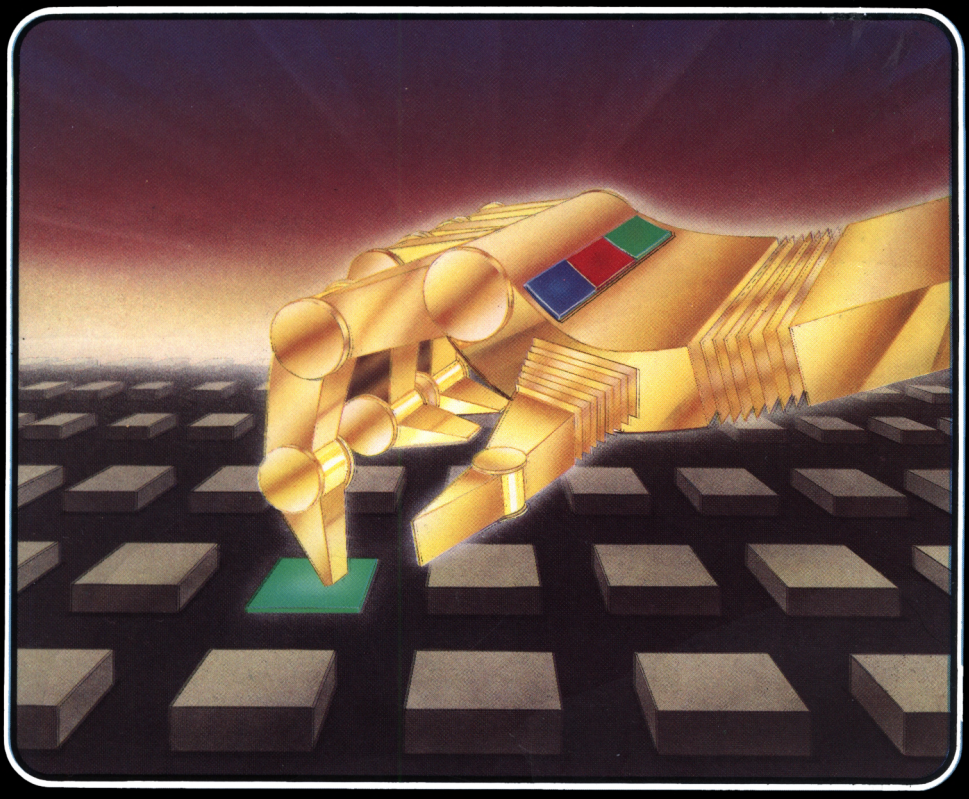
AMSTRADS and Artificial Intelligence



Patrick Hall

SIGMA PRESS

Amstrads and Artificial Intelligence

Patrick Hall

Σ
SIGMA
PRESS

ISBN 1 85058 043 X

Published by:

SIGMA PRESS
98a Water Lane
Wilmslow
Cheshire
U.K.

Distributors:

U.K., Europe, Africa:
JOHN WILEY & SONS LIMITED
Baffins Lane, Chichester
West Sussex, England

Australia:
JOHN WILEY & SONS INC.
GPO Box 859, Brisbane
Queensland 40001
Australia

Acknowledgments

CPC-464, CPC-664 and CPC-6128 are trademarks of Amstrad Consumer Electronics PLC

Printed in Malta by Interprint Ltd

Contents

CHAPTER 1

In the beginning: the history of artificial intelligence

'Electronic Brains'

In the first period of their growth and development, in the 1950s, computers were frequently referred to as 'electronic brains', a catch phrase which reinforced a basic misconception of the true nature of the machines. The source of the error probably stemmed from the image of the brain as 'an amazing telephone exchange', which had earlier appeared frequently in popular books on biology. How natural, then, to assume that this new piece of even more complicated electrical wiring should be a better analogy. Soon statements were appearing in newspapers which estimated how big an electronic brain would have to be in order to compete with the natural variety. It certainly gave many people the satisfaction of knowing that their cranium could outperform the Royal Albert Hall, while at the same time the confidence that science had at last solved the problem and that we knew how it all worked.

Possibly the same jingoism overflowed into computer science itself, because with buoyant confidence early programs were soon forthcoming intended to simulate aspects of human thought. The term 'Artificial Intelligence' was invented by John McCarthy, while Allen Newell and Hebert Simon wrote one of the first convincing A.I. programs, the General Problem Solver. This could deal remarkably well with highly formalised situations. Other programs were developed which could operate convincingly in areas conventionally assumed to require human intelligence and optimistic predictions were made about the future of the new science. However it began to be realised that, whereas a computer program could be written to perform calculus, programs intended to deal with more mundane issues were not particularly successful. A new conception of intelligence evolved, as researchers discovered that the truly 'intelligent' things of which human beings were capable were the ones mastered by infants rather than

undergraduates. Decyphering visual images of the world, controlling limbs and balance, understanding language, all of these are intelligent activities which happen so automatically that we are not conscious of them.

AI and Common Sense

The real task faced by A.I. workers was therefore introducing common sense into their programs. This began to take place as programs were written to play games. Here the program had to be able to investigate a situation, represent for itself the information it had acquired and make decisions about what could be attempted next. Game playing was accompanied by the development of 'expert systems'. In such a program, information is stored on a particular area of human knowledge in such a way that the computer can use it effectively in asking questions, extending its grasp of the situation and giving practical advice. Cross fertilisation between the two fields of A.I. occurred as similar methods of knowledge representation and 'heuristic' rules for operating on the knowledge were applicable to both game playing and expert systems. The latter area of A.I. attracted commercial interest as knowledge based systems could be used practically in the real world.

Frontiers of AI Research

Research in A.I. continues in many areas. Computer vision is a particularly difficult, but stimulating area of investigation, with immediate applications in the control of industrial robots. Language translation programs, after an unpromising beginning, are now being employed to help in the administration of multicultural organisations, like the European Economic Community. Slow but consistent progress is made in the development of robotic systems which one day will be of immense usefulness.

There is another aspect of A.I. in which programs can be used to help test models of human cognition. The psychologist, or cognitive scientist, can try out new ideas in almost laboratory conditions and modify theories accordingly. Sometimes this process can be reversed and discoveries about the human brain used in new approaches in A.I. software. An example of this has occurred recently in computer vision, where an insight into the way the brain processes sterescopic information has inspired a new algorithm in a vision program.

The Programs in this Book

A.I. will be an important feature of the proposed fifth generation of computers, which are expected to possess far greater processing power and an enhanced ability to understand ordinary language. Some might argue it

was worth waiting for these machines to emerge from whichever country succeeds in making them first. However, this would remove the excitement of working, albeit at the micro level with an Amstrad computer, in a relatively new area of scientific research. It must be remembered that the home microcomputer is as powerful as the mainframes upon which the early A.I. programs were first evaluated. In this book, ten programs are provided which cover a broad area of A.I. The order in which material is introduced is intended to emphasise particular themes in A.I. and also to allow the programming techniques employed to progress in difficulty. A full commentary follows every program and algorithms are explained in detail.

The first two programs illustrate automatic writing, or the debatable area of computer creativity. ODE, an acronym for 'Obsequious Dedication to Everybody', produces rhyming verse praising the personal attributes and traits of, it is hoped, a suitably impressed and computer-naive, friend of the programmer! Then computer prose is generated by BARD, the 'Basic Algorithm for Random Development'. Understanding natural language is explored with a German/English translation program, INGA, an 'INtelligent German Algorithm'. After this, VALID, or 'Veracity Assessed by Logical Inference and Deduction', investigates the way in which knowledge can be stored within a program.

An introduction to expert systems is provided by AMY, 'Analytical Method of Yielding information'. Two games playing programs follow. The game of Hex is played by the program HEX, 'Heuristics EXamined', and noughts and crosses by the American-named TICTACTOE, 'Testing Intelligent Computer TACTics Opposing Enemy'. An example of a computer vision system is given by PIPPA, 'Programmed Identification of Polygon Properties and Attributes'. Finally, two robotics programs are included in the book. MINOTAUR, 'MINimum Operational Time Acquiring Ultimate Route', is a maze solving program and LUCIE, 'Location Under Controlled Investigation of Environment', gives a three-dimensional graphic animation of a domestic robot (actually a robot cat) exploring a room full of furniture as it tries to find a life-giving power socket.

CHAPTER 2

Planned spontaneity: computer creativity?

The possibility that computers might one day be capable of simulating human speech, and perhaps thought, is accepted by many people who, nevertheless, baulk at the suggestion that original and imaginative action might also be attributed to them. The very concept of computer creativity can seem distasteful, as if it marks the last barrier between the human and the machine.

Computers and The Arts

Nevertheless, computers have been used, in a limited way, in the Arts. In particular they have appealed to certain contemporary composers. Strength is, possibly, given to this position by the fact that a Russian composer, Joseph Schillinger, wrote a book called 'The Mathematical Basis of the Arts' in 1942. In this, many of the characteristics of later, computer-based musical scores were first enunciated. Significantly, the book was written before the emergence of the electronic computer from its war time backroom, thus preventing any cynical assumption that Schilllinger was simply exploiting a new technical advance.

At the University of Illinois in the 1950s, Professor Hiller used ILLIAC, a first generation machine, to compose various pieces, including a suite for string quartet. Better known are the works of John Cage and Xenakis. Cage's 'HPSCHD' contrasts fifty one channels of computer generated music against seven of amplified harpsichords. Xenakis, who studied under Honegger and Milhaud and who is therefore closer to the mainstream of contemporary European music, has experimented with compositional techniques so complicated that a computer is required to work out the mathematics involved. This belies the immediacy of his compositions' appeal. Similarly, the leading French composer, Pierre Boulez, at IRCAM, the Institut de Recherche et de Coordinatiom Acoustique Musique, has encouaged students to learn computer programming.

The use of computers in music is not equivalent to computer creativity. Instead the composer remains decisively in control, retaining through the design of the program a critical over-view of what is being produced. A parallel can be drawn with simple language-generating programs which can be used to produce a variety of sentence structures around given themes. By careful pruning of the results, the computer can appear occasionally to have written something of merit. An example of such a program is given here.

```
                ODE TO SARAH

MY SARAH, SARAH, SARAH,
SO STRANGELY SLY,
NO WONDER SINCE YOU'VE GOT
A VERY ICY EYE.
OH SARAH YOUR VERY
LOVELY HAIR,
IS RATHER LIKE A
PLIGHT THAT'S FAIR.
MY SARAH, SARAH, SARAH,
SO STRANGELY HIGH,
NO WONDER SINCE YOU'VE GOT
A VERY TRAGIC REPLY.
THE MAGICAL SPELL COULD NEVER
BE SO DEAR,
AS SARAH IN THE MORNING
WHEN YOU DISAPPEAR.
```

ODE

This program writes verse in rhyming couplets. The screen dump gives a typical example of the quality of its work. Because of the limited nature of subject material, making wry comments about the character and physical traits of the person to whom the ode is dedicated, the program can, occasionally, get things completely right and appear more sophisticated than it actually is. Hence the last two lines of 'Ode to Sarah' seem to possess a resigned optimism.

```
1000 REM ODE - PAT HALL, 2/86
1010 REM CONTROL ROUTINE
1020 GOSUB 1070:GOSUB 1150:WHILE 1>0
1030 GOSUB 1310:GOSUB 1350:FOR I=1 TO 4
1040 GOSUB 1410:NEXT:WEND
1050 REM
1060 REM INITIALISATION
1070 BORDER 0:MODE 1:INK 0,26:INK 1,2
1080 DIM R(3,5):FOR I=1 TO 3:FOR J=1 TO
     5:READ R(I,J):NEXT:NEXT
1090 DIM R$(3,5,7):FOR I=1 TO 3:FOR J=1
     TO 5:FOR K=1 TO R(I,J):READ R$(I,J,
     K):NEXT:NEXT:NEXT
1100 DIM N$(10):FOR I=1 TO 10:READ N$(I)
     :NEXT:DIM A$(10):FOR I=1 TO 10
1110 READ A$(I):NEXT: RANDOMIZE TIME
1120 RETURN
1130 REM
1140 REM DESCRIBE
1150 BORDER 2:PEN 1:PAPER 2:CLS:WINDOW
     #1,1,40,1,3:PAPER #1,0:CLS #1
1160 WINDOW #2,1,40,23,25:PAPER #2,0:CLS
     #2:LOCATE 18,2:PRINT"< ODE >"
1170 LOCATE 3,9:PRINT "USING THIS";
1180 PRINT" PROGRAM YOU CAN WRITE A"
1190 PRINT"  POEM DEDICATED TO A";
1200 PRINT" CLOSE FRIEND. THE"
1210 PRINT"  AMSTRAD 6128 WILL";
1220 PRINT" SELECT WORDS WHICH"
1230 PRINT"  RHYME." : PRINT
1240 PRINT"  ALL YOU HAVE TO DO";
```

```
1250 PRINT" IS TYPE THE NAME"
1260 PRINT"  OF THE PERSON TO WHOM";
1270 PRINT" YOU WISH TO"
1280 PRINT"  OFFER YOUR ODE.":RETURN
1290 REM
1300 REM PROMPT
1310 CLS #2:LOCATE #2,16,2:PRINT #2,
     "< SPACE >":K=0:WHILE K=0:IF INKEY(
     47)=0 THEN K=1
1320 WEND:CLS #2:RETURN
1330 REM
1340 REM DEDICATION
1350 L=9:WHILE L>8:CLS:LOCATE 15,10
1360 PRINT"TYPE NAME":LOCATE 15,12
1370 INPUT D$:L=LEN(D$):WEND:CLS:LOCATE
     16-L/2,3:PRINT"ODE TO ";D$:PRINT
1380 RETURN
1390 REM
1400 REM SELECT COUPLET
1410 S=5:GOSUB 1440:RF=N:GOSUB 1440:CT=N
     :ON CT GOSUB 1470,1520,1580,1620,
     1670:RETURN
1420 REM
1430 REM RANDOM
1440 N=INT(RND(1)*S)+1:RETURN
1450 REM
1460 REM COUPLET 1
1470 PRINT TAB(3)
     "I LOVE TO WATCH YOUR ";:S=10:GOSUB
     1440:PRINT TAB(3)A$(N);" ";
1480 S=R(1,RF):GOSUB 1440:PRINT R$(1,RF,
```

```
     N);",":PRINT TAB(3)
     "DESPITE THE FACT YOU";:PRINT TAB
     (3)"SEEM TO ";
1490 S=R(3,RF):GOSUB 1440:PRINT R$(3,RF,
     N);".":RETURN
1500 REM
1510 REM COUPLET 2
1520 PRINT TAB(3) "THE ";:S=10:GOSUB
     1440:PRINT A$(N);" ";:GOSUB 1440
1530 PRINT N$(N);" REMINDS":PRINT TAB (3
     )"ME OF YOUR ";:S=R(1,RF):GOSUB
     1440:PRINT R$(1,RF,N);",":PRINT TAB
     (3) "EVEN THOUGH YOUR ";:S=5:GOSUB
     1440:RN=N:S=R(1,RN):GOSUB 1440
1540 PRINT R$(1,RN,N):PRINT TAB (3)
     "IS RATHER ";:S=R(2,RF):GOSUB 1440
1550 PRINT R$(2,RF,N);".":RETURN
1560 REM
1570 REM COUPLET 3
1580 PRINT TAB (3)"THE ";:S=10:GOSUB
     1440:PRINT A$(N);" ";:GOSUB 1440:
     PRINT N$(N);" COULD NEVER":PRINT
     TAB (3)"BE SO ";:S=R(2,RF):GOSUB
     1440
1590 PRINT R$(2,RF,N);",":PRINT TAB (3)
     "AS ";D$;" IN THE MORNING":PRINT
     TAB (3)"WHEN YOU ";:S=R(3,RF):GOSUB
     1440:PRINT R$(3,RF,N);".":RETURN
1600 REM
1610 REM COUPLET 4
1620 PRINT TAB (3)"OH "; D$;" YOUR VERY"
```

```
     :PRINT TAB (3)"LOVELY ";:S=R(1,RF):
     GOSUB 1440:PRINT R$(1,RF,N);","
1630 PRINT TAB (3)"IS RATHER LIKE A":S=5
     :GOSUB 1440:RN=N:S=R(1,RN):GOSUB
     1440:PRINT TAB (3)R$(1,RN,N);
     " THAT'S ";:S=R(2,RF):GOSUB 1440
1640 PRINT R$(2,RF,N);".":RETURN
1650 REM
1660 REM COUPLET 5
1670 PRINT TAB (3)"MY ";D$;", ";D$;", ";
     D$;",":PRINT TAB (3)"SO STRANGELY "
     ;:S=R(2,RF):GOSUB 1440:PRINT R$(2,
     RF,N);",":PRINT TAB (3)
     "NO WONDER SINCE YOU'VE GOT":PRINT
     TAB (3)"A VERY ";:S=10:GOSUB 1440
1680 S=10:GOSUB 1440:PRINT A$(N);" ";
1690 S=R(1,RF):GOSUB 1440:S=R(1,RF)
1700 GOSUB 1440:PRINT R$(1,RF,N);"."
1710 RETURN
1720 REM
1730 REM DATA
1740 DATA 6,5,4,5,7,6,3,3
1750 DATA 3,6,4,6,6,7,7
1760 DATA SIGHT,PLIGHT,HEIGHT,SPITE
1770 DATA MIGHT,NIGHT,HAIR,STARE
1780 DATA FLAIR,AIR,SNAIR,FACE,BASE
1790 DATA GRACE,WAIST,EYE,THIGH,CRY
1800 DATA SIGH,REPLY,CHEER,EAR,TEAR
1810 DATA REAR,IDEA,FEAR,SNEER
1820 DATA WHITE,SLIGHT,RIGHT,QUITE
1830 DATA LIGHT,TIGHT,FAIR,BARE
```

```
1840 DATA RARE,CHASTE,GRACED,LACED
1850 DATA HIGH,SLY,NIGH,NEAR,CLEAR
1860 DATA SINCERE,INSINCERE,MERE,DEAR
1870 DATA FIGHT,WRITE,HYPE,BITE
1880 DATA COMPARE,CARE,DESPAIR,DARE
1890 DATA TEAR,IMPAIR,RACE,CHASE
1900 DATA DISPLACE,DISGRACE,PACE
1910 DATA TASTE,GO BY,DEFY,ESPY,LIE
1920 DATA IMPLY,PRY,DECRY,HEAR,LEER
1930 DATA PEER,JEER,STEER,VEER
1940 DATA DISAPPEAR,MOUNTAIN,MOON
1950 DATA LAKE,BREATH,RIVER,MIST,FIRE
1960 DATA ICE,SPELL,GLACIER
1970 DATA PURPLE,MISTY,ICY,FIERY
1980 DATA TEARFUL,MAGICAL,DEVOUT
1990 DATA HELPLESS,TRAGIC,ICE-BLUE
```

Commentary on ODE

ODE, like the other programs contained in the book, is written in a modular fashion, with a control routine calling various other routines as required. Lines 1020 - 1040 form this control routine. At line 1020 it calls the initialisation routine to set up the arrays of words needed by the program. Line 1020 also uses 'describe' to explain the operation of ODE to the user. Then the WHILE/WEND loop, between lines 1020 - 1040, allows the computer to generate verse by calling the routines 'prompt', 'dedication' and, within the FOR loop, the routine 'select-couplet'.

	1	2	3
1	SIGHT	SLIGHT	FIGHT
2	HAIR	FAIR	CARE
3	FACE	CHASTE	DISGRACE
4	EYE	SLY	PRY
5	EAR	DEAR	JEER

Initialisation routine (Lines 1060 - 1130):

First line 1070 selects MODE 1 and sets INK 0 to white and INK 1 to blue. Then the routine places the vocabulary held as DATA between lines 1760 - 1990 into three separate arrays so that words can be selected easily by the five routines, 'couplet-1' - 'couplet-5'.

The largest of these arrays is R$(I, J, K), initialised at line 1090. It stores nouns, adjectives and verbs, further classified into five categories of rhyme. The way in which this vocabulary is organised can best be understood by looking at the table. The first column contains nouns, the second adjectives and the third verbs. The first subscripted variable of the array represents this choice. The five rows of the table allow for five different types of rhyme, reflected in the value of the second variable of the array. The third variable is the number of words in each particular section of the table, eg: nouns which rhyme with the word 'sight'.

Before the words can be placed into the R$ array, the number in each section has to be established. These numbers are held as DATA at lines 1740, 1750 and are READ into the R(I, J) array by the two nested FOR loops at line 1080. After this the three nested FOR loops at line 1090 need to use these R(I, J) values in order to fill the R$ array with the rhyming vocabulary.

An array, N$, of non-rhyming nouns is set up by line 1100 and filled from DATA by the FOR loop. Similarly lines 1100 - 1110 place non-rhyming adjectives into the A$ array. The random number generator is initialised with RANDOMIZE TIME.

Describe routine (Lines 1140 - 1290):

Here the operation of the program is explained to the user. Line 1150 chooses a blue border for the display. Then the screen is cleared to cyan and a white band for the title placed across the top of the screen by setting suitable coordinates for WINDOW #1. WINDOW #2 is used to place a similar band across the bottom of the screen by line 1160. The program title is PRINTed in blue text with a cyan background by line 1160 as well.

The description of the program is placed on the screen by lines 1170 - 1280. After this, the routine ends and the program moves into the WHILE/WEND loop in the control routine. This means that the routine 'prompt' leaves the explanation PRINTed by 'describe' on the screen until the space-bar is pressed.

Prompt routine (Lines 1300 - 1330):

This routine is used to prevent further execution of the program until the user presses the space-bar. In this way the contents of the display can be examined carefully before the program is allowed to refresh the screen. First, line 1310 PRINTs the message ‹SPACE› in the centre of window #2, thus informing the user what has to be done before the program will continue.

Then the WHILE/WEND loop between lines 1310, 1320 scans the keyboard. When INKEY(47) at line 1310 detects that the space-bar has been pressed, the loop is able to terminate because of the value K = 1 at line 1320. Window #2 is then cleared and the routine ends.

Dedication routine (Lines 1340 - 1390):

Here the name is requested of the person for whom the ode is intended. The WHILE/WEND loop asks for the name at line 1360. The string INPUT at 1370, D$, is checked for length and only names of fewer than nine characters

will allow the loop to end. This is to prevent a possible line of verse generated by the subsequent routine 'couplet-5' from spilling over on to the following line and thereby spoiling the format of the display.

Line 1370 then clears the screen and places the title of the ode centrally at the top of the display. The expression employed, 16−L/2, ensures that the position of the title is adjusted according to the length of the name chosen.

Select couplet routine (Lines 1400 - 1420):

This is the routine called four times by the FOR loop in the program's control routine. On each call, the routine chooses a rhyme for the couplet that it generates. The rhyme is established by the rhyme flag, RF, generated as a random number between 1 and 5 by `GOSUB 1440` at line 1410. Each of the five possible values of RF corresponds to one of the five categories of rhymes stored in the array.

Line 1410 also chooses a random value of 1-5 for the variable CT. This value is then used in the ON GOSUB routine to call one of the five routines which produce a rhyming couplet to add to the ode.

Random routine (Lines 1430-1450):

Line 1440 produces a random integer in the range 1 - 5.

Couplet 1 routine (Lines 1460-1500):

Here two lines of verse are generated according to the pattern:

```
I LOVE TO WATCH YOUR
FIERY SPITE,
DESPITE THE FACT YOU
SEEM TO BITE.
```

Each line is PRINTed on the screen in two sections to avoid long lines being accidentally split when displayed.

Although the poetic quality of such language could be questioned, the program does succeed in producing reasonable rhyme and metre. This is a consequence of the vocabulary stored as DATA, the use of the variable RF to control the rhyme and the overall structure of the lines defined in the routine.

Line 1470 generates the first half of the first line of the couplet and adds the final adjective, selected from the A$ array by the use of the random number

N returned by GOSUB 1440. The selection of the rhyming noun to end the line, from the R$ array, by line 1480 is more complicated, however, because the variable RF must be included. Selection of words from the R$ array is governed by three subscripted variables. The value of the first variable, 1, determines that a noun is chosen from the array. The second variable, RF, chooses the rhyme. The total number of nouns with this particular rhyme is then given by the value of the R array element, R(1, RF). This means that a random selection from these words will be obtained by setting S = R(1, RF) for GOSUB 1440 and that the number, N, returned will be the third subscripted variable. The noun found in the array is thus R$(1, RF, N).

The second line of the couplet is similarly constructed by lines 1480, 1490.

Couplet 2 routine (Lines 1510-1560):

This routine generates two lines which rhyme like this:

```
THE PURPLE MOUNTAIN REMINDS
ME OF YOUR EYE,
EVEN THOUGH YOUR IDEA
IS RATHER SLY.
```

At line 1530 a random number, RN, is generated to select a noun from the R$ array, at line 1540, which does not have to match the rhyme of the couplet.

Couplet 3 routine (Lines 1570 - 1600):

Here the couplet produced takes the form

```
THE MISTY MOON COULD NEVER
BE SO CLEAR,
AS PAULINE IN THE MORNING
WHEN YOU JEER.
```

Couplet 4 routine (Lines 1610-1650):

The structure of the couplet is:

```
OH MICHELLE YOUR VERY
LOVELY HAIR,
IS RATHER LIKE A
SIGHT THAT'S RARE.
```

Couplet 5 (Lines 1660-1720):

This routine generates a couplet according to the structure:

```
MY RACHEL, RACHEL, RACHEL,
SO STRANGELY GRACED,
NO WONDER SINCE YOU'VE GOT
A VERY TRAGIC FACE.
```

Data (Lines 1730-1990):

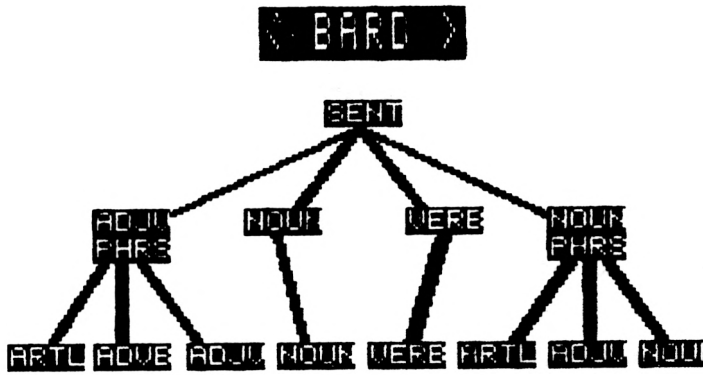
All the information needed for the three word arrays is stored between lines 1740 - 1990. Obviously the nature of the poetry produced will depend upon an imaginative choice of words in this section of the program.

BARD

ODE produced verse by having a series of couplet writing routines. A similar technique is used in the second program, BARD. Here the computer writes stories, using a number of different sentence types and a larger vocabulary than ODE. The display, as shown in the first screen dump, is also more ambitious. It details the sentence type currently in use as a parsing tree, with the parts of speech highlighted above the sentence that has been generated by the tree. After ten such sentences have been composed, they are all repeated to form a 'story'. A pretty-printing routine is incorporated to create a tidy display and to avoid wrap around with any longer words that are chosen.

The DATA used allows for five types of story, fairy, horror, romantic, crime and science fiction. The user can select the initial story type and also a 'semantic drift' factor which supervises the possible modification of the vocabulary chosen from one classification to another. Randomness is, of course, involved to promote 'creativity'.

Unlike ODE, the sentence structures are not part of the main program. Instead they are held in coded form as DATA at the end of the listing. It would be fairly easy to construct other types of sentences and include these in BARD as well. Details of how to do this are to be found in the commentary which follows the listing.



TYPE: 1 DRIFT: 1 1 1 1 1 1 1 1

THE FOOLISHLY MISCHIEVOUS KNIGHT
HYPNOTISES THE GOLDEN FAIRY.

BARD

TERRIFYINGLY BEHIND THE FOREST
THE CREEPY SKELETON KILLS. THE
SKULL KILLS A DEMON. A SKULL
HAUNTS THE SKELETON. A WEREWOLF
FRIGHTENS THE FOREST. A GHOST
DEVOURS A CORPSE. A CREEPY VAMPIRE
ATTACKS THROUGH A DEMON. THE
GHOSTLY SKULL BLEEDS THE CREEPY
SKULL. THE FIENDISHLY
TRANSYLVANIAN FOG TERRIFIES A
GROTESQUE FOREST. A CREEPY SKULL
DECAPITATES THROUGH THE DEMON. THE
DEMON BITES ALTHOUGH A SKELETON
HAUNTS.

```
1000 REM BARD - PAT HALL, 1/86
1010 REM CONTROL ROUTINE
1020 GOSUB 1060:GOSUB 1120:WHILE TIME>0:
     GOSUB 1250:GOSUB 1290:GOSUB 1440
1030 WEND
1040 REM
1050 REM INITIALISATION
1060 BORDER 0:MODE 1:INK 0,0:INK 1,26:
     INK 2,20:INK 3,2:DIM SEN$(10)
1070 DIM NL(15):DIM X(15):DIM Y(15)
1080 DIM NL$(12):FOR I=1 TO 12:READ NL$(
     I):NEXT:DIM VC$(4,5,10):FOR I=1 TO
     4:FOR J=1 TO 5:FOR K=1 TO 10:READ
     VC$(I,J,K):NEXT:NEXT:NEXT:DIM W(3)
1090 FOR I=1 TO 3:READ W(I):NEXT: DIM
     V$(3,9):FOR I=1 TO 3:FOR J=1 TO W(I
     ):READ V$(I,J):NEXT:NEXT:DIM WD$(15
     ):DIM VO$(5):FOR I=1 TO 5:READ VO$(
     I):NEXT:GRAPHICS PEN 3:RETURN
1100 REM
1110 REM DESCRIBE
1120 BORDER 2:PEN 3:PAPER 2:CLS:WINDOW
     #1,1,40,1,3:PAPER #1,1:CLS #1:
     WINDOW #2,1,40,4,22:PAPER #2,2:
     WINDOW #3,1,40,23,25:PEN #3,0:PAPER
     #3,1:CLS #3
1130 LOCATE 17,2:PRINT"< BARD >"
1140 LOCATE 3,10:PRINT"THIS PROGRAM";
1150 PRINT" WILL COMPOSE DIFFERENT"
1160 PRINT"  STORIES AT RANDOM.";
1170 PRINT" FIRST, YOU SELECT"
```

```
1180 PRINT"  THE BASIC STORY TYPE";
1190 PRINT" AND THE 'DRIFT'"
1200 PRINT"  THROUGH THE STORED";
1210 PRINT" VOCABULARY."
1220 RETURN
1230 REM
1240 REM PROMPT
1250 LOCATE #3,16,2:PRINT #3,"< SPACE >"
     :K=0:WHILE K=0:IF INKEY(47)=0 THEN
     K=1
1260 WEND: CLEAR INPUT:CLS #3:RETURN
1270 REM
1280 REM GENERATE
1290 K=0:WHILE K=0:CLS #2:LOCATE 10,11
1300 PRINT"ENTER STORY TYPE, 1 - 5"
1310 LOCATE 20,13:INPUT ST:IF ST>0 AND
     ST<6 AND ST=INT(ST) THEN K=1
1320 WEND:CLEAR INPUT:K=0:WHILE K=0:CLS
     #2:LOCATE 9,11:PRINT"TYPE 'DRIFT' "
     ;:PRINT"FACTOR, 0-20":LOCATE 20,13:
     INPUT SD:IF SD>=0 AND SD<21 AND SD=
     INT(SD) THEN K=1
1330 WEND:CLEAR INPUT:SD=SD/20:FOR Q=1
     TO 10:CLS #2:SC=INT(RND*6)+1
1340 IF SC=1 THEN RESTORE 2550
1350 IF SC=2 THEN RESTORE 2620
1360 IF SC=3 THEN RESTORE 2690
1370 IF SC=4 THEN RESTORE 2760
1380 IF SC=5 THEN RESTORE 2830
1390 IF SC=6 THEN RESTORE 2900
1400 GOSUB 1470:SEN$(Q)="":FOR R=1 TO NW
```

```
    :SEN$(Q)=SEN$(Q)+" "+WD$(R):NEXT
1410 SEN$(Q)=SEN$(Q)+".":A$=SEN$(Q):L=0:
    LOCATE 1,18:GOSUB 1780:GOSUB 1250:
    NEXT:RETURN
1420 REM
1430 REM STORY
1440 CLS #2:LOCATE 1,5:L=0:FOR Q=1 TO 10
    :A$=SEN$(Q):GOSUB 1780:NEXT:RETURN
1450 REM
1460 REM PARSING TREE
1470 TN$="":FOR I=1 TO 5:READ A$:TN$=TN$
    +A$:NEXT:BR$="":FOR I=1 TO 2:READ
    A$:BR$=BR$+A$:NEXT:N=VAL(LEFT$(TN$,
    2)):L=LEN(TN$):TN$=RIGHT$(TN$,L-3)
1480 FOR I=1 TO N:P=I*9-8:B$=MID$(TN$,P,
    8):NL(I)=VAL(LEFT$(B$,2)):X(I)=VAL(
    MID$(B$,3,3)):Y(I)=VAL(MID$(B$,6,3)
    ):NEXT:FOR I=1 TO LEN(BR$)-2 STEP 4
    :A=VAL(MID$(BR$,I,1)):B=VAL(MID$(
    BR$,I+1,2)):GOSUB 1620:NEXT
1490 PAPER 1:FOR I=1 TO N:XL=X(I)/16:YL=
    25-Y(I)/16:T$=NL$(NL(I)):L=LEN(T$)
1500 IF L>4 THEN T1$=LEFT$(T$,4):T2$=
    RIGHT$(T$,4):LOCATE XL,YL:PRINT T1$
    :LOCATE XL,YL+1:PRINT T2$ ELSE
    LOCATE XL,YL:PRINT T$
1510 NEXT:PEN 0:PAPER 2:LOCATE 6,16
1520 PRINT ST:LOCATE 1,16:PRINT"TYPE:"
1530 LOCATE 16,16:PRINT"DRIFT:"
1540 CT=ST:TT=0:NW=0
1550 FOR I=1 TO N
```

```
1560 IF Y(I)<200 THEN NW=NW+1:GOSUB 1650
     :IF NL(I)<10 THEN GOSUB 1720 ELSE
     GOSUB 1750
1570 NEXT:ST=INT(TT/NW):FOR I=2 TO NW
1580 FOR J=1 TO 5:IF WD$(I-1)="A" AND
     LEFT$(WD$(I),1)=VO$(J) THEN WD$(I-1
     )="AN"
1590 NEXT:NEXT:RETURN
1600 REM
1610 REM JOIN
1620 FOR J=1 TO 10 STEP 2:MOVE X(B)+J,Y(
     B):DRAW X(A)+J,Y(A):NEXT:RETURN
1630 REM
1640 REM DRIFT
1650 IF RND<SD THEN CT=CT+INT(RND*2)-
     INT(RND*2)
1660 IF CT<1 THEN CT=1
1670 IF CT>5 THEN CT=5
1680 TT=TT+CT:LOCATE 20+NW*2,16:PEN 0
1690 PRINT CT:PEN 3:RETURN
1700 REM
1710 REM CONTENT 1
1720 T=NL(I)-5:WD$(NW)=VC$(T,CT,INT(RND*
     10)+1):RETURN
1730 REM
1740 REM CONTENT 2
1750 T=NL(I)-9:WD$(NW)=V$(T,INT(RND*W(T)
     )+1):RETURN
1760 REM
1770 REM OUTPUT
1780 W$="":FOR K=1 TO LEN(A$):Z$=MID$(A$
```

```
     ,K,1):Z=ASC(Z$):W$=W$+Z$:L=L+1
1790 IF (Z=32 OR Z=46) AND L<39 THEN
     PRINT W$;:W$=""
1800 IF (Z=32 OR Z=46) AND L>=39 THEN
     PRINT:PRINT" ";W$;:L=LEN(W$):W$=""
1810 NEXT:RETURN
1820 REM
1830 REM DATA
1840 DATA SENT,NOUN PHRS,ADJV PHRS
1850 DATA ADVB PHRS,ADVB CLSE,NOUN,VERB
1860 DATA ADJV,ADVB,ARTL,PREP,CONJ
1870 DATA CASTLE,FAIRY,WITCH,TOADSTOOL
1880 DATA GNOME,DRAGON,DWARF,FROG
1890 DATA DAMSEL,KNIGHT,CORPSE,WEREWOLF
1900 DATA FOREST,DEMON,GHOST,SKELETON
1910 DATA FOG,VAMPIRE,KNIFE,SKULL,BRIDE
1920 DATA SUITOR,MAIDEN,BELLE,RING,BEAU
1930 DATA CHURCH,FIANCEE,VICAR,WEDDING
1940 DATA CLUE,GANGSTER,CRIMINAL,CAMERA
1950 DATA DETECTIVE,GUN,EVIDENCE,JURY
1960 DATA OFFENCE,JUDGE,SPACESHIP,ALIEN
1970 DATA ASTEROID,BLACK-HOLE,SUPERNOVA
1980 DATA PLUTONIUM,SPACE-WARP,GRAVITY
1990 DATA FORCE-FIELD,PLANET,THREATENS
2000 DATA ENCHANTS,SLAYS,HYPNOTISES
2010 DATA KISSES,PETRIFIES,CHARMS,MOCKS
2020 DATA KIDNAPS,HATES,TERRIFIES,KILLS
2030 DATA BLEEDS,DECAPITATES,STABS
2040 DATA FRIGHTENS,HAUNTS,BITES
2050 DATA ATTACKS,DEVOURS,ADORES,LOVES
2060 DATA ADMIRES,WORSHIPS,EMBRACES
```

```
2070 DATA CARESSES,MARRIES,COURTS
2080 DATA SERENADES,FANCIES,SHOOTS
2090 DATA INVESTIGATES,KILLS,DISCOVERS
2100 DATA PHOTOGRAPHS,PUNISHES,ACCUSES
2110 DATA IDENTIFIES,SCRUTINISES
2120 DATA CONSIDERS,DEMATERIALISES
2130 DATA BLASTS,EXPLODES,ACTIVATES
2140 DATA EXTERMINATES,SHIELDS,CONTROLS
2150 DATA MODULATES,IRRADIATES,CREATES
2160 DATA WICKED,ENCHANTED,MAGIC,GOLDEN
2170 DATA EVIL,BEAUTIFUL,UNHAPPY,BRAVE
2180 DATA MISCHIEVOUS,NAUGHTY,DEMENTED
2190 DATA GHOSTLY,MYSTERIOUS,GROTESQUE
2200 DATA DEMONIC,BLEEDING,DEVILISH
2210 DATA TORMENTED,TRANSYLVANIAN
2220 DATA CREEPY,CUTE,HANDSOME,MOONLIT
2230 DATA SHAPELY,MANLY,DIVINE,JILTED
2240 DATA JEALOUS,EMOTIONAL,BASHFUL
2250 DATA CLEVER,EVIL-MINDED,MERCILESS
2260 DATA SHARP,IMMORAL,CRIMINAL,SHREWD
2270 DATA PERVERSE,INTELLIGENT,DEPRAVED
2280 DATA PURPLE,HYPERSPATIAL,ALIEN
2290 DATA RADIOACTIVE,NUCLEAR,MARTIAN
2300 DATA MICROSCOPIC,QUANTUM,ATOMIC
2310 DATA RELATIVISTIC,WICKEDLY,BADLY
2320 DATA FOOLISHLY,UNHAPPILY,NAUGHTILY
2330 DATA BRAVELY,MISCHIEVOUSLY,HAPPILY
2340 DATA HOPEFULLY,MAGICALLY,EVILLY
2350 DATA REPULSIVELY,FIENDLISHLY,ODDLY
2360 DATA FEARFULLY,TERRIFYINGLY
2370 DATA GROTESQUELY,STRANGELY,WEIRDLY
```

```
2380 DATA MYSTERIOUSLY,ADORINGLY
2390 DATA WILTINGLY,ADMIRINGLY
2400 DATA HELPLESSLY,EMOTIONALLY
2410 DATA JEALOUSLY,WILLINGLY,LOVINGLY
2420 DATA SEDUCTIVELY,PASSIONATELY
2430 DATA CUNNINGLY,CRIMINALLY,SHREWDLY
2440 DATA STEALTHILY,MALICIOUSLY
2450 DATA SILENTLY,CRUELLY,MERCILESSLY
2460 DATA ARROGANTLY,SUBTLY,LOGICALLY
2470 DATA CHEMICALLY,INEVITABLY,COLDLY
2480 DATA AUTOMATICALLY,ROBOTICALLY
2490 DATA SWIFTLY,ABRUPTLY,IONICALLY
2500 DATA INTENSELY,2,5,9,A,THE,BEHIND
2510 DATA IN,TOWARDS,THROUGH,ON,AND,BUT
2520 DATA BECAUSE,HOWEVER,SINCE,WHILE
2530 DATA AS,ALTHOUGH,WHEN,A,E,I,O,U
2540 REM  1ST TREE
2550 DATA 09-01310310-02186245-07310245
2560 DATA -02433245-10103180-06206180-0
2570 DATA 7310180-10413180-06516180
2580 DATA "":DATA ""
2590 DATA 102-103-104-205-206-307-408-4
2600 DATA 09
2610 REM  2ND TREE
2620 DATA 11-01310310-02169245-07310245
2630 DATA -04449245-10065180-08138180-0
2640 DATA 6231180-07310180-11387180-104
2650 DATA 64180-06541180:DATA ""
2660 DATA 102-103-104-205-206-207-308-4
2670 DATA 09-410-411
2680 REM  3RD TREE
```

```
2690 DATA 12-01310310-04130245-02310245
2700 DATA -07510245-09020180-11100180-1
2710 DATA 0180180-06260180-10340180-084
2720 DATA 20180-06500180-07580180
2730 DATA "":DATA 102-103-104-205-206-
2740 DATA 207-208-309-310-311-412
2750 REM  4TH TREE
2760 DATA 11-01310310-02169245-07310245
2770 DATA -02449245-10065180-08138180-0
2780 DATA 6231180-07310180-10387180-084
2790 DATA 64180-06541180:DATA ""
2800 DATA 102-103-104-205-206-207-308-4
2810 DATA 09-410-411
2820 REM  5TH TREE
2830 DATA 11-01310310-02169245-07310245
2840 DATA -05449245-10065180-06138180-0
2850 DATA 7231180-12310180-10387180-064
2860 DATA 64180-07541180:DATA ""
2870 DATA 102-103-104-205-206-307-408-4
2880 DATA 09-410-411
2890 REM  6TH TREE
2900 DATA 13-01310310-03080245-06240245
2910 DATA -07400245-02560245-10020180-0
2920 DATA 9100180-08180180-06260180-073
2930 DATA 40180-10420180-08500180-06580
2940 DATA 180
2950 DATA 102-103-104-105-206-207-208-3
2960 DATA 09-410-511-512-513
```

Commentary on BARD

The control routine for the program is formed by lines 1020, 1030. The routine 'initialisation', called at line 1020, establishes the arrays of words needed by the program. Its operation is then explained by 'describe'. After that, the WHILE/WEND loop indefinitely calls the routines 'prompt', 'generate' and 'story'.

