"█████Congratulations !!  You
 have succeeded in your mission !!"
33300 print"██After only a few hours wor
```



```
k you can    take off in your ship ";
33310 print"and head for deep space."
33320 print"■■And Princess aurora has de
cided she    doesn't want to marry ";
33330 print"the cruel ruler ofZen after
all !":print" She much prefers you ";n$
33340 print"■■Pretty neat eh ?":pokev+21
,0:end
33350 gosub18000:print"■Aurora gives you
 a kiss for luck as you set out once ";
33360 print"more to find the crystal.":h
h=2:goto12090
33370 rem**
33380 rem found crystal
33390 rem**
33400 gosub35100:print"■":print"■Okay -
so you've got the energy crystal.";
33410 print"Now all you've got to do is
find the    Princess !"
33420 print"Don't forget to leave the cr
ystal here.":goto21500
33430 rem**
33440 rem no crystal
33450 rem**
33460 gosub35100:print"■":print"■Okay.
aurora is safe and you can ";
33470 print"leave her here if you wish."
:goto21500
```

Line 12070: This line comes from the DROP routine and applies if you type LEAVE AURORA in Location 59. It sends you to a little section near the end of the program (see later in this module).

Line 21205: I nearly forgot this line (as you'll see from the odd line number) which checks if you have both Aurora and the crystal in the inventory. If you do then you GOTO Line 33280 which is the final success message of the game.

Line 21210: As you enter Location 58 (the airlock) from the outside this line starts to check the state of your inventory. If you haven't got the crystal this line sends you to Line 21240 where it will check on whether you have Aurora.

Lines 21220–21230: These lines assume that you < do> have the crystal (otherwise you'd have bypassed them) and set CV to 2, print out a nice little message, and direct you to Line 21380 (the last line in this Location section in Module 9.1) which will allow you to move into the cabin. Note that we have missed out all that previous text about the shred of Aurora's gown etc., which is not really applicable any more.

Line 21240: Having checked for the crystal in Line 21210 we now check to see if the Princess has been rescued. This line also checks if you have her in your inventory and if you have then directs you to Line 21330 which will say "BACK WITH AURORA . . .".

Line 21250: This line is the start of a slightly complicated sequence. Suppose you have Aurora with you and you take her into the ship. You now have to pass through the airlock again to go back out to find the crystal, and so all the text in the airlock becomes a bit superfluous (bits of Aurora's gown, etc.). Thus I have broken this text into two sections:

1. The simple words YOU ARE IN THE AIRLOCK (Line 21270) which will appear whenever you pass through the Location 58.
2. The rest of the airlock text (Lines 21290–21320). Thus this line notes if HH = 2 (which is only true once you have taken Aurora into the ship and left her there) and if so sends you to the line YOU ARE IN THE AIRLOCK.

Line 21260: This is the line for those players who have rescued Aurora but forgotten to "GET" her.

Line 21280: This line comes after the YOU ARE IN THE AIRLOCK line and is there to check if you have rescued the Princess from the natives yet (PR = 2). If not then you will get the rest of the text displayed. If you have you bypass and move to the final line of this Location.

Lines 21330–21370: The appropriate responses.

The problem I had with this section was quite considerable, trying to ensure I covered all the eventualities. I think to a degree this area of the program was a perfect example of how a little bit of writing needs an awful lot of thought — and also how the longest part of creating an Adventure is in the tidying and debugging at the end. Don't underestimate this.

Now we start the second section — the same responses for Location 59. The same conditions apply only this time I referred most of the lines to an area further on in the structure of the program.

Line 21400: If you have both the crystal and the Princess together now then we GOTO the success message.

Line 21410: No Princess but we have the crystal.

Line 21420: No crystal but we have Aurora.

Line 21450: This is another line inserted between the text and just cuts out all the stuff about signs of a struggle and the Princess has gone if you have rescued her.

Lines 22000–22010: This couplet is inserted in the native village. You probably won't remember but when you typed this sequence in there was a moment when you heard a woman singing sadly to herself nearby. Of course this is supposed to be Aurora, so if you rescue her we need to omit these lines as you make your way back from the hut.

Lines 33280–33340: What I have loosely termed the success message. Once you have both Aurora and the crystal then you enter this routine. Victor appears, and the congratulations message comes up. Finally Victor is turned off and the program ends.

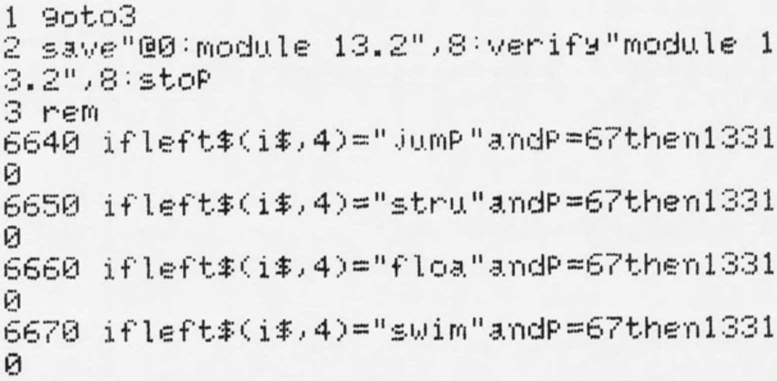
Lines 33350–33360: If you have Aurora but no crystal this message is shown when you "LEAVE" her in Location 59 (remember Line 12070). Note setting HH to 2. Finally you are returned to the DROP section so that Aurora can be 'dropped' and the appropriate subtraction from the inventory will occur.

Lines 33400–33470: These two routines apply to the text that appears when you arrive in the cabin with either Aurora or the crystal — but not both.

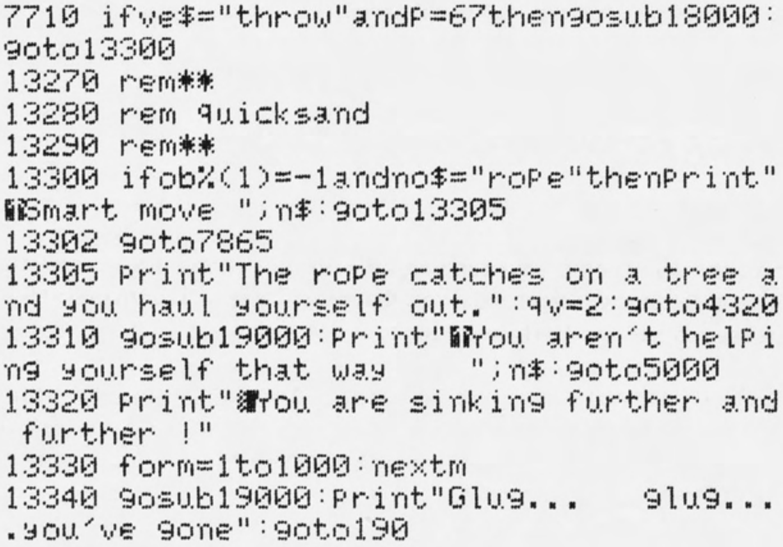
Reading through the explanation to this module I found it pretty confusing (and I wrote it) so let's ease off a bit now with a look at the other perils in the Adventure.

## Marshes and monsters

MODULE 13.2



```
1 goto3
2 save"@0:module 13.2",8:verify"module 1
3.2",8:stop
3 rem
6640 ifleft$(i$,4)="jump"andp=67then1331
0
6650 ifleft$(i$,4)="stru"andp=67then1331
0
6660 ifleft$(i$,4)="floa"andp=67then1331
0
6670 ifleft$(i$,4)="swim"andp=67then1331
0
```



