Valentine

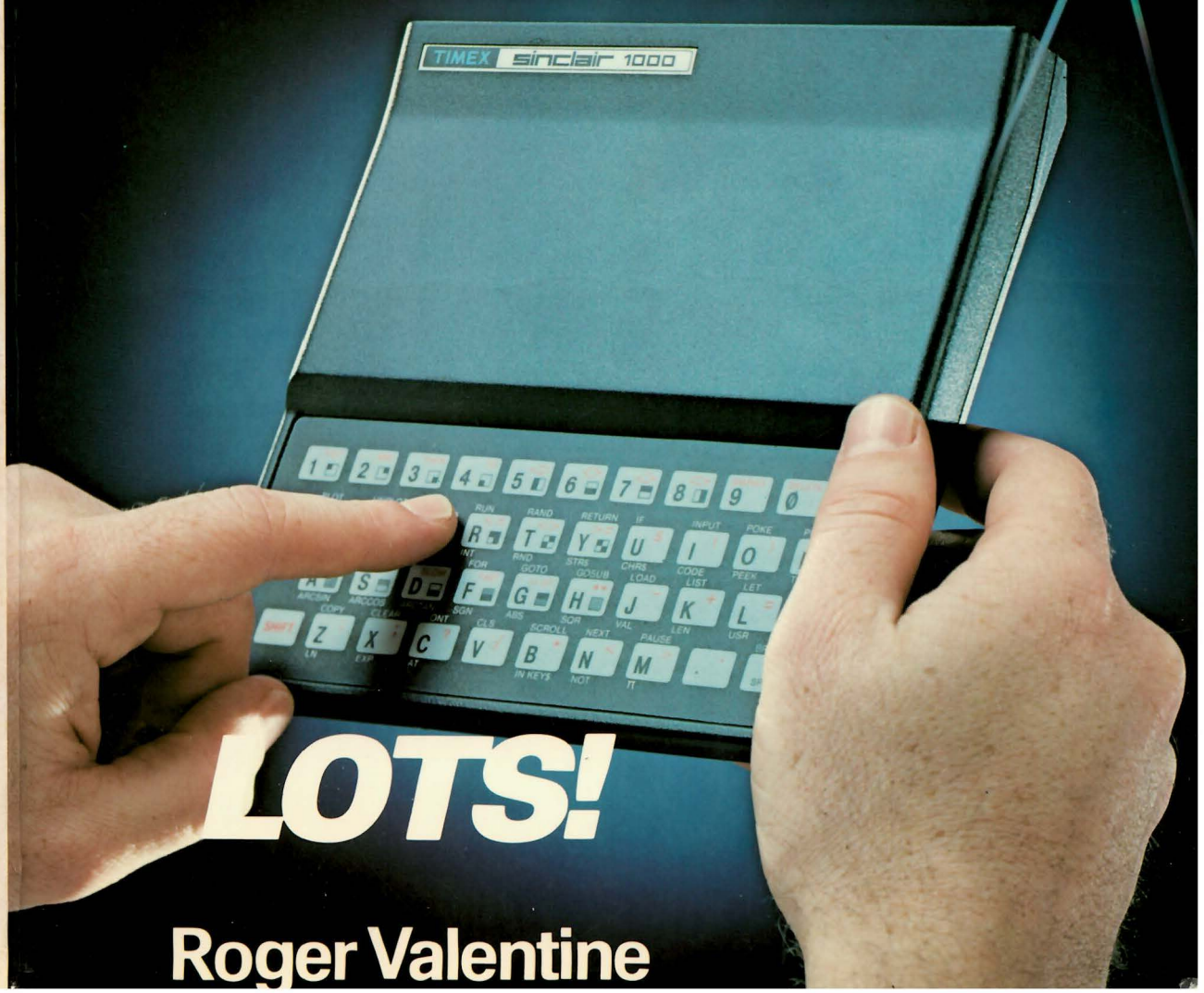
WHAT CAN I DO WITH MY TIMEX SINCLAIR 1000? *LOTS!*

Wiley Press

56 Programs for the Timex Sinclair 1000 and ZX81

WHAT CAN I DO WITH MY TIMEX sinclair 1000 ?

Programs available on cassette!



LOTS!

Roger Valentine

WHAT CAN I DO WITH MY TIMEX SINCLAIR 1000? Lots!

56 Programs for the Timex Sinclair 1000 and ZX81

Roger Valentine

A Wiley Press Book
JOHN WILEY & SONS, Inc.
New York • Chichester • Brisbane • Toronto • Singapore

Library of Congress Cataloging in Publication Data

Valentine, Roger, 1949-
What can I do with the Timex Sinclair 1000?

"Originally published in England in two volumes: What can I do with the 1K and What can I do with the 16K"—Pref.
Includes index.
1. Timex 1000 (Computer)—Programming. 2. Sinclair ZX81 (Computer)—Programming. 3. Basic (Computer program language) I. Title.
QA76.8.T48V34 1983 001.64'25 83-1339
ISBN 0-471-88730-7

Contents

Preface

This book is written to provide you with some sensible programs for your Timex 1000 or Sinclair ZX81. Many programs for serious use of the Timex 1000 are given in the book. They include, Banking Records, Credit Cards, and Data Files. For fun there are Chapters on Games for 1K, Games for 16K and Casino Gambling.

This book was originally published in England in two volumes: WHAT CAN I DO WITH THE 1K and WHAT CAN I DO WITH THE 16K. These two volumes correspond to Parts I and II in terms of what is required to run the programs. I have modified programs specifically tied to English systems to be in line with American standards. For example coin search is in dollars and cents not pounds. In any event I hope using these programs encourages you to write more on your own and get the most from your Timex 1000.

Introduction To Part I

Part 1 of this book was originally published in England under the title What Can I Do With 1K? The programs will all run on either a 1K Sinclair ZX81 or a 2K Timex 1000.

I was tempted to expand some of the programs slightly to make full use of the additional RAM provided on the Timex 1000, but as all the programs illustrate memory-saving techniques that are just as valid for 2K as they are for 1K, I eventually decided to leave them in their original form.

Most of the programs have survived the Atlantic crossing intact, but the 'Business' section has suffered rather severely. The program 'SALES TAX' was originally designed to deal with a peculiarly European tax called V.A.T. It is still appropriate for state taxes, but you will have to amend the first line so that it contains the correct rate of tax for your particular state. (I understand that some states do not have a sales tax at all — O.K., so the program is no use at all for you, but you get the book cheaper than everybody else!)

Two programs which deal with the complexities of the British income tax system have been dropped completely. To make up for this, I have added an extra 'GAMES' section at the end. Four games in exchange for 2 business programs cannot be such a bad deal!

Roger Valentine
London Nov. 1982

Listing Conventions

The following conventions have been used throughout Part I of this book. Please study them carefully before keying in the programs.

l = The letter l
1 = The number one
O = The letter O
Ø = The number zero
' = The image quote (SHIFT Q)
← = Space (where essential)
↑ = All items between 2 up-arrows are to be entered in GRAPHICS
UNDERLINING = All underlined items are to be entered in INVERSE

Note: These conventions have NOT been followed in Line 1Ø of DATA PRINT, or in DICE.
The conventions for DATA PRINT have been noted in the text.
For DICE, the A graphic square has been used.
(USE Graphic A, H or Space, as you wish.)

EXAMPLES

```
10 PRINT "↑T↑'FIRST'←EXAMPLE↑Y↑"
```

Enter the string as:

LISTING	KEYSTROKE
↑	SHIFT 9 (graphics ON)
T	SHIFT T (graphic quarter square)
↑	SHIFT 9 (graphics OFF)
'	SHIFT Q
F	F
I	I
R	R
S	S
T	T
'	SHIFT Q
←	SPACE
E	SHIFT 9 (inverse ON)
X	E
A	X
M	A
P	M
L	P
E	L
↑	E
	NO KEYSTROKE ! actually:
	SHIFT 9 (inverse OFF)
	SHIFT 9 (graphics ON)
Y	SHIFT Y (graphic quarter square)
↑	SHIFT 9 (graphics OFF)

Also:

```
10 PRINT "        "
```

is not very helpful, so would be shown as:

```
10 PRINT "←←←←←←←←"
```

Chapter 1
Fate and Fortune

No apologies for starting with this section. We know that logical-minded computer people are not supposed to be 'into' this kind of thing, but WE are, and we suspect that secretly you may be too.

Most of these programs provide data only. You must consult a book on the relevant subject for an interpretation of that data.

Numerology

This program converts any name into a single-digit number, according to the rule A = 1, B = 2, Z = 26 = 2 + 6 = 8, etc. The number produced can be taken as a 'lucky number,' but is supposed to be indicative of your personality type. Technically you should use the name given to you at birth, but if you also try your nickname, or the name by which you are usually known, you can see how you have subconsciously attempted to alter your personality over the years.

I Ching

Just run this program to come up with your hexagram for the day. For demonstration purposes, you can continue to produce new hexagrams indefinitely by pressing NEWLINE. However, serious users should run the program just once a day, so delete the last 4 lines if you do not want the repeat facility.

NUMEROLOGY

```
5 PRINT "ENTER YOUR NAME"
10 LET N=0
15 INPUT N$
20 CLS
25 IF N$=" STOP " THEN STOP
30 PRINT AT 3,0;"NAME"
35 PRINT AT 5,0;N$
40 FOR J=1 TO LEN N$
45 LET C=CODE N$(J)-37
50 IF C<1 OR C>26 THEN GOTO 80
55 IF C<10 THEN GOTO 70
60 LET C=C-9
65 GOTO 55
70 LET N=N+C
75 IF N>9 THEN LET N=N-9
80 NEXT J
85 PRINT AT 7,0;"NUMBER"
90 PRINT AT 9,0;N
95 PRINT AT 11,0;"NEXT ONE PLEASE"
100 PRINT
105 RUN
```

I CHING

```
5 RAND
10 LET P=6
15 LET B$="⟵⟵⟵⟵"
20 FOR K=0 TO 5
25 LET A$="⟵⟵"
30 IF RND<.5 THEN LET A$="⟵⟵"
35 PRINT AT P,11;B$;A$;B$
40 LET P=P+2
45 NEXT K
50 INPUT Q$
55 IF Q$=" STOP " THEN STOP
60 CLS
65 RUN
```

Chinese Horoscope

Most people know their birth sign according to Western astrology (even those who claim not to be interested in the subject), but with this program you can discover not only your Chinese birth sign, but also your ascendant, your ruling element, and the sign governing the MONTH of your birth.

To run, press GOTO Ø. Do not press RUN (for reasons we shall see shortly). The 4 inputs required are all numeric.

YEAR: e.g., 1959
MONTH: e.g., 4 (do not type April)
DATE: e.g., 23
HOUR: MUST be according to 24-hour clock, e.g., 133Ø (for 1:30 PM). If you do not know the hour of your birth, enter any 4-digit number, but ignore the ascendant given.

NOTES

Whenever you declare a variable in a program, it takes up room both in the Program Area and in the Variable Store. This can be extravagant at the best of times, but when, as in this case, you have a string variable 84 characters long, it is going to take a big chunk out of 1K.

However, it is possible to declare variables in immediate mode, and then refer to them in a program. Unlike some more expensive computers, the Timex 1000 will SAVE variables designated along with the program itself.

The problem with this technique is that you must not press CLEAR or RUN, but must use GOTO . . . (line number of start of program). Fortunately GOTO can be used with non-existent line numbers. so GOTO Ø is always adequate. If you find you have difficulty remembering when to use RUN, and when to use GOTO Ø. consider using the 'Auto-Run' facility dealt with later.

CHINESE HOROSCOPE

FIRST ENTER IN IMMEDIATE MODE:-

```
LET A$="MONKEY←ROOSTERDOG←←←←BOAR←←←RAT
←←←←OX←←←←←TIGER←←RABBIT←DRAGON←SNAKE←←
HORSE←←SHEEP←←"

LET E$="METALWATERWOOD←FIRE←EARTH"

 5 PRINT "YEAR"
 10 INPUT Y
 15 PRINT "MONTH"
 20 INPUT M
 25 PRINT "DATE"
 30 INPUT D
 35 PRINT "HOUR"
 40 INPUT H
 45 CLS
 50 IF M=1 THEN LET Y=Y-1
 55 LET S=(Y-INT (Y/12)*12)*7+1
 60 PRINT "BTH.←SIGN:";A$(S TO S+6)
 65 LET S=INT ((Y-INT (Y/10)*10)/2)*5+1
 70 PRINT "ELEMENT←←:";E$(S TO S+4)
 75 IF D>21 THEN LET M=M+1
 80 LET M=M+5
 85 IF M>12 THEN LET M=M-12
 90 LET S=(M-1)*7+1
 95 PRINT "MONTH←←←←:";A$(S TO S+6)
 100 LET S=INT (((INT (H/100))+1)/2)+5
 105 IF S>12 THEN LET S=S-12
 110 LET S=S*7-6
 115 PRINT "ASCENDANT:";A$(S TO S+6)
```

Biorhythm

In this program, you are asked to enter your date of birth (format as for Chinese Horoscope) and the current date. The program then calculates where you are in each of the three biorhythm cycles: physical, emotional and intellectual.

These cycles are 23, 28 and 33 days long, respectively. 'Peak' condition is ¼ of the way through a cycle, and 'low' is ¾ of the way through. The start/finish and mid-points of each cycle are believed to be 'danger' areas. (So don't bother attempting any complex computer programming on days when you get intellectual: 16.)

NOTES

The program utilizes the handy fact that Timex 1000-Basic gives every number a code of 28 more than the number itself. By an amazing coincidence, 28 happens to be the lowest number of days in a month, so the 'data string' (M$) is used to hold the number of days in each month as a single digit. M$(1) for instance is 3, and so CODE M$(1) is 31, the number of days in January. (M$ is only 11 characters long, as data for prior months only are required, so December is omitted.)

Line 155 adjusts for leap years, but by a strange quirk of the calendar, 1900 was not a leap year, so the program won't work if you are over 81! (2000 however WILL be a leap year, so it is valid for a considerable time yet.)

BIORHYTHM

```
5 LET M$="30323233232"
10 PRINT "D.O.B."
15 GOSUB 100
20 LET P=Q
25 PRINT "DATE TODAY"
30 GOSUB 100
35 LET P=Q-P
40 PRINT TAB 10;D;".";M;".";Y
45 PRINT AT 3,0;"PHYSICAL (23):";TAB 27;
     P-23*INT (P/23)
50 PRINT AT 5,0;"EMOTIONAL (28):";TAB 27;
     P-28*INT (P/28)
55 PRINT AT 7,0;"INTELLECTUAL (33):";
     TAB 27;P-33*INT (P/33)
90 STOP
100 PRINT "DAY"
105 INPUT D
110 PRINT "MONTH"
115 INPUT M
120 PRINT "YEAR"
125 INPUT Y
130 CLS
135 LET Q=0
140 FOR J=1 TO M-1
145 LET Q=Q+CODE M$(J)
150 NEXT J
155 IF Q>59 AND Y/4=INT (Y/4) THEN LET Q=Q+1
160 LET Q=Q+D+INT (365.25*(Y-1))
170 RETURN
```

Biochart

If you prefer the more conventional graphic representation of biorhythms, this program prints out your chart for the next 3 weeks. Three inputs are required, these being the current biorhythm data as obtained from the previous program.

The horizontal axis of the graph represents days, the first '—' being today. The vertical axis is, of course, arbitrary.

The 3 graphs are superimposed on one another: If you find this confusing, try modifying the program to print them separately.

BIOCHART

```
5 FOR J=0 TO 23
10 PRINT AT 6,J;"-"
15 NEXT J
20 LET P=23
25 PRINT AT 0,0;"PHYSICAL"
30 GOSUB 100
35 LET P=28
40 PRINT AT 0,0;"EMOTIONAL"
45 GOSUB 100
50 LET P=33
55 PRINT AT 0,0;"INTELLECTUAL"
60 GOSUB 100
90 STOP
100 INPUT N
105 FOR J=0 TO 46
110 PLOT J,SIN ((J+(2*N))*PI/P)*10+30
115 NEXT J
120 RETURN
```

Decisions

Finally a less serious method of fortune-telling! The inspiration for this program came from one of those swinging pendulum executive toys, which just goes to show that you can get programming ideas from all sorts of sources!

NOTES

Line 5: The program doesn't really care what question is asked, so dimensioning Q$ to 1 element allows the user to type away to his heart's content, without consuming valuable memory. This technique can be used in many more serious applications, where the action the computer takes is dependant only on the first letters of the user's input.

Line 135: The letter Z is not a misprint! True, the variable Z has not occurred elsewhere in the program, so line 135 will cause the program to terminate with error code 2/135 (undefined variable), but of course this is exactly what we want. (If you feel uneasy about deliberately inducing errors in your programs, note that the Sinclair manual refers to 'Report' Codes, not Error Codes!)

Lines 200–205: This is a dummy loop, which we prefer to PAUSE because of the screen flicker. It's more expensive on memory, of course, but in this case that is counteracted by the loop-control variable ending up with a value which is useful elsewhere. At the end of the loop, X = 9 (not 10!!), which is precisely the value required in line 35, so that option 10 is not chosen twice. Don't waste loop-control variables!

DECISIONS

```
5 DIM Q$(1)
10 SCROLL
15 PRINT "ASK ANY QUESTION"
20 INPUT Q$
25 IF Q$="←" THEN GOTO 20
30 LET X=10
35 CLS
40 LET X=INT (RND*X)+1
45 IF X=1 THEN PRINT "YES"
50 IF X=2 THEN PRINT "NO"
55 IF X=3 THEN PRINT "MAYBE"
60 IF X=4 THEN PRINT "HOW SHOULD I KNOW?"
65 IF X=5 THEN PRINT "NEVER"
70 IF X=6 THEN PRINT "OF COURSE"
75 IF X=7 THEN PRINT "THAT IS UP TO YOU"
80 IF X=8 THEN PRINT "POSSIBLY"
85 IF X=9 THEN PRINT "I GIVE UP-WHAT DO
          YOU THINK?"
90 PRINT
95 IF X=10 THEN GOTO 200
100 PRINT "ARE YOU SATISFIED WITH MY
          REPLY?"
105 INPUT Q$
110 IF Q$="N" THEN GOTO 30
115 CLS
120 PRINT "ANY MORE?"
125 INPUT Q$
130 IF Q$="Y" THEN RUN
135 PRINT "O.K.BYE";Z
200 PRINT "THIS IS TRICKY...LET ME
          THINK..."
205 FOR X=100 TO 10 STEP-1
210 NEXT X
215 GOTO 35
```

Chapter 2
Printing With More Frills

(If you don't know where the 'more' comes from, you haven't read the Sinclair manual properly!)

Here are 3 routines to enhance your screen displays.

Advertising

So named because we sometimes leave this running in our shop window. You can, of course, change lines 15, 30, 45 and 55 to anything you like.

NOTES

The main routine (lines 1000–1070) prints the strings OUTWARDS FROM THE CENTRE.

The secondary routine (lines 2000–2030) inverts the strings on alternate printings.

K simply alternates between 'true' and 'false' (line 930) to call the inversion routine when required.

N is the horizontal print coordinate, and can be varied at will.

Always ensure that A$ contains an even number of characters, or the first one will be lost. If you like, you could add a routine to add a space to A$ whenever necessary, but it is far more economical to do so manually.

AD.

```
5 SLOW
10 LET K=0
15 LET A$="SINCLAIR ZX 81"
20 LET N=2
25 GOSUB 1000
30 LET A$="BY SINCLAIR RESEARCH"
35 GOSUB 1000
40 LET N=N+5
45 LET A$="SOFTWARE FROM:"
50 GOSUB 1000
55 LET A$="V+H←←182C KINGSTON RD. STAINES"
60 GOSUB 1000
900 FOR I=1 TO 100
910 NEXT I
920 CLS
925 REM :DELETE 920 FOR OTHER EFFECT
930 LET K=NOT K
940 GOTO 15
1000 LET L=LEN A$
1010 LET N=N+2
1020 IF K THEN GOSUB 2000
1030 FOR I=1 TO L/2
1040 PRINT AT N,16-I;A$(L/2+1-I)
1050 PRINT AT N,15+I;A$(L/2+I)
1060 NEXT I
1070 RETURN
2000 FOR J=1 TO L
2010 LET A$(J)=CHR$ (CODE (A$(J))+128)
2020 NEXT J
2030 RETURN
```

Inverse Print

If you are unfamiliar with machine code listings, the procedure is as follows:

1: Spot the program! The only part you have to enter is the second column, beginning Ø6 16 2A . . . and ending 1Ø EF C9. It doesn't matter whether there are 1, 2 or more 'bytes' per line: They are arranged like this to help you understand the program, which, with the help of the Sinclair manual, you should be able to do.

2: Count the number of bytes; in this case there are 23.

3: Enter a Machine Code Loader. There is a very simple one listed on page 51 if you don't have one already. Line 1 should contain 23 (for this routine) X's, and LIMIT in line 1Ø should be set to 16536.

4: Run the loader, and enter the 23 2-character bytes.

5: (Optional) Poke the routine to line Ø (using another routine you'll find later) and delete the loader.

6: Finally, if you are wondering if all this was worth the effort, enter the DEMO program which follows.

Call the routine by: LET K = USR 16514.

Demo

Assuming that you have entered the machine code Inverse Print program, this demonstration shows the fantastic difference in speed between machine code and Basic.

Lines 1ØØ–14Ø translate the Inverse Print routine into Basic. (Not quite: the m/c routine searches the whole screen; this Basic version just covers the screen area which we know is in use, but the principle is exactly the same.)

Because the m/c routine changes the whole screen INSTANTLY, option 2 introduces a delay loop (lines 5Ø–55) to allow time for you to take your finger off the key. Option 3 omits this loop, so holding down key 3 creates the flashing effect.

INVERSE PRINT

```
00    06 16      START :LD B,22          ;22 LINES
02    2A 0C 40          LD HL,(16396);D.FILE
05    23         LOOP  :INC HL           ;NEXT POSN.
06    7E                LD A,(HL)       ;CHR. CODE
07    FE 76             CP 118          ;IS IT 'N/L'
09    28 09             JRZ 9           ;IF SO,SKIPNL
0B    FE 00             CP 0            ;IS IT SPACE
0D    28 03             JRZ 3           ;IF SO,SKIPSP
0F    C6 80             ADD A,128       ;INVERT CHR.
11    77                LD(HL),A        ;PRINT
12    18 F1      SKIPSP:JR 241          ;NEXT CHR.
14    10 EF      SKIPNL:DJNZ 239        ;NEXT LINE
16    C9                RET             ;RETURN
```

DEMO

```
5 REM -M/C INVERSE PRINT-
10 PRINT AT 1,1;"INVERSE PRINTING DEMONSTRATION"
15 PRINT AT 4,5;"1: BASIC"
20 PRINT AT 6,5;"2: MACHINE CODE"
25 PRINT AT 8,5;"3: ←←←'←←←←←'←←FLASHING"
30 IF INKEY$<"1" OR INKEY$>"3" THEN GOTO 30
35 IF INKEY$="1" THEN GOTO 100
40 LET K=USR 16514
45 IF INKEY$="3" THEN GOTO 30
50 FOR K=1 TO 30
55 NEXT K
60 GOTO 30
100 LET H=PEEK 16396+256*PEEK 16397
110 FOR K=H TO H+102
120 IF PEEK K=0 OR PEEK K=118 THEN GOTO 140
130 POKE K,PEEK K-128
140 NEXT K
150 GOTO 30
```

Data Print

There have been many programs published (and you have probably written one yourself) which enable you to 'draw' on the screen. But how can you SAVE your masterpiece? The idea of Data Print is to store your graphic displays in a data string. The program also offers a couple of useful aids to simplify your drawing.

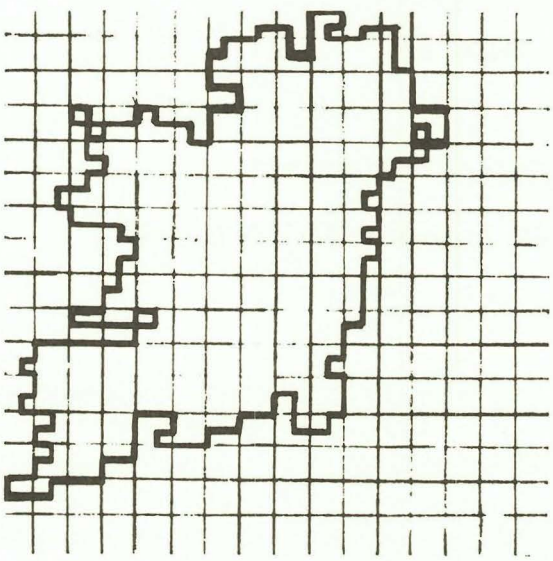
If you want a number of blank spaces, just enter that number into the string. Similarly, if you want a number of black spaces, enter the INVERSE number. To force a line feed, i.e., if all the remainder of the line is blank, enter N.

Before you run the program, you must first create your drawing. Use a sheet of graphic paper. Then enter the data string, using the facilities mentioned above for large areas of black or white. You will probably find it easier to 'read' direct from your drawing, rather than write down the string first, but don't forget the N at the end of each line.

The example we have chosen to illustrate this very useful technique is a pretty appalling map of Ireland! None of us at V&H are artists, and this is the best we could come up with! If any of you can do any better, let us know and we will use the best example(s) in the next edition of this book.

Note: If your data string is not much longer than 'Ireland,' it will fit in the program as line 1Ø: otherwise enter it in immediate mode and use GOTO Ø to run.

Please note that, for the Ireland map, we have abandoned the usual listing conventions. The first number should be entered as a normal number. The characters in the central column are either inverse numbers (when underlined) or graphics. The N of course is a normal N. The entire string should be entered continuously with NO SPACES.



Think you could do any better?
(Could you possibly do any worse?)

DATA PRINT

```
20 FOR J=1 TO LEN A$
30 LET N=CODE A$(J)
40 IF N<>51 THEN GOTO 70
50 PRINT
60 GOTO 90
70 IF (N>28 AND N<38) OR (N>156
        AND N<166) THEN GOTO 100
80 PRINT CHR$ N;
90 NEXT J
99 STOP
100 FOR K=128*(N>128) TO N-29
110 PRINT CHR$ (128*(N>128));
120 NEXT K
130 GOTO 90
```

DATA FOR 'IRELAND'

```
10 LET A$="7    34E34       N
           6    Q1W25       N
           6    75          N
           2    Y6W46R      N
           2    28E1        N
           1    30Q7E       N
           2    7R6E        N
           3    36E         N
           3    Q65         N
           2    77Q51       N
                38E         N
                37R1        N
           1    Q2317
           1    21          N
           1    Q17         N
                71"
```

Chapter 3
Casino Gambling

We are certainly NOT trying to encourage gambling. On the contrary, you will lose so much hypothetical money playing the games in this section, and the Timex 1000 will lose so much money playing roulette, that you should be convinced that the only winners are the casino owners! Nevertheless, casino games almost always hold a fascination for computer users, and so we are including a few of the more unusual examples for you to play. First, though, let's give the Timex 1000 a night out, and see if it can break the bank at . . .

Roulette

Many people claim to have created winning systems by using a computer. Can it be done? That is up to you to discover; we are going to provide a 'standard' roulette program, and then use it to test 4 of the most popular systems. You can slot your own ideas in, and see if you can come up with a winner.

The Timex 1000 is not going to try for spectacular wins by playing the 'long shots.' It knows that the best odds in roulette are for the 'even money' changes: Pair, Impair, Rouge, Noir, Manque and Passe (or, in English, Even, Odd, Red, Black, 1–18, 19–36). It doesn't matter which of these is chosen, so for this series of games, the computer will always bet 'Even.'

THE STANDARD PROGRAM

This is a 'non-system' in which the computer bets a constant stake throughout. It should show a small but ever-worsening loss. (Problem: What is the expected loss over 370 games for a constant $1 stake?)

NOTES

N is the actual winning number, which is not required unless it happens to be 0, in which case half of the stake is returned (lines 150 and 270).

X is the winning number, expressed as even, (X = 1) or odd (X = Ø). X is also set to Ø if N = Ø, so that, for the systems, Ø is considered to be a losing spin.

The 'menu' (lines 5Ø–7Ø) is there in case you don't trust the computer to play fairly! Select option 2 to input numbers from, say, a toy roulette wheel (or recorded from an actual casino, if you want to get really fanatical about this).

ROULETTE

```
10 RAND
20 LET P=0
30 LET T=1
40 LET S=T
50 PRINT "1:COMP WHEEL"
60 PRINT "2:YOUR←←←"
70 INPUT A
80 CLS
90 PRINT "I BET EVENS:$";S
100 LET P=P-S
110 LET N=INT (RND*37)
120 IF A=2 THEN INPUT N
130 IF N<0 OR N>36 THEN GOTO 120
140 LET X=0
150 IF NOT N THEN GOTO 270
160 IF N/2=INT (N/2) THEN LET X=1
170 PRINT N
180 PRINT "I←";
190 GOSUB 400-50*X
200 PRINT "NET $";
210 IF SGN P=1 THEN PRINT "+";
220 PRINT P,,"AFTER←";T;"←SPINS"
230 FOR X=1 TO 50
240 NEXT X
250 LET T=T+1
260 GOTO 75
270 LET P=P+S/2
280 GOTO 170
350 LET P=P+S+S
360 PRINT "WIN"
370 RETURN
400 PRINT "LOSE"
410 RETURN
```

Line 13Ø is just an input check, and serves little purpose (none whatever if you select option 1). Similarly, line 21Ø is purely cosmetic. Both these lines may be deleted, as may lines 5Ø, 6Ø, 7Ø and 12Ø, if you require more room to test a particular system.

SAVE a copy of the standard program, as it is the basis for all of the next 4.

Martingale

This system involves doubling the stake after a loss, and reducing it to 1 after a win. It is infallible! (But see the comments on page 32).

Reverse Martingale

This is the exact reverse of the above, and it should make a small but progressive loss. However, there will be the occasional big win, which SHOULD leave you in profit. The system fared badly in our tests — possibly because its most effective use is not over a set number of games, but just up to a point when the user is in profit. As this seems to go against the psychology of gambling, the authors could see little value in the system.

D. Alambert

Raise the stake by 1 unit after a loss. Reduce it by 1 after a win. This tends to keep the stakes lower than with the Martingale, but notice that it did produce the most spectacular losses in the series of test games!

Labouchere

This was so amazingly successful that I think I'll head straight to the nearest casino, rather than explain to you how it works!

(And while you are working that out, see if you can see the flaw in all these systems, and why you can't actually break the bank using any of them.)

MARTINGALE

```
365 LET S=1
405 LET S=S*2
```

REVERSE MARTINGALE

```
365 LET S=S*2
366 IF S>256 THEN LET S=1
405 LET S=1
```

D.ALAMBERT

```
365 LET S=S-1
366 IF S=0 THEN LET S=1
405 LET S=S+1
```

LABOUCHERE

```
40 REM ** DELETE LINE 40 **
130 REM ** DELETE LINE 130 **
73 LET A$="↑1274↑"
75 LET S= CODE A$(1)+CODE A$(LEN A$)
255 IF A$="" THEN GOTO 73
365 LET A$=A$(2 TO LEN A$-1)
403 IF S>255 THEN LET S=255
405 LET A$=A$+CHR$ S
```

Results Analysis

All 5 roulette programs were tested over a series of 8 'sessions,' each session consisting of 370 games. The results were as follows:

	1	2	3	4	5	6	7	8	Average
Standard	-69	-54	-17	-5	8	16	-22	-36	-22
Martingale	212	262	207	203	186	223	178	210	210
Reverse Martingale	-167	-178	-177	90	-164	92	91	88	-41
D. Alambert	-264	-655	48	184	-673	212	80	78	-124
Labouchere	735	664	766	603	773	512	744	227	628

CONCLUSIONS

As only one of the 'control' sessions came close to the expected loss, our main conclusion was that even 2,960 games was insufficient for serious statistical analysis!

However, we did conclude that, provided you go into a casino with the understanding that you will almost certainly lose money, both the D. Alambert and Labouchere would probably provide a more entertaining evening than simply random betting.

Before you get too enthusiastic about LABOUCHERE, we must tell you that in one final test session, this produced a loss of $3,205!!!

For further details, see page 32.

Boule

If all that theorising hasn't put you off to gambling for life, here is a game which might! Boule is a sort of 'mini-roulette' which is often found in France, but is illegal in England because the casino advantage is so high.

There are 9 numbers on the wheel; 1, 3, 6 and 8 are black; 2, 4, 7 and 9 are red, or in the Timex 1000 version, white. The 5 has a function similar to 0 in roulette, i.e., it can be backed as a number but not in any of the combination bets. Unlike roulette, however, all bets except those on the number 5 lose completely whenever 5 is spun. The bets available are:

Noir (any of the 4 black numbers)
Blanc (any of the 4 white numbers)
Manque (any of the low numbers 1–4)
Passe (any of the high numbers 6–9)
Pair (any of the even numbers)
Impair (any of the odd numbers EXCEPT 5)
All the above pay even money
Any number individually (including 5) pays 7 to 1.

There is a board for playing Boule printed on page 28.

NOTES

The main program loop (lines 15-65):

1) Prints the numbers of the 'wheel' (lines 30 and 45).
2) Is of variable duration (line 20).
3) Slows down towards the end (lines 50 and 55).
4) Returns the winning number in N.

