A MARSHALL CAVENDISH **34** COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE

UK £1.00 Republic of Ireland £1.25
Malta 85c Australia $2.25
New Zealand $2.95

INPUT

Vol. 3 **No 34**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS
Front cover, Dave King, Snakes by Mike Culling. Pages 1049, 1050, 1051, 1054, 1055, Ellis Nadler. Pages 1052, 1053, 1068, 1070, Peter Reilly. Pages 1057, 1058, 1060, 1061, 1062, Dave King, Snakes by Mike Culling. Page 1064, Kevin O'Keefe. Pages 1068, 1070, Berry Fallon Design. Pages 1069, 1071, 1072, 1073, Oracle. Pages 1074, 1077, 1078, Ann Axworthy.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Typeset by MS Filmsetting Limited, Frome, Somerset. Printed by Cooper Clegg Web Offset Ltd, Gloucester and Howard Hunt Litho, London.

There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland:
Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

THE MATHEMATICS OF GROWTH

Computers are more often associated with mathematics than nature, but you'll find that they can give important insights into the way things grow and develop

Computers can be used to model all sorts of real-life situations. Surprisingly, this does not just apply to inanimate objects, and one field where this is very useful is in exploring the way living creatures behave in nature. It turns out that nature is closely linked to mathematics—often in unexpected ways—and, as usual, whenever there is a mathematical connection computers can be called in to study the process, do the calculations and display the results.

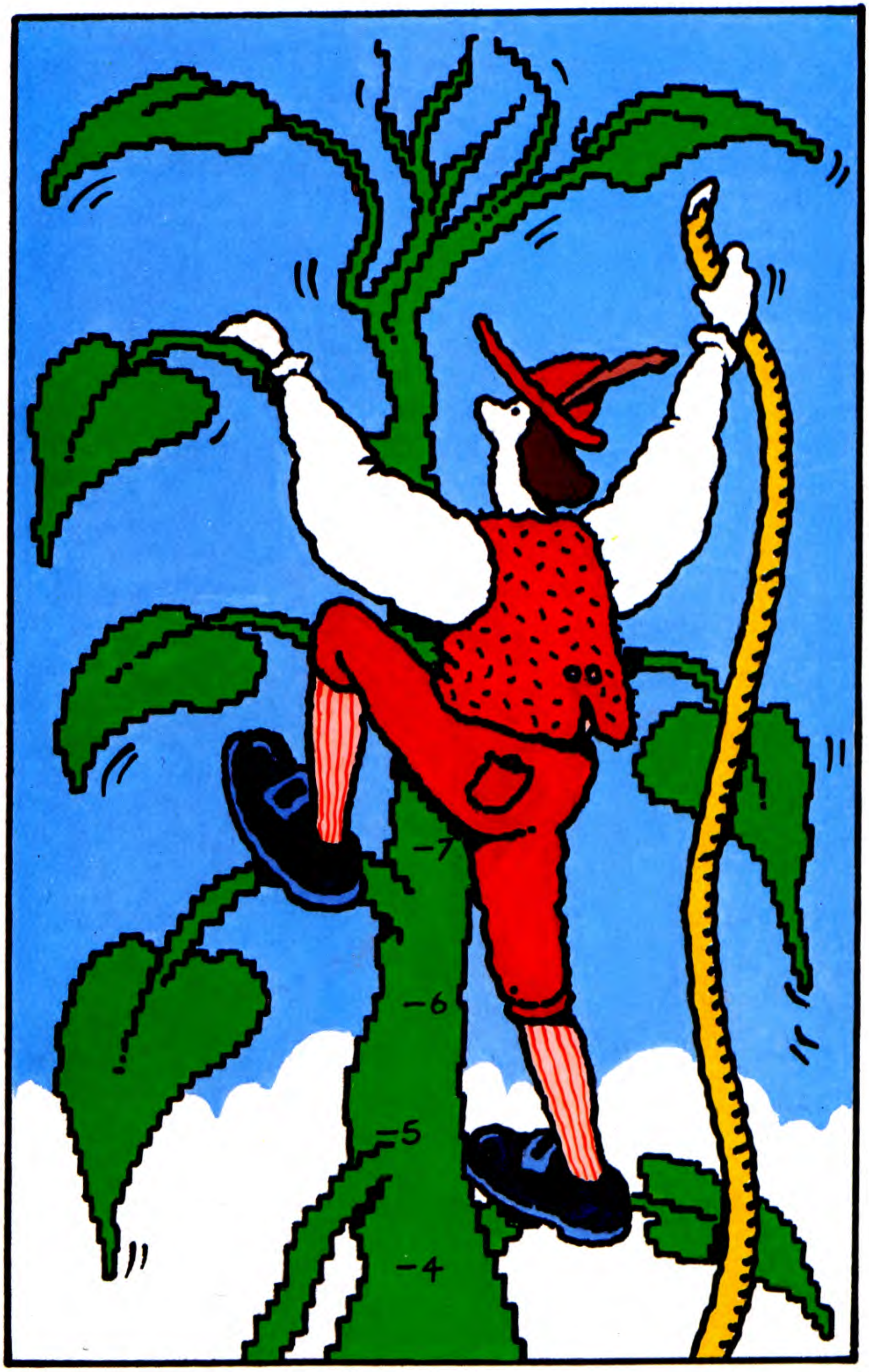
This article looks at a few ways that computers can be used to study the way things grow. All forms of life are at some stage capable of changes in size or number or form—given suitable conditions. And all these can be referred to as growth. Only the first two will be covered here, along with a few other interesting ways in which maths explains nature, or nature follows maths.

HOW THINGS GROW

First of all, it is best to understand a little more about the ways things do actually grow. Growth—change in size that is—involves the formation of new structural materials. Both plants and animals find raw materials from the environment and from respiration. Overall growth of a creature is achieved in two different ways, either as an increase in the number of cells, by cell division, or an increase in the size of individual cells by cell growth.

