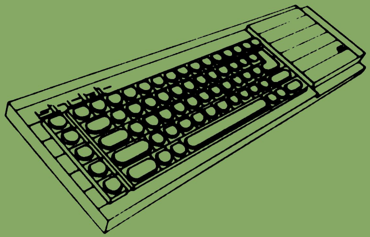
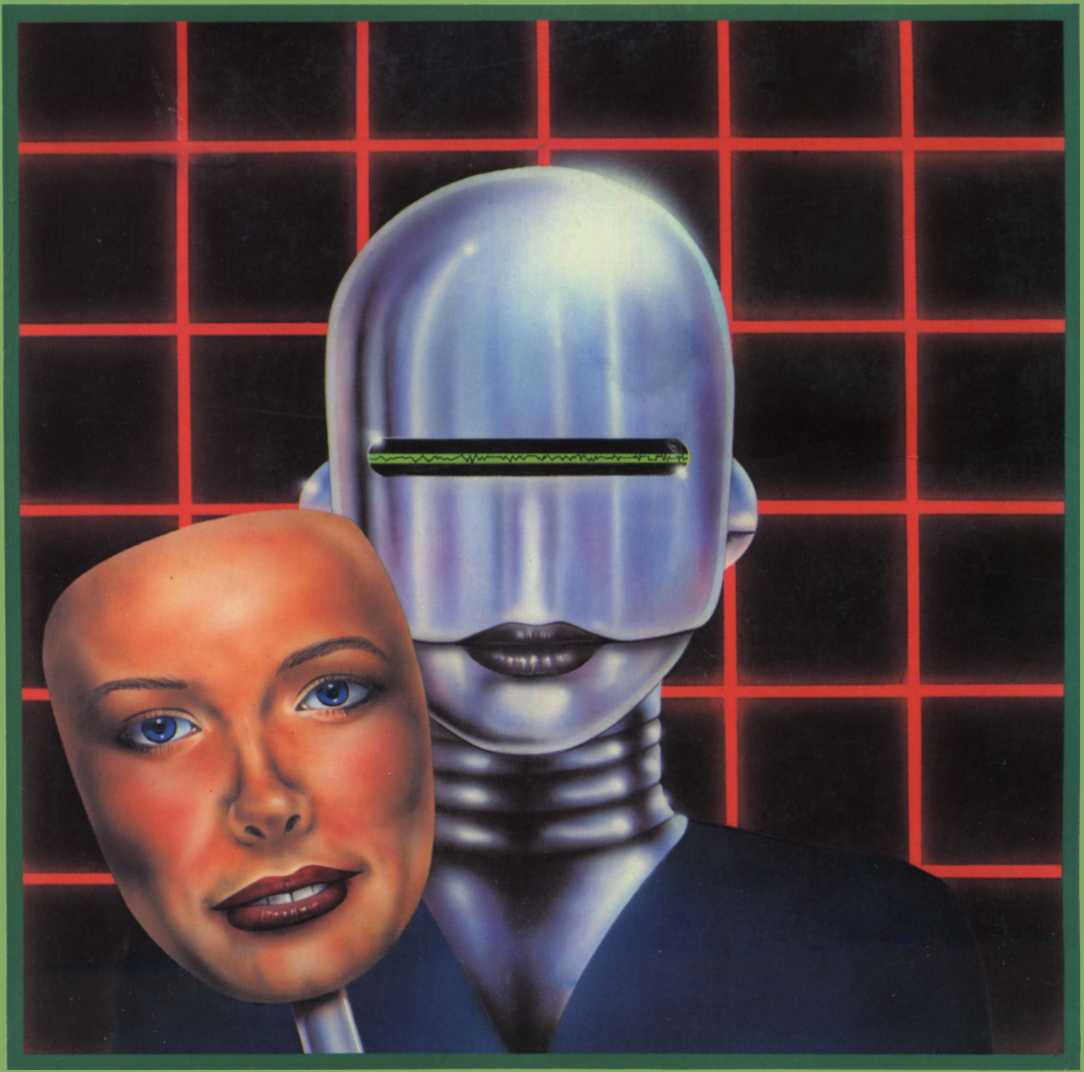
Introduction to Simulation Techniques on the Sinclair QL

Practical methods

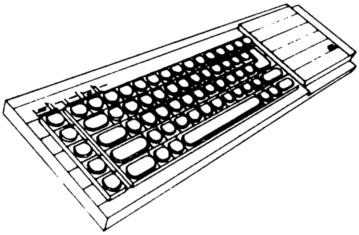


John Cochrane



Introduction to Simulation Techniques on the Sinclair QL

Practical methods



John Cochrane

First published 1984 by:
Sunshine Books (an imprint of Scot Press Ltd.)
12–13 Little Newport Street
London WC2R 3LD

British Library Cataloguing in Publication Data
Cochrane, John
  Introduction to simulation techniques on the Sinclair QL.
  1. Computer simulation
  2. Sinclair QL (Computer) — Programming
  I. Title
  001.4'24 QA76.9.C65

  ISBN 0-946408-45-9

Cover design by Grad Graphic Design Ltd.
Illustration by Richard Dunn.
Typeset and printed in England by Commercial Colour Press, London E7.

CONTENTS

Contents in detail

CHAPTER 1

Made to Measure

Introduction — who the book is for — what the book is about — who can use simulations — who develops simulations — which techniques can be used on micros — what is the scope of this book — organisation of the book.

CHAPTER 2

When I was a Boy....

A background to simulations — what are simulations — systems thinking — the user — early hardware and simulations — operational analysis — industrial simulations.

CHAPTER 3

But What Does This Little Black Button Do?

How simulations are used — the basic types of simulation — emulation simulations — analytical simulations — predictors — use and abuse of simulations.

CHAPTER 4

Getting It Together

Practical techniques — The importance of defining the requirement — diagrams — feedback — simulation structure — choosing the basic techniques — Monte Carlo — deterministic — Markov chain — technique to suit the simulation — development of the simulation — verifying the model.

CHAPTER 5

QLued Up and Ready To Go

What the QL can do — basic description — 68008 — other hardware — software — simulations on microcomputers — QL benefits.

CHAPTER 6

Laughing All the Way to the Bank

Money models — background — example — simulation development — results.

CHAPTER 7

Testing Testing Testing

Product development — background — example — simulation development — results — program description — program listing.

CHAPTER 8

What We Want is Information

What do you want to know? — background — example — simulation development — results — program description and use — program improvements — program listing.

CHAPTER 9

And Out of the Unknown....

Simulating the unknown — background — example — simulation development — results — program description — program improvements — program listing.

CHAPTER 10

Campaign For Real-time

Real-time simulation — background — example — simulation development and results — program description — program use and further development — program listing.

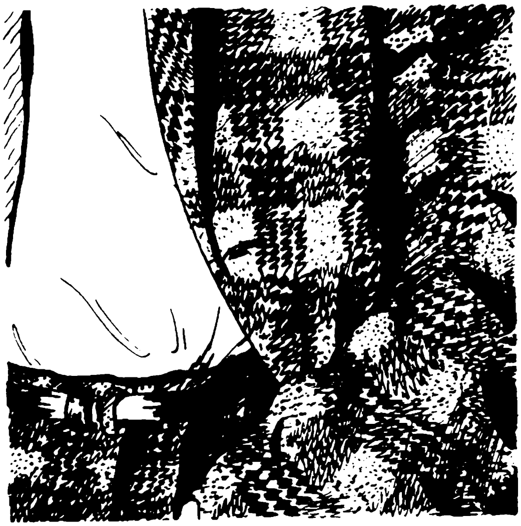
Foreword

```
100 REMark FORWARD
110 DIM P(11):MODE 256:CSIZE 2,0
120 INK 6:PAPER 0:CLS
130 P(0)=83:P(1)=73:P(2)=77:P(3)=85:P
(4)=76
            S          I          M       U
  L         A          T          E       -
140 P(5)=65:P(6)=84:P(7)=69:P(8)=45
            I          T                 !
150 P(9)=73:P(10)=84:P(11)=33
160 C=1:D=1:A=4:B=4:E=0
170 REPeat LOOP
180  PROA
190  SELect B=0 TO 5:D=1:C=RND(4)-2:F
=1
200  SELect B=15 TO 20:D=-1:C=RND(4)-
2:F=1
210  SELect A=0 TO 5:C=1:D=RND(4)-2:F
=1
220  SELect A=31 TO 37:C=-1:D=RND(4)-
2:F=1
230  SELect B=9:PROB
240  SELect E=0:PROD Q,R,I,G,F
250  PROD A,B,H,G,F
260 END REPeat LOOP
270 DEFine PROCedure PROA
280  F=0:Q=A:A=A+C
290  R=B:B=B+D
300  H=111:G=6:I=32
310 END DEFine
320 DEFine PROCedure PROB
330  SELect A=0 TO 11,24 TO 37:RETurn
340  J=(A-12):H=P(J):F=1:I=32:G=3
350 END DEFine

360 DEFine PROCedure PROD (K,L,M,N,O)
370  CURSOR 12*K,10*L
380  E=0:SELect M=33 TO 100:E=1
390  INK N:PRINT CHR$(M)
400  IF O=1 THEN BEEP 2,K,L,2,2
410 END DEFine
```

CHAPTER 1

Made to Measure

Introduction



...."tailored to meet a particular need"...

Who the book is for

This book is intended to give the inexperienced computer user a theoretical and practical introduction to the development and use of computer simulations. In particular, simulation techniques appropriate for use on the unexpanded and expanded (with 0.5 megabyte memory expansion) Sinclair QL microcomputer. The reader is expected to have a basic understanding of the principles of computing and, preferably, to have read the Sinclair QL manuals. Practical examples are given, using Sinclair SuperBASIC and the QL Application Software.

What the book is about

So, what are computer simulations? What are they used for? What are the benefits?

None of these questions have simple answers, although Chapters 3 and 4 go some way towards answering them. For now, we can think of computer simulations as special-purpose programs designed to duplicate the workings of the real world and to provide us with output in a very particular form. Every simulation must be developed to fulfil the specific requirements of the user (the user is quite simply the person, or group, who wishes to use the results produced by a computer simulation) and a great variety of types of simulation have been developed. Well-known simulations based on computers include the control routines for full-scale flight simulators used in the training and testing of airline pilots, weather forecasting programs (a particularly difficult simulation task — see Chapter 4), video games of many types, the programs used by television producers to predict election results on live broadcasts, and the (sometimes) hugely complex programs used by engineers to help in the design of such diverse stuctures as North Sea oil-rigs, aircraft, and modern motor vehicles. In addition, much of the research into artificial intelligence and expert systems centres on the simulation of the workings of the human brain.

Who can use simulations

Until very recently, the number of viable simulations which could be used by 'the man in the street' have been very few, and very expensive. Typically, a finite-element model (used for analysing the strength of large and small mechanical structures, such as bridges, machine parts, and buildings) would only be available to the largest engineering companies or, at high cost, to other companies through computer bureaux. The running of such a model would cost the user many hundreds of pounds for every run, and the total cost of designing a product using computer simulation techniques would run to many tens or hundreds of thousands of pounds. Obviously these models have been beyond the reach of most companies and individuals. The big changes which have occurred recently are the dramatic fall in cost, and rise in capability, of the microcomputer and the development of a number of simulation techniques suitable for the smaller user.

Recently, many new applications for simulations on microcomputers have been recognised, particularly in the video games area. Simulation-based programs for the home computer market, other than games, include pools predictors, voice synthesizers, and some of the newer expert systems. These applications are of restricted use however; both memory and processing speed limitations are against the commercial exploitation of most practical simulations in the home. Video games, of course, provide a huge market for the skilled simulation writer. With a brief stretch of the imagination, a basic technique can be used to create hurtling asteroids, mutant sparrows, or undersea fishes. Flight simulators, motor racing games, and tank-battle simulators take a little more effort, but can successfully be implemented on even the cheapest of home micros.

Business simulations on the other hand are an entirely different kettle of fish. Simulations can be of immense value to a thriving business, but each simulation is likely to be unique. This limits the value of general-purpose programs, and most simulations are developed by professional consultants or by the most enthusiastic employees. Obviously, in this situation, the development of useful programs lags behind the potential demand for those programs. This book seeks to extend the knowledge of 'ordinary' programmers to include the basic techniques required to develop functional simulations.

Who develops simulations

The principal current users of large non-game simulations are development engineers, university research workers, and financial go-getters. The simulations used tend to be experimental; they are used to test out different ideas and determine alternative possible futures. Engineers use detailed simulations, usually home-grown, of prospective constructions, to investigate the potential problems/cost-savings of alternative designs. Universities either work on a contract basis for local industries or use incredibly advanced (at least they think so) techniques for the pure pleasure of the development. (In fact, these techniques can lead to genuine advances in 'simulator technology' — more of this later.) Financial modellers are usually interested in developing either sophisticated and large data-base programs for the rapid analysis of trends or specialised simulations of specific commercial organisations.

Which techniques can be used on micros

Unfortunately, some of the most striking simulations demand very high access speeds to huge reserves of memory. It is still the case that microcomputers have severe limitations in the area of memory access. Real-time simulations demand high processing speeds, another weak area for microcomputers. Despite the restricted capabilities of most microcomputers when compared to mainframes, many simulation techniques are usable on microcomputers. However, it is not possible to pack a quart into a pint pot.

Real-time simulations when run on microcomputers remain rather simple. Large data-base programs often require expensive hardware add-ons, such as hard-disk memories, and can be exasperatingly slow to run. Similarly, applications such as some of the latest artificial intelligence techniques are usually of limited practical value unless kept very basic and then tailored to meet a particular need.

The display capabilities of microcomputers are a particular trouble area. Display resolution is kept low in order to preserve memory and to reduce the processing demands on the microprocessor. The following program indicates one of the display problems. A second hand on a clock face is represented and drawn using the turtle graphics commands. Because the screen display is limited, there is a rounding effect on the representation of the second hand which produces slightly different images depending on the direction of movement of the turtle. INK is used first to draw the second hand and then to remove it whilst the turtle moves out and back. PAUSE at line 220 would need increasing if you wanted to show an accurate sweep-speed for the second hand. (See Chapter 3 for an improved version of this program.)

```
100 MODE 256
110 CSIZE 2,0
120 INK 7
130 PAPER 1
140 CLS
150 CIRCLE 80,50,31
160 POINT 80,50
170 TURNTO 0
180 PENDOWN
190 REPeat clock_hand
200  INK 7
210  MOVE 30
220  PAUSE 10
230  TURN 180
240  INK 1
250  MOVE 30
260  TURN 180-360/60
270 END REPeat clock_hand
```

Many useful techniques can be implemented on microcomputers however, as described in this book. Allowance must be made for the limitations of the machine being used, of course: the Sinclair QL, with its 32-bit processor and large on-board memory is particularly well suited to the running of simulations. Real-time simulations — even when written in BASIC, a notoriously slow-running language — can be developed if the screen display is kept very simple (of course, Sinclair SuperBASIC is considerably in advance of most other forms of BASIC, in terms of speed and versatility). Financial and developmental programs of quite a high standard can be implemented.

What is the scope of this book

As a basic introduction, this book gives a guide to the most useful techniques available and the way in which the techniques can be incorporated into practical simulations. Techniques are explained by way of a series of examples, covering a wide range of possible application areas. These examples are kept as simple as possible in order better to demonstrate the techniques used.

It is not possible in this book to describe fully all potential applications, and it is left to you, the reader, to assess your own particular requirement for such techniques. Further books in this series will give full listings of a number of developed simulations for business and personal use.

I draw on my own experience throughout the book and the examples presented in Chapters 6 to 10 are taken from my own case-book.

Organisation of the book

The general theories of simulation development and use are presented in Chapters 2, 3 and 4. Chapter 5 indicates the points of practical interest when programming the Sinclair QL to run simulations, and examines the potential of the machine. The remaining chapters examine the particular techniques available for the major applications areas and give example programs. A glossary provides a reference list of words and phrases which may be unfamiliar to you.

CHAPTER 2

When I was a Boy . . .

A background to simulations

. . . "the first computers. . . . . "

In this chapter I shall begin to introduce you to the concept of computer simulation. That is, what simulations are, where they came from, and why they are used.

Firstly, however, a short word of warning. Inevitably, there is a fair amount of jargon associated with this particular brand of computer science (you see, it's started already!). I'm not especially fond of jargon for jargon's sake, but some is bound to creep in from time to time — for which you must forgive me. More importantly, I shall be introducing you to a number of new concepts and must include some technical description. Don't worry about either of these, the concepts are based on the application of logic and the technical descriptions only need be remembered if you want to delve deeper into the formal side of simulations.

What are simulations

Although I have just called simulations a branch of computer science, it would be more truthful to say that what I am talking about is the 'art' of computer simulation. I say this because despite attempts being made (successfully!) from the earliest days of computers to run simulations, there is still no widely-accepted formal approach to their creation, little information is shared between users, and no yardstick exists to measure performance. Hence, in strict terms , there is no such thing as 'simulation technology'. Every new simulation is started afresh, and the final success of the program will depend on the inspiration and experience of the simulation designer. In this book I attempt to describe the vital first few steps required for the development of practical and useful simulations.

Already in this chapter I have innocently touched on two subjects which are fundamental to my concept of computer simulation. Firstly, the computer simulations covered by this book are programs. This may seem a little obvious to most of us but it is important to think this way right from the start — I shall explain further presently. The second message is that simulations are run on computers, and that computers are tools.

Computers mean different things to different people. Some people want instant access to stored information; the computer is a memory device. Some people want entertainment; the computer is a video-game machine. Some people want to produce documents; the computer is a word processor. Some people want to manipulate data; the computer is a programmable calculator. And so on and so on. Given sufficient time and imagination you could fill a library of books with descriptions of what computers can do for different people. The point, with all these varied uses, is that the computer is a tool. Versatile and expensive perhaps, but a tool nonetheless. Computers can be made to do what you want and in the way you want (there are limitations of course). What is needed at first is a clear idea of just what it is that you require. You can't get anything out of a computer unless you (or somebody else who has sold you a computer package) tell it exactly what to do.

This is where the programming comes in. All the applications (uses) noted above are really just examples of different computer programs. In exactly the same way, there are a host of different types of computer simulation but all are just computer programs. The techniques of computer simulation are used to develop such programs. By applying these techniques, some of which are more straightforward than others, practical programs can be produced which satisfy the criteria of doing what is required in the way that is wanted.

But what is a computer simulation? — what separates it from all the other computer programs? I shall give a basic description and then go on to explain the important points. A computer simulation is a program which duplicates or copies some of the features of something else (a system — the first bit of terminology to get to grips with) and predicts what will happen given various possible occurrences in a manner which is acceptable to the user. You can see now why I avoided attempting to define computer simulations in the Introduction — the definition is so vague as to be almost useless without further explanation.

I can give you a practical example of a program which may at first sight appear to be working as a simulation. The intention is to use the SuperBASIC BEEP command to produce computer-generated music; to simulate the workings of a composer. RND is used to produce a series of sounds.

```
100 REPeat key:noise
110 DEFine PROCedure noise
120   d=INT(32768*RND)
130   p1=INT(255*RND)
140   p2=INT(255*RND)
150   gx=INT(32768*RND)
160   gy=INT(8*RND)
170   w=INT(15*RND)
180   f=INT(8*RND)
190   r=INT(8*RND)
200   BEEP d,p1,p2,gx,gy,w,f,r
210 END DEFine
```

The result does not sound like music. We need to define the difference between 'noise' and 'music'. Rhythm is a quality intrinsic to most types of music. Changing line 100 to;

```
100 REPeat key:noise:PAUSE 10
```

improves our noise by adding rhythm, but there is still something missing!

```
100 N=10:REPeat loop:PAUSE 10:N=10+N-
50*(N=50):RANDOMISE N:noise
```

This line introduces repetition, a little more complex than expected due to the cycling features of the BEEP command, so our program is giving us a rhythmic, varying set of noises. Re-writing the program further:

```
100 N=1:o=N+.4:REPeat loop:PAUSE 10:N
=.1+N-o*(N=o):m=SIN(N):noise
110 DEFine PROCedure noise
120   d=INT(32768*m)
130   p1=INT(255*m)
140   p2=INT(255*m)
150   gx=INT(32768*m)
160   gy=INT(8*m)
170   w=INT(15*m)
180   f=INT(8*m)
190   r=INT(8*m)
200   BEEP d,p1,p2,gx,gy,w,f,r
210 END DEFine
```

takes advantage of the cycling and enhances and controls it to give a weird series of musical scales, but should it be called a simulation of a composer? We are simply using a modified set of random responses, and no composer worthy of his name would call it a rival. It would perhaps be more accurate to call the program a sound generator.

Additionally, and more importantly, we have not even attempted to duplicate the process of composing; we have merely taken accidental advantage of one of the built-in features of the computer. If we played around with the program long enough we might end up with something which was not too objectionable. We could not, however, credit the *program* with having composed a masterpiece, and so the program is not a simulation of a composer.

Systems thinking

In order better to define simulations, let me introduce the concept of systems thinking. Calling something a 'system' is a technique adopted by analysts (people who analyse things for one reason or another — systems analysts, for example, specialise in computer systems and optimise such systems in terms of hardware and software to best meet particular requirements) to contain and describe the thing which they are analysing. For example, a car designer may think of a whole car as a system. Describing it thus provides a guide as to what is contained within the system (the car) and what is not contained within the system (the road, other vehicles, the post office, etc.). There are two important qualities of a system: the system itself, and the interface of the system (the way in which the system interacts and communicates) with the non-system. In the case of a car these two features are both highly complex. The car itself consists of thousands of moving and non-moving parts — all the result of many years research and development. The interface between the car and the outside world is multi-faceted (don't bother remembering that one — I don't intend using it again!), that is there are actually several interfaces: the car with the undulations on the road, the car with the air that it is moving through, the driver's eyes and the road ahead and the constant pushing and pulling of various control levers, and so on.

Now let's extend the idea of systems a little further. Think of the specialist engineer, working as part of a design team engaged in the development of new cars (our previous system), who may be responsible for just a single sub-system of the car. A sub-system, in this context, is itself a system (say one wheel and its suspension) but is recognised as part of the whole system (the car). This sub-system can be described conceptually in the same way as the major system. That is, the sub-system has an interface (relative movement of the car and the road) and it has qualities (spring rate, damping, mass of moving parts, dimensions etc...) which define the way the sub-system reacts to the things around it. Similarly, a part of the suspension can be described as a sub-system in its own right. Thus we have levels of systems and sub-systems which fully define the car design and are useful in different ways. At the more detailed levels, specialist engineers can consider individual components or functional groups of components. These can be designed, tested, modified, and developed almost in isolation from each other (careful here; obviously sub-systems must be compatible with each other — this compatibility is achieved through careful definition of the sub-system interfaces and by placing constraints on the system qualities, such as size and weight).

Now comes the pay-off for thinking in terms of systems and subsystems. Each system can be simulated, once properly defined on a computer. The sub-systems can be simulated and these simulations combined together to give a complete simulation of the whole thing. A car simulation can thus be developed which will allow the designer to test ideas and optimise designs. Using a computer simulation makes sense economically. The cost of hardware development for something as complex as a car is very high indeed whereas computer simulations are relatively cheap. Computer simulations also make sense from the point of view of time expended. Several ideas can be tested in one day, using a simulation, which would take several months without it. Nearly all car designers and manufacturers make extensive use of simulations. Let's not get carried away, though, the simulations can only be used as design-aids (tools for the use of the designer) and are only as good as the programming. There are many areas of technology used in car manufacture which are not easy to incorporate into a simulation, and in many cases the best solution from a technical point of view is not acceptable for safety or aesthetic reasons. The simulation is only useful up to a point.

With a simulation available, the coarse level decisions about a car design, such as engine type and size, suspension type, passenger seating, and so on, can be evaluated and practicable concepts identified at a very early stage. Further work, from the basic concepts, at a more detailed level of component and component-group development will use sub-system simulations to indicate the most appropriate hardware designs. Thus, a new car can rapidly be outlined and defined, to quite a detailed level, in terms of hardware design and performance — all on a computer.

So now I have outlined the basic concept of 'systems thinking' and its relevance to computer simulations. Before we move on, I want to give you something else to think about on the subject of systems. This is an extension to the use of the technique, which many people find difficult to understand fully at first. Don't worry too much now if you have difficulty with the next paragraph — most practical simulations can be fully understood by the application of common sense, and the full details of 'systems thinking' are not required. If, however, you intend to spend much time on simulations involving people or institutions then read on most carefully.

Anything can be described as a system, not just items of hardware as described above. Thus, for example, a company board of directors can be described as a system, as can a classroom full of kids. Anything which can be defined as a system can be used as the subject of a computer simulation. The sucess of a computer program in correctly simulating a system is dependent principally on how well the qualities and interfaces of the system are understood and described. Further, the characteristic which ultimately defines a system, as separate from other possible systems, is the description of the limits to the system. Thus the system of the car wheel plus suspension is actually defined by the functional qualities of the suspension, anything not contributing to the function of suspending the wheel from the car (for example, the wheel arch) is not, strictly, part of the system. The system limits may be functional, spatial, temporal, imaginary, or some combination of these and others. A space-based video game is a simulation of a system which is totally imaginary!

The user

But let's get back to the simulation business. We can now re-state the description of computer simulations thus: a computer simulation is a computer program which duplicates the workings of a system and predicts the responses of that system to external and internal events, giving output in a manner acceptable to the user.

OK. So now we can cope with the system bit, and the internal and external 'events' are just things which happen inside the system or outside the system, but what's this business about the output? In fact, this is one of the things which separates good simulations from bad simulations. It is vital that the simulation provides the information required of it in the most suitable form. Remember that the simulation is supposed to be a tool: that means that, even though it may be a masterpiece of programming, it will be useless if it cannot do what the user wants. A user is the person (or group of people) who requires to use the simulation. This is not quite the same thing as the computer user. Usually, the author of the simulation is not the user of the simulation but is probably the user of the computer, or, in the case of commercial software, the author may be writing for unknown users. This is of importance because the workings of the simulation will often have to be designed around the output requirements of the user and these are often not well defined at the start of program design. This makes things that much tougher for the programmer, who must chose a balance between the versatility of the simulation in meeting possible future requirements and the complexity, and hence cost, of the program.