```
7710 ifve$="throw"andp=67thengosub18000:
goto13300
13270 rem**
13280 rem quicksand
13290 rem**
13300 ifob%(1)=-1andno$="rope"thenprint"
■Smart move ";n$:goto13305
13302 goto7865
13305 print"The rope catches on a tree a
nd you haul yourself out.":qv=2:goto4320
13310 gosub19000:print"■You aren´t helpi
ng yourself that way    ";n$:goto5000
13320 print"■You are sinking further and
 further !"
13330 form=1to1000:nextm
13340 gosub19000:print"Glug...   glug...
.you´ve gone":goto190
```

This module is concerned with a few more I$ responses and the quicksand that you will encounter in Location 67 (just south of the ship). To get out of the quicksand you must have the rope (found in Location 15) and you must negotiate this peril to acquire the knife which you need to kill the eel (it's beginning to sound like the nursery rhyme The house that Jack built!!).

Lines 6640–6670: A variety of possible words that you might try.

Line 7710: The VE$ command that will lead to success.

Lines 13300–13305: If you have the rope then you can pull yourself out of the quicksand, set QV to 2, and go on your way. Line 13302 will just prevent the print statement in 13305 should you type THROW BLASTER or anything else.

Line 13310: If you don't happen to have the rope with you — you'll have a few futile chances on a timing loop before you sink down into the depths. Whilst you are trying to get out this response will come up each time you try and "THROW" something.

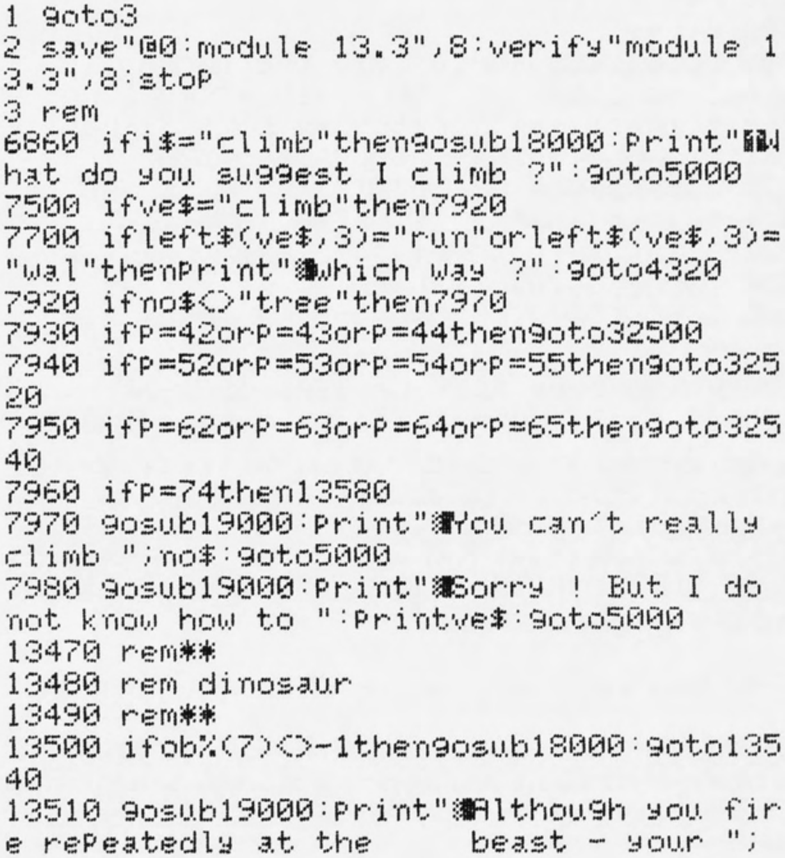
Lines 13320–13340: Finally after you have had four abortive attempts to escape this little routine takes over and "Glug . . . glug . . . You've gone!!". To fully understand this section you need to enter Module 13.5.

The next module concerns the dinosaur that you will meet on your way south towards the native village. I wanted to make the dinosaur just a little bit difficult to pass — but not a deadly peril that would zap you dead. Perhaps one of the little extras about creating your own game is the control you have over the potential player. He or she has no idea just how lethal your peril is likely to be, nor how many red herrings and traps you have sprinkled around.

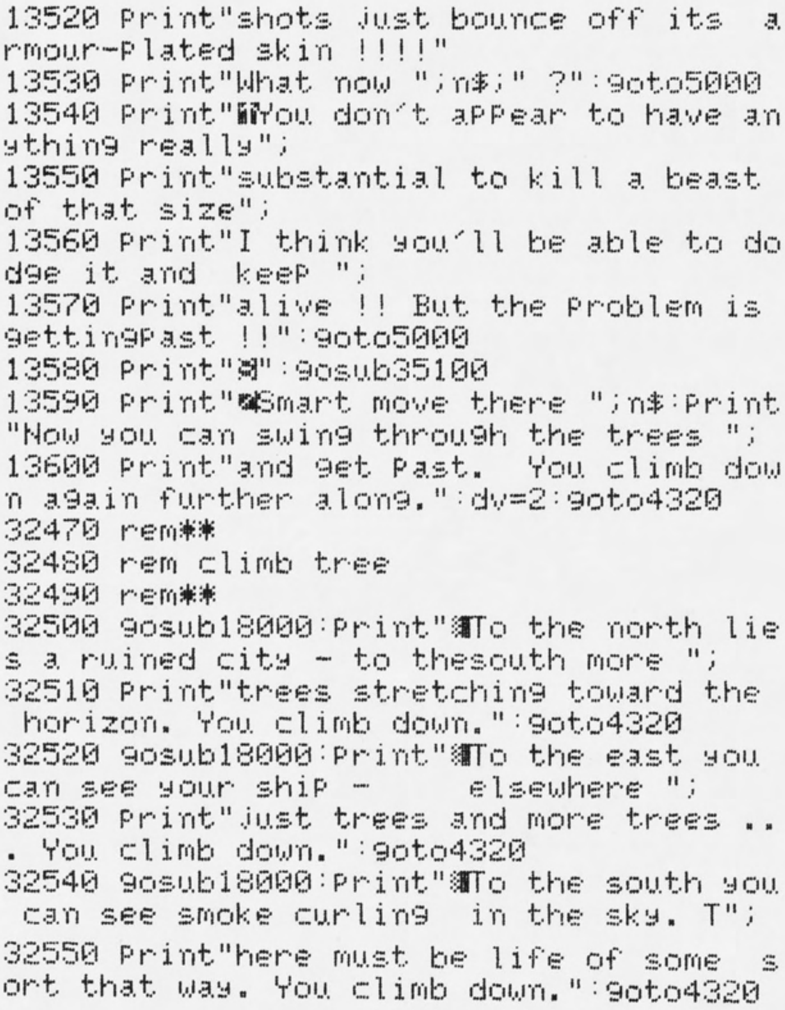
Back to the dinosaur. When I first wrote the story the idea was to kill it with a few well-aimed shots from the blaster — but the more I thought about it the unhappier I became. After all, I had the snake dying rather too easily with a short sharp burst of firepower. After some thought and a browse through my extensive science fiction collection of comics I decided the logical way to escape would be to climb a tree and swing past. And this is how the program ended up.

As the tree-climing idea occurred to me I decided to have a section devoted to climbing a tree in any of the forest locations — and this is what Module 13.3 is all about.

MODULE 13.3



```
1 goto3
2 save"@0:module 13.3",8:verify"module 1
3.3",8:stop
3 rem
6860 ifi$="climb"thengosub18000:print"█W
hat do you suggest I climb ?":goto5000
7500 ifve$="climb"then7920
7700 ifleft$(ve$,3)="run"orleft$(ve$,3)=
"wal"thenprint"█which way ?":goto4320
7920 ifno$<>"tree"then7970
7930 ifp=42orp=43orp=44thengoto32500
7940 ifp=52orp=53orp=54orp=55thengoto325
20
7950 ifp=62orp=63orp=64orp=65thengoto325
40
7960 ifp=74then13580
7970 gosub19000:print"█You can't really
climb ";no$:goto5000
7980 gosub19000:print"█Sorry ! But I do
not know how to ":printve$:goto5000
13470 rem**
13480 rem dinosaur
13490 rem**
13500 ifob%(7)<>-1thengosub18000:goto135
40
13510 gosub19000:print"█Although you fir
e repeatedly at the      beast - your ";
```