MEASURING SIZE

Since growth occurs in such different ways it is not immediately obvious how to measure it. Is it best to measure the weight, as with babies, or the height, as with growing children? Or would volume be a more accurate measurement, or surface area? As usual there is no one answer, and different types of measurements are suitable for different creatures. But one of the most interesting studies



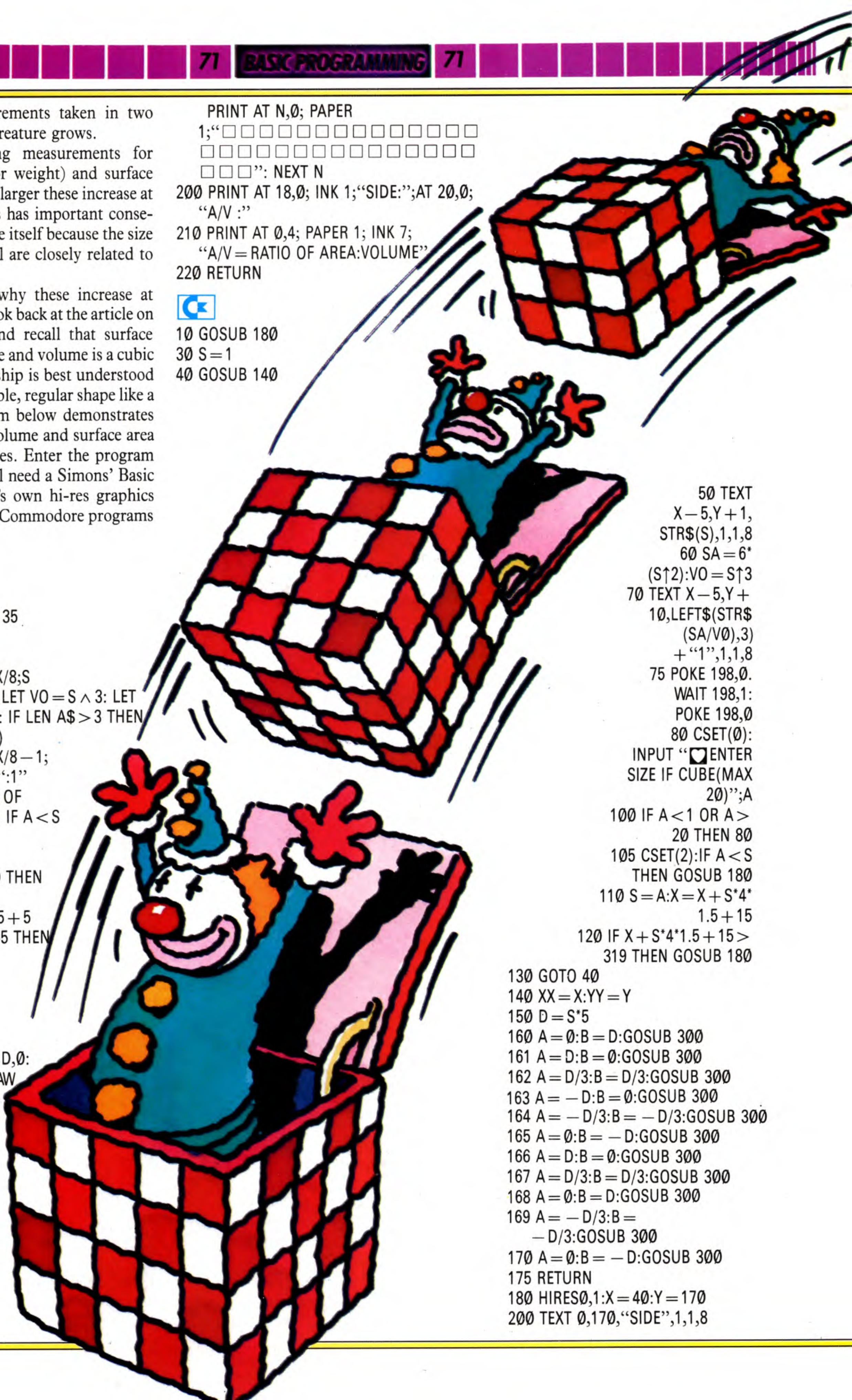
is to compare measurements taken in two different ways as the creature grows.

The most revealing measurements for animals are volume (or weight) and surface area. As an animal gets larger these increase at different rates and this has important consequences for the creature itself because the size and shape of an animal are closely related to its way of life.