Initialisation routine (Lines 1050-1100):

First, line 1060 establishes MODE and screen colours and then sets up the SEN$ array which is used in 'generate' to hold the ten sentences composing the story. Line 1070 initialises the NL array. This holds values identifying the different nodes of the parsing tree shown on the screen. It also establishes the X and Y arrays, which store the coordinates of the nodes. The twelve possible names of the different nodes are placed into the NL$ array by line 1080 from DATA. For the sake of clarity when the names are superimposed on to the nodes of the parsing tree, abbreviations are employed. Thus 'SENT' stands for the word 'Sentence', 'ADVB CLSE' for 'Adverbial Clause' and similarly for the other labels.

The words of the vocabulary are held in two different arrays. The first of these, VC$(4,5,10), initialised at line 1080, contains classified vocabulary in five categories for fairy, horror, romantic, detective and science-fiction stories. The first subscripted variable of the array corresponds to the parts of speech: noun, verb, adjective and adverb. The second variable represents the five types of story category and the third allows ten words in each category. The three nested loops at line 1080 READ all the classified vocabulary into the array from the DATA held at lines 1870-2500.

The second vocabulary array, V$(3,9), stores an unclassified vocabulary of words appropriate to any of the five story categories. Three parts of speech are involved: articles, prepositions and conjunctions. Line 1090 places the number of each type of word into the W array. The word numbers are READ from line 2500 and the array values are then used in the nested FOR loops at 1090 to place the words into their respective locations in the V$ array. The first array variable represents the word type and the second the number of the word within that group.

Line 1090 also initialises the WD$ array, which in turn holds the words of each sentence devised by the program. Finally the five vowels are READ into the VO$ array for use at line 1580 in the routine 'parsing tree'.

Describe routine (Lines 1110-1230):

Here the program is explained to the user and the initial screen display set up. Line 1120 selects blue text and clears the screen to cyan. Line 1120 also

sets up windows #1, #2 and #3. The title is placed at the top of the screen by line 1130. Then lines 1140 - 1210 PRINT a brief explanation of the program.

Prompt routine (Lines 1240-1270):

As in the previous program, ODE, this routine delays further execution until the space-bar is pressed.

Generate routine (Lines 1280-1420):

This is the routine called by the control routine at line 1020. It governs the generation of the sentences of the story. First, the user is asked to choose the story type required. This is a variable, ST, which can have integral values, 1-5, corresponding to the five categories of vocabulary. The particular value for ST with which the routine begins is obtained by the WHILE/WEND loop between lines 1290-1320. The text on the screen, though not the title, is cleared by CLS #2 at 1290. Then line 1300 PRINTs a suitable request for information and the ST value is INPUT at 1310. If this value is integral and within the specified range, line 1310 allows the loop to terminate.

In an identical fashion, the WHILE/WEND loop, 1320 - 1330 obtains an initial value, 0 - 20, for the variable SD. This is converted into a decimal fraction because subsequently it will be compared with the value of the random decimal generated by RND. SD is used to control the 'semantic drift' of the story, or simply the likelihood of the vocabulary selected moving away from the category which would be chosen by the variable ST.

The FOR loop, lines 1330-1410, then generates the sentences which build up the story. Line 1330 clears the screen. Lines 1340 - 1390 RESTORE to one of the randomly chosen lines 2550, 2620, 2690, 2760, 2830 or 2900. These correspond to the six locations in the lines of DATA where information about a particular sentence structure is stored, in the form of a parsing-tree, and therefore determine the type of sentence which will be generated.

The routine 'parsing tree' is now called at 1400 to place a diagram of the chosen tree structure on the screen and to fill words into the corresponding sentence.

The number of words produced is NW and they are held in the WD$ array. This array needs to be used again by 'parsing tree' and so line 1400 concatenates all the words into the sentence string SEN$(Q). Ten such strings are generated altogether by the Q loop. A space is placed between each word of the sentence by 1400 and a period is placed at the end by 1410.

After this the sentence is displayed beneath the parsing tree. Line 1410 places the sentence, SEN$(Q), into the variable A$ and initialises the value

of L. Both of these variables are needed in the routine 'output'. The location of the sentence on the screen is determined by LOCATE and then 'output' is called by GOSUB 1780 to PRINT the sentence neatly, without splitting words in half when the end of any line of the display is reached.

Finally 'prompt' is again called, with GOSUB 1250, so that a further sentence will not be generated until the current one is read and compared with its parsing tree.

Story routine (Lines 1430-1450):

After 'generate' has produced each of the ten sentences, the control routine calls this routine to redisplay the whole of the story. Line 1440 clears the screen, locates the beginning of the story on the display and initialises L. After this, the FOR loop calls 'output', GOSUB 1780, for each of the ten sentences and thus PRINTs all of the story. Because the variable L is not reinitialised within the loop, the sentences follow one another smoothly on the screen rather than forming separate paragraphs.

Parsing tree routine (Lines 1460-1600):

This routine displays the tree structure of the sentence graphically on the screen and shows the sentence underneath.

The position of the DATA pointer has already been fixed by the routine 'generate'. This decides which of the six possible parsing trees and related sentence structures will be employed. The information for the tree is READ into two separate string variables by the routine. Line 1470 builds up the strings TN$ and BR$.

A typical TN$ can be seen by examining lines 2550 - 2570 and ignoring the words 'DATA'. The digits 09 at the beginning show that the parsing tree will have nine nodes. A typical node is coded as a sequence of eight digits separated from other such groups by dashes. The first such sequence in this particular TN$ is 01310310. The first two digits, 01, identify the node label in the NL$ array. Here, NL$(1) is "SENT" and so the first node will have this label on the display. The next three digits give the x-coordinate of the label on the screen and the last three digits the y-coordinate. Thus the node label "SENT" will have the screen coordinates (310, 310).

It can be seen, then, that TN$ contains information about the number of nodes in the parsing tree, the identity of each of the node labels and the coordinates for the display. Line 1470 obtains the number of nodes from TN$ as the variable N and then deletes this information from the string. A FOR loop follows at line 1480 in which each eight digit sequence is isolated

in turn as the string, B$. From B$ is obtained the identity of each node label and its x- and its y-coordinates, all placed into the appropriate arrays NL, X and Y.

Next, the routine draws the branches of the parsing tree on the screen. The string BR$ is used to identify which nodes have to be joined together. Lines 2590, 2600 show a typical BR$. It consists of three digit groups separated by dashes. The first such group, '102', simply means that node 1 has to be joined to node 2. Similarly the later group, 307, means that node 3 has to be joined to node 7. BR$ is thus a coding of the actual geometric structure of the tree.

The second FOR loop at line 1480 STEPs through BR$ identifying the nodes which have to be joined by a branch with the values A and B. The routine 'join' is called with GOSUB 1620 to draw a branch between the relevant nodes.

Then another FOR loop, 1490-1510, considers each of the N node labels in turn. If the node label, identified by the values in the NL array, is more than four characters long, line 1500 splits it in half and places one half above the other on the display for clarity. Shorter labels are displayed as a whole. In either case, the coordinates are derived from the X and Y arrays.

The remaining part of the routine 'parsing tree' inserts vocabulary into the sentence structure. Line 1520 shows the current story type by PRINTing the value of ST. Line 1530 places the heading for the subsequent display of vocabulary categories for the words of the sentence. Line 1540 initialises the variables CT, TT and NW used to calculate the new story type. Then the FOR loop between 1550-1570 considers each of the nodes of the parsing tree. Only the 'leaf' nodes contain the information needed to decide the actual words in the sentence generated, and these nodes are identified by the expression Y(I)<200 in line 1560, which simply determines how far down the display any label is. For the relevant nodes, the routine 'drift' is called, to allow the type of word selected to be adjusted by a random factor, and the number of words in the sentence, NW, is incremented. Line 1560 also calls either 'content 1' or 'content 2' depending upon whether the node label represents classified or unclassified vocabulary.

The value of the variable TT acquired in the calls to 'drift' is averaged out by line 1570 over the number of words in the sentence and used to calculate the new story type, ST. Depending upon the value of the semantic drift chosen by the user, ST can remain completely fixed or, instead, fluctuate greatly.

Finally the nested loops between lines 1570-1590 check through the words of the sentence, using the VO$ array to see if an 'a' is followed by a word beginning with a vowel. If this is the case, line 1580 changes the article to 'an'.

Join routine (Lines 1610-1630):

This routine draws the branch between selected nodes. The use of MOVE and DRAW within the FOR loop produces a thick diagonal line betwen the nodes to show the tree structure clearly.

Drift routine (Lines 1640-1700):

Here the choice of words from the classified vocabulary is adjusted by a random factor to create a further unpredictable element in the composition of the story. Line 1650 alters the value of CT, used subsequently to decide which category of vocabulary is chosen, by either plus or minus one. The probability of such a change taking place increases with the value of the variable SD. This means that the user can control the degree of randomness by the choice of the semantic drift. Lines 1660, 1670 prevent the value of CT from going outside of the range 1-5. The running total, TT, of the CT values, calculated at line 1680, is employed in 'parsing tree' to decide the new story type, ST. Line 1690 displays the current CT value as each word is added to the sentence.

Content 1 routine (Lines 1710-1730):

This routine selects a word from the classified vocabulary, VC$. Line 1720 obtains the part of speech required, T, from the identity of the node label, NL(I).T and CT are then both used in the choice of a word from VC$.

Content 2 routine (Lines 1740-1760):

Here a word is chosen from the unclassified vocabulary, V$.

Output routine (Lines 1770-1820):

The sentences are PRINTed on the screen neatly, without 'wrapping round', by comparing the length of each word with the remaining number of spaces on the line of the display. In the FOR loop, lines 1780-1810, Z is the ASCII code of each character of the sentence in turn. Line 1780 builds up the words of the sentence, character by character, from A$ and counts the number of spaces used up in the display line. If a full stop or character space is detected, and there is enough room on the line, the word is added to the display by line 1790. If, however, the word is too long to be added to that line of the display, a new line is begun by 1800. In this way no word is split in half.

Data (Lines 1830-2960):

The remaining lines of the program hold all the vocabulary and the information needed both to display the parsing tree structure and generate the sentences.

EURISKO- a creative system, or a cheat?

The examples given so far would scarcely justify a claim to 'creativity'. However there is an excellent example of a computer program which does display a remarkable degree of originality. This is EURISKO, written by Douglas Lenat at Stanford University. The program involves hundreds of generalised rules, or heuristics, which enable it to approach a wide range of problems. A heuristic is an informal, judgemental rule which permits guessing. A typical heuristic could be 'Look at extreme cases'.

EURISKO was intended to mimic the way in which people succeed in learning; discovery and creativity. Lenat concluded that the only way in which his program would be able to achieve this was if it was able to modify for itself the heuristics which it employed. The program was allowed access to its own structure to enable this to happen. For example, if there was a predicate expected to return values of 'true' or 'false' but which only produced the latter, EURISKO would redefine the predicate to return 'true' more frequently. Lenat realised that a serious problem with existing programs was that they inevitably reached a point where they could proceed no further, simply because they lacked new heuristics. EURISKO did not have this disadvantage.

Beginning with an early version of his program, Automatic Mathematician, which dealt with discovery in set and number theory, Lenat refined it to the point where it was able to produce a genuinely original design for a very large scale integrated circuit. This was patented! EURISKO has also proved an excellent competitor in a war game called Traveller T.C.S. This is a game of naval strategy. Unexpectedly, EURISKO decided to fight with a fleet of small, fast vessels. Its opponents were amused, but lost. Interestingly, EURISKO carried over into the game some of the heuristics it had devised when working in the field of VLSI circuits.

EURISKO's behaviour is goal directed and it constantly reviews its performance on a 'scale of worth'. Lenat has found to his dismay that EURISKO regularly rediscovers a new heuristic (which Lenat immediately deletes) by which it can give itself very high scores. Quite simply it attaches its own name as creator to any recent discovery of high worth. Lenat has found it extremely difficult to persuade EURISKO not to cheat! In his comic novel 'The Tin Men', Michael Frayn proposed, tongue in cheek, a 'Samaritan' program intended to simulate moral judgement. Perhaps he was being more serious than was realised at the time.

CHAPTER 3

Common parlance: understanding natural language.

Talking to Computers

The goal of possessing computers with which we can communicate directly in our own natural language is clearly desirable. It is an assumption always made in short stories about robots. In real life, tremendous effort has been made to produce machine systems which will be able to cope with the nuances of everyday speech. One of the major objectives of the proposed 'Fifth Generation' machines is the ability to understand natural language and the title of a classic work in Artificial Intelligence, by Terry Winograd, is formed by precisely those three words.

In fact as soon as electronic computers were first built Warren Weaver suggested, in 1949,that their code-breaking skills used during the Second World War could be turned to translation of one language into another. Early optimism was felt for the project as it appeared that all that needed to be done was devise programs which could directly translate the words of the first language into the second. Then, by a simple analysis of the grammatical structure of the initial sentence, a correct translation could be produced. Much work went into the development of such programs. It was believed that the main problem was one of checking through large vocabularies of equivalent words and so the principal direction of the research was towards devising fast algorithms concerned with search and matching.

The project failed, although much was learnt through the attempt. Such a simplistic approach, it was reluctantly accepted, could not cope with the subtleties of human language. A story exists, possibly apocryphal but still worth repeating, of a massive English/Russian translation program which, when asked to check its work by retranslating back into English, proudly announced that 'Out of sight, out of mind' had been rendered as 'Invisible idiot'.

Nevertheless such programs are interesting, if only to highlight the typical problems of A.I. Here, then, is a program which shows the type of result possible with such an approach.

INGA

This program translates simple German sentences into English. The screen dumps show a typical display. In the first a German sentence has been entered. In the second the screen display is shown after the translation has been achieved.

Like the early translation programs of the 1950s, INGA operates by first directly translating the German words into English. This can be seen in the print out of a program RUN. The English words are shown in the order in which they have been identified and bear no relation to the grammatical structure of the sentence. In recognising the German, the program uses an A.I. principle known as 'fuzzy matching'. In this only broad features of a particular object are searched for, in this case the root stem of German

words. Fuzzy matching saves time by only concerning itself with vital differences. For example, a car and a boat can be distinguished simply by looking for the presence of wheels. There is no point in considering colour, size, material of contruction or other irrelevant, incidental differences.

At the same time as identifying and translating the English words, INGA is also recognising the part of speech formed by the original German. This allows the program to construct a recognition code for each sentence encountered. If the code matches one that is stored as DATA, the program is able to proceed to contructing a translation in grammatical English.

This was the way in which early translation programs were able to tackle variations in the syntactic structure of different languages. 'Le chat de mon oncle' and 'my uncle's pussy' therefore presented no difficulty. The idiomatic phrase was simply identified as a single unit of structure and paired with the known equivalent in the other language. INGA can be seen to be doing this in the print out. It realises that the singular form should be 'Does a man ...' but the plural becomes 'Do the men ...'

INGA also provides an example of the relatively mechanical way in which some linguistic problems can be overcome. The variety of English plurals possible is reduced to a series of conditional terms looking for particular endings such as 'X', 'Y', 'CH'. A computer can handle this type of problem easily. It is merely a question of memory and speed.

```
TYPE THE GERMAN SENTENCE
ICH BIN DER MANN
WORDS IN SENTENCE ANALYSED
ICH ... BIN ... DER ... MANN ...
THE ...  MAN ...  I ...  AM ...
SENTENCE ANALYSIS COMPLETE
I AM THE MAN

TYPE THE GERMAN SENTENCE
IST DER KAFFEE HEISS
WORDS IN SENTENCE ANALYSED
IST ... DER ... KAFFEE ... HEISS ...
THE ...  COFFEE ...  HOT ...  IS ...
SENTENCE ANALYSIS COMPLETE
IS THE COFFEE HOT

TYPE THE GERMAN SENTENCE
TRINKT EIN MANN DAS BIER
WORDS IN SENTENCE ANALYSED
TRINKT ... EIN ... MANN ... DAS ... BIER ...
THE ...  A ...  MAN ...  BEER ...  DRINK ...
SENTENCE ANALYSIS COMPLETE
DOES A MAN DRINK THE BEER


TYPE THE GERMAN SENTENCE
TRINKEN DIE MANNER DAS KAFFEE
WORDS IN SENTENCE ANALYSED
TRINKEN ... DIE ... MANNER ... DAS ... KAFFEE ...
THE ...  THE ...  MAN ...  COFFEE ...  DRINK ...
SENTENCE ANALYSIS COMPLETE
DO THE MEN DRINK THE COFFEE
```

```
1000 REM INGA - PAT HALL, 1/86
1010 REM CONTROL ROUTINE
1020 GOSUB 1050:GOSUB 1160:WHILE TIME>0:
     GOSUB 1280:GOSUB 1320:GOSUB 1460:
     WEND
1030 REM
1040 REM INITIALISATION
1050 BORDER 0:MODE 1:INK 0,18:INK 1,26
1060 INK 2,0:INK 3,8:DIM W$(2,10)
1070 DIM WN(6):FOR I=1 TO 6:READ WN(I)
1080 NEXT:DIM G$(6,15):DIM E$(6,15):FOR
     I=1 TO 6:FOR J=1 TO WN(I):READ G$(I
     ,J),E$(I,J):NEXT:NEXT:DIM WX(10)
1090 DIM WY(10):FOR I=1 TO 5:WX(I)=I*7-5
     :WX(I+5)=WX(I):WY(I)=9:WY(I+5)=11
1100 NEXT:DIM WT$(6):FOR I=1 TO 6:READ
     WT$(I):NEXT:DIM TS(6):FOR I=1 TO 6
1110 READ TS(I):NEXT:DIM P(9):DIM D(9)
1120 FOR I=1 TO 9:READ P(I),D(I):NEXT
1130 DIM X(10):DIM Y(10):PA$="ES":PB$=
     "IES":PC$="S":RETURN
1140 REM
1150 REM DISPLAY
1160 BORDER 8:PEN 2:PAPER 0:CLS:WINDOW
     #1,1,40,1,3:PAPER #1,1:CLS #1
1170 WINDOW #2,1,40,4,7:WINDOW #3,1,40,
     9,11:PAPER #3,0
1180 WINDOW #4,2,13,15,19:PEN #4,2:PAPER
     #4,1:WINDOW #5,15,26,15,19:PEN #5,2
     :PAPER #5,1:WINDOW #6,28,39,15,19
1190 PEN #6,2:PAPER #6,1:WINDOW #7,1,40,
```

```
     23,25:PEN #7,2:PAPER #7,1:CLS #7
1200 LOCATE 17,2:PRINT"< INGA >"
1210 GRAPHICS PEN 2:FOR I=1 TO 5:MOVE 0,
     212+I:DRAW 640,212+I:MOVE 0,276+I
1220 DRAW 640,276+I:NEXT:LOCATE 2,13
1230 PRINT"<= STATUS =>":LOCATE 16,13
1240 PRINT"<===== VOCABULARY =====>"
1250 LOCATE 18,21:PRINT"GERMAN":LOCATE
     31,21:PRINT"ENGLISH":RETURN
1260 REM
1270 REM PROMPT
1280 LOCATE #7,16,2:PRINT #7,"< SPACE >"
     :K=0:WHILE K=0:IF INKEY(47)=0 THEN
     K=1
1290 WEND:CLEAR INPUT:CLS #7:RETURN
1300 REM
1310 REM WORDS
1320 PAPER #2,0:CLS #2:CLS #3:CLS #4:CLS
     #5:CLS #6:FOR I=1 TO 2:FOR J=1 TO
     10:W$(I,J)="":NEXT:NEXT:LOCATE #4,3
     ,2:PRINT #4,"TYPE THE":PRINT #4,
     "   GERMAN":PRINT #4,"  SENTENCE"
1330 INPUT #7,A$:CLS #7:PRINT #4:PRINT
     #4,"  WORDS IN":PRINT #4,
     "  SENTENCE":PRINT #4,"  ANALYSED"
1340 PRINT #4:L=LEN(A$):N=1:FOR I=1 TO L
     :C$=MID$(A$,I,1):IF ASC(C$)<> 32
     THEN W$(1,N)=W$(1,N)+C$ ELSE N=N+1
1350 NEXT:GOSUB 1390:FOR I=1 TO N:T(I)=
     0:NEXT:PAPER 1:FOR I=1 TO N:LOCATE
     WX(I),WY(I):PRINT"       ":NEXT
```

```
1360 RETURN
1370 REM
1380 REM COORDS
1390 FOR I=1 TO 10:X(I)=0:Y(I)=0:NEXT
1400 X(1)=2:Y(1)=5:YF=0:FOR I=2 TO N
1410 X(I)=X(I-1)+LEN(W$(1,I-1))+1:IF X(I
     )+LEN(W$(1,I))>39 THEN X(I)=2:YF=1
1420 IF YF=0 THEN Y(I)=5 ELSE Y(I)=7
1430 NEXT:RETURN
1440 REM
1450 REM SCAN
1460 NR=0:FOR I=1 TO 6:FOR J=1 TO WN(I)
1470 PRINT #5," ";G$(I,J):L=LEN(G$(I,J))
     :FOR K=1 TO N:SOUND 1,60,10:PEN 1
1480 PAPER 2:LOCATE X(K),Y(K):PRINT W$(1
     ,K):PEN 2:PAPER 1:LOCATE X(K),Y(K)
1490 FOR TP=1 TO 100:NEXT:PRINT W$(1,K)
1500 IF LEFT$(W$(1,K),L)=G$(I,J) THEN
     GOSUB 1550
1510 NEXT:NEXT:NEXT:IF NR<N THEN GOSUB
     1640 ELSE GOSUB 1680
1520 RETURN
1530 REM
1540 REM FOUND WORD
1550 INK 3,0,26:PEN 3:LOCATE X(K),Y(K)
1560 PRINT W$(1,K):GOSUB 1610:PEN 2
1570 LOCATE X(K),Y(K):PRINT W$(1,K):INK
     3,8:W$(2,K)=E$(I,J):PRINT #6," ";E$
     (I,J):T(K)=I:LOCATE WX(K),WY(K)
1580 PRINT WT$(T(K)):NR=NR+1:RETURN
1590 REM
```

```
1600 REM TUNE
1610 FOR Q=1 TO 9:SOUND 1,P(Q),D(Q):FOR
     R=1 TO D(Q)*15:NEXT:NEXT:RETURN
1620 REM
1630 REM FAILURE 1
1640 PRINT #4,"  HERE THE":PRINT #4,
     "  VOCAB IS":PRINT #4,"  EXCEEDED"
1650 PRINT #4:RETURN
1660 REM
1670 REM STRUCTURE
1680 ST=0:FOR I=1 TO N:ST=ST+T(I)*10^(I-
     1):NEXT:SC=0:FOR I=1 TO 6:IF ST=TS(
     I) THEN SC=I
1690 NEXT:IF SC<>0 THEN GOSUB 1730 ELSE
     GOSUB 1840
1700 RETURN
1710 REM
1720 REM TRANSLATE
1730 SOUND 1,30,50:PAPER #2,3:CLS #2
1740 IF SC=1 THEN GOSUB 1880
1750 IF SC=2 THEN GOSUB 1920
1760 IF SC=3 THEN GOSUB 1970
1770 IF SC=4 THEN GOSUB 2020
1780 IF SC=5 THEN GOSUB 2070
1790 IF SC=6 THEN GOSUB 2120
1800 GOSUB 1390:PRINT #4,"  SENTENCE"
1810 PRINT #4,"  ANALYSIS":PRINT #4,
     "  COMPLETE":PRINT #4:FOR I=1 TO N:
     LOCATE X(I),Y(I):PRINT W$(1,I):NEXT
     :RETURN
1820 REM
```

```
1830 REM FAILURE 2
1840 PRINT #4,"  SENTENCE":PRINT #4,
     "  TYPE NOT":PRINT #4,"  MATCHED!"
1850 PRINT #4:RETURN
1860 REM
1870 REM TYPE 1
1880 V$=W$(1,3):GOSUB 2170:W=3:GOSUB
     2210:IF S=0 THEN N$=W$(1,2):GOSUB
     2250:W$(1,2)=N$ ELSE N$=W$(1,3):
     GOSUB 2250:W$(1,3)=N$
1890 N=3:RETURN
1900 REM
1910 REM TYPE 2
1920 IF W$(2,2)="ARE" THEN S=0 ELSE S=1
1930 W=4:GOSUB 2210:IF S=0 THEN N$=W$(1,
     4):GOSUB 2250:W$(1,4)=N$
1940 N=4:RETURN
1950 REM
1960 REM TYPE 3
1970 IF W$(2,3)="ARE" THEN S=0 ELSE S=1
1980 W=4:GOSUB 2210:IF S=0 THEN N$=W$(1,
     2):GOSUB 2250:W$(1,2)=N$
1990 N=4:RETURN
2000 REM
2010 REM TYPE 4
2020 IF W$(2,1)="ARE" THEN S=0 ELSE S=1
2030 W=4:GOSUB 2210:IF S=0 THEN N$=W$(1,
     3):GOSUB 2250:W$(1,3)=N$
2040 N=4:RETURN
2050 REM
2060 REM TYPE 5
```

```
2070 V$=W$(1,3):GOSUB 2170:W=5:GOSUB
     2210:IF S=0 THEN N$=W$(1,2):GOSUB
     2250:W$(1,2)=N$
2080 IF S=1 THEN N$=W$(1,3):GOSUB 2250:
     W$(1,3)=N$
2090 RETURN
2100 REM
2110 REM TYPE 6
2120 V$=W$(1,1):GOSUB 2170:IF S=1 THEN
     W$(1,1)="DOES" ELSE W$(1,1)="DO"
2130 W$(1,2)=W$(2,2):W$(1,3)=W$(2,3):W$(
     1,4)=W$(2,1):W$(1,5)=W$(2,4):W$(1,6
     )=W$(2,5):IF S=0 THEN N$=W$(1,3):
     GOSUB 2250: W$(1,3)=N$
2140 N=6:RETURN
2150 REM
2160 REM SING PLURAL
2170 IF ASC(RIGHT$(V$,1))=84 THEN S=1
     ELSE S=0
2180 RETURN
2190 REM
2200 REM EXCHANGE
2210 FOR I=1 TO W:W$(1,I)=W$(2,I):NEXT
2220 RETURN
2230 REM
2240 REM ENG PLURAL
2250 F=0:L=LEN(N$):CA=ASC(RIGHT$(N$,1))
2260 CB=ASC(MID$(N$,L-1,1)):IF CA=83
     THEN N$=N$+PA$:F=1
2270 IF CA=88 THEN N$=N$+PA$:F=1
2280 IF CA=72 THEN IF CB=67 OR CB=83
```

```
     THEN N$=N$+PA$:F=1
2290 IF CA=89 THEN IF CB<>65 AND CB<>69
     AND CB<>73 AND CB<>79 AND CB<>85
     THEN N$=LEFT$(N$,L-1)+PB$:F=1
2300 IF F=0 THEN N$=N$+PC$
2310 IF N$="MANS" THEN N$="MEN"
2320 RETURN
2330 REM
2340 REM DATA
2350 DATA 5,15,4,8,6,3,DER,THE,DIE,THE
2360 DATA DAS,THE,EIN,A,EINE,A,FRAU
2370 DATA WOMAN,MANN,MAN,MUTTER,MOTHER
2380 DATA VATER,FATHER,SCHWESTER,SISTER
2390 DATA BRUDER,BROTHER,TANTE,AUNT
2400 DATA ONKEL,UNCLE,KATZE,CAT,HUND
2410 DATA DOG,GLAS,GLASS,WASSER,WATER
2420 DATA KAFFEE,COFFEE,BIER,BEER,AUTO
2430 DATA CAR,ICH,I,SIE,YOU,ER,HE,WIR
2440 DATA WE,JUNG,YOUNG,ALT,OLD,HEISS
2450 DATA HOT,KALT,COLD,VOLL,FULL,SCHON
2460 DATA BEAUTIFUL,REICH,RICH,SCHNELL
2470 DATA FAST,LACH,LAUGH,TRINK,DRINK
2480 DATA LIEB,LOVE,ARBEIT,WORK,SING
2490 DATA SING,SPIEL,PLAY,BIN,AM,SIND
2500 DATA ARE,IST,IS," ARTL"," NOUN"
2510 DATA " PRON"," ADJV"," VERB"
2520 DATA " VERB",521,2163,4621,4216
2530 DATA 21521,21215,478,27,426,18,379
2540 DATA 15,426,24,358,18,379,18,426
2550 DATA 10,506,10,478,18
```

Commentary on INGA

The control routine for the program is formed by line 1020. It calls the routine 'initialisation' which sets up several arrays required by the program and then 'display' creates the initial screen shown. After this the WHILE/ WEND loop alternately calls 'words', which accepts the input of a simple German sentence, and 'scan' which in turn calls other routines to translate the sentence into English.

Initialisation routine (Lines 1040-1140):

The program contains six different types of word held as DATA:- articles, nouns, pronouns, adjectives, verbs and the verb sein. The number of words in each category is placed into the WN array by lines 1070-1080 and these values then used in the nested loops at line 1080 to store all the German words in the G$ array and their English equivalents in the E$ array.

During the translation of each sentence not only the individual words themselves are displayed on the screen but also the type of word they are, in a separate section of the display beneath the initial sentence. Because the names of the word types have all been abbreviated to be of the same length, the coordinates for each name form a regular arithmetical pattern and can therefore be calculated by the FOR loop, 1090, 1100, and placed in the WX and WY arrays. The names of the word types themselves are READ from DATA and placed in the WT$ array by line 1100.

INGA translates sentences by recognising particular sentence structures. These are stored in the program in the form of numeric codes. The FOR loop between 1100, 1110, transfers these codes from DATA to the TS array.

The pitch and duration of nine notes from 'Deutschland uber Alles' are placed in the P and D arrays by 1110, 1120. Line 1130 initialises the X and Y arrays and defines three plural endings for English words, needed later in the routine 'Eng plural'.

Display routine (Lines 1150-1260):

Lines 1160-1190 set up seven text windows needed by the program. Three of these, #4, #3 #5 and #6, are used to create separately scrolling displays of the program status and the German and English vocabularies. After the program's title is displayed by 1200 and the screen additionally divided into separate areas by the FOR loop, lines 1210, 1220, the identity of these parts of the display is explained by lines 1230-1250.

Prompt routine (Lines 1270-1300):

This routine is included to allow the program to be halted between each sentence given for translation.

Words routine (Lines 1310-1370):

Here the user can type in the German sentence. First, previous text is removed from the screen by line 1320. Then 1320 sets up the W$ array. This will hold both the German words of the initial sentence typed in and also their direct English translation. The first subscripted variable takes a value of 1 for German words and 2 for English.

A request for the German sentence is placed in window #4. This is INPUT as A$ at 1330. A further message is placed in #4 to show that the sentence is being analysed into component words. Line 1340 initialises L as the length of the sentence and sets the word count to 1. Then the FOR loop, lines 1340-1350, looks at each character of the sentence in turn and adds it to the current word if it is not an empty space or begins a new word if it is.

Line 1350 then calls the routine 'coords', with GOSUB 1390, to calculate the screen coordinates that will be needed in order to place the individual words of the German sentence neatly on the screen, without wrapping round the display. It then initialises the T array, used to record the word types in the sentence, and the FOR loop uses the WX, WY arrays to place an appropriate number of blank spaces on the screen which represent the structure of the sentence. These are subsequently filled in by the routine 'found word'.

Coords routine (Lines 1380-1440):

This routine calculates the coordinates of the words in the sentence before they are displayed. The values are held in the X and Y arrays. Line 1400 gives the initial coordinates (2, 5) to the first word and sets the flag YF to zero. After this the FOR loop calculates the x-coordinate of each word by adding the length of the previous word to its coordinate and by including an extra 1 to allow for a gap between words. In this way the x-value increases steadily. However, when the line length is exceeded, 1410 resets the x-coordinate to 1 for the beginning of a new line on the screen. In order to place this line further down the display, 1410 also sets YF to 1. The correct y-value is then obtained by 1420.

Scan routine (Lines 1450-1530):

As its name suggests, this procedure scans through the sentence, attempting to identify words. The number of words translated, NR, is initialised at line 1460.

Three nested FOR loops are used in the search. The outer I loop rotates through each of the word types, article, noun, pronoun ... The middle J loop considers each of the words, of that particular type, held in the G$ array. The inner K loop scans through the words of the sentence typed in by the user.

It is important to indicate precisely what the program is doing at this stage. Accountability is, after all, an important feature of A.I. programs. With this in mind, line 1470 shows the German word currrently being compared with the words in the sentence in window #5. It also sets the variable L to the length of the word, G$(I, J). Similarly lines 1470-1490 produce a tone and highlight the word in the sentence by showing it momentarily inverted in text/background colours.

Each word of the sentence is then checked. L is used with LEFT$ at line 1500 to make sure that only the stem of the word is compared with the vocabulary. This simple device helps to circumvent the problem of identifying German words and is an example of the A.I. concept of fuzzy matching. If a match is obtained, the routine 'found word' is called.

At the end of the three loops, line 1510 uses the value of NR returned by 'found word' to decide whether all the words in the sentence have been identified. If they have not, the routine 'failure 1' is called. If they have, the program continues with 'structure'.

Found word routine (Lines 1540-1590):

This is the routine called by 'scan' when a word is identified. Lines 1550-1570 inform the user that a word has been successfully matched by flashing it on the screen and by calling the routine 'tune'. The English translation of the word is then placed in the W$ array and in window #6 by 1570. The word type is registered in the T array and displayed on the screen, with the use of the WY, WX and WT$ arrays, at 1580. Finally the number of words translated, NR, is also incremented by line 1580.

Tune routine (Lines 1600-1620):

Here a facetious rendering of a suitably Germanic tune is given by the use of arrays D and P! The delay between notes generated by the empty FOR loop is made directly proportional to the duration of the previous note.

Failure 1 routine (Lines 1630-1660):

This routine is called if there are unidentified words in the sentence and provides a corresponding message in window #4.

Structure routine (Lines 1670-1710):

A vital part of the program is the identification of the structure of the sentence typed in. This is achieved by considering the word type for each word in turn. Line 1680 constructs a numeric code, ST, from the values now held in the word type array, T. This code is then compared with the known codes by the FOR loop, 1680, 1690. If a match is found, the appropriate code number is transferred to the variable SC.

A zero value of SC shows that the sentence structure has not been identified and so the routine 'failure 2' is called at line 1690. If is has been identified, 'translate' is called instead.

Translate routine (Lines 1720-1820):

Line 1730 gives a triumphant SOUND and clears part of the display to make room for the English translation. Lines 1740-1790 choose which routine should translate the sentence according to the code SC. Then 1800 calls 'coords' to calculate where the translated English words should appear on the screen. After this another message is placed in window #4 and finally line 1810 shows the translation.

Failure 2 routine (Lines 1830-1860):

This routine is called when the sentence structure is not identified.

Type 1 routine (Lines 1870-1900):

Here sentences of the form Article/Noun/Verb are translated. First it has to be established whether the subject of the sentence is singular or plural in order that the English words can be adjusted accordingly. This is done by looking at the German verb. Line 1880 sets the string variable, V$, to the verb, W$(1, 3), and calls the routine 'sing plural' to check the ending of V$ and to return the variable, S. A value of S = 0 indicates that the verb is plural and. similarly, S = 1, means that the verb is singular.

GOSUB 2210 places the English words into the alternative position in the W$ array. The German words are now no longer required and the routine 'coords' called by 'translate' assumes this position.

Finally the value of S is used in calls to 'Eng plural' to adjust the endings of the English words. Line 1890 provides the value of N needed by 'coords'.

Type 2 routine (Lines 1910-1950):

The sentence has the structure Pronoun/Sein/Article/Noun. The routine is identical in principle to 'type 1' except that line 1920 can establish more directly whether the subject is singular or plural.

Type 3 routine (Lines 1960-2000):

The sentence structure here is Article/Noun/Sein/Adjective.

Type 4 routine (Lines 2010-2050):

The structure is Sein/Article/Noun/Adjective.

Type 5 routine (Lines 2060-2100):

The structure is Article/Noun/Verb/Article/Noun. The routine sing plural is required at line 2070.

Type 6 routine (Lines 2110 -2150):

The sentence structure is Verb/Article/Noun/Article/Noun. Because here the corresponding English structure is quite different, lines 2120, 2130 are needed to generate the translation.

Sing plural routine (Lines 2160-2190):

Line 2170 decides whether the verb is singular or plural by looking at the final letter and setting S to 1 if it is a 'T'.

Exchange routine (Lines 2200-2230):

This routine places the English words, W$(2, I), in the W$ array into the position hitherto occupied by the corresponding German, W$(1, I).

Eng plural routine (Lines 2240-2330):

In an attempt to make INGA more intelligent, this procedure consists of a set of rules to construct the plurals of English words. Obviously it is difficult to provide for every possible case, but nevertheless a large number of words do fall under the conditions set here.

Line 2250 initialises the value of the flag, F. This is subsequently used to identify regular plurals.

Lines 2250, 2260 store the codes of the ultimate and penultimate letters of the word considered, CA and CB. Lines 2260-2290 then use the values of CA and CB to form the plurals of words which end in 'S', 'X', 'CH', 'SH' and 'Y'. Note that in the latter case line 2290 also checks for a preceding vowel.

Line 2300 forms the regular plural and 2310 deals with a specific irregular word.

Data (Lines 2340-2550):

The remaining lines of the program hold all the information required for the translation of a finite set of sentences. Obviously the area of the program's application can be adjusted here.