```
13520 print"shots just bounce off its  a
rmour-plated skin !!!!"
13530 print"What now ";n$;" ?":goto5000
13540 print"█You don't appear to have an
ything really";
13550 print"substantial to kill a beast
of that size";
13560 print"I think you'll be able to do
dge it and  keep ";
13570 print"alive !! But the problem is
gettingpast !!":goto5000
13580 print"█":gosub35100
13590 print"█Smart move there ";n$:print
"Now you can swing through the trees ";
13600 print"and get past.  You climb dow
n again further along.":dv=2:goto4320
32470 rem**
32480 rem climb tree
32490 rem**
32500 gosub18000:print"█To the north lie
s a ruined city - to thesouth more ";
32510 print"trees stretching toward the
 horizon. You climb down.":goto4320
32520 gosub18000:print"█To the east you
can see your ship -      elsewhere ";
32530 print"just trees and more trees ..
. You climb down.":goto4320
32540 gosub18000:print"█To the south you
 can see smoke curling  in the sky. T";
32550 print"here must be life of some  s
ort that way. You climb down.":goto4320
```

Lines 6860 and 7500: The I$ and the VE$ command for CLIMB. Here would be a good moment to stress the difference between the two. The I$ command is for the player who just types in CLIMB and nothing else, and the response is "WHAT DO YOU SUGGEST I CLIMB?". The VE$ command CLIMB will always be the first part of a command, which could be CLIMB WALL or CLIMB TREE.

Line 7700: This line deals with the possible command WALK or RUN.

Lines 7920 and 7970: The only valid noun you can CLIMB is TREE so if the player types anything else at all this couplet responds with the reply "YOU CAN'T REALLY CLIMB" and then the appropriate noun (eg wall or stones).

Lines 7930–7960: If you climb a tree in a northern forest location you get one response, if you climb a tree in the centre you get another, and if you shin up that tree in the southern area of the forest you will get a third. These lines point to the appropriate response. Finally Line 7960 is for climbing the tree when confronted by the dinosaur.

Line 7980: A final response to the VE$ commands which deals with an invalid word (such as STRANGLE or READ).

Lines 13500 and 13540–13570: If you try and kill the dinosaur without having a blaster then you will get this response.

Lines 13510–13530: If you do have the blaster then these lines tell you the shots bounce off his skin. What now?

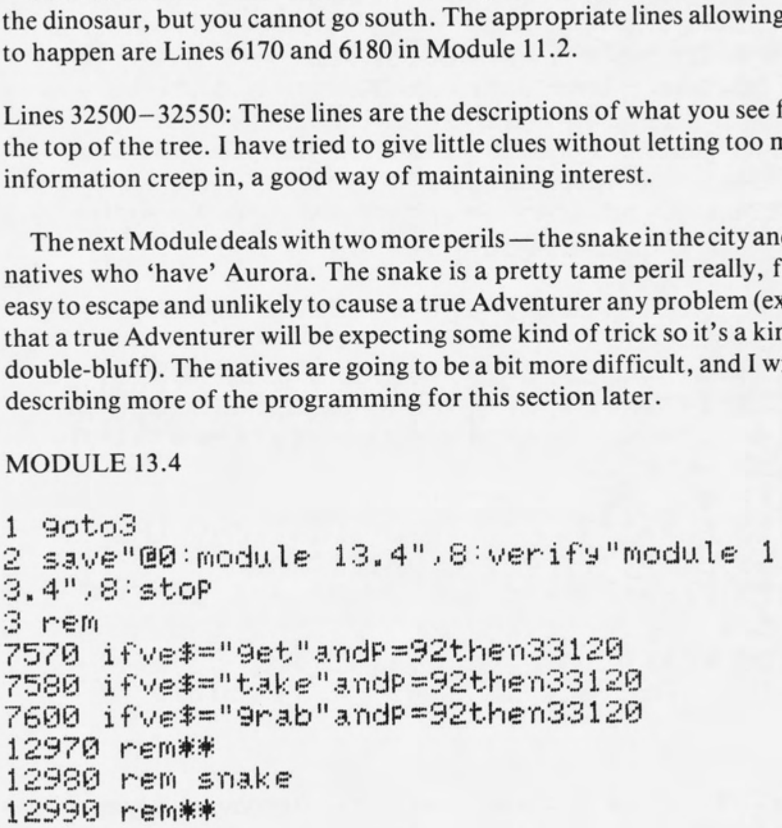
Lines 13580–13600: Success at last. You've climbed the tree and can get past.

Just a note here. In this location you can go north again should you meet the dinosaur, but you cannot go south. The appropriate lines allowing this to happen are Lines 6170 and 6180 in Module 11.2.

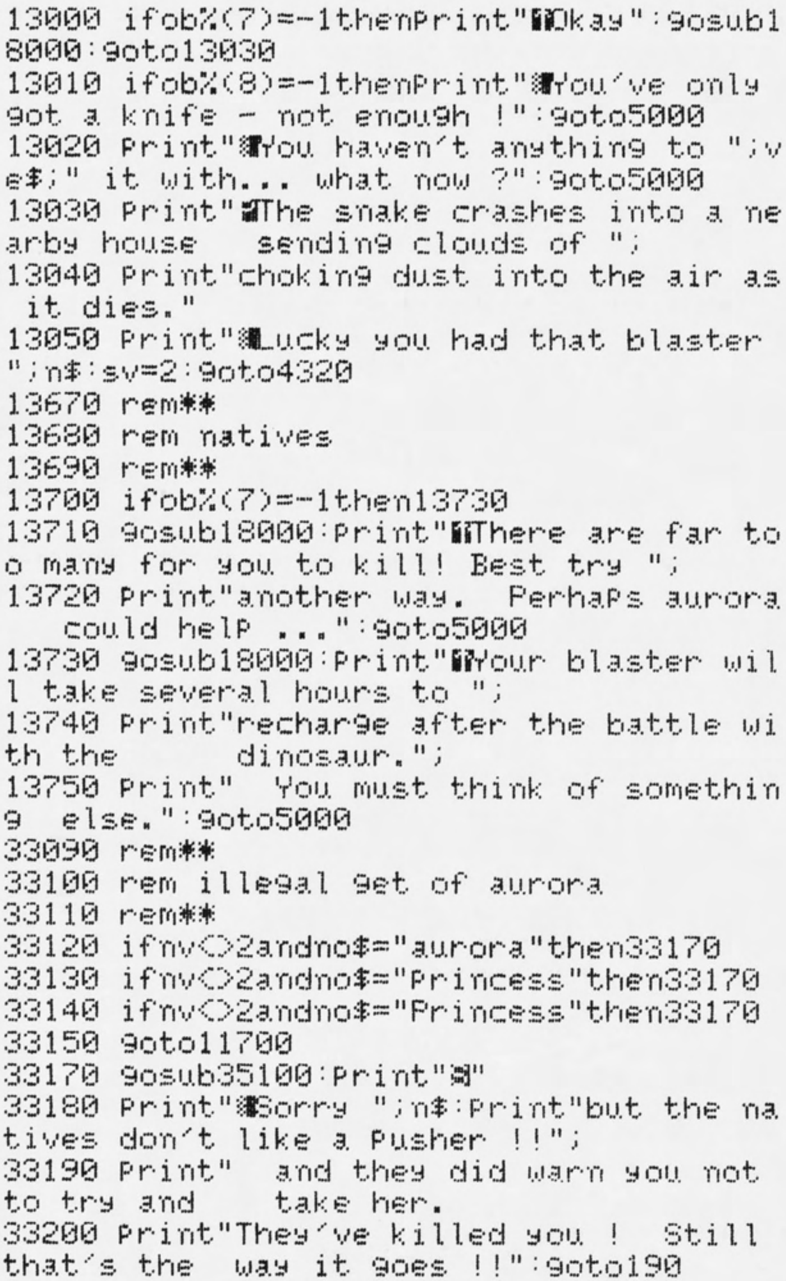
Lines 32500–32550: These lines are the descriptions of what you see from the top of the tree. I have tried to give little clues without letting too much information creep in, a good way of maintaining interest.

The next Module deals with two more perils — the snake in the city and the natives who 'have' Aurora. The snake is a pretty tame peril really, fairly easy to escape and unlikely to cause a true Adventurer any problem (except that a true Adventurer will be expecting some kind of trick so it's a kind of double-bluff). The natives are going to be a bit more difficult, and I will be describing more of the programming for this section later.

MODULE 13.4



```
1 goto3
2 save"@0:module 13.4",8:verify"module 1
3.4",8:stop
3 rem
7570 ifve$="get"andp=92then33120
7580 ifve$="take"andp=92then33120
7600 ifve$="grab"andp=92then33120
12970 rem**
12980 rem snake
12990 rem**
```



```
13000 ifob%(7)=-1thenprint"█Okay":gosub1
8000:goto13030
13010 ifob%(8)=-1thenprint"█You've only
got a knife - not enough !":goto5000
13020 print"█You haven't anything to ";v
e$;" it with... what now ?":goto5000
13030 print"█The snake crashes into a ne
arby house   sending clouds of ";
13040 print"choking dust into the air as
 it dies."
13050 print"█Lucky you had that blaster
";n$:sv=2:goto4320
13670 rem**
13680 rem natives
13690 rem**
13700 ifob%(7)=-1then13730
13710 gosub18000:print"█There are far to
o many for you to kill! Best try ";
13720 print"another way.  Perhaps aurora
   could help ...":goto5000
13730 gosub18000:print"█Your blaster wil
l take several hours to ";
13740 print"recharge after the battle wi
th the       dinosaur.";
13750 print"  You must think of somethin
g  else.":goto5000
33090 rem**
33100 rem illegal get of aurora
33110 rem**
33120 ifnv<>2andno$="aurora"then33170
33130 ifnv<>2andno$="princess"then33170
33140 ifnv<>2andno$="princess"then33170
33150 goto11700
33170 gosub35100:print"█"
33180 print"█Sorry ";n$:print"but the na
tives don't like a pusher !!";
33190 print"  and they did warn you not
to try and    take her.
33200 print"They've killed you !  Still
that's the  way it goes !!":goto190
```

Lines 7570–7600: Should you try and "GET" Aurora in this location then these lines refer you to the appropriate area in the program.

Line 13000: If you have the blaster (and *any* Adventurer worth his salt will be sure to have picked it up right at the beginning) then you get this response if you 'kill' or 'shoot' the snake.

Lines 13010–13020: If you don't happen to have the blaster you will drop to Line 13010 which checks to see if you have the knife. If you have then this response is shown. If you have neither then you'll go to Line 13020.

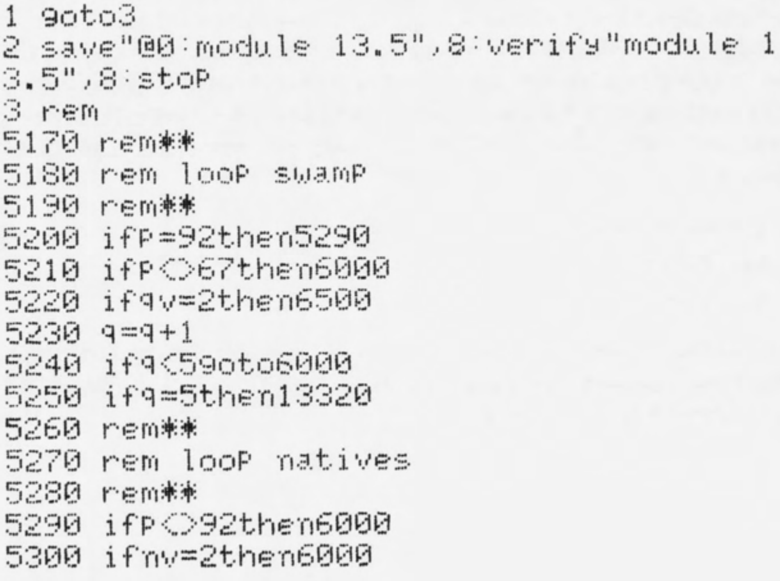
Lines 13030–13050: If you succeed in killing the snake the appropriate message appears and the variable SV is set to 2.

Lines 13700–13750: This section refers to an attempt to 'kill' the natives. First it checks to see if you have the blaster (Line 13700) and if you have then you get the bad news that it needs recharging (Lines 13730–13750). If you don't have the blaster then Lines 13710–13720 give you a little clue about the fact that Aurora might help.

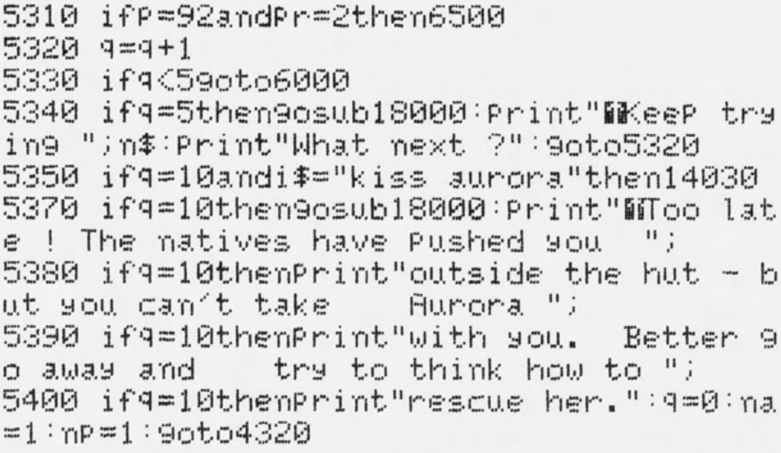
Lines 33120–33200: Suppose you tried to get Aurora in this Location (92) then you would have been directed to this part of the program. First it checks to see if you typed "aurora", "princess" or "Princess" (note the capital P just in case) and if not then Line 33150 will send you back to the "GET" routine.

If you did however Lines 33170–33200 send for Victor who tells you that the natives "..don't like a pusher .." and they've killed you.

MODULE 13.5