Thus, simulations are designed and developed to do a specific job and in a specific way. They are in practical terms very limited copies of the systems which they represent, often only representing a single characteristic of the system. Simulations can be described as 'functional' representations of systems. That is, they copy some feature of the functioning of the system. For example, when we looked at the system consisting of a car suspension and wheel, there are a number of possible simulations. One of these may indicate the dimensions of alternative designs, whilst another will simply indicate how the forces from the suspension are transmitted to the car, and yet another will investigate in detail the forces within the components of the suspension itself.

Although we now have a description of computer simulations, I will not attempt to develop this into a formal definition. I do not believe that it is possible rigidly to define computer simulations as separate from other computer programs. Many programs not usually thought of as simulations use the same techniques, and can, in certain situations, be used as the basis for simulations. Later in this book I give an example based on the use of the QL Abacus spreadsheet program (Chapter 6).

Early hardware and simulations

The first computers were designed as number-crunchers. They performed relatively simple (by today's standards anyway), but highly repetitive, calculations. Typical of the first applications were the generation of mathematical functions and tables, and the calculation of shell trajectories for artillery during the Second World War.

The UNIVAC 1 was the first commercially available electronic digital computer (UNIVAC was purchased by IBM in 1949 — a very astute business move indeed!) but it was not really until 1965 that commercial computers really became big business, with the introduction of the IBM 360 series.

Many more mainframe computers are bought for business than for laboratory or scientific use. Business computers are used, in the main, to process a great volume of data whereas scientific computers tend to be used to process less raw data but a great many calculations are performed per datum. It is the latter set of program applications which tend to include simulations. The upshot of all this is that the simulation business is still a young business. In practical terms, although much has been told of the more glamorous simulations, the majority of simulations are under 10 years old.

As is to be expected, the first simulations were rather rudimentary and simplistic. This was for three main reasons. Firstly, the computers available before the mid-1970s were restricted in terms of speed and memory availability. Secondly, simulation techniques and the approach required to develop satisfactory simulations took time to evolve. Thirdly, simulations need data and this data was often simply not available.

An example of this last point is the aircraft industry. It is possible to simulate the aerodynamics (the interaction of a body with the air it is passing through and the resultant forces produced on the body) of quite complex designs. Such simulations use equations and formulae which have been available, at least in principle, since the middle of the century. When computers of sufficient power to run such simulations became available to the aircraft industry there was no immediate rush to transfer research funds into the development of such simulations. Basic aerodynamic research was at that time very much an empirical discipline. Knowledge of the aerodynamic qualities of particular shapes of body was built up through the testing of models, full-size test-rigs, and production hardware in wind-tunnels, water-channels, and in flight tests. Combinations of shapes (eg wings plus aircraft body) again were tested in model form to obtain basic aerodynamic data. The data thus obtained was not directly usable in simulations — it was never intended for such a use. The development of simulations as useful tools for research and development was a long, slow, and expensive exercise. The traditional methods had to be adapted slowly to provide data and to verify the accuracy of the simulations as they started to produce results. These days, however, no self-respecting aircraft/missile/car designer is without a sophisticated suite of aerodynamic simulations.

An interesting example of a simple but very effective simulation from the seventies is the well-known game called ‘LIFE’. Invented at Cambridge University in 1970, this game simulates successive generations of a simple life-form as it flourishes or dies, depending on a few basic rules governing the life or death of individuals. In its simplest form, each member of a ‘community’ is represented as a letter O (or an asterisk or anything else will do) set on a grid pattern. A member will survive if it has two or three neighbours but die in any other circumstance (either of over-population or under-population). A new member is born in any empty space which has three neighbours. These rules are applied to the full grid area (typically a VDU screen or teletype page) and repeated to represent succeeding generations. The simulation is based on very simple rules but does simulate some of the important aspects of life. The interest in the game comes from attempting to set up an initial population which will eventually survive in some stable form. This game, along with the original Adventure game, was rumoured to be so popular with computer programmers at one stage that many companies banned it from their computers.

Operational analysis

Operational analysis is the study of the operations of a system. Often the systems analysed are financial or commercial systems although the more famous examples include traffic flow on roads and around airports, critical-path analysis, and time and motion studies. This has been the happy hunting-ground for simulation engineers for a number of years.

A well-known development of the late sixties and early seventies much used in operational analysis is the 'Monte Carlo Simulation'. This is in fact one of the basic techniques used in the simulation business (see Chapter 10) which was rapidly and successfully implemented. The Monte Carlo process really came into its own with the arrival of high-speed digital computers. The technique is used to predict the final results of a series of possible happenings. For example, if you wanted to find out how long it might take to write a computer program, you might chose to simulate the process using a Monte Carlo model. Suppose the program will consist of ten subroutines, which must be written and tested in sequence, each of which will take a minimum of one hour to write. Also, having written a subroutine, there will only be a 10 per cent chance of the subroutine working as intended first time. There is a 50 per cent chance that the subroutine will contain bugs which will take a further hour to sort out and a 40 per cent chance that the subroutine will have bugs which will take two hours to find and eradicate. How long will the program take to complete? We know that the 10 subroutines will take a minimum of 10 hours to write. Debugging time will vary from nothing to a maximum of 20 hours. Thus we can say that the program will take from 10 to 30 hours to complete. But this is no good for planning purposes, we need to know the average time as well as the possible extremes.

A Monte Carlo simulation considers each of the activities (in this case the writing of a subroutine is an activity) and selects a particular outcome at random but determined according to the in-built probabilities (10, 50, or 40 per cent). Thus, for each subroutine a particular writing and debugging time is determined at random and a running total is built up of the overall programming time. At the end of the simulation you are given a single example of the time taken to write a ten-subroutine program. Unfortunately you only have a single example so far. The simulation must be run many times before you can be reasonably sure that you have sufficient data, in the form of a whole series of simulated programming tasks, to derive an average time. The average time taken is found by adding all the results and dividing by the number of results. If you are particularly keen to demonstrate this for yourself then you can write a simple simulation as described above and run it a few times. There are a number of ways that such a simulation can be written and I include below a listing of one of the most obvious.

```
 10 REMark      SIMPLE MONTE-CARLO SIM
ULATION (SINGLE PASS)
 20 P1=0.1:P2=0.5:P3=0.4:REMark
  BASIC PROBABILITIES
 30 TIME=0:REMark    INITIAL TIME
 40 FOR N=1 TO 10:REMark          LOOP
FOR TEN "SUBROUTINES"
 50  IND=RND:REMark               FIND
 RANDOM NUMBER (0 TO 1)
 60  EVENT=2:REMark    THESE THREE LIN
ES SELECT ONE EVENT OUT OF
 70  IF IND<=P1 THEN EVENT=1:REMark T
HE THREE POSSIBILITIES
 80  IF IND>(P1+P2) THEN EVENT=3
 90  SELect ON EVENT
100   ON  EVENT=1
110    DELT=1:REMark      TIME FOR EVE
NT ONE
120   ON EVENT=2
130    DELT=2:REMark      TIME FOR EVE
NT TWO
140   ON  EVENT=3
150    DELT=3:REMark      TIME FOR EVE
NT THREE
160  END SELect
170  TIME=TIME+DELT:REMark      RUNNIN
G TOTAL OF TIME SPENT
180 END FOR N
190 PRINT "TIME TAKEN TO WRITE PROGRA
M = ";TIME;" HOURS"
```

Running this program around 20 times is sufficient to give, within one hour, a result of the true average time (around 23 hours). You will notice a few other points here which explain why Monte Carlo simulations were quick to appear, what their benefits are, and what their drawbacks are. They are simple to create: the simulation follows very closely the actions of the system being simulated. But a large number of runs may be required before a reliable average result is obtained, although this may not be important. After a number of runs have been made, additional data is available, such as the range of resultant times in the example above, which can be very difficult to obtain any other way. Of particular note is the value of Monte Carlo techniques to real-time simulations because of the close representation of the actual workings of the simulated system.

Obviously, the Monte Carlo process requires a large number of calculations to be made to complete a single cycle of the simulation. This is where computers came to the aid of a process already in use as the basis for many non-computer simulations and really made it into a feasible method for widespread utilisation (I'm sorry, I'm beginning to slip back into jargon — computerspeak you might say!).

Industrial simulations

Another early application area for the use of simulations came from the engineering industries. I have already talked of the car and aircraft industries. These two examples are perhaps the most widely quoted, but the general impact of computers on engineering as a whole has been very pronounced. Many of the basic tools of the engineer are built around the laborious repetition of various calculations. The desk calculator and, more recently, the computer have changed the way in which engineers work.

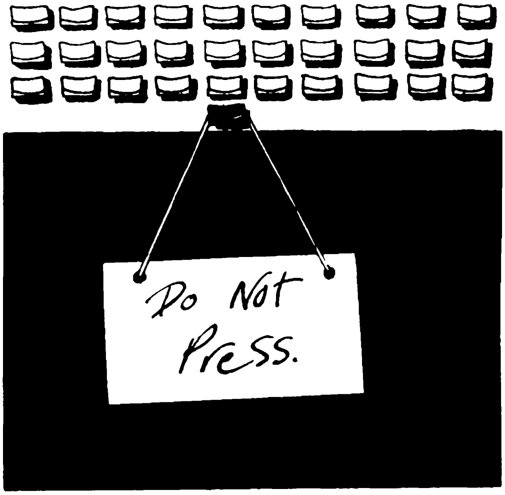
In order to design a product (whether a bridge or a typewriter or anything else) an engineer will always simulate the finished article. Until the advent of the computer, such simulations consisted of drawings and hand calculations. These were, in principle at least, readily transferable to computer. In practice, of course, reality is a little different. Simulations are certainly big business in the engineering industry but a lot of diagrams are still drawn and desk calculators are well used. Simulations for today's engineer take the design and development process into a completely new area. One of the truisms of computing is that computers do not replace people, but they do change the way that people do things. Ten years ago, it would not have been practical to use finite-element techniques to simulate the time-transient behaviour of a North Sea oil-rig in a force 10 gale. Today it is considered by many to be dangerous not to. In the main, the simulations developed for and used by engineers tend to be complex and large.

The great growth of simulations has really occurred in the last 10 years. Early work tended to be done in universities, in some of the larger engineering companies, and in support of various studies by operational analysts. As 'computer literate' graduates started to appear and computers were made available to many more companies, the use of simulations became quite common. Recently, serious financial simulations have started to appear (other than those used by the US government!), expert systems are available commercially, video games are becoming more and more sophisticated, and educational software is not far behind.

CHAPTER 3

But What Does This Little Black Button Do?

How simulations are used



"... simulations can be used or abused...."

The basic types of simulation

There are three basic functions for simulations — emulation, analysis, and prediction. In the following sections I describe each of these, the uses to which they are put, and some of the more important approaches adopted in developing simulations.

Emulation simulations

Emulations are direct copies of real-life systems and are used either as low-cost substitutes for use of the real thing or to represent situations which would be impracticable to achieve using real hardware. Examples of emulators include flight simulators, video games, war games, and some engineering simulations.

The purpose of the emulation is to represent as realistically as possible the time-dependent workings of the simulated system. It is this time-dependency which is important, and programs are written around the requirement to simulate the state or condition of the system at regular intervals. The time taken between successive time intervals is known as the simulation time-step. This time-step is not the time taken to perform one loop of calculations but the time represented in the real system by the calculations. In many situations the complexity of the simulation will lead to a computational time far in excess of the time-step. For example, a simulation of the flight of a rocket may have a time-step of, say, a tenth of a second whereas the computing time for each time-step may be several seconds. The time-step chosen for a simulation of the tidal effects in a river estuary, for example the River Thames upstream of the flood barrier, may be of the order of 15 minutes but the computing time may be from under a second to a few minutes depending on the complexity of the representation. In special circumstances it may be desired that the simulation time-step and the computational time should be the same. Such simulations are real-time simulations and are intended for direct interaction with a human operator.

Here is a program that draws a clock with a second hand. It is a modification of the program given in Chapter 1 and uses the turtle commands to draw and 'undraw' the second hand.

```
100 MODE 256
110 CSIZE 2,0
120 INK 7
130 PAPER 1
140 CLS
150 CIRCLE 80,50,31
160 TURNTO 0
170 PENDOWN
180 REPeat clock_hand
190  POINT 80,50
```

```
200    INK 7
210    TURN -360/60
220    MOVE 30
230    PAUSE 10
240    POINT 80,50
250    INK 1
260    MOVE 30
270  END REPeat clock_hand
```

But is it 'real-time'? Changing the pause at line 230 will increase or decrease the sweep-speed of the second hand. As long as the time the hand takes to complete one cycle is the time which you require (it could be 1 minute, 10 minutes or even 1 hour), then it is.

Usually the sophistication of a real-time simulation is limited by the time available for computing. If a long time-step is chosen, then the response of the simulator to the actions of the operator will be slow. This may not be important if you are engaged in the education of nuclear power station operators where time-steps of the order of one second would be more than adequate. A space invader game must respond very much more quickly, however.

Obviously, all the computations for a single time-step must be completed before the end of the time-step. All inputs must be made, responses of the simulated system to the inputs must be estimated, outputs must be recorded, and the simulation set up for the next time-step. Very often, simply writing down the list of operations to be performed for each time-step will provide a structure for the program itself. Much more care is required for real-time simulations, however, particularly with the timing of inputs and outputs.

Analytical simulations

The second type of simulation is that used for system analysis. Examples include financial models, expert systems, management information systems, computer-aided design packages, and reliability assessment programs. Note here that not all the applications noted above need be simulations, but simulations may well be involved one way or another. It is rather dangerous to generalise about this group of simulations; they are designed for a specific job and their program structures are very varied. Some simulations are similar to the emulators described above but adapted for a different use and modified accordingly. For example, a financial package may be based on the emulation of the Stock Market, representing the buying and selling processes, but modified to draw attention to particular points of interest, perhaps to highlight shares which are likely to change rapidly in value. Rather than simply performing a time-step and dumping a full set of data, additional calculations will be performed on the latest set of output to analyse for points of particular interest. Also, comparative runs with different starting assumptions and different input data may be used to indicate not just the absolute response of the system to varying inputs but also to investigate the system sensitivity to inputs. This is the 'rate' at which the system response varies with a changing input. Real-time emulators are not usually used directly for analysis but the results of a number of runs of such a program may be recorded for future analysis.

Reliability assessment and critical-path analysis are worth a bit of further consideration here. These are good examples of applications for programs which take input in the form of a number of very similar data items (all related in some way), and manipulate the data in such a way that high-level information is output. In the case of reliability assessment, raw data on the number and type of breakdowns of a product is combined and processed to give an assessment of the overall reliability of the product, or to identify particular problem areas, or to identify how the reliability has been changing during development. Critical-path analysis takes a planner's estimates for the duration and requirements of a number of tasks and combines these to indicate the overall project timing and resource requirements. In addition, the set of tasks which are most important to the completion of the project (the critical path of tasks) are shown and the program may indicate the best way to sequence tasks to minimise risks and to make the best use of available resources. Large projects, such as building works or product development programmes, rely heavily on careful planning of the individual tasks. Raw materials must be ordered in advance, a stable workforce must be maintained and provided for, use of special equipment (such as expensive test equipment) must be planned in advance, and so on. The difficulty of doing all this planning by hand and being able rapidly to update such plans if required leads to a high demand for programs which can automatically perform the necessary analyses.

Predictors

Finally we come to simulations designed to make predictions. Included in the list are weather predictors, profit and loss predictors, reliability-growth programs, and crisis forecasters. Yet again, these often are based on emulation programs but are further developed in two important areas. Firstly, these simulations take a starting situation (typically representing time now) and simulate a number of time-steps into the future. Stored data may be used to provide inputs as the program runs. The second important feature of predictors is that they are usually developed to be able to make use of poor data. This is data which is not well defined, may be non-existent, or is known to be unreliable. As simulations predict further into the future, or into other areas where knowledge is lacking, it is important that the simulation give some indication of the confidence which can be placed in the validity of its output. By making special provision for the use of data which is known to be poor, estimates can be automatically made of the reliability of the output. Emulations without the added sophistications noted above can be used to predict future events but must be used with great caution.

A second major form of predictor is that which is based on the analysis of trends in past data to predict forwards. Football pools predictors take data in the form of recent results of games played, select those results of particularly relevant games and use these to indicate the current performance of the teams involved. Armed with this indicator and the data on where the game is to be played the program predicts the likely outcome of the match, usually along with an indication of the probability of a suprise result. This process is repeated for all the games appearing on the pools coupon and a selection of the most likely results prepared.

Another example of a sophisticated trend-analysis predictor comes from the financial world. Knowledge of the variation in value of stocks, shares, commodities, etc., available on the world financial markets is vital for investors and up-to-date knowledge can be the making of fortunes. Many individuals and groups maintain predictors for various uses and of various types. The most obvious are trend analysers, working on past data and predicting forward days or even months. Other predictors remove the time-dependency by relating one factor, such as the value of a particular commodity (eg gold or oil), to another, for example some measure of overall level of trade. In times of recession or low levels of trading on certain markets, such predictors can be useful in indicating the value of moving invested funds from one market to another.

Use and abuse of simulations

In the preceding paragraphs I have given a brief introduction to the simulation business, its history, and some of the major approaches adopted. I may have made it sound as if simulations can be used for almost any purpose. In a simplistic sense this is in fact true; you can if you wish simulate almost anything.

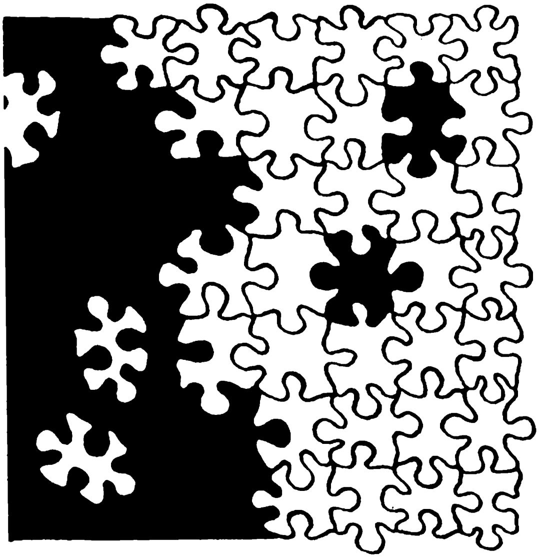
However, there are many reasons why you shouldn't bother. Many problems are more easily solved without going to the expense of creating a special simulation program. The example I gave of the simulation of writing a computer program is a case in point. That particular problem can be easily solved with a hand calculation, so why bother with a simulation that requires to be run at least 20 times? Often, the simulator required to mimic a given system would be too complex or difficult to verify (check the correct operation of) and any results obtained would be suspect. This does not mean that simulators should not be considered, it just means that other means of obtaining the desired results should be investigated as well. Weather prediction programs fall into this category at present — there is simply not enough data and knowledge available at present to make a very large-scale simulation which produces acceptable results, but work goes on to develop just such simulations. It must also be remembered that simulations are simplified representations of systems and thus cannot fully replace the real thing. Indeed, most simulators are very approximate representations of systems which themselves are not fully understood and thus simulators can only be used to give a rough idea of the system behaviour.

I want to make two more important points here. Firstly, as for most computer programs, a simulation is only as good as its programmer and is only as useful as its initial design constraints. Secondly, the major limitation on most simulations is the basic understanding of the system to be simulated. I don't want to say more than this at present, but please remember what I said at the beginning of this chapter, that simulations are tools. Tools can be used or abused but if used properly they can be of great benefit.

CHAPTER 4

Getting It Together

Practical techniques



"...a pattern for the program..."

This chapter looks at the basic techniques which I suggest should be adopted if you want to develop simulations which work as you want. I'm not going to give you a great long list of points to check off, rather a set of basic approaches which will help you avoid the major pitfalls and will bypass the common causes of delay.

The importance of defining the requirement

First define the requirement. This is the message that applies to all forms of programming; any half-way decent guide to programming should stress this one. Starting to write any computer program without defining the ultimate requirement in some way is a bit like trying to train a monkey to drive you to work, bits of the trip may be of interest but the journey is likely to end in tears.

But what is a requirement and how do you go about defining it? The requirement is based on what the user (the user, not the programmer!) wants from the simulation. This simple observation becomes very clouded when you actually start work on a new simulation; it is usual for the user to have a very loose idea of what they want at the start of development and to revise these wants as development continues. This in fact is one reason why it is important to try and define as much as possible of the requirement right from the start. If changes do occur, it is possible to go back to the original requirement and decide whether modifications can be accepted or whether the original requirement should be kept.

A first step is to try and write down the requirement. If this can't be done then forget the whole thing at this stage: if you can't manage even a description in written English then how can you expect to develop a concise description in the very much more formal medium of a computer program?.

The description should contain the following:

A definition of the system to be simulated, its limits, its functioning, and the interfaces (remember, the inputs and outputs to the non-system).

What is desired from the simulation.

What data is, or will be, available and in what form.

What form of output will be possible and what will be most desirable.

How much time/effort is available for program development.

Other points.

This last is a catch-all and will vary from case to case. Some of the points which may be important are: what computing hardware will be used for development; what computing hardware will be used to run the developed simulation; will any other hardware be involved; what sort of organisation will be using the simulation; is the simulation to be sold commercially; what sort of testing of the simulation will be done; what developments might be required in the future; and will the user require continued software support. Some of these points are covered in the following paragraphs but I make no apologies for avoiding discussion of each of them in detail. In certain cases each of the points above may become important: just how important and what form the importance may take will vary from case to case.

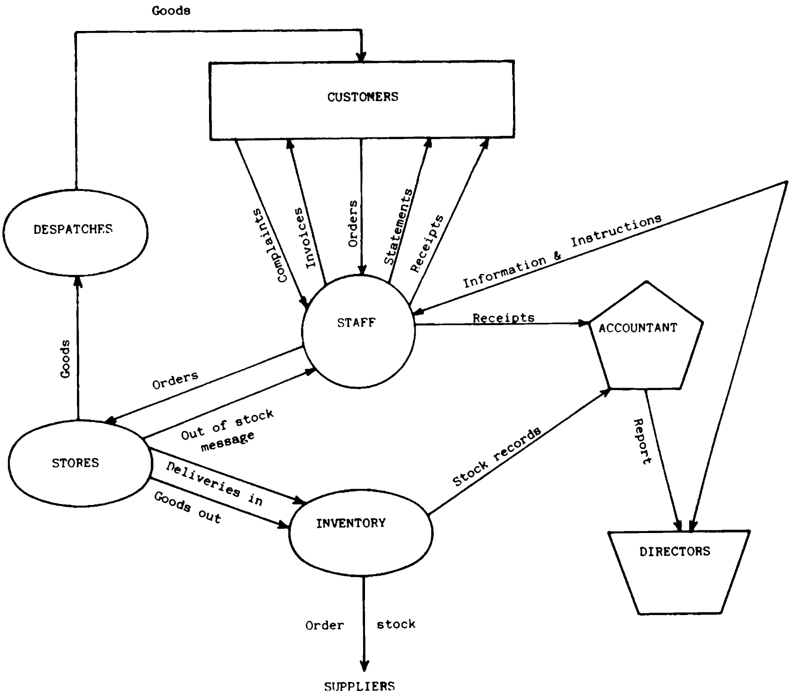


Figure 4.1: Rich Picture

Diagrams

Diagrams can be very useful for defining and analysing the system to be simulated. These can start off as simple lists of the system components but can include many useful factors. For example, computer systems analysts may use 'rich pictures' (**Figure 4. 1**) which show the parts of the organisation which is to be provided with a computer system, say a wholesaler of electrical components, on which communications can be indicated as arrows. Thus the rich picture can show all the actions from receipt of an order through the post, to retrieval of the desired order from the store, to packaging the order, sending out the goods, recording remaining stock levels in store, sending out bills, and so on. It is vital to identify these communication lines if a computer simulation is to be installed, as the computer will be providing many of them. Also, it is very likely that improvements to the existing communications will be required and these are most easily identified using a communications diagram.

Another form of chart used by systems analysts and simulation engineers is the 'function chart' (**Figure 4. 2**). This quite simply is a means of noting down, in a reasonably formal manner, a breakdown of the functions of an organisation. Thus, using our example above, the function of Joe Bloggs of the packing department may be to collect a store's order form and pass this to Bert (stores), wait for the items to be dug out of the box at the back of the warehouse (or take back an out-of-stock message), sign a receipt for the goods, obtain packaging materials (probably with more paper work involved), package the order, and so on. The function chart, when expressed in this way, has obvious connections with the rich picture.

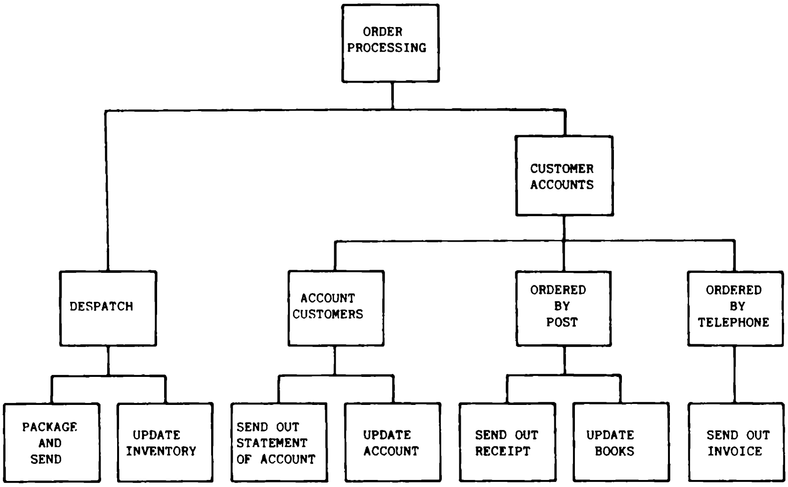


Figure 4.2: Function Chart

A more basic diagram technique, which I often use when struggling to define the operation of a new system, is the 'causal diagram'. This diagram concentrates on the possible responses of a system to different situations. For example, a very simple causal diagram might consist of:

Fire → Smoke → Alarm

This means fire causes smoke which causes an alarm. The arrows show the 'causal links' between possible system characteristics. If we are interested in looking more closely at the consequences of fire then we can extend the diagram as in **Figure 4.3**.