You can soon see why these increase at different rates if you look back at the article on pages 404 and 412 and recall that surface area is a square measure and volume is a cubic measure. The relationship is best understood by an example of a simple, regular shape like a cube—and the program below demonstrates what happens to the volume and surface area for different sized cubes. Enter the program now and RUN it. You'll need a Simons' Basic cartridge (or *INPUT*'s own hi-res graphics utility) for most of the Commodore programs in this article:

```
10 GOSUB 180
20 LET X=45: LET Y=35
30 LET S=2
40 GOSUB 140
50 PRINT AT 18,S/5+X/8;S
60 LET SA=6*(S∧2): LET VO=S∧3: LET
   A$=STR$ (SA/VO): IF LEN A$>3 THEN
   LET A$=A$ ( TO 3)
70 PRINT AT 20,S/5+X/8-1;
   PAPER 0; INK 7;A$;":1"
80 INPUT "ENTER SIZE OF
   CUBE (MAX 20)";A: IF A<S
   THEN GOSUB 180
90 LET S=A
100 IF S<1 OR S>20 THEN
   GOTO 80
110 LET X=X+S*5*1.5+5
120 IF X+S*5*1.5>255 THEN
   GOSUB 180:
   LET X=45
130 GOTO 40
140 PLOT X,Y
150 LET D=S*5
160 DRAW 0,D: DRAW D,0:
   DRAW D/3,D/3: DRAW
   -D,0:DRAW -D/3
   -D/3:DRAW0,-D
   DRAW D,0: DRAW
   D/3,D/3: DRAW
   0,D: DRAW -
   D/3,-D/3:
   DRAW 0,-D
170 RETURN
180 BORDER 0:
   PAPER 4: INK
   0: CLS
190 FOR N=0 TO 8:
   PRINT AT N,0; PAPER
   1;"□□□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□□□
   □□□": NEXT N
200 PRINT AT 18,0; INK 1;"SIDE:";AT 20,0;
   "A/V :"
210 PRINT AT 0,4; PAPER 1; INK 7;
   "A/V=RATIO OF AREA:VOLUME"
220 RETURN
```

```
10 GOSUB 180
30 S=1
40 GOSUB 140
50 TEXT
   X-5,Y+1,
   STR$(S),1,1,8
60 SA=6*
   (S↑2):VO=S↑3
70 TEXT X-5,Y+
   10,LEFT$(STR$
   (SA/V0),3)
   +"1",1,1,8
75 POKE 198,0.
   WAIT 198,1:
   POKE 198,0
80 CSET(0):
   INPUT "♥ENTER
   SIZE IF CUBE(MAX
   20)";A
100 IF A<1 OR A>
   20 THEN 80
105 CSET(2):IF A<S
   THEN GOSUB 180
110 S=A:X=X+S*4*
   1.5+15
120 IF X+S*4*1.5+15>
   319 THEN GOSUB 180
130 GOTO 40
140 XX=X:YY=Y
150 D=S*5
160 A=0:B=D:GOSUB 300
161 A=D:B=0:GOSUB 300
162 A=D/3:B=D/3:GOSUB 300
163 A=-D:B=0:GOSUB 300
164 A=-D/3:B=-D/3:GOSUB 300
165 A=0:B=-D:GOSUB 300
166 A=D:B=0:GOSUB 300
167 A=D/3:B=D/3:GOSUB 300
168 A=0:B=D:GOSUB 300
169 A=-D/3:B=
   -D/3:GOSUB 300
170 A=0:B=-D:GOSUB 300
175 RETURN
180 HIRES0,1:X=40:Y=170
200 TEXT 0,170,"SIDE",1,1,8
```



```
    210 TEXT 60,0,"A/V=RATIO
      OF AREA: VOLUME",1,2,8
  220 RETURN
  300 LINE XX,YY,XX+A,YY-B,
1:XX=XX+A:YY=YY-B:RETURN
```

```
10 MODE1
20 PROCSCREEN
30 INPUT"ENTER SIZE OF CUBE",S:
   S=INT(S):IF S<1 OR S>40 THEN 30
40 PRINT:PRINT
50 PROCCUBE:GOTO30
100 DEF PROCCUBE
110 IF PX+S*20>1150 THEN PROCSCREEN
120 MOVEPX,PY+16*S:DRAWPX,PY:DRAW
   PX+16*S,PY:DRAWPX+16*S,PY+16*S:
   DRAWPX,PY+16*S:DRAWPX+4*S,PY+
   20*S:DRAWPX+20*S,PY+20*S:DRAW
   PX+20*S,PY+4*S:DRAWPX+16*S,PY
130 MOVEPX+16*S,PY+16*S:DRAWPX+
   20*S,PY+20*S
140 VDU5:MOVEPX+S*8-32,PY-32:
   PRINT;S
150 SA=6*S*S:VOL=S*S*S:R=INT(10*SA/
   VOL)/10:MOVEPX+S*8-64,124:PRINT;R
   ":1":VDU4
160 PX=PX+S*20+120
170 ENDPROC
180 DEF PROCSCREEN
190 VDU 26,12
200 PRINT"A:V THE RATIO OF SURFACE
   AREA TO VOLUME"
210 COLOUR131:COLOUR0:PRINTTAB(0,28)
   "A:V":COLOUR3:COLOUR128
220 PX=188:PY=188
230 VDU 28,0,31,39,29
240 ENDPROC
```

```
10 GOSUB 180:CLS
20 X=45:Y=156
30 S=2
40 GOSUB 140
60 SA=6*S*S:VO=S*S*S
70 PRINT" SIZE =";S;TAB(11);"SA/VOL
   RATIO =";:PRINTUSING"#.##:1";
   SA/VO:PRINT
80 INPUT"ENTER SIZE OF CUBE (1-20)";A:
   IF A<S GOSUB180
90 S=A
100 IF A<1 OR S>20 THEN 80
110 X=X+S*7.5+5
120 IF X+S*7.5>255 GOSUB180:X=45
130 GOTO 40
140 SCREEN1,0
150 DRAW"BM"+STR$(INT(X))+","+
   STR$(INT(Y))+"S"+STR$(INT(S*3))+
   "C1E2NR6U6C4R6G2L6NE2D6R6NU6E2U
   6"
160 IF INKEY$<>"□" THEN 160
170 RETURN
180 PMODE3:PCLS2
190 RETURN
```

The program lets you enter a number for the dimensions of a cube. It then draws the cube and prints out the ratio of surface area to volume. Start off with a small cube, say with a side of four units, then enter progressively larger numbers to simulate growth. You'll see that as size increases, the ratio of area to volume decreases. In other words the volume (or weight) increases at a much faster rate than the surface area.

Most of the program is concerned with formatting the screen and drawing the cubes but the important part is contained in Line 60 or in Line 150 on the Acorn. S is the length of one side of the cube, so the surface area of one face of the cube is S*S and the surface area of the whole cube (there are six faces) is 6*S*S. The volume on the other hand is S cubed, or S*S*S. The ratio is simply area divided by volume, worked out later on in the same line or in Line 70 on the Commodore and Vic. For example, when S equals 2 the ratio is 6*2*2 over 2*2*2 which equals 3:1. If the size is doubled the ratio is 6*4*4 over 4*4*4 which is 3:2 or 1.5:1 showing that the area is relatively much less.