BOULE

```
5 RAND
10 LET N=0
15 LET A$="1234*6789"
20 LET M=(18*(INT (RND*5)+1))+(INT
       (RND*9)+1)
25 FOR J=1 TO M
30 PRINT AT 0,N*3;"←"
35 LET N=N+1
40 IF N>9 THEN LET N=N-9
45 PRINT AT 0,N*3;A$(N)
50 FOR K=M-10 TO J
55 NEXT K
60 LET D=RND*RND
65 NEXT J
100 PRINT N;" PAYS 7-1"
105 IF N=5 THEN GOTO 155
110 LET A$="NOIR"
115 IF N=2 OR N=4 OR N=7 OR N=9
       THEN LET A$="BLANC"
120 GOSUB 200
125 LET A$="MANQUE"
130 IF N>5 THEN LET A$="PASSE"
135 GOSUB 200
140 LET A$="PAIR"
145 IF N/2<>INT (N/2) THEN LET A$=
       "IM"+A$
150 GOSUB 200
155 PRINT "ALL OTHER BETS LOSE"
160 CLEAR
165 INPUT A$
170 CLS
175 RUN
200 PRINT A$;" PAYS EVEN"
210 RETURN
```

Line 60 is another form of pause, again preferable to the PAUSE statement. It can be omitted if you want to play the game at a faster pace.

The remainder of the program prints the winning bets. Line 160 is included before the re-run option; otherwise there may not always be room for even a null input.

Noir et Blanc

Another obscure French game, Noir et Blanc is normally played with cards. The dealer lays a horizontal row of cards, and continues adding to the row until the 'pip-count' totals more than 30 (A = 1; picture cards = 10). He then deals a second row, again until it totals more than 30. The winning row is the row with the LOWEST total. The top row is called Noir, and the second row is called Blanc.

Four bets are allowed, which all pay even money:

Noir: The top (noir) row will win.

Blanc: The second (Blanc) row will win.

Couleur: The color of the first card dealt will be the same as the NAME of the winning row.

Inverse: The color of the first card dealt will NOT be the same as the name of the winning row.

It is these last 2 bets which give the game an appeal totally out of proportion with its complexity!

NOTES

If both rows total the same, it is a draw and all bets are returned, UNLESS they both total 31, in which case there is a REFAIT, i.e., all bets remain on the table, to be returned only if they win on the next deal.

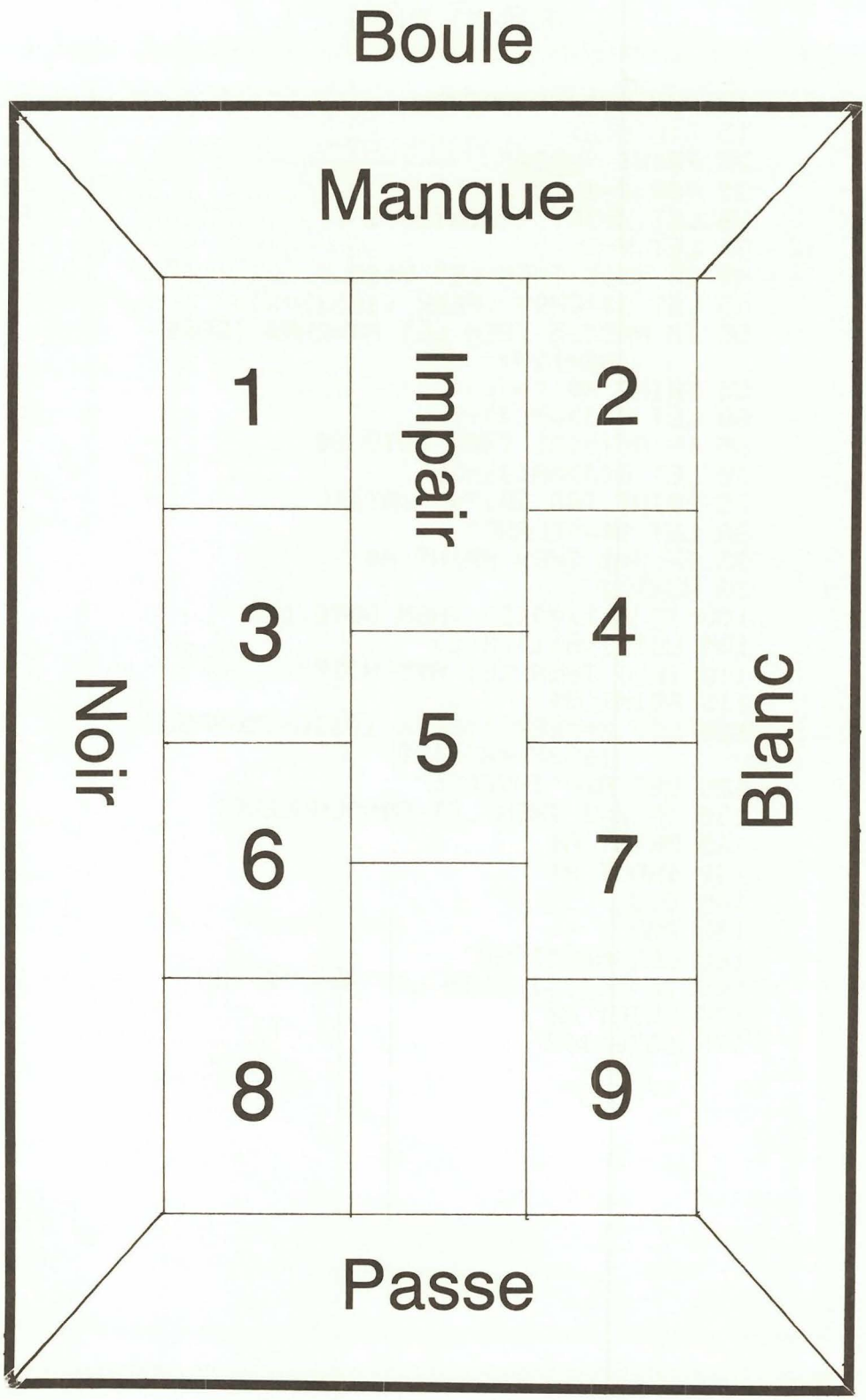
Again there is a main program loop (lines 25–90) and a 'tail' which prints out the winniners. Line 10 is another form of data statement; rather than hold the cards in a string, putting them in the first program line makes them easily accessible by PEEK (line 45).

Line 50 randomly inverts the selected card, so that the entire 'pack' need only contain 13 cards.

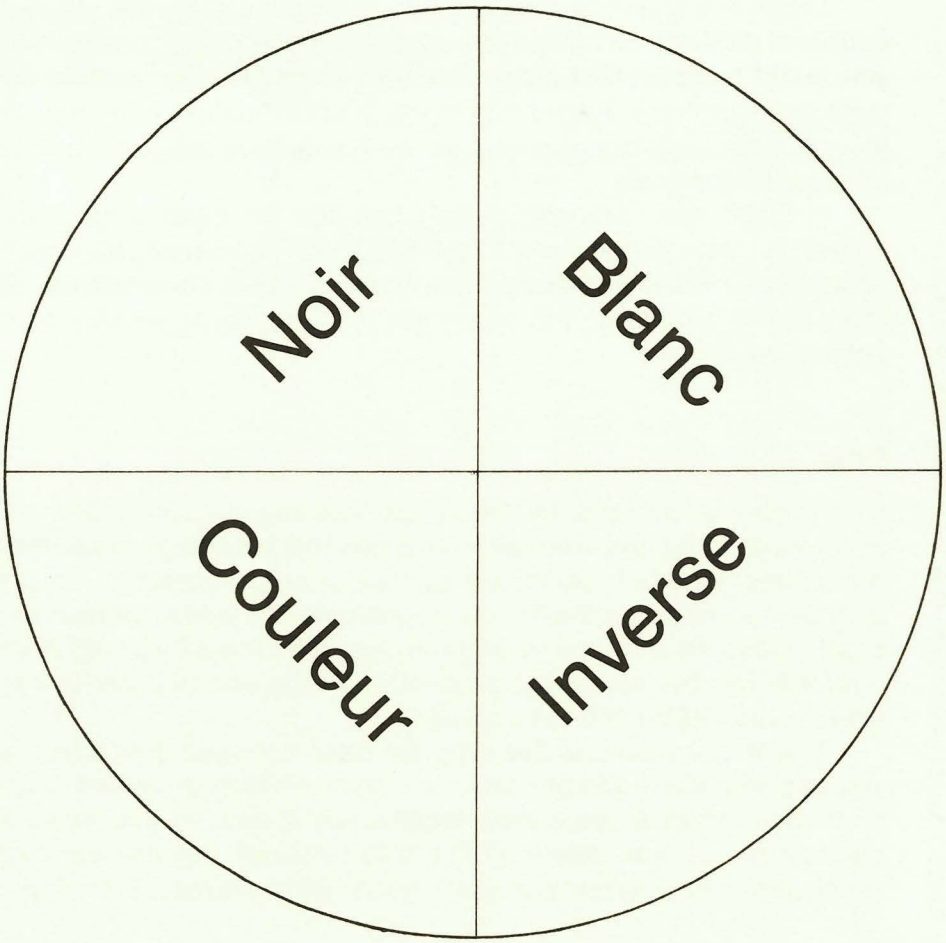
The winners of the two pairs of bets are determined by the Boolean logic of lines 105 and 120. The couleur/inverse bet is settled in line 120 by first PEEKing at the first card (6 positions into the display file, because the display always starts with N, O, I, R and Newline), and setting a flag (X = 1) if the card is black. (Otherwise X = 0.) The flag J has already been set to 1 if the Noir line wins (line 105), so the Couleur bet will win if X and J are the same, regardless of whether they are both 0 or 1.

NOIR ET BLANC

```
10 REM A23456789TJQK
15 DIM A(2)
20 PRINT "NOIR"
25 FOR J=1 TO 2
30 LET X=INT (RND*13)+1
35 LET Y=X
40 IF Y>10 THEN LET Y=10
45 LET A$=CHR$ (PEEK (16513+X))
50 IF RND<.5 THEN LET A$=CHR$ (CODE
       A$+128)
55 PRINT A$;"←";
60 LET A(J)=A(J)+Y
65 IF A(J)<31 THEN GOTO 30
70 LET A(J)=A(J)-30
75 PRINT TAB 20;"= ";A(J)
80 LET A$="BLANC"
85 IF J=1 THEN PRINT A$
90 NEXT J
100 IF A(1)=A(2) THEN GOTO 160
105 LET J=A(1)<A(2)
110 IF J THEN LET A$="NOIR"
115 PRINT A$
120 LET X=PEEK ((PEEK 16396+256*PEEK
       16397)+6)>128
125 LET A$="INVERSE"
130 IF X=J THEN LET A$="COULEUR"
135 PRINT A$
140 INPUT A$
145 CLS
150 RUN
160 LET A$="DRAW"
165 IF A(1)=1 THEN LET A$="REFAIT"
170 PRINT A$
175 GOTO 140
```


Boule
Manque
1
Impair
2
3
4
5
Noir
Blanc
6
7
8
9
Passe

Noir et Blanc



Craps

We must state at one that this is the name of the game, and not a description of it. (Though some readers might disagree!)

Craps is a typical example of how a simple activity like rolling a couple of dice can be turned into a game so amazingly complex that you would need an IBM megacomputer to hold all the possible options and variations. Faced with trying to cram such a game into 1K, there are two possible approaches, and indecisive as ever, we have adopted both of them.

In DICE, the computer merely handles the mechanics of the game (i.e., just rolls the dice). CRAP SHOOTER eliminates all players except one, thereby reducing the variety of bets considerably, as there are no 'side bets' and, of course, the 'shooter' is not allowed to bet that he will lose.

DICE

Dice throwing programs for the Timex 1000 are not uncommon, but we have included this one because it has the advantage of allowing the number of dice to be thrown at once to be pre-selected. Up to 4 dice will be shown on the screen at one time. If a larger number has been requested, then the screen will clear after the pause loop (lines 164–166) and the next 4 will be printed. At the end of each 'throw,' simply press NEWLINE to throw again.

The line numbers of this program have not been 'tidied up,' so you can see where the pre-selection routine has been added to the original program. If you merely require, say, 2 dice, as you would to play craps, you can delete all the 'late additions' (all line numbers which don't end in Ø) and change D in line 18Ø to the required number.

DICE

```
4 PRINT "NO. OF DICE?"
6 INPUT D
10 RAND
20 DIM A$(6,15)
30 LET A$(1)="▒<<<<<<<▒"
40 LET A$(2)="▒<<<<<<<<<<<<<▒"
50 LET A$(3)="▒<<<<<<▒<<<<<<▒"
60 LET A$(4)="▒<<<▒<<<<<▒<<<▒"
70 LET A$(5)="▒<<<▒<<▒<<▒<<<▒"
80 LET A$(6)="▒<<<▒▒▒<<<▒▒▒<<<▒"
90 CLS
100 LET N=0
110 LET X=INT (RND*6)+1
120 FOR J=1 TO 11 STEP 5
130 PRINT TAB N;A$(X,J TO J+4)
140 PRINT
150 NEXT J
160 LET N=N+8
162 IF N/32<>INT (N/32) THEN GOTO 170
164 FOR X=0 TO N
166 NEXT X
168 CLS
170 PRINT AT 0,0;
180 IF N<8*D THEN GOTO 110
190 INPUT Q$
200 GOTO 90
```

CRAP SHOOTER

As already stated, this is a one-player version of craps, with no side bets. The shooter (you) starts with $100 and stakes $1 per game. A 'game' proceeds as follows:

First roll:
7 or 11: NATURALS: Win immediately
2, 3 or 12: CRAPS: Lose immediately
Anything else: COME-OUT ROLL: Proceed to roll #2
Second and subsequent rolls:
Continue rolling until you roll either:
Your come-out roll (again): WIN
7: LOSE

Note that 7, the most common roll with two dice, wins if rolled first, but otherwise loses. 2, 3, 11 and 12 have no significance other than for the first roll.

FINAL THOUGHTS ON ROULETTE

Before we leave gambling completely, here are the answers to the questions posed on pages 20 and 21.

The expected loss over 370 games (constant $1 stake) is $5.

The reason that none of the systems work is that all casinos have a maximum AND a minimum stake, i.e., even if you start at the minimum stake, any system which involves increasing the stake is eventually going to exceed the maximum, at which point, of course, it will break down.

The Labouchere tends to avoid the problem of reaching the maximum by reducing the stake more often than increasing it. (Almost twice as often—if you still can't see how it works from the listing, drop us a line and we will explain it.) As you can see from page 24, it works quite often . . . but when it fails, it really fails!

CRAP SHOOTER

```
10 PRINT TAB 10;"CRAP SHOOTER"
15 LET N=101
20 LET C=0
25 LET N=N-1
30 PRINT AT 15,0;"YOU HAVE $";N
35 LET Y=0
40 PRINT
45 PRINT "PRESS N/L TO ROLL"
50 INPUT Q$
55 CLS
60 LET X=INT (RND*6)+1
65 PRINT X,
70 LET Y=Y+X
75 IF Y=X THEN GOTO 60
80 PRINT
85 IF C THEN GOTO 105
90 IF Y<4 OR Y=12 THEN GOTO 200
95 IF Y=7 OR Y=11 THEN GOTO 250
100 LET P=Y
105 LET C=C+1
110 PRINT
115 PRINT "COME OUT ROLL: ";P
120 IF C=1 THEN GOTO 30
125 PRINT
130 IF Y<>P THEN GOTO 160
135 PRINT
140 PRINT TAB 10;"**POINT MADE**"
145 LET N=N+2
150 GOTO 20
160 IF Y=7 THEN GOTO 200
165 PRINT "ROLL: ";Y
170 GOTO 30
200 PRINT
205 PRINT TAB 13;Y;" CRAP"
210 GOTO 20
250 PRINT
255 PRINT TAB 10;"**NATURAL** ";Y
260 GOTO 145
```

Chapter 4
Data Files

One feature noticeably absent from the standard Timex 1000 is the ability to read and write external data files. However, by extending an idea we have used previously (that of SAVEing variables along with a program), we can effectively emulate a data file by SAVEing a copy of the program, with the LATEST value of its variables, AFTER it has been run.

The next 2 programs are for fun. Later, in the Business section, we will put this principle to more serious use.

Top Ten (Chart Data)

This program reflects the author's joint obsession with pop charts and statistics (?!) Every year, the pop music papers publish a 'points table' of artists who have had hit records that year. Each week they score 30 points for a record at number 1, 29 for number 2, etc. Each year, I am so fascinated with these 'charts of charts,' that I vow not to wait another 12 months for the next one, but to keep my own weekly up-date of the points table. Even with the aid of a computer, this is a formidable task, and rather beyond the ability of 1K. This program, therefore, is a 'mini' points table; you select 10 of your favourite artists, and follow their relative chart progress throughout the year.

NOTES

The program has been squeezed considerably to allow it to hold data on 10 artists. Line 6 is a long-winded but, believe it or not, more economical way of saying LET C=10. (Check this with the 'Byte-Counting' routine on page 47. Similarly, line 7 sets D = 1. The first 6 lines have been given these low line numbers just to allow line 115 to say GOTO C instead of GOTO 10!

G$ has been dimensioned to hold the first 10 characters (only) of the 10 artists' names (line 8).

Lines 5, 10 and 25 allow entry of artists' names on the first run only. On subsequent runs, A$ is a null string (as execution commences at line 10), so line 25 is skipped.

TOP TEN (CHART DATA)

```
5 LET A$="NAME/"
6 LET C=CODE "↑S↑"
7 LET D=C/C
8 DIM G$(C,C)
9 DIM S(C)
10 PRINT AT C,D;" INPUT ";A$;"POSN."
15 FOR N=D TO C
20 SCROLL
25 IF A$<>"" THEN INPUT G$(N)
30 PRINT G$(N),
35 INPUT X
40 IF X THEN LET S(N)=31-X+S(N)
45 PRINT S(N)
50 NEXT N
55 FOR N=D TO C
60 SCROLL
65 NEXT N
70 PRINT "PRESS S TO SAVE DATA"
75 LET A$=""
80 IF INKEY$<>"S" THEN GOTO 80
85 SAVE "TT"
90 CLS
95 PRINT
100 FOR N=D TO C
105 PRINT G$(N),S(N)
110 NEXT N
115 GOTO C
```

Computer Dating

The Timex 1000 can do many things; whether it can help create perfect marriages remains to be seen! This is intended as a 'fun' program, to be used among your school or work mates or at parties. But if any successful partnerships result, we'd be very pleased to hear! (For gay parties, delete lines 5Ø0 and 5Ø5—the Timex 1000 is nothing if not versatile!)

PRELIMINARIES

At the back of this book, you will find a questionnaire. Photocopy and distribute as many copies as possible.

Complete the 'input document' at the foot of each questionnaire. Box 1 should always contain M or F. Boxes 2–7 should contain 1 or 2, according to the item chosen. Boxes 8–1Ø should contain the 3 'likes' letters (in alphabetical order); boxes 11–13 the 3 'dislikes.'

Establish a databank as follows:

Run the program selecting menu item 1, and enter the relevant data. (Name/Box 1/Boxes 2–7/Boxes 8–13). Repeat this until you have SAVEd data on all the applicants. Note that this part of the program will end by naming each applicant as his/her ideal partner (fair enough!). Enter NEWLINE to proceed to the next applicant.

Now rewind the tape, and you are ready to start matchmaking. Select menu item 2 and enter the first applicant's data. The program will search through the tape for suitable partners. (If it fails to find a match for everyone, it serves them right for being thick, humorless, disco-going, poetry fanatics!)

NOTES

The interesting feature of this program is that it uses the program NAME as a data store, a concept which you should be able to use elsewhere.

Lines 5Ø0–5Ø5 reverse the first element of the data string P$ (1 is female: Ø is male) so that a match of the opposite sex is found.

COMPUTER DATING

```
5 PRINT TAB 9;"COMPUTER DATING"
10 PRINT "1: DATA"
15 PRINT "2: PARTNERS"
20 INPUT M
25 PRINT "NAME"
30 INPUT N$
35 PRINT "SEX M/F"
40 INPUT S$
45 LET P$="1"
50 IF S$="M" THEN LET P$="0"
55 PRINT "PERSONALITY CODE "
60 GOSUB 200
65 PRINT "LIKE/DISLIKE CODE "
70 GOSUB 200
75 IF M=2 THEN GOTO 500
80 PRINT "START TAPE: PRESS S TO SAVE DATA"
85 IF INKEY$<>"S" THEN GOTO 85
90 SAVE P$
95 CLS
100 PRINT "YOUR IDEAL PARTNER IS"
105 PRINT N$
110 PRINT "WANT AN ALTERNATIVE?"
115 INPUT N$
120 CLS
125 IF N$<>"Y" THEN RUN
130 GOTO 510
200 INPUT S$
210 IF LEN S$<>6 THEN GOTO 200
220 LET P$=P$+S$
230 RETURN
500 LET M=NOT VAL P$(1)
505 LET P$(1)=STR$(M)
510 PRINT "PRESS L TO LOAD DATA"
515 IF INKEY$<>"L" THEN GOTO 515
520 LOAD P$
```

Check Digit

This program has nothing whatever to do with data files, and appears in this section due to a computer error. (Well, that's the excuse everyone else uses!)

A check digit is a very important but much neglected form of data validation. The LEAST serious consequences of invalid data being entered into a program are that the program will crash or will produce inaccurate results (the well-known GIGO phenomenon: Garbage In, Garbage Out). The consequences can be far more serious if the invalid entry is made fraudulently by an unauthorized person. The principle of a check digit is that, whenever a numeric entry is required (typically a stock code number, for example), the final digit of the number is in some way related to the rest. Therefore a randomly chosen entry will not be accepted, as it does not satisfy the check digit's validation rules.

The simplest form of check-digiting is to add the digits of a number together (reducing if necessary to a single digit, as we did in 'Numerology'), and append the result to the original number (thus 153 becomes 1539). This program adopts a more complex approach, and should appeal to the more security-conscious readers.

First you invent a 'code': any series of numbers, as long or as short as you like. Then enter your number to be 'check-digitized.' The check digit is formed by taking each digit of the number in turn, multiplying it by the corresponding digit in the code, and then adding these numbers together as before.

NOTES

The routine at lines 45–60 temporarily extends the check code in cases where the number entered is longer than the code.

To run the program for a series of numbers, press newline after each 'security number' appears. To change the check code, enter N.

CHECK DIGIT

```
5 LET A$=""
10 PRINT "CHECK CODE ",
15 IF A$="" THEN INPUT A$
20 PRINT A$
25 PRINT "NUMBER",
30 INPUT B$
35 PRINT B$
40 LET Y=0
45 LET C$=A$
50 IF LEN C$>=LEN B$ THEN GOTO 65
55 LET C$=C$+"1"
60 GOTO 50
65 FOR J=1 TO LEN B$
70 LET X=VAL C$(J)*VAL B$(J)
75 GOSUB 150
80 LET Y=Y+X
85 NEXT J
90 LET X=Y
95 GOSUB 150
100 PRINT "SECURITY CODE ",B$;X
105 PRINT AT 10,0;"N/L= SAME CODE "
110 PRINT "  N= NEW CODE "
115 PRINT "  S= STOP "
120 INPUT Q$
125 CLS
130 IF Q$="N" THEN RUN
135 IF Q$="S" THEN STOP
140 GOTO 10
150 IF X<=9 THEN RETURN
155 LET X=X-9
160 GOTO 150
```

Chapter 5
Business Programs

We are not suggesting that the 1K Timex 1000 is the BEST computer for business use, but neither have we time for suggesting that it is 'just a toy.'

The first program in this section is of the 'extended calculator' variety: you CAN work out your sales tax with a calculator, but it's a lot easier on the Timex 1000. BANK RECORDS is more 'personal finance' than 'business': again you could do this with a calculator, but many people don't bother. The Timex 1000 not only makes it easier, but also more fun. CREDIT CARDS is a very useful program to assist the retailer, for whom credit card sales often cause more problems than they are worth.

Sales Tax

No explanations necessary. The most important feature of this program is that it is menu-driven, so that you can calculate sales tax either from gross OR net figures.

Line 5 is the current sales tax rate, and is the only one you should ever have to change.

Line 500 is one you should memorize and use everywhere. It rounds floating point figures to the nearest hundredth.

SALES TAX

```
5 LET V=.15
10 LET V=V+1
15 PRINT AT 3,10;"VAT CALCULATOR"
20 PRINT AT 7,0;"1: VAT FROM GROSS"
25 PRINT AT 9,0;"2: VAT FROM NET"
30 LET I$=INKEY$
35 IF I$<"1" OR I$>"2" THEN GOTO 30
40 CLS
45 GOTO 100*VAL I$
100 PRINT "GROSS"
110 INPUT G
120 LET T=G/V
130 GOSUB 500
140 LET N=T
150 GOTO 250
200 PRINT "NET"
210 INPUT N
220 LET T=N*V
230 GOSUB 500
240 LET G=T
250 CLS
260 PRINT AT 3,5;"GROSS",G
270 PRINT AT 5,5;"NET",N
280 PRINT AT 7,5;"VAT",G-N
290 PRINT AT 20,4;"PRESS NEWLINE TO CONTINUE"
300 INPUT I$
310 CLS
320 RUN
500 LET T=INT (T*100+.5)/100
510 RETURN
```

Bank Records

This is a useful program for those of you who don't like paying for bank statements too often. When you DO get a bank statement, look out for bank charges and any other items you haven't included, and make the appropriate adjustment on your next run. You have one opportunity to put in an adjustment figure, at the 'Standing Order ETC.' entry.

NOTES

Line 22Ø prevents running out of memory from having too big a display file.

Line 26Ø may look confusing because of all those A's, but all it does is leave your entry positive when the first letter of A$ is A (i.e., A$ = "AMOUNT"). Otherwise the entry is taken as negative.

The program uses the Auto-run feature (actually 'auto-goto 3Ø) so after you have SAVEd your data, the words 'NUMBER OF CHECKS' appears. Enter an alpha character to break. If at any time you want to set a new opening balance, break immediately after loading, and press run.

Readers with unusually large bank accounts, or those who receive unusually large checks, may have to alter the formatting line 28Ø. Alternatively, they could reduce their bank balance to a more manageable amount by sending all excess money to the address at the front of this book!

BANK REC.

```
10 PRINT "OPENING BALANCE"
20 INPUT O
30 PRINT "NUMBER OF CHEQUES"
40 LET A$="CHQ."
50 GOSUB 200
60 PRINT "NUMBER OF PAYMENTS"
70 LET A$="AMT."
80 GOSUB 200
90 PRINT "VALUE OF STANDING ORDERS ETC."
100 LET N=1
110 LET A$="S.O."
120 GOSUB 210
130 PRINT "PRESS S TO SAVE DATA"
140 SCROLL
150 IF INKEY$<>"S" THEN GOTO 150
160 SAVE "BANK REC."
170 CLS
180 GOTO 30
200 INPUT N
210 FOR X=1 TO N
220 IF X/6=INT (X/6) THEN CLS
230 SCROLL
240 PRINTA$;X;
250 INPUT A
260 LET A=A-(2*A AND A$(1)<>"A")
270 LET O=O+A
280 PRINT TAB 8;A,"BALANCE ";O
290 NEXT X
300 PRINT AT 0,0
310 RETURN
```

Credit Cards

At the time of writing, there is some controversy about whether retailers should be allowed to charge a premium on credit card sales. The reason they want to do so is that credit card sales not only cost the retailer money in commission, but also create extra accounting problems. This program will not solve the first problem, but should alleviate the second.

This program is written for the ZX printer. This is to provide the user with a hard-copy listing of credit card batches submitted. It is of less value without these listings, but if you don't have a printer, change all LPRINT lines to PRINT, and add a PAUSE after line 1Ø0.

Line 5Ø is the name of the credit card company. A different copy of the program should be used for each company for whom this system applies.

The '96' in line 85 is 1Ø0 less the percent commission charged by the particular company. Amend this to the actual figure, if necessary.

The ZX printer provides a tally-roll of submissions (gross and net), which should be compared with statements received from the company.

The screen, meanwhile, displays the NET amount still outstanding.

The write-off routine (lines 13Ø–14Ø) allows for reconciliation in the event of invalid cards being dishonored by the company.

CREDIT CARDS

```
5 LET T=0
10 LET C=0
15 LET S=C
20 CLS
50 LPRINT "DINERS EXPRESS"
55 PRINT "NO. OF BATCHES"
60 INPUT N
65 FOR J=1 TO N
70 PRINT J
75 INPUT X
80 LET S=S+X
85 LET Y=INT (X*96+.5)/100
90 LET C=C+X-Y
95 LET T=T+Y
100 LPRINT " AMOUNT ";X,"NET ";Y
105 CLS
110 NEXT J
115 PRINT "RECEIPTS"
120 INPUT J
125 LET T=T-J
130 PRINT "WRITE OFF"
135 INPUT J
140 LET T=T-J
145 CLS
150 PRINT"OUTSTANDING: ";T
155 LPRINT "SUBMISSIONS ";S
160 LPRINT "COMMISSION  ";C
165 PRINT AT 5,0;"PRESS S TO SAVE DATA"
170 IF INKEY$<>"S" THEN GOTO 170
500 SAVE "DE"
505 GOTO 10
```

Chapter 6
Utilities

Below is a desultory collection of routines which we hope you will find useful.

Byte Counting

Byte counting is probably THE most useful routine you will come across while working in 1K. Used during program development, GOTO 9999 will reveal exactly the amount of space occupied by your program so far. It won't tell you if your program will actually RUN in 1K, as it measures only the program itself, not the variable or display areas. Its function therefore is to test various alternative methods of coding, to see which is the most economical. Try it: You may get some surprises, and should find yourself writing 'tighter' code almost immediately.

Line Ø

You have seen the expression 'GOTO Ø' already in this book, but have you ever actually seen a line number Ø? If you can spare the time to key in this 'program,' you will! Run and list the one line and then try to edit or delete your new line number Ø! What use is it? Well, apart from amazing your friends and impressing people, none whatever in its present form. However, if you use POKE 1651Ø,Ø in immediate mode, it will always give line number Ø to the FIRST line of your program. (The only reason it is shown as a program line here is because it has no effect if you haven't at least 1 program line to renumber. Note also, that that line number must originally be less than 256, or you must POKE 165Ø9,Ø as well.)

We can think of at least 2 uses for this:

1: Once you have laboriously entered a machine code routine into a REM statement of a basic program, it can be a little annoying to accidentally delete this line by a careless entry, so POKE the machine code to line Ø and it is perfectly secure.

2: Why not POKE to line Ø the following:

Ø REM COPYRIGHT B. BLOGGS 1982

Now your program is safe from piracy by anyone who doesn't know the secret.
You have already seen variations of the next 3 routines in the DATA FILES section. If you were not quite sure what was happening, read on.

BYTE-COUNTING

```
9999 PRINT PEEK 16396+256*PEEK 16397-16562
```

LINE 0

```
255 POKE 16510,0
```

Auto-Run

Auto-run is a feature of the Timex 1000 which is not available even on many more expensive computers! If you append these 3 lines to any of your programs, and then SAVE by entering GOTO 9998, subsequent LOADing will cause the program to auto-run. If you are very short of space, omit line 9997 (if your program stops anyway); reduce the program name to a single character; change line 9999 to 'RUN.'

Chains

This is a sort of 'auto-load,' which, if used in conjunction with AUTO-RUN, will enable you to simulate quite long programs on your 1K Timex 1000 by chaining any number of shorter programs together. You can either load each program on a tape directly from the previous one (no need in that case for a program name in line 9995) or, as in the example, end each program with an option to re-run or recall a common menu, from which any other program may be loaded. (In this case, you must either remember to rewind the cassette after each loading, or make a cassette on which each program, and the menu, is recorded several times.)

Menu

A menu, in computer terms, as otherwise, is a list of options available. It is a luxury which few 1K programs can afford, but an EXTERNAL menu like this one, which offers the first 3 programs in the book, used in conjunction with CHAINS and AUTO-RUN, can give a very professional touch to your program cassettes.

AUTO-RUN

```
9997 STOP
9998 SAVE "PROGRAM NAME"
9999 GOTO 0
```

CHAINS

```
9990 CLS
9991 PRINT "PRESS M TO RETURN TO MENU"
9992 PRINT "OR R TO RE-RUN"
9993 LET I$=INKEY$
9994 IF I$="R" THEN GOTO 0
9995 IF I$="M" THEN LOAD "MENU"
9996 GOTO 9993
```

MENU

```
5 LET A$="CHINESE HOROSCOPE"
10 LET B$="NUMEROLOGY"
15 LET C$="I CHING"
20 PRINT AT 0,2;"FORTUNE TELLING←←←←←BY V+H"
25 PRINT AT 5,0;"1:";A$
30 PRINT AT 7,0;"2:";B$
35 PRINT AT 9,0;"3:";C$
40 PRINT AT 17,6;"ENTER YOUR SELECTION"
45 LET I$=INKEY$
50 IF I$<"1" OR I$>"3" THEN GOTO 45
55 IF I$="1" THEN LOAD A$
60 IF I$="2" THEN LOAD B$
65 IF I$="3" THEN LOAD C$
```

Machine Code Loader

A machine code loader can be as simple or as complex as you like. This one has been deliberately made as simple as possible so that it can be keyed in in a matter of seconds, rather than loaded from tape every time it is required. You can, of course, add as many frills as you desire once you become seriously involved in machine code programming.

To run, amend line 1 to contain the required number of X's. (Actually any character will do.)