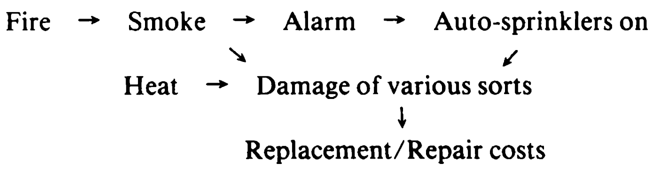


Figure 4.3: Causal Diagram.

Notice that I am mixing several types of item on the diagram: some real components of the system, some financial, and some just describing responses of the system.

Feedback

I'm going to digress a little here, while I'm on the topic of causal diagrams, and introduce the concept of feedback and feedback loops of various types. In the example above, the switching on of the sprinklers should result in a reduction of the intensity of the fire, less heat and smoke, and so less damage (but some water damage). Here is a simple form of loop — the fire causes the sprinklers to come on which damps down the fire. This loop is called a closed loop, the loop operates automatically and is contained within the system itself. An open loop requires the action of something external to the system, for example the local fire brigade, to complete the loop.

The program listed below demonstrates a feedback loop. If you type in and run the program you will see a '*' in the centre of the screen. You can move the '*' using the cursor keys to the left and right of the space-bar. Notice how the program reacts to your movement of the '*'!

```
100 MODE 256
110 CSIZE 2,0
120 INK 7
130 PAPER 1
140 CLS
150 CURSOR 200,100
```

```
160 PRINT "*"
170 GX=-.1
180 GY=-.1
190 X=0
200 Y=0
210 REPeat FEED_BACK
220   I$=INKEY$
230   IF I$<>"" THEN
240     REPeat KEY_LOOP:J$=INKEY$:IF J
$<>I$ THEN EXIT KEY_LOOP
250   END IF
260   I=CODE(I$)
270   SELect ON I
280     ON I=208
290       Y=Y-10
300       SCROLL -10
310     ON I=216
320       SCROLL 10
330       Y=Y+10
340     ON I=192
350       PAN -12
360       X=X-12
370     ON I=200
380       PAN 13
390       X=X+12
400     END SELect
410   DX=INT(GX*X+.5)
420   PAN DX+(DX>0)
430   X=X+DX
440   DY=INT(GY*Y)
450   SCROLL DY
460   Y=Y+DY
470 END REPeat FEED_BACK
```

This is an example of feedback being used to control something, in this case the position of the '*'.

Many simulations involve feedback of one form or another and this can become a major problem during the development of the program. Feedback can be represented in a number of different ways. If the wrong approach is adopted, then it is possible for the program to become numerically unstable (the output starts to oscillate and will usually produce results which are obviously incorrect) or the feedback item will be ignored. If you experience such problems then all I can realistically suggest here is to read an introductory book on control theory — I cannot do justice to the practical representation of all possible forms of feedback. Some odd comments though: a common source of error in feedback representation is selection of too long a time-step, try a shorter time-step if you suspect errors; sometimes a process of iterating on to a valid result, by repeating a feedback operation several times over for each time-step, will be required; pay very careful attention to the limits within which your various equations and so on are valid, many problems only appear when you approach or accidentally exceed these limits - a knowledge of mathematics is very useful here.

Now back to the main feature.

Simulation structure

Having written down the requirement, supported by diagrams, a number of possible actions are open to you. You can, if you so desire and you want to take the risk, dive straight into computing. This is only advisable, however, if the simulation is very simple (like the example given in Chapter 2) or you are very familiar with all aspects of the simulation techniques to be used and the system being simulated. In most cases the next stage after defining the initial requirement is either to re-write the requirement in greater and greater detail, until you are satisfied that you can do no better, or else to begin to define the structure of the simulation and the techniques which you are going to use. The first course is better if the simulation requirement is particularly well defined or particularly badly defined. If well defined, then the simple process of repeatedly writing down the requirement will very often also serve as a pattern for the computer program. A straight translation of the text will give the functional parts of the simulation, all that need be added will be the supervisor routines (to guide the simulation activities) and input/output routines. If the basic requirement is particularly badly defined, the repeated re-writing of the requirement will help to concentrate the mind on those features which are probably more important.

Defining the overall structure of the simulation and beginning to define the techniques to be used is a logical precursor to the writing of the program. I have the rather bad habit of attempting to do most of this in my head, with the consequence that I often lose 'brilliant' ideas somewhere 'twixt brain and keyboard. My advice to you is to start off at least by getting something down on paper. Start at a high level, that is decide on the overall workings of the simulation before looking at the detail of any one part. Use diagrams, words, notes, whatever helps you think about the problems of programming. Also bear in mind that you will probably want to refer back to these scribblings at a later date to try and find out what has gone wrong or to find out just what it was you were thinking of in the first place!

Chosing the basic technique

Having gone through the process of defining just what is wanted from the simulation, it is time to look at how to get the thing working.

The descriptions given in Chapter 3 of the basic types of simulation which can be developed will have given you an indication of the range of possible approaches which will be needed. Very few simulations are of systems which will react in only one way to a given situation. In nearly all cases we must allow for the system to react in several possible ways to any particular set of circumstances. It is normal to define the system reactions in terms of probabilities. To each possible reaction, a probability of that reaction is defined. Let's put that into simpler terms. If a system can react in one of several ways, we must know which reaction is most likely and by how much before we can simulate that system.

Monte Carlo

The differences between the basic types of simulation centre on the way in which we deal with these probabilities. The Monte-Carlo process described in Chapter 2 is one way of simulating system responses to various situations. In the Monte-Carlo process, a single response is selected randomly according to the probabilities of all possible responses.

Deterministic

Another process which is often used instead of the Monte Carlo process, is the deterministic process. In the deterministic process each system response is determined from the most probable response. If, for example, three responses are possible in a given situation with probabilities 0.15, 0.45, and 0.4, the deterministic process will always select the response with a probability of 0. 45. The other system responses will never be simulated. The deterministic process is useful in emulations of large systems, particularly real-time simulations, where a single run-through of the simulation is all that will be made. Examples of simulations often based on the deterministic process are weather predictors, election forecasters, video games, financial models, and some of the simulators used for training personnel.

A note here about deterministic models. Often a deterministic approach is the only way of realistically simulating large and complex systems but it can lead, in some cases, to unreliable conclusions. A good example is in the long-term prediction of the weather. A huge quantity of data is required and much of this is difficult to obtain. In consequence it is difficult to fully describe the starting system-state: the current condition of the system is known as the system-state. At each time-step in the simulation process, a new system-state is derived. However, real life does not always follow the most probable course and thus there will be differences between the predicted system-state and the real weather. These differences will grow with each time-step and it has been found that the long-term prediction of weather patterns is a very difficult task. The real-life system is simply too complex and possible responses too varied to permit reliable simulation, even on the most sophisticated computers available.

A common extension to the deterministic process is to incorporate a form of accumulator which records system responses and biases future probabilities accordingly. Thus the system response can be prevented from entering into a long series of actions which can be justified individually but which are unlikely when combined together.

Markov chain

Both Monte Carlo and deterministic processes, and the simulators based on them, result in serial operations. One action follows another, and each action leads on to a further operation. There is another class of processes which result in parallel operations. These are processes which simulate more than one possible response of the system to a given situation. Obviously, a simulation such as this must maintain a running record of several system-states. One process which keeps a record of all potential system-states is the Markov chain process. In its most useful form, two matrices are set up (I'm not about to explain matrices to you — in this case they are used as a means of storing probabilities and formulae). The first matrix holds a running record of the various possible system-states. There must be a limit to the total number of possible system-states and the state-matrix holds a set of probabilities of the system being in each of these possible states. At the start of the simulation the state-matrix will usually hold a probability of one in the location representing a single starting state and zero probabilities representing all the other possible system-states. The second matrix is called the transformation-matrix and holds a series of equations which determine the way in which the system can change from one state to another. Often the equations in the transformation-matrix are reduced to simple probabilities. Possible system-states are numbered and the equations of the transformation-matrix give the individual relationships of each numbered system-state to each of the other system-states.

At each time-step, or point in the simulation where a system response is to be modelled, the state-matrix and the transformation-matrix are multiplied together to give a new state-matrix. The probability of the system being in any of the possible system-states is thus continuously available.

There are other parallel processes, such as the branch-search sometimes used in chess-playing computers, but the Markov process is the most widely known.

Technique to suit the simulation

It is not unusual, particularly in the analysis field, to create a new process for each new simulation. This is because the requirements of the simulation are likely to be too individual and the computations too complex to make use of one of the text-book processes. For example, many practical parallel processes do not keep track of all the possible system-states but only those which are of most interest. This can greatly reduce the computing time.

Another possibility is, at each stage in the simulation, to calculate an 'average' system response or state. Typically this will use a number of separate system responses, much as used in a Markov process, which are multiplied by their relevant probability of occurring, and added together. This technique is valuable in situations where the system is free to respond in a range of ways rather than in a limited number of discrete ways. An example, which also shows up the dangers of this particular technique, could be a simulation of the throwing of darts at a dartboard. Suppose we represent the throwing action and the flight of the dart as a simple probability distribution — that is we define a set of probabilities (adding up to one if totalled) of the dart ending up at any position on the dartboard for any particular throw. The averaging process will typically calculate the distances of each possible position on the board from the centre and the relative angle of each position from the upright. Each such distance/angle is multiplied by a probability provided from the input probability distribution and added into a total figure which directly gives an average result. If the probability distribution is symmetric — that is, there is an equal chance of hitting above or below the centre and left is as likely as right — then the 'average throw' will land dead centre. Obviously this result is of little use to us when stated in this form; we know that the important thing about darts is not what the average throw gives but what happens on average over a series of throws. It is the spread of results about the average (or mean) position which is more likely to be of interest, and some additional calculations are necessary to obtain this information.

The example just given is a useful one with which to compare the responses of the various processes which I have been discussing. A Monte Carlo simulation would give a random dart position, different for each run of the simulation. This would give the closest representation of the actual system in action. A deterministic simulation would give a single dart position each time, dependent on the highest probability. If the thrower has a tendency to drop every fifth dart in their foot then this is probably the result which would always be indicated by the deterministic simulation. The averaging process, as indicated above, will show what is the average result of a number of runs. A parallel process simulation will give an output indicating all possible dart positions and their likelihood (in this simple example this will just be an output of the probability distribution which we input in the first place) and will probably also give some further analysis of the most likely dart positions.

Some simulations combine characteristics of Monte Carlo, deterministic, averaging, and parallel processing, using the appropriate techniques in different parts of the simulation. The choice of a particular technique will depend on the requirements of the simulation, the nature of the system, and the various restrictions on computing power, time available, data availability, and so on. The specialist chapters of this book give examples of my decision-making process and the consequent adoption of various techniques. One of the important restrictions on the simulations covered by this book is that I must keep the programs as short as I can whilst demonstrating the techniques. In your simulations you will not have this constraint but it is likely that some other factor will be important: you must think long and carefully about the basic techniques available to you and just what the advantages/disadvantages of each will be.

Development of the simulation

You know what you want to do, you know how you want to do it, but how do you actually set out to create a program to make the thing real?.

There is more than one way to skin a cat (or write a program!). Below is given a program which moves a '*' around the screen using the PRINT command:

```
100 MODE 256
110 CSIZE 2,0
120 INK 7
130 PAPER 1
140 REPeat square
150  X=12
160  FOR Y=4 TO 14
170   X_Y_PLOT X,Y
180  END FOR Y
190  Y=14
200  FOR X=12 TO 22
210   X_Y_PLOT X,Y
220  END FOR X
230  X=22
240  FOR Y=14 TO 4 STEP -1
250   X_Y_PLOT X,Y
260  END FOR Y
270  Y=4
280  FOR X=22 TO 12 STEP -1
290   X_Y_PLOT X,Y
300  END FOR X
310 END REPeat square
320 DEFine PROCedure X_Y_PLOT (X
,Y)
330  LOCal n
340  CLS
350  FOR n=1 TO Y
360   PRINT
370  END FOR n
```

```
380   FOR n=1 TO X
390    PRINT " ";
400   END FOR n
410   PRINT "*"
420  END DEFine
```

Run it and note how fast the '*' moves.

The QL also has a CURSOR command, and using it instead of PRINT gives quite a saving on program lines. Replace lines 330 and 340 with the lines given below;

```
330   CLS
340   CURSOR 12*X,10*Y
```

and delete lines 350 to 400. Do you think the '*' moves any faster?

Another alternative is to use SCROLL and PAN. An even greater saving in program lines, but what about the speed of movement this time?

```
100  MODE 256
110  CSIZE 2,0
120  INK 7
130  PAPER 1
140  CLS
150  CURSOR 120,40
160  PRINT "*"
170  REPeat square
180   FOR N=1 TO 10:SCROLL 10
190   FOR N=1 TO 10:PAN 12
200   FOR N=1 TO 10:SCROLL -10
210   FOR N=1 TO 10:PAN -12
220  END REPeat square
```

It is important when writing programs to consider all possible alternatives.

Different programmers work in different ways. Some draw up detailed flowcharts (diagrams showing the functions of each part of the program and indicating connections between different parts of the program) and only convert these into a program when happy that they have the whole thing sorted out. Other programmers work very much from their heads and start writing program lines right from the start. I tend to be one of the latter, but I can see the value in being more systematic about programming.

At the very least, I think that it is well worthwhile writing down a basic block diagram of the major subroutines which you are going to have to use. This diagram will, in the first case, be based on your description of the requirement and your choice of simulation technique. You will need a main or controlling routine: this is a routine which determines the overall functioning of the program, calling specialist subroutines as required. Next come the input and output subroutines, based on data availability and the user requirements. One or more routines will be required to set up initial values for various program variables and to do any manipulation of the input data required before the main functional parts of the simulation are started. These main functional routines will be determined by the techniques to be used and the system to be simulated. A small number of simple subroutines may be required, called directly from the main routine, or, at the other end of the scale, many levels of subroutine may be required representing different possible responses of the system. Whichever the case, this is where the value of a block diagram is really shown: keep a close eye on how the simulation is developing and you are much less likely to suffer unpleasant surprises.

How you go about writing the subroutines themselves (programmers call this coding, despite the fact that it has been many years since computers would only accept machine or assembler code instructions) is up to you. I think that the most logical method is to keep on dividing up the description of the functioning of the subroutine until you can't describe it in any greater detail. At this stage each functional description can be converted into a program line. That is the logical approach but not the one which I usually adopt, I tend to start at the beginning and work my way through to the end. This is not always a good idea and I would advise you to attempt to do things the logical way rather than pick up my bad habits.

Keep notes on your use of variables, try to make the names of the variables meaningful so that you will be able easily to understand the program when reading through a listing, but do keep those notes. Note down the description of each variable and, if possible, the relationship of that variable to other variables.

Verifying the model

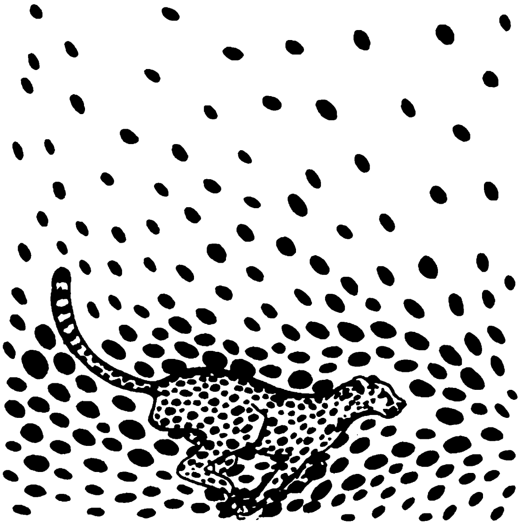
As each subroutine is completed, it should be possible to make up a simple test to make sure that the subroutines work as they are supposed to. A more important level of testing must be performed when you get the whole program together. The full workings of the program must be verified before the simulation can be used for its intended purpose. This isn't just a simple process of performing various runs and correcting any 'bugs' that appear. A systematic series of test runs must be planned and executed. Records must be kept of the results of these tests and the results carefully evaluated to detect any sources of inaccuracy. Tests to be completed include such things as running the simulation with zero inputs, in the expectation that the simulated system will remain in a stable state; running the simulation with very much simplified inputs; and running the simulation with a set of very extreme inputs. These tests will indicate any major errors but no test can be guaranteed to reveal everything: this is why careful planning and evaluation of the test runs is important.

One last piece of information, before I quit this chapter, concerning the use of a simulation for analysing the sensitivity of a system to changes in input. I am bringing this to your attention here because a simple form of sensitivity analysis can also be used to indicate possible errors in the program which would otherwise go unnoticed. This sensitivity analysis is typically performed by keeping all but one of the inputs constant for a short series of runs. The remaining input must be given several different values as the test proceeds and the responses of the simulated system noted. By plotting the results of such a series of tests on graphs for each of the various inputs, any particular sensitivity of the system can be readily identified. For test purposes watch out for unexpected sensitivities, behaviour which appears unpredictable, and any response which otherwise does not agree with your understanding of the system. If any of these circumstances applies, then get suspicious and check the workings of the program once more, preferably setting additional runs.

CHAPTER 5

QLued Up and Ready To Go

What the QL can do



"....speed is required...."

Although simulations are programs, they are also designed around the particular characteristics of the computer hardware being used. Obviously, not all simulations can be run on all computers. The power, memory, operating system, and hardware peripherals all determine what can and what cannot be simulated on a given computer. This chapter considers the Sinclair QL and determines its strengths and weaknesses.

Basic description

If you have a QL, or have thought seriously about getting one, then you will already know the basic construction details of the machine. I want to go over these once more, however, and point out those features which are particularly important to the budding simulation engineer.

The Sinclair QL is a microcomputer. This means that it is based around a special integrated circuit (usually called a chip because the circuit itself is formed on a very small chip of silicon) called a microprocessor. This device is special because it permits logical operations to be programmed by a user and can hence be used as the basis for a number of types of equipment, including computers. The microprocessor is also important because it is relatively cheap. A microcomputer can provide many of the facilities of a full-scale computer (now usually called a mainframe computer, or a minicomputer if it can be easily installed in an ordinary room) but at a fraction of the cost.

68008

Different microprocessors have different capabilities and can perform at different speeds. It is the Motorola 68008 microprocessor which is used in the Sinclair QL and this is a particularly powerful microprocessor, both in terms of the types of operation which it can perform and the size of numbers which it can manipulate internally. The size of numbers which can be manipulated by the microprocessor is often referred to as the number of bits that the microprocessor can work to. The bits talked of are actually a string of ones and zeros used by the machine to represent numbers. With more bits, larger numbers can be represented directly and time will not be lost pushing data into and out of memory whilst computing.

Early microprocessors were 4-bit machines and could thus represent numbers between zero and 15 (one less than two to the power four). When numbers are larger than this, the microprocessor must be programmed to perform a series of chained operations shunting data backwards and forwards internally and in memory. This takes time and uses up the available memory.

The home computer business has been founded on the use of 8-bit microprocessors. These can represent numbers between zero and 255 (one less than two to the power eight). For a typical microcomputer application, between four and six of these 8-bit numbers are combined together to represent realistic numbers and special algorithms are programmed in to perform number operations. The Motorola 68008 uses numbers internally which are 32-bits long and can thus be used directly to represent real numbers from zero to over four thousand million: obviously with this level of direct representation it is easier to construct short algorithms to perform operations on real numbers. The number of computing steps required to perform a given operation which is arithmetic can be reduced as the number of bits increases and hence the time taken to perform such operations comes down. Thus we would expect the Sinclair QL to be a fast machine, and such is the case.

However, although the 68008 is internally a 32-bit microprocessor, it is a special processor which communicates with the memory and external devices using eight bits only — the full 32-bit numbers are transferred in a series of four steps. This is done in order to reduce the cost of the rest of the computer but does mean that the full potential for high speed operation is not realised. This situation is quite common with the most recent crop of microcomputers. A common choice of microprocessor is the Intel 8088, which is a 16-bit processor which communicates with eight bits. The 8088 is used in the IBM microcomputer range and by many of the business machines which are designed as compatible with the IBMs.

I used the short BASIC program given below to give a comparison of the Sinclair Spectrum (8-bit processor), the QL (32-bit processor with 8-bit communications), and a contemporay 16-bit business machine (16-bit processor with 16-bit communications).

```
10 DIM O(1000)
20 FOR N=1 TO 1000
30 LET M=N
40 IF (M>500) THEN LET M=500
50 LET O(N)=INT(N*M*LOG(N)+.5):REM FO
R QL USE LOG10(N)
60 NEXT N
70 PRINT "LOOP FINISHED"
80 STOP
```

The Spectrum took 1 minute 33 seconds to run, and the business machine took 24 seconds to run. If you try running this program on the QL, you will find that it is comparable with the business machine. Re-write the program using SuperBASIC's short-form FOR command:

```
10 DIM O(1000)
20 FOR N=1 TO 500:O(N)=INT(N*N*LOG10(
N)+.5)
```

```
30 FOR N=501 TO 1000:O(N)=INT(N*500*L
OG10(N)+.5)
40 PRINT "FINISHED LOOP"
50 STOP
```

and you will find that the QL's run-time is considerably reduced. This simple bench test, involving a selection of functions, comparisons, and loops, shows how well the QL compares with other machines. Notice however that the bench test does not include any screen operations. The QL does not display things quickly, the sophisticated window facilities slow things down somewhat.

Other hardware

Sinclair has provided a second processor on the QL, the Intel 8049, which controls communications between the main processor, the keyboard, an RS232-C receiver, and a sound generator. The two processors are also supported by two specially-designed integrated circuits which handle the TV display, memory operations, and other hardware operations. All these hardware sophistications take some of the computing load off the main processor and thus leave it more time to run programs.

In addition to the powerful microprocessor, the Sinclair QL also has a number of other hardware features which are of interest. The size of the memory is 128 kilobytes (eight bits make one byte and a kilobyte is roughly a thousand bytes). This is considerably more memory than most other home computers, but is a common size in business machines. Memory is required for holding programs and for storing data. The larger the memory, the more complex the programs which can be stored or the more data which can be accessed by the program. An expansion of the memory is available which gives a total of five times the basic memory. This is a very large memory capability for a microcomputer and is provided principally to allow several programs to be held in memory and run at one time. It also has important implications for running certain types of simulation, as large data-bases and arrays of data can be built up.

Two microdrives are provided which can be used for program storage or, more importantly from the point of view of the running of simulations, allows programs to access data stored previously in a data-bank which can be very much larger than the memory directly available on the microcomputer. The microdrives are not as fast to use as other possible storage methods and only provide a limited storage potential, but they at least provide some capability.

A network facility is built into the QL, allowing it to communicate with other Sinclair computers at high speed. This facility can be very useful for simulations which represent a number of interacting systems, particularly if real-time operation is required. Such applications are beyond the scope of this book, however, and I shall resist the temptation to talk about the rather interesting simulations which can be set up using networks.

Display facilities on the QL are as good as, or slightly better than, most home computers and as good as most of the non-specialist business machines. The computer is designed to be compatible with ordinary UHF television receivers and this puts a limitation on the detail which can be displayed. There is also provision for display using a monitor, which provides improved capabilities. The display is not markedly superior to many other machines, however, and real-time simulations showing highly detailed scenes cannot be expected to be possible. Special-purpose microprocessor systems are available which can produce very high resolution displays for use in graphics applications. Such machines are very expensive, however, and rely on special hardware for the storage and display of screen information. Such machines are used for computer-aided design, network manipulation, and graphic representation. The QL is not outstanding in this area and its capabilities in these particular applications are limited.

Other hardware features are of lesser importance to the running of simulations: the quality of the keyboard is not usually a limiting feature; the types of noise which can be produced are important to speech synthesis and music production and the QL really needs additional external hardware to be used for these applications; input and output facilities to external hardware can be vital for some types of simulation but the QL offers no notable features in this respect and provides facilities as good as most other microcomputers.

Software

This brings us on to software. There are three points that I want to make: the internal operating software of a microcomputer is important to the applications which can be run, the software packages available for a machine may be useful for simulations, and the language used by the computer is also important.

Firstly the operating system. The way in which a computer operates determines how fast it runs, how much memory is actually available for programs and data, and what sort of operations can be attempted. The Sinclair QL is particularly strong in most respects, access to memory is good, use of various parts of the screen is easy, input and output using the RS232-C interface and the NETwork is made simple. The creation of machine code subroutines is not so easy, however, and really requires the use of additional software.

Software packages available with the QL are intended primarily for the small business and provide the usual basic business facilities of spreadsheet, word processor, and simple data-base. Also included with the

QL is a graphics package which can be used to set up quite good formats for the display of various forms of data. This package is less common on business microcomputers but is a boon for many applications: it is often of great use to be able to present the output of a simulation in graphical form, a well thought-out graph being worth many thousands of numbers. The data-base program is of less use. I tend to find general-purpose data-base programs are rather slow to use as part of a simulation and it is usually easier to create a special-purpose subroutine.

Finally I come to the programming language provided as standard on the microcomputer. For the QL, this is Sinclair SuperBASIC, which provides a versatile language for the programmer and is considerably in advance of that provided on most other microcomputers. However, although SuperBASIC is in advance of most other forms of BASIC and some of the other alternatives in being reasonably fast whilst simple to use, it is nowhere as fast in operation as the compiling languages available on mainframe and minicomputers, on which many of the simulations developed to date have been run.