Animals are not shaped like a cube, of course, and you might like to adapt the program to make it more realistic. For example small, round animals like mice are better represented as a sphere, while tall, thin animals like humans are closer to a collection of cylinders—one each for the body, arms and legs. When you adapt the program you'll need to know that the surface area of a sphere is $4*PI*radius^2$ and the volume is $4/3*PI*radius^3$. And for a cylinder the area is $2*PI*radius*height$ plus $2*PI*radius^2$ and the volume is $PI*radius^2*height$.

However the general principles are the same and for any of the shapes you'll find that the area increases at a slower rate than the volume.

GROWTH AND LIFE

These mathematical relationships are very important in the animal kingdom and affect the habitats and activities of animals. A mouse has a small volume and mass and so has a high relative surface area. This means that it will radiate heat rapidly from its body (depending on its environment). In order to survive the mouse must keep warm. It does this by burning up food for energy. Due to its size the mouse is forced to eat as much as one half of its body weight in food every day, just to survive.

In comparison, an elephant has a very large mass and volume with a low relative surface area. This means that it radiates proportionately less heat than a mouse so it does not require such a high proportion of its body weight in food every day. Such size relations explain why large animals are much better at survival in cold Arctic conditions where food is scarce.

BONES, MICE AND ELEPHANTS

Mass to area relations also help to explain why animals and plants are limited to a maximum physical size. Given the proportions of a particular species there is a very definite limit to the size it can reach. The ultimate size of an animal is limited by the size of its supporting bones.

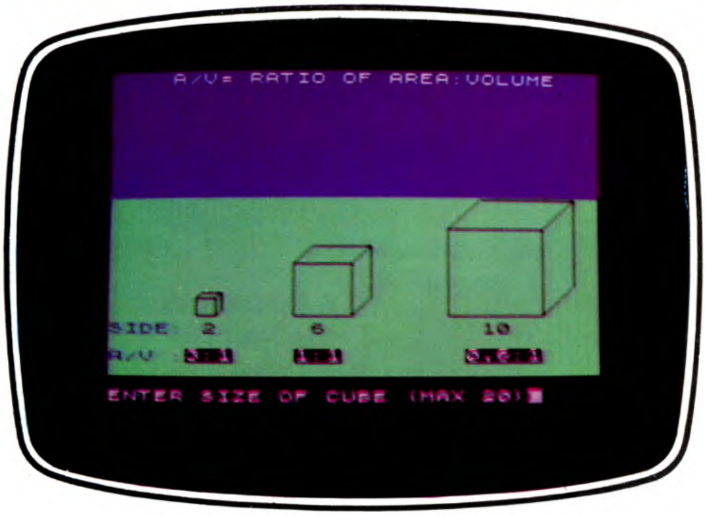
If you double the size of an animal (that is height, length and width) the weight increases by eight times but the area of supporting bone increases by only four times. To support the increase in weight the bones would have to be disproportionately thicker, for the size of the animal, as the scale increases. Eventually the animal would be so cumbersome it would be unable to move on land.

So simple scaling up won't do. If a mouse were enlarged to the size of an elephant, keeping the same proportions, then its limbs would be totally unable to support it. This is why species of different sizes have such different overall proportions.

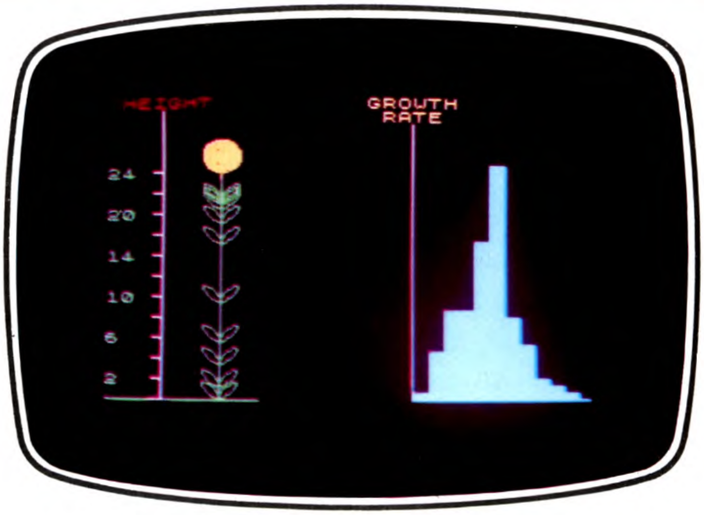
A weight of ten tonnes is near the limit for the largest land animal, the elephant. The very largest dinosaurs weighed up to 80 tonnes but may have spent much of their time standing partially immersed in water. Marine animals have the advantage of support from the surrounding water. This reduces the limits to their growth; blue whales can weigh more than 150 tonnes and reach 30 metres in length. Obviously it would be virtually impossible for such an animal to have limbs that could support its bulk out of water. Growth limits to marine animals are mainly related to heat losses.