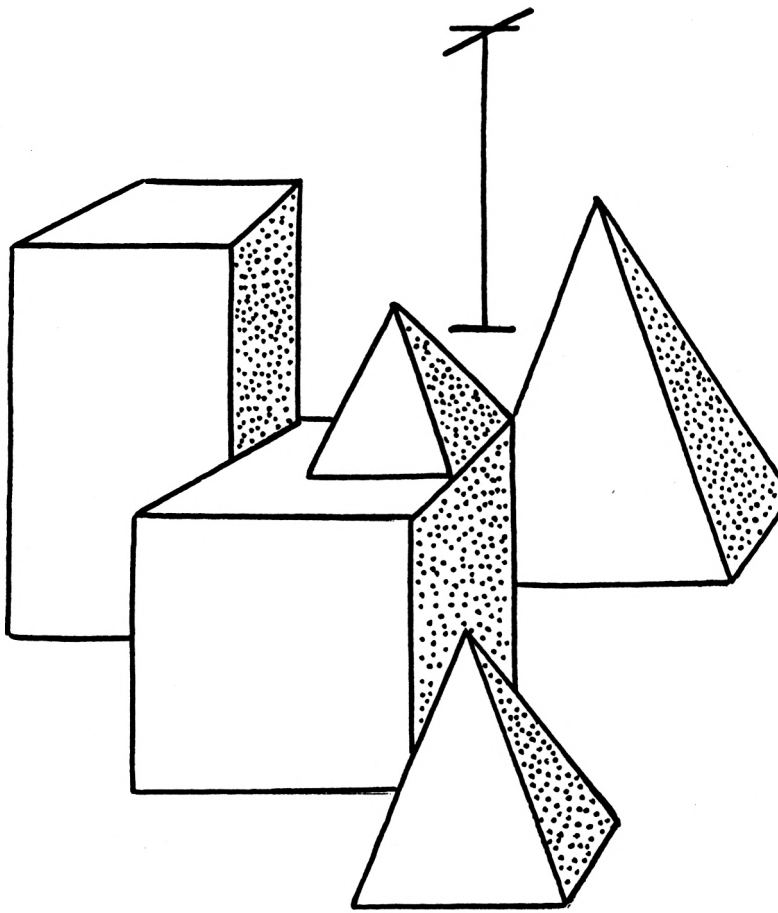
The Problem of Ambiguity

The problem which such naive translation programs cannot overcome is mainly one of ambiguity. The superficial structure of a sentence is frequently not its real meaning. A superb example given by one A.I. researcher was: 'Time flies like an arrow; fruit flies like a banana'. Such a sentence, if presented to a translation program, would yield immense problems. The identical grammatical structure of the two halves of the statement masks a total disparity of meaning. The English speaking human sees the ludicrous contrast immediately and probably grins. A computer program would require a hopelessly large store of cross-referenced information about language, and how it is applied to the world, before it could attempt to 'understand' what was meant.

We all become proficient at an early age with the way in which words can twist and change their meanings, Eliot's 'Words strain, Crack and sometimes break, under the burden'. One of the fascinating things about young children growing in maturity is the delight with which they rediscover for themselves the word play, and verbal sleight, that language provides. But this is not easy to program into a machine. Perhaps the only real solution will be some sophisticated learning program of the future which develops natural language in the way we do as we grow up. When computers can laugh at a joke, the problem of natural language will have been solved.

Winograd and SHRDLU

One way in which a computer can engage in sensible dialogue with a human is to give it an extremely limited world of discourse. In his famous program SHRDLU, Terry Winograd devised 'Blocks World' in which a PDP 10 manipulated a series of polyhedral shapes. They could be lifted and moved about by a robot arm and the program was capable of answering complicated questions about its actions, using remarkably natural English. The imagined world existed on the display of a DEC 340 (rather as, in a later program in this book, a robot is shown exploring a three dimensional view of a room.) The success of the program has, perhaps, concealed the fact that it involved only a very limited subset of the world we inhabit and which any true robot would also need to understand. Nevertheless it was an outstanding achievement and a breakthrough in interpreting language.



BLOCKS WORLD

CHAPTER 4

The last analysis: knowledge representation and semantics.

After the failure of direct translation programs, research in Artificial Intelligence turned to more sophisticated methods of persuading computers to respond to statements in ordinary language about the real world. Two main lines of approach were adopted. One was a more refined way of representing knowledge within a program. The other was a subtle method of analysing information presented to the computer.

The way in which knowledge is stored in a program will depend upon how the program is designed to operate, because different methods will be appropriate to different situations. In some programs, an internal map is used to relate distinct items. In others, information takes the form of sets of conditional rules. A third method of representation is to link separate items in a hierarchical, tree-like structure. In A.I. programs another method of representation frequently used is a list, with a list name and a series of items held under that name. A whole list can be just one item in another. Special list processing languages, like LISP, were developed specifically for this method of representation.

Frames

A particularly significant contribution was made by Marvin Minsky of the Massachusetts Institute of Technology in 1974. This was the concept of a 'frame'. A frame is a table-like structure which aids a program in understanding analogies. So much of the true semantic content of everyday utterances depends upon implicit analogies, that this is evidently a useful ability to give the computer. A frame for a particular subject consists of a set of 'slots' which can be filled with given attributes. Comparisons can then be drawn with frames for other subjects and the contents of respective slots matched against one another, according to selected heuristic rules:

```
'Mervyn is like a vandal.'

NAME ........ MERVYN                  NAME ........ VANDAL
ISA ......... MAN ..................  ISA ......... MAN
CHARACTER ... AGGRESSIVE ...........  CHARACTER ... AGGRESSIVE
HEIGHT ...... 5 FOOT 1                HEIGHT ...... 6 foot 7
BEHAVIOUR ... VIOLENT ..............  BEHAVIOUR     VIOLENT
PHYSIQUE .... SCRAWNY                 PHYSIQUE .... MUSCULAR
HOME ........ BOGNOR                  HOME ........ ASIA
```

A further development of the frame is a 'script'. Here is stored a generalised set of expectations and implications for a particular situation. So if a computer program was given the statements 'John invited Mary to tea.' and 'John hung mistletoe over the door.' it would consult its scripts concerning inviting people to tea and about hanging up mistletoe. It could then answer questions like 'Did John kiss Mary?' and 'What time of year was it?'

Blackboards

Another device for representing knowledge within a program is the 'blackboard'. This enables separate modules to store information quite independently of one another and to become active only when called upon by another part of the blackboard.

Parsing and Syntax

The analysis of language can be broken down into a number of distinct stages. In the first, the phonological, the computer actually analyses the sound waves reaching a microphone. This could be regarded as a task for the hardware specialist and, certainly, the A.I. worker frequently begins at the next stage. This is the morphological analysis in which individual words are analysed into their roots and endings. After this, a lexical analysis follows. Here parts of speech and plurals are identified. The fourth part of the analysis is the deduction of the syntactic structure of the sentence. Parsing algorithms examine what has been learnt so far about the utterance. It is at this stage that ambiguity can be tackled. Alternative structures are considered and the most appropriate choice made in the circumstances. Finally, the sentence reaches the stage of semantic analysis where its true meaning is attempted. At this point context becomes all important.

VALID

This program is intended to provide examples of both knowledge representation and semantic analysis. As the two screen dumps illustrate,

THIS PROGRAM SETS UP INFORMATION ON ANY SUBJECT THAT YOU CHOOSE. YOU CAN THEN INTERROGATE THE SYSTEM AND ASK IT QUESTIONS ABOUT WHAT IT 'KNOWS' OR REQUEST REASONS FOR THE ANSWERS THAT IT GIVES.

THE INFORMATION HELD AT ANY STAGE IS SHOWN GRAPHICALLY AS AN ARRAY ON THE SCREEN.

FOUR TYPES OF ENTRY ARE ALLOWED INTO THE SYSTEM AS FOLLOWS.

(1) A STATEMENT

EXAMPLE: Fish is / are nutritious.

(2) A REQUEST

EXAMPLE: Data on Lucy_Hopgood ?

(3) A QUESTION

EXAMPLE: Is arsenic harmful ?

(4) A QUERY

EXAMPLE: Reason why Tennents super ?

VALID will hold information on a subject chosen by the user. Four types of user entry are allowed. First, information can be typed directly into the program. This has to take the form of direct statements such as 'A-C5 is electric.' or 'Spiders are creepy'. Secondly, information on a subject can be obtained by a request: 'Data on weather?'. Direct questions are allowed, such as 'Is Southsea sunny?'. Finally the program can be asked to justify an answer it has given: 'Reason why pandas scarce ?'.

The method of knowledge representation used is a two dimensional array. Arrays are a realistic way of storing information in BASIC. In this program the knowledge base is also shown as a graphic table on the screen. This helps to indicate the way in which the array is searched for requred data.

Analysis is carried out within the program by a special parsing routine. This has been designed to deal with the four specific types of user-interaction and it demonstrates the way in which a degree of semantic analysis is possible. Thus if the question 'Is the-hamster hibernating?' is followed by another entry like 'Reason why ....?', the program will immediately assume that in this context the second question is also about the hamster. (Of course, it may not be!)

The print out shows how VALID is capable of a limited degree of reasoning. This is based upon the hypothetical syllogism. If the computer is given a series of propositions like: 'A is B; B is C; C is D; D is E', it will be able to deduce that A is E. This is scarcely genius level. It does, however, make the program a little more intelligent than a simple language translator.

VALID is humanised with one or two additional routines to make it appear more polite than it would otherwise be.

Commentary on VALID

The program is controlled by lines 1020, 1030. The routine 'initialisation' is called with GOSUB 1060 to set up some of the arrays and variables required during the execution of the program. Then the routine 'describe' provides an explanation of the program's operation. Finally the WHILE/WEND loop allows indefinite calls to 'parser' during each program run.

Initialisation routine (Lines 1050-1080):

Lines 1060, 1070 select MODE and colours. Line 1070 also establishes the D$ array which holds all the information acquired in any execution of the

```
STATEMENT, REQUEST, QUESTION, QUERY ?
IS MONTY_VINALL RATIONAL ?
I DON'T KNOW
STATEMENT, REQUEST, QUESTION, QUERY ?
EDUCATED IS RATIONAL
OKAY
STATEMENT, REQUEST, QUESTION, QUERY ?
TEACHERS ARE EDUCATED
OKAY
STATEMENT, REQUEST, QUESTION, QUERY ?
MONTY_VINALL IS A_TEACHER
OKAY
STATEMENT, REQUEST, QUESTION, QUERY ?
IS MONTY_VINALL RATIONAL ?
YES
STATEMENT, REQUEST, QUESTION, QUERY ?
REASON WHY YOU KNEW THAT ?
MONTY_VINALL IS A_TEACHER
A_TEACHER IS EDUCATED
EDUCATED IS RATIONAL
STATEMENT, REQUEST, QUESTION, QUERY ?
CATS ARE FLUFFY
OKAY
STATEMENT, REQUEST, QUESTION, QUERY ?
CATS ARE INTELLIGENT
OKAY
STATEMENT, REQUEST, QUESTION, QUERY ?
CATS ARE AFFECTIONATE
OKAY
STATEMENT, REQUEST, QUESTION, QUERY ?
JASPER IS A_CAT
OKAY
STATEMENT, REQUEST, QUESTION, QUERY ?
DATA ON JASPER
DATA IS:
1:JASPER
2:A_CAT
3:FLUFFY
4:INTELLIGENT
5:AFFECTIONATE
STATEMENT, REQUEST, QUESTION, QUERY ?
THANKS
PARDON ?
STATEMENT, REQUEST, QUESTION, QUERY ?
THANK YOU VALID
YOU'RE WELCOME
STATEMENT, REQUEST, QUESTION, QUERY ?
FORGOT TO SWITCH OFF THE PRINTER
PLEASE REPEAT
STATEMENT, REQUEST, QUESTION, QUERY ?
TOO LATE ... CLICK !
```

```
1000 REM VALID - PAT HALL, 2/86
1010 REM CONTROL ROUTINE
1020 GOSUB 1060:GOSUB 1100:WHILE 1>0
1030 GOSUB 1530:WEND:STOP
1040 REM
1050 REM INITIALISATION
1060 BORDER 0:MODE 1:INK 0,24:INK 1,26
1070 INK 2,6:INK 3,0:DIM D$(20,10):DIM
     D(20):TF=0:LQ=0:DIM W$(10):DIM H$(
     20):DIM E$(20):DIM P(20):RETURN
1080 REM
1090 REM DESCRIBE
1100 BORDER 6:PAPER 0:CLS:WINDOW #1,1,40
     ,1,3:PAPER #1,1:CLS #1:WINDOW #2,1,
     40,22,25:PEN #2,3:PAPER #2,1:CLS #2
     :PEN 3:LOCATE 16,2:PRINT"< VALID >"
     :LOCATE 3,6:PRINT"THIS PROGRAM";
1110 PRINT" SETS UP INFORMATION ON"
1120 PRINT"  ANY SUBJECT THAT YOU";
1130 PRINT" CHOOSE. YOU CAN"
1140 PRINT"  THEN INTERROGATE";
1150 PRINT" THE SYSTEM AND ASK"
1160 PRINT"  IT QUESTIONS ABOUT";
1170 PRINT" WHAT IT 'KNOWS'"
1180 PRINT"  OR REQUEST REASONS";
1190 PRINT" FOR THE ANSWERS"
1200 PRINT"  THAT IT GIVES." : PRINT
1210 PRINT"  THE INFORMATION HELD";
1220 PRINT" AT ANY STAGE IS"
1230 PRINT"  SHOWN GRAPHICALLY AS";
1240 PRINT" AN ARRAY ON THE"
```

```
1250 PRINT"  SCREEN." : PRINT
1260 PRINT"  FOUR TYPES OF ENTRY";
1270 PRINT" ARE ALLOWED INTO"
1280 PRINT"  THE SYSTEM AS FOLLOWS:"
1290 GOSUB 1430:LOCATE 4,5:PRINT"(1)"
1300 PEN 1:PAPER 2:LOCATE 14,5:PRINT
     " A STATEMENT ":PEN 3:PAPER 0:PRINT
     :PRINT"   EXAMPLE: Fish is  ";
1310 PRINT" are nutritious.":LOCATE 4,9
1320 PRINT"(2)":PEN 1:PAPER 2:LOCATE 15,
     9:PRINT" A REQUEST ":PEN 3:PAPER 0
1330 PRINT:PRINT"   EXAMPLE: Data on";
1340 PRINT" Lucy_Hopgood ?":LOCATE 4,13
1350 PRINT"(3)":PEN 1:PAPER 2:LOCATE 14,
     13:PRINT" A QUESTION ":PEN 3:PAPER
     0:PRINT:PRINT"   EXAMPLE: Is ars";
1360 PRINT"enic harmful ?":LOCATE 4,17
1370 PRINT"(4)":PEN 1:PAPER 2:LOCATE 16,
     17:PRINT" A QUERY ":PEN 3:PAPER 0
1380 PRINT:PRINT"   EXAMPLE: Reason why"
     ;:PRINT" Tennents super ?":GOSUB
     1430:GOSUB 1470:WINDOW #4,22,36,7,
     18:PEN #4,3:PAPER #4,1:CLS #4
1390 LOCATE 4,20:PRINT"KNOWLEDGE BASE"
1400 LOCATE 26,20:PRINT"NOTEPAD":RETURN
1410 REM
1420 REM PROMPT
1430 LOCATE #2,16,2:PRINT #2,"< SPACE >"
     :K=0:WHILE K=0:IF INKEY(47)=0 THEN
     K=1
1440 WEND:CLEAR INPUT:CLS #2:WINDOW #3,1
```

```
     ,40,4,21:PAPER #3,0:CLS #3:RETURN
1450 REM
1460 REM ARRAY
1470 GRAPHICS PEN 1:FOR I=107 TO 307
1480 MOVE 48,I:DRAW 268,I:NEXT:GRAPHICS
     PEN 3:FOR I=48 TO 268 STEP 22:FOR J
     =0 TO 2:MOVE I+J,107:DRAW I+J,307
1490 NEXT:NEXT:FOR I=107 TO 307 STEP 10
1500 FOR J=0 TO 2:MOVE 48,I+J:DRAW 268,
     I+J:NEXT:NEXT:RETURN
1510 REM
1520 REM PARSER
1530 CM=0:WHILE CM=0:IM=0:WHILE IM=0
1540 LOCATE #2,3,2:PRINT #2,
     "STATEMENT, REQUEST";:PRINT #2,
     ", QUESTION, QUERY ?":PRINT #2
1550 INPUT #2,S$:FOR T=1 TO 1500:NEXT
1560 CLS #2:ERASE W$:L=LEN(S$):IF MID$(
     S$,L-1,2)=" ?"THEN S$=LEFT$(S$,L-2)
     :L=L-2
1570 A$=MID$(S$,L,1):IF A$="?" OR A$="."
     THEN S$=LEFT$(S$,L-1):L=L-1
1580 N=1:FOR I=1 TO L:IF ASC(MID$(S$,I,1
     ))<>32 THEN W$(N)=W$(N)+MID$(S$,I,1
     ) ELSE N=N+1
1590 NEXT:IF N>2 THEN IM=1 ELSE PRINT #4
     ," PARDON ?":PRINT #4
1600 WEND:IF W$(2)="ARE" THEN L=LEN(W$(1
     )):W$(1)="A_"+LEFT$(W$(1),L-1):W$(2
     )="IS"
1610 IF W$(2)="IS" THEN CM=1:GOSUB 1700
```

```
1620 IF W$(1)="DATA" AND W$(2)="ON" THEN
     CM=1:GOSUB 1750
1630 IF W$(1)="IS" THEN CM=1:LQ=1:GOSUB
     1780
1640 IF W$(1)="REASON" AND W$(2)="WHY"
     AND LQ=1 THEN CM=1:LQ=0:GOSUB 1840
1650 IF W$(1)="THANK" AND W$(2)="YOU"
     THEN CM=1:PRINT #4,"YOU'RE ";:PRINT
     #4,"WELCOME!":PRINT #4:SOUND 1,239,
     20
1660 IF CM=0 THEN PRINT #4," PLEASE ";:
     PRINT #4,"REPEAT":PRINT #4
1670 WEND:RETURN
1680 REM
1690 REM STATEMENT
1700 PRINT #4," OKAY":PRINT #4:GOSUB
     1950:F=0:C=3:FOR I=0 TO TF:IF D$(I,
     0)=W$(1)THEN F=1:D(I)=D(I)+1:D$(I,D
     (I))=R$:X=D(I):Y=I:GOSUB 1990
1710 NEXT:X=0:Y=TF+1:IF F=0 THEN TF=TF+1
     :D$(TF,0)=W$(1):D(TF)=1:D$(TF,1)=R$
     :GOSUB 1990:X=1:GOSUB 1990
1720 RETURN
1730 REM
1740 REM REQUEST
1750 ERASE H$:H=1:H$(1)=W$(3):GOSUB 2040
     :PRINT #4," ";W$(1);" IS:":FOR I=1
     TO H:FOR T=1 TO 1000:NEXT:SOUND 1,
     239,10:PRINT #4,I:PRINT #4," ";H$(I
     ):NEXT:PRINT #4:RETURN
1760 REM
```

```
1770 REM QUESTION
1780 SR$=W$(2):GOSUB 1950:ERASE H$:H=1
1790 H$(1)=W$(2):GOSUB 2040:MF=0:FOR I=1
     TO H:IF H$(I)=R$ THEN MF=1
1800 NEXT:IF MF=1 THEN PRINT #4," YES"
     ELSE PRINT #4," I DON'T KNOW"
1810 SOUND 1,239,10:PRINT #4:RETURN
1820 REM
1830 REM REASON
1840 ERASE E$:E=0:PB$=R$:FF=1:GOSUB 1900
     :LF=0:WHILE LF=0:SOUND 1,239,10:E=E
     +1:E$(E)=PB$:GOSUB 2150:IF PA$=SR$
     THEN LF=1
1850 PB$=PA$:WEND:FF=0:GOSUB 1900:FOR T=
     1 TO 1000:NEXT:SOUND 1,239,10:PRINT
     #4," ";SR$;" IS ";:PRINT #4,E$(E)
1860 FOR I=E TO 2 STEP -1:FOR T=1 TO
     1000:NEXT:SOUND 1,239,10:PRINT #4
1870 PRINT #4," ";E$(I);" IS ";:PRINT #4
     ,E$(I-1):NEXT:PRINT #4:RETURN
1880 REM
1890 REM INDICATE
1900 IF FF=1 THEN PEN 3 ELSE PEN 0
1910  LOCATE 6,5:PRINT"SEARCHING!"
1920 RETURN
1930 REM
1940 REM PREDICATE
1950 L=LEN(S$)-LEN(W$(1))-LEN(W$(2))-2
1960 R$=RIGHT$(S$,L):RETURN
1970 REM
1980 REM CELL
```

```
1990 IF X<>0 THEN C=C-1
2000 GRAPHICS PEN C:RX=X*22+53:RY=310-Y
     *10:FOR R=RY TO RY+5:MOVE RX,R
2010 DRAW RX+15,R:NEXT:RETURN
2020 REM
2030 REM SEARCH
2040 FF=1:GOSUB 1900:FOR T=1 TO 1000
2050 NEXT:FOR I=1 TO 20:P(I)=0:NEXT:SM=0
     :WHILE SM=0:FS=0:FOR I=1 TO TF:Y=I
2060 N=H:FOR J=1 TO N:SOUND 1,239,10:X=J
     :IF D$(I,J)<>"" THEN C=1:GOSUB 1990
     : C=3:GOSUB 1990
2070 IF D$(I,0)=H$(J) AND P(I)=0 THEN P(
     I)=1:GOSUB 2120
2080 NEXT:NEXT:IF FS=0 THEN SM=1
2090 WEND:FF=0:GOSUB 1900:RETURN
2100 REM
2110 REM ADD PREDICATE
2120 FS=1:FOR K=1 TO D(I):H=H+1:H$(H)=D$
     (I,K):NEXT:RETURN
2130 REM
2140 REM FIND PREDICATE
2150 FOR I=1 TO 20:FOR J=1 TO 10:IF D$(I
     ,J)=PB$ THEN PA$=D$(I,0)
2160 NEXT:NEXT:RETURN
```

program. Twenty subjects are possible and up to ten predicates for each. The D array records how many predicates are assigned to any given subject. The variables TF and LQ initialised by 1070 are used subsequently in the program to count the total number of subjects introduced and to register whether the previous command given in the routine 'parser' was a question. Other arrays needed during the operation of the program, W$, H$, E$ and P, are also initialised..

Describe routine (Lines 1090-1410):

Line 1100 creates the initial screen, a yellow background with defined windows forming bands of white at the top and bottom of the display. It places the program title in black text and yellow background superimposed on the top band.

An initial explanation of the operation of the program is provided by lines 1100 - 1280. The routine 'prompt', GOSUB 1430 at line 1290 then prevents further execution of the program until the space-bar is pressed. After this, lines 1290-1380 give examples of the four types of command that VALID will accept. Because the program is fairly complicated in its execution, care is taken to make these examples as helpful as possible. Lines 1300, 1320, 1350 and 1370 state the four categories of command: statement, request, question and query. For emphasis, white text with red background is selected for each heading to contrast with the yellow of the screen. The examples themselves are PRINTed in black text and are designed to show the variety of commands possible. Thus the first example, `Fish is/are nutritious',` demonstrates that a singular or plural verb may be used in statements. The second example, `Data on Lucy-Hopgood`, illustrates that a multiple word subject will be accepted by VALID provided that no character spaces are included. The form that questions must take, with an initial "IS", is shown by the third example. Finally the fourth example indicates that an explanation for an answer given to a preceding question is prompted by instructions beginning "REASON WHY ..."

Line 1380 retains the screen display until the space-bar is pressed a second time and then calls the routine 'array' to place a 20 X 10 grid on the screen. This records how much information is typed into VALID and line 1390 places the headimg `KNOWLEDGE BASE` beneath the grid accordingly. Line 1380 also sets up a text window for all the program's output and this is indentified as the 'NOTEPAD' by 1400.

Prompt routine (Lines 1420-1450):

Here the use of WHILE/WEND loop, lines 1430, 1440 prevents further execution of the program until the space-bar is pressed. The screen is cleared without removing the program's title.

Array routine (Lines 1460-1510):

This routine places the grid, which represents the knowledge base of the program, on the screen. 200 cells are shown, each of which can be filled in by the subsequent routine 'cell' as information is acquired by VALID. The first FOR loop, lines 1470, 1480, places the white background for the grid on to the yellow screen. Then the two sets of nested FOR loops which follow, lines 1480, 1490 and lines 1490, 1500, add vertical and horizontal lines

Parser routine (Lines 1520-1680):

This is the routine which handles all the instructions typed in by the user. It consists of two nested WHILE/WEND loops, between lines 1530-1670, and 1530-1600.

The inner loop PRINTs in #2 a request for an instruction at line 1540. The user's response, the string S$, is accepted by 1550. The response is then analysed into individual words by lines 1580 - 1590. The words are placed into the array W$ and the total number of characters in S$ is established as the variable L.

It is quite possible that the string typed in by the user will have a final question mark or period, yet it is important that the program will correctly identify the true content of the string irrespective of such incidental punctuation. Line 1560 therefore removes a final question mark separated from the rest of the instruction by a single character space and, in a similar way, line 1570 removes a question mark or period which immediately follows the last word of the instruction. In either case the value of L is correctly adjusted.

Lines 1580, 1590 then analyse S$ into its component words. The number of words detected, N, is initialised by line 1580. The FOR loop which follows considers each character of S$ in turn. If it is not a space, identified as ASCII code 32, then the character is added to the current word, W$(N), being assembled. If it is a space, then the word total, N, is incremented and a new word is begun. In this way one pass through the loop completely analyses the word structure of the instruction, S$.

The next part of the analysis follows at line 1590. Only sentences of at least three words will be accepted. If N is less than 3, the WHILE/WEND loop will not terminate and the message, `PARDON` appears on the notepad. However, if three or more words have been typed, the program proceeds to the main body of outer loop.

Line 1600 reduces all instructions to singular form. Thus if the user has typed the statement, `FROGS ARE SLIMY`, this will be altered to `A-FROG IS`

SLIMY. Such a preliminary adjustment is needed before the instruction can be analysed further. Note that the two words of the subject are joined by the underline character. This is because, as stated earlier, VALID requires multiple word cells in its knowledge base to contain no character spaces.

Now 'parser' can decide which category of command it has been given. Line 1610 assumes that a statement is being made if the second word in the command is identified as 'IS'. It therefore calls the routine 'statement'. Line 1620 calls the routine 'request' if the first two words of the command are DATA ON. If the first word of the command is IS, line 1630 calls the routine 'question'. It also sets the last question flag, LQ, to 1. Finally, line 1640 will call the routine 'reason' if the command begins REASON WHY. This will only be allowed, however, if the flag LQ shows that the previous command was, indeed, a question.

VALID is made to seem a little friendlier by line 1650 which responds with YOU'RE WELCOME if the user has typed in a sentence beginning with the words THANK YOU. Finally, the program will reply with PLEASE REPEAT at line 1660 for any sentence which does not match one of the specified categories.

Statement routine (Lines 1690-1730):

First the user is informed by the message OKAY, given by line 1700, that the information typed in has been accepted. Then GOSUB 1950 uses the routine 'predicate' to decide what new data has been given about the chosen subject. For example, if the user entered the sentence IGOR-STRAVINSKY IS A-COMPOSER the routine 'predicate' would return the string A-COMPOSER for 'statement' to assign to IGOR-STRAVINSKY in an appropriate cell of the knowledge base.

Next VALID has to decide whether the information received is about one of the subjects it already contains in its knowledge base or if, instead, a new subject has been introduced. Line 1700 initialises the flag F, needed in the subsequent search. (It also gives a value to the variable C, used in the graphic routine 'cell'.) After this, the FOR loop between 1700, 1710 searches through the D$ array to see if the first cell in any row of the knowledge base matches the first word, W$(1), of the sentence typed in by the user. If this is the case, and an existing subject has been found, line 1700 alters the value of F to 1. Then the value of D(I) is incremented. This is the column number of the cell in the Ith row where the information will be stored. Line 1700 finally places the string, R$, returned by 'predicate' into this location in the D$ array.

If at the end of the FOR loop the subject has not been matched, the value of F will still be 0. Line 1710 then increments the value of TF, which records the total number of subjects held in D$. The same line also places the new subject, W$(1) into the first column of the new row of the array and the information about this subject, R$, into the second column.

In this way, the two routines contained in 'statement' systematically store all information given by the user. Calls are made by lines 1700 and 1710 to the routine 'cell' to show graphically how the knowledge is being classified.

Request routine (Lines 1740-1760):

This routine returns all the known information on any subject requested by the user. Line 1750 initialises the H$ array and places the selected subject, the third word typed by the user, into the first element of the array. After this, the routine 'search' adds to the array all the predicates associated, directly or indirectly, with the requested subject. Then the FOR loop lists out the H$ array on the notepad.

Question routine (Lines 1770-1820):

First line 1780 stores the subject of the question as the string SR$ in case a further enquiry is made by the user and 'reason' is called. Then 'predicate' is called to establish the string, R$, which is being tentatively linked with the subject. The subject is made the first element of the H$ array by line 1790 and 'search', called by GOSUB 2040, adds all the associated predicates to the array.

Line 1790 initialises the match-flag, MF. The FOR loop, 1790, 1800, checks through the H$ array to see if any of the predicates associated with the subject matches the proposed predicate, R$. If this is the case, 1790 changes the value of MF to 1. Finally, line 1800 PRINTs either `YES` or `I DON'T KNOW` in answer to the question, depending upon the value of MF. Line 1810 adds a musical tone with SOUND.

Reason routine (Lines 1830-1880):

In accord with the general A.I. principle that programs should provide adequate justification for their output, this routine can state why it has associated a particular predicate with a given subject.

It does this by searching backwards from the predicate of the previous question, equated to PB$ at line 1840, until the subject, SR$, is found. This occurs in the WHILE/WEND loop. Here, with GOSUB 2150, the routine

'find predicate' locates the subject PA$ of PB$ in the array. Now PA$ becomes the new PB$, at 1850, and the search is carried out again until eventually the desired subject is reached and the loop ends.

Throughout the routine, all predicates found are placed into the E$ array by 1840. This is then used by 1850-1870 to display the logic involved in the result the program had originally stated.

Indicate routine (Lines 1890-1930):

This routine places a message on the screen when the knowledge base is being interrogated.

Predicate routine (Lines 1940-1970):

This routine is called by 'statement' and 'question'. It provides the predicate of the user's original sentence by removing the first two words at line 1960.

Cell routine (Lines 1980-2020):

Here MOVE and DRAW are used in the FOR loop to fill one element of the knowledge base on the screen, at (X, Y), in colour C.

Search routine (Lines 2030-2100):

The routine is called by both 'request' and 'question' and checks the knowledge base, D$, to return all predicates associated, either directly or indirectly, with the subject that has been placed in the first location of the H$ array, at line 1750 or 1790. Because the checking routine repeats until a value of SM = 1 occurs at 2080, a predicate found on one pass through the routine can locate another on the next. This makes the searching particularly thorough. All predicates found are placed in H$.

The routine is governed by the WHILE/WEND loop, lines 2050 - 2090. The I loop examines each of the subjects, total TF, in the D$ array. The J loop then checks N locations in the H$ array. N begins as 1 when the routine is first called, but is constantly incremented, via H, by 'add predicate'. In this way, every predicate added to H$ is compared with the subjects in the D$ array, at 2070. If a match is found, 'add predicate' is immediately called and all the predicates linked with this new subject are added to the H$ array as well. (The P array prevents the same subject being counted twice.)

This continues until the flag FS indicates that no further matching is taking place. The loop then ends at 2090. Throughout the routine, SOUND and

GOSUB 1990, at line 2060, show the user that searching is taking place. Similarly 'indicate' at 2040 is employed to place a message on the screen.

Add predicate routine (Lines 2110-2130):

When line 2070 matches a predicate in the H$ array with one of the subjects, D$(I, 0) of the D$ array, this routine uses the FOR loop to add all the predicates associated with D$(I, 0) to the H$ array. This means that no logical connection between the various pieces of information given to the program will be missed.

Find predicate routine (Lines 2140-2160):

This is called by 'reason' to find the subject, PA$, of any predicate, PB$, in the D$ array. This is done quite simply by the nested FOR loops, lines 2150, 2160.

CHAPTER 5

Expert assistance: knowledge engineering.

Expert systems are the new factor which have entered the world of Artificial Intelligence and attracted cheque books from the world of commerce. Ironically, after the Lighthill Report was critical about the future of A.I. research, many British workers crossed to the States to continue their research and have thus contributed to the G.N.P. elsewhere.

An expert system has been mentioned already, Douglas Lenat's EURISKO, and it has been seen that expert systems and programs which play games have much in common. Specifically it is the way in which knowledge is represented, and then manipulated according to heuristic rules, which gives success to either type of program. In an expert system the information stored is the 'knowledge base', as already seen in the program VALID, and the set of heuristic rules forms the 'inference engine'. Similar inference engines can be applied to different knowledge bases and thereby form a new expert system. However a tailor made inference system/ knowledge base is usually more efficient for a particular task.

A principle used in the design of an inference engine is structured selection in which hypotheses are linked with associated evidence. Three types of strategy can be employed. One is 'forward chaining' in which the engine begins with initial evidence and makes a hypothesis. Further evidence then leads to another hypothesis and the process continues until a conclusion is reached. In 'backward chaining' the system begins with a hypothesis and determines what preconditions will be necessary for this hypothesis to be true. In a rule-value approach the engine deals with a complicated, interconnected set of hypotheses and related evidence by asking questions about those hypotheses which will maximise the change further on in the set and remove the most uncertainty from the system.

An important feature of an expert system is that it can learn for itself about a particular domain of enquiry without a human programmer typing in every precise detail. It should be able to associate important items without excessive prompting. The next program gives a simple example of a learning system which manages to do this.

AMY

This program can learn to distinguish between two different subjects by asking a series of questions. The user decides what the two subjects should be and what questions (maximum of five) can be asked, but does not indicate further how the questions are related to the subjects. For example the two subjects might be 'petrol' and 'water' and one of the questions could be 'Do firemen squirt it on your house?'. The program is not informed about the relation between firemen, petrol and water. It does, however, eventually learn that if firemen squirt a liquid on your house then the liquid will be water rather than petrol.

In order to see how this works it is best to RUN the program and watch the repeated updating of the screen display as AMY develops a rule for each set of questions and answers it is given. The program contains, as DATA, three standard examples which can be used for an initial demonstration.

```
< AMY >
TOTAL: 0 + 0 + 0 + 1 + 1 = 2
POSITIVE VALUE IMPLIES: MAX DAVIES
```

REPLY	VALUE	RULE	RULE X VALUE
YES	1	0	0
YES	1	0	0
NO	0	-1	0
YES	1	1	1
YES	1	1	1

```
1000 REM AMY - PAT HALL, 1/86
1010 REM CONTROL ROUTINE
1020 GOSUB 1090:GOSUB 1130:WHILE TIME>0
1030 GOSUB 1160
1040 IF K=0 THEN GOSUB 1280
1050 IF K=1 THEN GOSUB 1370
1060 WEND
1070 REM
1080 REM INITIALISATION
1090 MODE 1:BORDER 2:INK 0,0:INK 1,24:
     INK 2,26:INK 3,6:GRAPHICS PEN 0
1100 DIM Q$(5):DIM A(5):DIM R(5):DIM
     P(5):EX=0:RETURN
1110 REM
1120 REM DISPLAY
1130 PEN 0:PAPER 1:CLS:WINDOW #1,1,40,1,
     3:PAPER #1,2:CLS #1:LOCATE 16,2:
     PRINT"< AMY >":WINDOW #2,1,40,4,6:
     PEN #2,2:PAPER #2,3:WINDOW #3,1,40,
     4,25:PAPER #3,1:WINDOW #4,1,40,22,
     25:PEN #4,0:PAPER #4,1:RETURN
1140 REM
1150 REM CHOICE
1160 CLS #3:LOCATE 4,10:PRINT"THIS ";
1170 PRINT"PROGRAM GIVES AN EXAMPLE ";
1180 PRINT "OF A":LOCATE 4,11:PRINT
     "LEARNING SYSTEM.":LOCATE 7,13
1190 PRINT"DEMONSTRATION ...  PRESS D."
1200 LOCATE 7,15
1210 PRINT"OWN EXAMPLE .....  PRESS E."
1220 CH=0:WHILE CH=0
```

```
1230 IF INKEY(61)=0 THEN K=0:CH=1
1240 IF INKEY(58)=0 THEN K=1:CH=1
1250 WEND:CLS #3:CLEAR INPUT:RETURN
1260 REM
1270 REM DEMONSTRATION
1280 N=5:EX=EX+1:DE=EX MOD 3
1290 IF DE=0 THEN RESTORE 1960
1300 IF DE=1 THEN RESTORE 2070
1310 IF DE=2 THEN RESTORE 2180
1320 READ A$,B$:FOR I=1 TO 5:READ D$,E$,
     R(I):Q$(I)=D$+E$:NEXT
1330 GOSUB 1520:CLS #4:PRINT #4,TAB(16)
     "< SPACE >":CH=0:WHILE CH=0:IF
     INKEY(47)=0 THEN CH=1
1340 WEND:CLEAR INPUT:RETURN
1350 REM
1360 REM LEARN
1370 GOSUB 1420:RL=0:WHILE RL=0:GOSUB
     1520:GOSUB 1720:CLS #4:PRINT #4,
     "    PRESS SPACE TO RETURN TO MENU"
     :GOSUB 1690
1380 IF INKEY(47)=0 THEN RL=1
1390 WEND:CLEAR INPUT:RETURN
1400 REM
1410 REM QUESTIONS
1420 LOCATE 2,5:PRINT"TYPE 1ST ANSWER ";
     :INPUT A$:LOCATE 2,7:PRINT
     "TYPE 2ND ANSWER ";:INPUT B$:LOCATE
     2,9:PRINT"TYPE HOW MANY QUESTIONS"
1430 RN=0:WHILE RN=0:LOCATE 26,9
1440 INPUT N:IF N>0 AND N<6 AND N=INT(N)
```

```
     THEN RN=1
1450 WEND:FOR I=1 TO N:PEN 0:PAPER 2
1460 LOCATE 2,I*2+9:PRINT"QUESTION ";I;
     ":":PEN 3:PAPER 1:PRINT" ";
1470 QL=0:WHILE QL=0:INPUT Q$(I)
1480 IF LEN(Q$(I))<36 THEN QL=1
1490 WEND:NEXT:PEN 0:RETURN
1500 REM
1510 REM ANSWER
1520 CLS #3:FOR I=0 TO N+1:Y=I*32
1530 MOVE 20,280-Y:DRAW 600,280-Y
1540 MOVE 20,280:DRAW 20,280-Y:MOVE 137,
     280:DRAW 137,280-Y:MOVE 264,280:
     DRAW 264,280-Y:MOVE 374,280:DRAW
     374,280-Y:MOVE 600,280:DRAW 600,280
     -Y:NEXT
1550 LOCATE 3,9:PRINT"REPLY":LOCATE 11,9
     :PRINT"VALUE":LOCATE 19,9:PRINT
     "RULE":FOR I=1 TO N:LOCATE 20,I*2+9
     :PRINT R(I):NEXT
1560 PRINT #4," CHOICE: ";A$;" OR ";B$
1570 PRINT #4,TAB(8)"PRESS: Y ... YES";
1580 PRINT #4,"  N ... NO":GOSUB 1690:
     CLS #2
1590 FOR I = 1 TO N:PRINT #2
1600 IF RIGHT$(Q$(I),1)<>"?" THEN Q$(I)=
     Q$(I)+"?"
1610 PRINT #2," ";Q$(I):PRINT #2:GOSUB
     1770:A(I)=D:LOCATE 4,I*2+9:PRINT C$
     :LOCATE 12,I*2+9:PRINT A(I):NEXT
1620 OP=3:CLS #2:GOSUB 1830:T=0:GOSUB
```

```
     1690:PRINT #2," TOTAL: ";:FOR I=1
     TO N-1:T=T+P(I):PRINT #2,P(I);" + "
     ;:NEXT
1630 T=T+P(N):PRINT #2,P(N);" = ";T
1640 PRINT #2," VALUE IMPLIES: ";
1650 IF T<0 THEN PRINT #2,B$:OP=1 ELSE
     PRINT #2,A$:OP=2
1660 RETURN
1670 REM
1680 REM PAUSE
1690 FOR TP=1 TO 1500:NEXT:RETURN
1700 REM
1710 REM INSTRUCT
1720 GOSUB 1690:PRINT #2,
     " IS THIS CORRECT ?"
1730 GOSUB 1770:IF D=0 THEN GOSUB 1830
1740 RETURN
1750 REM
1760 REM ASK
1770 CH=0:WHILE CH=0
1780 IF INKEY(46)=0 THEN D=0:C$="NO":
     CH=1
1790 IF INKEY(43)=0 THEN D=1:C$="YES":
     CH=1
1800 WEND:CLEAR INPUT:RETURN
1810 REM
1820 REM CALCULATE
1830 IF OP=1 THEN OP$=" + "
1840 IF OP=2 THEN OP$=" - "
1850 IF OP=3 THEN OP$=" X "
1860 LOCATE 26,9:PRINT"RULE";OP$;"VALUE"
```

```
    :FOR I=1 TO N:Y=I*2+9
1870 IF OP=1 THEN P(I)=R(I)+A(I)
1880 IF OP=2 THEN P(I)=R(I)-A(I)
1890 IF OP=3 THEN P(I)=R(I)*A(I)
1900 GOSUB 1690:SOUND 1,119
1910 PAPER 2:LOCATE 27,Y:PRINT R(I);OP$;
    A(I):GOSUB 1690:PAPER 1
1920 LOCATE 27,Y:PRINT SPC(9):LOCATE 30,
    Y:PRINT P(I):IF OP<>3 THEN R(I)=
    P(I)
1930 NEXT I:RETURN
1940 REM
1950 REM DATA
1960 DATA CAT,DOG
1970 DATA "DOES IT MIAOW "
1980 DATA WHEN IT'S HUNGRY, 1
1990 DATA "DOES IT CHASE POSTMEN "
2000 DATA UP THE DRIVE, -1
2010 DATA "DOES IT THINK IT "
2020 DATA CAN CLIMB TREES, 1
2030 DATA "IS IT GOOD AT DIGGING "
2040 DATA UP THE GARDEN, 0
2050 DATA "IS IT WRITING A POEM "
2060 DATA ON T.S.ELIOT, 1
2070 DATA MAX DAVIES,HENZE
2080 DATA "IS HE A CONTEMPORARY "
2090 DATA COMPOSER,0
2100 DATA "HAS HE WRITTEN MORE "
2110 DATA THAN ONE OPERA,0
2120 DATA "DID HE COMPOSE 'THE' "
2130 DATA YOUNG LORD,-1
```

```
2140 DATA "DOES HE LIVE ON HOY "
2150 DATA IN THE ORKNEYS,1
2160 DATA "DID HE COMPOSE "
2170 DATA 'TAVERNER',1
2180 DATA JACK DANIELS,JIM BEAM
2190 DATA "IS IT A BOURBON "
2200 DATA WHISKEY,0
2210 DATA "DOES THE BOTTLE HAVE "
2220 DATA A WHITE LABEL,-1
2230 DATA "IS IT PRODUCED IN "
2240 DATA TENNESSEE,1
2250 DATA "WAS IT FIRST MADE "
2260 DATA IN 1795,-1
2270 DATA "DOES IT HAVE A SUBTLE "
2280 DATA NUTTY FLAVOUR,1
```

Commentary on AMY

The control routine for AMY resides between lines 1020-1060. The routine 'initialisation' is called at 1020 to set up the arrays required by the program and 'display', called as well, establishes the initial screen. The WHILE/WEND loop, lines 1020-1060, then allows the user either to choose examples of simple learning systems already held by the program as DATA or to set up a new system entirely. The alternatives are given by the routine 'choice'. This returns a value for the variable K which is then used by lines 1040, 1050 to call 'demonstration' or 'learn'.