Amend line 10 so that the loop is executed the required number of times, i.e., replace LIMIT with 16514 plus the number of X's minus 1.

Run, entering your code 2 hex digits at a time.

SAVE ! (Most important, because if the program crashes you will have to switch off and start again.)

Run the m/c program by executing the USR function. The most foolproof method is:

LET DUMMY = USR 16514

although PRINT USR 16514 or GOTO USR 16514 may be appropriate on occasions.

If you make a mistake during entry, it is usually advisable to start again. Once you get used to machine code, however, you will find that you can edit many of the bytes directly from the 'garbage' which appears on screen when you LIST line 1.

Line Renumbering

It is a short step from 'Line 0,' where we renumbered the first line of a program, to a complete routine which renumbers an entire program. This is very useful if you want your programs to look neat and tidy, and essential if you want to sell or publish them. It is also a handy routine to have loaded if you ever find that you have left insufficient room between line numbers to add a later amendment.

NOTES

As listed, the routine will renumber in steps of 5 (S in line 8960), starting with 5 (F in line 8965). You may alter these values as you wish.

The routine works by scanning the Program Area (addresses 16509 onwards) looking for the Newline character (118). When this is found, the following 2 bytes must contain the next line number, so the new value is POKEd in (lines 8990–8985).

Line 9005 prevents the routine renumbering itself, by causing a break (list) when line number 8960 is encountered. Note, however, that the FIRST new line number is always POKEd in immediately, without the program scan, so the routine WILL renumber its own first line if you do not have a program entered for it to act on.

To run, enter GOTO 8960 (or RUN 8960) at any time. When the listing appears (eventually), you must amend all GOTO and GOSUB commands to suit the new line numbers, and finally delete the routine.

The major problems with this routine are that it is very slow and it takes up a lot of room itself, so it can be used only on very short programs. (Also it takes no less than 65 keystrokes to delete it.) There is a way around these problems, of course . . . use Machine Code.

MACHINE CODE LOADER

```
1 REM XXXXX (AS MANY AS BYTES IN M/C PROGRAM)
10 FOR A=16514 TO LIMIT
15 INPUT H$
20 POKE A,(CODE H$(1)-28)*16+CODE H$(2)-28
25 NEXT A
```

LINE RENUMBERING

```
8960 LET S=5
8965 LET F=S
8970 LET A=16509
8975 LET P=INT (F/256)
8980 POKE A,P
8985 POKE A+1,F-(P*256)
8990 LET D=PEEK A
8995 LET A=A+1
9000 IF D<>118 THEN GOTO 8990
9005 IF PEEK A=35 THEN GOTO 9020
9010 LET F=F+S
9015 GOTO 8975
9020 LIST
```

Machine Code Renumbering

If you are going to enter this routine using the loader on page 51, put 27 X's in line 1 and set LIMIT to 16540.

Important: Because this routine is translated directly from the previous Basic program, it still searches for line number 9860 (byte 16h). Therefore, call by entering the basic line:

8960 LET D = USR 16514

Do not call in immediate mode, unless you insert a dummy line 8960 (e.g., REM). (If you can't remember that, 9000 will do just as well, as only the high-order byte of the line number is tested.)

Notice that bytes 01 and 02 commence renumbering at address 16542, and not 16509 as in the Basic program. This is because the routine itself is the first program line; if you make any changes, therefore, remember to set these bytes to the start of the SECOND program line.

M/C RENUMBERING

```
00    21 9E 40  START :LD HL,16542;1ST LINE
03    11 00 00        LD DE,0     ;LINE # 0
06    06 05     RENUMB:LD B,5     ;STEPS OF 5
08    13        STEP  :INC DE     ;NEW NO.IN DE
09    10 FD             DJNZ 253  ;TO STEP
0B    72                LD(HL)D   ;  INSERT
0C    23                INC HL    ;  NEW
0D    73                LD(HL),E  ;  LINE NO.
0E    23        NXTBYT:INC HL     ;NEXT BYTE
0F    7E                LD A,(HL) ;
10    FE 76             CP 118    ;IS IT N/L
12    20 FA             JRNZ 250  ;IF NOT,NXTBYT
14    23                INC HL    ;NEXT LINE NO.
15    7E                LD A,(HL) ;
16    FE 23             CP 35     ;IS IT 8960
18    20 EC             JRNZ 236  ;IF NOT,RENUMB
1A    C9                RET       ;RETURN
```

Chapter 7
Games

At last, the section you have all been waiting for. Don't get too excited, though; it is our belief that the best source of computer games is in your own head, and in any case you will almost certainly get more enjoyment out of writing games than actually playing them. What we will do in this section, then, is show you a few of the many TYPES of computer games, with an example of each. After that, if you want to discover the next craze to replace Space Invaders, it is up to you.

Play Your Cards

Before you rack your brains thinking up new games, search the attic and the local toyshop for the hundreds of existing games just crying out for computerization. Many have already been done. (If you haven't written a mastermind program yet, you must be the only computer programmer who hasn't, so stop reading immediately and get on with it!) But there are many more waiting to be 'discovered.'

All you have to do in this program is guess whether the next card laid will be higher or lower than the last one. As soon as you make a wrong guess, the computer takes its turn, and the winner is determined by who has the greatest number of correct guesses. Simple but addictive: the definition of a good computer game.

The computer's strategy, such as it is, is to make the logical choice, except for the mid-card 8, in which case it chooses randomly (line 17Ø).

Lines 535–54Ø are yet another pause loop. This time the pause is greater when the computer is playing (pseudo-thinking time) as N<>Ø.

PLAY YOUR CARDS

```
10 REM 23456789TJQKA
20 LET X=0
30 LET N=X
40 LET A$=""
50 GOSUB 500
60 PRINT AT 1,N;"HIGHER OR LOWER"
70 LET I$=INKEY$
80 IF I$<>"H" AND I$<>"L" THEN GOTO 70
90 GOSUB 500
100 GOSUB 400
110 IF A THEN GOTO 70
120 PRINT AT 1,N;"YOUR SCORE←←←←";LEN A$-2
130 LET A$=""
140 LET N=2
150 GOSUB 500
160 LET I$="LOWER←"
170 IF X+RND<6.5 THEN LET I$="HIGHER"
180 PRINT I$
190 GOSUB 535
200 GOSUB 500
210 GOSUB 400
220 IF A THEN GOTO 160
230 PRINT "MY SCORE←←←←←←";LEN A$-2
240 GOSUB 535
250 CLS
260 RUN
400 LET A=1
410 IF (I$(1)="H" AND X<=Y) OR (I$(1)=
        "L" AND X>=Y) THEN LET A=A-A
420 RETURN
500 LET Y=X
510 LET X=INT (RND*13)
520 LET A$=A$+CHR$ PEEK (16514+X)
530 PRINT AT N,0;A$
535 FOR A=1 TO N*30+20
540 NEXT A
550 RETURN
```

Headlines

Computers are excellent for playing word games. Another program which you MUST write for yourself is Hangman, and for the more ambitious there are the 'pseudo-conversational' programs.

Headlnes is a variation of another computer classic. It prints endless and often amusing rubbish onto your screen. Obviously you can modify the strings as you like; obscene versions are fun, but definitely not for publication!

NOTE

The subroutine at line 90 is not the most efficient way of selecting items from a string, but was chosen because it prints slowly, letter by letter, and also allows the items to be of varying length.

HEADLINES

```
5 LET A$="←TOPLESS←MAD←NUDE←BORING
          ←SEXY←DEAD←FAT←UGLY←"
10 GOSUB 90
15 LET A$="←POTATO←MADMAN←POP-STAR←
          ACTRESS←TEACHER←VICAR←CAT←PRINCE←"
20 GOSUB 90
25 PRINT "←IN";
30 LET A$="←STRIKE←SEX←PEACE←DEATH
          ←RACE←SHOCK←BLACK←WHITE←"
35 GOSUB 90
40 LET A$="←DRAMA←WAR←RUMOUR←RIOT←
          MOVIE←HORROR←SCANDAL←PLEDGE←"
45 GOSUB 90
50 FOR J=1 TO 20
55 NEXT J
60 CLS
65 RUN
90 LET X=INT (RND*8)
95 FOR J=1 TO LEN A$
100 IF A$(J)="←" THEN GOTO 110
105 NEXT J
110 LET X=X-1
115 IF X>-1 THEN GOTO 105
120 PRINT A$(J);
125 IF A$(J+1)="←" THEN RETURN
130 LET J=J+1
135 GOTO 120
```

Royalty Check

This is our contribution to the growing range of educational games. (Ugh! Turn immediately to Bowls! Alternatively, don't TELL anyone it is supposed to be educational.)

You have just written a wonderful computer program to control the world's economy, solve the mysteries of the universe, or, better still, play chess in 1K! Naturally you send it off to 'Your World of Practical Printing Out Today,' who agree to publish it and pay you so much per published page. You are very pleased when you see that your program occupies a complete page, but it has been mixed in with a number of advertisments, so to work out your commission you must calculate the proportion of the page which actually contains your listing.

The 'text' represents your program, and naturally the ads have been blanked out. You have two attempts to work out your payment. The first attempt tests your spatial recognition as well as your math, as you must estimate (or count, until you get good at this) the ratio of program:ads.

If your first attempt is wrong, you will be told the number of ads, and also whether your guess was high or low, so the second attempt is pure mental arithmetic. Remember, however, that you must be accurate to the nearest penny.

NOTE

Line 25 does work, because RND takes a different value each time it is called. This is a very economical 'random branch,' as it uses no numbers or variables, and it tends to produce a ROUGHLY equal division of text/blanks.

incidentally, there is about one chance in 10 billion that this program actually will produce a valid machine code program to control the world's economy.

ROYALTY CHECK

```
5 LET Z=0
10 FOR J=0 TO 14
15 FOR K=0 TO 9
20 PRINT AT J,K;
25 IF RND>RND THEN GOTO 45
30 PRINT "←"
35 LET Z=Z+1
40 GOTO 50
45 PRINT CHR$ (RND*16+28)
50 NEXT K
55 NEXT J
60 LET K=INT (RND*10)+1
65 PRINT
70 PRINT "£";K;"/PAGE"
75 INPUT A
80 LET J=K-K*Z/150
85 LET J=INT (J*100+.5)/100
90 IF A=J THEN GOTO 150
95 LET A$="TOO LOW←."
100 IF A>J THEN LET A$(5 TO 8)="HIGH"
105 PRINT A$;" THERE ARE ";Z;" ADS."
110 INPUT A
115 IF A=J THEN GOTO 150
120 PRINT "NO,£";J
125 CLEAR
130 INPUT A$
135 CLS
140 RUN
150 PRINT "YES"
155 GOTO 125
```

Moving Graphics

The following games all adopt a different solution to the problem of fitting a meaningful screen display, a moving graphics routine and a sensible program, into 1K.

In BOWLS, the playing area is restricted to just the top 10 lines of the screen, which makes perfect sense in the context of the game, and leaves plenty of memory free for everything else. CATCH has possibly the least impressive screen display since you first turned the Timex 1000 on! But, having only 2 characters on screen does mean that they can wander over the entire screen area without any problem. SPACE RESCUE fills (almost) the entire screen with characters, and they all move! How? By using the very concise and economical Timex 1000 command SCROLL.

Bowls

This game is played in much the same way as the 'real' version. Two players (you can be both of them if you like) take turns rolling their bowls toward the jack. The object is not to hit it, but to land closest to it. Player 1 takes bowls 1 and 3; player 2 takes 2 and 4. The winner of each 'end' bowls first in the next end. The Timex 1000 places the jack for you, to prevent the first player having any advantage.

If you hit any other bowl, or the jack, this is dispatched to the gutter and out of play. Clearly this can often be a good thing to do to your opponent's bowls, but if the jack is out of play, the end must be re-played, so try to avoid deliberately hitting the jack except as a last resort. Your own bowls will land in the gutter if they stray off the sides of the green (which is 10 lanes wide: there is 1 lane on each side of the 8 marked mats), or off the end (any roll of distance greater than 30).

Three inputs are required to set up each bowl: Mat number: In the game as listed, mat numbers are deleted as they are used to discourage continuous bowling from the same mat. However, you still CAN bowl from any mat (or indeed from mats 0 or 9, which are not marked), and can easily modify the progam to leave mat numbers on screen.

Bias: This is the most interesting feature of the game. The bowls do not roll in a straight line! Enter any number from minus 5 (severe left bias) to 5 (severe right bias). Obviously an entry of zero will not be accepted (line 65).

Distance: You must judge for yourself how far to roll the bowl, and also the extent to which the distance of the roll affects the bias. Remember that a distance of 30 will reach the end of the screen. Entering 'dist' also sets the bowl rolling. After all 4 bowls have been bowled, enter any NUMERIC value for the next end.

NOTES

The number zero is used a lot in this program: Hence a variable has been set to this value in line 5 to conserve memory.

The moving graphics routine is in lines 85 (to delete the previous bowl position) and 100 (to print the new position).

A$ (CHR$(K + 157)) is the inverse numbered bowl (1–4).

Line 95 deals harshly with all bowls which stray out of range. What problems arise without this line? (There are 3 possibilities: see if you can predict all 3, and then try it.)

BOWLS

```
5 LET Z=0
10 FOR J=Z TO 8
15 PRINT J
20 NEXT J
25 PRINT AT RND*8,RND*10+20;"."
30 FOR K=Z TO PI
35 PRINT AT Z,Z;"MAT←"
40 INPUT L
45 LET A$=CHR$(K+157)
50 PRINT AT L,Z;A$
55 PRINT AT Z,Z;"BIAS"
60 INPUT B
65 IF ABS B>5 OR NOT B THEN GOTO 60
70 PRINT AT Z,Z;"DIST"
75 INPUT S
80 FOR J=Z TO S
85 PRINT AT L,J;"←"
90 IF J>.7*S THEN LET L=B/5+L
95 IF L<Z OR L>9 OR J=31 THEN GOTO 110
100 PRINT AT L,J+1;A$
105 NEXT J
110 NEXT K
115 INPUT B
120 CLS
125 RUN
```

Catch

This is the archetypical moving graphics game. You can elaborate on the basic routine as much as you like to produce a vast variety of games. In this version, the $ character moves randomly about the screen, and you must try to catch it in the 'jaws' of your character, the letter C. As has become almost a standard for Timex 1000 games, the keys 5–8 have been re-defined to control direction of movement (lines 65 and 75). The POKEs in lines 25 and 30 start the timer, and the PEEKs in line 100 read it. The PEEKs in line 85 check the current PRINT position (which is immediately to the right of the C, if you haven't forgotten the semi-colon in line 40) to see if there is a $ character there. Lines 50, 60, 70 and 80 prevent the characters straying off screen. Notice that, as there are only 2 characters on screen, the CLS function is used to avoid 'overprinting' the characters with blanks, as in the previous program.

Space Rescue

Well, we had to include a space game somewhere! In this one, you have to rescue the captives (.) from their guards (*). You score 1 point for each rescue, but lose 5 points every time you run into a guard (at which point the action pauses momentarily for your score to be displayed). By using SCROLL, the only character which actually has to be moved by the program is your ship (X). See lines 20, 25 and 35. (Note that there is no routine to prevent you crashing off screen.) In this program, line 40 sets the PRINT position immediately BELOW the X, and line 45 looks at what is in this position, as before.

The string in line 3 is the arrangement of captives and guards, and you can alter this to make the game easier or harder, as you like. We found the arrangement as shown hard enough, and rarely managed to obtain a positive score! Try omitting the inner 2 asterisks at first.

CATCH

```
 5 LET A=11
 10 LET B=15
 15 LET X=A
 20 LET Y=20
 25 POKE 16436,250
 30 POKE 16437,255
 35 PRINT AT A,B;"$"
 40 PRINT AT X,Y;"C";
 45 LET A=A+1-2*(RND<.5)
 50 LET A=A+(A<0)-(A>21)
 55 LET B=B+1-2*(RND<.5)
 60 LET B=B+(B<0)-(B>31)
 65 LET X=X+(INKEY$="6")-(INKEY$="7")
 70 LET X=X+(X<0)-(X>21)
 75 LET Y=Y+(INKEY$="8")-(INKEY$="5")
 80 LET Y=Y+(Y<0)-(Y>31)
 85 IF PEEK (PEEK 16398+256* PEEK 16399)=
          13 THEN GOTO 100
 90 CLS
 95 GOTO 35
 100 PRINT AT 0,0;INT (1310.6-(PEEK 16435
         +256*PEEK 16437)/50);"SECONDS"
 105 CLEAR
 110 INPUT X$
 115 CLS
 120 RUN
```

SPACE RESCUE

```
1 LET Y=15
2 LET S=0
3 PRINT AT 17,RND*30;"**.**"
20 PRINT AT 9,Y;"←"
25 LET Y=Y+(INKEY$="8")-(INKEY$="5")
30 SCROLL
35 PRINT AT 9,Y;"X"
40 PRINT TAB Y;
45 LET U=PEEK (PEEK 16398+256*PEEK 16399)
50 LET S=S+(U=27)
55 IF U<>23 THEN GOTO 3
60 LET S=S-5
65 PRINT S
70 GOTO 3
```

Code Maker/Code Breaker

This program (Code Maker) will convert any message you enter into an unbreakable code. Unbreakable, that is, unless the recipient of your message uses the second program, Code Breaker.

To use the programs, first enter a number in the range of 1 to 65535, which will act as the SEED for the RAND function (line 15). After you have entered your message, the coded version will appear on the screen, together with the seed number. (If you want to make your message totally secure, even from other people with a copy of this book, delete line 35 of Code Maker, so that the seed number will not appear.) If you have a printer, simply enter COPY, and pass the message on to a friend. If you do not have a printer, of course, you must copy the screen display manually.

Code Breaker works in exactly the same way; simply enter the seed number which precedes the text (or which has been agreed upon by you and the sender privately), and enter the coded message.

NOTES

Several of the programs in this book have used the RAND function, but none, so far, have used RAND X (where X is a KNOWN number in the range of 1 to 65535).

The Timex 1000's random number generator does NOT produce purely random numbers, but selects numbers from a pre-defined sequence. The purpose of the RAND function is to fix the starting point along this sequence. RAND Ø, or simply RAND, will select an indeterminate starting point, but RAND followed by any number will select a SPECIFIC point, so that although the numbers produced are still 'random' (in the sense that they have no apparent relationship to each other), the same sequence of numbers will be produced each time the program is used.

In Code Maker, each letter of your message is replaced by a 'random' letter (line 45). However, this 'randomness' depends on the RAND seed, and provided that the same seed is used in Code Breaker, the original message can be obtained.

The 2 programs are so similar that you need not enter both of them from scratch. Judicious use of the EDIT key can save you a lot of typing. To convert Code Maker to Code Breaker:

Delete line 35
EDIT lines 45, 5Ø and 55

CODE MAKER

```
5 PRINT "ENTER CODE NUMBER"
10 INPUT X
15 RAND X
20 PRINT "ENTER MESSAGE"
25 INPUT M$
30 CLS
35 PRINT X;CHR$ 0;
40 FOR J=1 TO LEN M$
45 LET A=CODE M$(J)+INT (RND*26)
50 IF A<26 THEN LET A=0
55 IF A>63 THEN LET A=A-26
60 PRINT CHR$ A;
65 NEXT J
```

CODE BREAKER

```
5 PRINT "ENTER CODE NUMBER"
10 INPUT X
15 RAND X
20 PRINT "ENTER MESSAGE"
25 INPUT M$
30 CLS
40 FOR J=1 TO LEN M$
45 LET A=CODE M$(J)-INT (RND*26)
50 IF A<1 THEN LET A=-26
55 IF A<38 THEN LET A=A+26
60 PRINT CHR$ A;
65 NEXT J
```

Guillotine

This is an early computer game, given a new lease on life by the incorporation of a graphics display.

The computer thinks of a number in the range 1 of to 1Ø0, and you have 6 guesses in which to identify the number. After each guess, the computer indicates whether its number is higher or lower than your entry.

The graphic display shows an executioner who raises the guillotine blade one notch with each incorrect guess. After the sixth wrong guess, the blade will fall, and your head will roll across the screen!

NOTES

The graphics in this game are quite complex, so I have used a new symbol (#) to indicate an INVERSE SPACE. Refer to the listing conventions to make sure you enter the other PRINT lines correctly.

Because the graphics occupy so much of the program, the 'feedback' after each guess is restricted to a single 'greater than' or 'less than' symbol (lines 11Ø–12Ø). After a correct guess, or an 'execution,' the mystery number is revealed. Press 'Newline' for another game.

GUILLOTINE

```
5 LET A=INT (RND*100)+1
10 FOR F=A/A TO 12
15 PRINT "#←←←#"
20 NEXT F
25 PRINT"#####←↑2121↑"
30 PRINT AT 8,7;"↑85↑"
35 PRINT TAB 6;"↑8HH5↑"
40 PRINT TAB 6;"↑7HG1↑"
45 PRINT "#-O-#←←↑58↑"
50 FOR F=-5 TO 7
55 PRINT AT 10,5;"/"
60 PRINT AT F,A/A;"←←←"
65 PRINT "#---"
70 PRINT "#←←/"
75 PRINT "#←/←"
80 PRINT "#/←←"
85 IF F>A-A THEN GOTO 125
90 PRINT "#←"
95 PRINT AT 10,5;"-"
100 INPUT G
105 IF A=G THEN GOTO 150
110 LET A$=">"
115 IF G>A THEN LET A$="<"
120 PRINT TAB 12;A$
125 NEXT F
130 PRINT AT 11,2;"↑6↑-"
135 FOR F=A/A TO 8
140 PRINT AT 13,F;"←O"
145 NEXT F
150 PRINT AT 11,12;A
155 INPUT A$
160 CLS
165 RUN
```

Bowling

I am absolutely hopeless at bowling! Every time I let go of the ball, I just know it is heading straight for the gutter. How I wish I could run down the lane and adjust its direction. . . .

That is precisely what you CAN do in this game, but that still does not make it any easier!

The balls are released automatically from the left of the screen, and head towards the pins on the right. Use keys "6" and "7" to control the direction of the balls. When you hit a pin, it will disappear from the screen, but it will not disturb any of the others, so you must actually hit all 10 pins. Therefore you are allowed 4 bowls per 'frame'—you will need all 4 if you are to score 10 points.

After the fourth bowl, press 'Newline' for the next frame.

NOTES

Enter line 10 carefully. These are the PLOT coordinates (in CODE form) of the 10 pins.

Lines 20 to 50 PLOT the pins on screen. Notice that the pins do NOT form a perfect triangle; no pin is directly behind any other. Also, no pin is in line with the starting position of the ball, so you MUST use the direction keys to score.

BOWLING

```
10 LET A$="W2S.O;W/S=K+W>O)S?W£"
20 FOR J=1 TO 10
30 PLOT CODE A$(1),CODE A$(2)
40 LET A$=A$(3 TO )
50 NEXT J
60 FOR J=1 TO 4
70 PRINT AT 0,0;"BOWL:";J
80 LET Y=19
90 FOR X=10 TO 62
100 LET Y=Y+(INKEY$="7" AND Y<43)-(INKEY$="6"
        AND Y>0)
110 PLOT X,Y
120 UNPLOT X,Y
130 NEXT X
140 NEXT J
150 INPUT A$
160 RUN
```

Missile Attack

This is a VERY fast-action moving graphics game. You operate the laser gun on the left of the screen, and the enemy missiles fly rapidly towards you from the right.

Use keys "6" and "7" to move your laser up and down, and "8" to fire.

The object of the game is to shoot down as many missiles as possible before either 10 missiles escape being shot down or a single missile manages to hit your laser gun.

A reminder of the current score is flashed on the screen every time you score a hit. The first figure shown is your total score, and the second is the the number of missiles which have passed you successfully.

NOTES

Line 50 is the 'laser beam.' Notice that it does not extend right across the screen, so you must wait for missiles to come into range. To enter line 50, use 10 'exponentiation symbols' (**), not asterisks. (The result is the same, but the exponentiation symbols use less memory).

Line 35 skips the INKEY$ routine when you are not pressing any key. Remove this line if you want to slow the game down slightly.

MISSILE ATTACK

```
5 LET L=0
10 LET S=L
15 LET P=11
20 LET X=INT (RND*21)
25 LET Y=30
30 PRINT AT P,0;"O-";AT X,Y;"<"
35 IF INKEY$<"6" THEN GOTO 60
40 LET P=P+(INKEY$="6" AND P<21)-(INKEY$="7" AND P>0)
45 IF INKEY$<>"8" THEN GOTO 60
50 PRINT AT P,2;"**********************"
55 IF P=X AND Y<21 THEN GOTO 85
60 LET Y=Y-2
65 IF Y=0 THEN LET L=L+1+9*(P=X)
70 IF L>=10 THEN GOTO 99
75 CLS
80 GOTO 20+10*(Y<>0)
85 LET S=S+1
90 PRINT AT X,Y;"*";S;"*";L
95 GOTO 20
99 PRINT "GAME OVER","SCORE ";S
100 INPUT A$
105 RUN
```

Introduction To Part II

The 16K RAM-pack is an invaluable add-on to either the Timex 1000 or Sinclair ZX81, if it is used wisely. Too many "16K" programs have been published which are really nothing more than poorly written 1K or 2K programs that need the RAM-pack because the writer has not bothered to squeeze them into the standard memory space.

This book is one of the very few which is dedicated exclusively to REAL 16K programs. If you want programs which you can key in and run in a few minutes, then look elsewhere!

A few of the programs have required 'Americanization' (although they were all written in England), and all will work on either Timex or Sinclair computers. 'Coin Analysis' has, of course, been adapted to handle dollars and cents instead of the original pounds and pence, but if you want to play 'Pontoon,' you must still gamble in pounds. (We have casinos in England, too, you know!)

Roger Valentine
London, Nov. 1982

Listing Conventions

The following conventions have been used throughout this section. Please study them carefully before keying in the programs.

I = The letter I
1 = The number one
O = The letter O
Ø = The number zero
' = The quote image (SHIFT Q)
← = Space (WHERE ESSENTIAL AND NOT OBVIOUS)
= Inverse space
↑ = All items BETWEEN 2 up-arrows are to be entered in GRAPHICS
UNDERLINING = All underlined items are to be entered in INVERSE

Note: These conventions are as used in Part I, except for the # character meaning inverse space.

Chapter 8
Graphics Programs

Tarot

Tarot cards are also known as "the devil's picture book,' and a picture book is precisely what this program is. Twenty-two different pictures are held in memory and selected at random for display.

Here immediately is a problem and one which will come as a surprise for those of you who thought that buying a 16K RAM-pack would forever abolish the dreaded error-code 4 (out-of-memory).

A screen-full of graphics occupies 22x32 = 704 bytes, so to hold 22 alternative displays would require 22x704 bytes, which is over 15K! Obviously a shorthand method of defining graphics is required. One such method would be to use the 'Data Print' technique explained in Part I. This would compress each screen-full of graphics into a string of, on average, 100–150 characters in length, so if you want to create a 'picture book' with over 100 screen 'pages,' this would be the method to use.

However, any intricate method of compressing data is going to require a correspondingly intricate method of de-coding that data into a display, and 'Data Print' has the disadvantage that it takes a long time to actually produce each picture. Therefore for 'Tarot,' we have used a totally different technique, and the program is really 2 programs in 1, as the routine for creating the graphics is actually a free-standing routine which you may like to incorporate into other programs of your own. For that reason, the routine is described in full, but first a description of 'Tarot' itself.

Lines 30–61 set up the array of card names. Don't try to save time by entering these in immediate mode, as they are manipulated during the course of the program, so must be re-set for each deal (see line 360). Lines 110–150 are the standard 'input-verification routine,' which you will see used time and time again. (Much better than 'INPUT T'.)

Line 165 is a POKE worth memorizing: it increases the screen display by 2 lines, so that 'PRINT AT 23,. . . . .' becomes valid. (Address 16418 contains the number of lines in the lower part of the screen. Normally it contains '2,' allowing room for the INPUT prompt and the INPUT itself, but you can safely change it to 0 PROVIDED that you change it back to 2 before any INPUT is required.)

TAROT

```
10 REM TAROT
20 GOTO 1000
30 DIM C$(22,20)
40 LET C$(1)="LE BATELEUR:"
41 LET C$(2)="LA PAPESSE:"
42 LET C$(3)="L.IMPERATRICE:"
43 LET C$(4)="L.EMPEREUR:"
44 LET C$(5)="LE PAPE:"
45 LET C$(6)="L.AMOUREUX:"
46 LET C$(7)="LE CHARIOT:"
47 LET C$(8)="LA JUSTICE:"
48 LET C$(9)="L.ERMITE:"
49 LET C$(10)="LA ROUE DE FORTUNE:"
50 LET C$(11)="LA FORCE:"
51 LET C$(12)="LE PENDU:"
52 LET C$(13)="LA MORT:"
53 LET C$(14)="TEMPERANCE:"
54 LET C$(15)="LE DIABLE:"
55 LET C$(16)="LA MAISON DE DIEU:"
56 LET C$(17)="L.ETOILE:"
57 LET C$(18)="LA LUNE:"
58 LET C$(19)="LE SOLEIL:"
59 LET C$(20)="LE JUGEMENT:"
60 LET C$(21)="LE MONDE:"
61 LET C$(22)="LE MAT:"
70 CLS
80 PRINT TAB 12;"TAROT";AT 3,3;"YOU MAY SELECT UP TO 22
      CARDS";AT 5,0;"OF THE MAJOR ARCANA."
90 POKE 16418,2
100 PRINT AT 10,0;"HOW MANY CARDS DO YOU WANT?"
110 INPUT N$
115 IF N$="" THEN GOTO 110
120 FOR J=1 TO LEN N$
130 IF N$(J)<"0" OR N$(J)>"9" THEN GOTO 110
140 NEXT J
150 LET T=VAL N$
154 IF T>22 THEN GOSUB 800
155 IF T<1 THEN GOTO 100
160 DIM P(T)
165 POKE 16418,0
170 FOR J=1 TO T
180 PRINT AT 23,5;"PRESS C TO DEAL A CARD"
185 IF INKEY$<>"C" THEN GOTO 185
190 LET C=INT (RND*22)+1
200 IF CODE C$(C,1)=0 THEN GOTO 190
210 LET P(J)=C
220 LET C$(C)="←"+C$(C)
260 GOSUB 600
270 NEXT J
280 PRINT AT 23,2;"PRESS L TO LIST YOUR CARDS"
290 IF INKEY$<>"L" THEN GOTO 290
```

Lines 170–270 select the cards. As a card is used, a blank is inserted in front of its name (line 220) to signify this. This is clearly an inefficient way of avoiding duplication, and would NOT be recommended if a large number of cards were to be dealt. However, for 22 (or fewer) cards it does not cause any noticeable delay.

Subroutine 600 prints the cards, with lines 700–760 printing the name in inverse lettering. Subroutine 800 is trivial, but it is a nice form of 'idiot-proofing' which lets the user KNOW he is an idiot!

Graphics Entry Routine

Line 1000 sets up an array of the 22 pictures, each 20 lines deep (not 22 as the bottom 2 lines are reserved for the title of the picture) and 24 characters long. This would seem to restrict the pictures to the 24 leftmost columns of the screen, but in fact the first character of each string is a TAB number (0–9; line 1035 ensures this), so that each picture can spread across the entire screen, provided that any line requiring graphics at the right of the screen does not require any on the left. This may sound a little complicated, but by the time you have entered the 22 pictures for 'Tarot,' you will see how well it works. It actually saves about 3.5K; try changing line 1000 to DIM P$(22, 20, 32) and see what happens!

Line 1030 is an 'ultra-shorthand' for entering a blank line: just press newline. The routine at 2000 is called by line 1035 whenever a line does not begin with a tab number. Usually line 2000 simply rejects the input and returns to line 1025 for another. However, if you enter X, the previous screen-line will be deleted, to make error correction simple (lines 2010–2015).

Line 1040 prints each line of the picture as you go along, and lines 1050–1080 give you the option to reject the entire picture when it is complete.

Running Instructions

If you were thinking that 'Tarot' was a nice short program which you could key in and run in a few minutes, forget it! Just as the program was written in 2 sections, running it involves 2 separate processes as well. Before you call up all those pretty graphics with the main program, you must first enter them, and this, I am afraid, is a longer job than typing the program itself! Don't worry, though, the results are well worth the effort, so arm yourself with a strong pot of coffee and off you go . . . . (You only have to do this once, by the way, not every time you use the program.)