```
1 goto3
2 save"@0:module 13.5",8:verify"module 1
3.5",8:stop
3 rem
5170 rem**
5180 rem loop swamp
5190 rem**
5200 ifp=92then5290
5210 ifp<>67then6000
5220 ifqv=2then6500
5230 q=q+1
5240 ifq<5goto6000
5250 ifq=5then13320
5260 rem**
5270 rem loop natives
5280 rem**
5290 ifp<>92then6000
5300 ifnv=2then6000
```



```
5310 ifp=92andpr=2then6500
5320 q=q+1
5330 ifq<5goto6000
5340 ifq=5thengosub18000:print"█Keep try
ing ";n$:print"What next ?":goto5320
5350 ifq=10andi$="kiss aurora"then14030
5370 ifq=10thengosub18000:print"█Too lat
e ! The natives have pushed you  ";
5380 ifq=10thenprint"outside the hut - b
ut you can't take    Aurora ";
5390 ifq=10thenprint"with you.  Better g
o away and    try to think how to ";
5400 ifq=10thenprint"rescue her.":q=0:na
=1:np=1:goto4320
```

This module adds two timing loops for the quicksand and the natives. Following the same pattern as the eel loop (Module 12.6) they just count the number of inputs and at 5 (for the swamp) and 10 (for the natives) direct the program to the appropriate response.

In the case of the quicksand — you've had it. In the case of the natives you are allowed to leave the hut to go away and think again.

## Summary

The program is almost complete now. I hope that this chapter and the next is beginning to give you ideas which you can develop yourself to make your own Adventure interesting and a challenge to players. As I've already mentioned one aspect I was a little unhappy with was the fact that in order to impart an air of atmosphere — so that you really felt as if you *were* on the planet — I used up a lot of memory in the text descriptions and this left me with too little for embellishing the perils and adding more complicated riddles and problems to solve. I still think I made the right decision because I dislike the:

> you are in a cell
>
> keys bottle
>
> what now?

display common to so many Adventures. *Nightmare Planet* is enriched with science-fiction prose (adapted a little by the constraints of the 64) and I wouldn't have it any other way.

# CHAPTER 14
# A kiss in time

We have just a few loose ends to tidy up now and our story is complete. It may have seemed a long haul, but I've had fun and I hope you have too.

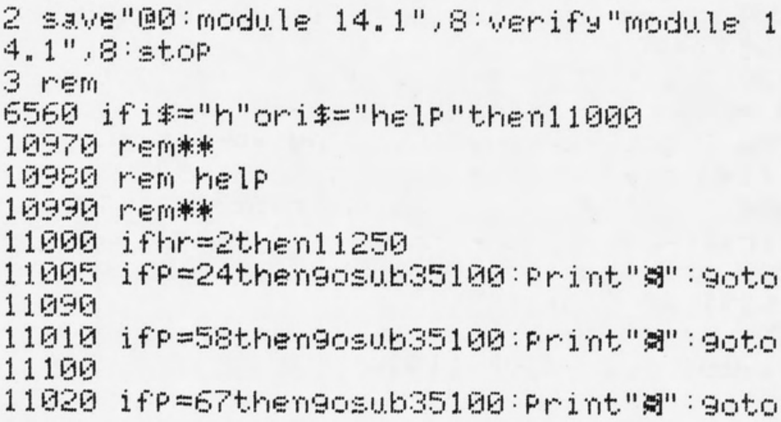
In this chapter we are concerned with some extra words that players might use — and as a grand finale the actual 'rescue' of the Princess.

## With a little help from my friends

HELP is probably the best-used and most annoying of all the words in Adventures. It can be the light that sparks you past the previously insurmountable problem and it can be the frustration that threatens the existence of your micro. There have been times I have wanted to beat the keyboard to pulp at some stupid response that helped me not one iota and there have been times I could have kissed it (metaphorically of course).

So when I came to *Nightmare Planet* I had to decide just how much help I was going to give to my potential player. I wanted to have Victor come down and introduce the "HELP" section and to have a few hints. Of course you can give misleading clues (very sneaky!) but I'm basically honest and decided to play it straight.

MODULE 14.1



```
1 goto3
2 save"@0:module 14.1",8:verify"module 1
4.1",8:stop
3 rem
6560 ifi$="h"ori$="help"then11000
10970 rem**
10980 rem help
10990 rem**
11000 ifhr=2then11250
11005 ifp=24thengosub35100:print"♥":goto
11090
11010 ifp=58thengosub35100:print"♥":goto
11100
11020 ifp=67thengosub35100:print"♥":goto
```

```
11120
11030 ifp=92then9osub35100:print"▨":9oto
11140
11040 ifp=1orp=2orp=11orp=12orp=22then9o
sub35100:print"▨":9oto11190
11050 ifp=42orp=62orp=63then9osub35100:p
rint"▨":9oto11170
11060 9osub35100:print"▨"
11070 print"▨I'm havin9 a little circuit
 trouble     myself ";n$
11080 print"Try to cope on your own for
a bit ...":9oto5000
11090 print"▨Try killin9 it ";n$:print"b
efore it kills you !!":9oto5000
11100 print"▨Now think hard ";n$:print"T
here must be somethin9 lyin9 ";
11110 print"around     that could help yo
u ...":9oto5000
11120 print"▨Sticky situation this ";n$:
print"I su99est throwin9 somethin9 ";
11130 print"to pull     yourself out.":9o
to5000
11140 print"▨▨▨As the natives think auro
ra is their  queen  ";
11150 print"it mi9ht help if you showed
them you're quite ";
11160 print"friendly with aurora.":9oto5
000
11170 print"▨Lost in the forest, eh ?  W
hy don't you climb a ";
11180 print"tree and take a look around.
":9oto5000
11190 print"▨We both appear to be lost "
;n$:print"Better just ";
11200 print"keep walkin9 and you're sure
to find the way eventually.":9oto4320
11250 print"▨":9osub35100:print"▨We're n
ot friends at the moment ";n$:9oto11310
11280 print"▨":9osub35100:print"▨You can
't kill me ";n$
11290 print"▨I'm the new invulnerable mo
del. But ... ":9oto11310
```

```
11300 print"▒":gosub35100:print"▒That wa
sn't very nice ";n$
11310 print"I shan't ▒helP▒ you anymore
now.";
11320 print"  You can coPe on your own."
:goto5000
```

Line 6560: The I$ command.

Line 11000: In Modules 10.4 and 12.1 you may remember Lines 7865 and 12305 catered for the possibility of the player either throwing or killing Victor. If this did happen it seemed logical that Victor would be a trifle 'miffed' and might not 'help' so readily, so to cater for this I set the variable HR to 2. This line directs the program past the help routines if HR does equal to 2.

Lines 11010–11050: These are all pointers that operate should you type HELP in any of the appropriate locations.

Lines 11060–11080: The standard response if you ask for help in any of the other locations.

Lines 11090–11200: These lines are the appropriate replies, again depending on the location.

Lines 11250–11320: If you have been nasty to Victor then this section deals with the response to killing or throwing him around. The little touch of Victor almost sulking because you'd abused him was suggested by Brendon Gore — and illustrates the importance of getting others to play your game before you use up all your memory. I knew as soon as Brendon made the suggestion it was genius — and was very glad I could incorporate it.

I left spaces in this section to allow me to add other responses at a later date, but sadly the shortage of memory prevented it in the end.

## Using and firing

MODULE 14.2