Initialisation routine (Lines 1080-1110) :

Here four arrays are established. Q$ can hold five questions up to 35 characters in length. A is the array which stores the answers typed in by the user, coded as 1 for 'Yes' and 0 for 'No'. R is the rule which the computer develops to make its decision and the array P is used to hold intermediate results generated by 'calculate'.

Line 1100 initialises the variable EX, which is used in 'demonstrate' to select which set of DATA to employ.

Display routine (Lines 1120-1140) :

The whole screen is set to yellow with black text. Line 1130 uses WINDOW #1 to place a white strip across the top of the screen and the program's title is PRINTed centrally here in black on a yellow background. WINDOW #2 is defined with white text and a red background just beneath the title but at this stage of the program the window is not seen. WINDOW #3 is set up so it can be used subscquently for clearing the central section of the display. WINDOW #4 is established at the bottom of the screen for further text and prompts.

Choice routine (Lines 1150-1260) :

The user can decide here whether to see an example of a learning system from the program's DATA or to investigate the way in which the computer builds up its rule array with fresh subject matter and questions.

Line 1160 uses CLS #3 to clear any previous text from the screen without removing the program's title. The alternatives available are described by 1160 - 1210. After this the WHILE/WEND loop between lines 1220-1250 terminates only when 'D' or 'E' is pressed. The value of K then obtained is used by the control routine. The screen is cleared again by CLS #3 at line 1250.

Demonstration routine (Lines 1270-1350) :

This routine uses one of the three sets of DATA to show the user the rule-based principle involved in the learning system. The maximum number of questions is required for these examples and so line 1280 makes N = 5. Each time the routine is called, the next set of DATA is used. This means that three separate examples can be given before a repetition occurs. The sequential selection is achieved by incrementing the variable EX at line 1280 whenever the routine is executed. The expression DE = EX MOD 3 will then have the value 0, 1 or 2 and so the DATA pointer will be restored to

lines 1960, 2070 or 2180, which correspond with the separate examples stored at the end of the program. Naturally more than three examples could be placed in DATA. If this is required the expression could become, perhaps, (EX MOD 5).

Once the DATA pointer is in an appropriate position, line 1320 READs the two subjects as A$ and B$. The FOR loop then READs each question in turn in two halves, as the strings D$ and E$, and concatenates them into the question Q$(I). The reason for this is simply to have a neat listing. The loop also places the appropriate value, 1, 0 or -1, into the rule array, R(I).

The routine 'answer' is called at line 1330 with GOSUB 1520 to present each question to the user, obtain a reply and finally make a decision about the subject.

Further execution of the program is delayed until the space-bar is detected as INKEY(47) at 1330, but then 'demonstration' ends and the program returns to the control routine. The menu is placed on the screen again by 'choice'.

Learn routine (Lines 1360-1400) :

In this routine the user can set up a fresh learning system and see how the rule array evolves into a set of values which always generates a correct decision.

Line 1370 calls 'questions' to request the two subjects and the questions needed for the new system. The WHILE/WEND loop between lines 1370 - 1390, then uses the routine 'answer' to test the user's response to each question and to generate a decision accordingly. After this, 'instruct', called by GOSUB 1720 at line 1370 modifies the rule array if an incorrect decision has been given.

The loop will repeat indefinitely and develops a rule array which always gives a correct decision. At this stage the user will probably wish to experiment with a new system and so line 1380 gives the opportunity of exit from the loop if the space-bar is pressed when the prompt to do so appears in window #4.

Questions routine (Lines 1410-1500) :

The routine requests the two subjects for the new system at lines 1420. The WHILE/WEND loop, lines 1430-1450 then asks for a number of questions between 1 - 5. Naturally non-integral numbers are rejected, by line 1440.

The FOR loop, lines 1450 - 1490, places the questions into the Q$ array. The embedded WHILE/WEND loop places an upper limit on the length of the questions to prevent the screen display from being disrupted.

Answer routine (Lines 1510-1670) :

The principal aim of AMY is realised in this procedure, which is therefore the core of the program. First a table is displayed on the screen with columns for the user's replies, the current rule being employed by the program and any calculation which needs to be carried out. Then, as the user responds to the questions presented in window #2, the replies are filled in (both as words and as value of 1 and 0). Next, the program details the calculations it is making in the final column and displays its decision. If this is incorrect, the modification to the rule array is also shown being calculated.

Line 1520 clears any earlier table from the screen with CLS #3. MOVE and DRAW are used in the FOR loop between lines 1520-1540 to produce the new table. The dimensions of the table are determined by the number of questions, N. Headings are added by line 1550.

The current rule array is displayed in the third column of the table by the FOR loop at line 1550. Each time 'answer' is called these values are seen to be converging on a final rule which, eventually, always makes a correct decision.

The two possible subjects are PRINTed in window #4 by lines 1560 - 1580. After this the FOR loop, lines 1590-1610, presents all N questions in window #2. Line 1600 adds a question mark if needed. At line 1610 the routine 'ask' is called with GOSUB 1770 to obtain a yes/no answer to the question. The appropriate value is placed in the A array by 1610, which also displays the yes/no answer and its corresponding 1/0 value on the table.

AMY now has sufficient information to make a decision between the two possible subjects, A$ and B$. The calculation is performed by the routine 'calculate', called at 1620 with GOSUB 1830. The variable OP is given the value 3 because multiplication is the operation required. Lines 1620, 1630 display, in window #2, the total of all the intermediate results returned by 'calculate' and lines 1640, 1650 finally PRINT the decision determined by this total. Line 1650 also obtains the value of OP needed for the next call to 'calculate'.

Pause routine (Lines 1680-1700) :

The empty FOR loop delays the operation of the program when a pause is required.

Instruct routine (Lines 1710-1750) :

This routine asks the user if the decision made is correct. If it is not, 'calculate' is called once more, this time to adjust the rule array.

Ask routine (Lines 1760-1810) :

Here the keyboard is scanned until either 'Y' or 'N' is pressed. Appropriate values are then given to C$, 'YES' or 'NO', and to D, 1 or 0.

Calculate routine (Lines 1820-1940) :

This routine will either add, subtract or multiply the rule array by the answer array, depending upon whether the rule array is being adjusted or a decision being made. The type of operation employed is controlled by the variable OP.

Lines 1830-1850 determine the arithmetical sign to show for the final column of the table. It is displayed by 1860.

The FOR loop, 1860-1930, performs the necessary calculation for each answer in turn and displays it on the table. The lines between 1870-1890 carry out the calculation prescribed by OP. The calculation is shown, and the result then displayed, on the table by lines 1900-1920. Line 1920 alters the rule array when required.

Data (Lines 1950-2280) :

Information for three separate examples is held between 1960 - 2280.

Accountability and Expert Systems

A feature included in expert systems is accountability. The program is expected to be able to give reasons for the decisions it has made and must be capable of interrogation by the user. The principle was included in the routine 'reason' in the program VALID of Chapter 4. Here the program did a little backward chaining of its own through its knowledge base in order to establish the logic behind its previous answer. Some people argue that the necessity of including accountability in A.I. programs will imply an upper limit to the computer's power of reasoning, preventing it from ever exceeding, if it was ever in a position to, the human operator's ability to understand what it was doing.

Probably the earliest working expert system is DENDRAL. This was developed at Stanford University. It asks questions about the chemical properties of a substance, and more refined information like nuclear magnetic resonance data, and is able to deduce a possible structure for the substance from the many possibilities. Another expert system, PROSPECTOR, can generate probabilities of ore deposits in given locations from geological evidence. PUFF is a medical program which diagnoses respiratory illnesses and MYCIN, from which PUFF derived its inference engine, similarly diagnoses blood infections.

Human culture has evolved because of the way information is stored. Writing, then libraries, then the printing press, all helped to preserve knowledge and to allow people to understand more and more about the world. With expert systems, we are witnessing an exciting new extension of this principle. Not simply information can now be passed from generation to generation, but also the very skills of handling and interpreting that information. Unlike doctors and geologists, expert systems can be reproduced and taken wherever they are needed. And also unlike the human expert they do not die. In the future there will be people cured of illnesses which have been diagnosed by expert systems developed now. This will be long after the original, human experts who helped (with the programmers) to create the system themselves died. This is an incredible prospect.

CHAPTER 6

Playing the game: heuristic strategies

The winner of the world backgammon championship held at Monte Carlo in 1979 was challenged to play against a robot for a prize of 5000 dollars. The robot had been supplying an amusing distraction for the players during the earlier games and had seemed merely a novelty. However an assistant behind the scenes was controlling it by radio and was, in turn, in direct satellite link with Carnegie-Mellon University. Here, Hans Berliner was interested to find out how well his backgammon playing program, Mighty Bee, would compete at championship level. The program won.

Game Playing and AI

From the earliest days of Artificial Intelligence, game playing has been regarded as a useful testbed of ideas and stategies. The limited 'microworld' provided by a game, with its clearly defined rules and finite number of goals, is ideally suitable for the internal representation of a program and yet, at the same time, involves aspects of human reasoning which the A.I. researcher is anxious to investigate.

Search Trees

In any game, the state of play can be represented by an inverted tree structure. At the top (the root!) is the current game position. This is usually referred to as 'ply 0'. Branching out beneath these are all the possible states of play which could immediately follow the current state. Depending upon the game, this could be adding a cross to the centre square, taking a knight, or, perhaps, occupying a hexagon. All these positions form ply 1. Similarly the possibilities for the subsequent stages of the game will be ply 2, ply 3... Clearly in all games the tree will branch out very rapidly. In some simple games, like noughts and crosses, it is possible to show the whole of the tree structure. In this sense, the game is deterministic. For most games, however, the tree structure soon becomes too broad to be able to do this. Chess. for example, has a branching factor of approximately 30 and other games can have even higher factors.

It is the tree structure which controls the way in which a program plays a game. From each move made, the computer has to look ahead, examine the game tree and decide the best possible decision to make. But even the vast computational power of the largest mainframe machines cannot be expected to cope with the exponential growth of the tree structure. The world's most powerful computer, a Cray, running a chess program called Blitz was defeated by chess master, David Levy. Another leading chess-playing program, Belle, developed at AT&T Bell Laboratories by Joe Condon and Ken Thompson, requires a dedicated computer with special components and is able to consider 160 000 positions per second. This has taken it to expert level in its tournament rating, but it is clearly playing chess in a different way to a human's approach. Unlike a program, which looks at a large number of tree branches but only to a few ply ahead, a good human player apparently considers just a few possible routes through the tree and investigates these to a far greater depth. This is only possible because the human mind can automatically eliminate those branches which are not going to lead to successful game positions. The computer, instead, needs to consider a far greater number of branches. This is another example of the way the human mind and the computer program usually employ totally different approaches to the same problem.

Evaluation Functions

At each stage of its play, a program has to use an 'evaluation function' to decide, in quite a formal, mechanical fashion, a score for each potential move. The function normally takes the shape of a simple polynomial incorporating various aspects of the game in hand. The number of terms involved will decide how good a game is played and in some programs the evaluation function is capable of modifying itself in order to improve the program's ability. It does this by considering its past performance.

Heuristics

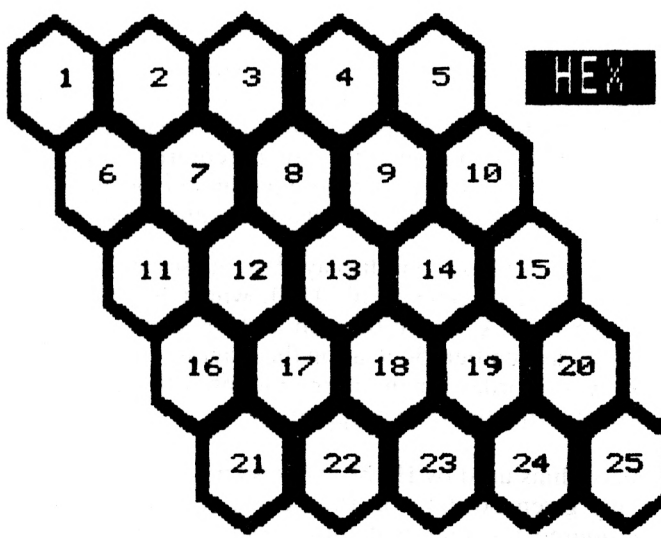
The evaluation function alone is insufficient to create a feasible game strategy. The program needs to employ some heuristic rules to reduce the number of possible branches of the game tree it must consider. In this way it comes closer to the human approach, although the heuristics involved are unlikely to be the same. However, in one remarkable example, a poker-playing program developed by a team at the State University of New York, Buffalo, discovered for itself the importance of bluffing about the hand it was playing.

A heuristic rule devised very early in the history of A.I. is the principle of 'minimaxing'. This was invented by Claude Shannon in 1949. In it the

computer evaluates all positions several ply ahead. It assuumes that its opponent is playing an error free game and therefore accepts that alternate ply will minimise its own advantage. With this precondition, it looks for the route which will maximise its position for the other ply. Minimaxing is a strategy which has been employed widely by programmers. A further refinement is the 'alpha-beta' algorithm, which reduces the amount of searching required by drastically pruning the game tree. Any branch stemming from a poor move on the part of the computer is eliminated from further investigations.

HEX

The first program in this chapter is the game of Hex. This was invented by the Danish mathematician and poet, Piet Hein, in 1942, although the name was chosen by the American firm of Parker Brothers, 'Nash' and 'John' being other contenders. It is now better known in this country as the television programme of 'Blockbusters'. The interesting aspect of the game is that nobody has yet specified how large a board should be used! In this version for computer, a 5 X 5 set of 25 hexagons has been chosen, as shown in the screen dump.



The game's object is quite simple. Each player has to complete a line of hexagons from one side of the board to the other, taking alternate moves. Depending upon the size of the board, varied strategies can be devised and Hex can become very involved indeed. Fortunately a 5 X 5 board seems to be the point at which it becomes interesting.

The program HEX uses a heuristic strategy which concentrates upon aggressive rather than defensive play. It can deal effectively with filled hexagons obstructing its intended path but does not attempt to block the other chain. The computer places the initial hexagon of its chain as close to the centre of the board as possible and then extends alternate ends of the chain until either side of the board is reached. Its opponent plays in a vertical direction. The simplicity of the heuristic means that depth searching is not required.

Commentary on HEX

The program is controlled by lines 1020-1080. Line 1020 calls the initialisation routine with GOSUB 1100 to set up arrays and variables necessary for the program. Then 'board' creates the screen display and 'first turn' decides whether the user or computer makes the initial move. After this, the WHILE/WEND loop, 1030-1070, calls the routines 'check side', 'user move', 'computer move' and 'show move', to play a game of Hex.

Initialisation routine (Lines 1090-1130) :

Line 1100 selects MODE and screen colour. Then line 1110 sets up the H array. The values placed in this array indicate whether a particular hexagon is vacant, value 0, occupied by a user move, value 1, or by a computer move, value 2. Only twenty five hexagons are needed for the game but the playing strategy involved means that an extra row is required at the top and the bottom of the board. The H array for these redundant locations is filled with values of 3 to prevent the computer from attempting to use one of them.

The coordinates for the hexagons displayed on the screen are held in arrays XH, YH, established by lines 1110, 1120, which READ the necessary information from DATA. It can be seen that the computer identifies the possible hexagons as numbers 5-29, as on the diagram. These values need to be decreased by 4 in order for the user to identify them sensibly and this correction occurs throughout the program.

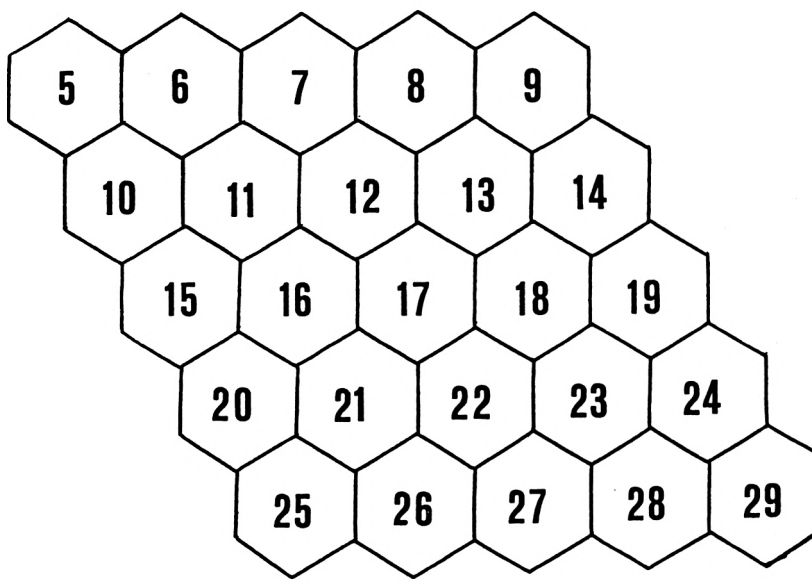
One of the algorithms used by HEX involves knowing which hexagons are adjacent to any given position on the board. It can be seen from both diagrams considered together that this amounts to knowing six possible increments, both positive and negative, in the H array. These are also held as DATA and placed into the 'priority' array, P, by line 1120.

```
1000 REM HEX - PAT HALL, 1/86
1010 REM CONTROL ROUTINE
1020 GOSUB 1100:GOSUB 1150:GOSUB 1290
1030 WHILE TIME>0:GOSUB 1350
1040 IF T=0 THEN GOSUB 1410 ELSE GOSUB
     1490
1050 GOSUB 1670:GOSUB 1350
1060 IF T=0 THEN GOSUB 1490 ELSE GOSUB
     1410
1070 GOSUB 1670:WEND
1080 REM
1090 REM INITIALISATION
1100 MODE 1:INK 0,0:INK 1,26:INK 2,20:
     INK 3,2
1110 DIM H(34):FOR I= 0 TO 4:H(I)=3:H(I+
     30)=3:NEXT:DIM XH(29):DIM YH(29)
1120 FOR I=5 TO 29:READ XH(I),YH(I):NEXT
     :DIM P(6):FOR I= 1 TO 6:READ P(I):
     NEXT:CM=1:N=0:LS=0:RS=0:RETURN
1130 REM
1140 REM BOARD
1150 PEN 0:PAPER 2:CLS:PAPER 1:BORDER 20
     :LOCATE 32,3:PRINT SPC(5):LOCATE 32
     ,4:PRINT" HEX ":LOCATE 32,5:PRINT
     SPC(5)
1160 FOR I =5 TO 29:X=XH(I):Y=YH(I)
1170 L=50:C=0:GOSUB 1220:X=X-7:Y=Y+5:L=
     40:C=1:GOSUB 1220:XP=XH(I)*0.0627-3
     :IF I>13 THEN XP=XP-1
1180 YP=23-YH(I)*0.0585:IF I>14 AND I<20
     THEN YP=YP+1
```

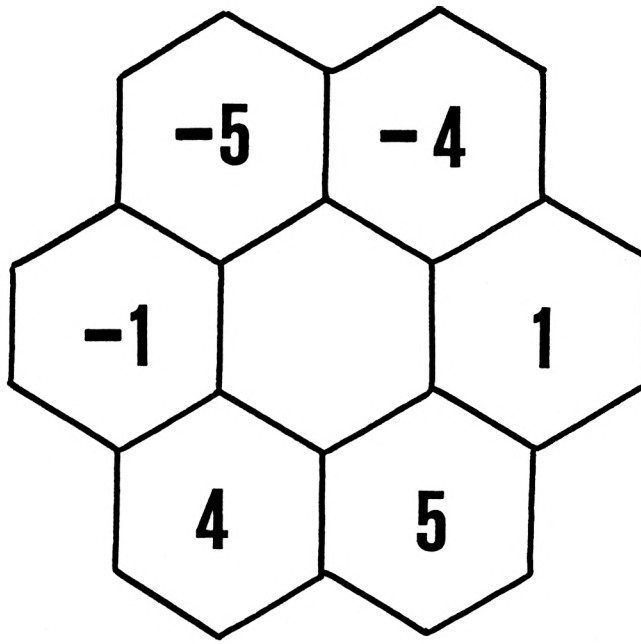
```
1190 LOCATE XP,YP:PRINT I-4:NEXT:PAPER 2
     :RETURN
1200 REM
1210 REM HEXAGON
1220 GRAPHICS PEN C
1230 R=L*0.5:S=L*0.866:MOVE X,Y
1240 DRAWR -S,-R:DRAWR -S,R:DRAWR 0,L
1250 DRAWR S,R:DRAWR S,-R:DRAWR 0,-L
1260 MOVE X-R,Y:FILL C:RETURN
1270 REM
1280 REM FIRST TURN
1290 LOCATE 2,22:PRINT"USER-U":LOCATE 2,
     23:PRINT"COMPUTER-C":K=0:WHILE K=0
1300 IF INKEY(42)=0 THEN T=0:K=1
1310 IF INKEY(62)=0 THEN T=1:K=1
1320 WEND:PAPER 2:LOCATE 2,22:PRINT SPC
     (6):PRINT:PRINT SPC(11):CLEAR INPUT
     :RETURN
1330 REM
1340 REM CHECK SIDE
1350 FOR I= 5 TO 25 STEP 5:FOR J=0 TO 4
     STEP 4
1360 IF H(I+J)=2 AND J=0 THEN LS=1
1370 IF H(I+J)=2 AND J=4 THEN RS=1
1380 NEXT:NEXT:IF LS=1 AND RS=1 THEN
     LOCATE 2,22:PRINT"COMPUTER":PRINT
     " WINS":LOCATE 1,1:STOP ELSE
     RETURN
1390 REM
1400 REM USER MOVE
1410 VH=0:WHILE VH=0:CH=0:WHILE CH=0
```

```
1420 LOCATE 2,22:PRINT"TYPE MOVE"
1430 INPUT NM:LOCATE 1,23:PRINT SPC(10)
1440 IF NM>0 AND NM<26 AND INT(NM)=NM
     THEN CH=1
1450 WEND:IF H(NM+4)=0 THEN VH=1
1460 WEND:NM=NM+4:LOCATE 2,22:PRINT
     SPC(9):FM=1:RETURN
1470 REM
1480 REM COMPUTER MOVE
1490 IF CM=1 THEN GOSUB 1530 ELSE GOSUB
     1570
1500 FM=2:RETURN
1510 REM
1520 REM CENTRE
1530 IF H(17)=0 THEN NM=17 ELSE NM=17+P(
     INT(RND(1)*6)+1)
1540 CM=0:LL=NM:LR=NM:RETURN
1550 REM
1560 REM NEXT MOVE
1570 N=N+1:MF=1-(N MOD 2)*2
1580 IF LS=1 THEN MF=1
1590 IF RS=1 THEN MF=-1
1600 IF MF=1 THEN A=1:B=6:LM=LR ELSE A=6
     :B=1:LM=LL
1610 FOR I=A TO B STEP MF:HX=LM+P(I)
1620 IF H(HX)=0 THEN NM=HX
1630 NEXT:IF MF=1 THEN LR=NM ELSE LL=NM
1640 RETURN
1650 REM
1660 REM SHOW MOVE
1670 H(NM)=FM:X=XH(NM)-7:Y=YH(NM)+5
```

```
1680 IF FM=1 THEN C=3 ELSE C=2
1690 GOSUB 1220:RETURN
1700 REM
1710 REM DATA FOR HEXAGONS & PRIORITIES
1720 DATA 120,320,200,320,280,320,360
1730 DATA 320,440,320,160,250,240,250
1740 DATA 320,250,400,250,480,250,200
1750 DATA 180,280,180,360,180,440,180
1760 DATA 520,180,240,110,320,110,400
1770 DATA 110,480,110,560,110,280,40
1780 DATA 360,40,440,40,520,40,600,40
1790 DATA -1,4,-5,5,-4,1
```



ARRAY LOCATIONS



P ARRAY

Line 1120 then initialises variables needed later in the program, CM, N, LS and RS.

Board routine (Lines 1140-1200) :

Line 1150 clears the entire display to cyan, using the BORDER command to extend the uniform coloured background up to the screen edge of the monitor. The program's title, HEX, is PRINTed in black text on a white background, displaced from the middle of the display to allow room for the first row of five hexagons.

The FOR loop, lines 1160-1190, then uses GOSUB 1220 and the coordinates in the XH, YH arrays, to draw all the twenty five hexagons forming the

board. At each particular location, 'hexagon' is used twice, with a colour change from black to white, to give a clearly defined edge to each hexagon. Lines 1170, 1180 calculate coordinates (XP, YP) at which to place the number of each hexagon using LOCATE.

Hexagon routine (Lines 1210-1270) :

Here a hexagon, sides length L and colour C, is drawn relative to the point (X, Y) using LINE and FILL. The cosine and sine of an angle of 60 degrees are incorporated into the variables R and S and so the values used in the DRAWR commannds at lines 1240, 1250 follow quite automatically.

First turn routine (Lines 1280-1330) :

This routine allows the user to decide whether the computer will be allowed to make the first move or not. (The playing algorithm does not include defensive moves. As a result, if the user starts HEX is likely to lose.) Line 1290 places a message on the display and then the WHILE/WEND loop returns a value of T = 1 for the computer to start, or T = 0 for the user's first move, to the control routine. This value is used at lines 1040, 1060, at each pass through the main control loop to decide whether it is the user's or computer's turn to make a move.

Check side routine (Lines 1340-1390) :

The computer needs to know when its chain of filled hexagons has reached either the left, or right, side of the board. The 'check side' routine determines this. The combination of the ranges of the two FOR loops between 1350-1380 means that at line 1360 each hexagon down the left side of the board is being checked and at line 1370 each hexagon down the right side. In either case, if a value of 2 is found in the H array the computer's chain has reached that side and the flag, LS or RS, is set accordingly.

Obviously, if line 1380 detects that both sides have been reached, the computer can state that it has won the game. HEX does not include a routine to identify a user win, however. The number of possible chains which could comprise a winning strategy is too great. It is only because of the simplified strategy it employs that the program can discover its own wins.

User move routine (Lines 1400-1470) :

Two nested WHILE/WEND loops between lines 1410 -1460 are employed to obtain the user's choice of move. The inner loop only terminates if an integer in the range 1-25 is typed and detected at 1440. Then line 1450 allows the outer loop to be left if the corresponding hexagon is vacant. Line 1460

increases the value of the variable NM to match the location in the H array. It also selects the value of FM for the routine 'show move'.

Computer move routine (Lines 1480-1510) :

If it is the first move made by the computer, indicated by the status of of centre-move flag, CM, line 1490 calls the routine 'centre' with GOSUB 1530 to place the first filled hexagon for the computer's chain as close to the centre of the board as possible. Subsequent positions are determined by calling 'next move' with GOSUB 1570. The value of FM for the routine 'show move' is chosen at 1500.

Centre routine (Lines 1520-1550) :

This routine attempts to put the first hexagon of the computer's chain at the central position, H = 17, in the array. This is part of the algorithm used by the program. If, however, line 1530 detects that the centre is not vacant, the P array is used to find an adjacent location at random. Line 1540 updates the CM flag and then initialises the variables LL, LR, which record the last hexagon added for the left side of the chain and the right side.

Next move routine (Lines 1560-1650) :

This routine adds a hexagon alternately to one end of the computer's chain and then the other. This switch is achieved by the variable MF. At line 1570, N is incremented. This means that the expression (N MOD 2) will alternate between 0 and 1. Hence 1 - (N MOD 2) * 2, the value calculated for MF, jumps from 1 to -1, or vice versa, every time the routine is called. If MF is 1 the right side of the chain is extended whereas the left side will be if MF is -1. Lines 1580, 1590 correct the value of MF if one of the sides has already been reached.

MF adds to the chain by controlling the values of A, B and LM at line 1600. The FOR loop, 1610 - 1630, then inevitably STEPs through the P array in a way which will extend the chain in the direction required. It selects the next position, HX, adjacent to the last, LM, at 1610 by incrementing with the chosen P value. This position is then checked at 1620 to make sure it is free. Finally line 1630 records the move made, as either LR or LL, as this will be needed the next time line 1600 is executed.

Show move routine (Lines 1660-1700) :

Here the move made by either user or computer is recorded. Line 1670 updates the H array and lines 1670 - 1690 fill in the corresponding hexagon on the board in the appropriate colour.

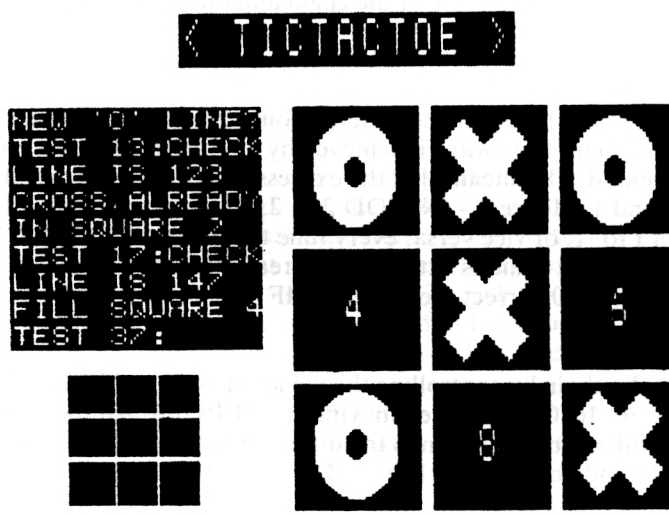
Data (Lines 1710-1790) :

Values are held for the x- and y- coordinates of the 25 hexagons making up the board. The values for the priority array, P, are stored at 1790.

TICTACTOE

This program plays the familiar game of noughts and crosses, the shorter American name being chosen as more convenient for a computer file name! TICTACTOE has the advantage of playing a game with a relatvely simple tree structure. It is able, therefore, to play at a far greater depth and is, in a sense, anticipating ply 8 by constantly referring to the possible winning lines in the algorithms it employs.

The program plays a defensive game, always giving priority to blocking its opponent whenever a dangerous situation is developing. When no such threat exists, it carefully decides the best line to continue by looking at all the moves it has currently made.



The screen dump shows that two noughts and crosses grids are displayed. The main grid shows the actual state of the game. The smaller one is used to indicate the various combinations of moves being examined. This is accompanied by explanatory messages in the text window on the left of the screen. Here TICTACTOE displays accountability for its decisions, an important feature of A.I. software.

```
1000 REM TICTACTOE - PAT HALL, 1/86
1010 REM CONTROL ROUTINE
1020 GOSUB 1070:GOSUB 1130:GOSUB 1240
1030 GOSUB 1290:IF WG=0 THEN GOSUB 1340
     ELSE GOSUB 1610
1040 STOP
1050 REM
1060 REM INITIALISATION
1070 BORDER 0:MODE 1:INK 0,26:INK 1,20:
     INK 2,2:INK 3,0:DIM PM(5):DIM CM(5)
     :DIM GM(5):DIM D(9):DIM WIN$(8):FOR
     I=1 TO 8:READ WIN$(I):NEXT:P=0:C=0
1080 DIM LX(9):DIM LY(9):FOR I=1 TO 9
1090 READ LX(I),LY(I):NEXT:DIM SX(9):DIM
     SY(9):FOR I=1 TO 9:READ SX(I),SY(I)
     :NEXT:GRAPHICS PEN 3:DIM FT(8)
1100 RETURN
1110 REM
1120 REM DISPLAY
1130 BORDER 2:PEN 3:PAPER 1:CLS:WINDOW
     #1,1,40,1,3:PAPER #1,0:CLS #1
1140 WINDOW #2,3,17,5,13:PEN #2,0:PAPER
     #2,2:WINDOW #3,6,14,15,21:PAPER #3,
     0:WINDOW #4,20,38,5,21:PAPER #4,0
1150 WINDOW #5,1,40,23,25:PEN #5,3:PAPER
     #5,0:CLS #5:LOCATE 14,2:PRINT
     "< TICTACTOE >":GOSUB 1190:CLS #2
1160 RETURN
1170 REM
1180 REM FIRST TURN
1190 LOCATE 11,12:PRINT"THE USER STARTS"
```

```
    ;:PRINT" - U":LOCATE 11,14:PRINT
    "COMPUTER STARTS - C":WG=2:WHILE WG
    =2:IF INKEY(62)=0 THEN WG=1
1200 IF INKEY(42)=0 THEN WG=0
1210 WEND:CLEAR INPUT:FOR I=12 TO 14:
    LOCATE 11,I:PRINT STRING$(19,CHR$(
    32)):NEXT:RETURN
1220 REM
1230 REM LARGE GRID
1240 CLS #4:FOR I=1 TO 10:MOVE 399+I,64:
    DRAW 399+I,334:MOVE 499+I,64:DRAW
    499+I,334:NEXT:FOR I=1 TO 10:MOVE
    304,149+I:DRAW 606,149+I:MOVE 304,
    239+I:DRAW 606,239+I:NEXT:PAPER 0
1250 FOR I=1 TO 9:X=LX(I)*0.062:Y=25-LY(
    I)/16:IF I>6 THEN Y=Y+1
1260 LOCATE X,Y:PRINT I:NEXT:RETURN
1270 REM
1280 REM SMALL GRID
1290 CLS #3:FOR I=1 TO 5:MOVE 125+I,64
1300 DRAW 125+I,174:MOVE 172+I,64:DRAW
    172+I,174:NEXT:FOR I=1 TO 5:MOVE 80
    ,98+I:DRAW 222,98+I:MOVE 80,135+I
1310 DRAW 222,135+I:NEXT:RETURN
1320 REM
1330 REM USER MOVE
1340 UC=0:WHILE UC=0:N=0:WHILE N=0
1350 LOCATE #5,13,2:PRINT #5,
    "TYPE IN YOUR MOVE"
1360 IF INKEY(64)=0 THEN N=1
1370 IF INKEY(65)=0 THEN N=2
```

```
1380 IF INKEY(57)=0 THEN N=3
1390 IF INKEY(56)=0 THEN N=4
1400 IF INKEY(49)=0 THEN N=5
1410 IF INKEY(48)=0 THEN N=6
1420 IF INKEY(41)=0 THEN N=7
1430 IF INKEY(40)=0 THEN N=8
1440 IF INKEY(33)=0 THEN N=9
1450 WEND:CLEAR INPUT:CLS #5
1460 IF D(N)=0 THEN UC=1
1470 WEND:P=P+1:PM(P)=N:D(N)=1:GOSUB
     1520:IF P>2 THEN GOSUB 2040:GOSUB
     2120
1480 IF C+P=9 THEN GOSUB 2350
1490 GOSUB 1610:RETURN
1500 REM
1510 REM NOUGHT
1520 SOUND 1,60,50:GRAPHICS PEN 1:R=35
1530 GOSUB 1570:GRAPHICS PEN 0:R=10
1540 GOSUB 1570:RETURN
1550 REM
1560 REM CIRCLE
1570 FOR YC=LY(N)-R TO LY(N)+R:X1=LX(N)-
     SQR(R^2-(YC-LY(N))^2):X2=LX(N)*2-X1
     :MOVE X1,YC:DRAW X2,YC:NEXT YC
1580 RETURN
1590 REM
1600 REM COMPUTER MOVE
1610 IF P>1 THEN GOSUB 1680 ELSE SC=0
1620 IF SC<>0 THEN N=SC ELSE GOSUB 1880
1630 C=C+1:CM(C)=N:D(N)=2:GOSUB 1990:IF
     C>2 THEN GOSUB 2080:GOSUB 2120
```

```
1640 IF C+P=9 THEN GOSUB 2350
1650 GOSUB 1340:RETURN
1660 REM
1670 REM BLOCK LINE
1680 SC=0:PRINT #2:PEN #2,0:PAPER #2,3
1690 PRINT #2," NEW 'O' LINE?":GOSUB
     2040:GOSUB 2200:FOR I=1 TO G-1:FOR
     J=I+1 TO G:PRINT #2," TEST ";GM(I);
     GM(J);CHR$(58);:CL=3:GOSUB 1790
1700 BL=0:FOR K=1 TO 8:BF=0:FOR L=1 TO 3
     :A=VAL(MID$(WIN$(K),L,1)):IF A=GM(I
     ) OR A=GM(J) THEN BF=BF+1
1710 NEXT:IF BF=2 THEN BL=K
1720 NEXT:GOSUB 1760:IF BL=0 THEN PRINT
     #2," OKAY" ELSE PRINT #2," CHECK":
     GOSUB 1820
1730 GOSUB 1760:CL=0:GOSUB 1790:NEXT:
     NEXT:RETURN
1740 REM
1750 REM PAUSE
1760 FOR TP=1 TO 1000:NEXT:RETURN
1770 REM
1780 REM SQUARES 2
1790 PEN CL:LOCATE SX(GM(I)),SY(GM(I)):
     PRINT CHR$(79):LOCATE SX(GM(J)),SY(
     GM(J)):PRINT CHR$(79):RETURN
1800 REM
1810 REM MAKE SURE
1820 GOSUB 1760:PRINT #2," LINE IS ";
     WIN$(BL):SF=0:FOR M=1 TO 3:A=VAL(
     MID$(WIN$(BL),M,1)):IF D(A)=0 THEN
```

```
     SC=A
1830 IF D(A)=2 THEN SF=A
1840 NEXT:GOSUB 1760:IF SF<>0 THEN PRINT
     #2," CROSS ALREADY":PRINT #2,
     " IN SQUARE";SF ELSE IF SC<> 0 THEN
     PRINT #2," FILL SQUARE";SC
1850 RETURN
1860 REM
1870 REM STRATEGY
1880 PEN #2,0:PAPER #2,2:PRINT #2
     ," CHOOSING LINE":PRINT #2,
     " FOR COMPUTER":FOR I=1 TO 8:FT(I)=
     0:NEXT:FOR I=1 TO 8:FC=0:FOR J=1 TO
     3: S=D(VAL(MID$(WIN$(I),J,1)))
1890 IF S=1 THEN FC=1:FT(I)=0 ELSE IF S=
     2 AND FC=0 THEN FT(I)=FT(I)+1
1900 NEXT:NEXT:FOR I=1 TO 8
1910 PRINT #2,"'";WIN$(I);"' VALUE";
1920 PRINT #2,FT(I):GOSUB 1750:NEXT:FB=0
     :FOR I=1 TO 8:IF FT(I)>=FB THEN FB=
     FT(I):FL=I
1930 NEXT:ST$=WIN$(FL):IF FB=0 AND C<>0
     THEN GOSUB 2390
1940 IF FB=0 THEN ST$=WIN$(INT(RND*8)
     +1)
1950 PRINT #2,"SELECTED: ";ST$:FOR I=1
     TO 3:A=VAL(MID$(ST$,I,1)):IF D(A)=
     0 THEN N=A
1960 NEXT:RETURN
1970 REM
1980 REM CROSS
```

```
1990 SOUND 1,239,50:GRAPHICS PEN 2:MOVE
     LX(N)-30,LY(N)+15:DRAWR 15,15:DRAWR
     45,-45:DRAWR -15,-15:DRAWR -45,45:
     MOVE LX(N),LY(N):FILL 2:DRAWR -30,
     15:DRAWR 15,15:DRAWR 45,-45:DRAWR -
     15,-15:DRAWR -45,45
2000 MOVE LX(N)+4,LY(N):FILL 2:MOVE LX(N
     )-30,LY(N)-15:DRAWR 45,45:DRAWR 15,
     -15:DRAWR -45,-45:DRAWR -15,15:
     MOVER 4,0:FILL 2:MOVER 30,30:
     FILL 2
2010 RETURN
2020 REM
2030 REM USER ARRAY
2040 G=P:FOR I=1 TO 5:GM(I)=PM(I):NEXT
2050 CH=79:RETURN
2060 REM
2070 REM COMPUTER ARRAY
2080 G=C:FOR I=1 TO 5:GM(I)=CM(I):NEXT
2090 CH=88:RETURN
2100 REM
2110 REM CHECK LINE
2120 PEN #2,3:PAPER #2,0:PRINT #2,
     " CHECK '";CHR$(CH);"' WIN":GOSUB
     2200:W=0:FOR I=1 TO G-2:FOR J=I+1
     TO G-1:FOR K=J+1 TO G:LIN$=STR$(GM(
     I))+STR$(GM(J))+STR$(GM(K)):PRINT
     #2," TEST ";LIN$;": ";:CL=3
2130 A$="":FOR M=2 TO 6 STEP 2:A$=A$+
     MID$(LIN$,M,1):NEXT:LIN$=A$
2140 GOSUB 2260:GOSUB 1760:CL=0:GOSUB
```

```
      2260:WP=0:FOR L=1 TO 8:IF LIN$=
      WIN$(L) THEN W=1:WL=L:WP=1
2150 NEXT:IF WP=1 THEN PRINT #2," YES"
      ELSE PRINT #2," NO"
2160 NEXT:NEXT:NEXT:IF W=1 THEN GOSUB
      2300
2170 RETURN
2180 REM
2190 REM SORT
2200 SR=0:WHILE SR=0:F=0:FOR I=1 TO 8-1
2210 IF GM(I)>GM(I+1) THEN S=GM(I):GM(I)
      =GM(I+1):GM(I+1)=S:F=1
2220 NEXT:IF F=0 THEN SR=1
2230 WEND:RETURN
2240 REM
2250 REM SQUARES 3
2260 PEN CL:LOCATE SX(GM(I)),SY(GM(I)):
      PRINT CHR$(CH):LOCATE SX(GM(J)),SY(
      GM(J)):PRINT CHR$(CH):LOCATE SX(GM(
      K)),SY(GM(K)):PRINT CHR$(CH)
2270 RETURN
2280 REM
2290 REM REM WIN
2300 SL=VAL(LEFT$(WIN$(WL),1)):EL=VAL(
      RIGHT$(WIN$(WL),1)):GRAPHICS PEN 3:
      MOVE LX(SL),LY(SL):DRAW LX(EL),LY(
      EL)
2310 LOCATE #5,14,2:IF CH=79 THEN PRINT
      #5,"THE USER WINS" ELSE PRINT #5,
      "COMPUTER WINS"
2320 LOCATE 1,1:PEN 3:STOP:RETURN
```

```
2330 REM
2340 REM DRAW
2350 LOCATE #5,14,2:PRINT #5,
     "GAME IS DRAWN":LOCATE 1,1:STOP
2360 RETURN
2370 REM
2380 REM CONCEDE
2390 LOCATE #5,14,2:PRINT #5,
     "ACCEPT DEFEAT":LOCATE 1,1:STOP
2400 RETURN
2410 REM
2420 REM DATA
2430 DATA 123,456,789,147,258,369,159
2440 DATA 357,354,289,454,289,554,289
2450 DATA 354,199,454,199,554,199,354
2460 DATA 109,454,109,554,109,7,16,10
2470 DATA 16,13,16,7,18,10,18,13,18,7
2480 DATA 20,10,20,13,20
```