TAROT *(Continued)*

```
300 CLS
310 FOR J=1 TO T
320 PRINT AT J-1,0;J;TAB 4;C$(P(J))
330 NEXT J
340 PRINT AT 23,0;"PRESS N/L TO DEAL AGAIN"
350 IF INKEY$<>CHR$ 118 THEN GOTO 350
360 GOTO 40
600 CLS
610 FOR K=1 TO 20
620 PRINT AT K-1,VAL P$(C,K,1);P$(C,K,2 TO )
630 NEXT K
700 LET A$=""
710 FOR K=2 TO 20
720 IF C$(C,K)=":" THEN GOTO 750
730 LET A$=A$+CHR$ (CODE (C$(C,K))+128)
740 NEXT K
750 LET C$(C,K)="←"
760 PRINT AT 21,(32-LEN A$)/2;A$
770 RETURN
800 PRINT AT 10,0;"←←←←THERE ARE ONLY 22 CARDS"
810 FOR J=1 TO 100
820 NEXT J
830 LET T=0
840 RETURN
1000 DIM P$(22,20,24)
1005 FOR J=1 TO 22
1010 CLS
1015 PRINT AT 21,0;"ENTER GRAPHIC NO.";J
1020 FOR K=1 TO 20
1025 INPUT P$(J,K)
1030 IF P$(J,K,1)="←" THEN LET P$(J,K)="0"
1035 IF P$(J,K,1)<"0" OR P$(J,K,1)>"9" THEN GOTO 2000
1040 PRINT AT K-1,VAL P$(J,K,1);P$(J,K,2 TO )
1045 NEXT K
1050 PRINT AT 21,0;"O.K.?←←←←←←←←←←←←←"
1070 IF INKEY$="N" THEN GOTO 1010
1080 IF INKEY$<>"Y" THEN GOTO 1070
1090 NEXT J
1100 CLS
1110 PRINT "NOW SAVE 'TAROT'"
1120 PRINT AT 2,0;"(PRESS S WHEN TAPE IS RUNNING)"
1130 IF INKEY$<>"S" THEN GOTO 1130
1140 SAVE "TAROT"
1150 RAND
1160 GOTO 30
2000 IF K=1 OR P$(J,K,1)<>"X" THEN GOTO 1025
2010 LET K=K-1
2015 PRINT AT K-1,0;"←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
2020 GOTO 1025
```

NOTES

Once you have typed in the program, please SAVE a copy on tape before RUNning. You are strongly urged to do this ALWAYS, and it is particularly important with programs using POKE statements, as a simple mistake could wipe out the entire program.

When you RUN 'Tarot' for the first time, you will be led straight into the graphics entry routine. The 22 pictures are all listed below. Refer to the listing conventions (page 74) if you have any problems understanding these rather cryptic lines. Believe it or not, this was the LEAST confusing method we could devise for describing Timex 1000 graphics in print. (You should have seen some of the ideas we rejected!) Take care with the spaces and inverse spaces where they occur within graphic strings, and notice also that some cards (L'Empereur and Justice) use 'normal' Timex 1000 characters like "O" and ":" as well as graphics characters.

Actually this process is a lot simpler than it appears from the graphics listings. You will see the pictures being compiled on screen line by line, so any errors will be easy to spot. Some of the cards are pictured in Appendix 1 (pages 162–163) so you can see what the end results should look like. (You may, of course, custom design your own tarot graphics. Save your tape of the 'bare' program so that you can design as many different sets of cards as you like.)

When you have entered all 22 pictures, the 'SAVE' routine will be called automatically. Make sure that your tape is long enough—we normally use C30 tapes for 16K programs containing large amount of data. The program will then auto-run each time it is loaded, and the graphics will be retained. DO NOT press BREAK and then RUN or CLEAR, or you will lose the graphics (although they will be retained on the tape, of course, so you only have to re-load). If you do press BREAK, then the safe way to re-start the program is by entering 'GOTO 40.'

Tarot Card Input Listings

1:Le Bateleur

```
9↑4←←374TY↑
9↑5←←Y←T431↑
8↑3W←←←7←21↑
8↑2EW←←←66↑
9↑5RW←8##5↑
9↑5←RW2##1↑
9←←↑2#WQW64↑
9←←←↑##GG##4↑
9←←←↑8#HH##W↑
9←←←↑8#HH###4↑
9←←←↑8#HH###R4↑
9I←←↑8#HH###←R4↑
9←←←↑8#HH##5←21↑
9←←←↑8#HH##5↑
9←←←↑8#HH##5↑
9←←←↑8#HH##5↑
9←←←↑8#HH##5↑
9←←←↑8#HH##5↑
9←←←↑8#HH##5↑
9←←←↑8#HH##5↑
```

2:La Papesse

```
9←←←←←←↑6Q#6↑
9←←←←←↑QHHHH5↑
9II←←←↑8HHHHH#↑
9←←←←↑ER777E75↑
9←←←←↑31R1←2E←8↑
9←←←←↑#HHHHHHH#↑
9←←←↑3R77E7R77E4↑
9←←←↑8R12E←R12E5↑
9←←←↑#FFFFFFFFF5↑
9←←↑8564←3648←←5↑
9←←↑5568←Y428←←5↑
9←↑3152#317←8←←8↑
9←↑8←525←1←←5←←8↑
9←↑5←5←YT←←←5←←8↑
9↑T←←5←661←←5←←24↑
8↑31←←82341←Q5←←←Y4↑
8↑8←←66W←←3Q#5←←←←2Y↑
8↑8QGHH#######←←←←←Y↑
8↑QHHHH###GHHH5←←←6E1↑
7↑QHHHHH###HHHH#6QGH5↑
```

3:L.Imperatrice

```
new line
9←←←←←↑34↑
6III←←←←←↑3##4↑
9←←←←←↑QHHW↑
9←↑8W←3#FF#4←Q5↑
9←↑8HWQGHHGWQH5↑
9←↑2F↑00000000↑F1↑
9←←↑2FFFFFFFF1↑
9←←↑QE←←←←←84↑
9←↑Q#57743778W↑
9←↑##12W←←Q18#W↑
9↑3##4←←←←←8##↑
9↑###5←←12←←###5↑
8↑8####←2661←####↑
9↑####W4←←3Q###5↑
9↑8####2771####5↑
9#####←←←#####
8↑6Q###Y←←←←T8###W↑
6↑37←###58←←←←58####74↑
5↑31←←####←Y66T←####5←24↑
```

4:L.Empereur

```
9←←←←←IV
new line
9←←←←←←←↑36666↑
9←←←←←←↑6QGGHHH#↑
9↑3←←←6Q↑0↑HHH2HHH5↑
9↑8#6QHHH↑0↑HHH2HH5↑
9←↑####HHH↑0↑HHH2#↑
9←↑2#2R#FHH↑0↑HHHQ↑
9←←↑837TR#HH↑0↑HH5↑
9←←↑T←7T8##FH↑0↑H5↑
9←↑31←←←8E###H#↑
9←↑8QY←←#3#T###←2#W4↑
9←←↑8#←8####T##63Q#5↑
9←←↑6Q#######T#####5↑
9←↑8##########T####5↑
9↑8#########7770##1↑
8↑8########7↑
8↑######1↑
7↑8####E↑
7↑27R71↑
```

5:Le Pape

```
9←↑6#6↑
9↑QHHHW↑←←←←V
7↑3QHHHHHW↑
6↑3GHHHHHHHW←←←←←3#4↑
6↑T7E77E77E7W←←←←R#E↑
6↑52E←2E←2E←←5←←←←#↑
6↑#GGGGGGGGGGW←←←W6#6Q↑
5↑8HHHHHHHHHHH#←←←1←#←2↑
4↑37R77R77R77R775←W66#66Q↑
4↑8←R1←R1←R1←R1←5←1←←#←←2↑
4↑2###E77777777#←←←W6#6Q↑
5↑8##←3774←T71#←←←1←#←2↑
5↑8##←3EW←8#E5#←←←←#↑
5↑##55←←618268#←←←←#↑
4↑8##W55←←324←85←←←←#↑
4↑8###88←36#6←##←←←←#↑
4↑##5###W##7#W##←←←←#↑
4↑###←8##EQ#WR←W6←←←#↑
4↑###←←8######←←←W6←←#↑
4↑###←←←2####←←←←←W6#↑
```

6:L.Amoureux

```
9←←←←←VI
new line
new line
new line
new line
9←←←←←←←←←←←←←↑376↑
9←←←←←←←←←←←↑3745←←Y↑
7↑T7677Y←←←←←←←T←YT←←8↑
5↑T7←←3GG4Y743GGT←←←←←←5↑
4↑8←←←3GHHWY←TQHHG4←←←←T↑
4↑5←←←←7FFF172FFF7←←←←←24↑
4↑2Y←←←←←31T←326←←←←←←←8↑
4↑31←←←←←←W267Y2W←←←←←←←5↑
5↑7R←←←636#W##E←76←6Y←T↑
6↑24←8←13131Y3←←←7←←7↑
7↑271←←271←3#↑
new line
new line
new line
new line
```

7:Le Chariot

```
9←←←←←←←←←←←←←↑4←←4↑
9←←←←←←←←←←←←↑Q#6Q5↑
9←←←←←←←←←←←←↑QER#ER↑
9←←←VII←←←←←↑Q#←←←←8↑
9←←←←←←←←←↑Q##←←←←8↑
9←←←←←←←←←↑8##←←←6←←5↑
6↑W←←W←←←←←←←##5←←27←←5↑
6↑#WQ#666←←←←8###8←←←←58↑
6↑#R#E8##W4←3###5←5←←←824↑
6↑52E1←8###W2####←8←←←←55↑
6↑5←←←←←8###W8###5←#434←8↑
6↑5←←←←←←8###5###5←#842←8↑
6↑W←←←3E←←8###8##←←5←R661↑
6↑25←←E←←5←8##5##←←5↑
7↑5←8←←8←←←8##8#←8↑
7↑5←8←←#1←←###5#←8↑
6↑85←8←8#←←←8###←←8↑
6↑85←Q6R#←←←←8##5↑
6↑25←2E←#←←←←←###↑
7↑R##1←8←←←←←8##↑
```

8:La Justice

```
new line
9←←←←←←↑Q#W↑
8##################
8:←←←←←←←#←←←←←←←:
8:←←←←←←←#←←←←←←←:
8:←←←←←←←#←←←←←←←:
8:←←←←←←←#←←←←←←←:
8:←←←←←←←#←←←←←←←:
8:←←←←←←←#←←←←←←←:
2VIII←←:←←←←←←←#←←←←←←←:
8:←←←←←←←#←←←←←←←:
8:←←←←←←←#←←←←←←←:
8:←←←←←←←#←←←←←←←:
6↑R###E←←←←←#←←←←←R###E↑
7↑R#E←←←←←←#←←←←←←R#E↑
8↑7←←←←←←←#←←←←←←←7↑
9←←←←←←←#
9←←←←←←←#
9←←←←←←←#
9←↑6##########6↑
```

9:L.Ermite

```
new line
9←←←←←←↑TY↑
9←←←←←↑8345↑
9←←←←←↑3##4↑
9←←←←↑3####4↑
9←←←↑3######4↑
9←←↑3########4↑
9←↑3##########4↑
9↑3############4↑
9←#←←↑8←3←←5←←#↑
9←#←←↑83#W←5←←#↑
9←#←←↑88##←5←←#↑
9←#←←↑8←R1←5←←#↑
9←#←←↑8←HH←5←←#↑
4XIV←←←#←←↑8←HH←5←←#↑
9←#←←↑8←HH←5←←#↑
9←#←←↑8←HH←5←←#↑
9←#←←↑8←HH←5←←#↑
9←#←←↑8←HH←5←←#↑
9←#############
```

10:La Roue de Fortune

```
9←←←←↑67RE76↑
9←←↑37←←85←←74↑
9←↑374←←85←←374↑
9←↑5←24←85←31←8↑
9↑8←←←248531←←←5↑
9↑T←←←←2QW1←←←←Y↑
9###############
9↑5←←←←T#8Y←←←←8↑
9↑8←←←T←85←Y←←←5↑
9←↑5←T←←85←←Y←8↑
9←↑8T←←←85←←←Y5↑
9←←↑Y←←←85←←←T↑
9←←←↑76←85←67↑
9←←←←←↑7RE7↑
9←←←←←←↑85↑
9←←←X←←↑85↑
9←←←←←←↑85↑
9←←←←←←↑QW↑
9←←←←←↑T85Y↑
9←←###########
```

11:La Force

```
new line
9←←←←←XI←←←←←↑T4↑
9←←←←←←←←←←←↑TT4↑
9←←←←←←←←←←←↑T←2↑
9←←←←←←←←←←←↑26↑
9←←←←↑33←←←←←←Y6↑
9←←↑3YYYY←←←←←←←76↑
8↑6666WYYY1←←←←←←←←Y↑
8↑R##QYYYY1←←←←←←←←←5↑
9↑2##YYYYY←←←←←←←←←5↑
8↑R##74YYYY←←←←←37667↑
8↑3YYY24WWW66###W↑
8↑3YY1TQ##########W↑
7↑3YYYY8###########5↑
7↑3YYYY#####77#####4↑
8↑2#E####7←←Q######↑
9↑1#58#7←←←8###R##W↑
9↑1#58#←←←←←R##←7##W↑
9↑1#58#←←←←←←##1←←#E↑
7↑3QE6#1←←←←3Q#1←←6QE↑
```

12:Le Pendu

```
new line
9###############
9#←←←←←←:↑66↑←←←#
9#←←←←←↑85↑←←←←#
9#←←←←←↑85↑←←←←#
9#←←←←←↑85←5↑←←#
9#←←←←←↑Q###5↑←←#
9#←←←←↑8E85↑←←←←#
9#←←←←↑8585↑←←←←#
9#←←←←↑8W85↑←←←←#
9#←←←←↑8##5↑←←←←#
3XII←←←#←←←#####←←←←#
9#←←←↑Q####W↑←←←#
9#←←↑81####25↑←←#
9#←←↑85####85↑←←#
9#←←↑2######1↑←←#
9#←←←←←↑QW↑←←←←←#
9#←←←←↑8##↑←←←←←#
9#←←←←↑8##↑←←←←←#
9#←←←←←←←←←←←←#
```

13:La Mort

```
new line
9←←←↑3T7777Y6↑
3XIII←←←←↑T1←←←←←←Y4↑
9←↑T←←←←←←←←←24↑
9↑T←←←←←←←←←←8↑
8↑8←←←←←←←←←←←5↑
8↑537←←←←←←4←←←8↑
7↑3Y1←←←←←←←24←←←5↑
7↑813#Y←←←T#4←24←←←5↑
7↑8←##61←←YQ#←←261←5↑
7↑8←2#1←←←2#1←←←←←5↑
8↑Y6←←←3←←←Q7←←←31↑
8↑8←Y←3#W←3#4←←←←T↑
8↑84←1←←←←12#6←←←5↑
9↑8##←←←←←←#1←←T↑
9←↑REYYYYYY#←←T↑
9←←↑5←←←←←←6T↑
9←←↑5←←←←←3Q#5↑
9←←↑24←←←31←6#↑
9←←←↑2666Y←←6QW↑
```

14:Temperance

```
9←←←←←XIV
new line
9←←←←←←←←←←←↑64↑
9←←←←←←←←←←↑T←24↑
9←←←←←←←←←←↑246T↑
9←←←←←←←←↑84←3#H#635↑
9←←←←←←←←↑8#6:#H#H#5↑
9←←←←←↑DDA8##:#H#H#5↑
9←←↑DAAAAAA8#7:#H#H#5↑
9↑DAAAS←←←81←2#H#725↑
7↑66666←←←←←←←←←7↑
7↑2###1↑
8↑8#5↑
7↑3GGG43Y↑
7↑#####1←5↑
6↑8HHHHH531↑
7↑#####Y1↑
7↑8HHH5↑
7↑3###4↑
7↑77777↑
```

15:Le Diable

```
new line
9↑366←←←←←←←664↑
7↑6Q####W←←←Q####W6↑
6↑Q#######W←←Q#######W↑
5↑3##E7R####58####E7R##4↑
5↑Q#E←←←R###WQ###E←←←R#W↑
5↑#E←←34←R#7←←7#E←34←←R#↑
5↑#5←←2#6654←←3866#1←←8#↑
5↑85←←←2#E←2431←R#1←←←85↑
5↑2#←←←←#5TY←←TY8#←←←←#1↑
6↑#6←←←8←Y#12#T←5←←←6#↑
7↑7←←←←5←←←←←8←←←←7↑
9←←←↑Y←2431←T↑
9←←←↑24←←←←31↑
9←←←←↑Y←77←T↑
9←←←←↑2Q34W1↑
9XV←←←↑5YT8↑
9←←←←←↑8YT5↑
9←←←←←↑8YT5↑
9←←←←←←↑YT↑
```

16: La Maison de Dieu

```
9←←←←←←←←←←←←←←←←←←←↑4↑
9←←←←←←←←←←←←←←←←←←←↑T↑
9←←←↑85858585←←←←←←←T↑
9←←←←######←←←←←←←↑T↑
9←←←←######←↑4←←←←T↑
9←←←←↑E#R#E2T24←←T↑
9←←←←↑W2←11T←←24T↑
9←←←←↑#W3←488←←2↑
3XVI←←←←←←←↑###WQ#←6↑
9←←←←↑##HH##↑
9←←←←↑##HH##←←5↑
9←←←←######←↑5←6↑
9←←←←↑#H###H#↑
9←←←←↑#H###H#←←6↑
9←←←↑3######4←←8↑
9←←↑3########4↑
9←←↑####12####←8↑
9←←↑###5←←8###↑
9←←↑###5←←8###←←6↑
9←←↑###5←←8###↑
```

17:L.Etoile

```
5*
2*←←*←←*←←←←XVII
3*←*←*
4***
1**********←←←←*
4***←←←←←←←*
3*←*←*←←*←←←*←←←*
2*←←*←←*←←←*←←*←←*
5*←←←←←←*←*←*
9←←←←←***
9**************
9←←←←←*****←←←←←←←←
9←←←←←*←*←←←←←←*←←*←←*
9←←←*←←*←←*←←←←←←*←*←*
9←←*←←←*←←←*←←←←←←***
9←←←←←*←←←←←←**********
9←←←←←*←←←←←←←***
9←←←←←←←←←←←←←←*←*←*
9←←←←←←←←←←←←←←*←←*←←*
9←←←←←←←←←←←←←←←*
```

18:La Lune

```
9←←←←←←↑66↑
9←←←←←↑0###W↑
9←←←←↑2←←7R#4↑
9←←←←←←←←↑#5↑
9←←←←←←←←↑#5↑
9←←←←←←←↑3E↑
9←←←←←←←↑1↑
newline
3XVIII←←←←←↑3124↑
9←↑4←←←←YT←←←←3↑
9←↑Y'Y←←←0W←←←TT↑
9←←↑R#W0##W0#E↑
9←↑2473####4731↑
9←↑2T7Y####T7Y1↑
9←←↑271T##Y271↑
9←←↑367T85Y764↑
9←←↑8←T3##4Y←5↑
9←←←←←↑RE↑
9←←←←←↑TY↑
newline
```

19:Le Soleil

```
newline
9←←↑3←←←←5←←←3↑
9←↑Y←Y←←←5←←31T↑
9↑4←Y←Y←←5←31T←3↑
9↑24←Y←Y←531T←31↑
8↑2424←Y←111T←3131↑
8↑42424←6776←31313↑
8↑242437←←←←743131↑
8↑Y243167437624 31T↑
2XIX←←←←↑Y312←6←←6←124T↑
7↑6668←←21←←21←←5666↑
9↑38←←←3←←4←←←54↑
8↑3124←←←12←←←3124↑
8↑13124←2661←31242↑
8↑313126←←←←612424↑
8↑1313147667←4Y242↑
8↑3131T31444Y24Y24↑
9↑31T31T←524Y24Y↑
9↑1T31T←←5←24Y241↑
9↑231T←←←5←←24Y2↑
```

20:Le Jugement

```
newline
9←←←←←←←←←←←←↑30#↑
9←←←←←←←←←←←↑0##E↑
9←←←←←←←←←←↑3###1↑
9←←←←←←←←←↑0##E1↑
9X%←←←←←←↑30##1↑
9←←←←←←←↑6###1↑
9←←←←←↑30##E1↑
9←←←←↑0###7↑
9←←↑30##E1↑
9←↑6###EAAA#AAA↑
7↑66####1←AAA#AAA↑
1↑3666#######1←←AAA#AAA↑
1↑86R#######1←←←#######↑
2↑#WR#####E←←←←AAA#AAA↑
2↑8#WR####←←←←←AAA#AAA↑
3↑R#WR##5←←←←←AAA#AAA↑
4↑R#WR#5↑
5↑R#WR↑
6↑7#0↑
```

21:Le Monde

```
9←←←←↑343#←34↑
9←←←↑3##6#6##4↑
9←←←↑##7###7##↑
9←←←↑#1←2#1←2#↑
9←←←↑5←←←7←←←8↑
9←←←←←←↑374↑
9←←←←←↑31←24↑
9←←←←←↑5←←←Q↑
8↑8W←←←8←←←T←5←←←Q5↑
6↑T4QW3Y←5←←T←←8←T4QW3Y↑
6↑58##588←←24←←←558##58↑
6↑88##558←←←Y←←←588##55↑
7↑Y##T←←5←←24←8←←Y##T↑
8↑##←←←8←←31←5←←←##↑
8↑85←←←←531←8←←←←85↑
9←←←←←↑81←←5↑
9←←←←←←↑Y←T↑
9←XXI←←↑374↑
9←←←←←←↑2←1↑
newline
```

Un-numbered:Le Mat

```
newline
newline
newline
9←↑Q##6←←T77Y↑
9↑Q####58←←←←Y↑
8↑3#####WT←←←←24↑
8↑Q######W←←←←←Y↑
7↑8##ER####←←TY←←5↑
7↑Q#E←2####←31←Y4Y↑
6↑Q#1←←←#######←←←26W↑
50←←←←↑3#W6←←64W↑←←←←←0
9←↑##161T62#5↑
9←↑#Y←28←1←85↑
9←↑#4←←4Y←←Q5↑
9←←↑R←4213←#↑
9←←↑8←8775←T↑
9←←←↑Y←77←T↑
9←←←↑8Y←←T5↑
9↑6660Q473W666↑
6↑Q########W0########W↑
```

Graphics

If I were to mention the words 'sketch-pad program,' I am sure that many readers would skip over this section and go on to something more original. But 're-inventing the wheel' is not always as fruitless a task as it is made out to be, and sometimes it is possible to find a new and improved solution to an old problem by approaching it from a new angle.

There are 2 aspects to screen drawing; positioning the cursor and defining the actual graphic character to be 'drawn.' Most sketch-pad programs give top priority to the first of these, but make changing the character difficult or even impossible. Consequently they tend to produce 'line drawings' which fail to make the most of the Timex 1000's already limited graphic capabilities. This program gives equal priority to cursor position and character changes, by operating in 4 totally distinct 'modes' (or 5, if you include the SAVE option).

In CONTROL mode, a flashing X cursor can be moved about the screen at will using the 'arrow' keys 5 to 8. To switch to 1 of the 3 DRAW modes, just touch the appropriate key; G for Graphics, N for Normal (alphanumeric) characters, or R for Reverse characters. (Also, because key 9 says 'GRAPHICS' on it, I have allowed this to be used as well as G to enter Graphics mode.) This causes a different flashing cursor to appear, and the required characters can then be drawn directly from the keyboard, without using the Shift key. In all 3 DRAW modes, the cursor will move 1 position to the right after each character is drawn, and will 'wrap around' the right-hand side of the screen to appear on the left of the next line. (Actually the 'wrap around' is 'continuous,' which means that if the cursor is taken off the bottom right of the screen, it will appear at the top left. Note that wrap around only occurs in DRAW modes; in CONTROL mode the cursor is restricted to the screen boundaries.

To change to a different character set, or to move the cursor to another position in CONTROL mode, press Newline, and the X cursor will reappear.

A complete description of all 5 modes follows below:

CONTROL MODE

Flashing X cursor: Keys 5, 6, 7, 8 (direction), S, G, R, N and 'Graphics' (9) are enabled. The cursor will travel through existing characters without deleting them.

GRAPHICS

```
10 LET G$="↑12345678←A←←DEFGH←←←←←←←←QRST←←W←Y#↑"
20 LET P=0
30 LET Q=0
40 LET X=P
50 LET Y=Q
60 PRINT AT Y,X;
70 LET O=PEEK (PEEK 16398+256*PEEK 16399)
80 PRINT "X"
90 LET P=X
95 LET Q=Y
100 LET A$=INKEY$
110 IF A$="5" THEN LET P=X-(X>0)
120 IF A$="8" THEN LET P=X+(X<31)
130 IF A$="6" THEN LET Q=Y+(Y<21)
140 IF A$="7" THEN LET Q=Y-(Y>0)
150 IF A$="9" OR A$="G" THEN GOTO 500
160 IF A$="R" THEN GOTO 600
170 IF A$="N" THEN GOTO 700
180 IF A$="S" THEN GOTO 2000
190 PRINT AT Y,X;CHR$ O
200 GOTO 40
500 REM GRAPHICS
505 LET O=0
510 IF INKEY$="9" OR INKEY$="G" THEN GOTO 510
520 PRINT AT Y,X;"G"
530 LET A$=INKEY$
540 IF A$=CHR$ 118 THEN GOTO 90
550 IF A$=CHR$ 119 OR A$="0" THEN GOSUB 800
560 IF A$<"1" OR A$>"Z" THEN GOTO 590
570 LET A$=G$(CODE A$-28)
580 GOSUB 1000
590 PRINT AT Y,X;CHR$ O
595 GOTO 520
600 REM REVERSE
605 LET O=0
610 IF INKEY$="R" THEN GOTO 610
620 PRINT AT Y,X;"R"
630 LET A$=INKEY$
640 IF A$=CHR$ 118 THEN GOTO 90
650 IF A$=CHR$ 119 THEN GOSUB 800
660 IF A$<CHR$ 11 OR A$>"Z" THEN GOTO 690
670 LET A$=CHR$ (CODE A$+128)
680 GOSUB 1000
690 PRINT AT Y,X;CHR$ O
695 GOTO 620
700 REM NORMAL
705 LET O=0
710 IF INKEY$="N" THEN GOTO 710
720 PRINT AT Y,X;"N"
730 LET A$=INKEY$
```

GRAPHICS MODE

Flashing G cursor: All alpha and numeric keys will produce the graphic character shown on the key, or SPACE, where there is no graphic shown on the key, or INVERSE SPACE: key 2. The RUBOUT (delete) key (Ø) will work with OR WITHOUT using Shift.

REVERSE MODE

Flashing R cursor: All keys will give the Reverse (or Inverse) of their normal character (except Space, of course, which will Break the program—use Graphics mode to obtain Inverse space).

The Shift key must be used to RUBOUT, as the inverse zero character on this key may also be required.

NORMAL MODE

Flashing N cursor: All keys give their normal character (except space, as above—use either Graphics or Control mode if you require spaces).

Again, Shift must be used to obtain rubout.

SAVE MODE

There is very little point in creating a beautiful screen drawing if you cannot save it on tape for recall later. Save mode, obtainable from control mode by touching key S, stores the entire screen display in a string array, and then allows the program, with the stored display to be SAVEd in the normal way. It is NOT necessary at this stage to decide that the drawing is 'complete,' LOADing a previous drawing from tape will allow the option of simply displaying the picture on screen or of continuing to work on it.

Note that formation of the array takes about 9Ø seconds. The indication that Save mode has in fact been entered is that the cursor disappears altogether when key S is pressed.

NOTES

This is a very easy program to type in, as the 3 DRAW routines (at lines 5ØØ, 6ØØ and 7ØØ) are very similar, and all 3 may be entered at once using the Edit function to renumber the lines immediately after you have entered them.

GRAPHICS *(Continued)*

```
740 IF A$=CHR$ 118 THEN GOTO 90
750 IF A$=CHR$ 119 THEN GOSUB 800
760 IF A$="" THEN GOTO 790
780 GOSUB 1000
790 PRINT AT Y,X;CHR$ O
795 GOTO 720
800 REM RUBOUT
810 PRINT AT Y,X;"←"
820 LET X=X-1
830 IF X>=0 THEN GOTO 870
840 LET X=31
850 LET Y=Y-1
860 IF Y<0 THEN LET Y=21
870 LET A$=""
880 RETURN
1000 REM PRINT
1010 PRINT AT Y,X;A$;
1020 LET X=X+1
1030 IF X<32 THEN GOTO 1070
1040 LET X=0
1050 LET Y=Y+1
1060 IF Y=22 THEN LET Y=0
1070 LET O=PEEK (PEEK 16398+256*PEEK 16399)
1080 RETURN
2000 REM SAVE
2005 PRINT AT Y,X;"←"
2010 DIM S$(22,32)
2020 FOR Y=1 TO 22
2030 FOR X=1 TO 32
2040 PRINT AT Y-1,X-1;
2050 LET S$(Y,X)=CHR$ PEEK (PEEK 16398+256*PEEK 16399)
2060 NEXT X
2070 NEXT Y
2080 CLS
2110 PRINT "SCREEN NOW STORED IN ARRAY S$()."
2120 PRINT AT 2,0;"ENTER NAME OF GRAPHIC:-"
2130 INPUT N$
2140 IF N$="" THEN GOTO 2130
2150 PRINT AT 4,0;"START TAPE, THEN PRESS S TO SAVE"
2400 IF INKEY$<>"S" THEN GOTO 2400
2500 SAVE N$
2600 CLS
2900 PRINT "PRESS D TO DISPLAY SCREEN"
3000 PRINT AT 5,0;"OR C TO DISPLAY AND CONTINUE"
3050 LET I$=INKEY$
3100 IF I$<>"D" AND I$<>"C" THEN GOTO 3050
3110 FOR Y=1 TO 22
3120 PRINT AT Y-1,0;S$(Y)
3130 NEXT Y
3140 IF I$="D" THEN GOTO 3140
3150 GOTO 40
9000 SAVE "GRAPHICS"
9005 RUN
```

Line 10 contains a string of all the available graphics characters, in the order of their corresponding keys, with spaces in the string for each key which does not have a graphic on it. Note the inverse space corresponding to the Z key, which is necessary because the normal Space/Break key cannot be used during the program run. You may of course change line 10 so that other characters (e.g., Normal or Inverse +, key K) are available in Graphics Mode.

Variables P, Q, X and Y are the previous and current Print coordinates, which are altered by the INKEY$ routine (lines 100–140). Line 70 PEEKS the character which occupies the current PRINT AT position, and saves it in variable O, so that it may be replaced once the cursor has passed over it. Lines 505, 605 and 705 reset O to zero so that, in DRAW modes, existing characters can be overwritten.

Lines 510, 610 and 710 are very important but often forgotten whenever INKEY$ has been used to cause a program 'branch.' Obviously it is necessary to detect that the finger has been removed from the key before proceeding with the Print routine, or you will get a string of unwanted Rs appearing every time you enter Reverse mode.

CHR$ 118 (lines 540, 640 and 740) is the Newline character which causes each of the DRAW routines to return to the normal CONTROL routine. CHR$ 119 is the RUBOUT character. Note that line 550 differs from 650 and 750 in that it regards key 0 as the same as RUBOUT (in Graphics mode only).

The Rubout routine beginning at line 800 works in the reverse manner to the Print routine beginning at line 1000. Compare lines 820–860 and lines 1020–1060, both of which allow cursor 'wrap around' with lines 110–140, which keep the cursor within the screen boundaries.

The Save routine beginning at line 2000 again uses the technique of PEEKing the 'contents of the contents' of system variable DF-CC (PRINT AT) to transfer each screen character into a 2-dimensional array S$.

The program, along with this array, is SAVEd at line 2500, so that on subsequent LOADing, the program will continue from this point. Line 3140 is an endless loop which enables the screen display to persist forever (or until Break is pressed), without being spoiled by the 'end-of-program' report code.

Finally, lines 9000 and 9005 are used in nearly all the programs in this book. They are normally not an essential part of the program (though there are exceptions—see 'Kami Kazi Drive'), but are included so that all the programs may be SAVEd by typing 'GOTO 9000,' which will then cause them to run automatically straight from loading.

Chapter 9
Game Programs

Pontoon

This is NOT the same game of Pontoon as the one you will find played at your local neighbourhood casino!

In Part I, there are several casino simulations, all of which serve to prove that the casino ALWAYS has the advantage over the player. Historically, however, this has not always been the case. In comparatively recent times, pontoon was played accoring to rules which, amazingly, were not only 'fair,' but which actually save the player using optimum strategy a marginal advantage over the casino. Unfortunately, these rules are no longer in force, but this program is an attempt to reconstruct those glorious bygone days.

Of course, you must still play VERY well to actually make a profit even when playing this 'old-rules' game. In certain respects, the 'house rules' employed in the program are even harsher than those in force in present-day casinos. There is no such thing as a 'stand off'; if player and dealer have equal hands, then the dealer wins! Also, the dealer (i.e., the computer) derives an advantage from the fact that there is only 1 player (yourself) to compete against. For example, if you have a 5-card trick, and the dealer has a score of 20 with 4 cards, in any 'real-life' situation the dealer would obviously stand, conceding defeat to you, but almost certainly beating most of the other players. With no other players involved, however, the dealer might just as well take a fifth card, in the unlikely hope of drawing an ace, and indeed that is what the computer would do according to the strategy of this program.