RATE OF GROWTH

Apart from determining how large a creature can grow, it is also interesting to find out how



Comparing volume and surface area



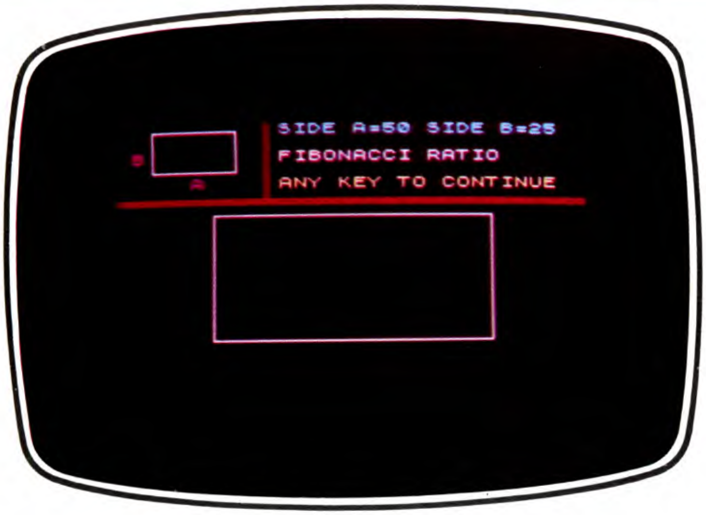
How the growth rate of a plant changes

fast it grows. The next program shows how the rate of growth changes as a plant grows from a seedling to maturity. The plant grows slowly at first then rapidly until growth limits are reached. It then slows down again until it stops and eventually dies. Enter and RUN the program now:

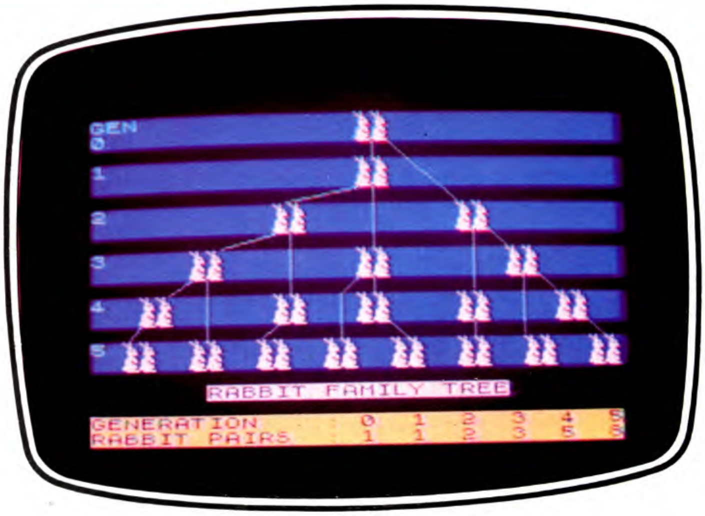
```
10 DIM G(11)
20 DATA 2,9,22,35,58,92,104,112,115,117,118
30 FOR N=1 TO 11: READ G(N): NEXT N
40 BORDER Ø: PAPER Ø: INK 4: CLS
50 LET X=6Ø: LET Y=Ø
60 DRAW 8Ø,Ø
70 PLOT 3Ø,Ø: DRAW INK 7;Ø,168
80 PRINT INK 2;AT Ø,1;"HEIGHT"
90 FOR N=Ø TO 138 STEP 12
100 PLOT INK 7;3Ø,N
110 DRAW INK 7;-5,Ø
120 NEXT N
130 PRINT AT 2Ø,Ø;"2";AT 17,Ø;"6";AT
   14,Ø;"1Ø";AT 11,Ø;"14";AT 8,Ø;"2Ø";AT
   5,Ø;"24"
140 PLOT 16Ø,Ø: DRAW INK 7;Ø,168
150 PLOT 16Ø,Ø: DRAW INK 7;95,Ø
160 PRINT AT Ø,17; INK 6;"GROWTH";AT
   1,18;"RATE"
170 LET C=1
180 LET GX=161
190 LET GY=1
200 FOR N=1 TO 117
210 IF G(C)<>N THEN GOTO 25Ø
220 PLOT X,Y: DRAW 9,9,PI/2: DRAW
   -9,-9,PI/2
230 DRAW -9,9,PI/2: DRAW 9,-9,PI/2
240 LET GX=GX+8: LET GY=1: LET
   C=C+1
250 PLOT X,Y
260 FOR K=Ø TO 3
270 PLOT INK 5;GX,GY+K: DRAW INK 5;8,Ø
280 NEXT K
290 LET GY=GY+4
300 LET Y=Y+1
310 NEXT N
320 FOR Y=117 TO 14Ø
330 PLOT X,Y
340 NEXT Y
350 FOR R=1 TO 1Ø STEP .3
360 CIRCLE INK 6;X,Y,R
370 NEXT R
380 GOTO 38Ø
```

```
10 DIM G(11)
20 DATA 2,9,22,35,58,92,1Ø4,112,115,
   117,118
30 FOR N=1 TO 11:READ G(N):NEXT N
40 HIRES Ø,1:MULTI 5,6,13:COLOUR 7,1
50 X=6Ø:Y=Ø
60 LINE Ø,199,159,199,2
70 LINE 3Ø,198,3Ø,4Ø,2
80 TEXT Ø,2Ø,"HEIGHT",2,1,6
90 FOR N=Ø TO 138 STEP 12
100 LINE 3Ø,187-N,27,187-N,1
105 NU=NU+2
130 TEXT Ø,184-N,STR$(NU),3,1,6:NEXT N
140 LINE 1ØØ,198,1ØØ,4Ø,2
160 TEXT 9Ø,2Ø,"GROWTH RATE",1,1,6
170 C=1
180 GX=1Ø1
190 GY=1
200 FOR N=1 TO 117
210 IF G(C)<>N THEN 25Ø
220 FOR Z=1 TO 5:LINE X,199-Y,X+15-
   Z,(199-Y)-Z*(N/1ØØ),INT(RND(1)*2)*
   2+1
230 LINE X,199-Y,X-15+Z,(199-Y)-Z*
   (N/1ØØ),1:NEXT Z
240 GX=GX+5:GY=1:C=C+1
250 PLOT X,199-Y,1
260 FOR K=Ø TO 3
270 LINE GX,198-GY-K,GX+3,198-
   GY-K,RND(1)*2+2
280 NEXT K
290 GY=GY+4
300 Y=Y+1
310 NEXT N
330 LINE X,199-Y,X,169-Y,1
345 LOW COL 7,2,8
350 XX=X:YY=169-Y:FOR R=Ø TO
   2*π STEP .1
355 X1=X+SIN(R)*(RND(1)*1Ø+1Ø):
   Y1=(169-Y)+COS(R)*(RND(1)*
   1Ø+1Ø)
360 LINE XX,YY,X1,Y1,3:XX=X1:YY=Y1
365 LINE X,169-Y,X1,Y1,RND(1)*3+1
370 NEXT R
380 GOTO 38Ø
```

```
10 MODE1
20 PROCINIT
30 PROCGROW
100 GOTO 1ØØ
1000 DEF PROCLEAVES
1010 VDU 29,PX;PY;:MOVEØ,Ø:DRAW -5Ø,
   4Ø
1020 DRAW -9Ø,2Ø:DRAW -45,8Ø
1030 DRAW Ø,Ø:DRAW 45,8Ø
1040 DRAW 9Ø,2Ø:DRAW 5Ø,4Ø
1050 DRAW Ø,Ø:VDU 29,Ø;Ø;
1060 ENDPROC
1070 DEF PROCFLOWER
1080 MOVEPX,G(11):DRAWPX,G(11)+15Ø
1090 GCOLØ,2:FOR T=Ø TO 2*PI STEP PI/15
1100 MOVEPX,G(11)+15Ø:PLOT17,5Ø*SINT,
   5Ø*COST:NEXT
1110 ENDPROC
1120 DEF PROCINIT
1130 PX=35Ø:GX=8ØØ:GY=1ØØ
1140 VDU 19,1,2,Ø,Ø,Ø
1150 DIM G(11)
```