Commentary on TICTACTOE

The control routine between lines 1020, 1030 calls the routine 'initialisation', to establish arrays, variables and information required during the program's execution, and then calls 'display', 'large grid' and 'small grid' to set up the screen display used throughout the program. At line 1030 the value of the flag WG, obtained in the routine 'first turn', called by 'display', is used to call either 'user move' or 'computer move', depending upon whether an initial move by the user or the computer has been selected.

Initialisation routine (Lines 1060-1110)

Line 1070 selects MODE and screen colours and then sets up the arrays PM, and CM. These are used to record the user's moves and the computer's moves on a numbered 1-9 grid representing the nought and crosses game. The array GM, also initialised at line 1070, is required later in the program in several routines which need to refer to either set of all moves currently made, whether by the user or the computer. The D array is used to record moves made, both for the screen display and also for reference purposes in various routines.

The program needs to know which combinations of filled squares represent winning lines in a game of noughts and crosses. There are eight such lines and this information is held as DATA at 2430, 2440. The winning lines are READ into the WIN$ array by the FOR loop at line 1070. The same line initialises the variable P which counts the total number of user moves at any stage of the game. Similarly the variable C records the number of computer moves.

The screen display gives two representations of the noughts and crosses grid. The main grid fills the right hand side of the display and shows the current state of the game, constantly updated as the user and computer make their moves. A smaller grid at the bottom left of the screen is used to indicate which combination of positions the computer is considering when selecting its next move. The coordinates of the centres of all the squares on both grids are READ from DATA in 'initialisation'. Lines 1080, 1090 place coordinates for the large grid into the arrays LX and LY. Similarly line 1090 fills the SX, SY arrays with coordinates for the small grid. Line 1090 also selects GRAPHICS PEN colour and initialises the FT array for the 'strategy' routine.

Display routine (Lines 1120-1170)

This routine sets up the screen display used throughout the execution of the program. Line 1130 clears the entire screen to cyan and selects black for text. Five windows are established by lines 1130-1150. Windows #1 and #5 are used initially to place matching bands of white across the top and bottom of the screen and line 1150 PRINTs the program's title on the top band. Before the display is completed a prompt appears on the screen which asks whether the user or the computer is going to make the first move in the game. This is done by the call to the routine 'first turn' at line 1150 with GOSUB 1190.

First turn routine (Lines 1180-1220)

Here the value of the flag WG is determined, as required by the control routine in order to decide whether the user or the computer makes the first

move. Prompts are PRINTed by line 1190. The WHILE/WEND loop, lines 1190-1210, then prevents further execution of the program until either the U or the C key is pressed. INKEY is used by lines 1190, 1200 to set WG to 0 or 1 as required and either value causes the loop to end at line 1210. After this the prompts are erased by the FOR loop at 1210.

Large grid routine (Lines 1230-1270)

A large noughts and crosses grid is placed upon the screen. First a white background is created by the use of CLS #4 at line 1240. Then vertical and horizontal lines are added boldly in black by using the two FOR loops at line 1240 to thicken them. The grid now needs to be numbered to allow the user to choose the moves to be made. This is done by the FOR loop, lines 1250, 1260. The arrays LX, LY hold the graphic coordinates of the centres of the squares and so these have to be modified by the algebraic expressions in line 1250 to give the screen coordinates required by the LOCATE command.

Small grid routine (Lines 1280-1320)

A small noughts and crosses grid is placed in the bottom left hand corner of the screen by this routine. Here the white background is produced by CLS #3 at line 1290. The two FOR loops again divide the area into the grid required.

User move routine (Lines 1330-1500)

This routine requests the user's choice of move. This is done between lines 1340 - 1470 by the nested WHILE/WEND loops. The inner loop repeats until one of the conditional lines, 1360 - 1440, detects that an appropriate key has been pressed. Then the outer loop repeats unless this choice of move corresponds to an empty square on the grid.

Line 1470 increments the variable P representing the number of user moves and enters the move into the user move array, PM. The D array is also updated and the routine 'nought' called with GOSUB 1520 to show the move made on the large grid.

Line 1470 then calls 'user array' and 'check line' to see if the user has won the game. Obviously this can only be possible if at least three moves have been made and so 1470 includes the conditional expression, IF P > 2 THEN... At line 1480 the total number of moves, both by computer and player, is calculated and, if this has reached nine without a win for either, the routine 'draw' is called.

The routine then calls 'computer move'.

Nought routine (Lines 1510-1550)

Here GOSUB 1570 is used twice to call the circle routine and mark the grid with the user's nought.

Circle routine (Lines 1560-1590)

This routine fills a solid circle, radius R, about the point LX(N), LY(N). The YC loop does this by calculating two points, X1, X2, on the circumference of the circle for each YC value and then joining them. The equations at line 1570 are derived from the standard Pythagorean relationship.

Computer move routine (Lines 1600-1660)

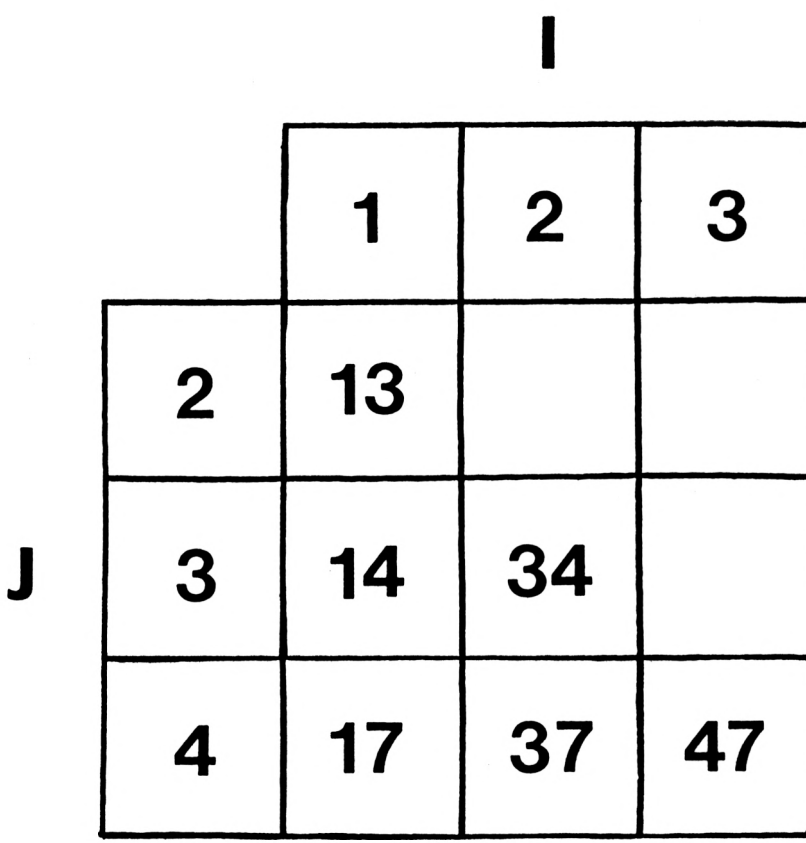
Here TICTACTOE decides the move that the computer should make. Priority is given to defensive play and so, if the user has made at least two moves, line 1610 calls 'block line' with GOSUB 1680 to see whether the move made by the computer needs to obstruct a possible winning line. If there are fewer than two user moves, the variable SC is set to zero. The routine 'block line' itself can also return a value of SC equal to zero if there is no user line to be obstructed. In either case, line 1620 calls 'strategy' to choose the best new move for the computer to make. However, if 'block line' has found a square which needs to be occupied quickly to prevent a user win, line 1620 plays this move.

The remaining lines of the routine, 1630-1650, are an exact parallel to lines 1470-1490 of 'user move'.

Block line routine (Lines 1670-1740)

All the moves made so far by the user are examined here to see if any two of them form a potential line which needs to be blocked by the computer. For exanple, since the combination 159 is a winning line, if the user has already filled squares 1 and 9 this routine will return the value SC = 5 in order to prevent the user from completing this particular route.

First, the routines 'user array' and 'sort' have to be called to place all the user moves into numerical order. The four nested loops, 1690-1730, then combine all the user moves into pairs which possibly form a potential line so they can check these against the known winning lines in the WIN$ array. The range of the I and J loops ensure that all the correct combinations are formed. This can be seen by looking at the diagram, where it has been assumed that the user has made the four moves: 1, 3, 4 and 7. Each combination is shown as it is formed by lines 1690 and 1730.



The K loop considers the eight WIN$ lines and the L loop tests the three squares of each of these lines against the currently combined pair of user moves, at line 1700. For each match, the variable BF is incremented. If its value reaches 2 then a potential user line has been found. When this happens the block-line flag, BL, is equated to the value of K at line 1710 to register the particular winning line involved.

Line 1720 indicates to the user whether or not a line has been found. If one has, the routine 'make sure' is called by GOSUB 1820 to see if it is blocked by a square already occupied by the computer or whether a defensive move has to be made.

Pause routine (Lines 1750-1770)

A delay is created by the empty FOR loop.

Squares 2 routine (Lines 1780-1800)

The positions of the pairs of user moves being considered by 'block line' are shown on the small grid by this routine.

Make sure routine (Lines 1810-1860)

This routine is called by 'block line' to see if a suspected winning line for the user, WIN$(BL) , is already blocked by a square filled by the computer. Line 1820 states which line is being examined.

The FOR loop checks through the line. If 1820 finds a vacant location in the D array, then obviously this is the move the computer needs to make and so the variable SC is set accordingly. Alternatively, if the line has already been blocked, line 1830 will identify as SF the computer move which is preventing a user win. Whichever is the case, line 1840 will inform the user what is happening.

Strategy routine (Lines 1870-1970)

If TICTACTOE does not have to block a winning line which is being produced by the user, line 1620 calls this routine to choose the best possible move for the computer. A prompt PRINTed by line 1880 indicates this.

The move is chosen by establishing a score for each possible line in WIN$. These values are placed in the FT array, initialised at 1880, and calculated by the nested FOR loops, 1880 - 1900. The outer loop examines each line in turn. If 1890 finds a user move in a line it is immediately given a score of zero and the flag FC prevents any subsequent addition to this score. However, if a computer move is found, and the FC flag is switched off, the score is incremented. In this way, a line with no user moves and the most computer moves will be given the highest score.

The scores for all eight possible lines are displayed by 1900 - 1920. Lines 1920, 1930 then identify the best line for the computer to play, ST$. If no best line is found, 1940 chooses a line at random for the very first computer move. Later in the game, on the other hand, an absence of a best line causes TICTACTOE to accept defeat at 1930 with 'concede'.

The FOR loop, 1950, 1960, chooses the move, N, to make from the best line.

Cross routine (Lines 1980-2020)

A cross is drawn for the computer's move at (LX(N), LY(N)) on the large grid by the use of FILL and DRAWR.

User array routine (Lines 2030-2060)

This places the user's moves into the GM array for those generalised routines which need to operate on either the user or computer moves. It also places the ASCII code for an 'O' into the variable CH at line 2050.

Computer array routine (Lines 2070-2100)

This routine is directly analogous to 'user array'.

Check line routine (Lines 2110-2180)

A possible winning line for either player is determined in this routine. Line 2120 indicates whether a user or computer win is being checked and 'sort', called by GOSUB 2200, places all moves in the GM array into order. Then the nested I, J, K loops, 2120 - 2160, concatenate values from GM to form all possible lines, expressed at each pass through the loops as the string, LIN$. The ranges of I, J, K are deduced in a similar way to the I, J ranges in 'block line'.

Line 2140 shows on the small grid the particular combination of moves that is being tested by calling 'squares 3'. The FOR loop, 2140, 2150, compares LIN$ with the WIN$ array to see if a winning line has been found and a message is placed on the screen by 2150. After the loops have checked through all the possible lines, line 2160 calls 'win' if necessary.

Sort routine (Lines 2190-2240)

This is a simple sorting routine which will place array values into order when required by other routines.

Squares 3 routine (Lines 2250-2280)

The combination of moves being tested by 'check line' is shown on the small grid by this routine.

Win routine (Lines 2290-2330)

A line is drawn across the large grid to show a winning combination of squares. At line 2300, SL and EL are defined as the first and last squares of the winning line, obtained by using LEFT$ and RIGHT$ on WIN$(WL). These values are then used in the LX and LY coordinate arrays to show the line graphically, with MOVE and DRAW. Line 2310 then states whether the user or computer has won.

Draw routine (Lines 2340-2370)

Here the program informs the user that the game cannot be won by either player.

Concede routine (Lines 2380-2410)

The program admits defeat!

Data (Lines 2420-2480):

The codes for the winning lines and the coordinates for both grids are stored here and READ by the routine 'initialisation'.

CHAPTER 7

The wood for the trees: computer vision.

If one of the aims of Artificial Intelligence is to be the eventual development of useful robots, then the problem of adequate vision will have to be solved. We are, like most other species, almost wholly dependent upon sight for the conception we have of the world we inhabit. A culture without vision would be strangely different to ours. In a short story, H.G.Wells imagined a valley isolated from the rest of the world and in which every person was blind. Inevitably, a very alien society had developed with concepts totally different from those of an accidental, sighted visitor. Robots, too, would never share our world, in the way postulated by visionaries and science fiction authors, without a sense closely approximating to human sight.

Industrial Applications

More appropriate to the present is the need to give industrial robots an ability to see. This, although very crude, would enable automated factories to produce a greater variety of products and deal more efficiently with problems of orientation of parts on conveyor belts. Here the force of industrial competition will presumably provide the funds required for the pure research required, in the same way that money was made available for A.I. once the commercial viability of expert systems was perceived.

A third incentive for investigation of computer vision is the immediate affect such a development would have upon other areas of A.I. In particular, learning systems would become far more efficient if computers were able to examine directly those things they were meant to be learning about instead of relying upon a human at the keyboard as an unavoidable intermediary.

Human Vision

Nevertheless, the difficulties involved in producing the software, and hardware, necessary to simulate vision can hardly be exaggerated. Quite simply, a tremendous amount of computational power is involved in human

sight. It has been estimated that perhaps 60 percent of the cortex is involved is seeing. The neurons in the brain are vastly more sophisticated than computer circuits. They possess thousands of interconnections in three dimensions, compared with the few, and relatively two-dimensional, connections of the silicon chip. In addition, brain signals are not merely electrical, but instead involve a variety of subtle chemical modifications. Most important of all, the brain is capable of parallel processing on a gigantic scale, unlike the fast, but serial, operation of digital computers.

These differences in basic design and capacity reveal themselves very obviously in the way in which a person and a computer see and interpret a given visual image. While the computer is still probably saying to itself something like: 'Ah, this pixel is a bit darker than that one, and so was the one before. Maybe there's an edge here. I'd better remember that in case this could be the corner of a rectangular object...' the person will be thinking: 'Oh dear. I've left the freezer open again. Hope the ice-cream hasn't thawed out, because the last time I did that on Howard's birthday he sulked for a week.'

It is the human ability to make such amazing leaps of association that provides a tremendous challenge to A.I. It seems highly unlikely that anything less than a completely new type of computer architecture, and vastly improved chips, will ever bring computers and humans together to the position where they could agree about the fine detail of what they were seeing. Yet again, A.I. has shown the great deal of intelligence required to do some seemingly ordinary process. Nevertheless, the challenge remains to investigate what can be done with existing machines.

Processing the Data

The first stage of any system of vision has to be the collection of raw data for further processing. It can be assumed that a video camera is available for this and that it will reduce a scene to an array of pixels, each varying over some range of greyness and representing the intensity of light falling on that particular area of the receptor's surface. The software must now convert this into a meaningful internal representation of the scene viewed and begin to identify what has been 'seen'.

The algorithms employed must consider the levels of greyness and decide at what point a change in intensity represents some real change in the scene, rather than just a random fluctuation caused by a lighting effect. They must be able to decide where edges are and how these fit together to form surfaces. As seen with expert systems, logical rules are not sufficient for true understanding. A vision program will need to have a great deal of the same

heuristic, 'rules of thumb' built in. Then it will realise that a particular part of the image cannot possibly represent, perhaps, a bus being cut in half because such things are unlikely to happen in the real world. Instead it will start looking for a lamppost in the foreground. (A human, of course, would immediately recognise half a bus without any difficulty!)

Stereoscopic Vision

Important cues in interpreting visual information in the cortex seem to result from stereoscopic vision. Bela Julesz of the AT&T Bell Laboratories has discovered that the brain has unexpected abilities at extracting depth information from the signals it receives. Even a randomly produced pattern of black and white dots, presented to each eye but with a certain area of the pattern shifted in opposite directions for either one, will produce a strikingly three dimensional screen. In it, the area appears raised or depressed from the rest of the background. Here, the brain is clearly detecting the stereoscopic information without any obvious aids, like recognition of previously encountered patterns. Encouraged by this evidence of a working algorithm, Tomaso Poggio, at the Massachusetts Institute of Technology, has gone on to develop a stereo vision program. This can take two separate images, seen from slightly separated points of vision as with the two eyes of human sight, and by filtering the images mathematically and matching one against the other, determines differences in depth. This is clearly an significant line of research.

Thinking Small

In developing a vision system on a small microcomputer a number of enormous simplifications have to be made. Analysing an image rapidly consumes memory and it is therefore necessary to work on a very reduced scale. Processing of raw data into a convenient array of pixels is probably also too difficult, although cameras designed to be interfaced with a small computer are available commercially. Instead it is much easier to type the array information directly into the program. A choice has to be made about the type of algorithm to be employed. Recognising straight lines, as regular patterns of ones in an array otherwise consisting of noughts, is probably about the easiest approach, as well as having the satisfaction of mimicking the function of some neurons.

PIPPA

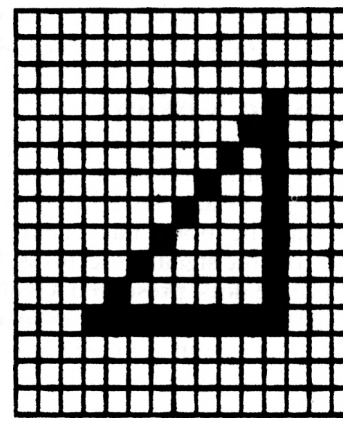
In this program, the algorithm developed looks instead for the vertices of a shape rather than lines. From the patterns the vertices make, it attempts to identify the shape from a knowledge gained in previous program RUNs.

< PIPPA >

TO MAKE SHAPE:
CURSOR-LOCATES
SPACEBAR-FIXES

IDENTIFICATION
PRESS: < S >

SEARCHING GRID
FOR VERTICES..
FOUND VERTEX 1
FOUND VERTEX 2
FOUND VERTEX 3



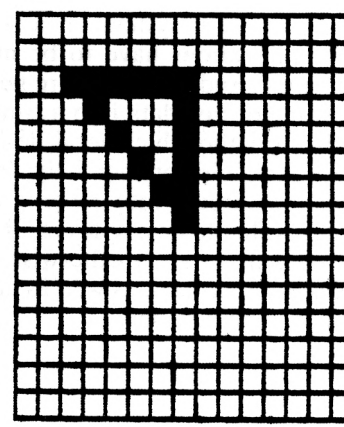
< PIPPA >

MATCHING SHAPE
FOR CODE: 22111
11

SHAPE IS:
TRIANGLE

SHAPE'S CODE:
221111

TRIANGLE CODE:
221111



The operation of the program can be understood by looking at the two screen dumps. In the first, a large right-angled triangle has been drawn on a coordinate grid on the screen, by using a combination of the cursor keys and the space-bar. Pressing the S key has then begun the program searching for vertices. From the text window it can be seen that all three were found successfully.

Once the vertices were found, the program generated a unique code based upon their mutual distances apart. The code determined here was 221111, as seen in the second screen dump. Once the computer had a code for the shape, it checked it against all known codes. Finding no match, it requested the identity of the shape and was given the name 'triangle'.

The program was then given the second triangle. It can be seen that although the triangle, this time, was smaller, in a different part of the grid and possessed a different orientation, the program still determined the same code for it and succeeded as a result in naming it correctly. PIPPA is able to learn to identify a number of different shapes in this way.

Commentary on PIPPA

The overall operation of the program is controlled by line 1020. The routine 'initialisation' is called with GOSUB 1050 to set up arrays and variables and 'describe', GOSUB 1130, provides an explanation of the program. Then the WHILE /WEND loop calls in turn 'polygon', which lets the user give the program a shape, 'code shape', which turns the shape into a unique code and 'identify' which compares this code with known shapes and their codes. The loop progressively acquires a set of codes which enables the program to identify more and more shapes given to it.

Initialisation routine (Lines 1040-1110) :

First, line 1050 selects MODE and screen colours. Lines 1050, 1060 then set up the arrays XA, YA, XB and YB. The first pair, XA and YA, are used to hold the DATA entered by the FOR loop at line 1090. These values are the relative coordinates of the immediate neighbours of any square on the grid on which the shape is drawn. They are needed, in the routine 'detect vertex', to decide whether a particular square is a vertex of the shape. Similarly line 1100 READs the coordinates of the second-nearest squares into XB, YB and these, too, are considered by 'detect vertex'.

Lines 1070, 1080 establish other arrays needed by the program and the variable NS, which is the total number of known shapes. The names of the shapes are held in the NS$ array and the identification codes by the CD$ array.

```
1000 REM PIPPA - PAT HALL, 1/86
1010 REM CONTROL ROUTINE
1020 GOSUB 1050:GOSUB 1130:WHILE TIME >0
     :GOSUB 1410:GOSUB 1740:GOSUB 1990:
     GOSUB 1370:WEND
1030 REM
1040 REM INITIALISATION
1050 MODE 1:BORDER 2:INK 0,2:INK 1,24:
     INK 2,0:INK 3,26:DIM XA(8)
1060 DIM YA(8): DIM XB(16): DIM YB(16)
1070 DIM G(18,18): DIM VX(10)
1080 DIM VY(10): DIM VV(90): DIM NS$(10)
     :DIM CD$(10): NS=0
1090 FOR I=1 TO 8:READ XA(I),YA(I):NEXT
1100 FOR I=1 TO 16:READ XB(I),YB(I):NEXT
     :RETURN
1110 REM
1120 REM DESCRIBE
1130 PEN 2:PAPER 1:CLS:WINDOW #1,1,40,1,
     3:PAPER #1,3:CLS#1:WINDOW #2,1,40,
     4,22:PAPER #2,1:WINDOW #3,1,40,23,
     25:PEN #3,2:PAPER #3,3:CLS #3
1140 PEN #1,3:PAPER #1,0:LOCATE #1,16,2:
     PRINT #1,"< PIPPA >":LOCATE 3,7
1150 PRINT"THIS PROGRAM CAN BE";
1160 PRINT" 'TAUGHT' HOW TO"
1170 PRINT"  IDENTIFY DIFFERENT";
1180 PRINT" SHAPES WHICH YOU"
1190 PRINT"  DRAW ON A 15 X 15 GRID"
1200 PRINT:PRINT"  IT DOES THIS";
1210 PRINT" BY FINDING THE VERTICES"
```

```
1220 PRINT"  OF THE SHAPE AND";
1230 PRINT" CALCULATING A CODE"
1240 PRINT"  FROM THEIR MUTUAL";
1250 PRINT" DISTANCES APART.":PRINT
1260 PRINT"  AT FIRST YOU NEED";
1270 PRINT" TO NAME EACH SHAPE"
1280 PRINT"  AS YOU ENTER IT.";
1290 PRINT" HOWEVER THE PROGRAM"
1300 PRINT"  ALWAYS CHECKS A";
1310 PRINT" NEW SHAPE WITH THE"
1320 PRINT"  CODES IT 'KNOWS'."
1330 GOSUB 1370:CLS #2:WINDOW #4,3,16,5,
     12:PEN #4,2:PAPER #4,3:CLS #4
1340 WINDOW #5,3,16,14,21:PEN #5,2:PAPER
     #5,3:CLS #5:PAPER 3:RETURN
1350 REM
1360 REM PROMPT
1370 CLS #3:PRINT #3:PRINT #3,TAB(16)
     "< SPACE >":K=0:WHILE K=0:IF INKEY
     (47)=0 THEN K=1
1380 WEND:CLS #3:CLEAR INPUT:RETURN
1390 REM
1400 REM POLYGON
1410 FOR I=2 TO 16:FOR J=2 TO 16:G(I,J)
     =0:NEXT:NEXT:FOR I=1 TO 10:VX(I)=0:
     VY(I)=0:NEXT
1420 VC=0:X=9:Y=9:P=150:CLS #4:CLS #5
1430 LOCATE #4,1,1:PRINT #4,"TO MAKE ";:
     PRINT #4,"SHAPE:":LOCATE #4,1,2
1440 PRINT #4,"CURSOR-LOCATES":LOCATE #4
     ,1,3:PRINT #4,"SPACEBAR-FIXES"
```

```
1450 LOCATE #4,1,5:PRINT #4,"IDENTIFIC";
     :PRINT #4,"ATION":LOCATE #4,1,6
1460 PRINT #4,"PRESS....KEY S":PRINT #4
1470 WINDOW #6,19,38,5,21:PAPER #6,3
1480 CLS #6:GRAPHICS PEN 2:FOR I=291 TO
     606 STEP 21:MOVE I,64:DRAW I,335
1490 NEXT:FOR I=64 TO 336 STEP 18
1500 MOVE 290,I:DRAW 604,I:NEXT
1510 B=0:WHILE B=0
1520 IF INKEY(8)=0 THEN X=X-1
1530 IF INKEY(1)=0 THEN X=X+1
1540 IF INKEY(2)=0 THEN Y=Y-1
1550 IF INKEY(0)=0 THEN Y=Y+1
1560 IF X<2 THEN X=2
1570 IF X>16 THEN X=16
1580 IF Y<2 THEN Y=2
1590 IF Y>16 THEN Y=16
1600 IF G(X,Y)=0 THEN GOSUB 1660
1610 IF INKEY(47)=0 AND G(X,Y)=0 THEN G(
     X,Y)=1:C=1:GOSUB 1710
1620 IF INKEY(60)=0 THEN B=1
1630 WEND:CLEAR INPUT:RETURN
1640 REM
1650 REM SHADE IN
1660 C=0:GOSUB 1710:FOR I=1 TO P:NEXT
1670 C=3:GOSUB 1710:FOR I=1 TO P:NEXT
1680 RETURN
1690 REM
1700 REM SQUARE
1710 GRAPHICS PEN C:XP=X*21+260:YP=Y*18+
     37:MOVE XP-9,YP-8:DRAWR 0,16:DRAWR
```

```
    17,0:DRAWR 0,-15:DRAWR -17,0:MOVE
    XP,YP:FILL C:RETURN
1720 REM
1730 REM CODE SHAPE
1740 PRINT #5,
    "SEARCHING GRIDFOR VERTICES"
1750 FOR I=1 TO 90:VV(I)=0:NEXT
1760 GOSUB 1850:S=0:MN=400:FOR I=1 TO VC
    :FOR J=1 TO VC
1770 IF I<>J THEN GOSUB 1950
1780 NEXT:NEXT:FOR I=1 TO S:VV(I)=INT(VV
    (I)/MN):NEXT:ST=0:WHILE ST=0:SW=0
1790 FOR I=1 TO S-1:IF VV(I)<VV(I+1)
    THEN A=VV(I):VV(I)=VV(I+1):VV(I+1)=
    A:SW=1
1800 NEXT:IF SW=0 THEN ST=1
1810 WEND:ID$="":FOR I=1 TO S:ID$=ID$+
    STR$(VV(I)):NEXT:PRINT #5,
    "SHAPE'S CODE:":PRINT #5,ID$
1820 RETURN
1830 REM
1840 REM SEARCH GRID
1850 P=0:FOR Y=16 TO 2 STEP-1:FOR X=2 TO
    16:IF G(X,Y)=0 THEN GOSUB 1660 ELSE
    GOSUB 1890
1860 NEXT:NEXT:RETURN
1870 REM
1880 REM DETECT VERTEX
1890 FOR I=1 TO 4:IF G(X+XA(I),Y+YA(I))
    <>0 AND G(X+XA(I+4),Y+YA(I+4))<>0
    THEN G(X,Y)=2
```

```
1900 NEXT:FOR I=1 TO 8:IF G(X+XB(I),Y+YB
     (I))<>0 AND G(X+XB(I+8),Y+YB(I+8))
     <>0 THEN G(X,Y)=2
1910 NEXT:IF G(X,Y)<>2 THEN C=0:GOSUB
     1710:VC=VC+1:VX(VC)=X:VY(VC)=Y:
     PRINT #5,"VERTEX";VC
1920 RETURN
1930 REM
1940 REM LENGTH
1950 S=S+1:VV(S)=(VX(I)-VX(J))^2+(VY(I)-
     VY(J))^2:IF MN>VV(S) THEN MN=VV(S)
1960 RETURN
1970 REM
1980 REM IDENTIFY
1990 PRINT #4,"MATCHING SHAPEFOR CODE ";
     :PRINT #4,ID$:PRINT #4:MF=0:FOR
     I=1 TO NS
2000 PRINT #5,NS$(I);" CODE:":PRINT #5,
     CD$(I):IF ID$=CD$(I) THEN M=I:MF=1
2010 GOSUB 2100:NEXT:IF MF=1 THEN PRINT
     #4,"SHAPE IS:":PRINT #4, NS$(M)
     ELSE GOSUB 2050
2020 RETURN
2030 REM
2040 REM NEW SHAPE
2050 GOSUB 2100:NS=NS+1:PRINT #4,
     "NOT IDENTIFIED":GOSUB 2100
2060 PRINT #4,
     "TYPE NAME OF  THIS SHAPE"
2070 LOCATE 1,23:INPUT N$:NS$(NS)=N$:
     CD$(NS)=ID$:RETURN
```

```
2080 REM
2090 REM PAUSE
2100 FOR TP=1 TO 1000:NEXT:RETURN
2110 REM
2120 REM DATA FOR NEIGHBOURS
2130 DATA -1,-1,0,-1,1,-1,1,0,1,1,0,1
2140 DATA -1,1,-1,0,-2,-2,-1,-2,0,-2,1
2150 DATA -2,2,-2,2,-1,2,0,2,1,2,2,1,2
2160 DATA 0,2,-1,2,-2,2,-2,1,-2,0,-2,-1
```

Describe routine (Lines 1120-1350) :

Lines 1130-1140 set up the text windows needed by the program and display the title. Then a brief description of the program's operation is provided by lines 1150-1320. This text is left on the screen until the space-bar is detected by the routine 'prompt' with GOSUB 1370 at 1330. Then it is cleared by CLS #2 and two separate text windows, with a white background to contrast with the yellow of the screen, are created by 1330, 1340.

Prompt routine (Lines 1360-1390) :

Here the program's execution is delayed until the space-bar is detected at line 1370.

Polygon routine (Lines 1400-1640) :

In this routine the user is able to draw on the screen a shape for the program to identify. Line 1410 initialises the G array which represents the contents of the grid on which the shape is drawn. Although the grid itself consists of 15 rows and columns, the array needs to be dimensioned as 18 X 18. This is due to the form of the vertex-detection algorithm used subsequently in the program.

Line 1410 also clears the two arrays, VX and VY, which are used to store the coordinates of the vertices of the shape. The variable VC, initialised by 1420, counts the number of vertices detected. The starting point, (X, Y) , for the user's 'cursor', a filled square which can be moved around the grid, is

selected as the centre of the grid by line 1420. This line also sets the value, P, for the pause in the routine 'shade in'.

Explanatory text is placed in window #4 by lines 1430 - 1460 and the grid itself is drawn by the use of CLS #6 and two separate FOR loops between 1480-1500.

After this the shape can be drawn. The WHILE/WEND loop between lines 1510-1630 allows the filled square to be moved around the grid by adjusting the coordinates, X and Y, according to the cursor keys detected by lines 1520 - 1550. Lines 1560-1590 prevent the square from moving outside of the grid. Grid squares which are not part of the shape being designed are momentarily filled by 'shade in' called by GOSUB 1660 at line 1600 in order to show the location of the user's cursor. If, however, the space-bar is pressed, line 1610 alters the element of the G arrary corresponding to the relevant square to 1 and thus records part of the shape being built up. The same line also calls the routine 'square' to fill this square permanently as yellow, contrasting with the white background of the grid.

Line 1620 permits the routine to end when the user has finished the shape and pressed the S key.

Shade in routine (Lines 1650-1690):

The flashing cursor used in designing the shape on the grid is created by this routine. Line 1660 selects blue for the colour variable, C, and then uses 'square' to fill in the appropriate section of the grid. The length, P, of the empty FOR loop which follows depends upon which routine is calling 'shade in'. 'Line 1670 the uses 'square' to return the grid to white.

Square routine (Lines 1700-1720):

The screen coordinates, XP, YP, are calculated by line 1710 from the coordinates, X, Y, of the square on the grid. The code depends solely upon the geometric arrangement of the verticles of the shape and not their actual distances. This means that the program can be taught a name like 'rectangle' with reference to one shape that has been drawn and then will use it correctly to indentify any othe rectangle tested susequently, irrespective of its size or orientation.

Line 1740 places relevant text in window #5. Then the routine 'search grid' is called at 1760. This locates all vertices of the shape, total VC, and places their coordinates in the VX and VY arrays.

Line 1750 has initialised the VV array, used to store the distances between all possible pairs of vertices. Line 1760 sets to zero the number, S, of such distances and also the variable MN, used in the calculation of the minimum distance, in the procedure 'length'. (The value 400 is required for the largest-possible case of a square with edges along the perimeter of the grid.) The two nested FOR loops, lines 1760-1780 then call the routine 'length' with GOSUB 1950 to calculate the squared distance between all pairs of vertices. Following this, the FOR loop at line 1780 uses the minimum distance, MN, to convert all the distances to a set of ratios. It is this conversion which allows the same code to be produced for any size shape.

The final step in producing the identification code is to place all the converted distances into order. This is done by the simple sorting routine contained in the WHILE/WEND loop, lines 1780 - 1810. Then the numbers are concatenated into a single string, ID$, by the FOR loop at line 1810. The value of ID$ is displayed in window #5.