```
1 goto3
2 save"@0:module 14.2",8:verify"module 1
4.2",8:stop
3 rem
7660 ifve$="use"then11500
```

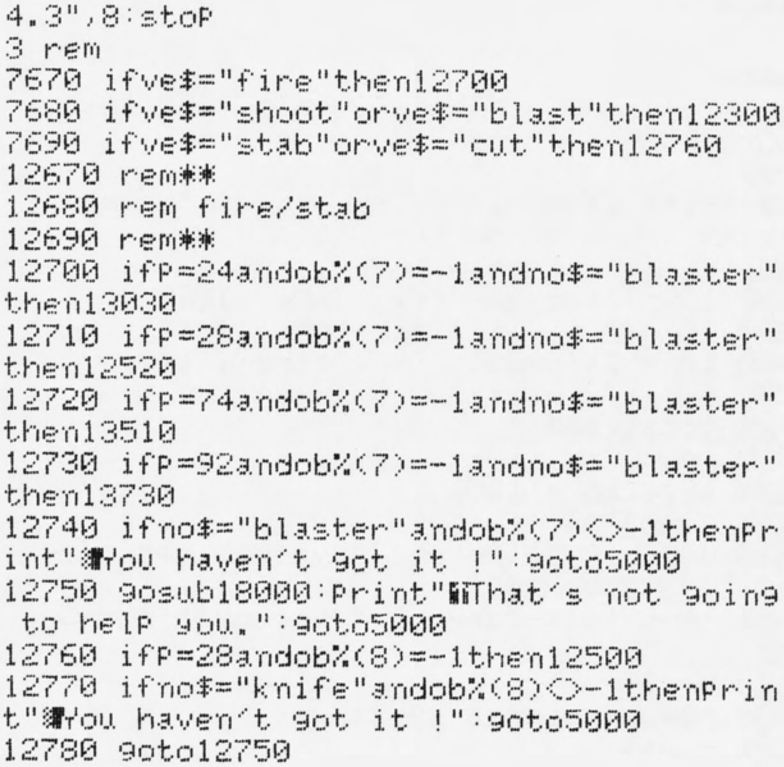
```
11470 rem**
11480 rem use
11490 rem**
11500 ifp<>24orp<>74andno$="blaster"then
11580
11510 ifp<>24orp<>74goto11610
11520 ifp=24orp=74andno$="blaster"then11
580
11530 ifp=24orp=74andno$="knife"then1155
0
11540 goto11660
11550 ifob%(8)<>-1thengosub19000:print"▓
You haven't got it !!":goto5000
11560 ifob%(8)=-1thengosub35100:print"▓"
:print"▓Just a knife ";n$;" ?"
11570 ifob%(8)=-1thenprint"▓That's a lit
tle ambitious isn't it ?":goto5000
11580 ifob%(7)<>-1thengosub19000:print"▓
You haven't got it !!":goto5000
11590 ifp=24andob%(7)=-1thengosub18000:p
rint"▓Good thinking !":goto13030
11600 ifp=74andob%(7)=-1thengosub35100:p
rint"▓":print"▓Good thinking!":goto13510
11610 fori=1to9
11620 ifob$(i)=no$then11630
11630 ifob%(i)=-1then11660
11640 nexti
11650 gosub19000:print"▓There's no point
 in using that !":goto5000
11660 print"▓How do you want to use the
";no$:gosub18000:goto5000
```

This module is a standard module following the pattern of others so far, and dealing with the command USE. I have concentrated on using the blaster or the knife, as these would seem to be the logical objects.

Note the last two lines in the module which allow for the object not being recognised as a valid one (Line 11650) and then being recognised (Line 11660).

MODULE 14.3

```
1 goto3
2 save"@0:module 14.3",8:verify"module 1
```



```
4.3",8:stop
3 rem
7670 ifve$="fire"then12700
7680 ifve$="shoot"orve$="blast"then12300
7690 ifve$="stab"orve$="cut"then12760
12670 rem**
12680 rem fire/stab
12690 rem**
12700 ifp=24andob%(7)=-1andno$="blaster"
then13030
12710 ifp=28andob%(7)=-1andno$="blaster"
then12520
12720 ifp=74andob%(7)=-1andno$="blaster"
then13510
12730 ifp=92andob%(7)=-1andno$="blaster"
then13730
12740 ifno$="blaster"andob%(7)<>-1thenpr
int"█You haven´t got it !":goto5000
12750 gosub18000:print"█That´s not going
 to help you.":goto5000
12760 ifp=28andob%(8)=-1then12500
12770 ifno$="knife"andob%(8)<>-1thenprin
t"█You haven´t got it !":goto5000
12780 goto12750
```

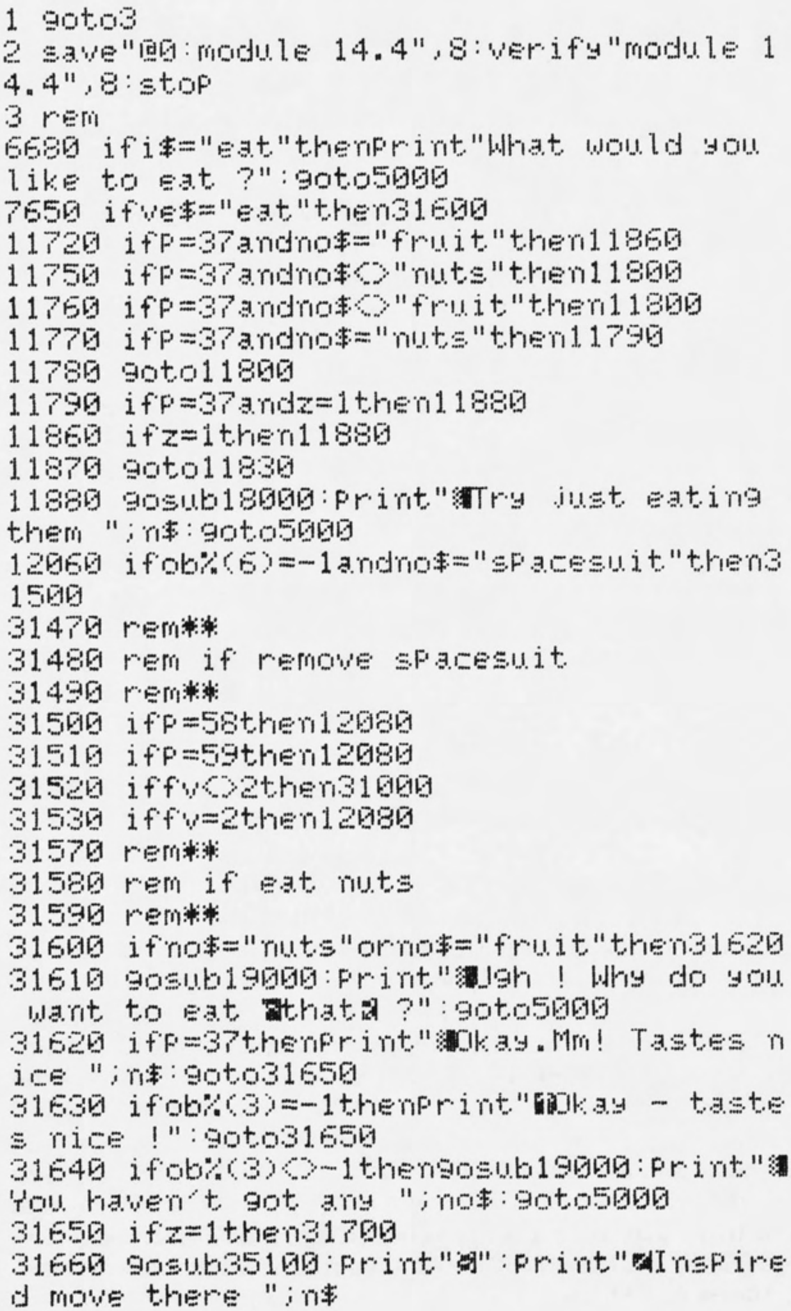
Another standard module which duplicates the KILL module but allows you to SHOOT, FIRE, BLAST, STAB or CUT and then directs the program on.

## Fruit and nut case

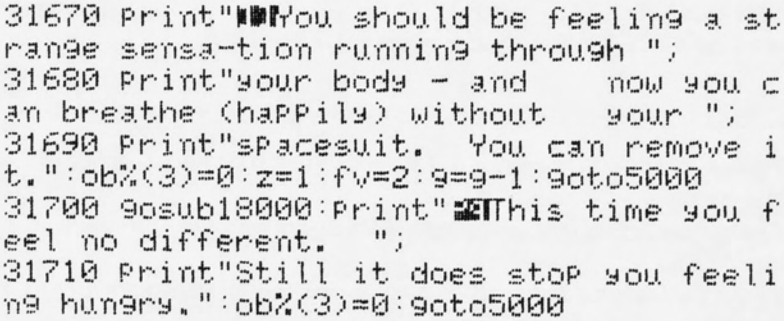
The fruit and nuts on the bush sitting innocently in the desert do have a special significance. Once eaten it's possible to remove your spacesuit without expiring in the poisonous atmosphere (suspension of belief required here) and if you don't remove that spacesuit you can't kiss Aurora and escape.

Programming this module into the 64 was another tricky bit of brainpower and late night work! I had to intersperse lines into the GET module quite considerably as I needed to be able to cope with situations where people would try and "GET" the nuts or fruit in Location 37, or just "EAT" them in that location or in another location. As well as that though, I also had to prepare for the player returning to Location 37 a second or third time and trying to GET or EAT the nuts and fruit a second, third or fourth time. The result is Module 14.4.

MODULE 14.4