Simulations on microcomputers

Simulations are greedy programs. They nearly always demand a lot of memory for the program or the data and also require a fast processing speed. This explains why microcomputers are not usually the ideal form of hardware on which to try to develop and run simulations, as microcomputers are usually rather slow and have very small memories.

Some generalisations may be in order here. Complex programs can be written in perhaps 10K of memory, but additional memory is required for data. This additional requirement could range from perhaps two to many tens of kilobytes. Thus, if a microcomputer has less than, say, 20K of memory readily available, then you can predict that problems with memory are going to be a prime concern with simulation programs.

Speed is required not just by real-time simulations but also by most other simulations in order to permit a large number of runs to be made and, of course, to stop the operator getting too bored. If a program takes much more than an hour to run, and this is not so unusual for many applications on microcomputers, then the development and testing stage can be something of a trial. There are various ways of speeding up computing on microcomputers, including using machine code and adopting special programming techniques. Such considerations are beyond the scope of this book and I can only say here that the faster the programming speed of the microcomputer the better.

Arithmetic accuracy can be a bit of a problem with microcomputers, particularly as simulations often rely on the precise calculation of probabilities. Generally, however, as long as the programmer is aware of the limitations of the software, it is relatively easy in most cases to compensate for any potential problems. Some microcomputers allow integer numbers, real numbers, or double precision numbers to be used, with the simpler data offering much higher processing speeds. The Sinclair QL offers a choice between integer and real numbers with a very high precision capability available for real numbers, higher in fact than most other machines operating in double precision mode.

Display capabilities affect output from the simulation, and a great variety of display formats are available on microcomputers, ranging from simple text displays in black and white only, to quite good resolution in colour. A point to note here, though, is that the more complex displays usually require more memory and this may well eat into the memory available for the programmer's use. Real-time simulations, and in particular video games, make the heaviest demand on display capabilities for most home-based microcomputers, whilst the display of complex designs and networks demand very high resolution capabilities for business use.

Facilities for microcomputers to communicate with external devices, such as measuring equipment, data loggers, other computers, and so on, are usually rather minimal on microcomputing systems. Some microcomputers are designed to be used in laboratory environments as well as at the office or at home, and suitable interfaces are provided. This tends to be an exception rather than the rule however. Sinclair computers have made provision for connection to external devices by providing a comprehensive expansion-connector which brings the main address, data, and control lines used by the main processor out of the machine. Such a means of connection is extremely versatile and caters for nearly all potential requirements.

QL benefits

Any computer has its own series of potential benefits and restrictions. At the start of this chapter I quickly described some of the important characteristics of the Sinclair QL and then went on to note some of the general comments which can be made about microcomputers as a class. In the next few paragraphs I will summarise these comments and pick out the most important factors, as I see them, when using the Sinclair QL for running simulations.

The Sinclair QL is a fast machine with a relatively large memory. The memory can be expanded to a level not previously available on any mass-production microcomputer. Both these features are vital for simulation programming. Bench tests of the speed of the QL compared to other popular home and business microcomputers indicate that it is indeed very fast. However this speed is still very much slower than that available on mini and mainframe computers.

Storage facilities on the QL, the microdrives, are adequate but by no means exceptional when considered technically. Of course, the low cost of the Sinclair QL derives largely from the use of microdrives rather than possible alternatives such as disk systems.

Software available on the machine can be useful for some simulations, although it is not designed with this use in mind. SuperBASIC is quite a good language which is very easy to use, but a language which could be compiled into a faster running format and one which included matrix operations would be better suited to simulation running.

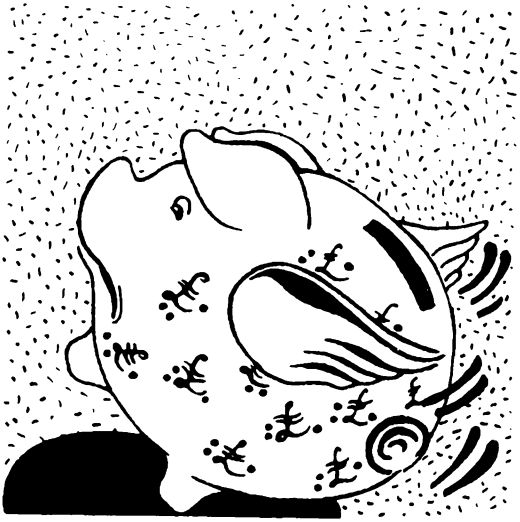
The special ability of the QL to run separate programs in parallel is of limited use for most simulations. It is possible to run separate parts of a simulation at the same time, or to treat separate programs as if they were being run on different machines, but the genuine applications are few.

Access to external devices is potentially very good, making practical use of the computer with other equipment a real possibility. The networking facility may be of considerable interest to some users of simulations, particularly in the games field, but does not add much to the general capabilities of the machine.

CHAPTER 6

Laughing All the Way to the Bank

Money models



"How much have I got?..."

This is the first of the practical chapters of this book. From here on, I discuss each of the major application areas in turn and present some simple practical examples of simulations which might be developed. It is not my intention to present long listings of general-purpose simulations. Rather, I point out the use of various techniques and convert these into rather obvious programs. At least I hope that they are obvious — if they are not, then I have fallen down on the job.

Background

Financial modelling is the first area that I consider and one in which most users show interest at some time or another. Although expressed in a variety of ways, most businesses and individuals want the same basic questions answered about their finances.

The most popular question is: 'How much money have I got now?' Next comes: 'How much money will I have tomorrow?' These two questions tend to form the basis for the vast majority of financial models (and also keep a lot of accountants very busy), although companies will talk about assets rather than money.

There are, however, a few other questions which can be asked, especially by those seeking to become healthy, wealthy, and wise. These typically are: 'What happened to the money that I had yesterday?', 'How can I make the most money tomorrow from that which I have today?' and 'How much money has everybody else got, and what are they going to do with it?' It is these last few questions that often need some form of simulation to be answered properly. The investigation of what will happen if alternative possible courses of action are taken is, in principle, a very straightforward problem. In fact, financial simulations often grow into very complex programs when all factors are allowed for.

Some examples of real applications which are based on the questions noted above are:

How much have I got?	— Stock maintenance program
How much tomorrow?	— Profit and loss projections
What happened to it?	— Personal accounts
How can I make more?	— Investment modelling
What about the others?	— Stock Market models

The last two tend to require not only a very good understanding of the system which is being modelled but also access to a very large amount of reliable data. Think carefully before you decide to take the Stock Market by storm with your Sinclair QL — some very clever people using some very expensive hardware are already operating there and have found that their simulations are mainly of use in aiding them to make their own decisions;

the programs cannot usually be used very reliably for unaided prediction in this complex environment.

Example

As my example in this chapter, I am going to look at the second question, predicting future finances based on current assets and past performance. The case that I have dug out of my case-book concerns the prediction of monies held in various accounts belonging to a working person (not me, let me hasten to add, but a friend of mine who wanted to visit a computer exhibition in America), with allowance made for income from a fixed source (the main employment) and from a second source, a part-time business. Outgoings are also allowed for and broken down into major types of expense. The simulation part of the program comes in when I define the rules for predicting future figures from known historical data. The basic approach that I outline here can equally well be used for the analysis of company performance or the monitoring of any form of funds, such as club assets.

Simulation development

This is one case which lends itself to development on a spreadsheet and thus provides me with a chance to describe the potential use of the software provided with the QL. But I am jumping the gun a little, am I not?

As I described in Chapter 4, we must first define the simulation requirements. I shall of necessity keep this bit brief: when you come to developing real simulations you will have to be more thorough. The system which we are going to simulate consists of the money possessed by a single person, Penny Pound. The money is kept in a current account at her local bank, National Vault Limited, a deposit account in a building society, Humpty Securities, and as cash. The cash is almost incidental as Penny keeps an average of about £10 in cash at all times. There is a little more to the system than just the money, however, as we must also look at the ways in which money enters and leaves the accounts and moves from one place to another.

The month is May, Penny has just had a pay rise and each month (calendar month rather than the often used four-week month) Penny is paid £600, after tax and other deductions have been made, which is paid directly into her bank. As a means of building up her savings, Penny has arranged a standing order of £60 pounds a month from her bank to her building society account. All major bills, such as electricity and phone bills, are paid by cheque from the current account at the bank. Day-to-day bills are paid in cash, which Penny withdraws from the bank as she needs it. Penny has, very conveniently, kept all her old cheque stubs and can thus say what she has paid, and to whom, over the last 16 months. I have made a note of the most important figures below.

Is that all that we need to know? We must also note the important changes in the past. Penny has just had a rise in pay — until last month she was only getting £550 a month. She has also been doing some freelance work and her income from this is given below: she expects this income (which is quoted after tax, etc.) to increase at about 10 per cent every month for the next few months. Every year she takes out most of her savings from the building society and goes on holiday. She wants to know what is likely to happen to her money over the next six months and whether or not she can go to an exhibition, costing a minimum of £950, in three months' time at Computaland in the USA.

Now we are getting a better idea of the simulation requirement and the system to be simulated. The requirement is a prediction for each month for the next six months, with a special test for three months' time. The predictions are to be based on monthly data which covers the past 16 months. There are a number of techniques for predicting values based on historical data. We could plot the historical data and visually pick out a trend. We could 'fit' a simple mathematical formula to the data and use this. These techniques, and others, could all be attempted, but I'm going for something a little simpler.

The value which we are going to predict for a given quantity is to be based on the value for that quantity for 12 months ago, but modified by an estimate of inflation. This estimate of inflation is based on a general comparison of the previous month's outgoings with those of twelve months before. This may not sound too simple, but will become much clearer when we get to setting up the spreadsheet.

Of course, I have already chosen the spreadsheet as the basic means of setting up the simulation and have thus avoided having to select between various options. Use of the spreadsheet gives a simple means of storing and manipulating data. The predictions are quite straightforward, not involving any decisions or alternative paths. The output requirements are also very simple, being a minimum of a single figure (total money) for each of six months plus one special figure for the middle of the six month period. In fact, we shall actually be taking more interest than this in the output, but I want to set up the simulation first.

I must assume that you know the basics of using a spreadsheet program such as Abacus. If you don't, then have a quick flip through the manual now, or take a glance at one of the many descriptions available in books on 'making the most of your micro', or ask someone else to describe the basics to you.

The following data has been supplied by Penny Pound and should be set up in columns A to G of a blank spreadsheet. Use the titles given as column headings in row one and the month numbers as references in column A.

Figures give expenditure for a full month.

	A	B	C	D	E	F	G
1		RENT	GASELEC	CAR	HOLIDAY	PHONE	MISC
2	FEB,A	148.00	0.00	58.00	200.00	0.00	261.00
3	MAR,A	148.00	51.00	0.00	0.00	0.0	312.00
4	APR,A	148.00	55.00	22.00	0.00	21.00	264.00
5	MAY,A	148.00	0.00	0.00	0.00	0.00	230.00
6	JUN,A	148.00	42.00	0.00	0.00	0.00	265.00
7	JUL,A	148.00	40.00	47.00	0.00	23.00	278.00
8	AUG,A	148.00	0.00	95.00	0.00	0.00	324.00
9	SEP,A	163.00	34.00	85.00	0.00	0.00	330.00
10	OCT,A	163.00	24.00	0.00	0.00	23.00	211.00
11	NOV,A	163.00	0.00	0.00	550.00	0.00	320.00
12	DEC,A	163.00	31.00	0.00	0.00	0.00	278.00
13	JAN,B	163.00	22.00	44.00	0.00	25.00	256.00
14	FEB,B	163.00	0.00	0.00	0.00	0.00	300.00
15	MAR,B	163.00	56.00	0.00	0.00	0.00	289.00
16	APR,B	163.00	61.00	0.00	0.00	35.00	268.00
17	MAY,B	163.00	0.00	12.00	0.00	0.00	268.00

Note: Abacus can be set up to display from 40 to 80 characters across the screen. I have not attempted to give a true representation of the Abacus screen display. Depending on the mode that you use you will be able to display a greater or lesser portion of the spreadsheet.

I suggest that before you start to enter data you set the default cell display to two decimal places — press F3 U(nits) D(efault) M(onetary) ENTER — and switch off the auto-calculate — press F3 D(esign) A(uto) ENTER.

In addition to expenditure, we must also set down Penny's income and the money which remains in her bank and building society accounts. We shall ignore cash; Penny keeps around £10 as cash, but this is a relatively constant figure and can be excluded from this system. The relevant figures, to be entered in columns H to K of the spreadsheet, are as follows:

	A	H	I	J	K
1		INCOME, A	INCOME, B	BLDSOC	BANK
2	FEB,A	550.00	0.00	10.00	71.00
3	MAR,A	550.00	0.00	70.00	43.00
4	APR,A	550.00	0.00	130.00	70.00
5	MAY,A	550.00	0.00	190.00	84.00

```
 6| JUN,A  550.00   0.00  250.00  161.00
 7| JUL,A  550.00   0.00  310.00  183.00
 8| AUG,A  550.00   0.00  370.00   91.00
 9| SEP,A  550.00   0.00  430.00    8.00
10| OCT,A  550.00   0.00  490.00    5.00
11| NOV,A  550.00   0.00    0.00  -35.00
12| DEC,A  550.00   0.00   60.00   14.00
13| JAN,B  550.00   0.00  120.00   54.00
14| FEB,B  550.00  35.00  180.00   25.00
15| MAR,B  550.00  50.00  240.00  113.00
16| APR,B  550.00  56.00  300.00  172.00
17| MAY,B  600.00  64.00  360.00  249.00
```

So far, we have set up the data to be used in the simulation, but nothing more. The next thing we need to do is to set up any modifications that may be required to the raw data. (In case you were wondering, raw data is data that has not been modified or analysed in any way. For some reason, which you may be able to work out for yourself, it is unheard of for people to talk of modified data as cooked data, it is more usual to refer to it as processed data.) One of the things that we want to know is how much money Penny actually has at the end of each month. This quantity is found simply by adding the two account figures. We also need to estimate the inflation in spending over the previous twelve months, based on actual expenditure, which in turn requires that we find out, month by month, just what the expenditure has been. We thus have three additional columns to add to the spreadsheet: spending, inflation, and money. These will be set up in columns L, M, and N respectively.

Spend is a simple sum of money out in any month. Move the cursor to L2 on the spreadsheet and, using the ability of Abacus to refer to other cells on the spreadsheet using row and column labels, enter the formula;

RENT.FEB,A + GASELEC.FEB,A + CAR,A + PHONE.FEB,A + MISC.FEB,A

Note that I have left the holiday expenditure out of the sum (which is why I did not use the SUM( ) function). This is because we are going to use spend to generate an inflation figure and the large expenditure on infrequent holidays would play havoc with this figure.

The spend formula given above must be E(choed) down column L to row 17, in order to give the spend for each month.

Inflation is to be set in column M, but it can only be set where at least twelve months of spend figures are available. We shall leave this blank for the moment. Column N is to be the money column. The formula to enter in cell N2 is:

BANK.FEB,A + BLDSOC.FEB,A

Again this must be E(choed) down the column. Enter X(ecute) to execute the formulae. Columns L, M, and N should now look as follows;

```
  |        |   L   |   M    |   N   |
 1|          SPEND  INFLATN  MONEY
 2|  FEB,A  467.00             81.00
 3|  MAR,A  467.00            113.00
 4|  APR,A  467.00            200.00
 5|  MAY,A  378.00            274.00
 6|  JUN,A  455.00            411.00
 7|  JUL,A  536.00            493.00
 8|  AUG,A  567.00            461.00
 9|  SEP,A  612.00            438.00
10|  OCT,A  421.00            495.00
11|  NOV,A  483.00            -35.00
12|  DEC,A  472.00             74.00
13|  JAN,B  510.00            174.00
14|  FEB,B  463.00            205.00
15|  MAR,B  508.00            353.00
16|  APR,B  527.00            472.00
17|  MAY,B  443.00            609.00
```

Now we can set up the simulation part of the spreadsheet. This means setting up formulae which will predict future performance of the system from stored data.

Extend column A downwards to give months JUN,B ; JUL,B ; AUG,B ; SEP,B ; OCT,B ; NOV,B. This will give predictions for the next six months (remember, when I first set up this simulation, the current month was May). Now we must enter the formulae, column by column, to fill rows 18 to 23.

First, rent. Enter;

RENT.MAY,B

into cell B18 and E(cho) down the column. This will keep the rent at a fixed level.

The general formula we shall use for predicting unknown expenditure for each month is;

Expenditure = Expenditure from 12 months before * inflation calculated to the previous month

For example, the precise formula to enter into cell C18 and E(cho) into all cells to C23 is:

```
GASELEC.JUN,A*INFLATN.MAY,B
```

Similar formulae must be set up for columns C, D, F, and G.

As for rent, the predicted values in columns E and H will remain constant. Use the formula;

```
HOLIDAY.MAY,B
```

in cell E18 and E(cho) down to E23. Enter:

```
INCOME,A.MAY,B
```

in cell H18 and E(cho) down to H23.

Income B is predicted by Penny to rise at 10 per cent per month. Hence the formula to enter at I18 is:

```
INCOME,B.MAY,B*1.1
```

This also must be E(choed) down the column.

Money in the building society account is added to (from the bank account) by £60 per month, and is only removed if required for holidays. Enter the following formula in cell J18 and E(cho) down to J23:

```
BLDSOC.MAY,B + 60
```

The ordinary bank account is a little more complex. Money in is the income figure from columns H and I. £60 goes to the building society. Miscellaneous expenses are paid for, in general, in cash which Penny withdraws as required from the account; hence the figure given in column G is subtracted directly from the bank account. Other major outgoings are paid by cheque. One point about cheques is that they take time to process. In order to give you an idea of how to deal with this, I am assuming that cheques written in one month are not actually cleared until the following month. This is an exaggeration, of course, but is a simple rule to apply and will do for this simulation. Perhaps you might like to think of other approaches, for instance to back up the assumption that it takes an average of six days to post and process a cheque.

Our formula for the bank account thus comes out as;

```
BANK.MAY,B + INCOME,A,JUN,B + INCOME,B,JUN,B-60-MISC.JUN,
B-RENT,MAY,B-GASELEC.MAY,B-CAR.MAY,B-PHONE.MAY,B
```

This should be entered into K18 and E(choed) to K23.

Columns L and N already have formulae set up, these merely need extending to row 23.

The last column to fill is the inflation column. This must be calculated from MAY,B to NOV,B. The formula is simple:

```
SPEND.MAY,B/SPEND.MAY,A
```

is entered at M17 and E(choed) down to M23. Again, this is a gross simplification and there are other approaches which could be adopted which would be more realistic and less prone to error, but much more complex to implement. The major problem with the method I have adopted is that the inflation figure can be distorted by any odd highs or lows in the data. Also, recent trends in data are not identified using this approach. For example, the most recent phone bill was unusually high, due, I suspect, to Penny's part-time job. Our method of prediction will not carry this high rate forward.

Results

The simulation is more or less complete now. If you X(ecute) you should get the following;

```
 |   A    |   B    |    C    |   D    |    E    |    F    |   G    |
1|           RENT   GASELEC    CAR    HOLIDAY   PHONE     MISC
2|  JUN,B  163.00    49.22     0.00    0.00      0.00    310.57
3|  JUL,B  163.00    45.96    54.00    0.00     26.43    319.42
4|  AUG,B  163.00     0.00   107.90    0.00      0.00    368.01
5|  SEP,B  163.00    38.31    95.78    0.00      0.00    371.86
6|  OCT,B  163.00    26.23     0.00    0.00     25.14    230.63
7|  NOV,B  163.00     0.00     0.00    0.00      0.00    338.25
```

```
 |   A    |    H     |    I     |   J    |   K    |
1|          INCOME A  INCOME B   BLDSOC    BANK
2|  JUN,B   600.00     70.40    420.00   373.83
3|  JUL,B   600.00     77.44    480.00   459.63
4|  AUG,B   600.00     85.18    540.00   427.41
5|  SEP,B   600.00     93.71    600.00   418.36
6|  OCT,B   600.00    103.07    660.00   533.70
7|  NOV,B   600.00    113.38    720.00   634.46
```

```
 |   A   |  L  |  M  |  N  |
1|        SPEND INFLATN MONEY
2| JUN,B  522.79  1.15   793.83
3| JUL,B  608.81  1.14   939.63
4| AUG,B  638.92  1.13   967.41
5| SEP,B  668.95  1.09  1018.36
6| OCT,B  445.01  1.06  1193.70
7| NOV,B  501.25  1.04  1354.46
```

If you do not get this result, then check through your entries from the start. Make sure that you have the spreadsheet set to calculate in row order (F3 D(esign) C ENTER).

Now we have our answer, Penny can just about afford to go to the USA in August — or can she? We have modelled the system and predicted forward for six months in steps of one month. The indications are that Penny will have over £900 in August and will not suffer drastically in the following months. However, in our haste to set up the simulation we have overlooked a few vital points. These are as follows:

1) Penny's rent is due to go up in September, probably by 10 per cent. Enter RENT. AUG,B*1.1 in cell B21.

2) We have not actually allowed for the cost of the trip in August. Set cell E20 to say 950. Modify the formulae in cells J20 and K20 to pay for the trip, £540 from the building society account and £410 from the bank.

3) Although Penny can take paid leave from her full-time job, she will lose her potential earnings from her part-time job, and will also lose some of her increased earnings in the following months. In fact, by talking to Penny I found out that she would lose all her money in August (enter zero in cell I20) and her earnings in September would be 10 per cent up on July (enter INCOME B,JUL,B*1.1 in cell I21).

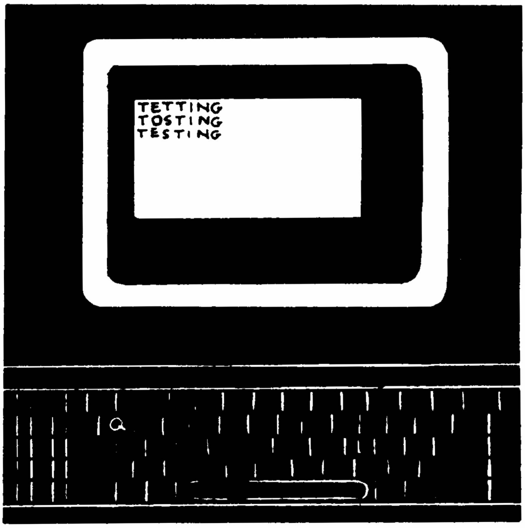
Now X(ecute) again. Do you still think Penny can afford to go to the USA? If she does, she will go £85 into the red in her bank account.

I'm not going to tell you what Penny finally decided to do (all the best mystery stories leave the ending open) but you see how careful you must be in setting up even a simple simulation and in making sure that all relevant information is allowed for. You must constantly be checking the workings of the simulation and the assumptions on which it is based. If at any point the simulation gives you a marginal result, ie one which is very close to a decision-making value, then be especially careful.

CHAPTER 7

Testing Testing Testing

Product development



"... simulation is obtained by a process of elimination..."

My second practical chapter dives into the world of hardware development.

Background

The potential roles for simulations in support of hardware development are so many and so varied that there is little that I can meaningfully add to my general comments in Chapters 2 and 3.

Most of my own experience has been concerned with the evaluation of different hardware solutions to meet a given need. This involves creating a simulation of the environment in which the hardware is to be used and then using this to compare the different performances of possible designs using the simulation. This form of simulation is usually developed at the initial planning stage of a hardware programme. In parallel, a financial assessment of the hardware designs will be made, possibly based on financial simulations of the development and production operations and the potential commercial market, which will lead to the selection of a preferred design.

Development of hardware typically follows the pattern: research, design, development, production, and update. Comparison of alternative possible designs is generally completed during the research stage. The design stage is based on the evolution of a detailed technical evaluation and description of the chosen concept, often based on the use of computer-aided design programs (CAD). The design process is based on the use of theoretical calculations and the application of old, proven technologies into new products. Use of computer assistance during the past few years has lead to a considerable reduction in the time taken to achieve a detailed design, a more thorough evaluation of the design prior to the manufacture of prototypes, and savings in product weight without any reduction in strength.

During the development stage, the designs are converted into hardware, tested, improved, and evaluated against the original requirement. Reliability assessment and prediction will identify problem areas and indicate whether or not the product will be fully ready for production when planned. Especially important during development is the setting up of the production facility. This may mean just making adjustments to existing machines, or creating new moulds and dies for casting work, or it may mean building a new factory, ordering and buying new machinery, employing and training staff, and so on. Obviously, it is vital to get this planning, which may be extremely complex, done right, or large amounts of money can be lost. Extensive use is made of project-planning computer programs, such as critical-path analysis, and these programs are becoming more and more sophisticated. At the heart of most of these is a simple simulation of the sequencing of the project plan.

Less use is made of simulations once production is under way: Chapter 8 looks at simulations for management information.

Updating the design may take place some time after the product has first entered production. This may be to deal with some defect appearing after production is under way, to reduce production costs, to improve performance, or simply to make the product look more modern. The updating process is, in theory, very similar to the first design phases, with research, design, development, and production. Computing requirements are also, in principle, the same.

In summary, greater and greater emphasis is being placed on the use of computers during development and many of these uses involve the simulation of either the hardware under development, the system in which the hardware is to be used, or the system which is to be used to make the hardware. I titled this chapter 'Testing Testing Testing', because hardware development has traditionally required considerable effort to be put into prototype testing. The emphasis is gradually changing—perhaps I should have called this chapter 'Evaluating Evaluating Evaluating'.