Finding a well-proportioned rectangle



A population explosion of rabbits

```
1160 FOR T=Ø TO 11:READ A:G(T)=A*5+
   100:NEXT
1170 DATA Ø,2,9,22,35,58,92,104,112,115,
   117,118
1180 MOVE15Ø,1ØØ:DRAW15Ø,9ØØ
1190 VDU 5:FOR T=Ø TO 12
1200 MOVE12Ø,1ØØ+T*64:DRAW15Ø,
   1ØØ+T*64
1210 MOVE2Ø,116+T*64:PRINT;T*2:NEXT:
   VDU4
1220 PRINTTAB(Ø,2)"HEIGHT□□□□□
   □□□□□□□□□□GROWTH RATE"
1230 MOVEGX,GY+8ØØ:DRAWGX,GY:DRAW
   GX+4ØØ,GY
1240 ENDPROC
1250 DEF PROCGROW
1260 FOR Y=Ø TO 1Ø
1270 FOR T=G(Y) TO G(Y+1)
1280 GCOLØ,3:MOVEGX+Y*32,GY+(T-G
   (Y))*4:DRAWGX+Y*32+31,GY+(T-G
   (Y))*4
1290 GCOLØ,1:PLOT69,PX,T
1300 NEXT
1310 PY=G(Y+1)
1320 PROCLEAVES
1330 NEXT
1340 PROCFLOWER
1350 ENDPROC
```

```
20 DATA 2,9,22,35,58,92,104,112,115,117,118
30 FOR N=Ø TO 1Ø:READ G(N):NEXT
40 PMODE3:PCLS:SCREEN1,1
50 X=6Ø:Y=19Ø
60 LINE(8,23)-(8,192),PSET:LINE-(8Ø,
   192),PSET
80 DRAW"BM2Ø,6S24C7D2BRUNLUBR2LDN
   RDRBRU2BR2LD2RUS8NLS24BED2BRUN
   LUBRS16RND3RC8"
90 FORN=47 TO 191 STEP 12
100 LINE(2,N)-(8,N),PSET
120 NEXT
140 LINE(158,23)-(158,191),PSET
150 LINE-(255,191),PSET
160 DRAW"BM176,6C6S24LD2RUS8NLS24BE
   ND2RDLFBRNU2RU2LBR2D2EFU2BRS16R
   ND3RS24BRD2BRUNLUBM182,24ND2RDL
   FBRU2RDNLDS16BR2U3LR2S24BR2LDNR
   DR"
170 GX=161:GY=19Ø
190 COLOR6,7
200 FOR N=1 TO 117
210 IF G(C)>N THEN 25Ø
220 CIRCLE(X,Y-4),15,6,.4,Ø,.5
230 CIRCLE(X-16,Y),16,6,.4,.75,1:CIRCLE
   (X+16,Y),16,6,.4,.5,.75
240 GX=GX+8:GY=19Ø:C=C+1
250 PSET(X,Y,6)
260 FOR K=Ø TO 3
270 LINE(GX,GY-K)-(GX+8,GY-K),
   PRESET
280 NEXT
290 GY=GY-4
300 Y=Y-1
310 NEXT
320 FOR Y=75 TO 58 STEP-1
330 PSET(X,Y,6)
340 NEXT:POKE178,54
350 FOR R=1 TO 15
360 CIRCLE(X,Y),R,,.8
370 NEXT
380 GOTO 38Ø
```

The programs show graphically what happens to the plant. The DATA in Line 2Ø (Line 117Ø on the Acorns) gives the size of the plant measured at regular intervals. This is used to draw both the plant and the graph showing the change in the rate of growth. The plant is drawn by the routine at Lines 2ØØ to 31Ø (or PROCLEAVES and PROCGROW on the Acorns). The axes for the graph and the scales are set up by Lines 4Ø to 16Ø (PROCINIT on the Acorns), and the bar chart itself is drawn by Lines 26Ø to 28Ø (Line 128Ø in PROCGROW on the Acorns). Finally, the flower is drawn by Lines 35Ø to 38Ø (or PROCFLOWER).

The graph shows clearly that the plant grows slowly at first, speeds up and then slows down when it has reached its maximum size. The DATA is taken from a real experiment which measured how the area of a cucumber leaf changes as it grows. But the same figures can be used to represent how a complete plant grows.