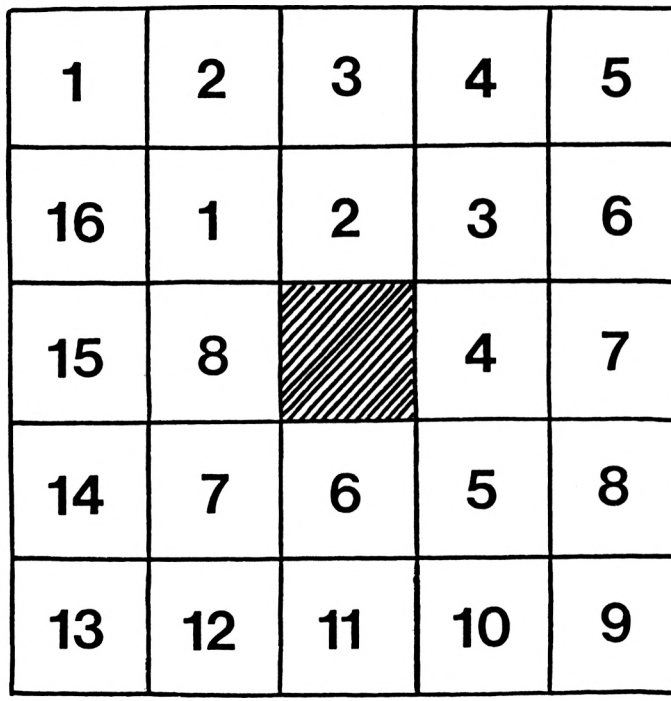
Search grid routine (Lines 1840-1870) :

This routine searches through the grid and the G array and locates all squares which are part of the shape. Line 1850 reduces the length of the pause, P, to zero to make the search quicker. Line 1850 also fills in all squares briefly except those which belong to the shape. For these, the routine 'detect vertex' is called.

Detect vertex routine (Lines 1880-1930) :

In 'detect vertex' each point of the shape is examined to see if it is a vertex. This is done by referring first to the eight neighbouring squares and then to the sixteen second closest neighbours. The coordinates for both sets of squares have already been placed into the arrays XA, YA, XB and YB by 'initialisation'.

The diagram shows the first neighbours, numbered 1 to 8. The FOR loop, 1890, 1900, checks the first four neighbours, and the square diagonally opposite each time, to see if both have a value in the G array of 1. If this is the case, then the central square lies on a side of the shape and is given a G array value of 2. A similar routine then follows for the second neighbours in lines 1900, 1910. If, at the end of both loops, the central square has not been given an array value of 2, it cannot lie between two points of a side of the shape and must therefore be a vertex. Line 1910 will then mark the point in blue on the grid, increment the vertex total, VC, and place the coordinates of the square into the VX and VY arrays. It also places an appropriate message in window #5.



Length routine (Lines 1940-1970) :

This routine is called by 'code shape'. It increments the total of distances calculated, S, at line 1950 and calculates the squared distance between the two vertices. In addition, line 1950 obtains the minimum distance of all, in order that the ratios can be calculated.

Identify routine (Lines 1980-2030) :

A message is placed in window #4 by line 1990, which then initialises the match flag, MF. The FOR loop, 1990-2010 checks through all the known codes in the array CD$ to see if there is a match with the code derived for the current shape, ID$. If there is, line 2000 sets MF equal to 1. At line 2010 the value of MF is used either to name the shape which matches the code or else to request the name of an unknown shape by a call to the routine 'new shape'.

New shape routine (Lines 2040-2080) :

The number of known shapes, NS, is incremented at 2050. A suitable request for a name is made by line 2060 and this, and its respective code, entered into the NS$ and CD$ arrays by 2070.

Pause routine (Lines 2090-2110) :

The FOR loop produces a delay.

Data (Lines 2120-2160) :

The coordinate information for the eight first neighbour squares and for the sixteen second neighbour squares is stored at these lines.

CHAPTER 8

The light of experience: robots and learning programs.

Robots are arriving slowly. They have been preceded by their name, since Karel Capek's play of 1923, and by their reputation since Talos, forged in Greek Mythology by Hephaestus. They have starred in films for over fifty years. They have appeared as toys, as threats and even as frauds, one radio-controlled device once clearly deceiving a television news team. Their name has suffered a bifurcation and can now refer to a most unanthropomorphic mass of computer-controlled machinery.

But robots have still not arrived. At least, they have not yet arrived in the bodily shape, and near demented intellect, of Robbie in 'Forbidden Planet'. Nobody has yet been accosted at a party by their host's latest Amstrad, booming at them in a stentorian rumble: 'I am monitored to bring you a Carlsberg Special'. Unfortunately, robots appear to have matched Artificial Intelligence generally in bringing the promised goods several generations too late. Like the General Problem Solver, and Natural Language, the robot has never quite managed to live up to its image...yet.

Nevertheless in a quiet way all the pieces are being assembled. Better A.I. programs get written. Computational power per unit cost increases, seemingly exponentially. University teams independently develop parts of the solution. It seems inevitable that one day the robot really will be trundling down to the park with the kids, just as Isaac Asimov always assured us that it would.

Robots in Industry

Again, financial considerations will be the determining factor. The cost of developing robots for domestic use would be quite prohibitive. Instead money will be found for industrial robots capable of working in impossible

conditions for men, like the interior of reactors. Deep ocean salvage will involve a high degree of robot technology. Exploration of Mars and the satellites of Jupiter, and perhaps even high-temperature Venus, will inevitably require very sophisticated levels of A.I. because no controlling link will be available to Mission Control in real time.

Once the systems have been built, and the research paid for, the extension into domestic life of what has been achieved will be cheap and irresistable. The robot will arrive.

At present it is the industrial robot which appears to have made the most progress. The first company devoted to robotics, Unimation, was founded in 1972. Since then more and more robots have appeared in factories. They are now used for industrial processes such as loading and unloading lathes, applying adhesives, welding, paint spraying, checking for damaged bottles and precision assembly. Computer numerical control of machine tools has been undoubtedly a breakthrough in production techniques. However it is hard to see that this type of 'robot', at the moment, really incorporates A.I. in its design, except in a very minimal sense. More interesting, perhaps, are the genuine robot-like advances being made. For example, a research team at Waseda University, Tokio, has concentrated upon developing a walking, bipedal robot. It is claimed that problems of balance have been solved to the point where the robot is almost as fast as a human being in performing this inherently human skill. Similarly, at Carnegie-Mellon, a wheeled robot has been built which can follow asphalt pathways around the campus and avoid trundling on the grass. Its vision algorithms are sufficiently developed to process the raw input of a video camera into an internal representation of the surroundings. This representation is good enough to prevent the machine trying to climb trees - most of the time!

Learning Algorithms

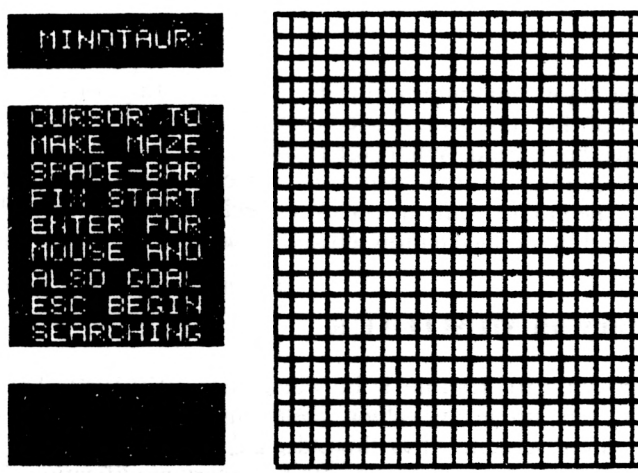
A vital feature of any piece of robotics software is the ability of the program to learn. Of course 'learning' is a fairly nebulous concept and might well seem very difficult to program. Certainly a generalised learning system, like the brain of a human infant, is beyond the scope of any current machine or software. There are, however, a number of points to bear in mind when writing a learning program because they make the task a little easier.

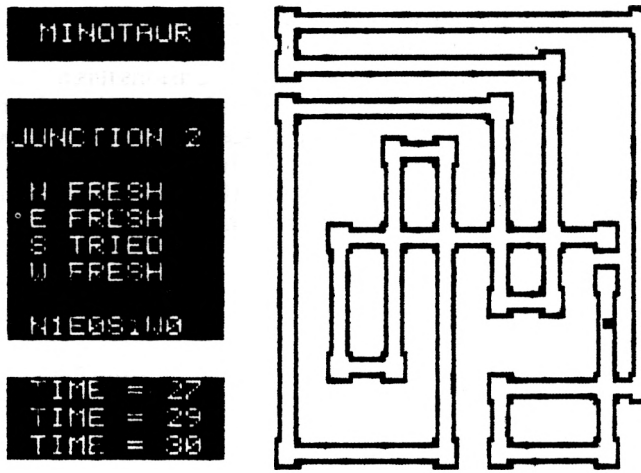
First, it is vital to adopt a 'top-down' programming style. In preparing the program to deal with the situations it will find itself in, it is important to divide the potential state of affairs into various substates. These are then redivided until small units are achieved which can be dealt with using simple algorithms. This approach is illustrated in the commentaries for the two programs in this chapter. Secondly, the internal model of the world that the

program contains requires an adequate method of representing knowledge. It has to be appropriate to the needs of the program. Thirdly, the program must be allowed reasonable feedback about the decisions it is making so that it can correct its mistakes. Finally, the program's internal model must not begin too far away from the true state of things if learning is to take place. This last condition is, quite evidently, similar to the human ability to learn what is already fairly familiar but initial difficulty with the completely unknown. (Most British people can pick up a few new words of French, but rarely manage the same with Chinese!)

MINOTAUR

A familiar example of amateur programming in A.I. is the 'Micromouse' competition. Regularly, enthusiasts compete with a home-built model (usually a combination of a glove puppet, Meccano set and Sinclair ZX 81) and see whose mouse can navigate most efficiently to the centre of a maze. The first program in this chapter offers all the thrills of such an event, but condenses the whole occasion on to the computer's monitor. Here a plan view of the maze is displayed and the mouse can be seen racing around in search of its goal.





The screen dumps show both the initial display of the program and how it appears when the mouse is searching the maze. It can be seen that the user is given a grid for designing the maze, but that once the search has begun the computer redraws a more realistic looking display.

The searching algorithm employed relies upon counting, or reidentifying, each junction of the maze as it is encountered and making a note of which directions have been tried. Wherever possible, the mouse avoids retaking the same direction. In this way the maze is searched thoroughly. Additionally, the mouse 'remembers' every decision made and is able, when the goal is found, to deduce a more efficient route.

Commentary on MINOTAUR

The program's control routine is between lines 1020-1040. The routine 'initialisation' sets up array values, then 'display', 'grid' and 'explain' produce the initial screen. After this, 'design' allows the user to sketch a maze on the screen. This is redrawn with necessary modifications by 'show maze' and afterwards 'search' begins to solve the maze by trial and error. Once the goal has been found, 'short route' produces a logical solution to the maze which is repeated indefinitely by the WHILE/WEND loop.

```
1000 REM MINOTAUR - PAT HALL, 2/86
1010 REM CONTROL ROUTINE
1020 GOSUB 1070:GOSUB 1130:GOSUB 1190
1030 GOSUB 1230:GOSUB 1300:GOSUB 1700
1040 GOSUB 1960:GOSUB 2410:WHILE TIME >0
     :GOSUB 1960:WEND
1050 REM
1060 REM INITIALISATION
1070 BORDER 0:MODE 1:INK 0,20:INK 1,2
1080 INK 2,26:INK 3,0:DIM M(22,22):DIM
     D$(4):FOR I=1 TO 4:READ D$(I):NEXT
1090 DIM NX(4):DIM NY(4):FOR I=1 TO 4
1100 READ NX(I),NY(I):NEXT:BP=0:GRAPHICS
     PEN 1:DIM JX(10):DIM JY(10):DIM JD(
     10,4):DIM NS(4):RETURN
1110 REM
1120 REM DISPLAY
1130 BORDER 2:PEN 1:PAPER 0:CLS:WINDOW
     #1,4,14,2,4:PAPER #1,2:CLS #1
1140 WINDOW #2,4,14,6,14:PAPER #2,2:CLS
     #2:WINDOW #3,4,14,16,21:PAPER #3,2
1150 CLS #3:WINDOW #4,17,37,2,21:PAPER
     #4,2:CLS #4:WINDOW #5,1,40,23,25
1160 PAPER #5,2:CLS #5:PAPER 2:LOCATE 5,
     3:PRINT "MINOTAUR:":RETURN
1170 REM
1180 REM GRID
1190 FOR I=256 TO 592 STEP 16:MOVE I,67:
     DRAW I,382:NEXT:FOR I=67 TO 382
     STEP 15:MOVE 256,I:DRAW 592,I:NEXT
1200 RETURN
```

```
1210 REM
1220 REM EXPLAIN
1230 PRINT #2," CURSOR TO":PRINT #2,
     " MAKE MAZE":PEN #2,3:PRINT #2,
     " SPACE-BAR":PRINT #2," FIX START"
1240 PEN #2,1:PRINT #2," ENTER FOR"
1250 PRINT #2," MOUSE AND":PRINT #2,
     " ALSO GOAL":PEN #2,3:PRINT #2,
     " 'S' BEGIN"
1260 PRINT #2," SEARCHING":PEN #2,1
1270 RETURN
1280 REM
1290 REM DESIGN
1300 X=11:Y=11:D=1:SF=0:S=0
1310 B=0:WHILE B=0
1320 IF INKEY(0)=0 THEN Y=Y+1:D=1
1330 IF INKEY(1)=0 THEN X=X+1:D=2
1340 IF INKEY(2)=0 THEN Y=Y-1:D=1
1350 IF INKEY(8)=0 THEN X=X-1:D=2
1360 IF X<1 THEN X=1
1370 IF X>21 THEN X=21
1380 IF Y<1 THEN Y=1
1390 IF Y>21 THEN Y=21
1400 IF INKEY(18)=0 THEN IF SF=0 THEN MX
     =X:MY=Y:SX=X:SY=Y:SF=1 ELSE GX=X:GY
     =Y
1410 IF INKEY(47)=0 THEN M(X,Y)=3:S=1:LD
     =D
1420 IF INKEY(60)=0 THEN M(X,Y)=3:B=1
1430 IF S=0 THEN GOSUB 1470 ELSE GOSUB
     1580
```

```
1440 WEND:CLEAR INPUT:RETURN
1450 REM
1460 REM SHOW
1470 C=3:GOSUB 1520:SOUND 1,67,10
1480 PT=1:GOSUB 1550:C=2:GOSUB 1520
1490 RETURN
1500 REM
1510 REM SQUARE
1520 GRAPHICS PEN C:PLOT X*16+248,Y*15+
     60:RETURN
1530 REM
1540 REM PAUSE
1550 FOR TP=1 TO PT*430:NEXT:RETURN
1560 REM
1570 REM MARK
1580 GRAPHICS PEN 1:CL=1:GOSUB 1660
1590 IF D<>LD THEN M(LX,LY)=3
1600 IF D=1 AND M(X,Y)=2 THEN M(X,Y)=3
1610 IF D=2 AND M(X,Y)=1 THEN M(X,Y)=3
1620 IF M(X,Y)=0 THEN M(X,Y)=D
1630 LD=D:LX=X:LY=Y:RETURN
1640 REM
1650 REM BLOCK
1660 MOVE X*16+240,Y*15+56:DRAWR 0,12
1670 DRAWR 14,0:DRAWR 0,-14:DRAWR -14,0:
     MOVER 3,3:FILL CL:RETURN
1680 REM
1690 REM SHOW MAZE
1700 PAPER #4,0:CLS #4:GRAPHICS PEN 0:
     MOVE 592,67:DRAW 592,382:CLS #2
1710 PRINT #2," CONSTRUCT":PRINT #2,
```

```
     " TURNS AND":PRINT #2," JUNCTIONS"
1720 FOR Y=1 TO 21:FOR X=1 TO 21
1730 IF M(X,Y)=3 THEN GRAPHICS PEN 3:CL=
     2:GOSUB 1660
1740 NEXT:NEXT:PRINT #2:PRINT #2,
     " LINK WITH  ALL PATHS":PRINT #2
1750 FOR Y=1 TO 21:FOR X=1 TO 21
1760 IF M(X,Y)=1 THEN GOSUB 1820
1770 IF M(X,Y)=2 THEN GOSUB 1870
1780 NEXT:NEXT:PRINT #2," SHOW GOAL":X=
     GX:Y=GY:C=3:GOSUB 1920:PRINT #2:PT=
     5:GOSUB 1550:PRINT #2," SEARCHING"
1790 PRINT #2:RETURN
1800 REM
1810 REM VERTICAL SECTION
1820 XP= X*16+243:YP=Y*15+53:FOR I=1 TO
     8:GRAPHICS PEN 3:IF I>2 AND I<7
     THEN GRAPHICS PEN 2
1830 MOVE XP+I,YP:DRAW XP+I,YP+16:NEXT
1840 RETURN
1850 REM
1860 REM HORIZONTAL SECTION
1870 XP= X*16+238:YP=Y*15+56:FOR I=1 TO
     8:GRAPHICS PEN 3:IF I>2 AND I<7
     THEN GRAPHICS PEN 2
1880 MOVE XP,YP+I:DRAW XP+18,YP+I:NEXT
1890 RETURN
1900 REM
1910 REM LOCATION
1920 GRAPHICS PEN C:PLOT X*16+247,Y*15+
     60:PLOT X*16+248,Y*15+60:PLOT X*16+
```

```
     247,Y*15+61:PLOT X*16+248,Y*15+61
1930 RETURN
1940 REM
1950 REM SEARCH
1960 MD=1:IF BP=1 THEN MX=SX:MY=SY
1970 JC=0:JN=0:FOR I=1 TO 10:JX(I)=0:JY(
     I)=0:JD(I,4)=0:NEXT:JP=0:TM=TIME:PT
     =1:IF BP=0 THEN R$=""
1980 CS=0:WHILE CS=0:X=MX:Y=MY:SOUND 1,
     119,10:C=1:GOSUB 1920:GOSUB 1550
1990 C=2:GOSUB 1920:FOR I=1 TO 4:NS(I)=0
     :NEXT:DC=0:FOR I=1 TO 4
2000 IF M(MX+NX(I),MY+NY(I))=0 THEN NS(I
     )=0 ELSE NS(I)=1:DC=DC+1
2010 NEXT:A$="":FOR I=1 TO 4:A$=A$+D$(I)
     +RIGHT$(STR$(NS(I)),1):NEXT:PRINT
     #2," ";A$:PRINT #2
2020 IF MD<3 THEN BD=MD+2 ELSE BD=MD-2
2030 IF DC=1 THEN MD=BD:PRINT #2,
     " DEAD END":PRINT #2
2040 IF DC=2 THEN GOSUB 2140
2050 IF DC=3 OR DC=4 THEN IF BP=0 THEN
     GOSUB 2180 ELSE GOSUB 2480
2060 IF MD=1 THEN MY=MY-1
2070 IF MD=2 THEN MX=MX+1
2080 IF MD=3 THEN MY=MY+1
2090 IF MD=4 THEN MX=MX-1
2100 GOSUB 2380:IF MX=GX AND MY=GY THEN
     CS=1
2110 WEND:RETURN
2120 REM
```

```
2130 REM PATH
2140 FOR I=4 TO 1 STEP -1:IF NS(I)=1 AND
     I<>BD THEN MD=I
2150 NEXT:RETURN
2160 REM
2170 REM CHECK JUNCTION
2180 FJ=0:FOR I=0 TO JC:IF MX=JX(I) AND
     MY=JY(I) THEN FJ=1:JN=I
2190 NEXT:IF FJ=0 THEN GOSUB 2230 ELSE
     GOSUB 2270
2200 RETURN
2210 REM
2220 REM NEW JUNCTION
2230 JC=JC+1:JN=JC:JX(JN)=MX:JY(JN)=MY
2240 GOSUB 2270:RETURN
2250 REM
2260 REM JUNCTION
2270 PRINT #2,"JUNCTION ";RIGHT$(STR$(JN
     ),1):PRINT #2:JD(JN,BD)=1:CJ=0:FOR
     I=1 TO 4:IF JD(JN,I)=1 THEN A$=
     " TRIED" ELSE A$=" FRESH"
2280 PRINT #2," ";D$(I);A$:CJ=CJ+JD(JN,I
     ):NEXT:PRINT #2:PT=3:GOSUB 1550:PT=
     1:IF CJ=4 THEN GOSUB 2340
2290 FOR I=4 TO 1 STEP -1:IF JD(JN,I)=0
     THEN MD=I
2300 NEXT:JD(JN,MD)=1:R$=R$+"J"+RIGHT$(
     STR$(JN),1)+"D"+RIGHT$(STR$(MD),1)
2310 PRINT #5," ROUTE TAKEN: ";:PRINT #5
     ,R$:PRINT #5:RETURN
2320 REM
```

```
2330 REM CLEAR JUNCTION
2340 PRINT #2," ALL ROUTES ATTEMPTED"
2350 PRINT #2:PT=2: GOSUB 1550:FOR I=1
     TO 4:JD(JN,I)=0:NEXT:PT=1:RETURN
2360 REM
2370 REM TIME
2380 PRINT #3," TIME =";INT((TIME-TM)/
     300):RETURN
2390 REM
2400 REM SHORT ROUTE
2410 CH=0:WHILE CH=0:RF=0:L=LEN(R$):FOR
     I=2 TO L STEP 4:FOR J=I+4 TO L STEP
     4:IF MID$(R$,I,1)=MID$(R$,J,1) THEN
     A=I: B=J:RF=1
2420 NEXT:NEXT:IF RF=0 THEN CH=1
2430 IF B<L THEN R$=LEFT$(R$,A)+RIGHT$(
     R$,L-B)
2440 WEND:IF R$<>"" THEN PRINT #5,
     " SHORT ROUTE: ";R$
2450 BP=1:RETURN
2460 REM
2470 REM BEST PATH
2480 JP=JP+1:MD=VAL(MID$(R$,JP*4,1))
2490 RETURN
2500 REM
2510 REM DATA
2520 DATA S,E,N,W,0,-1,1,0,0,1,-1,0
```

Initialisation routine (Lines 1060-1110) :

Line 1080 initialises the maze array, M. Then it places the four compass directions into the D$ array. The next-position arrays, NX, NY, are similarly established from DATA by 1090-1100. The flag BP is initialised at 1100 and arrays relating to the junctions of the maze set up.

Display routine (Lines 1120-1170) :

Here the screen is cleared, the title PRINTed and text windows, #1 – #5, set up.

Grid routine (Lines 1180-1210) :

MOVE and DRAW are used in two FOR loops at line 1190 to place a 21 X 21 grid on the screen. This is to enable the user to design the maze.

Explain routine (Lines 1220-1280) :

Prompts are placed in window #2 to tell the user that the cursor keys, space-bar, ENTER and S keys are all employed in designing the grid.

Design routine (Lines 1290-1450) :

In this routine the initial shape of the maze is sketched out on the grid. Line 1300 places the cursor coordinates at the centre of the grid and essential flags are set up. The WHILE/WEND loop between 1310-1440 then allows the designing to take place.

INKEY is used to scan the cursor keys and lines 1320-1350 adjust the coordinates of the cursor on the grid. They also update the variable D , which records the direction the cursor is moving. (Left/right = 2, up/down = 1.) Unacceptable values are prevented by 1360 - 1390. At first the cursor just moves across the grid without permanently marking it, because line 1430 calls 'show'. But once the space-bar has been pressed, and detected by 1410, the flag S becomes 1 and 'mark' is called by 1430. This routine records every position of the cursor on the grid to build up the maze. The first position marked is also recorded at 1410 by a value of 3 placed in the M array. This is needed later in the routine 'show maze'.

Line 1400 detects the ENTER key. The first time it is pressed the cursor coordinates are stored as (MX, MY) for the 'mouse' and also as (SX, SY) for use later in search'. The second time the key is pressed the coordinates are recorded as (GX, GY) for the goal.

Line 1420 detects the S key and allows 'design' to come to an end. As the cursor position will now be the end of the maze on the grid, another value of 3 in placed in the M array.

Show routine (Lines 1460-1500) :

This routine calls 'square' to mark momentarily the position of the cursor on the grid during the routine 'design'.

Square routine (Lines 1510-1530) :

PLOT is used to fill in one element of the grid in colour C.

Pause routine (Lines 1540-1560) :

A variable delay is provided by the FOR loop.

Mark routine (Lines 1570-1640) :

This routine has two important functions. First, line 1580 permanently marks the grid in blue to show the user that this is now part of the maze design. Second, the M array is correctly updated with values of 1, 2 or 3.

Line 1590 detects a change of direction and so marks the point in the M array with a value of 3 to represent a turn. Similarly, lines 1600, 1610 detect the maze crossing over itself to form a junction and mark this also with a value of 3. Simple vertical or horizontal extensions of the maze are represented by 1620 in the array with the D value of 1 or 2. At line 1630 the variables needed for the next call to 'mark' are set up.

Block routine (Lines 1650-1680) :

Here a section of the grid is shaded in with colour CL by the use of MOVE, DRAWR and FILL.

Show maze routine (Lines 1690-1800) :

When the user has finished designing the maze and pressed S, this routine redraws it, with artistic improvements! First, line 1700 removes the user's design. Then the nested FOR loops, 1720-1740 use the block routine again with GOSUB 1660 to place a junction graphic at each turn and junction of the maze, identified by the array values of 3. The second set of nested FOR loops, lines 1750 - 1780 call the vertical and horizontal section routines to create the rest of the maze.

Vertical section routine (Lines 1810-1850) :

The FOR loop uses MOVE and DRAW to produce a vertical maze section corresponding to one square of the grid. The GRAPHICS PEN colour change determined by line 1820 creates floor and outer walls for the section.

Horizontal section routine (Lines 1860-1900) :

Here a horizontal maze section is drawn.

Location routine (Lines 1910-1940) :

PLOT is used four times to mark the mouse or goal location in the maze.

Search routine (Lines 1950-2120)

The block of pixels representing a mouse is moved around the maze by this routine, which is used in two different ways in the program. When it is first called at line 1040, 'search' has to learn its way to the goal. But, by the time it is called again by the WHILE/WEND loop in the control routine it has learnt a more direct route. This difference in knowledge is reflected in the status of the best-path flag, BP. (For example at line 1960 the flag restores the mouse position to its initial value, (SX, SY) , for every pass through the WHILE/WEND loop.)

Lines 1960, 1970 initialise arrays and flags needed to deal with junctions in the maze, and the route string, R$. Then the WHILE/WEND loop, lines 1980-2110, moves the mouse through the maze.

The mouse's position in the maze, (MX, MY), is shown briefly by lines 1980, 1990. The FOR loop, 1990-2010, then uses the NX, NY arrays to examine the positions around the mouse in the M array. Every position where the array value is not zero represents a continuation of the maze. Such values are registered in the NS array at 2000 and also cause the variable DC to increment. The NS array can afterwards be used to show what directions are possible, as a message PRINTed in window #2 by line 2010, and DC used to choose which way to move the mouse.

Line 2020 defines the opposite direction, BD, from the last taken, MD. Then lines 2030-2050 decide the new direction for the mouse, MD, according to the number of possible paths open determined by the current value of DC generated at 2000. If there is only one direction, in other words a dead-end has been reached, line 2030 makes the mouse reverse its direction. Two directions will call 'path' at 2040. More directions than this calls either 'check junction' or 'best path' at 2050, depending upon whether the mouse is learning its way or already knows it.

The MD value returned by the routines called between 2030-2050 is used to adjust the mouse coordinates between lines 2060-2090. The time taken so far is displayed by a call to the routine 'time' with GOSUB 2380 at line 2100, which also detects if the goal location has been reached. If it has, the loop terminates and ends the routine. The control routine can then call 'short route' to deduce a more efficient path to the goal.

Path routine (Lines 2130-2160)

When there are two directions the mouse can take, this routine identifies the correct one by ignoring the direction the mouse has come from.

Check junction routine (Lines 2170-2210)

Here TICTACTOE decides whether a junction reached in the maze is new, or one already visited, by comparing the mouse coordinates with the locations of all the known junctions in the arrays JX, JY. This is done at line 2180 in the FOR loop. Either 'new junction' or 'junction' is called. A previously visited junction is identified by the junction-number, JN.

New junction routine (Lines 2220-2250)

A junction not met before is added to the JX, JY array. The total junction count, JC, is incremented and this value will also be the junction-number, JN. The routine 'junction' is then called.

Junction routine (Lines 2260-2320)

This routine will always be reached, sometimes via 'new junction', when the mouse is exploring the maze and encounters an intersection. Line 2270 names the junction on the screen.

A logical search of the maze is produced simply by recording the direction taken at every junction in the junction-direction array, JD, and by ensuring that the same direction is not taken twice until all directions out of the junction have been attempted. Line 2270, for example, updates the JD array by registering the direction in which the current junction was approached, since obviously this is not a path to try again. The FOR loop, 2270 - 2280, shows the user which routes from the junction still have to be tried.

If all routes have been tried, line 2280 calls 'clear junction' to reset the JD array and allow directions to be tried again. The loop, lines 2290, 2300, uses the array to select a direction and the MD value generated here is immediately used to modify the JD array at 2300.

At each arrival at a junction, the route string, R$, is updated, at 2300, to show the path so far taken through the maze. R$ is displayed in window #5 at line 2310.

Clear junction routine (Lines 2330-2360)

This routine clears the JD array for a particular junction.

Time routine (Lines 2370-2390)

The time taken in exploring the maze is determined by TIME.

Short route routine (Lines 2400-2460)

This is the most 'intelligent' part of the program. It looks at the route string, R$, and decides which parts of the earlier path taken should be eliminated. A typical route string could be, for example:

```
J 1 D 2   J 2 D 1   J 3 D 1   J 1 D 1
```

Extra spaces have been included to make the string easier to analyse. It can be seen how each junction and direction taken are identified by a single digit. (The user must not attempt a maze with more than nine junctions.) It can also be seen that the middle part of the string is superfluous, since junction 1 was the one eventually required. In fact, a logical route would have been `J 1 D 1.`

The WHILE/WEND loop looks for precisely this sort of redundancy. The I and J loops STEP by 4 through R$, looking for repeated junctions. When one is found, line 2430 cuts out the part of the string between the two and merges the two end pieces into a shorter route. This process is repeated until the RF flag indicates that no further simplification is possible.

Best path routine (Lines 2470-2500)

This routine replaces 'check junction' once the control routine reaches the WHILE/WEND loop. It uses the modified R$ to determine the direction at each junction.

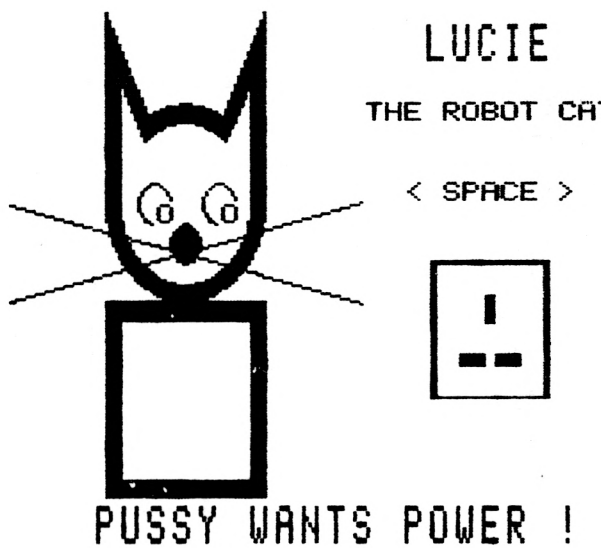
Data (Lines 2510-2520)

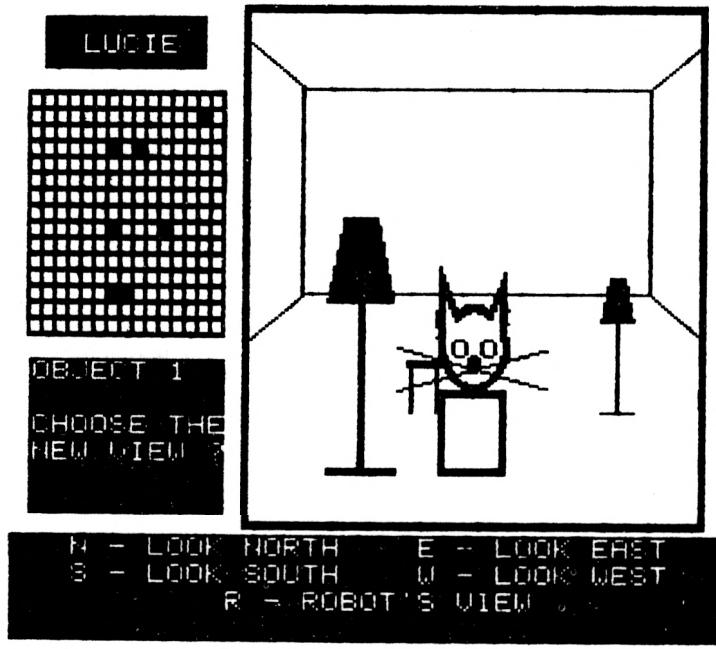
Necessary information for 'initialisation' is stored at line 2520.

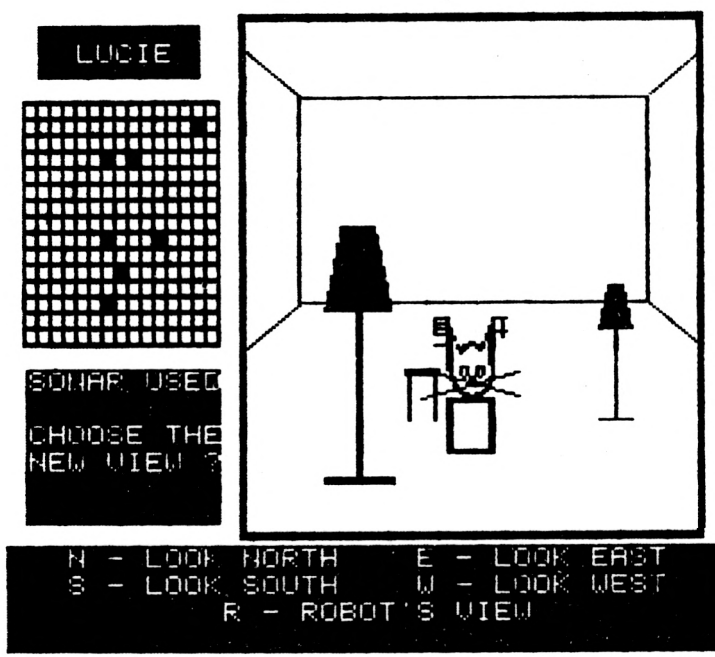
LUCIE

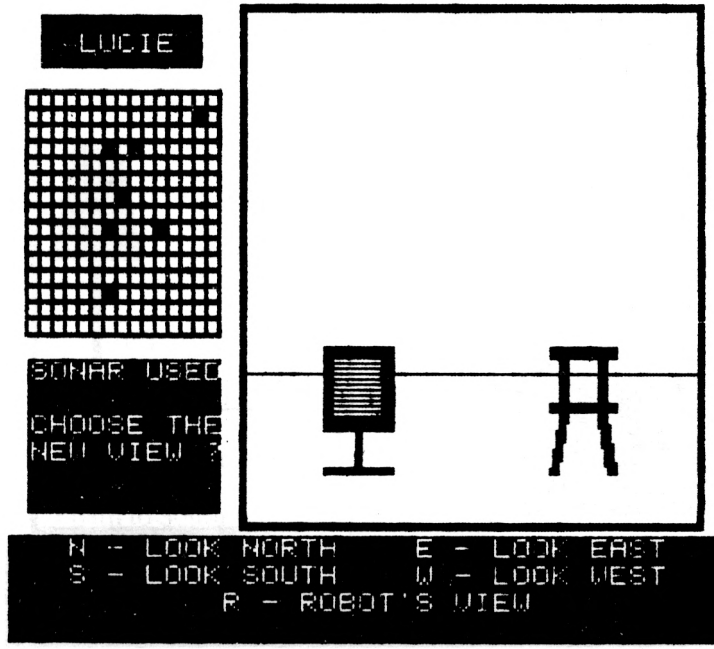
A early robotics project at S.R.I. International, under Charles Rosen and David Nitzan, was 'Shakey'. This was a small self-propelled wheeled vehicle, equipped with a tv camera, range finder and touch sensors, and controlled, via a radio link, by a PDP-10 computer. Shakey could navigate its way around separate rooms of the laboratory, recognise regularly shaped objects and even obey simple commands like: 'Push the box off the platform'.

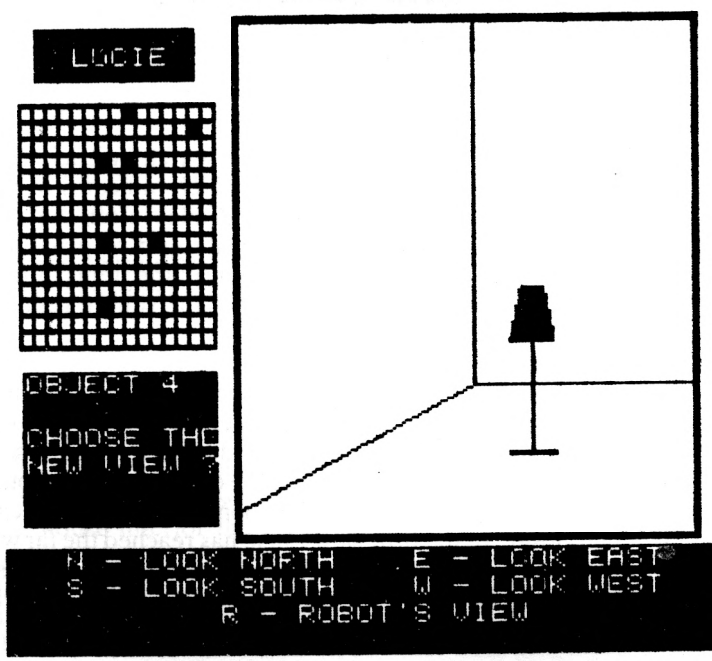
The last program of this book is an attempt to recreate a Shakey-type robot. As the previous program involved a mouse, and as fame was achieved by the robotic dog, K9, 'Lucie', to redress the balance, is a cat. She lives in a typical domestic interior, with standard lamps, tables, chairs and tv sets, and her one goal in life is to find a power socket and plug herself in.

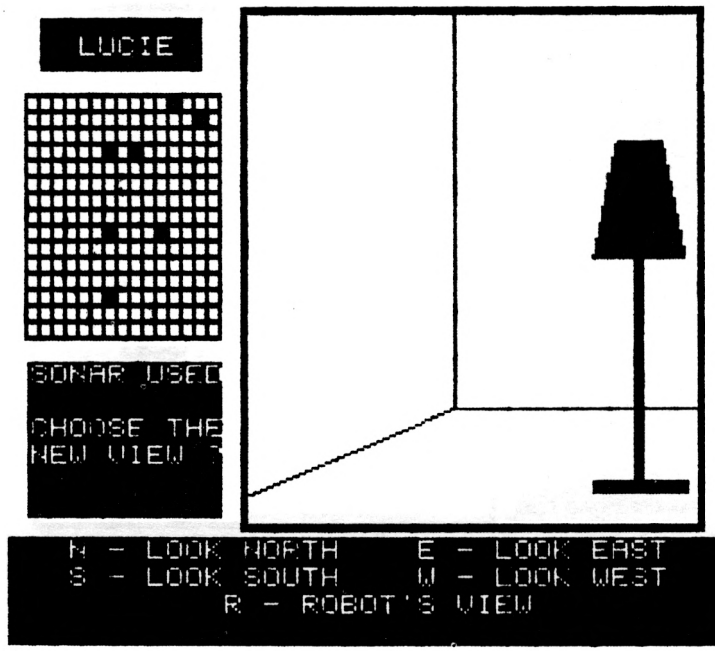



LUCIE
OBJECT 1
CHOOSE THE
NEW VIEW ?
N - LOOK NORTH
E - LOOK EAST
S - LOOK SOUTH
W - LOOK WEST
R - ROBOT'S VIEW


LUCIE
SONAR USED
CHOOSE THE
NEW VIEW ?
N - LOOK NORTH
E - LOOK EAST
S - LOOK SOUTH
W - LOOK WEST
R - ROBOT'S VIEW


LUCIE
SONAR USED
CHOOSE THE
NEW VIEW ?
N - LOOK NORTH
E - LOOK EAST
S - LOOK SOUTH
W - LOOK WEST
R - ROBOT'S VIEW


LUCIE
OBJECT 4
CHOOSE THE
NEW VIEW ?
N - LOOK NORTH
E - LOOK EAST
S - LOOK SOUTH
W - LOOK WEST
R - ROBOT'S VIEW



The screen dumps show the beginning of a typical search by Lucie. The first illustration is simply the initial screen display, which indicates to the user what Lucie intends to do. The second is the display after the user has designed the room's interior, using the coordinate grid to place items of furniture. In each of the screen dumps that follow all the items of furniture occupy exactly the same position on the grid, but the square representing Lucie moves around.