NOTES

A program of this length can be very easily understood if it is split into several distinct sections. I haven't used REM statements to distinguish the sections, but you may add your own if you find them helpful. The main sections are:

5–50 Initialization
5–110 The deal
115–445 Player's cards and options

PONTOON

```
1 GOTO 9010
5 DIM T$(2,5,3)
6 DIM T(2)
7 DIM A$(4,13)
10 RAND
15 LET S$="HISC"
20 LET N=1000
25 LET R$="STICK←←OR←←BUY←←OR←←TWIST"
30 LET T(1)=0
35 LET T(2)=0
40 FOR J=1 TO 4
45 LET A$(J)="A23456789TJQK"
50 NEXT J
55 PRINT "YOUR CARDS←←←←←YOU HAVE £";N
60 PRINT AT 11,0;"DEALERS CARDS"
65 LET P=2
70 LET Q=1
75 GOSUB 500
80 LET A=1
85 LET B=1
90 GOSUB 1000
95 LET P=13
100 GOSUB 600
105 LET A=2
110 GOSUB 1000
115 PRINT AT 21,8;"YOUR STAKE £(10)"
120 INPUT W$
125 IF W$="" THEN LET W$="10"
130 FOR J=1 TO LEN W$
135 IF W$(J)<"0" OR W$(J)>"9" THEN GOTO 120
140 NEXT J
145 LET S=VAL W$
150 IF S<10 OR S>1000 THEN GOTO 120
155 LET S1=S
160 GOSUB 560
165 PRINT AT 0,25;N
170 LET P=2
175 LET Q=7
180 GOSUB 500
185 LET A=1
190 LET B=2
195 GOSUB 1000
200 LET P=13
205 LET A=2
210 GOSUB 600
215 GOSUB 1000
220 IF T(1)=21 THEN LET R$="PONTOON"
225 POKE 16418,0
230 PRINT AT 23,0;"←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
235 PRINT AT 23,(30-LEN R$)/2;R$
240 IF B=5 OR LEN R$<10 THEN GOTO 700
245 LET I$=INKEY$
```

700–745 Reveal dealer's first 2 cards
750–830 Dealer's additional cards
2000–2110 End options
3000–6100 Scoring
9000– Rules
Subroutines:
500–550 Print card outline
560–585 Update 'cash' display
600–630 Print card back
1000–1110 Select card
1200–1270 'Ace-reduction'

INITIALIZATION

A full pack of cards is held in array A$, with the suit being defined by the corresponding element of S$. Scores are held in array T (1 = player; 2 = computer), with the actual cards (maximum 10 required) held in T$. N is the player's current 'capital.' R$ is a 'report string.'

The use of the report string in this program is two-fold. Not only is it a device for displaying user information on the screen, it also provides game status information for the computer's own use. At several points the program branches according to the contents of R$ (see lines 240, 700, 740, etc.).

THE DEAL

The mechanics of the game require that a card is dealt to both players, the initial stake is placed, and then a second 2 cards are dealt. This complicates matters slightly; it would be simpler from a programming point of view if all the player's options were completed first, and then the dealer's cards dealt out. However, this alternation between player and dealer is achieved by use of a 'flag' A (1 = player; 2 = dealer), a 'pointer' B (which represents the number of cards held by each player), and 2 'print pointers,' P and Q (P = line; Q = column at which the card is to be printed).

The actual printing of the cards is then delegated to the 2 subroutines 500 (card outline) and 600 (card back—the dealer's cards are not revealed at this stage). Subroutine 1000 selects a card from the pack; line 1050 assigns it to the appropriate position in T$, and line 1060 prints it within the correct card outline.

PONTOON *(Continued)*

```
250 IF I$<>"B" AND I$<>"S" AND I$<>"T" THEN GOTO 245
255 PRINT AT 23,0;"←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
260 IF I$="S" THEN GOTO 700
265 IF I$="B" AND LEN R$=25 THEN GOTO 400
270 IF I$<>"T" THEN GOTO 235
275 LET R$="STICK←←OR←←TWIST"
280 GOTO 415
400 LET S=S+S1
405 GOSUB 560
415 LET A=1
420 LET B=B+1
425 LET Q=Q+6
430 LET P=2
435 GOSUB 500
440 GOSUB 1000
445 GOTO 225
500 PRINT AT P,Q;"↑E777R↑"
510 FOR K=1 TO 5
520 PRINT TAB Q;"↑5←←←8↑"
530 NEXT K
540 PRINT TAB Q;"↑W666Q↑"
550 RETURN
560 LET N=N-S1
565 PRINT AT 21,20;"←←←←←"
570 PRINT AT 21,20;S
575 PRINT AT 0,25;"←←←←←←"
580 PRINT AT 0,25;N
585 RETURN
600 FOR K=0 TO 6
610 PRINT AT P+K,Q;"↑HHHHH↑"
620 NEXT K
630 RETURN
700 IF R$(1)="B" THEN GOTO 2000
705 LET A=2
710 LET P=13
715 FOR Q=1 TO 7 STEP 6
720 GOSUB 500
725 PRINT AT P+1,Q+1;T$(2,INT (Q/7)+1)
730 NEXT Q
735 IF T(2)=21 THEN GOTO 3000
740 IF R$(1)="P" THEN GOTO 4000
745 LET B=2
750 IF B=4 AND T(2)>16 THEN GOSUB 1200
755 IF T(2)<17 OR R$(1)="P" THEN GOTO 800
760 LET D$=STR$ (T(2)+1)
765 IF T(2)=21 THEN LET D$="PONTOONS / 5 CARDS"
770 LET D$="DEALER PAYS←"+D$
775 GOTO 6000
800 LET B=B+1
805 GOSUB 500
810 GOSUB 1000
815 LET Q=Q+6
820 IF T(2)>21 THEN GOTO 4500
825 IF B=5 THEN GOTO 3500
```

PLAYER'S CARDS AND OPTIONS

This section is quite straightforward, lines 115–220 accepting the stake and calling the same 3 subroutines as above to print the second cards, and lines 225–445 forming a loop for as many additional cards as are required. The string in line 230 (and subsequently) is 32 spaces, used to clear a screen line of unwanted text. There is a much neater way of doing this, which will be used in later programs, but I will try and keep things simple at this stage.

Line 125 is a useful way of setting a 'default' parameter, so that the player merely has to press Newline when a particular entry is expected. Here the default stake is set to £10; carefree players may prefer to increase the default value to something higher. The POKE in line 225 is the same as was used in 'Tarot,' to increase the size of the screen. The INKEY$ routine to enter the player's selection (lines 245–270 is conventional, but with a slightly unusual variation. Once option T (Twist) has been selected, R$ is re-defined (line 275), so that the player is no longer offered the option to Buy. Therefore line 265 not only detects the key depressed (INKEY$), but also the current value of R$. (LEN R$ = 25 only when R$ includes the word BUY, as defined in line 25.)

DEALER'S CARDS

Fortunately the dealer's options in Pontoon are clearly defined by the rules of the game, so there is very little need to give the computer any 'intelligent' strategy (other than, as already mentioned, the ability to play on regardless of the consequences whenever the player has a 5-card trick—see line 755). Line 750 is the only other sign of 'computer intelligence': if the dealer holds, say, ace, 2, 3 and 4, making a total of 10 or 20, the computer will not stick on 20 but will reduce its ace-count and take the inevitable 5-card trick (see the 'subroutines' section below).

The 'dealer' section commences with the overprinting of the 2 card backs with the (already selected) cards themselves (lines 705–730) and then continues adding cards in much the same way as before. Note line 700, which detects if the player has 'bust,' in which case the entire dealer section is skipped. D$ is a second 'report string,' which serves a similar dual function to R$, but which this time contains information concerning the dealer's hand.

END OPTIONS

Another completely conventional INKEY$ routine which branches either to line 25 to continue the game, or to a comparatively trivial (user-friendly is the more polite term) game summary and ending.

PONTOON *(Continued)*

```
830 GOTO 750
1000 LET X=INT (RND*52)+1
1010 FOR J=1 TO 4
1015 IF X<=13 THEN GOTO 1040
1020 LET X=X-13
1030 NEXT J
1040 IF A$(J,X)="X" THEN GOTO 1000
1050 LET T$(A,B)=A$(J,X)+"←"+S$(J)
1055 LET A$(J,X)="X"
1060 IF A=1 OR B>2 THEN PRINT AT P+1,Q+1;T$(A,B)
1070 IF X>10 THEN LET X=10
1080 LET T(A)=T(A)+X
1085 IF X=1 THEN LET T(A)=T(A)+10
1090 IF T(A)>21 THEN GOTO 1200
1100 IF A=1 AND B=5 THEN LET R$="FIVE CARD TRICK"
1110 RETURN
1200 FOR J=1 TO B
1210 IF T$(A,J,1)="A" THEN GOTO 1250
1220 NEXT J
1230 IF A=1 THEN LET R$="BUST"
1240 GOTO 1110
1250 LET T(A)=T(A)-10
1260 LET T$(A,J,1)="A"
1270 GOTO 1100
2000 POKE 16418,2
2010 PRINT AT 15,13;"PRESS D TO DEAL"
2020 PRINT AT 17,13;"OR Q TO FINISH"
2030 LET I$=INKEY$
2040 IF I$<>"Q" AND I$<>"D" THEN GOTO 2030
2050 CLS
2060 IF I$="D" THEN GOTO 25
2070 LET I$="LOST"
2080 IF N>1000 THEN LET I$="WON"
2090 PRINT "YOU ";I$;" £";ABS (N-1000)
2100 PRINT AT 10,0;"THANKS FOR PLAYING."
2110 GOTO 9999
3000 LET D$="DEALER HAS PONTOON"
3010 IF R$(1)="F" THEN GOTO 6000
3020 GOTO 3520
3500 LET D$="DEALER HAS 5 CARD TRICK"
3510 IF R$(1)="F" THEN GOTO 6000
3520 LET T(2)=50
3530 LET N=N-S
3540 LET S=S*2
3550 GOTO 6000
4000 LET D$="TOO GOOD"
4010 LET T(1)=90
4020 GOTO 3530
4500 LET D$="DEALER HAS BUST"
4510 LET T(2)=0
4520 IF R$(1)="F" THEN GOTO 3530
6000 LET I$="DEALER WINS"
6010 IF T(1)<=T(2) THEN GOTO 6040
6020 LET I$="YOU WIN"
```

SCORING

I won't bother to trace the various spaghetti-like paths through this section, as all the possible outcomes of a game must be accounted for, however unlikey. (Line 3510, for example, prevents the stake from being doubled when both player and dealer have a 5-card trick.) Suffice it to say that all paths lead to line 6000, where the winner is declared. Note that lines 6110 and 2010–2020 print the same message at alternative positions on the screen, depending on the number of cards displayed—a totally unnecessary cosmetic touch which nevertheless adds a touch of variety to the otherwise mundane process of message display.

RULES

All the rules and running instructions are contained in the listing itself. The rules are placed at the END of the program beginning at line 9080 for 2 very practical reasons. From the computer's point of view, it is desirable for the most frequently called lines to occur near the start of a program, and although, in the interests of 'structured programming,' it is seldom possible to adhere to this guideline very closely, it is clearly sensible to place seldom-used instruction lines at the end. The real reason, however, is that programmers invariably add instructions to their programs only as an afterthought (if at all!) when the program is virtually completed, by which time there is nowhere else to put them other than at the end!

Note that the spacing in these PRINT lines is designed to create a pleasing appearance on the screen, which unfortunately makes them look rather odd when LISTed. Where 2 words are printed together (e.g., MINIMUMBET in line 9110), you should enter them like this as they will be split onto two screen lines when the program is run. Similarly, double spaces should be entered exactly as printed.

The INKEY$ routine at lines 9150–9190 is rather interesting. The rules are to be displayed on screen until the user starts the game by pressing D. To make the display slightly more interesting, the figures £ 8000 are flashed (normal and inverse), NOT continuously, but in a regular pattern, with the inverse figures persisting for a short time between bouts of flashing. Notice that the INKEY$ line itself (9160) is called almost continuously, and that the 'flash subroutine' (9300) is very short. Whenever you use INKEY$ to detect a user response, beware of long loops which would cause a delay in a keypress being detected.

PONTOON *(Continued)*

```
6030 LET N=N+S+S
6040 PRINT AT 23,0;"←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
6050 PRINT AT 23,(30-LEN D$)/2;D$
6060 FOR J=1 TO 40
6070 NEXT J
6075 GOSUB 575
6080 PRINT AT 21,0;"←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
6085 PRINT AT 23,0;"←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
6090 PRINT AT 21,0;I$;" £";S
6100 IF B=2 THEN GOTO 2000
6110 PRINT AT 23,0;"PRESS D TO DEAL OR Q TO FINISH"
6120 POKE 16418,2
6130 GOTO 2030
9000 SAVE "PONTOON"
9010 PRINT AT 10,12;"↑E7777777R↑"
9020 PRINT TAB 12;"↑5↑PONTOON↑8↑"
9030 PRINT TAB 12;"↑W66666660↑"
9040 PRINT AT 18,25;"BY V+H"
9050 FOR J=1 TO 50
9060 NEXT J
9070 CLS
9080 PRINT TAB 10;"HOUSE RULES"
9090 PRINT AT 3,0;"MAX. STAKE: £1000"
9100 PRINT "MIN. STAKE: £10"
9110 PRINT "(JUST PRESS NEWLINE←←FOR MINIMUMBET)";AT
      8,0;"DEALER WINS EQUAL HANDS"
9120 PRINT AT 10,0;"YOU MAY TWIST OR BUY EXTRA CARDS";
      AT 12,0;"PONTOONS←←AND←←5 CARD TRICKS PAYDOUBLE
      ←(TO PLAYER OR DEALER)"
9130 PRINT AT 15,0;"MAXIMUM WIN PER HAND THEREFORE:";
      AT 17,13;"£ 8000"
9140 PRINT AT 19,0;"YOU HAVE £1000 TO START, BUT
      ←THEBANK←←WILL ALLOW←←YOU
      ←←UNLIMITEDCREDIT.←←PRESS D TO DEAL"
9150 FOR J=1 TO 40
9160 IF INKEY$="D" THEN GOTO 9500
9170 IF J>25 THEN GOSUB 9300
9180 NEXT J
9190 GOTO 9150
9300 PRINT AT 17,13;"£ 8000"
9310 LET S=RND*RND
9320 PRINT AT 17,13;"£ 8000"
9330 RETURN
9500 CLS
9510 RUN 5
```

SUBROUTINES

The two minor subroutines 500 and 600 have been mentioned already. Both merely print the card shape (front and back) at a position specified by the 'print pointers' P and Q.

Subroutine 560 updates the player's capital (N) and stake (S) by overprinting the previous figures with blanks and then printing the new values. Lines 565/570 and 575/580 could have been written as single lines, but do not compress the whole subroutine into one line, as only the second half of it is called in line 6075.

The 'ace reduction' routine beginning at line 1200 is called whenever either player's score exceeds 21. Any ace in the hand which has previously been scored as 11 is now scored as 1. This reduction operation must be performed one ace at a time, so once an ace has been reduced, this fact is 'flagged' by inverting the letter A (line 1260).

Incidentally, it is a question of semantics whether lines 1200 on should really be regarded as a subroutine at all. They are really a branch of subroutine 1000 (called by a GOTO in line 1090), but are used as a subroutine in their own right by line 750 (. . . GOSUB 1200), where the computer is not FORCED to reduce its ace-count, but CHOOSES to do so in order to take a 5-card trick.

Subroutine 1000 selects a random number in the range of 1–52, and then uses a FOR-TO loop to reduce this to a number in the range of 1–13. The loop control variable J is then used to determine how many times the number has been reduced, and therefore the suit of the selected card, S$(J).

Notice that, as in 'Tarot,' no attempt is made to avoid duplicate numbers being selected; duplicates are merely detected (line 1040) and re-selected. Because no more than 10 of the 52 cards will ever be required, this is even less likely to cause any delay than it did in 'Tarot.' But there ARE methods of selecting random numbers without any danger of duplication, as we shall see in the next program Battleships.

Battleships

If Pontoon is essentially an adult game (at least, when played for money), Battleships has traditionally been regarded as 'strictly for kids.' However, computer technology is changing all that, and I am sure that as many adults will enjoy this version of Battleships as youngsters will enjoy 'Pontoon.'

The program is far more complex than Pontoon; there is a tiny machine code routine to create a spectacular effect whenever a ship is hit, and the entire display is generated by POKEing to the screen, rather than relying on PRINT AT statements as before. Also, two levels of play are included; the first, where the computer plays randomly, is

BATTLESHIPS

```
1 REM XXXXXXXXXXXXXXXXXXXXX
2 RAND
3 GOSUB 3000
5 GOTO 95
10 LET A=P+CODE A$(1)-35+33*(CODE A$(2)-27+N)
15 LET PA=PEEK A
20 RETURN
25 INPUT A$
30 IF A$="QUIT" AND G<>0 THEN RETURN
35 IF LEN A$<>2 THEN GOTO 25
40 IF A$(1)<"A" OR A$(1)>"Z" OR A$(2)<"1" OR A$(2)>"9"
     THEN GOTO 25
45 GOSUB 10
50 RETURN
55 FOR J=13 TO 21
60 PRINT AT J,2;"←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
65 NEXT J
70 RETURN
75 FOR J=1 TO 12
80 LET U=USR 16514
85 NEXT J
90 RETURN
95 CLS
100 PRINT TAB 10;"BATTLESHIPS"
105 IF PEEK 16514=1 THEN GOTO 150
110 LET M$="0142002A0C40090609237EFE762805C6807718F510F3C9"
115 FOR J=1 TO 23
120 POKE 16513+J,(CODE M$(J*2-1)-28)*16+CODE M$(J*2)-28
125 NEXT J
150 LET P=PEEK 16396+256*PEEK 16397
155 FOR J=3 TO 28
160 POKE P+J+33,J+163
165 NEXT J
170 FOR J=68 TO 332 STEP 33
175 POKE P+J,INT (J/33)+155
180 POKE P+J+363,INT (J/33)+155
185 NEXT J
190 LET REPLAY=0
195 DIM C(234)
200 DIM M(234)
205 DIM S(2)
210 LET N=0
215 LET G=0
220 IF REPLAY THEN GOTO 390
250 PRINT AT 11,3;"ENTER YOUR SHIP POSITIONS"
255 PRINT TAB 15;"LEN FROM TO "
260 LET E$="A/CARRIER←←←(5)"
265 FOR J=1 TO 9
270 IF J>2 AND J<6 THEN LET E$="CRUISER←←←←←(4)"
275 IF J=6 OR J=7 THEN LET E$="DESTROYER←←←(3)"
280 IF J>7 THEN LET E$="FRIGATE←←←←←(2)"
285 PRINT AT J+12,3;E$
```

not terribly exciting, but at level 2 the program uses an 'intelligent strategy' and plays a reasonably competitive game.

SCREEN POKES

Poking to the screen is itself rather like playing Battleships (although, hopefully, less random!). The screen can be regarded as a 32 x 22 grid, and any point can be accessed directly. Unfortunately, the design of the Timex 1000 creates 2 slight problems not found on other computers; the grid is not really 32 columns wide, but 33, the 33rd column always being filled with character 118. Poking anything else into column 33 invariably causes disaster. Also, many of the Timex 1000 character codes cannot be poked to the screen without causing the display to crash. Neither of these problems should cause any great headaches, however. Assuming that you know which character you want on the screen, and where you want it to appear, there can never be any danger of poking into column 33, or of poking illegal characters anywhere.

The 'address' of the screen grid is not fixed, as on other computers, but can be found very simply by PEEKing the system variable D-FILE. PEEK 16396 + 256*PEEK 16397 gives the address of the 118 character immediately PRECEDING the screen grid, so the first screen position is 1 plus this value. (I usually assign variable P to D-FILE value—see line 15Ø.) The 'forbidden' column 33 starts at P + 33, and row 2 starts at P + 34, etc.

MACHINE CODE

The machine code routine in Battleships causes any 9 screen lines to be inverted. Bytes 2 and 3 of the routine define exactly which 9 lines are to be inverted, and these 2 bytes are actually changed during the course of the program. The machine code loader is contained within the Basic program—lines 11Ø–125, with line 11Ø containing the entire routine in hex form in a Basic string. Once the program has been RUN for the first time, line 1Ø5 detects the presence of the routine in line 1, and so skips the loader.

BATTLESHIPS *(Continued)*

```
290 GOSUB 25
295 IF PA<>0 THEN GOTO 290
300 POKE A,8
305 LET A1=A
310 PRINT AT J+12,20;A$
315 GOSUB 25
320 IF PA<>0 THEN GOTO 315
325 LET L=ABS (A-A1)
330 LET D=0
335 IF L=VAL E$(14)-1 THEN LET D=1
340 IF L=33*(VAL E$(14)-1) THEN LET D=33
345 IF D=0 THEN GOTO 315
350 FOR K=1 TO VAL E$(14)-1
355 POKE A1+K*D*SGN (A-A1),8
360 NEXT K
365 PRINT AT 12+J,24;A$
370 NEXT J
390 PRINT AT 11,2;"COMPUTER NOW PLACING SHIPS"
400 FOR J=3 TO 28
405 POKE P+J+396,J+163
410 NEXT J
415 GOSUB 55
420 FOR J=1 TO 9
425 LET R=INT (RND*234)+1
430 IF C(R)=1 THEN GOTO 425
435 LET L=1+(J<8)+(J<6)+(J<3)
440 LET D=(RND>.5)*25+1
445 IF RND>.5 THEN LET D=-1*D
450 IF ABS D<>1 THEN GOTO 465
455 LET K=L*D+R-1
460 IF INT (K/26)<>INT((R-1)/26) THEN GOTO 425
465 IF R+(L*D)>234 OR R+(L*D)<1 THEN GOTO 425
470 FOR K=R TO R+(L*D) STEP D
475 IF C(K)=1 THEN GOTO 425
480 NEXT K
485 FOR K=R TO R+(L*D) STEP D
490 LET C(K)=1
495 NEXT K
500 NEXT J
505 FOR J=1 TO 234
510 LET M(J)=J
515 NEXT J
600 PRINT AT 11,0;"ENTER YOUR TARGET COORDINATES"
605 PRINT AT 8,28;"HITS"
610 PRINT AT 19,28;"HITS"
612 PRINT AT 0,27;"SHOTS"
615 LET N=11
620 GOSUB 25
625 IF A$="QUIT" THEN GOTO 1000
630 PRINT AT 17,29;A$
635 LET R=(CODE A$(1)-37)+(26*(CODE A$(2)-29))
640 LET K=23
645 IF C(R)<>1 THEN GOTO 680+C(R)
650 LET K=151
```

The routine is as follows:

```
01 42 00          LD BC, 66          GRID POSITION (START)
2A 0C 40          LD HL, (16396)     D-FILE
09                ADD, HL, BC        START POSITION
06 09             LD B, 9            COUNTER
23          LOOP  :INC HL            SCREEN POSITION
7E                LD A, (HL)
FE 76             CP 118             IS IT END OF LINE?
28 05             JRZ, 5             IF YES, NEXTLN
C6 80             ADD A, 128         INVERT
77                LD(HL), A          PRINT
18 F5             JR 245             LOOP (next character)
10 F3       NEXTLN:DJNZ 243          LOOP (next line)
C9                RETURN
```

Thinking of the screen as a 33 x 22 grid numbered from 1 (top left) to 726 (bottom right), register BC is loaded with the grid number of the position immediately preceding the line from which the inversion is to begin (i.e., a multiple of 33). Register HL is loaded with the contents of D-FILE, and then the contents of BC are added to this. The effect, then, of the first 6 bytes of code are to load HL with the actual address (less 1) from which we want the inversion routine to start. Register B is then re-used, this time as a counter of the number of screen lines inverted (9). The main loop then inverts each character on the next 9 lines of the screen.

The machine code routine is called in a Basic FOR-TO loop (lines 75–85) to produce the flashing effect at the desired rate.

To use this routine in other programs, the following bytes can be altered:

Bytes 2 and 3, to commence inversion at a different screen line. (As used in Battleships—see Notes.)

Byte 9, to vary the number of lines to be inverted.

RUNNING INSTRUCTIONS

I hope that none of you have led such a sheltered life that I actually have to explain how to play Battleships! The program allows you to set up your own fleet positions on the top half of the screen, and then the computer sets up its ships on the lower half. (Your ships are visible, but the computer does NOT cheat! The computer's ships are, of course, invisible.) Enter the grid coordinate of one end of a ship (then Newline), and that position will be marked on screen. Then enter the grid position of the other end, and, provided you have made a valid entry, the whole ship will be drawn in. There are 9 ships in all to place.

BATTLESHIPS *(Continued)*

```
655 LET C(R)=2
660 LET S(1)=S(1)+1
665 POKE 16515,173
670 POKE 16516,1
675 GOSUB 75
680 POKE A,K
682 PRINT AT 20,29;S(1)
685 LET G=G+1
687 PRINT AT 1,29;G
690 IF S(1)=32 THEN GOTO 2000
700 PRINT AT 11,0;"<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
705 LET N=0
710 IF H<>-1 THEN GOTO 850
715 LET R=INT (RND*(235-G))+1
720 LET X=M(R)
725 LET M(R)=M(235-G)
730 LET M(235-G)=X
735 GOSUB 980
740 PRINT AT 6,29;A$
750 LET K=23
755 IF PA<>8 THEN GOTO 785
757 IF H>-1 THEN LET H=X
760 LET S(2)=S(2)+1
765 LET K=151
770 POKE 16515,66
775 POKE 16516,0
780 GOSUB 75
785 POKE A,K
790 PRINT AT 9,29;S(2)
795 IF S(2)=32 THEN GOTO 2000
800 GOTO 615
850 IF NOT H THEN GOTO 715
855 LET H2=H2+1
860 IF H2=1 THEN LET H1=X
865 FOR J=1 TO 4
870 LET X=H+H(J)
875 IF X<1 OR X>234 THEN GOTO 950
880 GOSUB 980
885 IF PA<>0 AND PA<>8 THEN GOTO 950
886 LET Q=1
887 IF M(X)=X THEN LET Q=X
890 FOR K=Q TO 235-G
895 IF M(K)<>X THEN GOTO 915
900 LET M(K)=M(235-G)
905 LET M(235-G)=X
910 LET K=235-G
915 NEXT K
920 LET K=H(J)
925 LET H(J)=H(1)
930 LET H(1)=K
935 PRINT AT 6,29;A$
940 GOTO 750
950 NEXT J
955 IF H1=H THEN GOTO 970
```

There is only one point I would like to stress; this is really so obvious that you may think that it is hardly worth mentioning, but I have watched many people play with this program, and at least half of them make a mistake here: when you enter a ship position to start at, say A1, and the ship's length is 5, then it will finish at either A5 or E1, and NOT at A6 or F1!

You then simply take pot-shots at each other's fleets, until one of you has totally destroyed the other (32 hits). If you lose, or enter the word QUIT instead of a grid coordinate, the computer will reveal the positions of its remaining ships, by drawing them onto its grid. Because the process of setting up your fleet takes some time and is boring, there is an option at the end of each game to replay using the same positions as before. The computer is so stupid that it will not remember these positions; it will, however, reposition its own ships each game. (It isn't THAT stupid!)

NOTES

Having already mentioned the machine code routine and the screen poke principle, the program can be broken down into easy sections, as before:

10–90 A collection of short subroutines
105–125 Machine code loader
150–185 Print player's grid (sea area)
190–370 Position player's ships
400–415 Print second grid
420–515 Position computer's ships
600–690 Player's shots
700–795 Computer's shots
850–975 Computer's strategy (level 2)
1000–1050 Reveal computer's ships
2000–2028 Declare winner
2030–2050 End options
2090–2155 Retain player's positions for new game
3000–3080 rules
3090–3150 Initialize computer strategy (level 2)
4000–4070 Utterly pointless 'end' routine

BATTLESHIPS *(Continued)*

```
960 LET H=H1
965 GOTO 865
970 LET H=0
973 LET H2=0
975 GOTO 715
980 LET A$=CHR$ (X-(INT ((X-1)/26)*26)+37)+CHR$ (INT
      ((X-1)/26)+29)
990 GOSUB 10
995 RETURN
1000 LET N=11
1005 FOR X=1 TO 234
1010 IF C(X)<>1 THEN GOTO 1030
1015 LET A$=CHR$ (X-(INT ((X-1)/26)*26)+37)+CHR$ (INT
      ((X-1)/26)+29)
1020 GOSUB 10
1025 POKE A,8
1030 NEXT X
1035 PRINT AT 11,3;"PRESS ANY KEY TO CONTINUE"
1040 IF INKEY$="" THEN GOTO 1040
1045 PRINT AT 11,0;"←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
1050 GOTO 2030
2000 LET W$="YOU"
2010 IF S(2)>S(1) THEN LET W$="I"
2015 PRINT AT 11,3;W$;"←WON BY←";ABS (S(1)-S(2))
2020 IF W$="I" THEN GOTO 1000
2025 FOR J=1 TO 25
2026 NEXT J
2028 GOTO 1035
2030 GOSUB 55
2035 PRINT AT 15,2;"PRESS N FOR NEW GAME";AT 17,8;"S FOR SAME
      ←POSITIONS";AT 19,8;"X TO END"
2040 IF INKEY$="N" THEN GOTO 2200
2045 IF INKEY$="X" THEN GOTO 4000
2050 IF INKEY$<>"S" THEN GOTO 2040
2090 GOSUB 55
2100 FOR J=P+69 TO P+358
2105 IF PEEK J=23 THEN POKE J,0
2110 IF PEEK J=151 THEN POKE J,8
2115 NEXT J
2120 PRINT AT 9,29;"←←"
2125 PRINT AT 8,28;"←←←←"
2130 PRINT AT 0,27;"←←←←←"
2135 PRINT TAB 29;"←←←"
2140 PRINT AT 6,29;"←←"
2145 LET REPLAY=1
2150 IF H<>-1 THEN GOSUB 3090
2155 GOTO 195
2200 GOSUB 55
2210 GOSUB 3035
2220 GOTO 95
3000 CLS
3002 PRINT TAB 10;"BATTLESHIPS"
3004 PRINT TAB 10;"-----------"
3006 PRINT AT 3,0;"SET UP YOUR←←SHIPS←←ON←←THE←←TOPHALF
```

I am not going to explain the various sections in great detail this time, but will concentrate on the more interesting features. But don't just type in the listing blindly; make sure you can understand exactly how each section works before entering the next one.

The key to the program is in the way grid coordinates, entered by the player or selected by the computer, are converted into screen memory positions to be POKEd with the appropriate character. This is handled by two of the early subroutines, and by one at line 980.

Subroutine 25 accepts the player's entry, and checks that it is a valid grid coordinate. (You will notice that, apart from the start and finish options, which are handled by INKEY$ routines. ALL player entries are in the form of grid coordinates, so line 10 is the only INPUT line in the entire program.)

Subroutine 10 converts the grid position into a screen position, with reference to variable P, which was defined as the D-FILE value at the start of the program (line 150). Line 10 sets A to the address of the screen position, and line 15 sets PA (PEEK A) to the character which currently occupies that position.

It is clear from this that, because the player's ships are visibly displayed on the screen, there is no need to use any other part of memory to define their positions. The same is not true, however, of the computer's ships; these positions must be stored in an array C, which duplicates the computer's grid, and in which 0 = absence of a ship, 1 = presence of a ship and 2 = a ship which has been 'hit.' Hence, in line 635, the effect of a player's shot is determined by converting the grid coordinate into a simple number (1–234), and the element of array C corresponding to this number is checked (line 645).

When the computer takes its shots, the process is reversed; it first chooses a number in the range of 1–234, and THEN converts this into a grid coordinate (subroutine 980).

The random number selection routine in Battleships is a vast improvement on those we have seen in the previous programs, and is worth studying in detail. There are 234 (9 x 26) possible 'target' positions, so the normal random number routine would use a line like:

```
LET X = INT (RND*234) + 1
```

and then a line which checked whether X had been chosen before, reverting to the above line for re-selection if it had. This would be fine in the early stages of the game; there are 234 targets to choose from, and the chance of duplication would be slight. As the game progressed, however, there would be fewer and fewer unused targets, and the computer would take longer and longer to find a 'new' value for X.

BATTLESHIPS *(Continued)*

```
      ←OF THE SCREEN."
3007 PRINT
3008 PRINT "ALL COORDINATES SHOULD BE IN THEFORM
      ←LETTER FOLLOWED BY NUMBER."
3009 PRINT
3010 PRINT "THE COMPUTER WILL THEN PLACE ITSSHIPS."
3011 PRINT
3012 PRINT "WHEN PROMPTED, ENTER COORDINATESOF YOUR FIRST SHOT."
3014 PRINT AT 20,3;"PRESS ANY KEY TO CONTINUE"
3015 IF INKEY$="" THEN GOTO 3015
3017 CLS
3020 PRINT "THE WINNING SCORE IS 32 HITS."
3021 PRINT
3022 PRINT "YOU MAY RESIGN BY TYPING 'QUIT'."
3023 PRINT
3024 PRINT "AFTER A←←COMPUTER←←WIN, OR←←YOURRESIGNATION,
      ←THE POSITION OF THECOMPUTERS←←REMAINING
      ←SHIPS←←WILLBE REVEALED."
3025 PRINT
3026 PRINT "AT THE END OF THE GAME, OPTION SWILL←←ALLOW
      ←←YOU←←TO RETAIN←←THESAME POSITIONS←←FOR
      ←YOUR←←SHIPS.(THE COMPUTER, OF←←COURSE,
      ←←WILLREPOSITION ITS SHIPS.)"
3035 PRINT AT 17,9;"LEVEL 1: EASY"
3040 PRINT AT 19,9;"LEVEL 2: HARD"
3045 PRINT AT 21,8;"(PRESS←←1 OR 2)"
3050 LET I$=INKEY$
3060IF I$<"1" OR I$>"2" THEN GOTO 3050
3070 LET H=VAL I$-2
3080 IF H THEN RETURN
3090 LET H=0
3095 LET H1=0
3100 LET H2=0
3105 DIM H(4)
3110 LET H(1)=-1
3120 LET H(2)=1
3130 LET H(3)=26
3140 LET H(4)=-26
3150 RETURN
4000 CLS
4010 LET W$="THANKS FOR PLAYING *BATTLESHIPS*"
4030 FOR J=1 TO LEN W$
4040 PRINT AT 10,0;W$
4050 LET W$(J)=CHR$ (CODE W$(J)+128-256*(CODE W$(J)>127))
4060 NEXT J
4070 GOTO 4030
9000 SAVE "BATTLESHIPS"
9010 RUN
```

Compare this with the actual routine used (lines 715–730). An array M is first set up (lines 505–515), in which the 234 elements contain the numbers 1 to 234, respectively. The computer then selects a random number R, in the range of 1 to the number of VACANT targets (line 715: G is a counter of the number of shots fired). X is then extracted from the Rth element of array M. Obviously, for shot number 1, X will have the same value as R, but lines 725 and 730 swap the contents of the Rth element of M with the contents of the (235 – G)th element, so that the array now contains all unused targets in its lower elements, (1 to (234 – G) and all used targets in its upper elements (235 – G to 234). In future passes through the routine, as G increases, so line 715 will ALWAYS select a value of R which relates to one of the UNUSED targets.