```
1 goto3
2 save"@0:module 14.4",8:verify"module 1
4.4",8:stop
3 rem
6680 ifi$="eat"thenprint"What would you
like to eat ?":goto5000
7650 ifve$="eat"then31600
11720 ifp=37andno$="fruit"then11860
11750 ifp=37andno$<>"nuts"then11800
11760 ifp=37andno$<>"fruit"then11800
11770 ifp=37andno$="nuts"then11790
11780 goto11800
11790 ifp=37andz=1then11880
11860 ifz=1then11880
11870 goto11830
11880 gosub18000:print"■Try just eating
them ";n$:goto5000
12060 ifob%(6)=-1andno$="spacesuit"then3
1500
31470 rem**
31480 rem if remove spacesuit
31490 rem**
31500 ifp=58then12080
31510 ifp=59then12080
31520 iffv<>2then31000
31530 iffv=2then12080
31570 rem**
31580 rem if eat nuts
31590 rem**
31600 ifno$="nuts"orno$="fruit"then31620
31610 gosub19000:print"■Ugh ! Why do you
 want to eat ■that■ ?":goto5000
31620 ifp=37thenprint"■Okay.Mm! Tastes n
ice ";n$:goto31650
31630 ifob%(3)=-1thenprint"■Okay - taste
s nice !":goto31650
31640 ifob%(3)<>-1thengosub19000:print"■
You haven't got any ";no$:goto5000
31650 ifz=1then31700
31660 gosub35100:print"■":print"■Inspire
d move there ";n$
```



```
31670 print"▮▮You should be feeling a st
range sensa-tion running through ";
31680 print"your body - and    now you c
an breathe (happily) without   your ";
31690 print"spacesuit.  You can remove i
t.":ob%(3)=0:z=1:fv=2:g=g-1:goto5000
31700 gosub18000:print"▦▮This time you f
eel no different.  ";
31710 print"Still it does stop you feeli
ng hungry.":ob%(3)=0:goto5000
```

Lines 6680 and 7650: The I$ and VE$ commands.

Lines 11720–11790: These lines interleave with the standard lines in the "GET" module (Module 10.3). Basically they allow you to type GET FRUIT or GET NUTS both in Location 37 and in any other location. The variable Z is set at 0 until you have eaten the nuts for the first time — then it is set to 1. This varies the reply (as you will see in a moment).

Lines 11860–11880: Still in the "GET" module Line 11860 starts by checking the value of Z. If Z = 1 then the response in Line 11880 is JUST TRY EATING THEM. The reason for this is for the situation where the player tries to GET the nuts a second time, after eating them once already. If Z = 0 then on the command GET FRUIT the program loops back here to Line 11830 which is the "OKAY" part of the "GET" module.

Line 12060: This line comes from the "DROP" module and refers to the situation if you decide to 'drop' or 'remove' your spacesuit before you've eaten the nuts. It directs you to Line 31500.

These next two sections of Module 14.4 are to cope with the fact that if you should try and remove your spacesuit too soon — you will die in the poisonous atmosphere... unless you have eaten the nuts.

Lines 31500–31530: Obviously you can remove the spacesuit in your spaceship — so Lines 31500 and 31510 cope with that. The variable FV controls eating the fruit or nuts (a value of 0 means you haven't eaten yet, whereas 2 means you have). Thus Line 31520 will direct you to the "YOU WILL DIE" area and Line 31530 will move to the "OKAY" response in the "DROP" module.

Lines 31600–31610: In response to the command EAT Line 31600 checks to see if NO$ is "NUTS" or "FRUIT". If not then you get the standard reply

"UGH! WHY DO YOU WANT TO EAT THAT?", in case the player types "EAT NATIVES" or some other daft command.

Lines 31620–31710: This section copes with the following situations:

1. You eat the nuts without "GETTING" them first (in Location 37) — Line 31620.
2. You eat the nuts after getting them (which can occur in any location) — Line 31630.
3. You try to eat the nuts out of Location 37 when they aren't in your inventory — Line 31640.
4. You eat the nuts a second, third, etc. time — Line 31650.

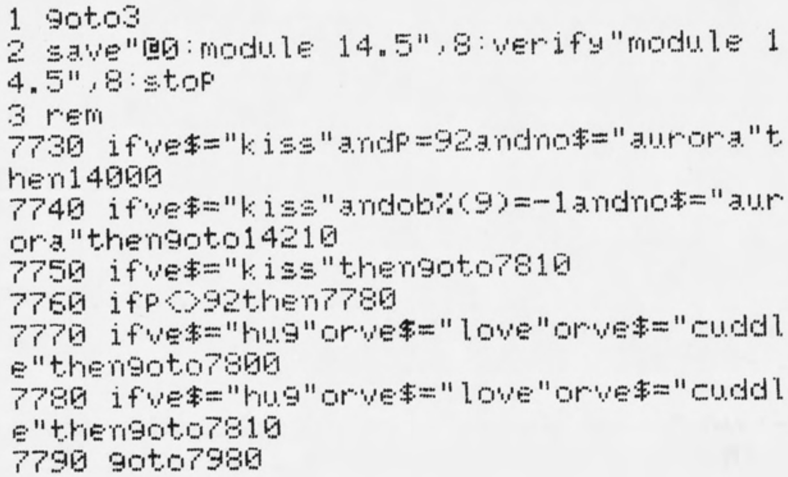
Note that Line 31690 contains several variables which do the following:

1. Remove the nuts from the inventory.
2. Set Z to 1.
3. Set FV to 2 to allow safe removal of spacesuit.
4. Subtract 1 from G (inventory variable).

## Terrific man!

This must be the highlight of the game — you've found Aurora and you rescue her. I wanted something a little bit different to mark the moment of triumph, and decided to extend my programming for the actual kiss to include sound and visuals which should sound and look a little bit extra. This final module is that moment. . .

MODULE 14.5



```
1 goto3
2 save"@0:module 14.5",8:verify"module 1
4.5",8:stop
3 rem
7730 ifve$="kiss"andp=92andno$="aurora"t
hen14000
7740 ifve$="kiss"andob%(9)=-1andno$="aur
ora"thengoto14210
7750 ifve$="kiss"thengoto7810
7760 ifp<>92then7780
7770 ifve$="hug"orve$="love"orve$="cuddl
e"thengoto7800
7780 ifve$="hug"orve$="love"orve$="cuddl
e"thengoto7810
7790 goto7980
```