Example

There are many examples that I could have chosen, most of these however would have been too lengthy to include in a book of this size. The example that I have chosen takes me back to my early simulation days. When I was first learning to use computers I was also keenly interested in motor racing, particularly go-kart racing and motor-cycle racing. Engine capacity ranged from 100 to 500 cc (cubic centimeters), and the engines, usually two strokes, were tuned to give maximum power outputs. Don't worry if you do not know much about engines, I shall shortly explain all you need to know.

One of the interesting things about these small capacity engines is that, once you start to look at them in detail, they turn out to be very simple (often to keep weight to a minimum) and rather under-developed. Advances in design have been rapid, but mainly because a large number of people have been trying out different ideas; some are bound to work and these are rapidly copied by all the manufacturers. It struck me that I could use my A level physics knowledge to simulate part of the engine and thus 'try out' different ideas without having to spend huge amounts of time and money testing hardware. (You may like to note that the Japanese motorcycle industry adopted a similar approach with some success.) I was particularly interested in simulating the exhaust system (a vital component for racing engines — still under-developed) and the ignition system.

My example for this chapter is based on my early efforts to develop a simple electronic ignition system for a 100 cc racing engine.

Simulation development

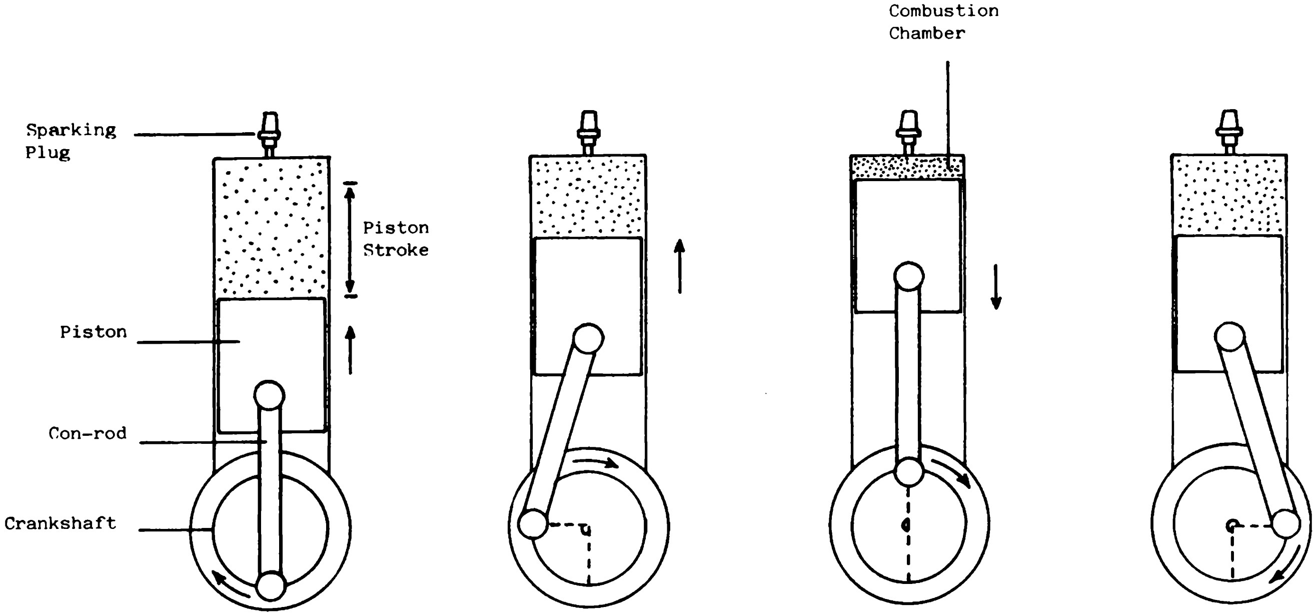
Piston engines work by using a piston moving in a cylinder to compress a mixture of air and fuel (in this case petrol); burning the mixture and thus using the chemical energy present in the fuel to heat the air/fuel mixture which in turn increases the pressure of the mixture; and using this high pressure to push the piston back down the cylinder. It is the pressure acting on the piston which does the work and provides power. Maximising the power output of the engine depends principally on burning the air/fuel mixture in such a way that the pressure achieved as the piston moves down is controlled to produce useful work, whilst not overheating or overstressing the piston and other engine parts.

A sparking plug (simply a screw-in plug which produces a high voltage spark in the combustion chamber — where the air/fuel mixture is compressed and burnt) is used to ignite the fuel, with the timing of the spark set by the ignition system. If the spark is too early, the pressure rise may force the piston back down the cylinder or may overheat the engine: if too late, then some of the work potential of the burnt air/fuel mixture will be lost. The fuel takes time to ignite (there is a timing delay, between the spark and the mixture starting to burn, known as the ignition delay) and the fuel takes time to burn, thus maximum pressure occurs some time after the spark. As the engine speeds up, the spark must be set to occur earlier and earlier to ensure that the peak pressure comes just after the piston has reached its highest point in the cylinder and is on its downward, or power stroke.

In its simplest form, and remember that racing engines are often very simple, the ignition should automatically provide the correct ignition advance as the engine accelerates through its speed range to give maximum power. My simulation is of the compression, burning, and power-producing expansion of the air/fuel mixture. The desired ignition advance is found by gradually changing the simulated ignition setpoint until a maximum power output is found. This process is repeated for various engine speeds until a full picture is built up of the required ignition characteristics.

I have defined the basic simulation requirement. In order to make the simulation suitably short for inclusion in this book, I have kept the simulation as simple as possible, as explained below. The technique to be adopted for the simulation is obtained by a process of elimination. I cannot use a spreadsheet program as I did in the last chapter, as the detail required would rapidly use up the limited space available on a spreadsheet. Also, the resulting simulation would not be versatile enough to permit future development of the simulation, and the finding of optimum ignition advance would probably have to be done by hand. Thus a special program must be written.

There is no decision-making required within the simulation, as the air/fuel mixture can be taken to follow known laws of fluid dynamics. As the mixture is compressed, burned, and expanded we can uniquely define



Bottom of piston cycle.
Angle of crank
= 0 radians (0°).

Angle of crank
= 1.57 radians
(90°).

Top of piston cycle.
Angle of crank
= 3.14 radians
(180°).

Angle of crank
= 4.71 radians
(270°).

Figure 7.1: Crankshaft Angle and Piston Position.

its state at all times — there is no question of assigning probabilities to alternative possible states. Thus the techniques available for handling probabilities, such as Monte Carlo, Markov chain, deterministic and special purpose, cannot be used. We could possibly use a model which splits up the engine cycle into a small number of phases — eg a compression phase, a burning phase, an expansion phase, etc. — with calculations giving the engine state at the end of each phase depending on the conditions at the start of each phase. This is essentially the approach adopted for hand calculations but is not sufficiently accurate for our needs.

The only approach which I think merits use is to base the simulation on a model of the air/fuel mixture and to step through the engine cycle from start to finish. Here we do have a choice, however; we can step through the cycle using a constant time-step or we can use the rotation of the crankshaft. (See **Figure 7. 1** for the relationship between the piston and the crankshaft. The crankshaft is used to convert the linear motion of the piston into rotational motion required for driving wheels.) It is much simpler to base the model on crankshaft position than on time, hence I have chosen to base the model steps on the rotation of the crankshaft.

As I mentioned above, I have had to use a very much simplified model of an engine, and have had to adopt a very simple model of the behaviour of the air/fuel mixture. Those of you who are familiar with engines and fluid thermodynamics will probably realise that the representation, as it stands, can only be used to give a very rough guide to engine performance; the model can be extended and improved quite easily, however.

The model steps through a single engine cycle consisting of compression of the air/fuel mixture, ignition and burning of the mixture, and expansion of the burnt mixture. Other parts of a real engine cycle — the burnt mixture and bringing in a fresh charge of air and fuel — are ignored. A total is calculated for the work output of the engine, which is obtained by continuously calculating the force acting on the piston, multiplied by the distance moved by the piston from one crankshaft position to the next. As the air/fuel mixture is being compressed, this work term will be negative, ie the piston must supply work to compress the mixture. As the hot burnt gas expands, the work is positive; work is done on the piston. The total work available at the end of the cycle indicates, when multiplied by the engine speed, the power produced by the engine.

I am not going to explain the program development in any great detail. **Figure 7. 2** gives a flowchart for the program (this is the next item to produce once the simulation technique has been selected) and below I give a brief description of each of the program routines. Use these and work through the program listing to see how I have developed the program: you should have no real difficulty. Program variables are described at the end of the listing.

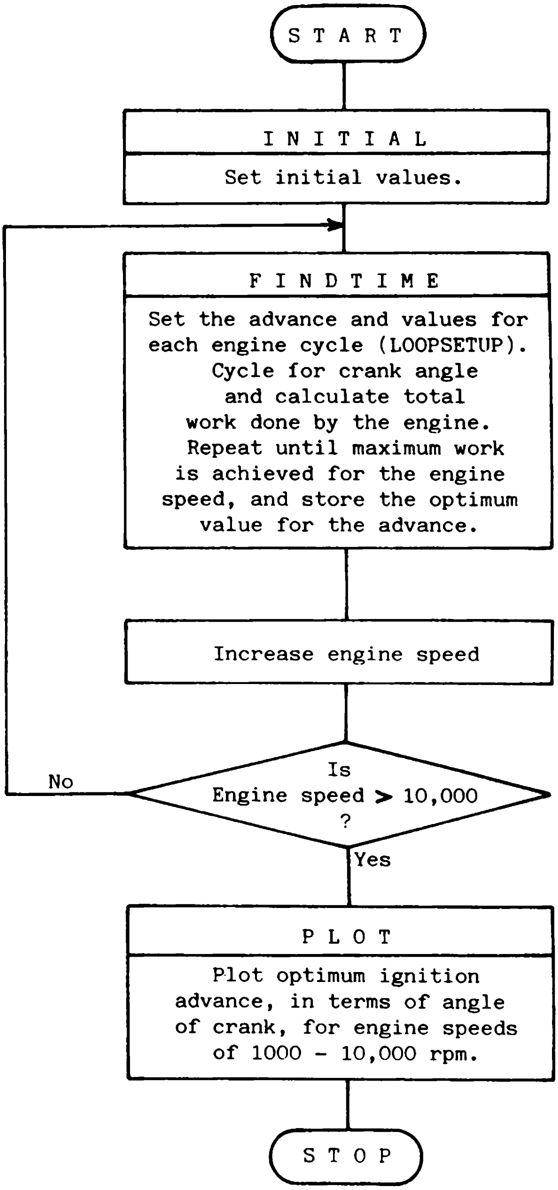

START
INITIAL
Set initial values.
FINDTIME
Set the advance and values for each engine cycle (LOOPSETUP). Cycle for crank angle and calculate total work done by the engine. Repeat until maximum work is achieved for the engine speed, and store the optimum value for the advance.
Increase engine speed
Is Engine speed > 10,000 ?
No
Yes
PLOT
Plot optimum ignition advance, in terms of angle of crank, for engine speeds of 1000 - 10,000 rpm.
STOP

Figure 7.2: Flowchart for the Program 'Ignition'.

```
 90 :ADVANCE
    :(DEGREES)
    :
    :
    :
 60 :
    :
    :
    :                            ******
    :                      ******
 30 :                   ***
    :                ***
    :          ******
    :       ***
    :    ***
 +0 :
---------------------------------------
 -0 :
    :
-15 :
    :REVS 2000  4000  6000  8000  10000
```

Figure 7.3: Output Produced by Program 'Ignition'.

Results

The output from the program **(Figure 7.3)** is very simple. It gives the desired ignition advance, in terms of the crankshaft angle before the piston reaches the top of the cylinder, for engine speeds from 1,000 to 10,000 revolutions per minute.

Basically this is all I need in order to begin to design a full electronic ignition system. In practice, however, I would want to improve the simulation considerably before using the results, and I would want to start investigating factors such as what happens when the air inlet temperature varies (hot and cold weather), what happens if the engine compression ratio is changed, what happens if the ratio of fuel to air is changed, and so on, before deciding what I want from the ignition system.

Program description

Figure 7. 2 presents an overall flowchart for the program 'Ignition'.

The routines used in the program have the following functions:

PROCedure INITIAL: Sets the initial values of program variables.

PROCedure FINDTIME: Searches for the optimum ignition advance for each engine speed. This is done by setting the advance, rotating the crankshaft and working out the total work output, and then modifying the ignition advance until an optimum is found. At first the ignition setting is reduced. If a drop in work is recorded then the ignition is progressively advanced and the optimum point is taken as the point just before the fall in work is indicated.

PROCedure LOOPSETUP: Sets variables for each change of advance, setting the record of old values from the current values, and increasing or decreasing the advance as required.

FuNction newvol: Calculates the volume of air/fuel mixture for the next position of the crankshaft.

FuNction pressure: Calculates engine internal pressure at a given crankshaft angle.

FuNction burnheat: Calculates the temperature rise in one cycle due to the burning of fuel.

FuNction volindex: Calculates the temperature change index in one cycle due to a change in volume of the air/fuel mixture.

FuNction rate: Calculates the rate of burning of the fuel as a function of the air/fuel density (density = 1/volume).

FuNction burnrate: Calculates the actual energy supplied by burning the fuel in a single program cycle.

FuNction loopwork: Calculates the work done in the engine during one program cycle.

Program listing

```
100 REMark IGNITION
110 MODE 256
120 CSIZE 2,0
130 INITIAL
140 REPeat SPEEDLOOP
150  FINDTIME
160  OADV(INT(REVS/1000+.5))=OLDADV
170  REVS=REVS+1000
180  IF REVS>10000 THEN EXIT SPEEDLOO
P
190 END REPeat SPEEDLOOP
```

```
200 PLOT
210 STOP

220 DEFine PROCedure INITIAL
230  DIM OADV(10)
240  CAL=500000
250  CONSTA=.1075
260  CONSTB=4.135E-5
270  PINLET=101300
280  TINLET=273
290  VMOLE=2.24E-2
300  R=8.314
310  KENG=35
320  GAMMA=1.3
330  DELAY=3E-4
340  KRATE=2.5E-3
350  REVS=1000
360  STROKE=5.38E-2
370  BORE=4.82E-2
380  CAP=STROKE*PI*BORE*BORE/4
390  ROD=.1
400  VCOMB=9.5E-6
410  OADV(0)=2
420  WORK=0
430 END DEFine

440 DEFine PROCedure FINDTIME
450  LOCal a,b
460  ADV=OADV(INT(REVS/1000-.5))
470  INDICATE=0
480  REPeat TIMING
490   LOOPSETUP
500   CLS:PRINT "OPTIMISING IGNITION
ADVANCE":PRINT "REVS = ";REVS;" ADVAN
CE = ";ADV;" DEGREES"
510   REPeat WORKLOOP
520    LOOPANG=5
530    SELect CRANK=140 TO 218:LOOPAN
G=2
540    CRANK=CRANK+LOOPANG
550    IF CRANK>360 THEN EXIT WORKLOO
P
560    VOL0=VOL1
570    TEMP0=TEMP1
580    PRESS0=PRESS1
590    VOL1=newvol
600    TEMP1=(TEMP0+burnheat)*volinde
x
610    PRESS1=pressure
620    WORK=WORK+loopwork
630    IF CRANK<(BADV+DELAY*REVS*360/
60) THEN
640     BURN=0
```

```
650    ELSE
660     BURN=burnrate
670    END IF
680    RA=RA-BURN
690   END REPeat WORKLOOP
700   b=INDICATE
710   SELect ON INDICATE
720    ON INDICATE=0
730     b=1
740    ON INDICATE=1
750     IF WORK>OLDWORK THEN
760      b=2
770     ELSE
780      b=3
790     END IF
800    ON INDICATE=2 TO 3
810     IF WORK<OLDWORK THEN RETurn
820   END SELect
830   INDICATE=b
840  END REPeat TIMING
850 END DEFine

860 DEFine PROCedure LOOPSETUP
870  CRANK=0
880  BURN=0
890  VOL1=CAP+VCOMB
900  TEMP1=TINLET
910  PRESS1=PINLET
920  OLDWORK=WORK
930  WORK=0
940  OLDADV=ADV
950  DADV=5
960  IF INDICATE=3 THEN
970   SELect ADV=-40 TO 38:DADV=2
980  ELSE
990   SELect ADV=-38 TO 40:DADV=2
1000  END IF
1010  SELect ON INDICATE
1020   ON INDICATE=1 TO 2
1030    ADV=ADV-DADV
1040   ON INDICATE=3
1050    ADV=ADV+DADV
1060  END SELect
1070  BADV=180-ADV
1080  RA=CAL*CAP
1090 END DEFine

1100 DEFine FuNction newvol
1110  LOCal a,b
1120  b=CRANK*2*PI/360
1130  a=SQRT(ROD*ROD-(STROKE/2*SIN(b)
)^2)-STROKE/2*COS(b)-(ROD-STROKE/2)
1140 RETurn (CAP+VCOMB-(PI*BORE*BORE/
4)*a)
1150 END DEFine
1160 DEFine FuNction pressure
1170  LOCal v
1180  v=VOL1*VMOLE/(CAP+VCOMB)
1190  RETurn R*TEMP1/(v-CONSTB)-CONST
A/(v*v)
1200 END DEFine
1210 DEFine FuNction burnheat
1220  RETurn KENG*BURN
1230 END DEFine
1240 DEFine FuNction volindex
1250  RETurn (VOL0/VOL1)^(GAMMA-1)
1260 END DEFine
1270 DEFine FuNction rate
1280  RETurn KRATE*CAL*CAP*LOOPANG/(R
EVS*VOL1)
1290 END DEFine
1300 DEFine FuNction burnrate
1310  IF RA=0 THEN RETurn 0
1320  IF RA>rate THEN RETurn rate
1330  RETurn RA
1340 END DEFine
1350 DEFine FuNction loopwork
1360  RETurn (PRESS0+PRESS1)/2*(VOL1-
VOL0)
1370 END DEFine
1380 DEFine PROCedure PLOT
1390  LOCal a,b,c,n,q,p$
1400  CLS
1410  PRINT " 90:ADVANCE"
1420  PRINT "   :(DEGREES)"
1430  FOR n=0 TO 16
1440   SELect ON n
1450    ON n=3
1460     PRINT " 60";
1470    ON n=8
1480     PRINT " 30";
1490    ON n=13
1500     UNDER 1:PRINT "  +0";
1510    ON n=14
1520     PRINT " -0";
1530    ON n=16
1540     PRINT "-15";
1550    ON n=REMAINDER
1560     PRINT "   ";
1570   END SELect
1580   PRINT ":   ";
1590   FOR q=1 TO 10
```

```
1600      a=72-6*n
1610      b=a+5
1620      c=OADV(q)+.5
1630      SELect c=a TO b:PRINT "***";
1640      SELect c=-25 TO a-1:PRINT "
";
1650      SELect c=b+1 TO 73:PRINT "
";
1660     END FOR q
1670     SELect n=13:UNDER 0
1680     PRINT
1690    END FOR n
1700    PRINT "   :REVS 2000  4000  600
0  8000 10000"
1710 END DEFine
1720 REMark VARIABLES LIST
1730 REMark a,b=TEMPORARY VARIABLES
1740 REMark ADV=IGNITION TIMING ANGLE
BEFORE TOP OF PISTON CYCLE
1750 REMark BADV=IGNITION TIMING ANGL
E AFTER BOTTOM OF PISTON CYCLE
1760 REMark BORE=ENGINE BORE
1770 REMark BURN=RATE OF INCREASE IN
GAS TEMP BASED ON RATE OF FUEL BURN
1780 REMark CAL=CALORIFIC DENSITY OF
AIR/FUEL MIXTURE AT INLET CONDITIONS
1790 REMark CAP=ENGINE CAPACITY
1800 REMark CONSTA=CONSTANT IN VAN DE
R WAALS EQUATION
1810 REMark CONSTB=CONSTANT IN VAN DE
R WAALS EQUATION
1820 REMark CRANK=ANGLE OF CRANK
1830 REMark DADV=DELTA ADVANCE
1840 REMark DELAY=IGNITION DELAY
1850 REMark GAMMA=RATIO OF SPECIFIC H
EATS
1860 REMark INDICATE=INDICATOR 0-FIRS
T TIME ROUND FINDTIME LOOP1-SECOND T
IME ROUND LOOP 2-CONTINUE REDUCING AD
VANCE 3-CONTINUE INCREASING ADVANCE
1870 REMark KENG=CONSTANT RELATING TE
MP RISE TO ENERGY FROM FUEL
1880 REMark KRATE=CONSTANT
1890 REMark LOOPANG=ANGLE CRANK MOVES
THROUGH IN ONE INCREMENT
1900 REMark OADV(1 -> 10)=OPTIMUM ADV
ANCE AT REVS
1910 REMark OLDADV=LAST VALUE OF ADVA
NCE
1920 REMark OLDWORK=LAST VALUE OF WOR
K
1930 REMark PINLET=START PRESSURE
```

```
1940 REMark PRESS0=OLD GAS PRESSURE
1950 REMark PRESS1=NEW GAS PRESSURE
1960 REMark R=UNIVERSAL GAS CONSTANT
1970 REMark RA=REMAINDER OF AIR/FUEL
TO BE BURNT
1980 REMark REVS=ENGINE SPEED RPM
1990 REMark ROD=LENGTH OF CON ROD
2000 REMark STROKE=ENGINE STROKE
2010 REMark TEMP0=OLD GAS TEMPERATURE
2020 REMark TEMP1=NEW GAS TEMPERATURE
2030 REMark v=LOCAL VARIABLE
2040 REMark VCOMB=VOLUME OF COMBUSTIO
N CHAMBER
2050 REMark VMOLE=VOLUME OF ONE MOLE
AT STP
2060 REMark VOL0=VOLUME OF GAS OLD
2070 REMark VOL1=VOLUME OF GAS NEW
2080 REMark WORK=WORK DONE
```

CHAPTER 8

What We Want is Information

What do you want to know?

…"critical-path analyses"…

There is a significant role for simulations in the provision of information. The bulk of such information applications are in the area of project planning and management support. This chapter considers the use of simulations in this role.

Background

It is well recognised that the information business is big business. Newspapers, television, publishing, reporting: these represent the public side of a business that has continued to grow at a sustained high rate for the past 50 years. There is another, less well known, aspect which has been highlighted recently by the introduction of small computers into business. This is the provision of information services for management.

The management of modern companies is not usually based on the old idea of a single boss or small board making many day-to-day decisions and simply handing down the results of these decisions as actions to be carried out by employees. Such companies do still exist but tend to be small or in financial difficulties. In order to respond flexibly to the demands of a changing world, it is necessary for a larger proportion of any company to have a direct understanding of management aims and to have a responsibility for making management decisions. This approach can only work if good communications exist and information systems are built up.

Computer systems often provide the means for building up information facilities. This does not, however, mean that there is an important role for simulations. Requirements vary from company to company and different management groups work in different ways. Various financial and commercial models may be used but are not common. Representations of the internal workings of a company are becoming popular, usually to keep track of manpower and stock, but few of these use simulation techniques as yet.

The major area of interest for the implementation of simulations is currently in project management. This involves the long-term planning of programmes of work and the implementation of such plans on a day-to-day basis. In this application, the system being simulated is the project itself and the techniques used for the simulation range from simple critical-path analyses to highly sophisticated resource-optimisation programs.

Example

My example for this chapter is a simple project planning task to be carried out using a general-purpose program. The program is a basic critical-path analysis program which builds up a model of the whole project, identifies those parts of the project which are most important, and estimates an overall manpower requirement

I could have chosen from a number of everyday examples for this chapter, including preparing for a party, a year plan for a garden, writing a book, or re-decorating a kitchen. Almost anything which requires forward planning in fact. It is more usual, however, to apply critical-path analysis to business problems and I have selected a simple manufacturing process as my example.

Lookitthat Ltd. manufacture kites. The company is small (normally no more than six employees, two of which are part-time) but devotes a high proportion of its effort to the development of new designs. Before going into full production with a new kite design, Lookitthat Ltd. usually prepares a special batch of twenty kites in order to test out production methods, to give them a small stock to show their retailers, and for use as test models. Such small batches are often used by manufacturers and are known as pre-production prototypes.

Two people are available to make the kites and both can complete any of the tasks required. **Figure 8.1** gives a list of the tasks. What we want to know is: how long will the manufacture take, which are the important tasks (the critical-path), and how much of the available effort is actually used.

TASK	TITLE OF TASK (20 CHARACTERS)	NO OF TIME STEPS	MANPOWER REQUIREMENT	PRECEDING TASKS TO BE COMPLETED
A	Cut material	7	2	—
B	Stitch material	13	1	A
C	Cut struts	4	1	—
D	Cut tails	2	1	—
E	Wind tails	2	1	D
F	Cut control strings	1	1	—
G	Wind control strings	2	1	F
H	Fold kites	3	2	B
I	Collect components	4	1	H G E C
J	Package	3	2	I

No of tasks = 10
Unit of time = 15 minutes
Total manpower available = 2

Figure 8.1: List of Tasks Required to Manufacture 20 Pre-production Prototypes for Lookitthat Ltd.

Simulation development

We are going to use a general-purpose program, so we will not go through the full selection technique in this case. What I shall do however is describe the program itself, how to use it, and how it might be extended.

The program 'Critical' takes information on a number of tasks and works out a plan for completing the tasks within a given manpower limitation. Output consists of the plan itself, the actual manpower requirement, and an indication of those tasks which are critical to complete the planned work on time. Any delays occurring on one of these critical tasks will result in a delay in the project completion time.

Several basic approaches to the writing of a critical-path model are possible. The differences are principally concerned with the way in which the project plan is built up and also the control over this process desired by the user. Tasks may be given different relative priorities, or particular start-dates. Resources other than manpower may be added into the calculation, which must be kept within pre-set limits. It may be required to smooth out the use of certain resources, including manpower, as much as possible. Another common requirement is to build in an allowance for problems and delays and to have the program indicate just how sensitive the project plan is to such things.