The routine at lines 325–355 also deserves special attention. The player is allowed to enter any starting position for a ship (line 290) and an end position in any of the 4 possible directions (line 315). The routine checks not only the validity of the second entry in relation to the first (note that the length of each type of ship is specified by the 14th element of its name string E$, so make sure you enter all the strings correctly), but also determines the direction in which to draw in the complete ship. A similar routine is used when the computer is choosing positions (lines 435–480). This time both the starting positions and the direction are chosen randomly, and the rest of the routine ensures that the whole ship will fit in the grid.

Lines 665–670 and 770–775 modify the machine code in line 1 so that a player hit will flash the computer's grid, and vice versa.

COMPUTER STRATEGY

Obviously the main interest in this program is the way in which the computer is given an 'intelligent' strategy. At level 2 the computer will ensure that a complete ship is destroyed once 1 part of it has been hit, and (this part is based on my observations of the psychology of Battleships players) it will also search around the ends of each ship to detect fleets which are 'chained' together. The intelligence of the program is as follows:

Subroutine 3090 initializes an array H, which merely contains the 4 directions from any grid position. Simple variable H doubles as a pointer to the current hit position, and as a flag to indicate whether the intelligence feature is to be used at all (– 1 = no; 0 = yes, but no hit; any positive value = yes). H1 is a pointer to the first hit which located the ship currently being destroyed, and H2 is a flag to indicate whether or not the current ship has been completely sunk.

The intelligence routine (lines 850–975) is called whenever H <> – 1 (line 710), but line 850 reverts to random search strategy (non-strategy?) if a ship has not been located. When a ship is being 'worked

on,' shots are fired in each of the 4 directions around the first hit until the axis of the ship is determined. Then the elements of array H are swapped around so that the computer will continue to fire along this axis until the end of the ship is found. At this point, stage 2 of the strategy comes into play, and 'chains' are searched for. Whether or not this is an effective policy I don't know. Certainly it is very effective against 'typical' human opponents, but knowing that the computer ALWAYS uses these tactics means that it is possible to make it 'waste time' by NOT chaining your ships together.

When a chain is found, it is pursued to the end, again by swapping elements of array H, if necessary. H1, however, is retained, so that at the end of the chain, the other end of the original ship is destroyed. The H and H2 flags are not reset until the computer is satisfied that it has nothing further to gain from pursuing its strategy, and random search is resumed (lines 970–975).

The intelligence feature does, of course, tend to nullify the 'instant response' feature of the random number routine described earlier. Things are speeded up, however, by line 887, which checks to see if the relevant element of array M has been disturbed from its original value; if not, the routine works almost as quickly as the random search.

PROGRAM IMPROVEMENTS

I don't want you to think that the programs in this book are all perfect. (What do you mean, you didn't think that for a moment?) There are several improvements you could make to Battleships, particularly with regard to the intelligence features. When I wrote the program, it had been a long time since I played Battleships, and the strategy was good enough to beat me at least half the time. However, now that I am an addict, the next step is to add level 3, where the computer uses a 'bomb pattern' check, to avoid wasting shots (there is no point in firing at A1 if A2 and B1 have already been used), and does not use the 'chain search' so consistently.

Level 4 would be even closer to human intelligence. The computer could keep a check of the lengths of destroyed ships, and modify its bomb patterns to search most effectively for the remaining ships. This is not as easy as it sounds; it is possible to disguise ship lengths by close grouping, and even the computer's supposedly random ship positioning often manages to create some very deceptive groups. If any readers decide to develop 'Battleships' to level 4 or beyond, I would be interested to see their results.

Fortress

This program takes the concept of 'computer intelligence' one stage further. The computer plays 2 games: 1 is extremely difficult to beat; the other is impossible to beat!

The rules of the game are contained in the listing, lines 5000–5130, and you should read these first to familiarize yourself with the various options and objectives. The 'board' is illustrated on page 115, and you may find this diagram more helpful at first than the screen 'map,' as the connecting passages are NOT shown on the screen display. Notice that each room, marked with a letter, is connected to its 4 nearest neighbors.

The game is rather unusual in that the 2 players, attacker and defender, have completely different allowable moves, and hence the program is 2 games in 1; you can play the attacking role to the computer's defense, or defend against the computer's attack. (In fact, the program contains 3 games in 1, because there is also an option for 2 human opponents to play against each other.)

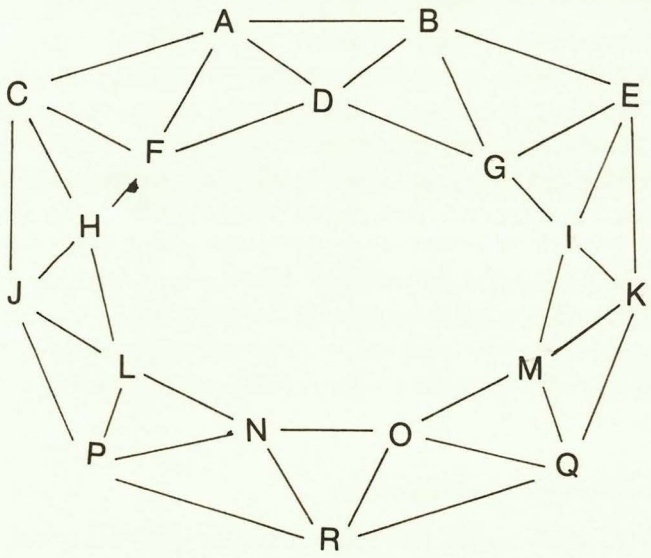
At first sight it would seem that the defender's lot is not a happy one. He only has a choice of 2 moves per turn, and has to spend most of the game sitting around waiting to be captured. The attacker not only has much greater mobility, but also has the first move, which must surely be an advantage. Unfortunately, if you believe this you are heading for trouble. You may be able to beat a human opponent playing the attacker, but the computer's defense is infallible!

So perhaps the defender's role is the best option after all? Try playing defender to the computer's attack, however, and you will still find that you will be beaten most of the time!

(This program is great for demoralizing any of your friends who claim that "computers will never be able to play as well as humans.")

NOTE

If you would like to work out the optimum strategy for both players yourself before I give away any secrets, please do not read any further, and key in the program 'blindfold' (if you can type blindfolded) without trying to comprehend the listing at this stage.



NOTES

The connecting passages will NOT be indicated when this map appears on screen.

The defender will be marked by the character : +
The attacker will be marked by the character : ✸
The defender starts in room D.
The attacker starts in room R.

INITIALIZATION AND GAME ORGANIZATION (lines 1–245)

The fortress map or playing board is POKEd to the screen by the subroutine in line 2, using the same technique as in the previous program. The only difference this time is that the required screen positions are not calculated by the program, but have been worked out on graph paper beforehand, and are held in array A. Array L$ holds the legal moves (for the attacker) from each room. Arrays C and G are used by the computer in its move analysis. Most of the variable names are mnemonic: AP is Attacker Position, NAP is Next Attacker Position, etc. NO was intended to be a mnemonic for Number Of players, but in fact in the 1-player game, it takes the rather strange values of 38 and 41, depending on whether the player is in attack or defense.

HUMAN ATTACK (lines 1000–1270)

Lines 1040–1080 check the legality of the entered move, and illegal moves are rejected and the appropriate Error report string (E$) is flashed (lines 1090–1120 and subroutine 20–24). Obviously the condition where the attacker has no legal move is fatal, so line 1130 terminates the game at this point. Once a move has been accepted, the passage used is blocked, IN BOTH DIRECTIONS, by lines 1160–1190. (The mechanism for this is that the appropriate elements of array L$ are adjusted to 128 plus their original value.)

Line 1210 checks for a win, and finally the new position is POKEd to the screen in lines 1200 and 1260.

HUMAN DEFENSE (lines 2000–2150)

This is naturally a much simpler version of the same routine. Move validation is confined to line 2030; there is only 1 necessary error report (line 2040) and the 'passage-blocking' is omitted entirely.

In the 2-player game, play alternates between routines 1000 and 2000 until such time as a win is signaled and the 'end routine' at line 7000 is called.

COMPUTER STRATEGY

The computer's strategy for defense is contained in lines 8000–8020. (Yes, that's all there is!) Lines 8500–8775 are the computer's attack. Yout will notice immediately one of the truisms of computer games programming: an infallible strategy is much easier to program than a merely 'best' strategy!

I an NOT going to give the game away completely by explaining how the defense strategy works. It is not too difficult to deduce by looking at these few lines in relation to the way array C was initialized at lines 30–48.

FORTRESS

```
1 GOTO 30
2 LET P=PEEK 16396+256*PEEK 16397
3 FOR J=1 TO 18
4 POKE P+A(J),J+37
5 NEXT J
6 RETURN
7 PRINT AT 20,0;"PRESS M←←TO RECALL MAP OR R←←FORMORE RULES."
8 LET I$=INKEY$
9 IF I$<>"M" AND I$<>"R" THEN GOTO 7
10 RETURN
15 CLS
16 PRINT TAB 11;"FORTRESS."
17 RETURN
20 PRINT AT 21,0;E$
21 FOR J=1 TO 30
22 NEXT J
23 PRINT AT 21,0;"←←←←←←←←←←←←←←←←←←←"
24 RETURN
30 DIM C(18)
31 LET C(1)=1
32 LET C(2)=2
33 LET C(3)=3
34 LET C(4)=3
35 LET C(5)=3
36 LET C(6)=2
37 LET C(7)=1
38 LET C(8)=1
39 LET C(9)=2
40 LET C(10)=2
41 LET C(11)=1
42 LET C(12)=3
43 LET C(13)=3
44 LET C(14)=2
45 LET C(15)=1
46 LET C(16)=1
47 LET C(17)=2
48 LET C(18)=3
50 DIM A(18)
51 LET A(1)=46
52 LET A(2)=52
53 LET A(3)=106
54 LET A(4)=115
55 LET A(5)=124
56 LET A(6)=176
57 LET A(7)=186
58 LET A(8)=240
59 LET A(9)=254
60 LET A(10)=304
61 LET A(11)=322
62 LET A(12)=373
63 LET A(13)=385
64 LET A(14)=443
```

The attack strategy is based on the computer playing 'best possible' moves, until such time as the human player makes a mistake (as they tend to do, these humans). Then the computer is able to 'take over' the strategy it would use in defense (not the same moves, of course, but the same strategy, played outside the A–B–D area), and go on to win.

Therefore the major portion of the attack strategy is involved with determining:

1) Has the defender made an error? (i.e., Is a 'potential winning move' available?), and

2) If not, then what is the best possible move remaining?

Both these considerations are combined in a single routine which 'weights' all 4 options available according to the following table:

Illegal move	–30
Potential winning move	+15
Actual winning move	+30
Losing move	–10
Other moves	– 1 for each blocked exit from the new room

(The weighting for 'other moves' uses the look-ahead routine in lines 8600–8610.)

Lines 8700–8775 select the best move, and where moves are of equal merit, the move to the lowest lettered room is selected. Notice that 'potential winning move,' i.e., one which will enable the computer to adopt its infallible strategy and 'illegal move' are not singled out for special treatment, but both merely weighted along with the other possiblities. The losing situation where all 4 moves are illegal is detected in line 8745, causing the computer to concede defeat in line 1110.

I should mention of course that for the computer's first move (only), this entire process is skipped, and a random choice is made (line 8510).

FORTRESS *(Continued)*

```
65 LET A(15)=447
66 LET A(16)=471
67 LET A(17)=485
68 LET A(18)=544
70 DIM L$(18,4)
71 LET L$(1)="BCDF"
72 LET L$(2)="ADEG"
73 LET L$(3)="AFHJ"
74 LET L$(4)="ABFG"
75 LET L$(5)="BGIK"
76 LET L$(6)="ADCH"
77 LET L$(7)="BDEI"
78 LET L$(8)="CFJL"
79 LET L$(9)="GEKM"
80 LET L$(10)="CHLP"
81 LET L$(11)="EIMQ"
82 LET L$(12)="HJPN"
83 LET L$(13)="IKQO"
84 LET L$(14)="LOPR"
85 LET L$(15)="MNQR"
86 LET L$(16)="JLNR"
87 LET L$(17)="MKOR"
88 LET L$(18)="NOPQ"
89 DIM G(4)
90 LET AP=18
91 LET DP=4
95 LET G=0
100 GOSUB 15
105 PRINT AT 5,0;"1: 1 PLAYER GAME"
110 PRINT AT 7,0;"2: 2 PLAYER GAME"
115 PRINT AT 9,0;"3: RULES"
120 LET I$=INKEY$
125 IF I$<"1" OR I$>"3" THEN GOTO 120
130 IF I$="3" THEN GOTO 5000
140 IF I$="1" THEN GOTO 6000
150 LET NO=2
200 GOSUB 15
205 GOSUB 2
210 FOR J=1 TO 30
215 NEXT J
220 POKE P+A(AP),136
225 POKE P+A(DP),21
226 IF INKEY$=I$ THEN GOTO 226
230 IF NO<40 THEN GOTO 1000
235 GOTO 8500
240 IF NO<>38 THEN GOTO 2000
245 GOTO 8000
1000 PRINT AT 21,0;"ATTACKER MOVE (↑H↑)"
1010 LET I$=INKEY$
1020 IF I$="" OR I$=CHR$ 118 THEN GOTO 1010
1030 LET N=0
1040 FOR J=1 TO 4
1050 IF L$(AP,J)=I$ THEN GOTO 1150
1060 IF L$(AP,J)=CHR$ (CODE I$+128) THEN LET N=N+5
```

FORTRESS *(Continued)*

```
1070 IF L$(AP,J)>CHR$ 128 THEN LET N=N+1
1080 NEXT J
1090 LET E$="NO PASSAGE←←←←←←←←←"
1100 IF N>4 THEN LET E$="PASSAGE BLOCKED←←←←←"
1110 IF N=4 OR N=9 THEN LET E$="ALL PASSAGES BLOCKED"
1120 GOSUB 20
1130 IF E$(1)="A" THEN GOTO 2120
1140 GOTO 1000
1150 LET NAP=CODE I$-37
1160 LET L$(AP,J)=CHR$ (CODE L$(AP,J)+128)
1170 FOR J=1 TO 4
1180 IF L$(NAP,J)=CHR$ (AP+37) THEN LET L$(NAP,J)=
      CHR$ (CODE L$(NAP,J)+128)
1190 NEXT J
1200 POKE P+A(AP),AP+37
1210 IF PEEK (P+A(NAP))<>21 THEN GOTO 1250
1220 LET W$="ATTACKER"
1230 POKE P+A(NAP),136
1240 GOTO 7000
1250 LET AP=NAP
1260 POKE P+A(AP),136
1265 IF INKEY$=I$ THEN GOTO 1265
1270 GOTO 240
2000 PRINT AT 21,0;"DEFENDER MOVE (+)"
2010 LET I$=INKEY$
2020 IF I$="" OR I$=CHR$ 118 THEN GOTO 2010
2030 IF I$="A" OR I$="B" OR I$="D" THEN GOTO 2070
2040 LET E$="STAY IN YOUR AREA←←←"
2050 GOSUB 20
2060 GOTO 2000
2070 LET NDP=CODE I$-37
2080 IF NDP=DP THEN GOTO 2010
2090 POKE P+A(DP),DP+37
2100 IF PEEK (P+A(NDP))<>136 THEN GOTO 2140
2110 POKE P+A(NDP),21
2120 LET W$="DEFENDER"
2130 GOTO 7000
2140 LET DP=NDP
2150 GOTO 225
5000 CLS
5005 GOSUB 2
5010 PRINT AT 18,0;"THIS IS A MAP OF THE FORTRESS."
5015 GOSUB 7
5020 IF I$="M" THEN GOTO 5015
5025 CLS
5030 PRINT "THIS IS A GAME FOR 2 PLAYERS, OR1 PLAYER
      ←AND THE COMPUTER."
5035 PRINT
5040 PRINT "THE DEFENDER (+)←←IS←←RESTRICTEDTO ROOMS
      ←A,B AND D. THE ATTACKER (↑H↑) MAY OCCUPY
      ←ANY ROOM.←←EITHERPLAYER MAY PASS TO←←
      AN←←ADJACENTROOM BY←←A←←CONNECTING←←
      PASSAGE,BUT THE ATTACKER←←IS←←RESTRICTEDIN
      ←THAT,ONCE HE HAS USED A PASS-AGE, THAT
```

FORTRESS *(Continued)*

```
      ←←PASSAGE←←IS←←BLOCKED,AND MAY NOT
      ←BE REUSED."
 5045 PRINT
 5050 PRINT "THE←←OBJECT←←OF←←THE←←GAME IS←
      TOCAPTURE YOUR OPPONENT BY←←MOVINGINTO
      ←THE ROOM WHICH HE CURRENTLYOCCUPIES.
      (CAPTURES CAN ONLY TAKEPLACE IN ROOMS
      ←A,B OR D, AS←←THEDEFENDER CANNOT
      ←LEAVE THESE.)"
 5055 GOSUB 7
 5060 IF I$="R" THEN GOTO 5080
 5065 CLS
 5070 GOSUB 2
 5075 GOTO 5055
 5080 CLS
 5085 PRINT "THE←←ATTACKER←←MAY LOSE THE GAMEBY
      ←BECOMING←←TRAPPED←←IN←←A ROOMFROM WHICH
      ←←ALL←←PASSAGES←←←HAVEBECOME BLOCKED."
 5090 PRINT
 5095 PRINT "AT THE START OF PLAY:-"
 5100 PRINT
 5105 PRINT "THE ATTACKER IS IN ROOM R,"
 5110 PRINT "THE DEFENDER IS IN ROOM D."
 5115 PRINT
 5120 PRINT "N.B. THERE IS NO PASSAGE BETWEENROOMS
      ←C-D OR BETWEEN ROOMS D-E."
 5125 PRINT
 5130 PRINT "THE ATTACKER ALWAYS MOVES FIRST."
 5135 PRINT AT 19,0;"PRESS R TO REPEAT RULES."
 5140 PRINT TAB 6;"M TO RECALL MAP."
 5145 PRINT TAB 6;"P TO PLAY."
 5150 LET I$=INKEY$
 5155 IF I$="P" THEN GOTO 100
 5160 IF I$="R" THEN GOTO 5025
 5165 IF I$<>"M" THEN GOTO 5150
 5170 CLS
 5175 GOSUB 2
 5180 GOTO 5135
 6000 PRINT AT 11,0;"D: YOU WILL PLAY THE DEFENDER"
 6010 PRINT AT 13,0;"A: YOU WILL PLAY THE ATTACKER"
 6020 LET I$=INKEY$
 6030 IF I$<>"D" AND I$<>"A" THEN GOTO 6020
 6040 LET NO=CODE I$
 6050 GOTO 200
 7000 IF W$(1)="D" AND NO=41 THEN LET W$="YOU"
 7010 IF NO=38 OR (NO=41 AND W$(1)="A") THEN LET W$="I"
 7020 PRINT AT 21,0;W$;"←WIN";
 7030 IF NO=2 THEN PRINT "S";
 7035 PRINT "←←←←←←←←←←"
 7040 PRINT AT 18,0;"PRESS P TO PLAY AGAIN"
 7045 PRINT "OR X TO STOP"
 7050 IF INKEY$="P" THEN GOTO 71
 7060 IF INKEY$<>"X" THEN GOTO 7050
 7070 PRINT AT 21,0;"THANKS FOR PLAYING FORTRESS"
```

FORTRESS *(Continued)*

```
7100 GOTO 9999
8000 LET I$="A"
8010 IF C(AP)=2 THEN LET I$="B"
8020 IF C(AP)=3 THEN LET I$="D"
8030 PRINT AT 21,0;"MY MOVE:←";I$;"←(PRESS N/L)"
8040 IF INKEY$<>CHR$ 118 THEN GOTO 8040
8050 PRINT AT 21,11;"←←←←←←←←←←"
8060 IF NO=38 THEN GOTO 2070
8070 GOTO 1030
8500 LET G=G+1
8505 IF G>1 THEN GOTO 8520
8510 LET I$=L$(AP,INT (RND*4)+1)
8515 GOTO 8030
8520 PRINT AT 21,0;"I AM THINKING←←←←"
8525 FOR J=1 TO 4
8530 LET G(J)=0
8535 LET PM=CODE L$(AP,J)-37
8540 IF PM<128 THEN GOTO 8560
8545 LET G(J)=-30
8550 GOTO 8620
8560 IF C(PM)<>C(DP) THEN GOTO 8580
8565 LET G(J)=15
8570 IF PM=DP THEN LET G(J)=30
8575 GOTO 8620
8580 IF PM>4 OR PM=3 THEN GOTO 8600
8585 LET G(J)=-10
8590 GOTO 8620
8600 FOR K=1 TO 4
8605 IF L$(PM,K)>CHR$ 128 THEN LET G(J)=G(J)-1
8610 NEXT K
8620 NEXT J
8700 LET K=G(1)
8710 FOR J=2 TO 4
8720 IF G(J)>K THEN LET K=G(J)
8730 NEXT J
8740 IF K<>-30 THEN GOTO 8760
8745 LET N=9
8750 GOTO 1110
8760 FOR J=1 TO 4
8765 IF G(J)<>K THEN NEXT J
8770 LET I$=L$(AP,J)
8775 GOTO 8030
9000 SAVE "FORTRESS"
9010 RUN
```

Wordsearch

Wordsearch is a puzzle much loved by commuters for passing the time on monotonous train journeys. At the moment, of course, it is not terribly convenient to use your computer on a train (although maybe the

WORDSEARCH

```
1 RAND
5 CLS
10 DIM A$(10,10)
15 DIM L(10)
20 LET A$(1)="SINCLAIR"
25 LET L(1)=8
30 LET A$(2)="COMMODORE"
35 LET L(2)=9
40 LET A$(3)="APPLE"
45 LET L(3)=5
50 LET A$(4)="TANDY"
55 LET L(4)=5
60 LET A$(5)="ACORN"
65 LET L(5)=5
70 LET A$(6)="BASIC"
75 LET L(6)=5
80 LET A$(7)="FORTRAN"
85 LET L(7)=7
90 LET A$(8)="PASCAL"
95 LET L(8)=6
100 LET A$(9)="COBAL"
105 LET L(9)=5
110 LET A$(10)="FORTH"
115 LET L(10)=5
120 DIM S(10,2)
125 DIM D(10,2)
130 LET R$=""
135 DIM B$(32)
150 DIM W$(15,15)
160 PRINT AT 5,10;"WORDSEARCH"
170 PRINT AT 10,0;"PRESS C FOR COMPUTER WORDS";AT 13,0;
      "OR E TO ENTER YOUR OWN WORDS"
180 IF INKEY$="C" THEN GOTO 280
190 IF INKEY$<>"E" THEN GOTO 180
200 CLS
210 FOR J=1 TO 10
220 PRINT AT J*2,0;"WORD NO.";J,
230 INPUT T$
240 LET L(J)=LEN T$
250 IF L(J)<1 OR L(J)>10 THEN GOTO 230
255 PRINT T$
260 LET A$(J)=T$
270 NEXT J
280 CLS
285 PRINT AT 10,0;"GRID FORMATION IS IN FAST MODE"
290 LET L=SQR (RND*RND*RND)
300 FAST
310 FOR J=1 TO 10
320 LET L=L(J)-1
330 LET SX=INT (RND*15)+1
340 LET SY=INT (RND*15)+1
350 LET DX=INT (RND*3)-1
```

Timex 1000 will feature its own battery pack and flat screen?), but at least you will not have to worry about finishing before you arrive at Waterloo.

The program sets up a 15x15 grid of 'random' letters, in which 10 words have been cleverly hidden. The words may run horizontally, diagonally or vertically, backwards or forwards, which, even I can calculate, makes 8 possible directions. You have the option, of allowing the computer to use its vocabulary of 10 (rather single-minded) words, or of entering your own words each time. In fact, using the latter option does not, in my opinion, make the puzzle any harder (unless you use stupid words like 'A,' which will make it impossible), because the computer will create a new grid every run, so that even with its standard vocabulary no 2 puzzles will ever be the same.

Whichever option you chose, the computer will go into FAST mode to set up the grid, which may be a little disconcerting as it looks exactly as if the program has crashed! After a few seconds, however, the grid will appear, with the 10 words listed by its side. When (if?) you spot a word, 3 inputs are required to prove that you have located it correctly:

WORD NUMBER: (1 to 10)

START (VERTICAL): The vertical coordinate of the first letter of the word (1 to F; the Timex 1000 not only has a vocabulary composed entirely of computer words, it also numbers the grid in hex!)

START (HORIZONTAL): ditto (horizontal coordinate, 1 to F)

(It is not necessary to specify the direction of the word; if you get its starting position correct, obviously you have located the entire word.)

Assuming that your entries are correct, the word will be marked on the grid in inverse lettering, and an asterisk will appear next to that word on the list. Remember that words MAY overlap, so do not disregard inverse letters when searching for other words.

If you make an incorrect guess (as if you would ever do that!), the polite message 'WRONG' appears, and you go back to step 1. If, after several hours, you have still not found all the words, entering 'QUIT' in reply to 'WORD NO. ?' will reveal the remaining words.

If you want to practice before keying in the listing, a sample grid is shown on page 129.

INITIALIZATION AND 'MENU' (lines 1–290)

Lines 10–115 give the computer its vocabulary. If you are wondering why it is necessary to specify the length of each word (line numbers ending in 5), when the Timex 1000 has a perfectly good LEN function, remember that DIMensioning the string array (line 10) predetermines that each array element has 10 characters. Any unused elements will be padded out with blanks, so, for example, A$(5) is not "ACORN," but "ACORN ←←←←", and LEN A$(5) will always be 10.

WORDSEARCH *(Continued)*

```
360 LET DY=INT (RND*3)-1
370 IF DX=0 AND DY=0 THEN GOTO 350
380 LET FX=SX+(L*DX)
390 LET FY=SY+(L*DY)
400 IF FX<1 OR FX>15 OR FY<1 OR FY>15 THEN GOTO 330
410 FOR K=0 TO L
420 IF W$(SX+(DX*K),SY+(DY*K))=A$(J,K+1) THEN GOTO 440
430 IF W$(SX+(DX*K),SY+(DY*K))<>"←" THEN GOTO 330
440 NEXT K
450 LET S(J,1)=SX
460 LET S(J,2)=SY
470 LET D(J,1)=DX
480 LET D(J,2)=DY
490 LET R$=R$+A$(J, TO L+1)
500 FOR K=0 TO L
510 LET W$(SX+(DX*K),SY+(DY*K))=A$(J,K+1)
520 NEXT K
530 NEXT J
550 FOR J=1 TO 15
555 FOR K=1 TO 15
560 IF W$(J,K)="←" THEN LET W$(J,K)=R$(INT (RND*LEN R$)+1)
565 NEXT K
570 NEXT J
600 CLS
610 FOR J=1 TO 15
615 PRINT AT 0,J+1;CHR$ (J+156)
620 PRINT AT J+1,0;CHR$ (J+156);"←";W$(J)
630 IF J<=10 THEN PRINT AT J,18;J;TAB 21;A$(J)
640 NEXT J
650 PRINT AT 12,19;"TYPE 'QUIT'";AT 13,20;"TO RESIGN"
660 SLOW
670 LET SCORE=0
700 PRINT AT 18,0;"WORD NO. ?"
710 GOSUB 1000
712 IF T$="QUIT" THEN GOTO 2000
714 LET J=CODE T$-28
716 IF J>10 THEN GOTO 710
717 IF A$(J,1)="*" THEN GOTO 710
718 PRINT AT 18,0;A$(J)
720 PRINT "START (VERTICAL)"
730 GOSUB 1000
735 LET X=CODE T$-28
740 PRINT "START (HORIZONTAL)"
750 GOSUB 1000
755 LET Y=CODE T$-28
760 IF S(J,1)=X AND S(J,2)=Y THEN GOTO 800
770 PRINT "WRONG"
775 FOR K=1 TO 20
780 NEXT K
785 GOSUB 1500
790 GOTO 700
800 LET SCORE=SCORE+1
```

Arrays S and D will hold the starting coordinates of each word in the grid, and their respective directions. R$ is a 'dummy' string, the significance of which will be seen shortly. W$ is the grid itself, and B$ (line 135) is part of the improved 'delete a line' routine I promised you earlier. B$ is padded out entirely with blanks, and so a screen line can be overprinted with the simple command 'PRINT B$' (see subroutine 1500).

Lines 200–270 allow the user to replace the standard vocabulary with other words, if the appropriate response is made to the INKEY$ routine (lines 170–190). Note that line 250 ensures that each word is not more than 10 characters long, otherwise a 'bad subscript error' (report code 2/260) would occur.

Line 290 is utter nonsense; it merely gives the computer a 'little' calculation to do so that the message in line 285 will be retained on screen for a few moments, hopefully preventing the unwitting user from having a heart attack when fast mode is entered in line 300.

GRID FORMATION (lines 300–660)

This involves selecting a random starting position for each word (X and Y coordinates, lines 330–340), and a random direction (again X and Y coordinates, each in the range of -1 to $+1$, lines 340–350). Notice that these latter 2 lines theoretically allow a word to have a zero X-direction coordinate, i.e., the word is totally vertical, and a zero X-direction coordinate (totally horizontal)! Philosophers may care to speculate how such a word would be represented; for the puposes of this program however it is sufficient merely to reject this anomaly, and reselect proper coordinates (line 370).

Having checked that the chosen starting position and direction will allow the word to finish within the grid (line 400), the loop from line 410 to 440 checks to see if the word crosses a previously positioned word. If it does, and the letters at the cross-over point correspond, that is alright (line 420); otherwise a new starting position is selected (line 430). Once a position has passed all these tests, the temporary variables SX, SY, DX and DY are stored permanently in their respective arrays S and D (lines 450–480).

Line 490 is very important. It ensures that the dummy string R$ consists of all the letters of the hidden words. It is from this string that the 'random' letters which pad out the grid will be selected, thus creating maximum confusion by ensuring that the grid is composed entirely of letters used in the hidden words, and also that the frequency of each letter is similar to its frequency in the hidden words themselves (lines 550–570; the actual words having been placed in the grid in lines 500–520).

The screen is then PRINTed, while still in fast mode, a simple but often-neglected time-saving device.

WORDSEARCH *(Continued)*

```
810 PRINT AT J,20;"*"
820 GOSUB 3000
830 IF SCORE=10 THEN GOTO 4000
840 LET A$(J)="*"+A$(J)
850 GOTO 785
1000 INPUT T$
1010 IF T$="QUIT" THEN RETURN
1015 IF T$="10" THEN LET T$="A"
1020 IF LEN T$<>1 THEN GOTO 1000
1030 IF T$<"1" OR T$>"F" THEN GOTO 1000
1040 RETURN
1500 FOR K=18 TO 21
1510 PRINT AT K,0;B$
1520 NEXT K
1530 RETURN
2000 FOR J=1 TO 10
2010 GOSUB 3000
2020 NEXT J
2030 GOTO 4000
3000 FOR K=0 TO L(J)-1
3010 PRINT AT S(J,1)+(D(J,1)*K)+1,S(J,2)+(D(J,2)*K)+1;
3020 LET P=PEEK 16398+256*PEEK 16399
3030 IF PEEK P<128 THEN POKE P,PEEK P+128
3040 NEXT K
3050 RETURN
4000 GOSUB 1500
4010 IF SCORE=10 THEN PRINT AT 18,0;"CONGRATULATIONS"
4020 PRINT AT 20,0;"ANOTHER GAME (Y/N)"
4030 IF INKEY$="Y" THEN RUN
4040 IF INKEY$<>"N" THEN GOTO 4030
4050 CLS
4060 PRINT AT 10,0;"BYE."
4070 GOTO 9999
9000 SAVE "WORDSEARCH"
9010 RUN
```

INPUT HANDLING (lines 670–850)

This section is quite straightforward. Once again, all actual INPUTs are delegated to a single subroutine (1000). Line 1015 reconciles the apparent anomaly between the decimal number 10 (entered as a word number) and the hex number A (a grid coordinate). Notice how Timex 1000 character codes are arranged so that CODE "9" = 37; CODE "A" = 38, etc., i.e., converting a string input to a number by using CODE rather than VAL (as in lines 714, 735, 755) enables the alphabet to be used as an extension of the numerals.

INVERSION SUBROUTINE (lines 3000–3050)

Called after a correct entry, or by subroutine 2000 after 'QUIT,' this directs the PRINT AT position to the letter to be inverted (line 3010), then uses the DF-CC system variable to detect whether the character has already been inverted (lines 3020–3030).