```
7800 gosub18000:print"▓That's not friend
ly enough.Try somethingelse.":goto5000
7810 gosub19000:print"▒You're acting a b
it strange ";n$:goto5000
13970 rem**
13980 rem kiss princess
13990 rem**
14000 ifob%(6)=-1thengosub19000:print"▓Y
ou can't kiss her wearing a spacesuit";
14010 ifob%(6)=-1thenprint"  can you ? I
'd have thought you'd know   that !"
14020 ifob%(6)=-1thenprint"She's pointin
g at her mouth ...":goto5000
14030 forl=0to24:pokesc+l,0:nextl
14040 pokel3,3:pokew3,16:pokesc+3,1:poke
vo,143:pokes1,240:pokesc+4,65
14050 fr=5389:fort=1to200:fq=fr+peek(sc+
27)*225
14060 hf=int(fq/256):lf=fq-hf*256
14070 pokel1,lf:pokeh1,hf
14080 nextt
14090 printchr$(13)
14100 fort=1to5
14110 print"▓▒▓TERRIFIC▓ ▓MAN▓ ▓!!!▓"
14120 form=1to500:nextm
14130 print"▓                "
14140 form=1to500:nextm
14150 nextt
14160 pokevo,0
14170 print"▓▓▓That not only did you a p
ower of good but the natives ";
14180 print"are smiling now. You canesca
pe with aurora.":nv=2:pr=2
14190 print"▒▓▓▓Now you must return to t
he ship and   complete your ";
14200 print"mission. Don't forget to  ta
ke aurora !":goto4320
14210 gosub18000:print"▒Hm ! Nice. But y
ou should keep ";
14220 print"your mindon the game ";n$:go
to4320
```

Lines 7730–7750: In response to the command KISS these three lines cope with all possible eventualities. The first is for kissing Aurora inside the hut, the second for kissing her anywhere else in the game, and the third for kissing anyone or anything else.

Lines 7760–7790: Lines 7760 and 7790 are to allow the program to bypass these areas if not applicable. Lines 7770 and 7780 cover the responses "HUG", "LOVE" or "CUDDLE" instead of "KISS".

Lines 7800–7810: If you try to hug Aurora then the response is "NOT FRIENDLY ENOUGH — TRY SOMETHING ELSE". If you try and KISS anyone or anything else then Line 7810 comes back with "YOU'RE ACTING A BIT STRANGE". This line seems to cause quite a bit of amusement from the people I've watched playing.

Lines 14000–14020: These lines gives a response should you try and kiss Aurora with the spacesuit on. There is a subtle hint in Line 14020 — "She's pointing at her mouth" — which could mean many things but is meant to refer to the 'eating nuts' command.

Lines 14030–14220: This is it! The kiss! First we have a sound effect similar to a scale rising and falling, then the words "TERRIFIC MAN!!!" flash on and off five times on the screen. Line 14110 contains the actual "TERRIFIC MAN" but it has been printed as graphic symbols — just press shift and type away.

Finally a message about the natives being friendly comes up and you're allowed to leave with Aurora.

Lines 14210–14220 are just a little extra — should you decide to kiss Aurora again anytime.

*Testing Modules 14.1–14.5*

You don't really need to test these modules separately, because you have finished the game. All you need to do is quickly find your way down to the hut, save to tape, then try hugging, cuddling, kissing etc both inside the hut and outside. The appropriate replies should occur.

## Summary

This final chapter introduces you to the concept of just stretching yourself every once in a while to produce that little bit extra — in this case the kiss. I found that the time to add the music and visuals is nothing like as extensive as time in debugging and checking. And it does give your program that polish.

The marathon is over, the game is complete. Relax and play it awhile, deciding if you want to alter any lines and add your own personal touch before finally trying it on your friends. Above all, have fun.

# CHAPTER 15
# Looking to the future

## Extending Nightmare Planet

Now that you've finished the book you needn't stop here. You can try extending the program. You can add locations and take away locations. You can make it possible to play as a man or a woman (try adding a few more variables such as changing ''aurora'' for P$) then have the INPUT ''Are you male or female'' right at the start and change variables accordingly. There are many different possibilities open to you.

Several little ideas occur to me as I write this. Perhaps your hero could have oxygen cylinders which are in danger of running out of air — unless he can find some more somewhere in the Adventure. What about a clue in the pool in that clearing in the forest? Perhaps the gem might be of use after all. . . in rescuing Aurora! I'm sure you can think of more.

I've already mentioned the problem of memory — but if you cut out all the REM statements you should find you've quite a few extra bytes. Why not create the title sequence and then reload the program after the titles finish? Why not have a whole section of the Adventure underground (passages leading to an underground city which you reach by going ''down'' in the ruined temple) which loads in after you go through a certain section of the program. The possibilities are limited only by your own mind and imagination.

The 64 can give you graphics, sprites and sound. Use them.

## Over to you

Only you can sit down and create your own plot, write your own story, program your own Adventure. Don't be put off by the length of my program — your first can be half the length, one tenth the length. What matters is that you give it a try. There's a tremendous feeling of achievement when you actually start to move around your own locations, and start to pick up and drop things. I know you can remember that moment when the first truly original program you wrote actually worked. It's like hearing a record for the first time that really hits you — and whenever you hear it again you can remember your first time, you can almost feel as though you're back there in time, smelling the air, seeing the place. It's like the first time you rode a bike, only better because your program was just yours — no-one else had ever written exactly the same thing.

It's crazy but it's true! Your first really original Adventure will give you more satisfaction (much, much more) than finally completing *Nightmare Planet*. And don't get worried about programming skills — my own abilities were pretty negligible before I started to write this book — but it's not that hard really. If you write a really good one send it along to me at Sunshine Books — I'm still basically a player with a taste for Adventure.

# Afterword

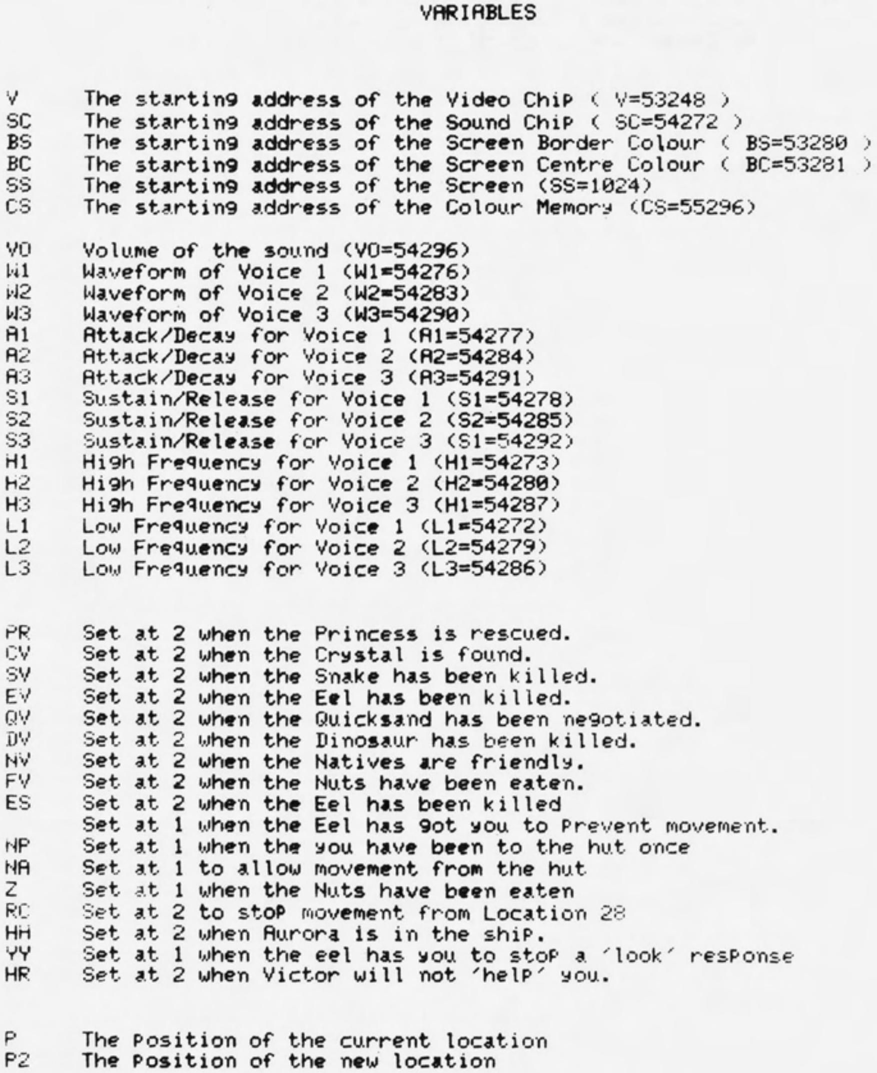
This has been an invigorating book to write — and quite an experience. I have tried checking, double-checking, triple-checking and then checking again to eliminate errors in both the programs and the text. Inevitably I will have missed something somewhere.

My grateful thanks go to everyone who has helped me in playing the game and writing the program. When I started I wasn't sure if I could do it — but I have. You have just seen how I, personally learnt how to write an Adventure and improve my own programming abilities along the way.

I still find Adventure my favourite computer game. It seems a whole bunch of other people do too. Enjoy yourself creating your own.

# Appendices

# APPENDIX A
# Table of Variables for Nightmare Planet



```
                            VARIABLES


V       The starting address of the Video Chip ( V=53248 )
SC      The starting address of the Sound Chip ( SC=54272 )
BS      The starting address of the Screen Border Colour ( BS=53280 )
BC      The starting address of the Screen Centre Colour ( BC=53281 )
SS      The starting address of the Screen (SS=1024)
CS      The starting address of the Colour Memory (CS=55296)

VO      Volume of the sound (VO=54296)
W1      Waveform of Voice 1 (W1=54276)
W2      Waveform of Voice 2 (W2=54283)
W3      Waveform of Voice 3 (W3=54290)
A1      Attack/Decay for Voice 1 (A1=54277)
A2      Attack/Decay for Voice 2 (A2=54284)
A3      Attack/Decay for Voice 3 (A3=54291)
S1      Sustain/Release for Voice 1 (S1=54278)
S2      Sustain/Release for Voice 2 (S2=54285)
S3      Sustain/Release for Voice 3 (S1=54292)
H1      High Frequency for Voice 1 (H1=54273)
H2      High Frequency for Voice 2 (H2=54280)
H3      High Frequency for Voice 3 (H1=54287)
L1      Low Frequency for Voice 1 (L1=54272)
L2      Low Frequency for Voice 2 (L2=54279)
L3      Low Frequency for Voice 3 (L3=54286)


PR      Set at 2 when the Princess is rescued.
CV      Set at 2 when the Crystal is found.
SV      Set at 2 when the Snake has been killed.
EV      Set at 2 when the Eel has been killed.
QV      Set at 2 when the Quicksand has been negotiated.
DV      Set at 2 when the Dinosaur has been killed.
NV      Set at 2 when the Natives are friendly.
FV      Set at 2 when the Nuts have been eaten.
ES      Set at 2 when the Eel has been killed
        Set at 1 when the Eel has got you to prevent movement.
NP      Set at 1 when the you have been to the hut once
NA      Set at 1 to allow movement from the hut
Z       Set at 1 when the Nuts have been eaten
RC      Set at 2 to stop movement from Location 28
HH      Set at 2 when Aurora is in the ship.
YY      Set at 1 when the eel has you to stop a 'look' response
HR      Set at 2 when Victor will not 'help' you.