Our program is a basic one, however, albeit with a capability for extension. The core of the program is a routine which builds up the project plan. This is achieved by initially considering all tasks, discounting those which cannot be commenced yet as other tasks on which they are dependent are still to be completed, and selecting the task which is of the longest duration from those which can be started within the manpower currently available. The principle behind selecting the longest-duration task is that such a task is most likely to be on the critical-path and thus should not be delayed. A more sophisticated approach might analyse the forward dependency of tasks, ie establish which tasks in the future will be dependent on current tasks, and, using this information, derive a relative priority for tasks. As I have already indicated, though, this is a basic program lacking many of the possible sophistications. If spare effort is available then additional tasks may be selected, on the same basis of priority given to long duration.

The duration of tasks is input as a multiple of some fixed base-unit, which will be 15 minutes in our example. In building up the project plan, the program steps through the project using the base-unit-of-time. This time unit is also used to display the completed project plan. The display shows all of the input tasks and their relative positions in the plan, and indicates if they are on the critical-path by marking their duration with a C. Tasks not on the critical-path are marked with *.

Results

Figure 8.2 is the output provided by the critical-path program.

Tasks A, B, H, I, and J provide the critical-path for this particular project. A total time of 7. 5 hours is indicated for the production test run, just right for a full day's work. Also, the utilisation (the proportion of time for which the manpower available is actually used) is quite high, and little benefit can be gained from playing around with the plan.

```
TASK (PAGE 1;0 TO 450 MINUTES)

A : CCCCCCC
B :       CCCCCCCCCCCCC
C :       ****
D :           **
E :             **
F :               *
G :                **
H :                  CCC
I :                     CCCC
J :                         CCC

SCALE IS ONE CHARACTER = 15 MINUTES
PROJECT LENGTH = 450 MINUTES
MANPOWER UTILISATION = .9 x 2
```

Figure 8.2: Production Plan Critical-Path Analysis.

Note that Task I, collecting together the components in preparation for final packaging, is marked as a single task dependent on the completion of all manufacturing tasks. In reality, of course, this task could be spread out and time saved. This task should really be broken down into sub-tasks.

Program description and use

Figure 8.3 presents an overall flowchart for the program 'Critical'.

Definitions of the variables used in the program are given in REMark statements at the end of the progam listing. The routines used in the program have the following functions.

PROCedure INITIAL: Sets up dimension statements and initialises variables.

PROCedure DATAIN: Inputs data on the project, including how many tasks there are, the time-step to be used for the project plan, total manpower available, the title of each task along with task duration and manpower requirement. Initial data on each task is also set.

PROCedure PLAN: Steps through the project plan one time-step at a time and gradually extends the project plan. Tasks are started as early as possible within the manpower available.

FuNction testend: Tests for the completion of the project plan. Returns a value of 0 if so.

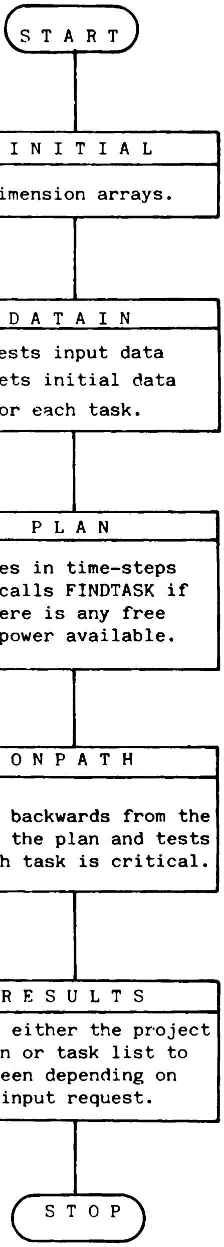

START
INITIAL
Dimension arrays.
DATAIN
Requests input data and sets initial data for each task.
PLAN
Cycles in time-steps and calls FINDTASK if there is any free manpower available.
ONPATH
Cycles backwards from the end of the plan and tests if each task is critical.
RESULTS
Prints either the project plan or task list to screen depending on input request.
STOP

Figure 8.3: Flowchart for the Program 'Critical'.

PROCedure REVISE: At the start of a new project time-step, this routine checks to see if tasks previously in progress have ended, and marks them as completed, using the indicator variable I(n), if so. The routine then checks all the tasks which are dependent on the completion of previous tasks and determines whether or not these can now be started.

FuNction effort: Returns a total manpower figure for tasks in progress.

PROCedure FINDTASK: If some manpower is available, then a new task is selected (longest duration given priority) from those tasks remaining.

PROCedure ONPATH: This takes a completed plan and works backwards through it, considering each task in turn. For each task, a check is made for other dependent tasks starting immediately after the current task ends. If no such dependent task exists and manpower is available, the task is moved on one time-step. This process is repeated until the task can be moved no further. All tasks are tested in this way. Those which cannot be moved lie on the critical-path and I(n) is set to 4.

FuNction deffort: Returns a total estimated manpower requirement for all tasks assuming latest possible start-time.

PROCedure RESULTS: This prints the results to screen. The user can choose between a listing of the project tasks or a diagram of the project plan itself. If the project plan is too long to be shown on a single screen, then several 'pages' are displayed.

Program use is relatively straightforward. The program works in a linear fashion, taking a full set of input, producing a project plan, and providing a set of results.

To run the example problem, simply enter the data provided in Figure 8. 1. Several minutes are required to run the complete program.

Program improvements

'Critical' obviously has much room for improvement. As it stands, the program is not very 'user friendly'; the raw data is not stored on microdrive and cannot be modified once entered, mistakes on entering data can cause the program to stop with an error message, the linear data entry used does not encourage manipulation of the raw data to answer 'what-if?' type questions, hardcopy of the input data and the resultant project plan is not available, and the limit of 15 tasks (chosen to simplify the PROCedure RESULTS) restricts practical applications.

Replacing the current main program with a menu-driven program could provide a means of providing additional facilities. Defined processes for microdrive and printer operations and for updating data could easily be added.

Also, notice how I have included a variable A(1,n) which stores the starting time of a task. This is currently calculated as part of the project planning process. It could equally well be input by the user, thus giving more control over the project plan.

A final facility which I think it would be useful to add is a means of defining a relative priority of tasks. This would force, or, encourage (depending on how the facility is implemented) the program to give some tasks priority over others.

The addition of the facilities noted above would allow the user to enter data in a simple form, produce a first project plan, and modify the plan as desired to suit requirements.

Program listing

```
100 REMark CRITICAL
110 MODE 256
120 CSIZE 2,0
130 INITIAL
140 DATAIN
150 PLAN
160 ONPATH
170 RESULTS
180 STOP

190 DEFine PROCedure INITIAL
200  DIM I(15)
210  DIM A(4,15)
220  DIM A$(2,15,20)
230  DIM O$(15)
240  O$=""
250  U$=""
260 END DEFine

270 DEFine PROCedure DATAIN
280  LOCal b,m,n,b$
290  CLS:INPUT "HOW MANY TASKS ARE TH
ERE?";TOTAL
300  REPeat TIMEIN
310   CLS:PRINT "DO YOU WANT TO PLAN
IN"
320   INPUT "(D)ays, (H)ours, OR (M)i
nutes?";b$
330   b=CODE(b$)
340   SELect ON b
350    ON b=68
360     U$="DAYS"
370     EXIT TIMEIN
```

```
380    ON b=72
390     U$="HOURS"
400     EXIT TIMEIN
410    ON b=77
420     U$="MINUTES"
430     EXIT TIMEIN
440   END SELect
450  END REPeat TIMEIN
460  CLS:PRINT "HOW MANY ";U$;
470  INPUT " IN ONE TIME-STEP?";LENGT
H
480  CLS:INPUT "WHAT IS THE MANPOWER
AVAILABLE?";MANPOWER
490  FOR n=1 TO TOTAL
500   CLS:PRINT "INPUT TITLE OF TASK
";CHR$(n+64)
510   PRINT "(UP TO 20 CHARACTERS)"
520   INPUT A$(1,n)
530   PRINT:PRINT "HOW MANY TIME-STEP
S OF ";LENGTH;" ";U$
540   PRINT "IS TASK ";CHR$(n+64);" ?
";
550   INPUT A(2,n)
560   PRINT:PRINT "WHAT IS THE MANPOW
ER REQUIREMENT"
570   PRINT "FOR TASK ";CHR$(n+64);"
?";
580   INPUT A(3,n)
590   REPeat DEPEND
600    PRINT:PRINT "WHAT TASKS IS TAS
K ";CHR$(n+64);" DEPENDENT UPON?"
610    INPUT "(INPUT STRING OF LETTER
S THEN ENTER)",A$(2,n)
620    b=LEN(A$(2,n))
630    IF b>0 THEN
640     FOR m=1 TO b
650      IF CODE(A$(2,n,m))<65 OR COD
E(A$(2,n,m))>64+TOTAL THEN EXIT m
660     NEXT m
670      EXIT DEPEND
680     END FOR m
690    ELSE
700     EXIT DEPEND
710    END IF
720   END REPeat DEPEND
730   A(4,n)=0
740   A(1,n)=0
750   I(n)=0
760  END FOR n
770 END DEFine

780 DEFine PROCedure PLAN
790  CLS:PRINT "PLANNING"
```

```
800  UTIL=0
810  TIME=-1
820  REPeat PLAN1
830   TIME=TIME+1
840   REVISE
850   IF testend=0 THEN EXIT PLAN1
860   REPeat STARTTASK
870    MFREE=MANPOWER-effort
880    IF MFREE=0 THEN
890     EXIT STARTTASK
900    ELSE
910     FINDTASK
920     IF T=0 THEN EXIT STARTTASK
930    END IF
940   END REPeat STARTTASK
950   UTIL=UTIL+(MANPOWER-MFREE)
960  END REPeat PLAN1
970  PLANEND=TIME-1
980  UTIL=UTIL/(MANPOWER*(PLANEND+1))
990 END DEFine
1000 DEFine FuNction testend
1010  LOCal n
1020  FOR n=1 TO TOTAL
1030   IF I(n)<>3 THEN RETurn 1
1040  END FOR n
1050  RETurn 0
1060 END DEFine
1070 DEFine PROCedure REVISE
1080  LOCal k,m,n,p
1090  FOR n=1 TO TOTAL
1100   IF I(n)=2 AND A(1,n)+A(2,n)-1<
TIME THEN I(n)=3
1110  END FOR n
1120  FOR n=1 TO TOTAL
1130   IF I(n)=0 THEN
1140    k=1
1150    m=0
1160    REPeat DEPENDS
1170     m=m+1
1180     IF LEN(A$(2,n))=0 THEN k=1:E
XIT DEPENDS
1190     IF m>LEN(A$(2,n)) THEN EXIT
DEPENDS
1200     p=CODE(A$(2,n,m))
1210     IF I(p-64)<>3 THEN k=0
1220    END REPeat DEPENDS
1230    I(n)=k
1240   END IF
1250  END FOR n
1260 END DEFine
1270 DEFine FuNction effort
1280  LOCal m,n
```

```
1290  m=0
1300  FOR n=1 TO TOTAL
1310   IF I(n)=2 THEN m=m+A(3,n)
1320  END FOR n
1330  RETurn m
1340 END DEFine

1350 DEFine PROCedure FINDTASK
1360  LOCal n,m,p
1370  p=0:m=0:T=0
1380  FOR n=1 TO TOTAL
1390   IF I(n)=1 AND A(3,n)<=MFREE AN
D A(2,n)>m THEN
1400     m=A(2,n)
1410     p=n
1420     T=1
1430   END IF
1440  END FOR n
1450  IF p<>0 THEN
1460   I(p)=2
1470   A(1,p)=TIME
1480   O$=O$&CHR$(p+64)
1490  END IF
1500 END DEFine

1510 DEFine PROCedure ONPATH
1520  LOCal m,n,p,q
1530  CLS:PRINT "FINDING CRITICAL PAT
H"
1540  FOR n=1 TO TOTAL
1550   A(4,n)=A(1,n)
1560  END FOR n
1570  m=TOTAL+1
1580  REPeat BACKWARDS_SEARCH
1590   m=m-1
1600   IF m=0 THEN EXIT BACKWARDS_SEA
RCH
1610   n=CODE(O$(m))-64
1620   REPeat MOVE_TASK
1630    TIME=A(4,n)+A(2,n)
1640    IF TIME>PLANEND THEN EXIT MOV
E_TASK
1650    FOR p=1 TO TOTAL
1660     IF A(4,p)=TIME AND LEN(A$(2,
p))>0 THEN
1670      FOR q=1 TO LEN(A$(2,p))
1680       IF CODE(A$(2,p,q))=n+64 TH
EN EXIT MOVE_TASK
1690      END FOR q
1700     END IF
1710    END FOR p
1720    MFREE=MANPOWER-deffort
1730    IF A(3,n)>=MFREE THEN A(4,n)=
A(4,n)+1
1740   END REPeat MOVE_TASK
1750  END REPeat BACKWARDS_SEARCH
1760  FOR n=1 TO TOTAL
1770   IF A(1,n)=A(4,n) THEN I(n)=4
1780  END FOR n
1790 END DEFine

1800 DEFine FuNction deffort
1810  LOCal m,n
1820  m=0
1830  FOR n=1 TO TOTAL
1840   SELect TIME=A(4,n) TO A(4,n)+A
(2,n)-1:m=m+A(3,n)
1850  END FOR n
1860  RETurn m
1870 END DEFine

1880 DEFine PROCedure RESULTS
1890  LOCal b$,c$,e$,m,n,p,q,r,s,v,w
1900  CLS
1910  REPeat RESLOOP
1920   CLS:INPUT "ENTER P FOR PROJECT
PLAN, T FOR LIST OF TASKS ";b$
1930   v=CODE (b$)
1940   SELect ON v
1950    ON v=80
1960     CLS
1970     p=0
1980     REPeat PLANPAGE
1990      p=p+1
2000      m=30*(p-1)
2010      q=m+29
2020      IF q>PLANEND THEN q=PLANEND
2030      CLS
2040      PRINT "TASK  (PAGE ";p;"; "
;m*LENGTH;" TO ";(q+1)*LENGTH;"  ";U$;
 ")"
2050      PRINT
2060      FOR n=1 TO TOTAL
2070       e$="  "&CHR$(n+64)&":"
2080       s=A(1,n)+A(2,n)-1
2090       w=A(1,n)
2100       SELect ON w
2110        ON w=0 TO m
2120         r=0
2130        ON w=m+1 TO q
2140         r=w-m
2150        ON w=REMAINDER
2160         r=-1
2170       END SELect
2180       SELect ON s
2190        ON s=0 TO m
```

```
2200         w=-1
2210         r=-1
2220        ON s=m+1 TO q
2230         w=s-m
2240        ON s=REMAINDER
2250         w=q-m
2260         IF r=-1 THEN w=-1
2270       END SELect
2280       s=w
2290       w=l(n)
2300       SELect ON w
2310        ON w=3
2320         c$="*"
2330        ON w=4
2340         c$="C"
2350       END SELect
2360       IF r>0 THEN
2370        FOR w=1 TO r:e$=e$&" "
2380       END IF
2390       IF r<>-1 THEN e$=e$&c$
2400       IF s>r THEN
2410       FOR w=r TO s-1:e$=e$&c$
2420      END IF
2430       PRINT e$
2440      END FOR n
2450     PRINT "SCALE IS ONE CHARACTE
R = ";LENGTH;" ";U$
2460     PRINT "PROJECT LENGTH = ";(P
LANEND+1)*LENGTH;" ";U$
2470     PRINT "MANPOWER UTILISATION
= ";1E-2*INT(UTIL*100+.5);"x";MANPOW
ER
2480     IF q<PLANEND THEN
2490      INPUT "PRESS ENTER FOR NEXT
PAGE OF PLAN ";e$
2500     ELSE
2510      INPUT "PRESS ENTER TO CONTI
NUE ";e$
2520      EXIT PLANPAGE
2530     END IF
2540    END REPeat PLANPAGE
2550    ON v=84
2560    CLS
2570    FOR n=1 TO TOTAL
2580     PRINT "TASK ";CHR$(n+64);" =
";A$(1,n)
2590    END FOR n
2600    INPUT "PRESS ENTER TO CONTINU
E ";e$
2610   END SELect
2620  END REPeat RESLOOP
2630 END DEFine
```

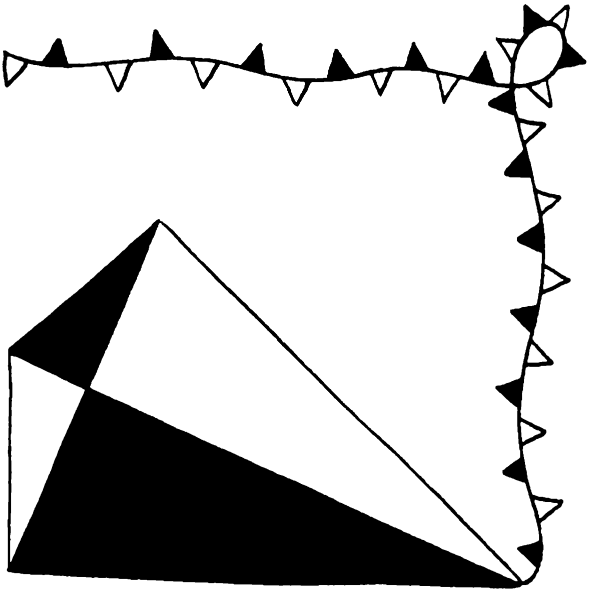
```
2640 REMark VARIABLES LIST
2650 REMark A(1,1 -> 15)=START OF TAS
K (UNITS OF TIME FROM START OF PROJEC
T)
2660 REMark A(2,1 -> 15)=DURATION OF
TASK (UNITS OF TIME)
2670 REMark A(3,1 -> 15)=MANPOWER REQ
UIREMENT OF TASK (PEOPLE)
2680 REMark A(4,1 -> 15)=LATEST START
TIME  POSSIBLE  FOR  TASK
WITHOUT DELAYING PROJECT END
2690 REMark A$(1,1 -> 15)=TITLES OF T
ASKS
2700 REMark A$(2,1 -> 15)=RECORD OF T
ASKS WHICH MUST BE  COMPLETED BEFORE
CURRENT TASK
2710 REMark O$(1 -> 15)=RECORDS ORDER
OF TASKS AS PLAN IS BUILT UP
2720 REMark I(1 -> 15)=INDICATOR OF T
ASK STATUS AS PROGRAM  STEPS THROUGH
PLAN 0-TASK WAITING FOR COMPLETION OF
 OTHER TASKS
2730 REMark    1-TASK WAITING FOR EFF
ORT
2740 REMark    2-TASK IN PROGRESS
2750 REMark    3-TASK COMPLETE
2760 REMark    4-TASK ON CRITICAL-PAT
H
2770 REMark FREE=MANPOWER CURRENTLY U
NALLOCATED
2780 REMark LENGTH=LENGTH OF TIMESTEP
2790 REMark MANPOWER=MAXIMUM AVAILABL
E MANPOWER
2800 REMark PLANEND=END OF PROJECT PL
AN
2810 REMark T=INDICATOR (IF T=0 THEN
CANNOT START NEW TASK)
2820 REMark TIME=CURRENT TIME
2830 REMark TOTAL=TOTAL NO OF TASKS (
UP TO 15)
2840 REMark UTIL=OVERALL UTILISATION
OF AVAILABLE MANPOWER
2850 REMark B$=UNITS FOR TIMESTEP
2860 REMark b$=INPUT CHOICE FOR TASK
LIST OR PLAN
2870 REMark c$=HOLDS "C" OR "*"
2880 REMark e$=HOLDS SINGLE TASK PLOT
& MISC.
2890 REMark m=START TIME FOR PLOT
2900 REMark n=LOOP VARIABLE
2910 REMark p=PAGE NO
2920 REMark q=END TIME FOR PLOT
```

```
2930 REMark r=START TIME FOR TASK
2940 REMark s=END TIME FOR TASK
```

CHAPTER 9

And Out of the Unknown...

Simulating the unknown



"...he will be able to fly his kites..."

How is it possible to simulate the unknown?

Background

Nearly all simulations involve elements of the unknown. One of the major classes of simulation however, as I described in Chapter 3, is the analytical simulation which is designed to investigate the unknown characteristics of a system. Analytical simulations take known features of the system in question and, by making various assumptions about the workings of the system, fill in the unknown areas to give a practical working model of the system.

Analytical simulations of unknown system-characteristics are extremely experimental in nature: although you may have theories about how the simulation will work, it is only when you actually start to test the thing out that you begin to verify and extend those theories. Trial and error is the order of the day, and this sets the conditions under which a technique can be adopted. A highly detailed emulation technique is most unlikely to be useful if the unknown parts of the system are a significant part of the whole. It is better to start off with a simple simulation which models the major system responses only and gradually to extend this into a more complex model as you gain knowledge of the system.

Another well-used approach is to use knowledge of other similar systems to provide a model of the system to be investigated, and thus to predict its characteristics. For example, if you are the manager of a Formula 1 car racing team and wish to analyse the performance of a competitor's car then probably the best starting point would be to model your own car first and use this as a basis for your evaluation. A further example of the use of existing knowledge to predict the characteristics of something new is the development and use of commercial expert systems. These are specifically designed to build up a store of knowledge in such a way that new systems with unknown characteristics can be emulated and described.

Example

One of the more interesting examples from my case-book is a simple simulation which attempts something which I have described, in previous chapters, as extremely difficult and unreliable. The interest comes not only from the practical aspect of the simulation, but also in the techniques used.

Over the past few years, I have worked on several projects for the famous kite makers, Lookitthat Ltd. Their research department, consisting of the company founder, Ivor Problem, and his secretary, Sophie Aye, were having difficulties with their hardware development because they could never predict whether or not there would be sufficient wind to try out their latest kites. They were having trouble planning their work around our changeable weather and asked for a method for predicting the weather tomorrow given a simple description of today's weather.

My solution to their problem, which I originally designed for use on the Sinclair ZX81 but which becomes much more versatile on the QL, is a simple simulation of the weather changes from day to day, with a prediction of the likelihood that the weather tomorrow will be suitable for kite flying.

Simulation development

The requirement is for a model of weather conditions which vary from day to day, and which will indicate the likely conditions one day ahead. Obviously, I cannot expect Ivor Problem to spend a lot of time taking wind and temperature readings or constantly phoning the local meteorological office for information — data input must be simple. Also, I should be able to give Ivor a direct indication as to how probable it is that he will be able to fly his kites — output must be simple.

Any of the techniques mentioned in Chapter 4 for modelling probabilities could be adopted. I chose a Markov chain approach as the basic means of predicting weather changes (see Chapter 4), because of the relative simplicity of the model itself and the ease of running the developed simulation. The system-state matrix is based on a very much simplified description of the weather, concentrating on three fundamental attributes of temperature, wind strength, and rainfall (or other forms of precipitation, snow for example). The most important attributes from Ivor's point of view are wind and rain. I also include temperature, as this gives an easily understood indication of the general weather conditions, much easier than air pressure or humidity. Another possibility which I have not included, in order to keep the input as simple as possible, is the percentage cloud cover. I list below the possible weather attributes and the resultant possible system-states. Notice how I have chosen descriptions for the various weather conditions which are immediately understandable by Ivor and useful to him.

WEATHER ATTRIBUTE	DESCRIPTION	LABEL
WIND	Too light for kite flying	W0
	OK for kite flying	W1
	Too strong for kite flying	W2
RAIN	None: OK for kites	R0
	Light or infrequent: OK	R1
	Heavy: No good for kites	R2
TEMPERATURE	Hot: OK	T0
	Intermediate:OK	T1
	Cold: Rather stay at home	T2

SYSTEM-STATE	ATTRIBUTES	SYSTEM-STATE	ATTRIBUTES
0	W0 R0 T0	14	W1 R1 T2
1	W0 R0 T1	15	W1 R2 T0
2	W0 R0 T2	16	W1 R2 T1
3	W0 R1 T0	17	W1 R2 T2
4	W0 R1 T1	18	W2 R0 T0
5	W0 R1 T2	19	W2 R0 T1
6	W0 R2 T0	20	W2 R0 T2
7	W0 R2 T1	21	W2 R1 T0
8	W0 R2 T2	22	W2 R1 T1
9	W1 R0 T0	23	W2 R1 T2
10	W1 R0 T1	24	W2 R2 T0
11	W1 R0 T2	25	W2 R2 T1
12	W1 R1 T0	26	W2 R2 T2
13	W1 R1 T1		

The weather conditions which are suitable for kite flying are represented by system-states 9, 10, 12, and 13.

But what about the transition matrix? If it's warm, dry, and there's a steady wind blowing today, what will it be like tomorrow? The transition matrix is a collection of probabilities of the weather changing from its current state today into one of the 27 possible states tomorrow. The problem is that we don't know what these probabilities are. My solution to this problem is to give the simulation a capability to learn as more and more data is input, and thus build up a store of knowledge. This is where the analysis part of the program resides; the simulation builds up a model of the weather. In this way the program also works like an extremely simple expert system; the more information you give it, the more expert it becomes — up to a point.

Two more points must be covered: we must set up initial values (guesses) for the transition matrix and we must decide what we want in terms of input/output.

Initial values for the transition matrix could be determined in a number of different ways and then input manually. I have decided that it would be better to give the simulation a running start and attempt to base the initial transition matrix on a logical set of assumptions. Also, I want the program itself to set up the initial transition matrix.

My first assumptions for the transition matrix probabilities are based on the weather attributes of wind, rain, and temperature. Each of these can be described in one of three states. I think it reasonable to assume that there is a 60 per cent chance that a particular attribute will stay the same from one day to the next and that, if it changes, it will not change from one extreme to another. Thus, if it is raining lightly today, the chances of rain states R0, R1, and R2 tomorrow are 0. 2, 0. 6, and 0. 2 respectively. Similarly, if it is hot today, the probabilities of T0, T1, and T2 tomorrow are 0. 6, 0. 4 and 0.