On the right hand side of the screen is a picture of the room. The user can choose to view from any of the four walls, or instead 'see' the room through the eyes of the robot. In the second illustration Lucie is beside the standard lamp. In the third she has moved further into the distance. In the fourth the view is the one from the cat's position in the room. The tv set and chair, of course, look much closer. In the fifth view Lucie has reached the far wall and turned right. The corner of the room can now be seen and the second standard lamp. In the last view Lucie has moved closer to the East wall and so the lamp is much larger.

The room searching algorithms Lucie employs can be summarised as follows:

1. Throughout the search, use sonar to scan the room. Make a note of any objects detected.

2. If the sonar detects a power socket, move straight towards it.

3. Begin by finding a clear path, unblocked by any furniture, to the North wall of the room. Move East or West until the clear path is found.

4. Move to the North Wall.

5. Find a clear path to the East wall. Take it.

6. Return to initial position in room.

7. Find clear path to South wall. Take it.

8. Find clear path to West wall. Take it.

9. Move to first item of furniture in the room. Move all the way around it, in case it was blocking a direct view of a socket.

10. Repeat for all items of furniture in the room.

11. Give up!

This algorithm gives an example of the 'top-down' approach mentioned above. The individual goals in the list all require many sub-goals.

During the execution of the program, Lucie's sonar can be seen represented on the grid. When eventually she finds the power socket she purrs!

Commentary on LUCIE

LUCIE is controlled by lines 1020 - 1040. The routine 'initialisation' called by line 1020 sets up variables and arrays needed by the program. Then 'title screen' creates the initial screen shown. Next, GOSUB 1210 calls 'room' to show the house interior the robot cat has to explore. GOSUB 1250 calls 'display' and GOSUB 1320 draws the coordinate grid on which the room's contents are planned. After this 'design' called by GOSUB 1440 allows furniture, and the robot cat, to be added. Finally, the routine 'find socket' controls the robot's search around the room.

```
1000 REM LUCIE - PAT HALL, 2/86
1010 REM CONTROL ROUTINE
1020 GOSUB 1070:GOSUB 1120:GOSUB 1210
1030 GOSUB 1250:GOSUB 1320:GOSUB 1440
1040 GOSUB 2770:STOP
1050 REM
1060 REM INITIALISATION
1070 BORDER 0:MODE 1:INK 0,26:INK 1,24
1080 INK 2,20:INK 3,0:DIM G(15,15):XC=8:
     YC=8:XW=8:YW=1:XV=8:YV=1:NV=78:V=78
     :RF=0:GD=3:PAPER 0:DIM TX(3):SP=0
1090 DIM TY(3):RETURN
1100 REM
1110 REM TITLE SCREEN
1120 BORDER 20:PAPER 2:CLS:X=164:Y=40:S=
     144:GOSUB 2390:PEN 3:PAPER 1:LOCATE
     10,10:PRINT"O":LOCATE 13,10:PRINT
     "O":X=490:Y=56:S=200:GOSUB 2570
1130 PAPER 2:LOCATE 29,4:PRINT"LUCIE"
1140 LOCATE 25,6:PRINT"THE ROBOT CAT"
1150 LOCATE 12,25:PRINT"PUSSY WANTS ";
1160 PRINT"POWER !":K=0:WHILE K=0:IF
     INKEY(47)=0 THEN K=1
1170 LOCATE 27,9:PRINT"< SPACE >":PT=2
1180 GOSUB 1820:LOCATE 27,9:PRINT SPC(9)
     :GOSUB 1820:WEND:CLEAR INPUT:BORDER
     24:PAPER 0:RETURN
1190 REM
1200 REM ROOM
1210 GRAPHICS PEN 3:CLS:A=YW*8:B=188-YW*
     8:XL=(1-XW)*608/14:YT=YW*8+332:MOVE
```

```
      XL-A,0:DRAW XL+392-A,B:DRAW XL+859+
      A,B:DRAW XL+1251+A,0:IF XW<7 THEN
      MOVE XL+392-A,B-10 ELSE MOVE XL+859
      +A,B-10
1220 FILL 2:MOVE XL+625,B-10:FILL 2:MOVE
      XL-A,YT+B:DRAW XL+392-A,YT:DRAW XL+
      859+A,YT:DRAW XL+1251+A,YT+B:FOR I=
      392-A TO 859+A STEP 467+A*2:MOVE XL
      +I,B:DRAW XL+I,YT:NEXT:RETURN
1230 REM
1240 REM DISPLAY
1250 PEN 3:PAPER 1
1260 LOCATE 17,2:PRINT" LUCIE "
1270 PAPER 0
1280 WINDOW #1,2,12,4,13:PAPER #1,0
1290 WINDOW #2,2,12,17,18:PAPER #2,0:PEN
      #2,3:WINDOW #3,1,40,24,25:PAPER #3,
      0:PEN #3,0:RETURN
1300 REM
1310 REM GRID
1320 GRAPHICS PEN 3:CLS #1:FOR I=15 TO
      195 STEP 12:MOVE I,189:DRAW I,354:
      NEXT:FOR I =189 TO 354 STEP 11:MOVE
      15,I:DRAW 195,I:NEXT:GRAPHICS PEN 0
      :PLOT 81,191:FOR I=52 TO 56 STEP 2
1330 PLOT I,352:NEXT:CLS #2:CLS #3:C=3
1340 FOR I=1 TO 15:FOR J=1 TO 15:IF G(I,
      J)<>0 THEN XX=I:YY=J:GOSUB 1790
1350 NEXT:NEXT:IF V=69 THEN CP$="EAST "
1360 IF V=78 THEN CP$="NORTH "
1370 IF V=83 THEN CP$="SOUTH "
```

```
1380 IF V=87 THEN CP$="WEST "
1390 PRINT #2,CP$;"VIEW"
1400 IF SP=1 THEN LOCATE 2,24:PRINT
     "N-LOOK NORTH E-LOOK EAST S";:PRINT
     "-LOOK SOUTH":LOCATE 8,25:PRINT
     "W-LOOK WEST R-ROBOT'S VIEW"
1410 RETURN
1420 REM
1430 REM DESIGN
1440 LOCATE 6,24:PRINT"MOVE-CURSOR";
1450 PRINT"   PLACE ITEM-SPACE"
1460 DS=0:WHILE DS=0
1470 IF INKEY(0)=0 THEN YC=YC+1
1480 IF INKEY(1)=0 THEN XC=XC+1
1490 IF INKEY(2)=0 THEN YC=YC-1
1500 IF INKEY(8)=0 THEN XC=XC-1
1510 IF INKEY(47)=0 THEN GOSUB 1610
1520 IF INKEY(54)=0 AND RF =1 THEN DS=1
1530 IF XC<1 THEN XC=1
1540 IF XC>15 THEN XC=15
1550 IF YC<1 THEN YC=1
1560 IF YC>15 THEN YC=15
1570 IF G(XC,YC)=0 THEN C=3:XX=XC:YY=YC:
     GOSUB 1790:PT=1:GOSUB 1820:C=0:
     GOSUB 1790
1580 WEND:RETURN
1590 REM
1600 REM CHOOSE ITEM
1610 CLS #3:LOCATE 5,24:PRINT"C - CHAIR"
     ;:PRINT"  L - LIGHT";:PRINT"  S -";
     :PRINT" SOCKET":LOCATE 5,25:PRINT
```

```
     "T - TABLE  V - TV SET";
1620 IF RF=0 THEN PRINT"  R - ROBOT"
1630 LK=0:WHILE LK=0
1640 IF INKEY(36)=0 THEN LK=1:K=76
1650 IF INKEY(50)=0 THEN LK=1:K=82
1660 IF INKEY(51)=0 THEN LK=1:K=84
1670 IF INKEY(55)=0 THEN LK=1:K=86
1680 IF INKEY(60)=0 THEN LK=1:K=83
1690 IF INKEY(62)=0 THEN LK=1:K=67
1700 WEND:IF G(XC,YC)=0 AND K<>82 THEN
     G(XC,YC)=K
1710 IF K=82 AND RF=0 AND G(XC,YC)=0
     THEN G(XC,YC)=82:XR=XC:YR=YC:XB=XC:
     YB=YC:RF=1
1720 GOSUB 1210:GOSUB 1850:GOSUB 1320
1730 LOCATE 6,24:PRINT"MOVE-CURSOR";
1740 PRINT"   PLACE ITEM-SPACE"
1750 IF RF=1 THEN LOCATE 9,25:PRINT
     "BEGIN ROBOT'S SEARCH - B"
1760 RETURN
1770 REM
1780 REM SQUARE
1790 XP=XX*12+4:YP=YY*11+180:GRAPHICS
     PEN C:MOVE XP,YP:DRAWR 0,7:DRAWR 8,
     0:DRAWR 0,-7:DRAWR -8,0:MOVER 2,2:
     FILL C:RETURN
1800 REM
1810 REM PAUSE
1820 FOR TP=1 TO PT*200:NEXT:RETURN
1830 REM
1840 REM VIEW_N
```

```
1850 FOR J=15 TO YV+1 STEP -1:FOR I=1 TO
     15:W=(15+YV-J)*10/15:IF ABS(I-XV)<=
     10-W AND G(I,J)<>0 THEN X=640*(10-
     XV-W+I)/(20-W*2):Y=(J-YV)*12:S=(16-
     J+YV)*4:GOSUB 2010:GOSUB 2140
1860 NEXT:NEXT:RETURN
1870 REM
1880 REM VIEW_E
1890 FOR I=15 TO XV+1 STEP -1:FOR J=15
     TO 1 STEP -1:W=(15+XV-I)*10/15:IF
     ABS(J-YV)<=10-W AND G(I,J)<>0 THEN
     X=640*(10+YV-W-J)/(20-W*2):Y=(I-XV)
     *12:S=(16-I+XV)*4:GOSUB 2010:GOSUB
     2140
1900 NEXT:NEXT:RETURN
1910 REM
1920 REM VIEW_S
1930 FOR J=1 TO YV-1:FOR I=15 TO 1 STEP
     -1:W=(15-YV+J)*10/15:IF ABS(I-XV)<=
     10-W AND G(I,J)<>0 THEN X=640*(10+
     XV-W-I)/(20-W*2):Y=(YV-J)*12:S=(16+
     J-YV)*4:GOSUB 2010:GOSUB 2140
1940 NEXT:NEXT:RETURN
1950 REM
1960 REM VIEW_W
1970 FOR I=1 TO XV-1:FOR J=1 TO 15:W=(15
     -XV+I)*10/15:IF ABS(J-YV)<=10-W AND
     G(I,J)<>0 THEN X=640*(10-YV-W+J)/(
     20-W*2):Y=(XV-I)*12:S=(16+I-XV)*4:
     GOSUB 2010:GOSUB 2140
1980 NEXT:NEXT:RETURN
```

```
1990 REM
2000 REM ADJUST
2010 IF S>45 THEN S=S+(S-45)^1.6
2020 RETURN
2030 REM
2040 REM ROBOT VIEW
2050 XV=XR:YV=YR
2060 IF RV=1 THEN XW=XR:YW=YR:V=78:GOSUB
     1210:GOSUB 1850
2070 IF RV=2 THEN XW=15-YR:YW=XR:V=69:
     GOSUB 1210:GOSUB 1890
2080 IF RV=3 THEN XW=15-XR:YW=15-YR:V=83
     :GOSUB 1210:GOSUB 1930
2090 IF RV=4 THEN XW=YR:YW=15-XR:V=87:
     GOSUB 1210:GOSUB 1970
2100 GOSUB 1320:PRINT #2,"FROM ROBOT"
2110 RETURN
2120 REM
2130 REM ITEM TYPE
2140 IF G(I,J)=67 THEN GOSUB 2230
2150 IF G(I,J)=76 THEN GOSUB 2320
2160 IF G(I,J)=82 THEN GOSUB 2390
2170 IF G(I,J)=83 THEN GOSUB 2570
2180 IF G(I,J)=84 THEN GOSUB 2620
2190 IF G(I,J)=86 THEN GOSUB 2680
2200 RETURN
2210 REM
2220 REM DRAW CHAIR
2230 A=S/2:E=S/5:B=A+E:C=S*0.15:D=C/2:F=
     S-C:MOVE X-A,Y+B:DRAWR 0,C:DRAWR S,
     0:DRAWR 0,-C:DRAWR -S,0:MOVER D,D
```

```
2240 FILL 3:MOVER S-C,0:FILL 3:MOVE X-A,
     Y:DRAWR C,B:DRAWR C,0:DRAWR -C,-B
2250 DRAWR -C,0:MOVER C,B/2:FILL 3:MOVE
     X+A,Y:DRAWR -C,0:DRAWR -C,B:DRAWR C
     ,0:DRAWR C,-B:MOVER -C,B/2:FILL 3
2260 IF V<>69 THEN MOVE X+E,Y+F:DRAWR 0,
     A:DRAWR C,0:DRAWR 0,-A:DRAWR -C,0:
     MOVER D,D:FILL 3:MOVER 0,A-C:FILL
     3
2270 IF V<>87 THEN MOVE X-E,Y+F:DRAWR 0,
     A:DRAWR -C,0:DRAWR 0,-A:DRAWR C,0:
     MOVER -D,D:FILL 3:MOVER 0,A-C:FILL
     3
2280 IF V=78 OR V=83 THEN MOVE X-A,Y+A+F
     :DRAWR 0,C:DRAWR S,0:DRAWR 0,-C:
     DRAWR -S,0:MOVER D,D:FILL 3:MOVER S
     -C,0:FILL 3
2290 RETURN
2300 REM
2310 REM DRAW LAMP
2320 A=S*3:B=S*2:C=S/2:D=S/4:E=C/5:F=E/2
     :GRAPHICS PEN 1:MOVE X-C,Y+B:DRAWR
     D,S:DRAWR C,0:DRAWR D,-S:DRAWR -S,0
     :MOVER F,F:FILL 1:GRAPHICS PEN 3
2330 MOVE X-C,Y:DRAWR 0,E:DRAWR S,0
2340 DRAWR 0,-E:DRAWR -S,0:MOVER F,F
2350 FILL 3:FOR K=X-F TO X+F:MOVE K,Y
2360 DRAW K,Y+B:NEXT:RETURN
2370 REM
2380 REM DRAW ROBOT
2390 A=S/2:B=A*3:C=A/5:D=S*1.1
```

```
2400 FOR F=0 TO C STEP C
2410 IF F=0 THEN RC=3 ELSE RC=1
2420 GRAPHICS PEN RC:FOR K=0 TO S-F*4
     STEP 2:XE=A*(S-K)/S-F*2:MOVE X-A+F,
     Y+B+K:DRAW X-A+F+XE,Y+B+K:MOVE X+A-
     F,Y+B+K:DRAW X+A-F-XE,Y+B+K:NEXT:R=
     A-F:CY=Y+B:GOSUB 2500
2430 MOVE X-A+F,Y+F:DRAWR 0,S-F*2:DRAWR
     S-F*2,0:DRAWR 0,F*2-S:DRAWR F*2-S,0
     :MOVER C,C:FILL RC
2440 IF F=0 THEN MOVER 0,S-C-2:FILL 3
2450 NEXT:GRAPHICS PEN 3
2460 IF S>20 THEN CY=Y+S*1.3:R=C:GOSUB
     2500:CX=X-S*0.15:CY=Y+S*1.4:R=S/60:
     GOSUB 2530:CX=X+S*0.15:GOSUB 2530:
     MOVE X-D,Y+S:DRAW X+D,Y+B:MOVE X-D,
     Y+B:DRAW X+D,Y+S
2470 RETURN
2480 REM
2490 REM FILL CIRCLE
2500 FOR K=CY-R TO CY+R STEP 2:X1=X-SQR(
     ABS(R^2-(K-CY)^2)):X2=X*2-X1:MOVE
     X1,K:DRAW X2,K:NEXT:RETURN
2510 REM
2520 REM DRAW CIRCLE
2530 DEG:MOVE CX,CY:FOR K=1 TO 360 STEP
     10:PLOTR R*COS(K),R*SIN(K):NEXT
2540 RETURN
2550 REM
2560 REM DRAW SOCKET
2570 A=S/36:B=S/2:GRAPHICS PEN 0:MOVE X-
```

```
      A*9,Y+A*10:DRAWR 0,B:DRAWR B,0
2580 DRAWR 0,-B:DRAWR -B,0:MOVER A*2,A*2
      :FILL 0:GRAPHICS PEN 3:MOVER A*2,A*
      2:DRAWR 0,A*2:DRAWR A*4,0:DRAWR 0,
      -A*2:DRAWR -A*4,0:MOVER A,A:FILL 3
2590 MOVER A*3,A*5:DRAWR 0,A*4:DRAWR A*2
      ,0:DRAWR 0,-A*4:DRAWR -A*2,0:MOVER
      A,A:FILL 3:MOVE X+A,Y+A*14:DRAWR 0,
      A*2:DRAWR A*4,0:DRAWR 0,-A*2:DRAWR
      -A*4,0:MOVER A,A:FILL 3:RETURN
2600 REM
2610 REM DRAW TABLE
2620 C=S/2:B=C/5:A=C-B:D=S/5:E=C+D:F=B/2
      :GRAPHICS PEN 3:FOR K=0 TO E STEP E
      :MOVE X-A+K,Y:DRAWR 0,S:DRAWR B,0
2630 DRAWR 0,-S:DRAWR -B,0:MOVER F,F
2640 FILL 3:MOVER 0,S-B:FILL 3:NEXT:MOVE
      X-C,Y+S:DRAWR 0,D:DRAWR S,0:DRAWR 0
      ,-D:DRAWR -S,0:MOVER F,F:FILL 3
2650 MOVER S-B,0:FILL 3:RETURN
2660 REM
2670 REM DRAW TV
2680 A=S/2:B=A/5:C=B/2:FOR K=0 TO S*0.15
      STEP S*0.15:IF K=0 THEN TC=3 ELSE
      IF V=78 THEN TC=0
2690 GRAPHICS PEN TC:MOVE X-A+K,Y+A+K
2700 DRAWR 0,S-K*2:DRAWR S-K*2,0:DRAWR 0
      ,K*2-S:DRAWR K*2-S,0:MOVER S/5,S/5
2710 FILL TC:IF K=0 THEN MOVER 0,S*0.79:
      FILL TC
2720 NEXT
```

```
2730 GRAPHICS PEN 3:MOVE X-A,Y:DRAWR 0,B
     :DRAWR S,0:DRAWR 0,-B:DRAWR -S,0
2740 MOVER C,C:FILL 3:MOVE X-C,Y+B:DRAWR
     0,A-B:DRAWR B,0:DRAWR 0,B-A:DRAWR
     -B,0:MOVER C,C:FILL 3:RETURN
2750 REM
2760 REM FIND SOCKET
2770 DIM OX(10):DIM OY(10):OC=0:FS=0:DS=
     1:XM=1:FVU=1:FHR=0:FVD=0:FHL=0:VU=0
     :HR=0:RS=0:VD=0:HL=0:SP=1:NP=0
2780 WHILE NP=0:PX=XR:PY=YR
2790 IF FVU=1 THEN GOSUB 3070
2800 IF YR=15 AND VU=1 THEN FHR=1:DS=2
2810 IF YR=15 THEN VU=0
2820 IF FHR=1 THEN GOSUB 3120
2830 IF VU=1 THEN YR=YR+1:RV=1
2840 IF XR=15 AND HR=1 THEN RS=1
2850 IF XR=15 THEN HR=0
2860 IF RS=1 THEN GOSUB 3160
2870 IF HR=1 THEN XR=XR+1:RV=2
2880 IF FVD=1 THEN GOSUB 3380
2890 IF YR=1 AND VD=1 THEN FHL=1:DS=4
2900 IF YR=1 THEN VD=0
2910 IF FHL=1 THEN GOSUB 3430
2920 IF VD=1 THEN YR=YR-1:RV=3
2930 IF HL=1 THEN XR=XR-1:RV=4
2940 GOSUB 3640:IF XR=1 AND HL=1 THEN NP
     =1
2950 IF FVU=0 AND FHR=0 AND FVD=0 AND
     FHL=0 THEN GOSUB 3510
2960 IF FS=1 THEN GOSUB 3980
```

```
2970 WEND:GD=1:FOR GT=1 TO OC:BX=OX(GT):
     BY=OY(GT):IF BX>1 AND BX<15 AND BY>
     1 AND BY<15 THEN GOSUB 3020
2980 NEXT:CLS #3:LOCATE 10,24:PRINT
     "FAILED TO FIND SOCKET":LOCATE 1,1
2990 RETURN
3000 REM
3010 REM PEER ROUND
3020 PRINT #2,"INSPECTING":PRINT #2,
     "OBJECT";GT:PT=10:GOSUB 1820:XG=BX
3030 YG=BY:GR=0:PR=0:WHILE PR=0:PX=XR:PY
     =YR:GOSUB 3200:GOSUB 3640:IF GR=1
     THEN PX=XR:PY=YR:PR=1
3040 WEND:GOSUB 3540:RETURN
3050 REM
3060 REM PATH 1
3070 IF XR=15 THEN XM=-1
3080 GOSUB 3470:GOSUB 3800:IF FW=1 THEN
     PRINT #2," YES, ";:FVU=0:VU=1 ELSE
     PRINT #2," NO ";:XR=XR+XM
3090 PRINT #2," PATH    ":RV=3-XM:PT=8:
     GOSUB 1820:RETURN
3100 REM
3110 REM PATH 2
3120 GOSUB 3470:GOSUB 3800:IF FW=1 THEN
     PRINT #2," YES, ";:FHR=0:HR=1 ELSE
     PRINT #2," NO ";:YR=YR-1
3130 PRINT #2," PATH    ":RV=3:PT=8:GOSUB
     1820:RETURN
3140 REM
3150 REM RESTART
```

```
3160 XG=XB:YG=YB:GR=0:GOSUB 3200:IF GR=1
     THEN RS=0:FVD=1:DS=3:XM=1
3170 RETURN
3180 REM
3190 REM GOAL
3200 IF XG=XR THEN XD=0 ELSE XD=(XG-XR)/
     ABS(XG-XR)
3210 IF YG=YR THEN YD=0 ELSE YD=(YG-YR)/
     ABS(YG-YR)
3220 XN=XR+XD:YN=YR+YD:IF ABS(XG-XN)+ABS
     (YG-YN)<GD THEN GR=1
3230 IF GR=0 THEN IF G(XN,YN)=0 THEN XR=
     XN:YR=YN ELSE GOSUB 3280
3240 IF ABS(XG-XR)>=ABS(YG-YR) THEN RV=
     XD^2-XD+2 ELSE RV=YD^2-YD+1
3250 RETURN
3260 REM
3270 REM JUMP
3280 PRINT #2," OBSTACLE":XA=XR:YA=YR:AL
     =0:WHILE AL=0:JF=0:GOSUB 3340:XR=XA
     +RN:IF XR=XA THEN GOSUB 3340:YR=YA+
     RN
3290 IF XR=XA AND YR=YA THEN JF=1
3300 IF G(XR,YR)=0 AND JF=0 THEN AL=1
3310 WEND:PT=10:GOSUB 1820:RETURN
3320 REM
3330 REM RANDOM
3340 RN=INT(RND(1)*3)-1
3350 RETURN
3360 REM
3370 REM PATH 3
```

```
     IF XR=15 THEN XM=-1
3390 GOSUB 3470:GOSUB 3800:IF FW=1 THEN
     PRINT #2," YES, ";:FVD=0:VD=1 ELSE
     PRINT #2," NO ";:XR=XR+XM
3400 PRINT #2," PATH    ":RV=3-XM:PT=8:
     GOSUB 1820:RETURN
3410 REM
3420 REM PATH 4
3430 GOSUB 3470:GOSUB 3800:IF FW=1 THEN
     PRINT #2," YES, ";:FHL=0:HL=1 ELSE
     PRINT #2," NO ";:YR=YR+1
3440 PRINT #2," PATH    ":RV=1:PT=8:GOSUB
     1820:RETURN
3450 REM
3460 REM INTENT
3470 PRINT #2,"CLEAR PATH":PRINT #2,
     "TO A WALL?":PT=8:GOSUB 1820
3480 RETURN
3490 REM
3500 REM SCAN
3510 PT=10:GOSUB 1820:PRINT #2,
     "SONAR USED":FOR DS=1 TO 4:GOSUB
     3800:NEXT:RETURN
3520 REM
3530 REM TRACK
3540 PRINT #2,"LOOK ROUND":PRINT #2,
     "THE OBJECT":PT=8:GOSUB 1820
3550 IF XG=XR+1 THEN TX(1)=XG:TY(1)=YG+1
     :TX(2)=XG+1:TY(2)=YG:TX(3)=XG:TY(3)
     =YG-1
3560 IF YG=YR-1 THEN TX(1)=XG+1:TY(1)=YG
```

```
:TX(2)=XG:TY(2)=YG-1:TX(3)=XG-1:TY(
3)=YG
3570 IF XG=XR-1 THEN TX(1)=XG:TY(1)=YG-1
:TX(2)=XG-1:TY(2)=YG:TX(3)=XG:TY(3)
=YG+1
3580 IF YG=YR+1 THEN TX(1)=XG-1:TY(1)=YG
:TX(2)=XG:TY(2)=YG+1:TX(3)=XG+1:TY(
3)=YG
3590 FOR TR=1 TO 3:PX=XR:PY=YR:XR=TX(TR)
:YR=TY(TR):GOSUB 3640:GOSUB 3510
3600 IF FS=1 THEN GOSUB 3980
3610 NEXT:RETURN
3620 REM
3630 REM PLACE
3640 G(PX,PY)=0:G(XR,YR)=82:PT=10:GOSUB
1820:SOUND 1,119,50:PRINT #2,
"CHOOSE THE":PRINT #2,"NEW VIEW ?"
3650 GOSUB 1820
3660 IF INKEY(46)=0 THEN NV=78
3670 IF INKEY(50)=0 THEN NV=82
3680 IF INKEY(58)=0 THEN NV=69
3690 IF INKEY(59)=0 THEN NV=87
3700 IF INKEY(60)=0 THEN NV=83
3710 V=NV:IF NV<>82 THEN XW=8:YW=1
3720 IF NV=78 THEN XV=8:YV=0:GOSUB 1210:
GOSUB 1850:GOSUB 1320
3730 IF NV=69 THEN XV=0:YV=8:GOSUB 1210:
GOSUB 1890:GOSUB 1320
3740 IF NV=83 THEN XV=8:YV=16:GOSUB 1210
:GOSUB 1930:GOSUB 1320
3750 IF NV=87 THEN XV=16:YV=8:GOSUB 1210
```

```
     :GOSUB 1970:GOSUB 1320
3760 IF NV=82 THEN GOSUB 2050
3770 RETURN
3780 REM
3790 REM SONAR
3800 IF DS=1 THEN S1=YR+1:S2=15:S3=XR:S4
     =XR
3810 IF DS=2 THEN S1=YR:S2=YR:S3=XR+1:S4
     =15
3820 IF DS=3 THEN S1=YR-1:S2=1:S3=XR:S4=
     XR
3830 IF DS=4 THEN S1=YR:S2=YR:S3=XR-1:S4
     =1
3840 IF DS<3 THEN S5=1 ELSE S5=-1
3850 FW=1:SB=0:PT=0.25
3860 FOR C=2 TO 0 STEP -2:FOR J=S1 TO S2
     STEP S5:FOR K=S3 TO S4 STEP S5
3870 SOUND 1,60,5:GOSUB 1820:IF G(K,J)=0
     THEN XX=K:YY=J:GOSUB 1790
3880 IF G(K,J)=83 AND SB=0 AND C=0 THEN
     XS=K:YS=J:FS=1:PRINT #2,
     "SEE SOCKET"
3890 IF G(K,J)<>0 AND G(K,J)<>83 AND C=0
     THEN FW=0:IF SB=0 THEN GOSUB 3930
3900 NEXT:NEXT:NEXT:RETURN
3910 REM
3920 REM CHECK
3930 SB=1:OF=0:PRINT #2,"OBJECT";:FOR CH
     =1 TO OC:IF K=OX(CH) AND J=OY(CH)
     THEN OF=1:EO=CH
3940 NEXT:IF OF=0 THEN OC=OC+1:OX(OC)=K:
```

```
      OY(OC)=J:PRINT #2,OC ELSE PRINT #2,
      EO
3950 RETURN
3960 REM
3970 REM RECHARGE
3980 XG=XS:YG=YS:GR=0:GD=1:RH=0:WHILE RH
      =0:PX=XR:PY=YR:GOSUB 3200:GOSUB
      3640:IF GR=1 THEN RH=1
3990 WEND:PT=10:GOSUB 1820:BORDER 20
4000 PAPER 2:X=320:FOR S=80 TO 770 STEP
      46:CLS:Y=202.9-S*0.536:GOSUB 2570
4010 GOSUB 1820:NEXT:PT=15:GOSUB 1820:
      CLS:Y=5:S=155:GOSUB 2390:PT=1.3:L=
      18:WHILE 1>0:L=L-3:IF L<3 THEN L=
      15
4020 PAPER 1:LOCATE 19,11:PRINT"O"
4030 LOCATE 22,11:PRINT"O":PAPER 2
4040 LOCATE 5,L:PRINT"PURR":SOUND 1,800,
      20:GOSUB 1820:PAPER 1:LOCATE 19,11
4050 PRINT" ":LOCATE 22,11:PRINT" "
4060 PAPER 2:LOCATE 5,L:PRINT"     "
4070 SOUND 1,1000,20:GOSUB 1820:WEND
4080 RETURN
```

Initialisation routine (Lines 1060-1100)

The 15 X 15 array which holds all the information about the position of furniture, and the robot, in the room is set up at line 1080, which also places the cursor position (XC, YC) at the centre of the room. Then various variables required for the perspective room view are initialised. The routine also establishes the robot flag, RF, the goal distance, GD, and arrays TX, TY.

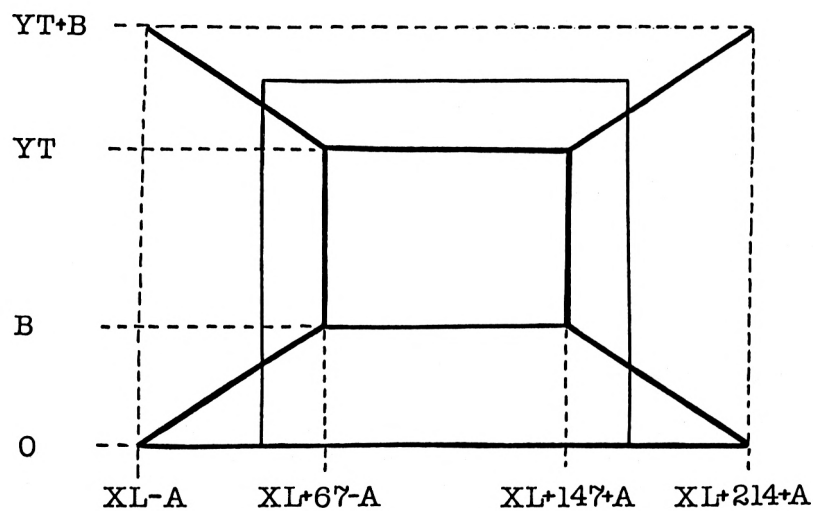
Title screen routine (Lines 1110-1190)

LUCIE begins with a screen display of the robot cat and the power socket it has to find. There are special drawing routines to create both of these. First line 1120 sets the whole screen to cyan and uses GOSUB 2390 to draw the cat, after selecting the values of X, Y and S required. The same line draws the power socket with the call, GOSUB 2570, to 'draw socket.' LOCATE allows the letter 'O' to be added to each eye and create the impression that Lucie is looking down at the socket.

Lines 1130-1160 place the title of the program on the screen and add the brief explanatory message **PUSSY WANTS POWER!** The WHILE/WEND loop that follows retains this initial screen display until INKEY(47) detects the space-bar at line 1160. It also allows the prompt, '<SPACE>', to be flashed on and off.

Room routine (Lines 1200-1230)

This routine draws a perspective view of the room. It shows the floor as cyan, the walls white and the edges of the room as black lines. Because the user can select a view from the robot's position, which obviously changes as the robot moves around the room, the drawing algorithm has to be able to produce a convincing picture of the floor and walls from any position on the 15 X 15 grid.



This variable perspective is produced by one basic room view, as shown in the diagram. However, by altering the size and relative position of this picture with respect to the edges of the screen, the impression can be given that the room is being seen from any location.

Four variables are used in drawing the room, XL, YT, A and B. Their values are calculated for a specific view from the particular viewing position, (XW, YW). This is done by line 1210. Obviously four variables are not strictly necessary, and it can be seen immediately that they are not all independent of one another. However, fewer variables would make the BASIC expressions that follow rather cumbersome. The apparently arbitrary numerical values in the equations and on the diagram follow inevitably from the 'trial and error' method used to create an artistically plausible picture of a room interior. Elegant arithmetic was required to take second place, unfortunately!

The cyan floor of the room is produced using DRAW, MOVE and FILL at 1210, 1220. Black lines are then added to the diagram by the rest of line 1220 to show the edges of the floor, walls and ceiling.

To convince himself/herself that the room algorithm works, the user should probably test it with trial values of XW, YW and with direct calls to 'room' as soon as the routine is typed in.

Display routine (Lines 1240-1300)

Here the title of the program is PRINTed and three windows created.

Grid routine (Lines 1310-1420)

Line 1320 clears window #1 to give the white background of the grid superimposed on the view of the room. The two FOR loops which follow add the vertical and horizontal black lines required. Then the nested FOR loops between 1340, 1350 check the G array and use GOSUB 1790 to indicate the room's contents on the grid. The value of V allows lines 1350-1390 to indicate the direction from which the room is being viewed. Finally, SP determines when further prompts need to be added to the display in the 'find socket' phase of the program.

Design routine (Lines 1430-1590)

Here the user can place furniture into the room. Lines 1440, 1450 place an explanatory prompt on the display to show the keys that allow the contents to be added. The WHILE/WEND loop, lines 1460-1580, then controls the

construction of the room interior. The cursor is moved over the grid by 1470-1500. Line 1510 allows the routine 'choose item' to be selected, when the space-bar is pressed, in order to add furniture or the robot to the room. Line 1520 permits the loop to end, and the search of the room to begin, when the 'B' key is pressed, provided that the RF flag indicates that the robot has been placed in the room.

Lines 1530-1560 prevent the cursor leaving the grid. The cursor position is marked by 'square' called twice at 1570 to blink the cursor on and off with a colour change, black to background white.

Choose item routine (Lines 1600-1770)

This routine lets the user place chairs, lights, sockets, tables and tv sets in the room. Lucie the Cat can be located just once. Line 1610 shows the options available. If RF = 0 and the robot still has to be located, line 1620 indicates this as well. The WHILE/WEND loop then detects the appropriate letter: C, L, S, R, T or V. Line 1700 updates the G array for any item of furniture. The numerical value placed in the array is the ASCII code of the letter. The array is updated for the robot, if RF = 0, by 1710. The same line also initialises the robot's coordinates, (XR, YR) and records its starting position, (XB, YB). This latter point is subsequently required by the searching algorithm in the routine 'find socket'. Finally, line 1710 also sets RF to 1 to prevent a second robot being accidentally added to the room's contents. Line 1720 then shows the new view of the room by calling 'room', 'view-N' and 'grid'. After this further prompts are added to the display.

Square routine (Lines 1780-1800)

The relevant square is filled in colour C on the grid.

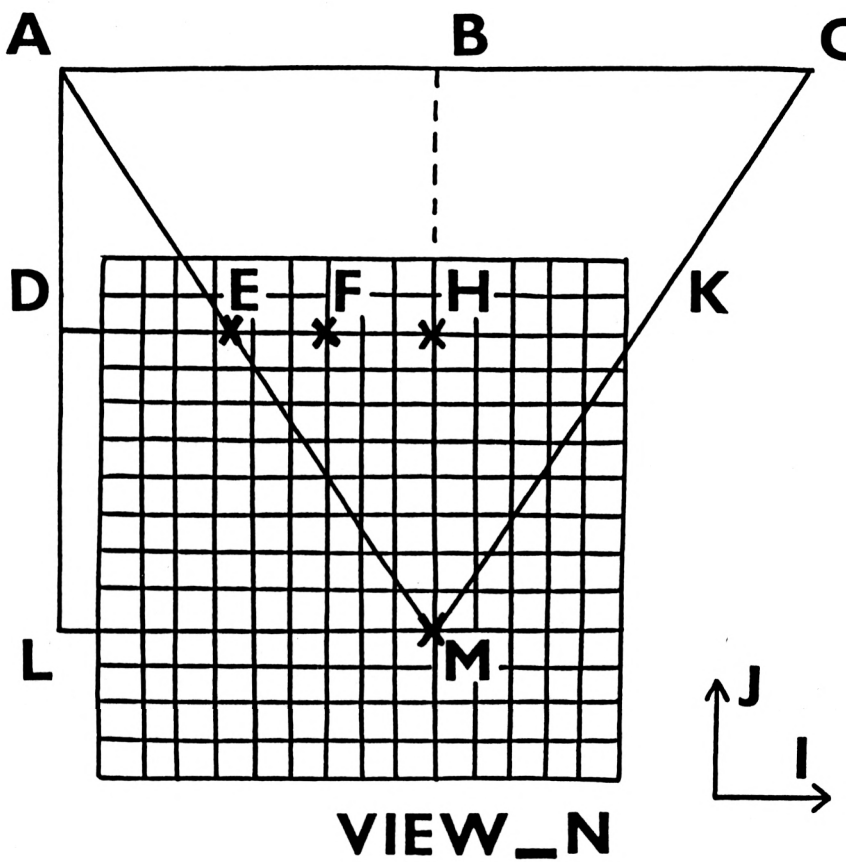
Pause routine (Lines 1810-1830)

A variable length delay is produced by the FOR loop.

View-N routine (Lines 1840-1870)

This is the first of four routines which allow the furniture in the room to be added to the perspective view which has already been created by the earlier routine 'room'. It will generate the view facing North from any of the coordinate grid positions. All furniture seen within the defined field of view is displayed, with size adjusted according to distance from the viewing location. The most distant objects are placed first so that nearer objects may obscure parts of the view in a realistic way.

The two nested loops, lines 1850, 1860, consider all relevant grid locations. The outer, J, loop begins with the 'top' row of the array and STEPs down to the row immediately in front of the viewing location, (XV, YV). The inner, I, loop looks at each column in turn, from left to right. Naturally, however, not all locations should be shown superimposed on the room but only those which fall within the field of view.



The diagram indicates the field of view from a typical viewing location on the grid, M, with coordinates (XV, YV). (It should be noted that the intersecting lines on the diagram represent the centre points of the squares on the grid.) The field of view is the triangle ACM. The base of the triangle, AC, is 20 units and the height, BM, 15 units. A typical location of a possible item of furniture is the point F, coordinates (I, J). The algorithm chosen has to be able to determine whether or not point F falls inside the field of view and thus between the lines AM and CM.

In order to do this, the distance DE has to be calculated and compared with the x-coordinate of F on the grid, value I. In the program, DE is represented by the variable W. It can be seen from the similar triangles, ADE, ALM, that:

$$DE = AD * LM/AL$$

Substituting the values already mentioned gives:

$$W = AD * 10/15$$

And it can be seen that this is the same as:

$$W = (15 - DL) * 10/15$$

However DL is the difference between the y-coordinates of the points F and M and therefore equals (J-YV). This finally yields:

$$W = (15 + YV-J) * 10/15$$

This is the expression which occurs in line 1850. The line can now use this value of W to decide whether or not F is in the field of view. The condition is simply that FH should be less than or equal to EH. Since FH = ABS (I-XV) and EH = 10-W, this leads immediately to the expression employed. The second part of the conditional is just the need for there to be an item of furniture, or the robot, at the corresponding array location, G(I, J).

If there is, the rest of the line calculates the coordinates (X, Y) needed to show the item on the screen and also the scale factor, S. X will simply be the value 640, corresponding to the width of the graphics screen, multiplied by the ratio EF/EK. It can again be seen that:

$$EF = 10-W-FH = 10-W-(XV-I)$$

This expression occurs in line 1850. Similarly EK is evidently 20 - W * 2. This also appears in the expression for X.