P       The position of the current location
P2      The position of the new location
```

```
OB%    Integer value of Object in Object Array.
OB$    Single word description of Object.
SI$    Long description of Object for Location.
A$     Local variable to wait for Input.

N$     Name of Player.
I$     Command expected from Player.
VE$    Verb expected in a 2 word command from Player.
NO$    Noun expected in a 2 word command from Player.

PI     Local variable used to split I$
SP     Local variable used to split I$
M      Local variable used to create a timing loop for Pausing.
G      Local variable used to add objects in Inventory.
I      Variable used to count objects in the Object Array.
```

# APPENDIX B
# The Line Numbers

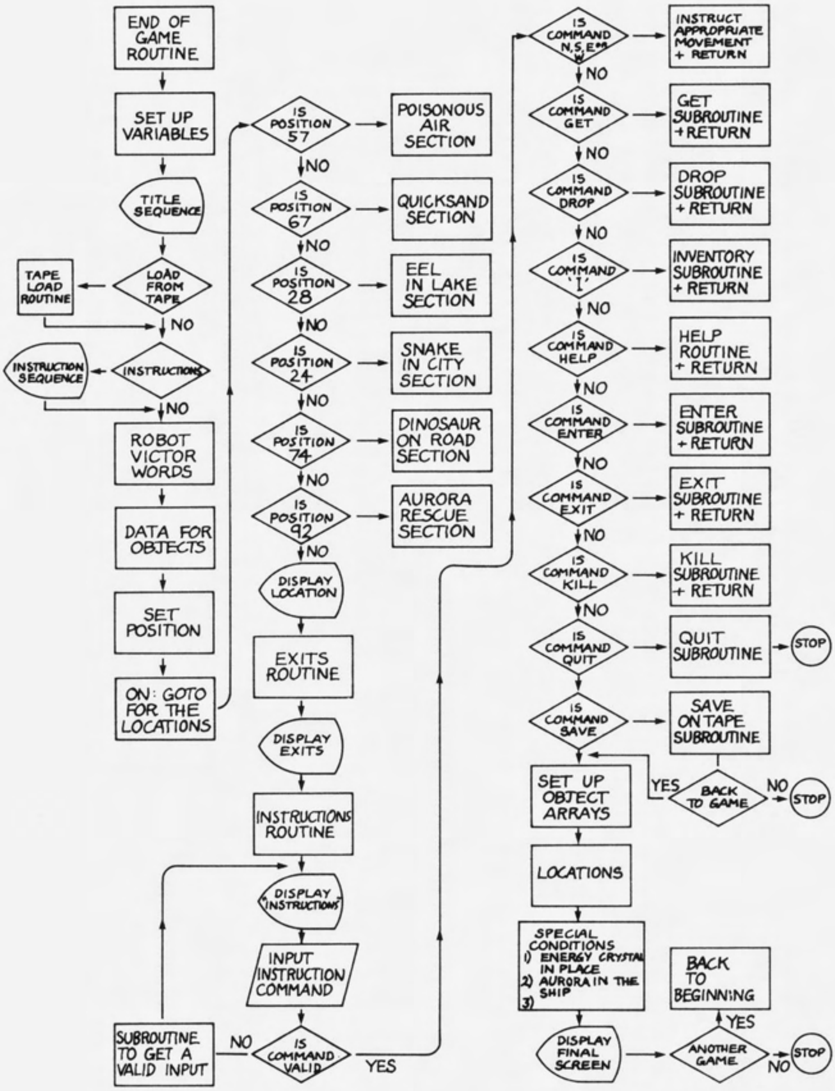
| | | | |
|---|---|---|---|
| PRE-CREDITS | 130 | OBJECTS DATA | 3700 |
| YOU'RE DEAD | 190 | POSITION AND LOCATIONS | 4000 |
| VARIABLES | 280 | OBJECT LOCATION | 4150 |
| SOUND EFFECT | 360 | PERIL POINTERS | 4200 |
| TITLE PAGE | 1000 | EXITS | 4320 |
| TITLE SCRIPT | 1300 | "INPUT INSTRUCTION" | 5000 |
| SPACESHIP NOISE | 2030 | LOOP EEL | 5070 |
| 'S' TO START | 2100 | LOOP QUICKSAND | 5200 |
| SPACESHIP CRASH | 2170 | LOOP NATIVES | 5290 |
| VICTOR SCRIPT | 2400 | NO EXITS | 6000 |
| | | PREVENT EXITS | 6110 |
| | | | |

| | |
|---|---|
| LOCATIONS | 20000 |
| SPECIAL LOCATIONS | 21960 |
| SNAKE DESCRIPTIONS | 30000 |
| EEL DESCRIPTION | 30500 |
| POISONOUS AIR DESCRIPTION | 31000 |
| IF REMOVE SPACESUIT | 31500 |
| IF EAT NUTS | 31600 |
| CRYSTAL DESCRIPTION | 32000 |
| CANNOT GET CRYSTAL | 32070 |
| CLIMB TREE | 32500 |
| QUICKSAND DESCRIPTION | 32600 |
| DINOSAUR DESCRIPTION | 32700 |
| NATIVES DESCRIPTION | 33000 |
| BACK TO HUT AGAIN | 33060 |
| ILLEGAL GET OF AURORA | 33120 |
| FOUND AURORA | 33240 |
| FOUND CRYSTAL | 33380 |
| NO CRYSTAL | 33440 |
| MEALY BUG SPRITE | 35000 |
| VICTOR SPRITE | 35100 |
| MEALY BUG DATA | 35240 |
| VICTOR DATA | 35330 |
| SET UP SPRITES | 35420 |
| | |

| | | | |
|---|---|---|---|
| I$ COMMANDS | 6500 | VERB COMMANDS | 7500 |
| BYPASS DIRECTION | 7030 | | |
| AIRLOCK | 7070 | | |
| | | | |
| | | | |
| | | | |
| MOVEMENT | 7200 | INVENTORY | 10000 |
| CHECK I$ FOR SPACE | 7300 | HELP | 11000 |
| | | USE | 11500 |
| I$ TO VE$ + NO$ | 7420 | GET | 11700 |
| | | DROP | 12000 |
| | | | |

| | | | |
|---|---|---|---|
| | | | |
| KILL | 12300 | | |
| | | | |
| EEC PERIL | 12500 | GENERAL INSTRUCTIONS | 15000 |
| FIRE + STAB | 12700 | LOAD | 16000 |
| | | SAVE | 17000 |
| SNAKE PERIL | 13000 | QUIT | 17500 |
| | | PIP | 18000 |
| QUICKSAND PERIL | 13300 | BURP | 19000 |
| DINOSAUR PERIL | 13500 | | |
| NATIVES PERIL | 13700 | | |
| KISS PRINCESS | 14000 | | |

# APPENDIX C
# Overall Flow Chart for Nightmare Planet



Other titles from Sunshine

**THE WORKING SPECTRUM**
David Lawrence
0 946408 00 9 £5.95

**THE WORKING DRAGON 32**
David Lawrence
0 946408 01 7 £5.95

**THE WORKING COMMODORE 64**
David Lawrence
0 946408 02 5 £5.95

**DRAGON 32 GAMES MASTER**
Keith Brain/Steven Brain
0 946408 03 03 £5.95

**FUNCTIONAL FORTH for the BBC Computer**
Boris Allan
0 946408 04 1 £5.95

**COMMODORE 64 machine code master**
David Lawrence and Mark England
0 946408 05 X £6.95

**Advanced Sound and Graphics for the Dragon computer**
Keith and Steven Brain
0 946408 06 8 £5.95

**Spectrum Adventures**
Tony Bridge and Roy Carnell
0 946408 07 6 £5.95

## 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 35p 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 £8.00 in the UK and £14.00 overseas.

For further information contact:
Sunshine
12–13 Little Newport Street
London WC2R 3LD
01-734- 3454

Printed in England by Commercial Colour Press, London E.7.

**Commodore 64 Adventures is a blueprint for the construction and playing of Adventure programs based on a full text Adventure. The emphasis is on the creation, design and planning of Adventure games as well as the programming techniques.**

**Diagrams, maps and charts simplify the process of adapting your ideas for a story into a format for programming. Understanding and designing maps will allow you to get more enjoyment from playing as well as programming. The charts for subroutines will have countless other applications.**

**A modular design for the program makes understanding the structure easier. All the modules are explained in detail, line by line, allowing you to understand the basic concepts used. Skills in programming will emerge as you gradually construct your Adventure — allowing you the flexibility of altering the demonstration Adventure or branching out to create your own.**

**The book is divided into two sections:**
**1) A simplified Adventure framework that will work for any story and helps you understand how to create your own.**
**2) A complete Adventure, Nightmare Planet, to enter and play with notes and anecdotes on the problems and pitfalls of Adventure programming and how to overcome them.**

**Delight and amuse your friends with your own ability to become a dream-maker — an Adventurer.**

The author, Mike Grace, has been writing since childhood on a variety of topics and has been captivated by microcomputers for some time. His lifetime fascination with fantasy and science-fiction led him to Adventures, allowing him to combine two of his passions. He is a regular contributor to Popular Computing Weekly.

SUNSHINE

ISBN 0 946408 11 4

£5.95 net