PROGRAM IMPROVEMENTS

The following modifications occurred to me after 'finishing' this program. Please feel free to add these features or any others of your own if they appeal to you.

1) Add a timing routine (using system variable FRAMES; see 'Clock/Calendar')

2) Add an end option to reform the same grid (using a reinvert subroutine similar to subroutine 3000), so that several players may compete 'against the clock.'

3) Add an end option to allow 'user-defined' vocabularies to be SAVEd on tape and/or reused immediately.

SAMPLE 'WORDSEARCH' GRID

Here is a typical example of the kind of grid created by this program, using its own vocabulary:

```
   123456789ABCDEF

                        1  SINCLAIR
1 PHYENCLDOEPCSCO       2  COMMODORE
2 ATLDLCAESRCBEOA       3  APPLE
3 ARACRPTTMONFFBS       4  TANDY
4 MOOCCSPPLDLCOOM       5  ACORN
5 TFCRASMALORRDLA       6  BASIC
6 BMYAOMCYNMPOCAO       7  FORTRAN
7 AL INLOTFLMANNCM      8  PASCAL
8 SAAAPADRROCCRPP       9  COBOL
9 INDONOCLACNARRP       10 FORTH
A CCRDRLPSFSSIRCI
B SRYROACOACYRCAL
C NCIRCCOFAPCCAYB
D ASLFANSINCLAIRR
E AMTFORTRANCRAAR
F NMICIRAOMRTPEDF

WORD NO. ?
```

Note: The Timex 1000 characters are formed in a 'square matrix' (8x8), whereas this printer produces rectangular (7x6) characters. Therefore the grid produced on screen will be perfectly square, unlike this reproduction which is elongated.)

Kami Kazi Drive

This one is included for my benefit. It is my all-time favourite computer game. You will probably hate it! There are no spaceships, no aliens, no moving graphics of any kind—well at least it is different.

The intention was to create a game which balances the twin elements of skill and chance as near perfectly as possible. When you think about it, most games, computer and otherwise, either rely so heavily on skill that expert players will ALWAYS beat beginners, or so much on chance that you might just as well toss a coin to decide the outcome. 'Kami Kazi Drive,' however, is organized in such a way that, although the expert will USUALLY score better than the novice, less skillful players will win sufficiently often for them not to be discouraged.

(Note: Kami Kazi Drive is essentially a solo game, so 'winning' in this context may mean either beating the 'best so far,' or setting up a personal best, or simply scoring higher than an opponent in a particular several-player session.)

RUNNING INSTRUCTIONS

As usual, the rules of the game are contained in the program listing beginning at line 5020. These will give you some idea of what to do, but the best way to understand the game is to have a few trial runs.

Each game consists of an unspecified number of moves, and for each move 1 INPUT only is required. This should be a NUMBER, indicating the speed at which you wish to travel for that move. Notice that 2 non-numeric 'default' inputs are acceptable: 'P' will maintain the speed at which you are traveling at the present time, and Newline (on its own) will cause an increase of 40 m.p.h. which is maximum acceleration. Any entry which exceeds current speed, plus 40, will not be rejected, but will also be interpreted as maximum acceleration.

The race-track style signboards will illustrate the next bend ahead, and tell you the maximum safe speed for the bend and the number of MOVES (not the distance) before you reach it. Your strategy therefore should be to drive as fast as possible while still managing to get safely around the bends. Penalties are incurred for:

1) Braking (i.e., reducing speed) too severely—probable damage to brakes; possible damage to tires, and

2) Cornering at greater speed than the indicated safe maximum—certain damage to tires; possible damage to brakes; possible drop in oil pressure.

I will not explain these probables and possibles any further at this stage. If you want to work out a mathematically optimum strategy, you must first deduce the exact likelihood of each type of penalty by unraveling the listing. Alternatively you may adopt a 'trial-and-error' approach, which may well be just as successful!

KAMI KAZI DRIVE

```
10 DIM N$(10)
20 LET D$="↑SSSSSSSSSSSSSSSSSSSSSSSSSSSSS↑"
25›DIM E$(32)
30 LET F=0
35 LET S$="M.P.H."
40 LET P$="O/O"
45 LET M$="MILES"
50 LET PR=-1
60 DIM B$(3,7,6)
61 LET B$(3,1)="←←←↑3##↑"
62 LET B$(3,2)="←←↑3#1↑"
63 LET B$(3,3)="←↑3#1↑"
64 LET B$(3,4)="↑3#1↑"
65 LET B$(3,5)="#"
66 LET B$(3,6)="#"
67 LET B$(3,7)="#"
71 LET B$(2,1)="←←↑3##4↑"
72 LET B$(2,2)="←↑3#12#↑"
73 LET B$(2,3)="↑3#1↑"
74 LET B$(2,4)="#"
75 LET B$(2,5)="#"
76 LET B$(2,6)="#"
77 LET B$(2,7)="#"
81 LET B$(1,1)="←↑3##4↑"
82 LET B$(1,2)="↑3#12#4↑"
83 LET B$(1,3)="↑#1←←2#↑"
84 LET B$(1,4)="#←←←←#"
85 LET B$(1,5)="#"
86 LET B$(1,6)="#"
87 LET B$(1,7)="#"
90 GOSUB 5000
95 RAND
100 LET L=0
105 LET O=80
110 LET T=100
120 LET B=100
130 LET D=0
140 LET S=0
142 LET X$="OIL"
144 LET Y$="BRAKES"
146 LET Z$="TYRES"
148 CLS
150 FOR J=0 TO 28 STEP 2
160 PRINT AT 13,J;"←↑3↑"
170 NEXT J
180 GOSUB 800
190 GOSUB 500
195 IF O<5 OR T=0 OR B=0 THEN GOTO 1000
200 PRINT AT 21,5;"ENTER DESIRED SPEED"
210 INPUT T$
220 IF T$="" THEN LET T$=STR$ (40+S)
225 IF T$(1)="P" THEN LET T$=STR$ S
```

Notice that the ever-decreasing oil pressure is unrelated to your driving tactics (apart from the possible drop for cornering too fast, as mentioned above). This means that the game will NOT go on for ever, so the 'ultra-cautious' strategy of keeping within all speed limits is not (generally) to be recommended.

The game is over when either your tire efficiency or brake efficiency has dropped to zero, or your oil pressure has fallen to below 5 p.s.i. (Sparing no expense to ensure absolute realism in this game, I drove my Ford Capri for several days with falling oil pressure, and can certify that 5 p.s.i. is indeed the point at which engines tend to blow up.) Players who beat the previous best are invited to enter their name, and this will subsequently be displayed together with the 'target' score. An option is provided at the end to SAVE the best score on tape as a target for future games.

NOTES

The 3 main sections of the program are:

Initialization/Rules	10–180 and subroutine 5000
Game Loop	190–390 and subroutines 500, 700, 800 and 900.
Ending	1000–4110

Obviously the Game Loop is of prime importance, so I will explain this section in detail.

Lines 190–390 form a 'game handling' loop, which accepts the user input and then delegates all other aspects of the program to its various subroutines. The only exit from the loop is in line 195; until any of the conditions specified in this line are satisfied, the loop will simply continue to accept inputs and delegate accordingly.

INSTRUMENT PANEL—SUBROUTINE 500

The figures on the instrument panel are updated almost every move, so in fact the entire panel is reprinted each time. Line 500 prevents the signboard (see subroutine 800) from being reprinted except after a bend has been negotiated. Line 580 prints a 'thermometer' type speedometer which serves no useful purpose whatever (a digital speedometer is provided as well), but enhances the screen display!

KAMI KAZI DRIVE *(Continued)*

```
230 FOR J=1 TO LEN T$
240 IF T$(J)<"0" OR T$(J)>"9" THEN GOTO 210
245 NEXT J
250 LET P=VAL T$
255 PRINT AT 21,0;E$
260 IF P<S-40 THEN LET B=B-40
265 IF P<S-10 THEN GOSUB 700
270 IF P>S+40 THEN LET P=S+40
280 LET S=P
290 LET Y=Y-1
300 LET O=O-INT (RND*5)
310 IF Y=-1 THEN GOSUB 800
320 IF Y=0 AND S>XX THEN GOSUB 900
330 LET D=D+INT (S/10)
340 IF B<0 THEN LET B=0
350 IF T<0 THEN LET T=0
360 IF O<20 THEN LET X$="OIL"
370 IF B<=25 THEN LET Y$="BRAKES"
380 IF T<=25 THEN LET Z$="TYRES"
390 GOTO 190
500 IF L>0 THEN GOTO 550
510 FOR J=1 TO 7
520 PRINT AT J+2,7;B$(X,J)
530 NEXT J
540 PRINT AT 0,6;XX;"←";S$
550 PRINT AT 11,6;Y;"←AHEAD"
560 LET J=S/10
570 IF J=0 THEN LET J=1
575 IF J>30 THEN LET J=30
580 PRINT AT 14,0;D$(1 TO J);E$(J TO)
590 PRINT AT 15,0;"SPEED";TAB 10;S;"←←";TAB 15;S$
600 PRINT X$;TAB 10;O;"←";TAB 15;"P.S.I."
610 PRINT Y$;TAB 10;B;"←←";TAB 15;P$
620 PRINT Z$;TAB 10;T;"←←";TAB 15;P$
630 PRINT "DISTANCE";TAB 10;D;TAB 15;M$
640 PRINT "TARGET";TAB 10;F;TAB 15;M$;TAB 22;N$
650 LET L=Y
660 RETURN
700 LET Z=INT (RND*2)
710 IF P<S-20 THEN LET Z=INT (RND*3)
720 IF P<S-30 THEN LET Z=2
730 IF Z>0 THEN LET B=B-10
740 IF Z=2 THEN LET T=T-10
750 RETURN
800 LET X=INT (RND*6)+1
810 LET XX=X*20
815 LET X=INT (X/2+.5)
820 LET Y=INT (RND*4)+3
830 RETURN
900 LET T=T-((INT (RND*2)+1)*(S-XX))
910 LET B=B-((RND<.3)*5)
920 LET O=O-(RND<.2)
```

BRAKING PENALTIES—SUBROUTINE 700

Three fixed penalties are defined, and 1 is chosen according to the decrease in speed. Note that the lowest 'penalty' is in fact no penalty at all: A decrease in speed of between 11 and 20 m.p.h. will have a 50% chance of incurring no penalty, 21–30 m.p.h. will have a 33% chance, and greater than 30 m.p.h., no chance. A small decrease in speed will not cause the subroutine to be called at all (line 265), and a large decrease will incur a fixed penalty (line 270) IN ADDITION to the penalty imposed by the subroutine.

SIGN BOARDS–SUBROUTINE 800

This is called once by line 180 (to display the first bend) and thereafter by line 310 whenever a new bend is required. Line 800 selects a random number in the range 1 to 6, which defines the severity of the bend. XX (line 810) is the number converted to an actual speed (20 to 120 m.p.h.), and line 830 then determines which of the 3 signboards (initialized in lines 60–87) will best illustrate the bend's sharpness. Finally, line 820 selects Y, the number of moves before the bend is reached, in the range of 3 to 6.

CORNERING PENALTIES–SUBROUTINE 900

Line 900 ALWAYS imposes a tire penalty when the safe speed is exceeded; the penalty may be either 1 or 2 times the amount of the excess. Lines 910 and 920 occasionally impose brake and oil pressure penalties as well.

KAMI KAZI DRIVE *(Continued)*

```
930 RETURN
1000 DIM A$(2,32)
1005 LET A$(1,13 TO 17)="CRASH"
1010 LET A$(2,13 TO 17)="CRASH"
1015 IF O>4 THEN GOTO 1030
1020 LET A$(1,13 TO 17)="BOOOM"
1025 LET A$(2,13 TO 17)="BOOOM"
1030 FOR J=0 TO 20
1035 PRINT AT 5,0;A$(J/2-INT (J/2)+1)
1040 IF J=5 OR J>11 THEN GOTO 1055
1045 PRINT AT J,0;E$
1055 NEXT J
1060 CLS
2020 IF D>F THEN GOTO 3000
2030 PRINT AT 10,0;"PRESS P TO PLAY AGAIN"
2040 PRINT AT 13,0;"OR X TO STOP"
2050 IF INKEY$="P" THEN GOTO 95
2060 IF INKEY$<>"X" THEN GOTO 2050
2070 GOTO 4000
3000 PRINT D;"←IS THE BEST SCORE SO FAR"
3010 PRINT "ENTER YOUR NAME"
3020 INPUT N$
3030 LET F=D
3040 PRINT AT 5,0;←←←←←←WELL DONE←";N$
3050 GOTO 2030
4000 CLS
4010 PRINT "THE BEST SCORE IS←";F;"←";M$
4020 PRINT "SCORED BY←";N$
4030 IF F<=PR THEN GO TO 9999
4035 LET PR=F
4040 PRINT AT 10,0;"THIS IS BETTER THAN THE PREVIOUS"
4050 PRINT "BEST AS HELD ON YOUR TAPE"
4060 PRINT AT 15,0;"IF YOU WANT TO SAVE THIS SCORE"
4070 PRINT "START TAPE, THEN PRESS S"
4080 PRINT "(OTHERWISE PRESS X)"
4090 IF INKEY$="S" THEN GOTO 9010
4100 IF INKEY$<>"X" THEN GOTO 4090
4110 GOTO 9990
5000 CLS
5010 PRINT TAB 8;"KAMI KAZI DRIVE"
5014 PRINT AT 10,3;"DO YOU WANT INSTRUCTIONS?"
5016 IF INKEY$="N" THEN RETURN
5018 IF INKEY$<>"Y" THEN GOTO 5016
5020 PRINT AT 3,0;"THE OBJECT OF THIS←←GAME←←IS←←TODRIVE AS
      ←FAR AS POSSIBLE←←BEFOREYOUR CAR CRASHES
      ←OR BLOWS UP."
5030 PRINT
5040 PRINT "OBVIOUSLY YOU WILL COVER GREATERDISTANCE
      ←BY DRIVING FAST."
5050 PRINT
5060 PRINT "HOWEVER,←←CORNERING←←AT←←←EXCESSSPEED, OR
      ←BRAKING TOO←←SEVERELY,CAN DAMAGE
      ←YOUR HEALTH."
```

ENDING

Much of the end routine of 'Kami Kazi Drive' is cosmetic. That is, it is not vital to the game, but adds the finishing touches which distinguish 'work in progress' from a completed program. However, the SAVE routine at line 9000 is unusual, and deserves some comment.

Before I explain the various PEEKs and POKEs, let me pose you a common programming problem: Very many programs require a 'Data tape' (i.e., in the case of the Timex 1000, a copy of the program) to be recorded after a run in which data has been altered. Most business programs will require this, and Kami Kazi Drive requires a new tape to be made if the 'best score so far' is to be retained. Importantly, this 'Record Data Tape' option (or command) must be the VERY LAST action taken by the user, and there should be no further chance of amending data once the tape has been made.

A normal 'programmed save' (i.e., the inclusion of SAVE as a program line, followed by RUN, or GOTO . . .) will result in the program re-running when the SAVE is complete. This is just a Timex 1000 quirk, and is usually of no importance; in fact, when using the GOTO 9000 convention adopted throughout this book to SAVE a program in the first instance, it is rather encouraging to see the program reappear! Unfortunately, there is nothing more absurd than for a program to re-run AFTER the user has indicated that he has finished.

The problem then is how to write a programmed save routine which will recognise:

1) Whether the SAVE routine itself has just been called, in which case in will NOT run, or

2) whether the program has just been LOADed, in which case it WILL run, or even

3) whether the SAVE routine is being called for the first time, i.e., from an 'immediate' command, in which case it will also run.

This is not as easy as it might sound. Remember, that a program is LOADed in exactly the same state as it was SAVEd, and that execution will continue immediately from the line following the SAVE instruction. Therefore it is impossible to use any variables as flags to indicate whether the program is to run or end.

The solution used here makes unconventional use of a Timex 1000 system variable (16390 MODE). You will see from the Sinclair manual (p. 177) that the early system variables are NOT saved. Therefore it is possible to use MODE not for its normal purpose (to indicate the kind of cursor to be used), but as a flag! MODE is given the value 4 when the program is SAVEd (line 9010), and of course retains this value at line 9030, causing the program to fall through to its natural end at line 9999. When the tape is next LOADed, MODE will not have this strange value (4 indicates Graphics mode!), and so line 9030 will cause a branch to the desired line 90.

KAMI KAZI DRIVE *(Continued)*

```
5070 PRINT
5080 PRINT "ALSO, YOUR←←CAR←←HAS←←AN←←ENGINEFAULT, AND THE
      ←OIL PRESSURE WILLDROP UNCONTROLLABLY."
5090 PRINT
6000 PRINT "YOU←←MUST←←WORK←←OUT←←←THE←←BESTSTRATEGY
      ←BY PLAYING THE GAME."
6010 PRINT AT 21,3;"PRESS ANY KEY TO CONTINUE"
6020 IF INKEY$="" THEN GOTO 6020
6030 CLS
6040 PRINT "THE INSTRUMENT PANEL←←WILL←←TELLYOU←←YOUR
      ←←CURRENT←←SPEED,←←←OILPRESSURE, THE
      ←EFFICIENCY OF YOURBRAKES AND TYRES
      ←(PERCENTAGE) ASWELL AS THE DISTANCE
      ←COVERED ANDTHE BEST SO FAR."
6050 PRINT
6060 PRINT "A SIGN BOARD WILL ILLUSTRATE THENEXT BEND
      ←AHEAD, AND←←WILL←←SHOWTHE MAXIMUM SPEED
      ←←AT←←WHICH←←ITMAY BE SAFELY NEGOTIATED."
6070 PRINT
6080 PRINT "SIMPLY ENTER THE SPEED WHICH YOUWISH TO
      ←DRIVE AT EACH MOVE."
6090 PRINT
6100 PRINT "MAX. ACCELERATION=40 M.P.H./MOVE"
6110 PRINT
6120 PRINT "(JUST PRESS N/L FOR MAXIMUM,←←ORENTER←←'P'
      ←TO←←MAINTAIN←←CURRENTSPEED.)"
7000 PRINT AT 21,5;"PRESS ANY KEY TO PLAY"
7010 IF INKEY$="" THEN GOTO 7010
7020 RETURN
9000 LET F=0
9005 LET PR=-1
9010 POKE 16390,4
9015 SAVE "KAMI KAZI DRIVE"
9020 CLS
9025 IF F<>PR THEN RUN
9030 IF PEEK 16390=0 THEN GOTO 90
9040 POKE 16390,0
9990 CLS
9999 PRINT AT 11,3;"THANKS FOR PLAYING. BYE."
```

The other half of the problem is much simpler: lines 9000 and 9005 are never called by the program, but are only used by the immediate command GOTO 9000. Therefore a flag device may be used to indicate whether the program should run (after an immediate SAVE) or end (after a programmed SAVE). In this case, F and PR always have the same value at the end of a run, so setting them to different values (any values would do) will cause the branch in line 9025.

STRATEGIC CONSIDERATIONS

As I am the writer of this program, I suppose that for the moment at least, I must be the world's best (i.e., only) player. My own personal best score, achieved by adopting a driving technique not dissimilar to that which I use in real life, is 703 miles, in a game which terminated in a crash at over 400 m.p.h.!

While you may be able to better this score with a strategy of your own, the following points may help you in getting beyond the 'beginner' stage:

There is no point in finishing a game with any tires left if you have run out of brakes. (Or vice versa; or with either if you have no oil pressure). If you do not exhaust all 3 items, you have almost certainly missed some scoring opportunity.

It is possible ONCE (only) to get away with a spectacular braking feat. Reducing speed by 40 m.p.h. or more will cost you 50% of your (original) brakes, and 10% of your tires. Use this to maximum advantage by accelerating into a bend, and braking at the last minute. If you are traveling at less than 300 m.p.h. when you use this ability, you have probably wasted it.

Cornering at 'safe + 30' m.p.h. MAY only cost you 30% of your tires (or it may cost you 60): Try it at a 120 m.p.h. bend, and then accelerate, hoping that the next bend is a long way ahead!

Even when you are driving 'sensibly' (usually in the early part of the game), calculate your braking carefully. Take care not to waste odd points by driving slower than you need to.

Learn to recognize the point where you cannot possibly survive the next bend, and ACCELERATE.

Good luck . . . and drive careLESSly!

Chapter 10
Functional Programs

Clock/Calendar

This program was written one day in February. I had thrown away last year's calendar, and realised that I hadn't yet bought a new one. The easiest way I could think of to find out the date was to write the program! This may not, at first, seem to be a particularly sound economic proposition, as a Timex 1000 with RAM-pack costs rather more than a normal calendar, but if you use your computer exclusively for running this program, it will have paid for itself by the mid 22nd century.

The screen will display any chosen month of any year, together with a menu of 6 options as follows:

N and L for Next and Last month respectively
D to mark a particular Date
C and X to turn the clock function off
B to blank out the menu section for a tidy screen display. (The menu is still operative even when it has been blanked out, and pressing B again will return it to the screen.)

The clock function deserves some special mention. The Timex 1000 does not have a 'real-time clock' as some other computers do, so it is necessary to create one by using the system variable FRAMES at address 16436/7. (The Sinclair manual suggests using the PAUSE statement, but this is only an indirect way of accessing FRAMES.) TIM (line 2) is a unit of time unique to the Timex 1000, being the number of seconds it take for the ms byte of FRAMES (16437) to decrease by 1. The actual decrease (variable D) in value of this byte is detected in subroutine 3500, and variable S is increased accordingly (line 3525). The subroutine is called constantly during the normal display loop (lines 700–860), but when a menu item is selected the clock 'freezes,' and is reset to the correct time as soon as the loop is re-entered.

A word of warning about using address 16437 for timing purposes: The value of this address decreases steadily from 255 to 128 (NOT zero) It is possible to poke some lower value into 16437, and it will continue to decrease as normal, until it reaches zero, at which point the computer will be thrown into confusion, and will take the easy way out

CLOCK/CALENDAR

```
1 LET TY=1982
2 LET TIM=256/50
3 LET TIM=TIM*.992
10 LET M$="*JANUARY*FEBRUARY*MARCH*APRIL*MAY*JUNE*JULY*AUGUST*
     SEPTEMBER*OCTOBER*NOVEMBER*DECEMBER*"
20 LET N$="←1←2←3←4←5←6←7←8←910111213141516171819202122
     232425262728293031"
30 LET D$="SUNMONTUEWEDTHUFRISAT"
40 LET C$="303232332323"
50 DIM B$(32)
60 LET B=0
70 LET C=0
100 PRINT TAB 12;"CALENDAR"
110 PRINT AT 4,0;"DO YOU REQUIRE←";TY;"←?"
120 IF INKEY$="Y" THEN GOTO 200
130 IF INKEY$<>"N" THEN GOTO 120
140 PRINT AT 4,0;"YEAR REQUIRED ?←←←←←←"
150 INPUT T$
160 IF LEN T$<>4 THEN GOTO 150
170 FOR J=1 TO 4
175 IF T$(J)<"0" OR T$(J)>"9" THEN GOTO 150
180 NEXT J
185 LET TY=VAL T$
190 IF TY<1753 THEN GOTO 150
195 PRINT AT 4,14;":←";TY
200 PRINT AT 6,0;"MONTH REQUIRED ?"
205 INPUT T$
210 LET M$=M$+T$+"*"
215 LET L=LEN T$
220 IF L=0 THEN GOTO 205
225 IF T$(1)<"A" THEN GOTO 300
230 LET M=0
235 FOR J=1 TO LEN M$
240 IF M$(J)<>"*" THEN GOTO 255
245 LET M=M+1
250 IF T$=M$(J+1 TO J+L) THEN GOTO 315
255 NEXT J
300 IF L=1 THEN GOTO 310
305 IF T$(2)<"0" OR T$(2)>"2" OR L>2 THEN GOTO 205
310 LET M=VAL T$
315 IF M<1 OR M>12 THEN GOTO 205
320 CLS
325 IF C THEN GOSUB 3500
330 LET Y=0
335 FOR J=1 TO LEN M$
340 IF M$(J)="*" THEN LET Y=Y+1
345 IF Y<>M THEN GOTO 360
350 LET L=J+1
355 LET Y=20
360 IF Y=21 THEN GOTO 370
```

(i.e., crash!). This is why PAUSE statements, if you insist on using them, should be followed with a POKE 16437,255 (mentioned, but not explained, on p. 127 of the Sinclair manual).

PROGRAM NOTES—THE CALENDAR

The 4 main stages of the calendar part of the program are:

Initialization	1–50
Input	100–370
Calculation	380–440
Display	450–570

INITIALIZATION

Four different types of data string are used: The month names (M$) are of varying length, so they are separated by asterisks; the day names (D$) are all 3 letters long, so there is no need for any separating symbol; N$ seems like a waste of time, but putting a list of consecutive numbers into a string allows ease of manipulation, and also ensures that single-digit numbers will be aligned correctly with dual-digit numbers; finally C$ is a CODE string—look up the CODE of each element and you will see that these are the number of days in each month.

TY (This Year) is set to a default value in line 1, which saves 1 input, and B$ is the 'blank line' string once again.

INPUT

The 'year' input is straightforward: Newline will cause the default value of TY to be implemented, otherwise any 4-digit year will be accepted (although line 190 excludes ancient dates prior to the introduction of the present calendar system).

The 'month' input is more complex, as it will accept and verify either a numeric (e.g., 12) or an alpha (e.g., DEC or DECEMBER) entry. The two possible types of entry are verified by separate routines, line 225 detecting numeric entries and branching to line 300 to verify these. (The numeric section merits no discussion as we have seen this many times before.)

An alpha entry is checked by first tagging the entry itself onto the end of M$ (line 210). This ensures that when M$ is searched for a 'match,' one will always be found, evenif the entry was not a valid month. The loop from line 235–255 detects the entry M$ (only if it is preceded, in M$, by an asterisk, so entry 'A' will be matched by 'April', not 'jAnuary'), and sets the month number M accordingly. Note that because a match will always be found, line 250 will always cause

CLOCK/CALENDAR *(Continued)*

```
365 NEXT J
370 LET T$=M$(L TO J-1)
380 LET LY=0
385 IF TY/4<>INT (TY/4) THEN GOTO 400
390 LET LY=1
395 IF TY/100=INT (TY/100) AND TY/400<>INT (TY/400)
      THEN LET LY=0
400 LET N=CODE C$(M)
405 IF M=2 AND LY THEN LET N=N+1
410 LET Y=0
415 FOR J=1 TO M
420 LET Y=Y+CODE C$(J)
425 NEXT J
430 IF LY AND M>1 THEN LET Y=Y+1
435 LET Y=Y+1-N+(TY-1)*365+INT ((TY-1)/4)-INT ((TY-1)/100)
      +INT ((TY-1)/400)
440 LET Y=Y-7*INT (Y/7)
450 PRINT AT 0,0;TY;TAB (31-LEN T$)/2;T$;TAB 28;TY
460 FOR J=1 TO LEN D$ STEP 3
465 PRINT AT J/3*2+2,0;D$(J TO J+2)
470 NEXT J
480 LET P$=N$( TO N*2)
500 FOR J=1 TO Y
505 LET P$="←←"+P$
510 NEXT J
515 LET R$=P$
520 LET DATE=0
530 LET X=5
535 LET Y=1
540 FOR J=1 TO LEN P$ STEP 2
545 LET Y=Y+2
550 IF Y<16 THEN GOTO 565
555 LET Y=3
560 LET X=X+4
565 PRINT AT Y,X;P$(J TO J+1)
570 NEXT J
575 IF B THEN GOTO 700
600 PRINT AT 17,0;"L=LAST MONTH←←:←←N=NEXT MONTH"
610 PRINT AT 19,0;"C=CLOCK (ON)←←:←←V=CLOCK (OFF)"
620 PRINT AT 21,0;"D=SET DATE←←←←:←←B=BLANK TEXT"
700 IF C THEN GOSUB 3500
800 IF INKEY$="B" THEN GOTO 1000
810 IF INKEY$="L" THEN GOTO 2000
820 IF INKEY$="N" THEN GOTO 2100
830 IF INKEY$="C" THEN GOTO 3000
840 IF INKEY$="X" THEN GOSUB 4000
850 IF INKEY$="D" THEN GOTO 5000
860 GOTO 700
1000 IF NOT B THEN GOSUB 1500
1010 LET B=NOT B
1020 IF INKEY$="B" THEN GOTO 1020
1030 GOTO 575
```

an exit from this loop eventually, so no line is required after line 255 to prevent the 'numeric' routine from being entered accidentally. Non-valid entries will result in M having a value greater than 12, and will therefore be rejected in line 315.

Once an input has been accepted, another routine (lines 330–370) extracts the complete month name from M$. Finding the START of the month name is easy; a 'counter' variable, Y, counts the number of asterisks, up to the value of M. To find the END of the name, one more asterisk must be detected, so Y is given the dummy value of 20 (line 355), and the loop is continued until Y equals 21 (line 360).

CALCULATION

Given that the final calendar displayed on screen will always start with Sunday at the top, the object of this section is simply to determine how many 'blank' days there are before day 1 is printed. This is done in 3 steps:

1) Determine whether TY is a leap year: The leap year flag LY is first set to zero (no), then to 1 (yes) if the year is divisible by 4, then back to zero if the year is divisible by 100 but not by 400. (Don't blame me; that's the way it's been done for the past 400 years!)

2) Calculate the number of days since the year dot: lines 405–430 set Y equal to the number of days so far this year (I will refer to TY as 'this year' for simplicity, although it may of course be any chosen year), and line 435 adds the number of days in all previous years, according to the same rules regarding leap years.

3) Reduce the result module 7 to give, amazingly enough, the required answer (line 440).

In the course of all this, N has been set to the number of days in the chosen month (lines 400–405).

DISPLAY

The day numbers to be displayed (P$) are extracted from N$, and then the required number of blanks (calculated above) are tagged onto the start of P$ (lines 500–510). An exact copy of P$ (R$) is taken in line 515, as P$ will be manipulated by the 'Date' option (see below). The calendar is then printed by a standard FOR-TO loop, but with coordinates X and Y being incorporated so that each number is printed at the correct position.

THE OPTIONS

The menu (lines 600–620) adds a certain amount of interest to this program; at least you can do something other than just look at it.

CLOCK/CALENDAR *(Continued)*

```
1500 FOR J=17 TO 21 STEP 2
1510 PRINT AT J,0;B$
1520 NEXT J
1530 RETURN
2000 IF TY=1752 AND M=10 THEN GOTO 600
2010 LET M=M-1
2020 IF M<>0 THEN GOTO 315
2030 LET M=12
2040 LET TY=TY-1
2050 GOTO 320
2100 IF TY=9999 AND M=12 THEN GOTO 600
2110 LET M=M+1
2120 IF M<>13 THEN GOTO 315
2130 LET M=1
2140 LET TY=TY+1
2150 GOTO 320
3000 LET C=1
3005 GOSUB 1500
3010 PRINT AT 19,0;"TIME (HOUR)?"
3015 GOSUB 3100
3020 LET H1=H
3025 PRINT AT 21,0;"TIME (MINS)?"
3030 GOSUB 3100
3035 GOSUB 1500
3040 GOSUB 3600
3045 GOTO 575
3100 INPUT H$
3105 POKE 16437,255
3110 LET S=0
3115 LET P=255
3120 IF H$="" OR LEN H$>2 THEN GOTO 3100
3125 FOR J=1 TO LEN H$
3130 IF H$(J)<"0" OR H$(J)>"9" THEN GOTO 3100
3135 NEXT J
3140 LET H=VAL H$
3145 RETURN
3500 LET P1=P
3505 LET P=PEEK 16437
3510 LET D=P1-P
3515 IF D<0 THEN LET D=D+128
3520 IF NOT D THEN GOTO 3900
3525 LET S=TIM*D+S
3530 IF S<60-(TIM/2) THEN GOTO 3900
3550 LET S=S-60
3555 LET H=H+1
3560 IF H<60 THEN GOTO 3600
3565 LET H=0
3570 LET H1=H1+1
3575 IF H1=24 THEN LET H1=0
3600 LET H$=":"
3610 IF H<10 THEN LET H$=":0"
3620 LET H$=STR$ H1+H$+STR$ H
```

Options 'C' and 'L' use very similar routines (2000 and 2100) to recalculate M and reform the display for a different month.

Option 'D' uses the routine at line 5000 to reformat P$, inserting inverse characters to mark the chosen date. Notice that lines 5045–5060 allow you to 'turn off' the date marker by entering the date which is already shown in inverse. (Variable DATE, which normally takes the value of the marked date, is given the default value of zero in line 520).

Option 'B' uses the simplest routine ever (line 1000) to print blank lines over the menu, and reverse the flag B, so that when B is pressed again, the menu will be reprinted. (This happens so quickly that line 1020 is necessary to ensure that key 'B' has been released, otherwise the menu would flash on and off repeatedly.)

Option 'C' turns on the clock, as described below, and 'X' simply reverses flag C, which indicates whether the clock is to be displayed.

THE CLOCK

The principle of the clock function has already been explained. I should draw attention to line 3, however, which is a small adjustment factor to TIM. Remember that FRAMES was not intended as a real-time clock, and you will probably find that any program which uses it as such will require some adjustment of this nature to give complete accuracy.

Selection of option 'C' calls 2 subroutines: subroutine 3100 handles the initial time inputs, and subroutine 3600 converts the current time into a 5-character inverse display. Option 'C' also sets flag C, so that the main 'time update' subroutine 3500 is called at regular intervals.