Finally, input and output. Each day the program needs to be informed of the current weather conditions and will make a prediction based on the estimated probability of system-states 9, 10, 12, or 13. It would not be very kind to demand of Ivor that he remember all the system-states, which would be the simplest form of input. The simulation asks about each of the weather attributes in turn and, from this, determines the full system-state.

Output is in the form of a prediction of the suitability of the weather tomorrow for kite flying and also an indication as to how much confidence can be placed on the prediction. This confidence factor is an important element of predictor simulations, as I have pointed out in Chapter 3. In this case, it is not possible easily to derive a statistical confidence level for the prediction because we do not know what the random element is, if any, in the change of one weather state to another. We could make an assumption based on a binomial distribution, or a Poisson distribution, or one of the other common methods, but I see no justification for any of these with this low-level simulation. What the program that I developed for Ivor does is to keep track of the number of times that the weather has been in a given state and use this to give a rough indication of the reliability of the relevant transition probabilities. If fewer than 10 previous similar weather states have been recorded then the prediction will be at best unreliable. Between 10 and 30 then not too bad, over 30 and the results are getting to be as good as the simulation can give. This defines the estimation of confidence and the way of outputting that confidence level.

Storage of the transition matrix and data is important and is done automatically (on request). A microdrive cartridge should be left in drive 1 whilst the program runs.

Results

This is a simple simulation which relies on the slow acquisition of data. It thus takes some time to start producing reasonable results (typically one to three months). This process can be shortened by making a trip to the local library, and getting daily weather reports from the newspapers stored there, and thus building up the stored data using historical data.

Once the data-base starts to build up, the predictions made by the simulation are quite good, considerably better than just guessing. Of course the output is very simple, which helps the prediction process, but the output is also tailored to the specific requirements of Ivor Problem and he finds the results more useful than trying to interpret the published weather forecasts.

Program description

Figure 9.1 presents an overall flowchart for the program 'Itmaybe'.

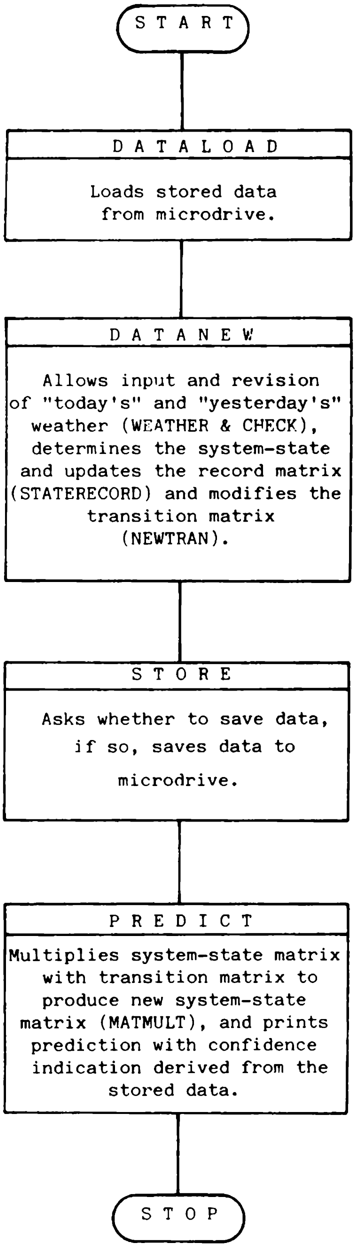

START
DATALOAD
Loads stored data from microdrive.
DATANEW
Allows input and revision of "today's" and "yesterday's" weather (WEATHER & CHECK), determines the system-state and updates the record matrix (STATERECORD) and modifies the transition matrix (NEWTRAN).
STORE
Asks whether to save data, if so, saves data to microdrive.
PREDICT
Multiplies system-state matrix with transition matrix to produce new system-state matrix (MATMULT), and prints prediction with confidence indication derived from the stored data.
STOP

Figure 9.1: Flowchart for Program 'Itmaybe'.

Definitions of program variables are given in REMark statements at the end of the program listing. The routines used in the program have the following functions.

PROCedure DATALOAD: Loads in the stored data from microdrive, and sets values for yesterday's weather from a set of temporary stores.

PROCedure DATANEW: Requests input for today's weather, verifies both today's and yesterday's weather, determines the appropriate system-state, updates the record matrix, and updates the transition matrix.

PROCedure STORE: Asks if data is to be stored on to microdrive, and, if so, saves the new data.

PROCedure PREDICT: Multiplies the system-state matrix and the transition matrix, and prints out the prediction for tomorrow's weather, together with an indication of the reliability of the prediction, derived from the historic data stored.

PROCedure WEATHER: Prints the request for weather data input, and gives the options available.

PROCedure CHECK: Prints out the data stored for today's and yesterday's weather and asks if it is correct. If not, allows for the modification of the data.

PROCedure STATERECORD: Determines the system-state from the data input, and updates the system-state matrix and the record matrix.

PROCedure NEWTRAN: Modifies the probabilities held in the transition matrix gradually, according to the historical data held in the record matrix.

PROCedure DATASAVE: Sets today's data into a temporary store, and saves all data files.

PROCedure MATMULT: Produces predicted system-states by multiplying the system-state and transition matrices together.

PROCedure WPRINT: Converts stored numerical values for the weather into a verbal description.

PROCedure START_UP: Initialises and saves the program data.

PROCedure INITIAL: Sets up initial data for the transition matrix.

PROCedure INITMISC: Requests initial data for today and yesterday, and sets up the record matrix.

PROCedure DATASAVE_NEW: Creates the file WEATHER_DATA on drive 1.

FuNction keyin1: Waits for keyboard input and checks if it is 0, 1, or 2.

FuNction keyin2: Waits for keyboard input and checks if it is Y/y or N/n.

FuNction setprob: Calculates the initial probabilities of the change of one system-attribute into another. Used in the initialisation of the transition matrix.

FuNction traninit: Calculates the probabilities of transition from a particular system-state into any other. Represents a single row of the initial transition matrix before modification by historical data.

To run the program, the program must first initialise and save the data arrays and data files, including the transition matrix. In order to do this, you must run the initialisation routine by calling up PROCedure START_UP directly. Simply type:

```
START_UP (enter)
```

and the program (with some assistance from you) will set up the initial arrays. You will be asked for information on yesterday's and today's weather, and the data will be saved to microdrive 1 (make sure there is a cartridge in the drive before you start!).

Once the data is on microdrive, the main program can be used, and the initialisation routine need not be used again.

Program improvements

I don't think that there is a lot of point trying to add to the sophistication of this model. If you want to fly a kite, sail a boat, go hang-gliding, or something else which requires prediction of some particular aspect of the weather, then use the program at its current level. My feeling (note that I'm resorting to a qualitative argument here) is that increased complexity will not improve the simulation and may, in fact, make it less reliable.

The basic technique, however, is very useful and can be applied in a number of other fields. Some applications will demand a more detailed approach. Watch out for any increase in the number of system-states — my experience is that computing time goes up with the fourth power of the number of system-states.

Program listing

```
100 REMark ITMAYBE
110 REMark INPUT "START_UP" AS A CIRE
CT COMMAND TO SET UP THE INITIAL DATA
120 REMark "START_UP" REQUIRES A CART
RIDGE IN DRIVE 1
130 MODE 256
140 CSIZE 2,0
150 DATALOAD
160 DATANEW
170 STORE
180 PREDICT
190 STOP
200 DEFine PROCedure DATALOAD
210  LOCal n,m
220  CLS:PRINT "LOADING HISTORIC DATA
FROM M/DRIVE"
230  DIM TRAN(26,26),REC(26,26),MISC(
2),S1(26),S2(26)
240  OPEN_IN #6;"MDV1_WEATHER_DATA"
250  FOR m=0 TO 26
260   FOR n=0 TO 26
270    INPUT #6;TRAN(m,n)
280   END FOR n
290  END FOR m
300  FOR m=0 TO 26
310   FOR n=0 TO 26
320    INPUT #6;REC(m,n)
330   END FOR n
340  END FOR m
350  FOR n=0 TO 2
360   INPUT #6;MISC(n)
370  END FOR n
380  CLOSE #6
390  W=MISC(0)
400  R=MISC(1)
410  T=MISC(2)
420 END DEFine
430 DEFine PROCedure DATANEW
440  a$="INPUT DATA FOR TODAY'S WEATH
ER"
450  WEATHER X,S,U,a$
460  CHECK
470  STATERECORD
480  NEWTRAN
490 END DEFine
```

```
500 DEFine PROCedure STORE
510  LOCal a
520  CLS:PRINT "DO YOU WANT TO UPDATE
THE DATA STORED ON MICRODRIVE (Y/N)?"
530  a=keyin2
540  SELect a=78,110:RETurn
550  DATASAVE
560 END DEFine


570 DEFine PROCedure PREDICT
580  LOCal total,m,n
590 CLS:PRINT "WORKING ON RESULT"
600  MATMULT
610  CLS:PRINT "THE PROBABILITY OF SU
ITABLE WEATHER FOR KITE FLYING TOMORR
OW IS; "
620  PRINT S2(9)+S2(10)+S2(12)+S2(13)
630  PRINT
640  PRINT
650  m=9*X+3*S+U
660  total=0
670  FOR n=0 TO 26
680   total=total+REC(m,n)
690  END FOR n
700  SELect ON total
710   ON total=0 TO 9
720    PRINT "THE PREDICTION IS UNREL
IABLE"
730   ON total=10 TO 30
740    PRINT "THE PREDICTION IS NOT T
OO BAD"
750   ON total=REMAINDER
760    PRINT "THE PREDICTION SHOULD B
E OK"
770  END SELect
780 END DEFine


790 DEFine PROCedure WEATHER (WIND,RA
IN,TEMP,a$)
800  CLS:PRINT a$
810  PRINT "PRESS 0 IF WIND = TOO LIG
HT"
820  PRINT "PRESS 1 IF WIND = OK"
830  PRINT "PRESS 2 IF WIND = TOO STR
ONG"
840  WIND=keyin1
850  CLS:PRINT a$
860  PRINT "PRESS 0 = NO RAIN ETC"
870  PRINT "PRESS 1 = RAIN NOT TOO BA
D"
880  PRINT "PRESS 2 = RAIN TOO HEAVY"
```

```
890  RAIN=keyin1
900  CLS:PRINT a$
910  PRINT "PRESS 0 IF TEMP = HOT"
920  PRINT "PRESS 1 IF TEMP = OK"
930  PRINT "PRESS 2 IF TEMP = TOO COL
D"
940  TEMP=keyin1
950  CLS
960 END DEFine
970 DEFine PROCedure CHECK
980  LOCal a,a$
990  REPeat CHECK1
1000   CLS:PRINT "THE WEATHER YESTERD
AY WAS; "
1010   WPRINT W,R,T
1020   PRINT "IS THIS OK (Y/N)?"
1030   a=keyin2
1040   SELect a=89,121:EXIT CHECK1
1050   a$="INPUT NEW DATA FOR YESTERD
AY'S WEATHER"
1060   WEATHER W,R,T,a$
1070  END REPeat CHECK1
1080  REPeat CHECK2
1090   CLS:PRINT "TODAY'S WEATHER IS;
"
1100   WPRINT X,S,U
1110   PRINT "IS THIS OK (Y/N)?"
1120   a=keyin2
1130   SELect a=89,121:EXIT CHECK2
1140   a$="WHAT IS TODAY'S WEATHER?"
1150   WEATHER X,S,U,a$
1160  END REPeat CHECK2
1170  CLS
1180 END DEFine
1190 DEFine PROCedure STATERECORD
1200  LOCal n,m
1210  CLS:PRINT "EVALUATING STATE MAT
RIX"
1220  FOR n=0 TO 26
1230   S1(n)=0
1240  END FOR n
1250  n=9*X+3*S+U
1260  S1(n)=1
1270  m=9*W+3*R+T
1280  REC(m,n)=REC(m,n)+1
1290 END DEFine
1300 DEFine PROCedure NEWTRAN
1310  LOCal m,n,total,p
1320  CLS:PRINT "REVISING TRANSITION
MATRIX"
1330  m=9*W+3*R+T
```

```
1340  total=0
1350  FOR n=0 TO 26
1360   total=total+REC(m,n)
1370  END FOR n
1380  SELect ON total
1390   ON total=0
1400    FOR n=0 TO 26
1410     TRAN(m,n)=traninit(m,n)
1420    END FOR n
1430   ON total=1 TO 99
1440    p=total/100
1450    FOR n=0 TO 26
1460     TRAN(m,n)=(1-p)*traninit(m,n
)+p*REC(m,n)/total
1470    END FOR n
1480   ON total=REMAINDER
1490    FOR n=0 TO 26
1500     TRAN(m,n)=REC(m,n)/total
1510    END FOR n
1520  END SELect
1530 END DEFine

1540 DEFine PROCedure DATASAVE
1550 LOCal m,n
1560  CLS:PRINT "SAVING NEW DATA"
1570  MISC(0)=X
1580  MISC(1)=S
1590  MISC(2)=U
1600  OPEN #6,"MDV1_WEATHER_DATA"
1610  FOR m=0 TO 26
1620   FOR n=0 TO 26
1630    PRINT #6;TRAN(m,n)
1640   END FOR n
1650  END FOR m
1660  FOR m=0 TO 26
1670   FOR n=0 TO 26
1680    PRINT #6;REC(m,n)
1690   END FOR n
1700  END FOR m
1710  FOR n=0 TO 2
1720   PRINT #6;MISC(n)
1730  END FOR n
1740  CLOSE #6
1750 END DEFine

1760 DEFine PROCedure MATMULT
1770  LOCal n,m
1780  FOR n=0 TO 26
1790   S2(n)=0
1800   FOR m=0 TO 26
1810    S2(n)=S2(n)+S1(m)*TRAN(m,n)
1820   END FOR m
1830  END FOR n
```

```
1840 END DEFine
1850 DEFine PROCedure WPRINT (WIND,RA
IN,TEMP)
1860  SELect ON WIND
1870   ON WIND=0:PRINT "WIND TOO LIGH
T"
1880   ON WIND=1:PRINT "WIND OK"
1890   ON WIND=2:PRINT "WIND TOO STRO
NG"
1900  END SELect
1910  SELect ON RAIN
1920   ON RAIN=0:PRINT "NO RAIN"
1930   ON RAIN=1:PRINT "NOT TOO MUCH
RAIN"
1940   ON RAIN =2:PRINT "TOO RAINY"
1950  END SELect
1960  SELect ON TEMP
1970   ON TEMP=0:PRINT "TEMP=HOT"
1980   ON TEMP=1:PRINT "TEMP=INTERMED
IATE"
1990   ON TEMP=2:PRINT "TOO COLD"
2000  END SELect
2010 END DEFine

2020 DEFine PROCedure START_UP
2030  REMark INITIALISATION ONLY
2040      INITIAL
2050      INITRAN
2060      INITMISC
2070      STATERECORD
2080      NEWTRAN
2090      DATASAVE_NEW
2100 END DEFine

2110 DEFine PROCedure INITIAL
2120  DIM TRAN(26,26),REC(26,26),MISC
(2),S1(26),S2(26)
2130 END DEFine
2140 DEFine PROCedure INITRAN
2150  LOCal W,R,T,X,S,U
2160  CLS:PRINT "SETTING UP TRANSITIO
N MATRIX"
2170  FOR W=0 TO 2
2180   FOR R=0 TO 2
2190    FOR T=0 TO 2
2200     FOR X=0 TO 2
2210      FOR S=0 TO 2
2220       FOR U=0 TO 2
2230        TRAN(9*W+3*R+T,9*X+3*S+U)
=setprob(W,X)*setprob(R,S)*setprob(T
,U)
2240       NEXT U
2250      NEXT S
```

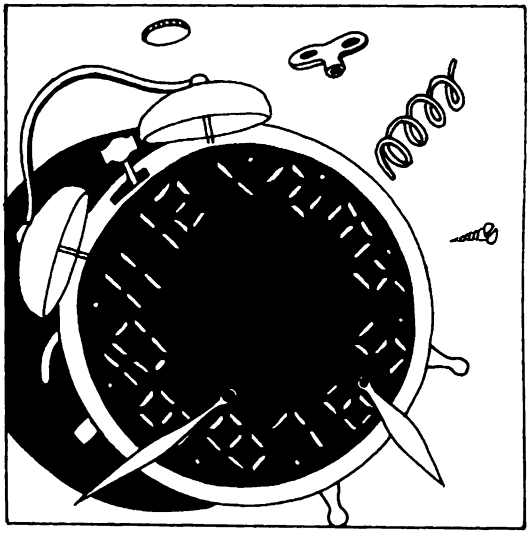
```
2260       NEXT X
2270      NEXT T
2280     NEXT R
2290    NEXT W
2300   END DEFine
2310   DEFine PROCedure INITMISC
2320    LOCal n,m,a$
2330    a$="SET-UP WEATHER (=YESTERDAY)
"
2340    WEATHER W,R,T,a$
2350    a$="SET-UP WEATHER (=TODAY)"
2360    WEATHER X,S,U,a$
2370    CHECK
2380    CLS:PRINT "BLANKING HISTORIC RE
CORD"
2390    FOR m=0 TO 26
2400     FOR n=0 TO 26
2410      REC(m,n)=0
2420     END FOR n
2430    END FOR m
2440   END DEFine
2450   DEFine PROCedure DATASAVE_NEW
2460    LOCal m,n
2470    CLS:PRINT "SAVING INITIAL DATA"
2480    MISC(0)=X
2490    MISC(1)=S
2500    MISC(2)=U
2510    OPEN_NEW #6,"MDV1_WEATHER_DATA"
2520    FOR m=0 TO 26
2530     FOR n=0 TO 26
2540      PRINT #6;TRAN(m,n)
2550     END FOR n
2560    END FOR m
2570    FOR m=0 TO 26
2580     FOR n=0 TO 26
2590      PRINT #6;REC(m,n)
2600     END FOR n
2610    END FOR m
2620    FOR n=0 TO 2
2630     PRINT #6;MISC(n)
2640    END FOR n
2650    CLOSE #6
2660   END DEFine
2670   DEFine FuNction keyin1
2680    LOCal a$,a
2690    REPeat KEY1_LOOP
2700     REPeat WAITLOOP:a$=INKEY$:IF a
$="" THEN EXIT WAITLOOP
2710     REPeat KEYGET:a$=INKEY$:IF a$<
>"" THEN EXIT KEYGET
2720     a=CODE(a$):SELect a=48 TO 50:R
```

```
ETurn (a-48)
2730  END REPeat KEY1_LOOP
2740 END DEFine

2750 DEFine FuNction keyin2
2760  LOCal a$,a
2770  REPeat KEY2_LOOP
2780   REPeat WAIT2:a$=INKEY$:IF a$="
" THEN EXIT WAIT2
2790   REPeat KEY2:a$=INKEY$:IF a$<>"
" THEN EXIT KEY2
2800   a=CODE(a$)
2810   SELect a=78,89,110,121:RETurn
a
2820  END REPeat KEY2_LOOP
2830 END DEFine

2840 DEFine FuNction setprob(a,B)
2850  LOCal c,d
2860  c=B-a
2870  SELect ON c
2880   ON c=2,-2
2890    d=0
2900   ON c=1,-1
2910    IF a=2 THEN
2920     d=.2
2930    ELSE
2940     d=.4
2950    END IF
2960   ON c=0
2970    d=.6
2980  END SELect
2990  RETurn d
3000 END DEFine

3010 DEFine FuNction traninit(m,n)
3020  LOCal X,S,U
3030  X=INT(n/9)
3040  S=INT((n-X*9)/3)
3050  U=n-X*9-S*3
3060  RETurn setprob(W,X)*setprob(R,S
)*setprob(T,U)
3070 END DEFine

3080 REMark GLOBAL VARIABLES
3090 REMark  MISC()=HOLDS  AND STORES
VALUES  OF  TODAY'S  WEATHER  ATTRIBU
TES  IN  ORDER TO BE RETRIEVEDAS YEST
ERDAY'S WEATHER  BY  A  FOLLOWING RUN
 OF THE PROGRAM
3100 REMark  R=YESTERDAY'S RAIN INDIC
ATOR
3110 REMark  RAIN=TEMPORARY RAIN INDI
CATOR
```

```
3120 REMark  REC()=MATRIX STORING THE
TOTAL NUMBER OF OCCURRENCES OF EACH
PARTICULAR SYSTEM-STATE.
3130 REMark  S1()=CURRENT SYSTEM-STAT
E MATRIX
3140 REMark  S2()=DERIVED MATRIX FOR
THE FOLLOWING DAY HOLDING THE PREDICT
ED PROBABILITIES FOR EACH SYSTEM-STAT
E
3150 REMark  S=TODAY'S RAIN INDICATOR
3160 REMark  T=YESTERDAY'S TEMPERATUR
E INDICATOR
3170 REMark  TEMP=TEMPORARY TEMPERATU
RE INDICATOR
3180 REMark  TRAN()=TRANSITION MATRIX
3190 REMark  U=TODAY'S TEMPERATURE IN
DICATOR
3200 REMark  W=YESTERDAY'S WIND INDIC
ATOR
3210 REMark  WIND=TEMPORARY WIND INDI
CATOR
3220 REMark  X=TODAY'S WIND INDICATOR
```

CHAPTER 10
Campaign For Real-time

Real-time simulation



"... normally the time-step
is constant..."

Training simulators and computer games: real-time simulations are a fast-growing business.

Background

Real-time simulators are usually designed to interact with humans. The interaction is often sophisticated in nature: complex images are produced, inputs of various kinds are recognised and acted upon, sounds are produced, and equipment external to the computer may be activated. Because of the inherent complexity of most real-time simulations and the speed of computation required, it is normally necessary to resort to the use of machine code when using a microcomputer. Thus the bulk of real-time simulations are beyond the scope of this book.

Having said that most real-time simulations require the use of machine-code programming, let me now indicate the sort of things that can be done with a high-level language.

In the training field, it is often not necessary to provide a complex screen image and programming time can be saved by having only a small proportion of the image moving at any one time. Thus, where dials and gauges, for example, are represented, they should be presented in as simple a form as possible and only the moving part of the image may need to be simulated. If hardware in addition to the microcomputer is to be used, then it is quite possible that the high-level language will be fast enough.

Simulators may be used in the testing of hardware, duplicating in a controlled and repeatable manner the extended use of a product. Specialised machinery is usually required for such testing, but a low-cost test facility based on a microcomputer can be a practical proposition in some cases. Again, the high-level language will often be sufficient.

Games written in high-level languages will always compare badly with rivals produced in machine code and thus will have a limited commercial value (the reason most computer games are produced). Some concepts are simple enough to allow the high-level language to be used but will almost certainly demand considerable skill from the programmer.

The simulation technique most often used in real-time is based on the Monte Carlo approach. Obviously, a real-time simulation will normally only represent a single (though changing) system-state. When alternative system responses are possible, a random response will be more useful than a deterministic response, unless repeatability is required. Thus the Monte Carlo approach is often used.

There is much more to a real-time simulation than just predicting the system response, however. A large portion of such a simulation program may be devoted to the presentation of the system, the display, sound, or movement of external hardware. It is often this portion of the program which can give the programmer the greatest difficulties, partly because it is here that fast-executing routines will be required and also because much of the programming task will demand the solution of unique problems.

Normally it is possible to trade off memory for execution speed by making use of pre-stored images, using look-up tables rather than performing calculations, and using repetetive blocks of program rather than calling subroutines. A detailed knowledge of the execution times of various program statements can indicate the quickest ways of doing things.

Real-time simulations use a time-step to move from one system-state to the next. A longer time-step will provide more computing time but may lead to unrealistic behaviour (jerky or slow responses). Normally the time-step is constant and it is important to make sure that, where alternative system-reponses are possible, the time taken by the program to represent such responses remains constant. Careful attention must be paid to program execution speed because of this, and it is common to use delays of variable length to keep all execution speeds constant. Such delays can also be used to control the overall speed of the running of the simulation. A reduction in an in-built delay whilst the simulation is running will reduce the time-step and hence speed up the simulation.

Example

My final example is a rudimentary game. I have chosen this to demonstrate some of the points made above.

The game is based on the concept of volley-ball. A high net divides the screen into two. On each side of the net, there is a single player who can move from side to side and hit a ball which must be knocked over the net. Players are represented as a short line known as a paddle.

The object of the game is to force the opposing player (in this case the computer) to drop the ball. Each player may hit the ball up to three times before passing it over the net; if the player hits the ball a fourth time then this counts as a dropped ball. If the ball is hit too hard and goes right over the opposition's playing area then again this counts as a dropped ball.

Points are scored when a ball is dropped. You score one point if your opponent drops the ball after you have served. If your opponent served and also dropped the ball then you do not score a point, but gain the right to serve and thus have a chance of scoring the next point.

One more thing. The game is being played on the moon! Why does it have to be on the moon? The answer is quite simply that I could not get the thing to run fast enough to represent realistically a genuine game of volley-ball and hence had to change reality a bit to fit the simulation.

Simulation development and results

I'm not going to tell you anything at all about program development. You must work your way through the listing itself to see the various tricks that I have used. For example, how have I arranged for the computer to play its moves? Or why have I used KEYROW(1) to read the keyboard rather than INKEY$? Or why have I not put a check in to stop the ball going off the top of the screen?

As for the results, see for yourself.

Program description

No flowchart for this one, even though it is the most complex program in the book — again I want you to sort through for yourself.

Here are some notes on the routines though.

PROCedure INITIAL: Sets up data for whole program.

PROCedure START_UP: Sets up data for each ball.

PROCedure MAKE_DISPLAY: Produces the initial screen display, the playing area. You are represented on the left of the pitch.

PROCedure PLAYERS_TURN: Examines the keyboard for arrow keys being pressed and moves the lefthand paddle if appropriate.

PROCedure COMPUTERS_TURN: Automatically decides the action to be taken by the computer.