GENERATIONS AND NUMBERS

As animals breed they multiply to form a number of generations. Take the case of rabbits. One pair of rabbits—the first generation—produces another pair. These form the second generation. The original pair or rabbits then produces another pair and the second generation contains two pairs. Eventually a series of generation numbers are built up as follows: 1, 1, 2, 3, 5, 8, 13, 21, and so on. The next program shows graphically how the numbers of rabbits increase over the generations and you'll see that these numbers are indeed built up:

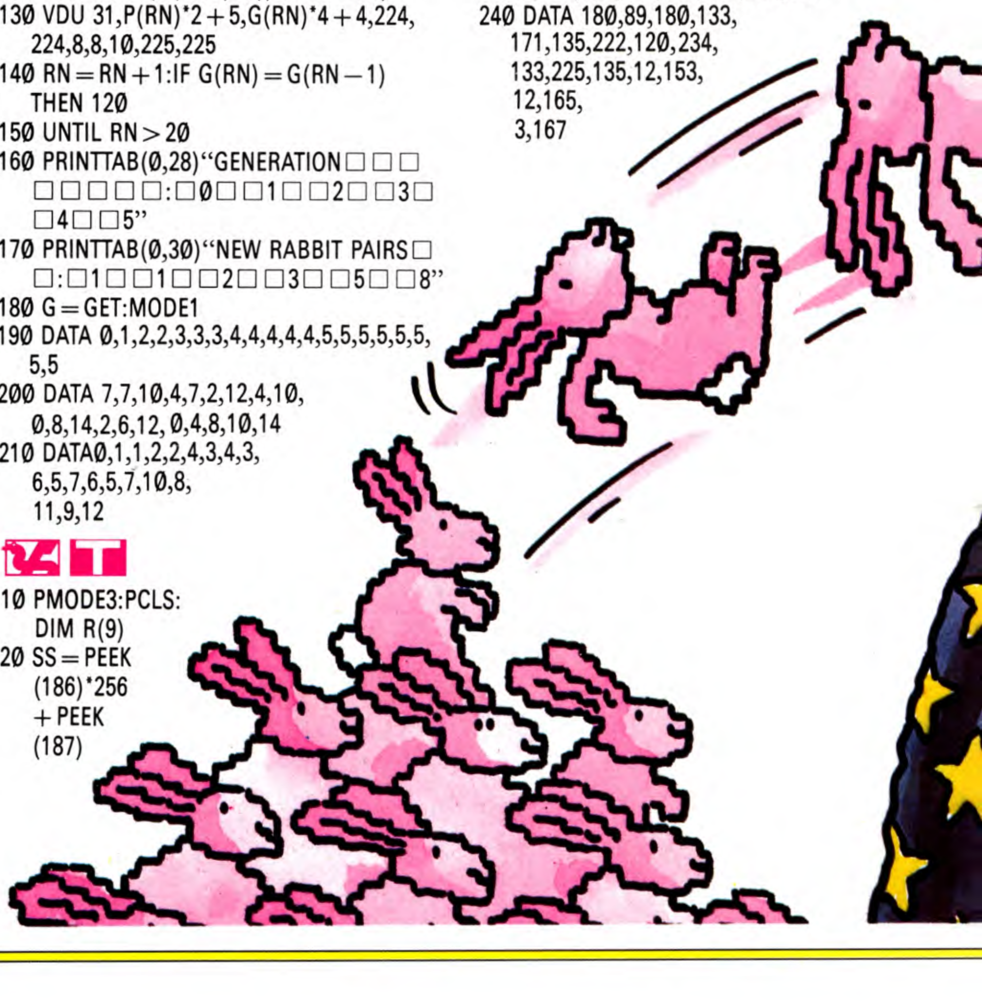
```
10 BORDER Ø: PAPER 1: INK 7: CLS
20 FOR N=Ø TO 7: READ A: POKE USR
   "a"+N,A: NEXT N
30 FOR N=Ø TO 7: READ A: POKE USR
   "b"+N,A: NEXT N
40 LET C$="□□□□□□□□□□□□
   □□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□□
   □□□□□□□□□"
50 LET A$=CHR$ 144: LET B$=CHR$ 145
```

```
60 PAPER 0: CLS : PAPER 1
70 FOR N=1 TO 6: PRINT C$'': NEXT N
80 PRINT INK 5;AT 0,0;"GEN";'"0"''"1"''
   "2"''"3"''"4"''"5"
90 FOR N=1 TO 20
100 IF N>1 THEN FOR P=1 TO 10: BEEP
   .01,P: NEXT P
110 READ X,Y: PRINT AT Y,X;A$;A$;AT
   Y+1,X;B$;B$
120 NEXT N
130 FOR N=1 TO 20
140 READ X,Y,XX,YY: PLOT X,Y: DRAW INK
   4;XX,YY
150 NEXT N
160 PRINT AT 18,7; INVERSE 1; "RABBIT
   FAMILY TREE"
170 INK 6: INVERSE 1: PRINT '"GENERATION
   □□□□:□0□□1□□2□□3□
   □4□□5"!"RABBIT PAIRS□□:□1
   □□1□□2□□3□□5□□8"
180 GOTO 180
190 DATA 144,80,48,28,52,62,62,24,60,126,
   118,120,126,254,252,63
200 DATA 16,0,16,3,22,6,11,6,16,9,6,9,11,12,
   25,9,22,12,3,12,6,15,14,15,16,12,28,12,26,
   15,2,15,10,15,18,15,22,15,30,15
210 DATA 136,159,0,-7,144,159,32,-31,
   128,135,-34,-7,136,135,0,-31,184,
   112,0,-31,192,112,8,-8,88,111,-25,
   -7,96,111,0,-31
220 DATA 48,87,-10,-7,56,87,0,-31,24,
   63,-5,-7,90,63,-8,-7,128,87,-8,
   -8,120,79,0,-23,136,87,0,-7
230 DATA 144,63,7,-7,184,63,0,-7,208,88,
   0,-31,216,88,7,-7,240,63,7,-7
```

```
1 PRINT "♥■"SPC(9)"■ππ■■■
   SPC(9)"1"
2 PRINT SPC(10)"■"
3 PRINT "■■■■■■■■■■■■■
   ■■■■■■■■■■■"
4 PRINT SPC(10)"■■■"
5 PRINT "1"SPC(8)"ππ■■■"SPC(6)
   "1"
6 PRINT SPC(9)"■■■■■■"
7 PRINT "□■■■■■■■■■■■■
   ■■■■□■■■■■"
8 PRINT SPC(7)"■□□■"SPC(4)"□"
9 PRINT "2"SPC(4)"■ππ■■■■
   ■ππ■■■■■2"
10 PRINT SPC(6)"■■■■■■■■
   ■■■"
11 PRINT "■■■■■■■■■■■■
   ■■■■■■■■■■■■"
12 PRINT "3■■■ππ■■■ππ
   ■■■■■ππ■■3"
13 PRINT SPC(4)"■■■■■■■■
   ■■■■■■"
14 PRINT SPC(4)"□■■■■■■■
   ■■■■■■■"
15 PRINT "■■■■■■■■■■■
   ■■■■■■■■■■■■■"
16 PRINT "4■ππ■■ππ■■π
   π■■ππ■■ππ5■■"
19 PRINT "■■■■■■■■■■■
   ■■■■■■■■■■■■"
 20 PRINT "■■□RABBIT FAMILY
   TREE□"
22 GOTO 22
```

```
10 MODE1:VDU 23;8202;0;0;0;