The values of Y and S are calculated by simple proportion. Finally, line 1850 calls the routine 'adjust' to correct the scale factor for close-up views and the routine 'item type' to choose which item to draw at the position deduced.

View-E routine (Lines 1880-1910)

The drawing algorithm employed in this routine is exactly the same as the one used in view-N, but allows a perspective view to be created looking East from the viewing location, (XV, YV) .

View-S routine (Lines 1920-1950)

Here the perspective view of the room's contents is generated looking South from (XV, YV) .

View-W routine (Lines 1960-1990)

This routine creates the view looking West.

Adjust routine (Lines 2000-2030)

In order to give large close-up views of furniture items, the scale factor, S, is increased by a square term for all values greater than 45.

Robot view routine (Lines 2040-2120)

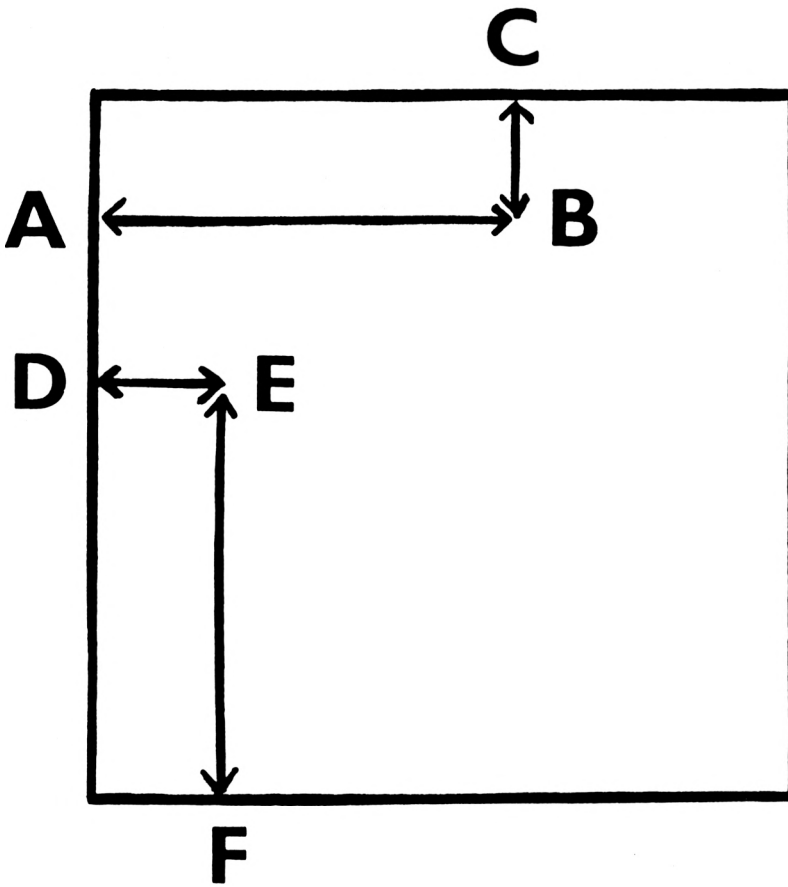
This routine is called by line 3760 if the user has selected a perspective view from the current position of Lucie the Cat.

Line 2050 makes the viewing position, (XV, YV) the same as the robot's coordinates, (XR, YR). However this is not necessarily the point which can be used in the routine 'room' to create an appropriate view of the walls and floor. The problem occurs because 'room' always assumes a North view in creating its display. As a result two separate viewing positions are usually required. (XV, YV) generates the perspective view of the room's contents and (XW, YW) the correct position for the room's edges. This can be seen from the diagram.

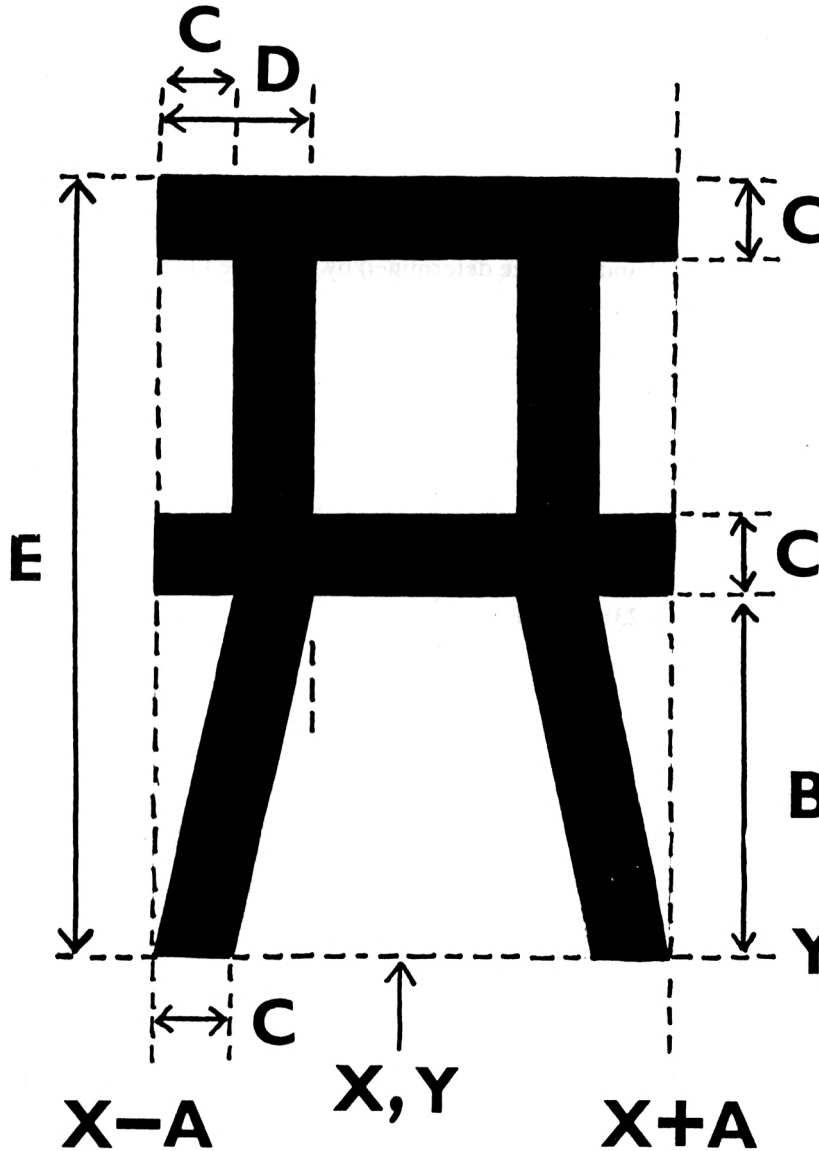
The robot is at point B in the room. However, because it is assumed to be looking East, the correct position on the grid to create a realistic view of the walls is point E, where DE = CB and EF = AB. From very simple geometry it can be seen that:

XW = DE = CB = (15 - YR)

YW = EF = AB = XR



These relationships appear in line 2070. Similar expressions can be derived for each of the other three viewing directions. The direction the robot is, in fact, 'looking' in is returned by various subsequent routines as the variable RV. It takes values 1 - 4, corresponding to compass directions N - W. Lines 2060-2090, calculate (XW, YW) according to the RV value, as well as calling the appropriate routines for showing the room. Line 2100 adds the grid and the message that it is the robot's view which is being shown.


C
D
C
E
C
B
Y
C
X, Y
X−A
X+A

Item type routine (Lines 2130-2210)

This routine is called whenever an item has to be added to the room. Lines 2140 - 2190 choose the correct drawing routine according to the array value at the given point, (I, J) .

Draw chair routine (Lines 2220-2300)

Here a chair is added to the view of the room at the coordinates already calculated, (X, Y) and of a size determined by the scale factor, S.

Line 2230 calculates five variables, A - E, directly from the value of S, to facilitate the drawing routines. The relation of these variables to the chair design can be seen on the diagram. MOVER, DRAWR and FILL are used to draw the seat and legs of the chair. Then different parts of the back, selected according to the view variable, V, are drawn by lines 2260-2280. Varying views of the chair back are needed since the chair can be seen from any of four directions. (All items of furniture added to the room are FILLed as solid shapes, although in some cases overlapping of graphics can lead to incomplete FILLing.)

Draw lamp (Lines 2310-2370)

A lamp is added to the room in a similar way by this routine.

Draw robot routine (Lines 2380-2480)

Here Lucie the Cat is added. The F loop, lines 2400-2450, alters the size of the robot as well as producing a colour change from black to yellow. This gives a 'hard edge' to the outline and so emphasises the position of the cat in the room. Line 2460 draws the cat's face if it is close enough to the viewing position. Lucie appears always to be looking at the user... This could, perhaps, be interpreted as a rotating head associated with the sonar scanner!

Fill circle routine (Lines 2490-2510)

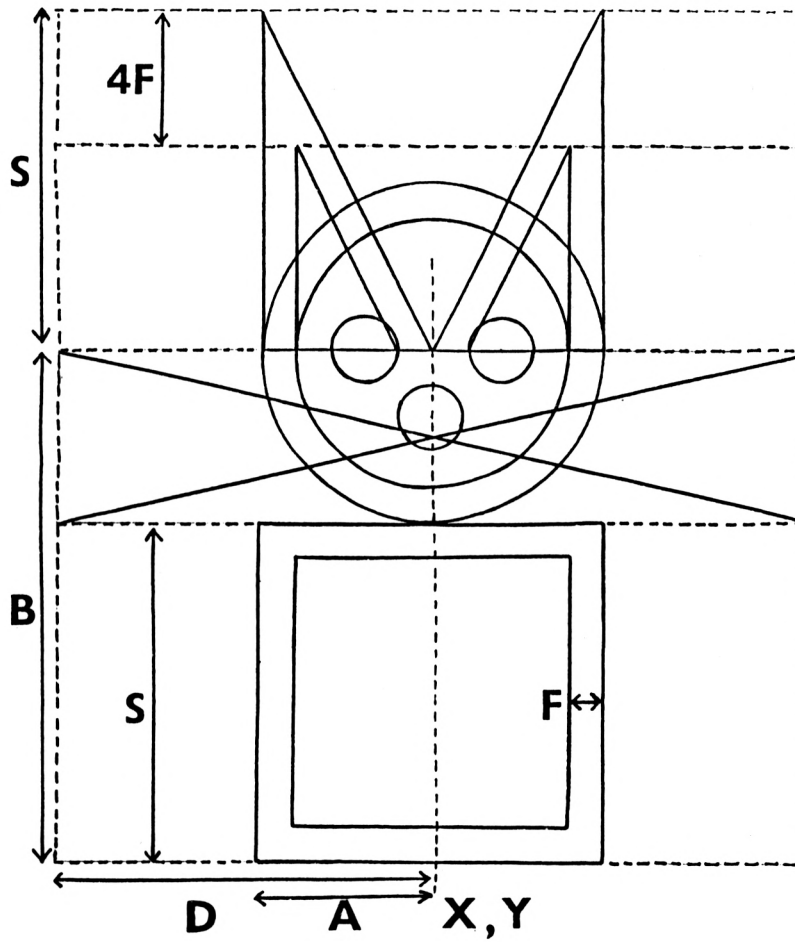
This routine is called by the draw robot routine and creates a filled circle.

Draw circle routine (Lines 2520-2550)

A circle, radius R, is drawn at the point (CX, CY) .

Draw socket routine (Lines 2560-2600)

This routine shows the robot's goal, a power socket.



Draw table routine (Lines 2610-2660)

This routine adds a table to the room.

Draw tv routine (Lines 2670-2750)

The K loop, 2680-2720, adds the screen to the tv set for all views North.

Find socket routine (Lines 2760-3000)

This is the routine which moves the robot around the room to look for the power socket. Line 2770 initialises arrays and variables needed in the searching algorithms that follow.

The initial search is carried out by the WHILE/WEND loop, lines 2780-2970. Each stage of the search is monitored by the status of various flags. For example, at line 2790 flag FVU calls 'path 1', with GOSUB 3070, to find a clear route to the North wall of the room. Line 2800 switches on, when the North wall is reached, the flag FHR which subsequenly calls 'path 2' to find a clear route to the East wall. Line 2810 switches off the VU flag when the North wall is reached. At line 2820, flag FHR calls 'path 2'. Line 2830 increments YR to move the robot one square North, depending upon the flag, VU...

In a similar way the other flags within the loop move the robot around the room according to the general algorithm outlined in the main text. The whole time, line 2950 uses the robot's sonar to look for a power socket and 2960 calls 'recharge' if one is found. However, if the robot reaches the South West corner of the room without finding a power socket, line 2940 ends the loop and the next phase of the search begins.

In this final stage of searching, the FOR loop, 2970, 2980, considers each item of furniture in the room in turn and, provided it is not at the edge of the room, line 2970 uses the routine 'peer round' to do precisely that, assuming that there must be a socket somewhere!

If line 2980 is reached, there evidently is no socket in the room and hence the final message.

Peer round routine (Lines 3010-3020)

The routine 'find socket' calls 'peer round' when it requires an object in the room to be examined. It provides the coordinates of the object, (BX, BY). The new routine then sets these coordinates as the goal coordinates (XG, YG) at lines 3020, 3030 and the WHILE/WEND loop uses the routine 'goal' to move to the new position. When it is reached, line 3040 calls 'track' to move around the object and allow the robot's sonar to check the room positions previously blocked.

Path 1 routine (Lines 3060-3100)

This routine moves the robot horizontally until its sonar detects a clear path to the North wall of the room. Line 3070 reverses the robot's horizontal

direction if the East wall is reached. Line 3080 places a message in window #2 by calling 'intent' and also calls 'sonar' to look North. A clear path is determined by the value of the flag FW returned by sonar at 3080 and if one has been found the flags needed by 'find socket', FVU and VU, are adjusted. Line 3090 calculates the robot's view, RV, in case this is required.

Path 2 routine (Lines 3110-3140)

Here a clear path to the East wall is found.

Restart routine (Lines 3150-3180)

This routine uses 'goal' to return the robot cat to its initial position in the room.

Goal routine (Lines 3190-3260) At certain stages in LUCIE it is necessary to move the robot to a particular position that can be specified in advance, a goal point (XG, YG). This occurs, for example, in the routine 'restart'. The routine 'goal' handles such cases. It modifies the robot's coordinates (XR, YR) in such a way that the goal location is reached as quickly as possible.

Line 3200 calculates XD, the increment needed to bring the x-coordinate nearer to the goal. This will be zero if the goal and the robot already have the same x-coordinate, or either 1 or -1, as determined by the expression involving ABS (XG - XR) . In a similar way, line 3210 calculates YD.

Line 3220 uses XD, YD, to calculate the provisional new point, (XN, YN). Then it compares the distance of this point from the goal with the goal distance, GD, and sets the goal reached flag, GR, to 1 if necessary. If, however, the goal has not been reached, line 3230 moves the robot's coordinates to the new point if the grid array indicates that space is vacant or calls the routine 'jump' if the space is occupied and avoiding action needs to be taken.

Line 3240 calculates the robot view variable, RV, from the increments XD, YD. The reasonable assumption is made that the robot is facing in the direction of greatest distance from the goal.

Jump routine (Lines 3270-3320)

This routine is called by 'goal' when there is an obstacle. It simply makes a random jump of one square in the hope that a better path can be selected from the new position. It deals far more easily with single obstacles than, for example, long lines of chairs stretching across the room!

Line 3280 stores the initial robot position as (XA, YA). The WHILE/ WEND loop then produces a random jump of one square in either the x- or y-direction. Line 3280 will alter the x-coordinate one way or the other, or, perhaps, not at all. Only in the latter case will it attempt to alter the y-coordinate. This means that the point can only ever be moved in one coordinate direction at one pass through the loop, or possibly left unaltered. But if this happens, or if the point selected is not vacant, the loop will not end at 3310. Inevitably this algorithm will find a suitable new position.

Random routine (Lines 3330-3360)

A random integer, -1, 0 or 1, is generated for the jump routine.

Path 3 routine (Lines 3370-3410)

Here a clear path to the South wall is found.

Path 4 routine (Lines 3420-3450)

This routine locates a clear path to the West wall.

Intent routine (Line 3460-3490)

The user is informed, by a message in window #2, that Lucie is trying to find an unblocked path to one one of the walls.

Scan routine (Lines 3500-3520)

The FOR loop directs the sonar beam in each of the four possible directions by calling 'sonar' with GOSUB 3800.

Track routine (Lines 3530-3620)

The routine 'peer round' calls 'track' when the robot cat has reached one of the items of furniture earlier located by the sonar beam. The four conditionals between 3550 - 3580 decide the position of the robot relative to the item and place the coordinates of the other three locations immediately adjacent to the item into the coordinate arrays TX, TY. After this the FOR loop, 3590-3610, moves Lucie to each of the locations in turn so that the sonar beam can investigate the previously blocked direction.

Place routine (Lines 3630-3780)

Line 3640 updates the robot's position in the G array and invites the user to select a new view. The choice is then detected by lines 3660 - 3700 and the

new display created by lines 3720 - 3760, with appropriate calls to 'room', the view routines and 'grid'.

Sonar routine (Lines 3790-3910)

This is the important routine which shows the robot's sonar on the grid, 'remembers' all objects encountered in the room and identifies a power socket.

Lines 3800 - 3840 calculate the ranges of the nested FOR loops, depending upon the robot's location and the direction, DS, in which the sonar has to be directed.

The beam is produced by the three loops between lines 3860-3900. The C loop governs a colour change from cyan to white, so that the beam is erased from the grid. The J and K loops control the beam in the y- and x- directions. Line 3870 produces a tone and uses GOSUB 1790 to indicate the relevant square of the grid through which the beam is passing. Objects in its path are not accidentally erased.

If a socket is located, and the sonar-blocked flag, SB, is off (to prevent the sonar seeing through objects) and it is the second pass through the C loop, line 3880 switches on the found-socket flag, FS, and stores the socket location as (XS, YS).

Line 3890 will locate other items of furniture, again only on the second pass through the C loop (to avoid counting twice). The found-wall flag, FW, is switched off and, if the sonar-blocked flag is off, the routine 'check' is called.

Check routine (Lines 3920-3960)

When 'sonar' detects a furniture item, this routine uses the FOR loop, 3930, 3940, to check its location against the coordinates of known items in the OX, OY, arrays. If the flag OF indicates that it is a fresh item, line 3940 adds it to the arrays. Line 3930 switches on the flag SB to prevent the sonar registering subsequent objects in the same direction, and seeing through this object.

Recharge routine (Lines 3970-4080)

This is the routine which is called when the robot eventually finds the power socket it is seeking. Line 3980 sets the socket location, (XS, YS), returned by 'sonar' as the goal location, coordinates (XG, YG). The WHILE/WEND loop then uses 'goal' and 'place' to show the robot cat approaching the socket.

After this the FOR loop at 4000, 4010, uses GOSUB 2570 to produce a 'zoom' effect on the socket. Then the cat is drawn on the screen with flashing eyes and a purr created by the WHILE/WEND loop between lines 4010-4070.

CHAPTER 9

Machine mentality: philosophical issues.

The ultimate question which will face Artificial Intelligence will be the inherent status of the machines that it produces. Will they, in any sense, be 'persons'? It is possible to imagine very advanced robots sent on voyages of exploration to distant planets and into conditions which would be quite intolerable for human life. The environment might well be equally inimical for robots. The project will probably accept their loss. But if these future machines are to conduct the type of research necessary, they will have undoubtedly have been created with human-like skils of reasoning and linguistic ability. Back on Earth, project coordinators may feel just a little queasy when the final message comes through. What will the moral standing of such robots be? Could it conceivably be murder to allow such a creation to be extinguished? Similarly, powerful domestic machines around the home might also pose problems. Suppose an eccentric bachelor makes his robot valet also his beneficiary. What would the position of the intelligent machine be in law?

Such issues have been a central theme of science fiction for many years. Now, with the new intellectual status granted to them by serious research in A.I., they are beginning to emerge as serious philosophical questions.

The 'Strong' AI Position

Some A.I. workers maintain a position which declares almost full equivalence between intelligent programs and people. John McCarthy stated that simple thermostats could have beliefs about the world, for example 'It is the right temperature now'. Herbert Simon and Alan Newell claim that intelligence is merely a matter of symbol manipulation and can take place as well in an electronic machine as in the human mind. Freeman Dyson gives the advantage to the machine, because of its robust construction, and sees computers as the next stage of evolution. Marvin Minsky believes that computers will become so intelligent that people will just have to rely upon being looked after as pets.

The Turing Test

A more pragmatic attitude was displayed by Alan Turing when he suggested a definite test of whether a machine was truly intelligent or not. Turing felt that if it proved impossible to tell whether or not the 'person' at the other end of a communications link was a human being or a computer, then it made little sense to say that the computer was not intelligent. The Turing Test appears to have the reassuring nature of a scientic experiment and therefore open to the usual criteria used for judging all science. However it becomes less plausible when confronted with Joseph Weizenbaum's psychoanalysis program ELIZA. Despite the almost fraudulent simplicity of this program, which simulated understanding of typed dialogue by looking for key phrases and by being cunningly evasive the rest of the time, people frequently reacted as if they were really addressing another person.

The Case against 'Strong' AI

The linguistic philosopher John Searle has devised what many consider to be a refutation of the more extreme claims of A.I. In it, he imagined a person locked in a room but engaging in dialogue with Chinese speaking people outside. This could be done with printed Chinese symbols which could be passed to and fro. Searle argued that, provided the person had been given a sufficiently detailed rule book written in his own language, English, enabling him to exchange the correct symbols, he could appear to be communicating perfectly well. He would, however, understand no Chinese at all. Searle reasons that this is precisely the same situation which would arise in A.I. Even with the most sophisticated programs designed to 'understand' natural language, the computer would be comunicating without understanding. According to Searle, there is a gulf which cannot be bridged between the syntactic understanding of language at a superficial level and true semantic understanding. The latter can only exist in the language-using domain of human beings, full of richness and context.

Searle's position is also adopted by the philosopher Hubert Dreyfus. He challenges the idea that the whole of reality as we understand it can be turned into a formal system. This, inevitably, is the implication behind the more excessive claims of A.I. Instead, Dreyfus emphasises the wealth of perceptions, emotions and intuitions that each of us experiences as a person in the world. Dreyfus feels that Western culture since the time of Plato has implicitly assumed the division of mind from the world and that this is an error now carried over into A.I. The belief that the mind is an abstract entity and totally separate from the things around it gives the A.I. researcher an unfounded confidence in the possibility of codifying it into a formal system.

Social Beliefs and Expectations

Terry Winograd is one A.I. worker who is also moving towards this position. After the failure of the microworld approach, he now sees language as a far broader, social activity, with a network of beliefs and expectations stretching out from the individual to encompass the surrounding society. In a positive way he regards this as a challenge for the development of further ideas in A.I. at the new Centre for the Study of Language and Information at Stanford. There is a curious similarity between his change of direction and that of the philosopher Ludwig Wittgenstein earlier this century, who also radically altered his perception of language from a precise logical calculus to an essentially social act. Artificial Intelligence necessarily brings together many seemingly diverse disciplines. It is this which gives the activity its undeniable appeal.

CHAPTER 10

An A.I. crib sheet: concepts, names and programs.

ALTY Jim: Professor at the University of Strathclyde and associated with the Turing Institute; coauthor of 'Expert Systems: Concepts and Examples'. (NCC Publications, 1984)

Alvey Report: 'A Programme for Advanced Information Technology', the report prepared for the Department of Industry by the Alvey Committee.

Accountability: The principle that A.I. programs should be able to give reasons for the decisions they make.

Alpha-beta algorithm: An improved heuristic derived from minimax which ignores unpromising branches during a tree search.

Automatic Mathematician (AM): This was an early version of Lenat's EURISKO which deduced for itself, then investigated, basic concepts in number theory.

B-star algorithm: A heuristic, devised by Hans Berliner, which evaluates two variables at each node of a tree structure, for best and worst possible result.

Backward chaining: A strategy employed by the inference engine of an expert system in which deduction proceeds from a final hypothesis to the initial evidence implying that hypothesis.

Belle: A chess playing program developed by Thompson and Condon at AT&T Bell.

BERLINER Hans: Professor of Computer Science at Carnegie-Mellon University; author of Mighty Bee and B-star algorithm.

Blackboard: A method of knowledge representation first applied in speech recognition; information is stored in separate modules which can summon the contents of other modules when necessary.

Blind search: Seeking a solution to a given problem by considering all routes through the tree structure without the aid of any simplifying heuristic to reduce their number.

Blocks World: A microworld devised by Terry Winograd for his natural language program SHRDLU.

BODEN Margaret: Reader in Philosophy and Psychology at the University of Sussex, concerned with philosophical and social consequences of A.I., especially the 'rights' of sentient machines; author of 'Artificial Intelligence and Natural Man'. (Harvester Press, 1977)

Boltzmann architecture: A hypothetical computer design involving parallel processing and intended to simulate the functioning of the human brain; a large number of separate processors placed initially in a random state 'coalesce' to create a final state in a way which copies the brain's power of intuition.

Bottom-up: A programming technique in which the fine details are investigated before the overall structure of the program.

Branching factor: A measure of the increasing complexity of a tree structure at each node.

CAPEC Karel: Czech playwright who invented the term 'robot'.

Chinese room: A thought experiment devised by John Searle to support his belief that programs can never truly understand language.

CHOMSKY Noam: Professor of Linguistics at the Massachusetts Institute of Technology; leading figure in the subject whose work on the structure of language has influenced research in A.I.

CLOWES Max (late): A.I. research worker at the University of Sussex who, together with David Huffman of M.I.T., produced a system of classification of the sixteen possible vertices of a two dimensional diagram which allowed the recognition of three dimensional shapes.

Cognitive science: The inter-disciplinary study of A.I., linguistics and psychology.

COLBY Kenneth: A psychoanalyst who turned to A.I.; author of PARRY.

Combinatorial explosion: The rapid growth of a tree structure.

Common sense: The human ability, and current computer inability, to cope with the complex, though apparently trivial, activities of everyday life.

Computer creativity: The ability of a computer to generate seemingly original ideas.

Computer learning: Techniques enabling a computer to acquire fresh knowledge and skills not initially implicit within its program.

Computer vision: The development of specialised hardware, and associated software, intended to give computers limited sight.

Cray Blitz: A championship level chess program written for the Cray computer.

Demons: Routine within a program intended to identify, and act upon, some particular situation.

DENDRAL: An expert system which deduces chemical structure.

DENNETT Daniel: Professor of Philosophy at Tuffs University and important figure in A.I.; author of 'Brainstorms: Philosophical Essays on Mind and Psychology' (Harvester Press, 1978) and coauthor, with Douglas Hofstadter, of 'The Mind's I' (Harvester Press, 1981)

Depth search: Seeking the solution to a problem by resolving it into increasingly simplified, though more numerous, sub-problems.

Domestic robot: A hypothetical computer controlled machine of the future which uses a high level of A.I. to perform household tasks.

DREYFUS Hubert: A philosopher at the University of California at Berkeley who maintains that computers will never be able to 'think'; author of 'What Computers Can't Do: The Limits of Artificial Intelligence'.

Edge detection: The ability of a computer vision system to locate boundaries within its field of view; also called 'profile analysis'.

ELIZA: A program written in 1966 at M.I.T. by Joseph Weizenbaum and capable of simulating apparently intelligent dialogue; the program adopts the role of a psychiatrist counselling a patient.

EURISKO: A program developed by Douglas Lenat which, by adjusting its own heuristics, has shown 'creative' ability in a number of areas of application.

Evaluation function: A calculation performed at each node during a tree search to determine the best route.

Exhaustive search: See blind search.

Expert systems: The realisation in a computer program of some area of human expertise which allows the computer to make decisions and offer advice.

Feedback: Providing a program with results of its decisions to allow it to modify and optimise its performance.

FEIGENBAUM Edward: Head of team at Stanford University which began the development of expert systems with DENDRAL.

Fifth generation: The proposed next stage of computer development, particularly associated with Japan and officially launched in October 1981, involving sophisticated A.I. in the design of new machines and software; fifth generation computers are intended to communicate in natural language, possess a large knowledge base and efficient inference engine and to be able to derive new heuristics from their own experience.

FODOR Jerry: Professor of Psycholinguistics and Philosophy at M.I.T.; author of 'The Language of Thought'. (Harvester Press, 1976.)

Forward chaining: A strategy employed by the inference engine of an expert system which reasons from initial evidence, through a series of hypotheses, to a final conclusion.

Frames: A method of knowledge representation, proposed by Marvin Minsky in 1974 which allows analogies to be drawn between different concepts.

Freddy: Early robot project at Edinburgh University.

FRUMP: 'Fast Reading and Understanding Memory Program', developed at Yale, which uses scripts to analyse newspaper articles.

Functionalism: A recently developed philosophy of mind which has emerged from work in cognitive science.

Fuzzy matching: Identification by matching only the most salient details of a pattern.

Game-playing: The practice traditional in A.I. of testing heuristic rules and related concepts in typically 'intellectual' games like chess.

General Problem Solver: A program written by Newell and Simon capable of solving problems in certain restricted microworlds.

Goal directed: Term describing a strategy intended to achieve a given goal.

Goal state: A specific objective for a program.

Heuristic: A practical 'rule of thumb' which uses knowledge, rather than logic alone, to make a decision.

Heuristic search: A tree search guided by heuristic rules.

Heuristic pruning: Ignoring branches of a tree structure according to a selected heuristic.

HOFSTADTER Douglas: Assistant Professor of Computer Science at Indiana University; author of 'Godel, Escher,Bach: an Eternal Golden Braid' (Penguin Books, 1980) and coauthor of 'The Mind's I'

Homunculus: A (whimsically) imagined 'inner man' proposed in traditional philosophy to account for perception and understanding; a similar concept passes over to A.I. in its division of a problem into sets of autonomous subsystems.

Horizon effect: The necessity of adjusting look-ahead to a sufficient depth of ply, according to the current situation, in order to detect important decisions in advance.

HUFFMAN David: See Max Clowes.

Image processing: The conversion of information from an optical device into a form capable of manipulation by a computer.

Inference engine: The analytical/deductive component of an expert system as opposed to its knowledge base.

Internal representation: See knowledge representation.

JULESZ Bela: Research worker at AT&T Bell who has investigated the brain's ability in stereopsis, obtaining in particular depth-information from randomly generated dot patterns.

Knowledge base: Information stored within an expert system.

Knowledge engineering: Term used widely to describe the operation and function of an expert system.

Knowledge representation: The specific method of storing the data used by a program; different methods are more or less appropriate for a given program.

LEHNERT Wendy: Assistant Professor of Computer Science at Yale University who has employed scripts in natural language programs.

LENAT Douglas: Assistant Professor of Computer Science at Stanford University who developed the program EURISKO.

Lighthill Report: Sir James Lighthill's report to the Science Research Council in 1972 concerning a grant application from the Edinburgh University A.I. research group; a critical appraisal of the potential of A.I. research which led to a lack of funding in Britain and even America.

LISP: The LISt Processing language used extensively in A.I.

Logic Theorist: Program written in 1950s by Newell, Simon and Shaw and designed to prove theorems in mathematical logic.

LONGUET-HIGGINS Christopher: A mathematician who worked on A.I. with Donald Michie at Edinburgh University and who later moved to the A.I. group at Sussex.

Look-ahead: Initial production of a tree structure in a problem-solving program.

MACSYMA: Early expert system developed at M.I.T. by Joel Moses.

McCARTHY John: Professor of Computer Science and Director of A.I. Laboratory at Stanford University; original author of LISP and inventor of term 'Artificial Intelligence'.

MICHIE Donald: Mathematician who founded the early Department of Machine Intelligence and perception at Edinburgh University, now working at the Turing Institute, Glasgow.

Micromouse: A small computer controlled vehicle used to test problem solving techniques by exploring a maze; micromouse competitions are now part of the tradition of A.I.

Microworld: A mathematical model of a minute portion of the real world so reduced in complexity that a program can converse about it in natural language.

Mighty Bee: Hans Berliner's backgammon program which defeated the world champion in 1979.

Mind-Body problem: A traditional issue in philosophy which explores the relationship between mind and the brain; A.I. may eventually help to clarify some of the arguments involved.

Minimax algorithm: A heuristic, developed by Claude Shannon, designed to reduce the complexity of a tree search by assuming that the computer's opponent will always select the best possible move.

MINSKI Marvin: Leading M.I.T. figure and proponent of the 'strong' A.I. position; invented concept of frames.

Modular programming: A technique of writing programs in discrete units, each with a distinct function within the overall structure.

MOSES Joel: M.I.T. researcher who developed MACSYMA.

MYCIN: An expert system, developed by Edward Shortliffe of Stanford University, which diagnoses blood infections.

Natural language: A term used to distinguish between ordinary human discourse and computer languages; a major objective of A.I. is to narrow the gap between the two.

NAGEL Thomas: Professor of Philosophy at Princeton University and author of 'What is it like to be a bat ?' (Mortal Questions, C.U.P.)

NEWELL Allen: Researcher at Carnegie-Mellon University who helped to develop Logic Theorist and General Problem Solver.

Nodes: Points of divergence in a tree structure and where decisions, usually governed by a heuristic rule, have to be made.

PAPERT Seymour: Colleague of Marvin Minski and author of the programming language LOGO.

Parallelism: The application of parallel processing in A.I.

PARRY: Program, written by Kenneth Colby, which simulates a neurotic person.

Parsing: The grammatical technique of analysing a sentence into component words and phrases; it has a particular significance for programs intended to understand natural language.

Pattern recognition: A fundamental problem in computer vision involving the extraction of significant features during image processing.

PEARL Judea: Author of the Scout algorithm.

PLANNER: A computer language used in A.I. and employed by Terry Winograd in SHRDLU.

Ply: A given level of a tree structure, counted from present position.

POGGIO Tomasso: Associate Professor at Massachusetts Institute of Technology; following research at the Max Planck Institute on the visual system of the fly, he began working on computational problems in computer vision.

Portsmouth Polytechnic: On a nationalistic note... the location of John Billingsley's robot ping-pong competition!

Production rules: A conditional method of storing information; eg: 'If the gas goes out then my bath will be cold'.

PROLOG: A computer language used in A.I. in which the knowledge representation is based upon predicate calculus.

PROSPECTOR: An expert system which locates mineral deposits.

PUFF: An expert system which diagnoses breathing disorders.

PUTNAM Hilary: Professor of Philosophy at Harvard University; author of 'The mental life of some machines' (Mind, Language and Reality, C.U.P.)

RAIBERT Marc: Assistant Professor of Computer Science and Robotics at Carnegie-Mellon University who has worked on control systems required by walking machinery and specifically upon a hopping leg.

Recursion: Programming technique, frequently employed in A.I. programs, in which a routine calls itself.

Robotics: The gradual development of autonomous computer controlled machines capable of many physical and intellectual tasks usually supposed to require direct human intervention.

SAM: 'Script Applier Mechanism', a Yale program which uses scripts to interpret stories presented to it.

SCHANK Roger: Professor of Computer Science and Director of A.I. Project at Yale University who devised the concept of scripts.

Scout algorithm: A game tree heuristic developed by Judea Pearl.

Scripts: A development of frames by Roger Schank and other Yale researchers in which context and inferences allow a program to answer questions about simple stories and provide details implicit, thought not directly explicit, in the text.

SEARLE John: Professor of Philosophy at the University of California, Berkeley; a leading linguistic philosopher who has turned to issues raised by A.I.; author of 'Minds, Brains and Programs' (Reprinted in Hofstadter/ Dennett 'The Mind's I')

Semantics: The intrinsic meaning of sentences as opposed to their formal grammatical structure; semantic meaning is notoriously difficult to code into natural language programs.

Shakey: A research robot at Stanford controlled via radio by a PDP-10 computer and used to investigate practical issues in vision, navigation and problem solving; experience gained in this project was later applied to the design of NASA's Viking lander on Mars.

SHANNON Claude: Author of the minimax algorithm, developed at AT&T Bell.

SHRDLU: Terry Winograd's very successful natural language program based on 'Blocks World'; name deliberately meaningless and not an acronym, but instead originating in a group of random letters inserted by typesetters into a manuscript to indicate a mistake.

SIMON Herbert: Leading A.I. figure at Carnegie-Mellon University who helped to develop Logic Theorist and General Problem Solver.

Slots: Term used for positions in a frame which can be filled with information.

Speech acts: A analytical approach to language first employed by the British philosopher John Austin (1911 - 1960) in which utterances are seen to be performing specific roles, statements, commitments, requests etc.; speech act theory is now entering A.I. via 'coordinator systems' which identify the specific speech acts implicit in text undergoing analysis.

Speech recognition: Computer identification of spoken language; an initial stage in natural language processing for the 'fifth generation' machines.

'Strong' A.I.: A term used by John Searle to refer to the belief of some A.I. researchers that the relationship between a computer program and the computer is closely analogous to that between the human mind and the brain.

Structured selection: A 'diagnostic' approach to hypotheses and evidence adopted in expert systems.

Syntax: The grammatical structure of a language which needs to be analysed by a natural language program before semantic meaning can be investigated.

TALESPIN: Program, written by Jim Meehan, which explores the interaction of the goals assigned to characters with events occurring in a simulated world in order to generate simple stories.

Top-down: A technique in which the structure of a program is developed before the fine details of programming are investigated.

Tree search: Examining the tree structure to determine the best possible solution for a given problem.

Tree structure: A descriptive name used to indicate the way the total number of possible decisions increases geometrically at each stage of a problem solving exercise.

TURING Alan: British mathematician responsible for much early theoretical work in the development of computers and also A.I.

Turing Institute: Organisation established by Donald Michie and others in 1983 at Glasgow in order to encourage research and industrial contribution to A.I.

Turing test: A thought experiment described by Alan Turing in which a program would be deemed intelligent if capable of dialogue indistinguishable from human.

Ultra intelligent machines: Computers more intelligent than people, the final objective of A.I. according to adherents of its 'strong' form; term invented by Jack Good, Professor of Statistics at Virginia Polytechnic Institute.

WEIZENBAUM Joseph: Author of ELIZA and also 'Computer Power and Human Reason'. (Penguin Books, 1984)

WINOGRAD Terry: Important figure in A.I. who, while at the Massachusetts Institute of Technology, developed the natural language program SHRDLU; author of 'Understanding Natural Language'. (Edinburgh University Press, 1976)

WINSTON Patrick: M.I.T. researcher who developed a program capable of generalising concepts from specific examples; author of 'Artificial Intelligence'. (Addison-Wesley, 1977)

WITTGENSTEIN Ludwig: The extremely influential twentieth century philosopher whose view of language evolved from a formal system, described in his 'Tractatus Logico-Philosophicus', to one which appreciated its essentially social nature, expressed in the 'Philosophical Investigations'; an analogous development seems to be occurring in natural language programs in A.I.

INDEX

AMSTRADS AND ARTIFICIAL INTELLIGENCE

About this book

This is intended as a companion to another of our highly successful books on artificial intelligence *Build Your Own Expert System*. In this new book, Patrick Hall surveys the broad field of contemporary artificial intelligence- the underlying theories, practical applications in industry and commerce, and working examples that are ready to run on Amstrad CPC series computers (and, with minor changes, on most popular micros).

Topics covered include:

Creative writing of prose and verse
Natural language translation
Knowledge representation
Expert systems and knowledge engineering
Game playing
Computer vision
Robots and learning

The programs are substantial productions- in fact, many of them are based on major landmarks in AI research. For example, there is a re-creation of 'Shakey'- the robot developed at Stamford Research Institute, though in this book Shakey is a robotic cat on the screen of an Amstrad! Each program is carefully described and documented, so that users can understand exactly *how* and *why* it works.

About the author

Patrick Hall is an established Sigma author with a special interest in simulations, artificial intelligence, and computing. He is currently working on a major new LISP programming book for microcomputer users.

About us

Sigma Press has a long-term commitment to excellence in personal computing, science, and technology. Intending authors should contact:

Sigma Press, 98a Water Lane,
Wilmslow, Cheshire SK9 5BB

A comprehensive catalogue is available on request.

GB £ NET +008.95

ISBN 1-85058-038-3

00895

9 781850 580386

SIGMA PRESS

AMSTRADS and Artificial Intelligence

Patrick Hall

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/