Subroutine 3500 detects any change in the value of PEEK 16437, and, if there is a change, increases S, the seconds counter accordingly. The seconds themselves are not displayed, as they are not updated EVERY second, but every 5 or 6 seconds (or longer, if the clock is frozen). Line 3530 however increases the minutes figure when S reaches a value within about 2.5 seconds of the exact minute, rather than waiting for S to exceed 60, thereby doubling the accuracy of the displayed clock.

NOTE

Although the Gregorian calendar, on which this program is based, was introduced in 1582, it was not adopted in Britain until 1752. You may make the following amendments to allow the display of earlier 'non-British' dates:-

```
190 IF TY  1583 THEN GOTO 150
2000 IF TY = 1582 AND M = 11 THEN GOTO 600
```

CLOCK/CALENDAR *(Continued)*

```
3630 IF LEN H$=4 THEN LET H$="0"+H$
3640 FOR J=1 TO 5
3650 LET H$(J)=CHR$ (CODE H$(J)+128)
3660 NEXT J
3900 PRINT AT 1,13;H$
3910 RETURN
4000 LET C=0
4010 PRINT AT 1,13;B$( TO 5)
4020 RETURN
5000 GOSUB 1500
5005 PRINT AT 19,0;"DATE ?"
5010 INPUT Q$
5015 IF Q$="" THEN GOTO 5010
5020 FOR J=1 TO LEN Q$
5025 IF Q$(J)<"0" OR Q$(J)>"9" THEN GOTO 5010
5030 NEXT J
5035 LET Q=VAL Q$
5040 IF Q>N OR Q<1 THEN GOTO 5010
5045 IF Q<>DATE THEN GOTO 5065
5050 LET P$=R$
5055 GOSUB 1500
5060 GOTO 520
5065 LET DATE=Q
5070 IF LEN Q$=1 THEN LET Q$="←"+Q$
5075 FOR J=2*DATE-1 TO LEN P$+1
5080 IF P$(J)<>Q$(1) OR P$(J+1)<>Q$(2) THEN NEXT J
5085 LET P$=R$( TO J-1)+CHR$ (CODE Q$(1)+128)+CHR$
      (CODE Q$(2)+128)+R$(J+2 TO )
5090 GOSUB 1500
5095 GOTO 530
9000 SAVE "CLOCK/CALENDAR"
9010 RUN
```

Scroll

Yes, I know that the Timex 1000 has a 'SCROLL' function already, and very useful it is too; not many other computers feature a programmable scroll command. But wouldn't it be nice if the screen could be scrolled horizontally as well as vertically? This little machine code routine does exactly that, and can be used to enhance your screen displays by pushing any unwanted information off screen to the left.

The Basic program shows the routine in action. Note that the routine itself works in the same way as the normal SCROLL command: it scrolls 1 line (or in this case, 1 column), at a time, and can therefore be included in a FOR-TO loop to clear the entire screen.

NOTES—MACHINE CODE ROUTINE

The routine works on each screen line individually, by taking each character in turn (ignoring the first character, hence HL is increased TWICE from the D-FILE value), replacing it with a space, 'back-spacing' 1 position (DEC HL), and replacing the character in this new position. This process is repeated a further 30 times until each character has been shifted 1 position to the left. The entire loop is then repeated for the second screen line, and so on.

As printed, the routine will scroll all 22 screen lines. However, it is possible to specify the number of lines to be scrolled by altering the 18th byte of the code. The A register is used as a counter of the number of 'scroll loops' executed, and when this reaches a limit value (CP 22), the routine is terminated (RET Z). To scroll just the top half of the screen, alter the 18th byte to hex 0B (decimal 11), so that the routine terminates after 11 scroll loops. The simplest way to do this is to POKE the required (decimal) value into the appropriate position in the code, as illustrated in the Basic program, where all the possible values (1 to 22) are poked in turn into address 16531 (i.e., the address of the 18th byte, assuming that the routine begins at address 16514).

NOTES—BASIC DEMONSTRATION

The basic program is totally silly, and I certainly do not intend to discuss it at any great length. Apart from demonstrating the machine code scroll, and the effect of POKEing 16531 (as mentioned above), it does, however, serve as an example of the simplest possible form of Machine Code Loading: the bytes of machine code are typed DIRECTLY from the keyboard into a Basic REM statement.

You can check that the Basic 'characters' in line 1 do in fact correspond to the bytes of machine code by referring to pages 181–187 of the Sinclair manual. Byte 1, AF, corresponds to the character 'Inverse J,' so why not type an Inverse J into the REM statement, rather than POKEing it there?

SCROLL

```
AF                      XOR A           CLEAR A REG.
2A 0C 40                LD HL,(16396)   D-FILE
06 1F           COUNT   LD B,31         COLUMN COUNTER
23              LOOP    INC HL
23                      INC HL
56                      LD D,(HL)       OLD CHARACTER
36 00                   LD (HL),0       PRINT 'SPACE'
2B                      DEC HL
72                      LD (HL),D       PRINT OLD CHR
10 F7                   DJNZ 247        LOOP
3C                      INC A           LINE COUNTER
FE 16                   CP 22           IS IT 22 *
C8                      RET Z           IF YES, RETURN
23                      INC HL
23                      INC HL          NEXT LINE
18 ED                   JR 237          COUNT
```

* Or whatever, see notes.

BASIC DEMONSTRATION

```
1 REM JE£RND↑T↑377?Q←F?( RUN W RETURN -COS 77/ GOSUB
2 POKE 16522,86
3 POKE 16526,114
100 PRINT "HORIZONTAL SCROLL DEMONSTRATION"
110 FOR J=1 TO 21
120 PRINT AT J,31;"*"
130 NEXT J
140 FOR J=1 TO 55
150 IF J<=22 THEN POKE 16531,J
160 RAND USR 16514
170 IF J<>23 THEN GOTO 200
180 PRINT AT 20,29,"↑W60↑"
190 PRINT TAB 29,"0←0"
200 NEXT J
```

There are 2 slight problems with this technique:

1) Where there is no character corresponding to a particular machine code instruction (e.g., LD D, (HL) - Hex 56 and LD (HL), D - Hex 72), it is obviously not possible to type a character. Therefore you should type ANY character (I have used a question mark) and then POKE the required DECIMAL value into place (see lines 2 and 3).

2) A certain amount of 'keyboard manipulation' is required in order to type some characters (particularly keywords). In this example, the Inverse J and the graphics character should be typed by changing the cursor to a 'G,' using the Graphics key, and RND and COS by changing the cursor to an 'F' (Function key).

RUN, RETURN and GOSUB, however, are keywords, and the Timex 1000's automatic syntax check does not permit the cursor to be changed manually to a 'K' . . . or does it? In fact it is very easy to change the cursor to a 'K': just enter 'THEN' (shift 3), and there it is! Now enter the required keyword and then delete THEN using the normal cursor controls and the delete key.

IMPORTANT: The only space to be typed is shown in the listing by the left arrow symbol (←); all the other APPARENT spaces will appear automatically as you enter the keywords.

Lines 160 calls the machine code routine; the word RAND is used merely to make the line syntactically correct. (Remember that USR is a Function, not a Keyword.) This use of RAND is a common 'shorthand' device, but it should not be used indiscriminately, particularly if a program relies elsewhere on the RND function producing a truely random number. If in doubt, use the 'correct' syntax of a USR statement: 'LET V = USR n' (where n is the required machine code address, and V is any Basic variable, possibly a dummy).

Incidentally, if you are wondering what a routine as short as this is doing in a book of 16K programs, the answer is that it does NOT detect the end of each screen line by testing for the 118 character as in the routine in Battleships, but assumes that ALL screen lines contain EXACTLY 32 characters. This is always the case with the 16K Timex 1000, but is NOT normally true when the RAMpack is disconnected.

Coin Analysis

If you have never worked with payrolls, then the function of this program may not be immediately obvious. Clearly a wages department needs to know the total amount payable to employees in order that the correct amount of money may be drawn from the bank. However, they ALSO need to know the exact breakdown of this total into nickels, dimes, etc., so that each wage packet may be made up correctly. (Simple example: If 2 people are to be paid $5 each, there is no point in

COIN ANALYSIS

```
10 FAST
20 DIM A$(9,6)
30 DIM B(9)
40 DIM C(9)
50 LET A$(1)="TWENTY"
60 LET A$(2)="TEN"
70 LET A$(3)="FIVE"
80 LET A$(4)="ONE"
90 LET A$(5)="50 P"
100 LET A$(6)="10 P"
110 LET A$(7)="5 P"
120 LET A$(8)="2 P"
130 LET A$(9)="1 P"
140 LET B(1)=20
150 LET B(2)=10
160 LET B(3)=5
170 LET B(4)=1
180 LET B(5)=.5
190 LET B(6)=.1
200 LET B(7)=.05
210 LET B(8)=.02
220 LET B(9)=.01
230 LET T=0
240 LET N=1
250 CLS
260 PRINT AT 0,0;"COIN ANALYSIS";AT 2,0;"ENTER NET AMOUNT
    ←PAYSLIP NO.";N;AT 4,0;"( OR 'X' TO FINISH )"
270 INPUT P$
280 IF P$="" THEN GOTO 270
290 IF P$="X" THEN GOTO 1000
300 LET X=0
310 FOR J=1 TO LEN P$
320 IF P$(J)<>"." THEN GOTO 360
330 LET X=X+1
340 IF X>1 OR LEN P$(J TO )>3 THEN GOTO 270
350 GOTO 370
360 IF P$(J)<"0" OR P$(J)>"9" THEN GOTO 270
370 NEXT J
380 LET P=VAL P$
390 LET N=N+1
400 LET T=T+P
410 FOR J=1 TO 9
420 LET P=P+1E-5
430 LET X=INT (P/B(J))
440 LET P=P-X*B(J)
450 LET C(J)=C(J)+X
460 IF ABS P<1E-4 THEN LET J=9
470 NEXT J
```

drawing a $10 bill from the bank.) This process is traditionally known as COIN analysis; I don't know why, as it obviously involves dollars as well as coins.

The program handles the following denominations of U.S. currency: $50, $20, $10, $5, $1, 25¢, 10¢, 5¢, 1¢. (You may, or course, extend the range of the program to include higher denominations, or less-frequently used denominations, such as $2 or 50¢.)

NOTES

There is a world of difference between a program which is crash-proof, idiot-proof, and a pleasure to use, and one which merely 'works.' Most of this program is concerned with such 'mundane' matters as input verification, screen formatting, 'user friendliness,' etc. The actual calculation is contained entirely in the loop from line 410 to 470!

THE CALCULATION

The principle of coin analysis is to divide each individual net pay figure by the highest denomination of 'coin,' divide the remainder by the next highest, and so on until there is no remainder: a repetitive process of simple mathematics, and therefore an ideal task for a computer. Unfortunately, the one thing computers are not particularly good at is 'simple' mathematics, and this leads to a slight complication, as we shall see.

The 3 arrays used are A$ (coin names), B (coin values) and C (required quantity of each coin). In theory, all that the program need do is divide the net pay (P) by $50 (B(1)), store the result in C(1), calculate the remainder, and continue for each element of the array. In practice, however, both subtraction and division are likely to cause errors due to the way in which the computer stores numbers in 'floating point' binary format.

The Timex 1000 stores numbers accurately to 9 digits only, so any calculation is likely to produce a result which is inaccurate to the final decimal place. Of course, the level of inaccuracy is very small, and it would not be significant in this program, EXCEPT for the presence of the INT function in line 430. The use of INT will magnify 'insignificant' errors tremendously. For example, 4.9999999 is a very close approximation to 5, but INT 4.9999999 is still 4! The way around this problem is to DELIBERATELY introduce an inaccuracy into the calculation which is larger than any inaccuracy the computer may introduce, but smaller than the level of accuracy required by the program (which of course is to 2 decimal places of a dollar. Hence in line 420, the addition of 1E–5 (.00001) will ensure that P is always fractionally higher than the

COIN ANALYSIS *(Continued)*

```
480 PRINT AT 6,11;"←←←←←←←←←←";AT 6,0;"LAST ENTRY:";P$
490 GOTO 260
500 IF VAL P$<>0 THEN GOTO 530
510 LET P$="←←←←←←0.00"
520 GOTO 590
530 IF P$(10)<>"←" THEN GOTO 560
540 LET P$="←"+P$( TO 9)
550 GOTO 530
560 IF P$(9)="." THEN LET P$=P$(2 TO )+"0"
570 IF P$(8)<>"." THEN LET P$=P$(4 TO )+".00"
580 IF VAL P$<1 THEN LET P$(7)="0"
590 PRINT P$
600 RETURN
1000 CLS
1010 FOR J=1 TO 9
1020 DIM P$(10)
1030 PRINT TAB (5-LEN STR$ C(J));C(J);"←X←";A$(J);"←=";
1040 LET P$=STR$ (C(J)*B(J))
1050 GOSUB 500
1060 NEXT J
1070 PRINT AT 11,7;"TOTAL←←←=";
1080 LET P$=STR$ T
1090 GOSUB 500
1100 PRINT AT 13,0;"NO. OF PAYSLIPS=";TAB 26-LEN STR$ (N-1);N-1
1110 CLEAR
1120 SLOW
1130 PRINT AT 20,0;"C= COPY : F= FINISH : R= RE-RUN"
1140 IF INKEY$="R" THEN RUN
1150 IF INKEY$="F" THEN GOTO 9999
1160 IF INKEY$<>"C" THEN GOTO 1140
1170 PRINT AT 20,0;"←←←←←←←←←←←←←←←←←←←←←←←←←←←←←←"
1180 COPY
1190 GOTO 1130
9000 SAVE "COIN ANALYSIS"
9010 RUN
9999 STOP
```

'true' value. Small positive inaccuracies introduced by line 430 would not make any difference anyway, but small negative inaccuracies will be nullified.

The inevitable outcome of the computer introducing errors on one hand, and the program introducing errors on the other, is that P is never likely to be reduced to exactly zero. Remember that P is always the remainder left after the previous division, and that the calculation is complete when P = 0. This is not terribly important, as there are only 9 divisions to be performed anyway (for the 9 elements of array B), and any small residual value of P can be disregarded. However, the test in line 460 saves time by determining whether P has reached a value less than 1E-4 (i.e., VIRTUALLY zero, allowing for all accumulated errors so far).

INPUT VERIFICATION

All the programs in this book have featured some sort of input check. This one is only slightly more elaborate than the standard routine, as it ensures that only valid cash amounts are accepted. Lines 270, 280, 310, 360 and 370 should be very familiar to you by now; they check that the input is entirely numeric. Line 320 detects the presence of a decimal point, and line 340 rejects entries where the point is followed by more than 2 figures. Variable X (lines 300 and 330) COUNTS the number of decimal points, to ensure that the entry contains no more than 1.

SCREEN FORMATTING

This is actually the longest part of the program, and occupies line 1000 onwards, and also subroutine 500. The final 'table' of results is laid out in 3 columns, the central one being predefined as the names of the various 'coins' (held in array A$). Column 1 (quantity of each coin: C(J)) and column 3 (value: C(J)*B(J)) may both contain figures of differing lengths, and so must be aligned correctly. Column 1 is a simple tabulation (see line 1030), but aligning cash figures is a little more complex.

The method I have used here is to define a standard string format for all cash figures, and then to convert each figure to this format in turn for printing. I have also exploited a useful Timex 1000 feature: re-

dimensioning an array simply erases the old array from memory. Hence line 1020 sets up a format string of 10 characters for each cash figure. The ultimate format will have a decimal point in the eighth element of P$, cents in elements 9 and 10, and up to 7 elements for dollars. (This SHOULD be sufficient; somehow I cannot envisage companies with a wage-bill of over 10 million dollars using a Timex 1000 for their payroll!

All this is handled by subroutine 500. Lines 530–550 'left fill' the array with blanks, shifting the actual cash value to the right. Lines 560–570 then 'right-fill' with zeros if necessary to align the decimal points. Lines 510–520 are not strictly necessary, but avoid all this string manipulation just to print a zero value. Line 580 is required because of a Timex 1000 quirk: it will print .1 as '0.1,' but .01 as '.01' (without a zero before the point.

User Friendliness

This is a horrible expression which usually means the addition of various messages and instructions which most users will find thoroughly obnoxious. True user friendliness, however, involves ease of operation of a program, anticipation of every problem which the user is likely to encounter and the provision of appropriate assistance when these problems occur.

In this program, the instructions, such as they are, are kept brief and to the point. The current payslip number is displayed on the screen, together with the 'Or X to finish' message. (A lot of programs tell you INITIALLY how to finish, but fail to remind you of this when you acutally need it.)

The most likely problem the user is likely to experience is that after entering several payslips, some distraction will cause him to lose his place, so to counteract this, line 480 will display the last amount entered.

The 'end options' at line 1130 are all trivial: the program could simply STOP, and the user could RUN or COPY as desired, but it is no problem to include these options in the program itself. Just because YOU know how to make a copy of the screen, it is wrong to assume that everybody who will ever use one of your programs will know how to do so. The ideal computer program should require the user to know NOTHING about computers!

Datafile

A data file is quite simply a file for storing data! (You can't say you haven't learned anything from this book, when it contains gems like that.) Address books, note books, diaries, etc. can all be stored on your Timex 1000, and numerous DEDICATED programs have been written to provide precisely the right kind of data file for a particular application. This program attempts the impossible in that it is intended as a GENERAL data file, suitable for any purpose whatever.

In deciding what features to include in a general purpose data file, and more importantly, what to leave out, I have adopted the following criteria:

1) The maximum possible amount of memory space should be reserved for data, and

2) both records and fields (see below) should be as flexible as possible in quantity and length.

The principle exclusions are:

1) There are no 'search' or 'sort' facilities, and

2) it is not possible either to amend or delete a record once it has been entered.

It is quite possible that you will not agree with these priorities, and feel that more 'file handling' features should have been included, at the expense of total file size. But this is the last program in the book, and your job now is to go back over all the programs, changing anything you don't like, adding any features you think would be desirable and generally adapting everything to your own taste.

NOTE

You have probably seen this in every computer book ever written, but it is impossible to talk about data files without first defining these three terms:

A FILE is a complete collection of data which is sub-divided into RECORDs, which all contain data on a related subject, and which themselves are sub-divided into FIELDs, which are the individual items of data.

RUNNING INSTRUCTIONS

The basic premise of this program was that the 16K Timex 1000 should hold at least 12K of data, so everything else has taken second place to this (with user friendliness coming a poor third!). There is a menu which lists the 9 options available, but a few notes on what to do from there would not be amiss:

DATAFILE

```
5 DIM B$(32)
10 LET N=0
20 LET P=N
30 LET R=12500
40 DIM A$(R)
50 CLS
60 PRINT "FILENAME ?"
70 INPUT F$
80 LET F$=F$+"."
90 CLS
100 PRINT TAB (30-LEN F$)/2;F$( TO LEN F$-1);AT 7,0;
      "1: DIRECTORY";AT 9,0;"2: ADD RECORD";AT 11,0;
      "3: INSPECT RECORD";AT 13,0;"4: SAVE "
110 LET I$=INKEY$
120 IF I$<"1" OR I$>"4" THEN GOTO 110
130 CLS
140 GOTO VAL I$*1000
1000 IF NOT N THEN GOTO 100
1005 LET Q=1
1010 FOR J=1 TO N
1015 GOSUB 4500
1020 PRINT J;TAB 3;
1400 GOSUB 5000
1430 PRINT A$(Q+2 TO Q+K)
1450 IF J/20=INT (J/20) THEN GOSUB 1500
1455 IF I$="I" THEN GOTO 3000
1460 NEXT J
1470 GOSUB 1500
1475 IF I$="I" THEN GOTO 3000
1480 GOTO 90
1500 PRINT AT 21,0;"C=CONTINUE : I=INSPECT"
1510 LET I$=INKEY$
1520 IF I$<>"C" AND I$<>"I" THEN GOTO 1510
1530 CLS
1540 RETURN
2000 LET N=N+1
2010 PRINT "FIELD NUMBER",N
2015 PRINT "FIELD 1=KEY : MAX 20 FIELDS"
2040 LET Q=P+1
2050 LET P=P+2
2110 FOR J=1 TO 20
2115 PRINT AT J+1,0;B$
2120 PRINT AT J+1,0;J
2130 INPUT T$
2135 IF T$="" THEN GOTO 2130
2140 IF J<3 OR T$<>"X" THEN GOTO 2200
2150 LET J=J-1
2160 PRINT AT J+1,0;B$
2165 PRINT B$
2170 LET P=P-T-1
2190 GOTO 2115
2200 LET T=LEN T$
2205 IF T<28 THEN GOTO 2300
```

1: DIRECTORY

Like the 'Contents' page of a book, this lists each of the records in the file by their number and KEY FIELD. The first field of each record is assumed to be the title or identifying field or, to continue the book analogy, the 'chapter heading,' and this is referred to as the key field. Obviously when you first run the program, there will be no records in the file, so option 1 will have no effect. However, once records have been entered, the directory becomes of great importance, particularly in the absence of any 'search' facilities.

The records are listed in batches of 20. At the end of each 'page' of the directory, there is the option to Continue or Inspect, Key 'C' will continue to the next page. if there is one, or otherwise return to the menu. Key 'I' will immediately jump to option 3 (see below).

2: ADD RECORD

The screen display will inform you of the record number (the computer will allocate the next unused number), and will also present the message: "Field 1 = Key : Max 20 Fields." This is a reminder that field 1 should contain some identifying feature, as it is this field which will be used for the directory entry. "Max 20 fields" means that you may enter any number of fields in a record, up to a maximum of 20.

What the display doesn't tell you is that:

1) each field may contain any number of characters up to a maximum of 27,

2) you should terminate this record entry (if you use less than 20 fields) by entering "F," and

3) you may delete a previous field entry by entering "X" (although you are NOT allowed to delete the KEY FIELD entry.)

(The maximums specified here are to allow each record to be displayed as a single-screen page.)

3: INSPECT RECORD

This requests the number of the record you wish to inspect. Of course, you have forgotten the number, so should press "0" to return to the menu and then choose option 1 to refer to the directory.

Having entered a record number, the entire record will be displayed. Any key will then return you to the menu.

4: SAVE

Selecting option 4 will immediately start the SAVE procedure, so make sure that the tape recorder is running first! Saving 12K of data takes a LONG TIME, so don't try to fit this onto 1 side of a C2 cassette.

DATAFILE *(Continued)*

```
2210 PRINT AT J+1,0;"FIELD LENGTH EXCEEDED"
2220 FOR K=1 TO 50
2230 NEXT K
2250 GOTO 2115
2300 IF T$="F" THEN GOTO 2500
2305 IF P+T+1>=R THEN GOTO 6000
2310 LET A$(P+1 TO P+T+1)=T$+"*"
2320 LET P=P+T+1
2325 PRINT AT J+1,3;T$
2330 NEXT J
2500 LET H=INT ((P-Q)/256)
2510 LET L=(P-Q)-256*H
2520 LET A$(Q TO Q+1)=CHR$ H+CHR$ L
2530 GOTO 90
3000 IF NOT N THEN GOTO 100
3005 PRINT AT 0,0;"RECORD NUMBER ?"
3010 INPUT K
3020 IF K>N THEN GOTO 3010
3025 IF K=0 THEN GOTO 90
3030 PRINT AT 0,14;K
3035 LET Q=1
3040 FOR J=1 TO K
3050 GOSUB 4500
3060 NEXT J
3070 LET L=Q+1+CODE A$(Q+1)+256*CODE A$(Q)
3100 FOR J=1 TO 20
3110 PRINT J;TAB 3;
3120 GOSUB 5000
3122 IF Q+K>=L THEN GOTO 3140
3123 PRINT A$(Q+2*(J=1) TO Q+K)
3125 LET Q=Q+K+2
3130 NEXT J
3140 IF J<=20 THEN PRINT AT J,0;B$
3145 PRINT AT 21,0;"PRESS ANY KEY FOR MENU"
3150 IF INKEY$="" THEN GOTO 3150
3160 GOTO 90
4000 SAVE F$
4010 GOTO 90
4500 IF J=1 THEN RETURN
4510 LET Q=Q+1+CODE A$(Q+1)+256*CODE A$(Q)
4520 RETURN
5000 FOR K=1 TO 28
5010 IF A$(Q+K)="*" THEN GOTO 5025
5020 NEXT K
5025 LET K=K-1
5035 RETURN
6000 PRINT AT 21,0;"FILE FULL:RECORD IGNORED"
6010 FOR K=1 TO 50
6020 NEXT K
6030 LET N=N-1
6050 GOTO 90
9000 CLEAR
9005 SAVE "DATAFILE"
9010 RUN
```

The data is saved under the name you entered in response to the very first question "Filename ?", FOLLOWED BY A FULL STOP. (The reason for this is that the last character of F$ is inverted by the Timex 1000's SAVE routine, preventing it being used later as a heading.) Subsequent LOADing of this file will result in the menu being displayed and records can be added or inspected as before.

NOTES

The heart of the program is the 'file,' contained in the 12500 element A$. This is a SINGLE-dimension array, so any record and field delineators must be embedded into the records and fields themselves. The program illustrates 2 methods for doing this: the end of each field is marked by the inverse asterisk character (line 2310), and the LENGTH of each entire record is stored as a 2-byte hex number, in the first 2 bytes of the particular record.

The reason for this second method is that, whereas all fields in a record are stored consecutively in the array and displayed consecutively by option 3, the key fields, as displayed by option 1, are not stored consecutively. Therefore, by using the first 2 bytes of a record as a POINTER to the next record, the computer is able to locate the next key field without having to search through every element of the array.

In order to make this pointer technique easier to understand, at least to those of you who are familiar with hex numbers in any other context, I have used variables H and L to represent the High and Low bytes of the pointer (see lines 2500–2520. Note that although theoretically this system will allow records of any length up to 64K characters (if the file itself were this large!), the arbitrary restrictions of 20 fields per record and 27 characters per field, which I have imposed merely for screen display purposes, means that in practice no record will exceed 540 characters in length. Therefore the High byte of this number is barely used; you might like to impose even tighter restrictions on the number of fields per record so that the 'length of record' pointer may be held as a single byte.

The value of this pointer can only be calculated AFTER a record has been completed. Therefore 2 other pointers, P and Q, are used; Q to mark the position (in A$) of the start of the current record (i.e., the position in which the H,L pointer is eventually going to be placed), and P which always points to the position of the current character in the array.

If you have any difficulty visualizing how this all works, set up a small 'test' data file using option 2, then press BREAK to stop the program, and then type PRINT A$. This will fill the screen with the first 704 characters of the array (it will, of course, give error code 5/0, as A$ is 12,500 characters long!), and you will be able to see exactly how the data has been stored. The first character will probably be a blank, if you have used only short records. Look up the CODE of the second character, and count that number of characters along the string (including the inverse asterisks). You will find that you have reached the position of the FIRST character (probably a blank again) of record number 2! Once again, look up the CODE of the next character, count that number of characters, and you will reach record number 3.

You can also use this test technique to examine what happens when you delete 1 or more fields of data (by entering X during option 2). Set up a test file of only 1 record, but as you are entering the fields of that record, enter a very long (27-character) field, enter X in response to the next field prompt, then enter F, which will terminate data entry and return to the menu. Examine A$ as before: The 'deleted' entry will still be there, but the POINTER will still point to the end of the 'true' data. In other words, although this 'garbage' has been entered into the array, it is ignored by the pointer, and any further data entries will overwrite it as if it were blank.

In the program, subroutine 4500 'counts' through A$ in much the same way as you have just done manually. This subroutine is called by option 1 (line 1015) to find EVERY key field, and by option 3 (line 3050) to find a specified key field. Q is used as a 'floating' pointer, and is reset to 1 before each time the subroutine is called. P is a 'fixed' pointer; that is, it is not affected by any count through the array. The only time P is adjusted is when new records are added (option 2, line 2000).

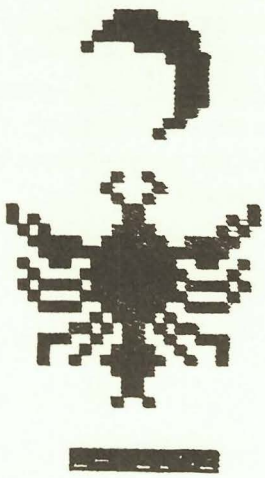
Subroutine 5000 performs the simpler 'field length' count, and is again called by both options 1 and 3.

APPENDIX 1: TAROT CARD DISPLAYS

Here, to assist you in deciphering the hieroglyphics on pages 76–86, and to give you an idea of the intended output of the 'Tarot' program, are some of the screen displays, copied to the printer.

II

XVIII



V

IV

APPENDIX 2: COMPUTERIZED DATING QUESTIONNAIRE

NAME .....

SEX (M/F)

Pick the word/phrase from each of the following pairs which most accurately describes yourself.

SHY/EXTROVERT
FUN LOVING/HUMOURLESS
INTELLECTUAL/THICK AS TWO SHORT PLANKS
FASHION CONSCIOUS/DRESS LIKE A TRAMP
CAREFREE/CAUTIOUS
SOCIABLE/LONER

Mark 3 of the following with a check if they are your interests, AND 3 with a cross if you are definitely NOT interested.

A POP MUSIC
B CLASSICAL MUSIC
C WATCHING SPORT
D PLAYING SPORT
E CINEMA
F THEATRE
G DISCO
H FOOD (RESTAURANT)
I FOOD (COOKING)
J COMPUTERS
K POLITICS (LEFTISH)
L POLITICS (RIGHTISH)
M POLITICS (GENERAL)
N TRAVEL
O ARTS/CRAFTS
P POETRY
Q ANIMALS
R READING

For official use only

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

NOW AVAILABLE

All the programs and routines listed in this book will help you get more enjoyment from your Timex 1000 than ever. Share the fun with your friends and family, and build on the programs to suit your own needs.

Save time and don't risk introducing keyboarding errors into your programs. Order the cassette for only $19.95 *today*!

The TIMEX SINCLAIR 1000 CASSETTE is available at your favorite computer store. Or use the handy order card below.

THE TIMEX SINCLAIR 1000 CASSETTE

Yes, I want to make my TS 1000 better than ever. Please send me _____ copies of Valentine's TIMEX SINCLAIR CASSETTE at $19.95 each.

1-88729-3 $19.95

__ Payment enclosed (including state sales tax). __ Bill me.
Wiley pays shipping and handling charges. __ Bill my company.

__ Charge to my credit card: __ Visa __ Master Card

Card Number ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Expiration date _____ Signature _____

Name _____ Title _____

Company _____

Address _____

City _____ State _____ Zip Code _____

1-88729-3 263 Signature _____ (Order invalid unless signed)

We normally ship within ten days. If payment accompanies your order and shipment cannot be made within 90 days, payment will be refunded.

TURN YOUR IMAGINATION LOOSE WITH TIMEX SINCLAIR PROGRAMS TODAY

Buy the cassette at your favorite computer store, or order from Wiley:

In the United States:	John Wiley & Sons 1 Wiley Drive Somerset, NJ 08873
In the United Kingdom and Europe:	John Wiley & Sons, Ltd. Baffins Lane, Chichester Sussex PO 19 1UD UNITED KINGDOM
In Canada:	John Wiley & Sons Canada, Ltd. 22 Worcester Road Rexdale, Ontario M9W 1L1 CANADA
In Australia:	Jacaranda Wiley, Ltd. GPO Box 859 Brisbane, Queensland AUSTRALIA

Valentine—TIMEX SINCLAIR 1000 CASSETTE 1-88729-3

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 2277, NEW YORK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

JOHN WILEY & SONS, Inc.
1 Wiley Drive
Somerset, N.J. 08873

Attn: Timex Sinclair 1000 Cassette

COMPUTERS

$9.95

What can I do with my Timex 1000, Timex 1500, or ZX81?

The answer may surprise you.

Don't let its price fool you. The Timex Sinclair may be inexpensive, but it packs a lot of computing punch—when it's programmed right.

Programming your Timex Sinclair to do useful work—or to have fun—is easy with this practical, step-by-step guide. It contains 56 complete programs ready to run on your TS 1000, TS 1500, or ZX81—*including 35 programs that run on the 1K or 2K versions!* Programs that are fun, including sophisticated games, fortune telling, and gambling. And programs that are practical—file programs, payroll deductions, graphics, a "checkbook," and many, many others. Each is fully documented, tested, and ready to use.

What if I want to learn to program myself?

Then this book is for you, too. Because these programs are designed to *teach* as well as run. The documentation explains the "how's" and "why's" every step of the way, so you can see and understand the structure and techniques used in each program as you key it in.

And if I don't have time to keypunch my Timex?

No problem. All 56 programs are available on a convenient cassette—ready to run and error-free. Buy it at your local computer store, or use the order card inside.

Roger Valentine is co-founder of V & H Computer Services, a software development firm, and is the author of *What Can I Do With 1K, What Can I Do With 16K, Spectrum Spectacular,* and *Dragon Extravaganza*. His articles and programs have appeared in *Practical Computing, Personal Computing Today,* and other popular computer journals.

Wiley Press paperbacks have taught more than two million people to program, use, and enjoy microcomputers. Look for them all at your favorite bookshop or computer store! For a complete list, write to:

WILEY PRESS
a division of John Wiley & Sons, Inc.
605 Third Avenue, New York, N.Y. 10158
New York • Chichester • Brisbane • Toronto • Singapore

Cover design: Larry Smith
Cover photograph: Arnold Katz

ISBN 0 471-88730-7