PROCedure MOVE_BALL: Updates the ball coordinates but does not print the ball.

PROCedure TEST_FOR_HIT: Tests the new ball coordinates for hits against the side, the net, the paddles, and the base.

PROCedure BALL_OUT: Calls SCORE or NO_SCORE depending on who served and where the ball fell.

PROCedure HIT_NET: Sets ball position next to the net and sets ball speed to zero.

PROCedure HIT_PADDLE: Gives new ball speeds according to how hard ball is to be hit. Tests for ball hit too many times.

PROCedure BALL_DROPPED: Similar to BALL_OUT.

PROCedure BALL_PRINT: Removes the old ball image and prints a new one.

PROCedure MOVE_PADDLE: Produces the moving images of the paddles. How does this work?

PROCedure SCORE: Updates the score and displays the totals.

PROCedure NO_SCORE: Notifies you that the service is changing after a ball is lost without scoring.

Program use and further development

Warning. Some early QLs (with bugs!) can run only small programs. LUNA_BALL is reaching the limit and will 'crash' after a few minutes running: save on to microdrive cartridge before running, to avoid losing your program.

The program runs automatically and even performs the serve (for both players). Use the left/right arrow keys to move your paddle and the up/down arrow keys to hit the ball hard or softly. Only one key should be pressed at a time, otherwise you will be ignored.

Improvements are many. Sound can be added, a limit to the number of points which can be scored would add to realism, the computer's play could be uprated, and so on. As I have mentioned before, the QL screen display is slow, and this limits the whole game: a more complex screen would probably slow things down even more.

That's about it. I hope you have found this book useful. All I can do now is wish you success with your simulations.

Program listing

```
100 REMark LUNA_BALL
110 MODE 256
120 CSIZE 2,0
130 INITIAL
140 REPeat game_loop
150  START_UP
160  MAKE_DISPLAY
170  REPeat ball_in_play
180   PLAYERS_TURN
190   COMPUTERS_TURN
200   MOVE_BALL
210   TEST_FOR_HIT
220   SELect ON E
230    ON E=1 TO 2
240     BALL_OUT
250     EXIT ball_in_play
260    ON E=3
270     HIT_NET
280    ON E=4 TO 5
290     HIT_PADDLE
300     IF H>4 THEN BALL_DROPPED:EXIT
ball_in_play
```

```
310       ON E=6 TO 7
320         BALL_DROPPED
330         EXIT ball_in_play
340     END SELect
350     BALL_PRINT
360     PAUSE 10
370   END REPeat ball_in_play
380 END REPeat game_loop
390 STOP
400 DEFine PROCedure INITIAL
410  G=5
420  SCL=0
430  SCR=0
440  S=RND(1)
450 END DEFine INITIAL
460 DEFine PROCedure START_UP
470  I=S
480  H=1
490  SELect ON S
500   ON S=0
510    VY=-40
520    VX=20
530    XB=72
540    IXB=6
550    OXB=6
560    YB=180
570    IYB=18
580    OYB=18
590   ON S=1
600    VY=-40
610    VX=-20
620    XB=360
630    IXB=30
640    OXB=30
650    YB=180
660    IYB=18
670    OYB=18
680  END SELect
690  XPL=7
700  XPR=29
710 END DEFine
720 DEFine PROCedure MAKE_DISPLAY
730  LOCal n
740  INK 0
750  PAPER 1
760  CLS
770  FOR n=0 TO 19
780   CURSOR 12,n*10
790   PAPER 2
800   PRINT " ":REMark 1 space
810   CURSOR 420,n*10
```

```
820    PAPER 3
830    PRINT " ":REMark 1 space
840   END FOR n
850   FOR n=13 TO 19
860    CURSOR 216,n*10
870    PAPER 7
880    PRINT " ":REMark 1 space
890   END FOR n
900   MOVE_PADDLE XPL,2
910   MOVE_PADDLE XPR,3
920   BALL_PRINT
930   PAUSE RND(50,100)
940  END DEFine
950  DEFine PROCedure PLAYERS_TURN
960   IPL=KEYROW(1)
970   SELect ON IPL
980    ON IPL=2
990     IF XPL>5 THEN XPL=XPL-1
1000   ON IPL=16
1010    IF XPL<14 THEN XPL=XPL+1
1020  END SELect
1030  MOVE_PADDLE XPL,2
1040 END DEFine
1050 DEFine PROCedure COMPUTERS_TURN
1060  LOCal a
1070  a=-1
1080  IF XPR=22 THEN a=0
1090  IF IXB>=OXB AND IXB>XPR-4 AND X
PR<31 THEN a=1
1100  XPR=a+XPR
1110  MOVE_PADDLE XPR,3
1120  IPR=4+124*RND(1)
1130 END DEFine
1140 DEFine PROCedure MOVE_BALL
1150  XB=XB+VX
1160  VY=VY+G
1170  YB=YB+VY
1180  IXB=INT(XB/12)
1190  IYB=INT(YB/10)
1200 END DEFine
1210 DEFine PROCedure TEST_FOR_HIT
1220  LOCal a,b
1230  SELect ON IXB
1240   ON IXB=-100 TO 1
1250    E=1
1260    RETurn :REMark ball off left
1270   ON IXB=35 TO 200
1280    E=2
1290    RETurn :REMark ball off right
1300   ON IXB=OXB
```

```
1310    IF IXB=18 THEN
1320     IF IYB>12 THEN E=3:RETurn :R
EMark ball drops on net
1330    END IF
1340   ON IXB=REMAINDER
1350    b=18
1360    a=IYB+(18-IXB)*(OYB-IYB)/(OXB
-IXB)
1370    IF a>=13 THEN
1380     IF IXB>OXB THEN
1390      SELect b=OXB TO IXB:E=3:RET
urn :REMark hit net
1400     ELSE
1410      SELect b=IXB TO OXB:E=3:RET
urn :REMark hit net
1420     END IF
1430    END IF
1440  END SELect
1450  IF IYB>18 THEN
1460   SELect ON IXB
1470    ON IXB=XPL-2 TO XPL+2
1480     E=4
1490     RETurn :REMark hit left padd
le
1500    ON IXB=XPR-2 TO XPR+2
1510     E=5
1520     RETurn :REMark hit right pad
dle
1530    ON IXB=2 TO 17
1540     E=6
1550     RETurn :REMark hit base on l
eft
1560    ON IXB=REMAINDER
1570     E=7
1580     RETurn :REMark hit base on r
ight
1590   END SELect
1600  END IF
1610  E=0
1620 END DEFine
1630 DEFine PROCedure BALL_OUT
1640  SELect ON E
1650   ON E=1
1660    IF S=0 THEN
1670     SCORE
1680    ELSE
1690     NO_SCORE
1700    END IF
1710   ON E=2
1720    IF S=1 THEN
1730     SCORE
1740    ELSE
```

```
1750     NO_SCORE
1760    END IF
1770  END SELect
1780 END DEFine

1790 DEFine PROCedure HIT_NET
1800  IF IYB>18 THEN IYB=18:YB=180
1810  VX=0
1820  YV=0
1830  SELect ON IXB
1840   ON IXB=2 TO 17
1850    XB=204
1860   ON IXB=19 TO 34
1870    XB=228
1880   ON IXB=18
1890    XB=204+24*RND(1)
1900  END SELect
1910  IXB=INT(XB/12)
1920 END DEFine
1930 DEFine PROCedure HIT_PADDLE
1940  SELect ON E
1950   ON E=4
1960    IF I=0 THEN
1970     H=H+1
1980    ELSE
1990     H=1
2000     I=0
2010    END IF
2020    SELect ON IPL
2030     ON IPL=4
2040      VY=-40
2050     VX=INT(RND(1,3)*-VY/4)
2060     ON IPL=128
2070      VY=-20
2080     VX=INT(RND(1,3)*-VY/4)
2090     ON IPL=REMAINDER
2100      VY=-VY
2110    END SELect
2120   ON E=5
2130    IF I=1 THEN
2140     H=H+1
2150    ELSE
2160     H=1
2170     I=1
2180    END IF
2190    SELect ON IPR
2200     ON IPR=128
2210      VY=-20
2220     VX=-INT(RND(1,3)*-VY/4)
2230     ON IPR=4
2240      VY=-40
2250     VX=-INT(RND(1,3)*-VY/4)
```

```
2260       ON IPR=REMAINDER
2270        VY=-VY
2280      END SELect
2290    END SELect
2300    YB=180
2310    IYB=18
2320  END DEFine

2330  DEFine PROCedure BALL_DROPPED
2340    SELect ON E
2350     ON E=6
2360      IF S=1 THEN
2370       SCORE
2380      ELSE
2390       NO_SCORE
2400      END IF
2410     ON E=7
2420      IF S=0 THEN
2430       SCORE
2440      ELSE
2450       NO_SCORE
2460      END IF
2470    END SELect
2480  END DEFine

2490  DEFine PROCedure BALL_PRINT
2500    CURSOR 12*OXB,10*OYB
2510    PAPER 1
2520    PRINT " ":REMark 1 space
2530    CURSOR 12*IXB,10*IYB
2540    INK 7
2550    PRINT "O"
2560    OXB=IXB
2570    OYB=IYB
2580  END DEFine

2590  DEFine PROCedure MOVE_PADDLE (XP
,C)
2600    CURSOR 12*(XP-3),190
2610    PAPER 1
2620    PRINT " ";:REMark 1 space
2630    PAPER C
2640    PRINT "     ";:REMark 5 spaces
2650    PAPER 1
2660    PRINT " ":REMark 1 space
2670  END DEFine

2680  DEFine PROCedure SCORE
2690    INK 1
2700    PAPER 7
2710    CLS
2720    CURSOR 72,100
2730    PRINT "THIS IS THE LATEST SCORE
;"
```

```
2740  SELect ON S
2750   ON S=0
2760    SCL=SCL+1
2770   ON S=1
2780    SCR=SCR+1
2790  END SELect
2800  CURSOR 168,130
2810  PRINT "YOU HAVE ";SCL
2820  CURSOR 168,150
2830  PRINT "I HAVE   ";SCR
2840  PAUSE 150
2850 END DEFine

2860 DEFine PROCedure NO_SCORE
2870  INK 7
2880  PAPER 1
2890  CLS
2900  CURSOR 180,50
2910  PRINT "NO SCORE"
2920  CURSOR 144,70
2930  PRINT "CHANGE SERVICE"
2940  PAUSE 150
2950  IF S=1 THEN
2960   S=0
2970  ELSE
2980   S=1
2990  END IF
3000 END DEFine

3010 REMark VARIABLES LIST
3020 REMark C=paddle colour
3030 REMark E=indicator of TEST_HIT
3040 REMark 0-nothing hit 1-off left
2-off right
3050 REMark 3-hit net 4-hit left padd
le 5-hit right paddle
3060 REMark 6-hit base to left 7-hit
base to right
3070 REMark G=effect of gravity
3080 REMark H=counts consecutive hits
of the ball
3090 REMark I=indicator of who last h
it ball
3100 REMark 0-player on left 1-player
on right
3110 REMark IPL,IPR=indicators for pl
ayer commands (left and right)
3120 REMark 0-no command 2-move paddl
e left 16-move paddle right
3130 REMark 128-hit ball softly 4-hit
ball hard
3140 REMark IXB,IYB=integer giving ba
ll print position (character coords)
```

```
3150 REMark OXB,OYB=previous time-ste
p value of IXB,IYB
3160 REMark S=service indicator 0-lef
t 1-right
3170 REMark SCR,SCL=score record for
left and right
3180 REMark VX,VY=ball speeds
3190 REMark XB=x-position of ball (CU
RSOR co-ordinates)
3200 REMark XP=x-position for paddle
 print
3210 REMark XPL,XPR=x-position of lef
t-hand end of left and right paddles
3220 REMark YB=y-position of ball (CU
RSOR co-ordinates)
```

Glossary

ALGORITHM: A step-by-step definition of the solution to a problem, often used as the basis of a program.

ARTIFICIAL INTELLIGENCE: A field of research into the simulation of aspects of human intelligence.

ASSEMBLER CODE: The set of binary instructions that the processor chip uses, a 'low-level' programming language.

AVERAGING PROCESS: The arithmetic average or 'mean' is obtained by adding all the values and then dividing the result by the number of values.

BASIC: Beginners All-purpose Symbolic Instruction Code.

BINOMIAL DISTRIBUTION: A common mathematical distribution.

BIT: A binary digit — a 0 or a 1 — the smallest piece of information a computer can handle.

BLOCK DIAGRAM: A coarse representation of a program, indicating the major subroutines, such as the main controlling routines, input and output routines, and specialist subroutines. Useful in program development.

CAUSAL DIAGRAM: A diagram which represents the possible responses of a system to different situations, and shows the links between possible system characteristics.

CLOSED-LOOP: A loop which operates automatically and is contained within the system itself.

COMPILER: Software that can translate a 'high-level' language into assembler code, which the computer executes.

COMPRESSION RATIO: The ratio of the volume of uncompressed air above the piston at the bottom of its cycle, to that of compressed air when the piston is at the top of its cycle, ie maximum to minimum volume.

COMPUTER-AIDED DESIGN (CAD): Using a computer to design products.

CONFIDENCE LEVEL: An indication of the confidence with which the results of a calculation or prediction should be viewed.

CONTROL THEORY: The mathematics and techniques used to analyse control systems.

CRITICAL-PATH ANALYSIS: An analysis of a number of dependent and independent items, designed to indicate which of the items are critical to the optimised execution of all the items.

DETERMINISTIC PROCESS: A process in which the most probable outcome is always chosen.

EMPIRICAL: Defined by experimental results; not theoretical.

EXPERT SYSTEM: A program that imitates some aspects of human knowledge.

EXTERNAL-EVENTS: Events which occur outside a defined system, and which may or may not affect the system.

FEEDBACK: The modification or control of a process or system by reference to its results or effects.

FINITE-ELEMENT MODEL: A model of a structure which represents the whole structure as a number of smaller, simple elements.

FLOWCHART: Diagrams which show the logical paths followed by a program. They can vary in detail, from a presentation of all the routines/processes and the order in which they are called, to an almost line-by-line description of a single routine or process.

FLUID DYNAMICS: A branch of physical science, concerned with the behaviour of fluids.

FORWARD DEPENDENCY: An indication of how events in the future will be affected by those happening in the present.

FUNCTION CHART: A chart or diagram which describes the components of a system in terms of their function within the system.

FUNCTIONAL DESCRIPTION: A description of a component of a system in terms of its function within that system.

HARDCOPY: Jargon for a printout.

INTEGER: A whole number, either positive or negative: 2, 100, -23 are integers, whereas -0. 5, 3/2, 3006. 10 are not.

INTERFACE: The interaction of a system with anything not defined as part of the system.

INTERNAL EVENTS: Events which occur within the defined system.

ITERATION: Repetition.

LIMITS: The definition of the extent of a system, in mathematical, spatial, logical, physical, or other appropriate terms.

LINEAR: A sequence of events in which one event is directly followed by another, ie in a line.

MACHINE CODE: See ASSEMBLER CODE

MAINFRAME: A large computer.

MARKOV CHAIN PROCESS: A method of handling probabilities so that a number of outcomes can be modelled.

MATRIX OPERATIONS: A matrix is simply a rectangular array of numbers or formulae. Matrices of the same dimensions can be added to or subtracted from each other by adding or subtracting corresponding elements, eg:

$$\begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 2+1 & 4+2 & 6+3 \\ 8+4 & 10+5 & 12+6 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{bmatrix} - \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 2-1 & 4-2 & 6-3 \\ 8-4 & 10-5 & 12-6 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Matrices can also be multiplied together, which can be very useful if the matrices contain definitions of various system-states and the probabilities of transition between one state and another (Markov Chain process).

Matrices can only be multiplied together if the number of rows in one is equal to the number of columns in the other, ie a 3 × 2 matrix may be multiplied by a 2 × 2 matrix, and the resultant matrix will have dimensions 3 × 2.

The multiplication process can be shown as follows:

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} * \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} [a*A]+[b*C] & [a*B]+[b*D] \\ [c*A]+[d*C] & [c*B]+[d*D] \\ [e*A]+[f*C] & [e*B]+[f*D] \end{bmatrix}$$

each element of the product matrix comes from a row in the first matrix combined with a column in the second matrix.

MEAN: See AVERAGING PROCESS.

MENU-DRIVEN: Program which offers a list of possible choices from which the user can select.

MONTE CARLO PROCESS: In the Monte Carlo process, a single response is selected randomly according to the probabilities of all possible responses. In order to obtain an average response, the process is repeated (each cycle is termed a pass), and the results of all passes added and then divided by the number of passes performed. See Chapters 4 and 10.

OPEN LOOP: A loop which requires something external to complete the loop. See Chapter 4.

OPERATING SYSTEM: Software that takes care of all the operations required by most programs, such as displaying characters on the screen, accepting characters input from the keyboard, and operating micro or disk drives.

OPERATIONAL ANALYSIS: The study of the operations of a system, eg traffic flow, critical-path analysis, and time and motion studies.

OPTIMUM: The best or most favourable; not necessarily the maximum.

PARALLEL PROCESS: A process in which more than one outcome or state can be represented at any one time, eg Markov chain.

PERIPHERALS: Devices that are separate from the main computer, such as printers and joysticks.

POISSON DISTRIBUTION: A common mathematical distribution.

PREDICTION: A forecast of the future based on a knowledge of past events.

PROBABILITY: The probability of an event happening is defined as:

The number of outcomes leading to the event

The total number of possible outcomes

Assuming that all outcomes are equally likely. *Note:*

1) The probability of an event lies between 0 and 1. The probability of a certain event is 1, and that of an impossible event is 0.
2) If the probability of an event happening is p, then the probability of it not happening is 1-p.
3) If two independent events have the probabilities p1 and p2, then the probability of both the events happening is p1*p2.

REAL-TIME PROCESS: A process for which the simulated time-step is identical to the actual time taken.

RELIABILITY: A measure of the failure of a system, usually as a probability or percentage, derived from the results of tests of the system or a record of failures encountered whilst using the system.

RELIABILITY GROWTH: An indication of trends in reliability, generally produced for a new product or piece of hardware, in order to estimate when it will be sufficiently reliable to be manufactured.

RICH PICTURES: Usually a freehand drawing depicting parts of a system.

SENSITIVITY ANALYSIS: Series of tests designed to indicate how responsive a simulation is to its various parameters.

SERIAL OPERATIONS: Operations that take place consecutively.

SUB-SYSTEM

SYSTEMS See Chapter 2.

SYSTEMS ANALYSTS

SYSTEM-STATE: Definition of the attributes of a system such that its state can be identified. See Chapters 4 and 9.

SYSTEM-STATE MATRIX: An array (matrix) which contains information about various attributes of a system, and uniquely defines a number of possible states of the system. See Chapters 4 and 9.

THERMODYNAMICS: A branch of physical science concerend with the effects of heat energy.

TIME-STEP: Time between consecutive system states as represented by the simulation.

TIME-TRANSIENT: Changing with time.

TRANSFORMATION MATRIX: An array (matrix) which contains the probabilities of a system changing from one state to another. Multiplication of the transition matrix with the system-state matrix will give an update of the system-state. See Chapters 4 and 9.

UTILISATION: A measure of the actual use of available resources.

VERIFICATION: Confirmation of correctness.

Other titles from Sunshine

SPECTRUM BOOKS

Artificial Intelligence on the Spectrum Computer Keith & Steven Brain	ISBN 0 946408 37 8	**£6.95**
Spectrum Adventures Tony Bridge & Roy Carnell	ISBN 0 946408 07 6	**£5.95**
Machine Code Sprites and Graphics for the ZX Spectrum John Durst	ISBN 0 946408 51 3	**£6.95**
ZX Spectrum Astronomy Maurice Gavin	ISBN 0 946408 24 6	**£6.95**
Spectrum Machine Code Applications David Laine	ISBN 0 946408 17 3	**£6.95**
The Working Spectrum David Lawrence	ISBN 0 946408 00 9	**£5.95**
Inside Your Spectrum Jeff Naylor & Diane Rogers	ISBN 0 946408 35 1	**£6.95**
Master your ZX Microdrive Andrew Pennell	ISBN 0 946408 19 X	**£6.95**

COMMODORE 64 BOOKS

Graphic Art for the Commodore 64 Boris Allan	ISBN 0 946408 15 7	**£5.95**
DIY Robotics and Sensors on the Commodore Computer John Billingsley	ISBN 0 946408 30 0	**£6.95**
Artificial Intelligence on the Commodore 64 Keith & Steven Brain	ISBN 0 946408 29 7	**£6.95**
Machine Code Graphics and Sound for the Commodore 64 Mark England & David Lawrence	ISBN 0 946408 28 9	**£6.95**
Commodore 64 Adventures Mike Grace	ISBN 0 946408 11 4	**£5.95**
Business Applications for the Commodore 64 James Hall	ISBN 0 946408 12 2	**£5.95**
Mathematics on the Commodore 64 Czes Kosniowski	ISBN 0 946408 14 9	**£5.95**
Advanced Programming Techniques on the Commodore 64 David Lawrence	ISBN 0 946408 23 8	**£5.95**

Commodore 64 Disk Companion
David Lawrence & Mark England ISBN 0 946408 49 1 **£7.95**
The Working Commodore 64
David Lawrence ISBN 0 946408 02 5 **£5.95**
Commodore 64 Machine Code Master
David Lawrence & Mark England ISBN 0 946408 05 X **£6.95**
Programming for Education on the Commodore 64
John Scriven & Patrick Hall ISBN 0 946408 27 0 **£5.95**

ELECTRON BOOKS

Graphic Art for the Electron Computer
Boris Allan ISBN 0 946408 20 3 **£5.95**
Programming for Education on the Electron Computer
John Scriven & Patrick Hall ISBN 0 946408 21 1 **£5.95**

BBC COMPUTER BOOKS

Functional Forth for the BBC Computer
Boris Allan ISBN 0 946408 04 1 **£5.95**
Graphic Art for the BBC Computer
Boris Allan ISBN 0 946408 08 4 **£5.95**
DIY Robotics and Sensors for the BBC Computer
John Billingsley ISBN 0 946408 13 0 **£6.95**
Essential Maths on the BBC and Electron Computer
Czes Kosniowski ISBN 0 946408 34 3 **£5.95**
Programming for Education on the BBC Computer
John Scriven & Patrick Hall ISBN 0 946408 10 6 **£5.95**
Making Music on the BBC Computer
Ian Waugh ISBN 0 946408 26 2 **£5.95**

DRAGON BOOKS

Advanced Sound & Graphics for the Dragon
Keith & Steven Brain ISBN 0 946408 06 8 **£5.95**
Artificial Intelligence on the Dragon Computer
Keith & Steven Brain ISBN 0 946408 33 5 **£6.95**
Dragon 32 Games Master
Keith & Steven Brain ISBN 0 946408 03 3 **£5.95**
The Working Dragon
David Lawrence ISBN 0 946408 01 7 **£5.95**

The Dragon Trainer
Brian Lloyd ISBN 0 946408 09 2 **£5.95**

ATARI BOOKS

Atari Adventures
Tony Bridge ISBN 0 946408 18 1 **£5.95**

Writing Strategy Games on your Atari Computer
John White ISBN 0 946408 22 X **£5.95**

GENERAL BOOKS

Home Applications on your Micro
Mike Grace ISBN 0 946408 50 5 **£6.95**

Sunshine also publishes

POPULAR COMPUTING WEEKLY

The first weekly magazine for home computer users. Each copy contains Top 10 charts of the best-selling software and books and up-to-the-minute details of the latest games. Other features in the magazine include regular hardware and software reviews, programming hints, computer swap, adventure corner and pages of listings for the Spectrum, Dragon, BBC, VIC 20 and 64, ZX 81 and other popular micros. Only 40p a week, a year's subscription costs £19.95 (£9.98 for six months) in the UK and £37.40 (£18.70 for six months) overseas.

DRAGON USER

The monthly magazine for all users of Dragon microcomputers. Each issue contains reviews of software and peripherals, programming advice for beginners and advanced users, program listings, a technical advisory service and all the latest news related to the Dragon. A year's subscription (12 issues) costs £10 in the UK and £16 overseas.

MICRO ADVENTURER

The monthly magazine for everyone interested in Adventure games, war gaming and simulation/role-playing games. Includes reviews of all the latest software, lists of all the software available and programming advice. A year's subscription (12 issues) costs £10 in the UK and £16 overseas.

COMMODORE HORIZONS

The monthly magazine for all users of Commodore computers. Each issue contains reviews of software and peripherals, programming advice for beginners and advanced users, program listings, a technical advisory service and all the latest news. A year's subscription costs £10 in the UK and £16 overseas.

For further information contact:
Sunshine
12–13 Little Newport Street
London WC2R 3LD
01-437 4343

Telex: 296275

Printed in England by Commercial Colour Press, London E7.

NOTES

NOTES

NOTES

Do you want to make your computer think it's a dog? Do you want to know what the stock market is going to do tomorrow? Do you want to fly a jumbo jet in your living room?

Here is a book written for people who know what they want from computers but don't know how to get started. This book leads you logically through the world of computer simulations. It builds up your knowledge of the techniques used by professionals to produce useful programs.

This book will give you the ability to look at any problem and apply a method of analysis that allows you to develop your own simulations. With the use of simple examples, John Cochrane snows you how to make the most of the considerable potential of the Sinclair QL, and also how to work within its limitations.

John Cochrane is a consultant engineer. He is a very experienced programmer and an expert in the field of computer simulations. His experience stretches back for over fifteen years and includes work on many major simulations. He has recently completed programs for Sinclair and Timex microcomputers. He 'relaxes' by teaching aerobics.

GB £ NET +006.95
ISBN 0-946408-45-9
00695
9 780946 408450

SUNSHINE
ISBN 0 946408 45 9

£6.95 net

JOHN COCHRANE

SIMULATION TECHNIQUES ON THE QL

SUNSHINE