20 DIM G(21),P(20),PAR(20)
30 FOR T=1 TO 20:READ G(T):NEXT:G
   (21)=0
40 FOR T=1 TO 20:READ P(T):NEXT
50 FOR T=1 TO 20:READ PAR(T):NEXT
60 VDU 23,224,144,80,48,28,52,62,62,24,23,
   225,60,126,118,120,126,254,252,63
70 PRINTTAB(11,1)"RABBIT FAMILY TREE"'
   "GEN"
80 FOR T=0 TO 5:PRINTTAB(1,T*4+4);T:
   NEXT
90 RN=1:REPEAT
100 IF RN=1 THEN 130
110 D=INKEY(150)
120 MOVEP(RN)*64+192,1024-(G
   (RN)*128+112):DRAW P(PAR(RN))*64+
   192,1024-(G(PAR(RN))*128+208)
130 VDU 31,P(RN)*2+5,G(RN)*4+4,224,
   224,8,8,10,225,225
140 RN=RN+1:IF G(RN)=G(RN-1)
   THEN 120
150 UNTIL RN>20
160 PRINTTAB(0,28)"GENERATION□□□
   □□□□□:□0□□1□□2□□3□
   □4□□5"
170 PRINTTAB(0,30)"NEW RABBIT PAIRS□
   □:□1□□1□□2□□3□□5□□8"
180 G=GET:MODE1
190 DATA 0,1,2,2,3,3,3,4,4,4,4,4,5,5,5,5,5,5,
   5,5
200 DATA 7,7,10,4,7,2,12,4,10,
   0,8,14,2,6,12,0,4,8,10,14
210 DATA0,1,1,2,2,4,3,4,3,
   6,5,7,6,5,7,10,8,
   11,9,12
```

```
10 PMODE3:PCLS:
   DIM R(9)
20 SS=PEEK
   (186)*256
   +PEEK
   (187)
30 FOR K=SS TO SS+480 STEP 32
40 READA,B:POKEK,A:POKEK+1,B
50 NEXT
60 GET(2,0)-(13,15),R,G
70 PUT(14,0)-(25,15),R,PSET:GET(2,0)-
   (25,15),R,G
80 PCLS4:SCREEN1,0
90 COLOR3:FOR K=0 TO 5
100 LINE(0,32*K)-(255,32*K+24),
   PSET,BF
110 NEXT
120 COLOR1:FOR N=1 TO 20
140 READ X,Y:PUT(X,Y)-(X+23,Y+15),R,
   PSET
150 IF N>1 THEN PLAY"01T0CCEFGAB"
160 READ X,Y,XX,YY:LINE(X,Y)
170 NEXT
180 GOTO 180
190 DATA 169,170,153,170,165,170,165,170,
   169,90,165,154,165,86,165,90,169,106
200 DATA 169,106,165,90,165,154,165,106,
   165,106,149,106,149,106,149,106,165,90
210 DATA 114,7,124,24,124,38,114,39,113,56,
   69,70,59,71,138,23,178,68,171,71,57,89,
   42,100,31,103
220 DATA 124,57,124,101,115,103,195,87,
   208,101,199,103
230 DATA 29,120,12,133,3,135,68,89,68,133,
   59,135,124,121,124,133,115,135
240 DATA 180,89,180,133,
   171,135,222,120,234,
   133,225,135,12,153,
   12,165,
   3,167
```



```
250 DATA 40,121,42,165,33,167,68,153,72,
    165,63,167,113,120,98,165,93,167,124,
    153,144,165,137,167
260 DATA 178,153,176,165,167,167,208,121,
    206,165,197,167,234,153,234,165,225,167,
    124,70,124,70
```

This time, the Commodore and Vic programs work without Simons' Basic or a Super Expander. They simply print the rabbits as π signs and position the generations straight from the strings in the program. The Spectrum, Dragon and Tandy programs start by creating the rabbit UDGs from DATA in Lines 190 and 200. Lines 40 to 80 on the Spectrum and 60 to 110 on the Dragon and Tandy print up bars on the screen for each generation then the next section up to Line 180 prints the rabbits and the lines joining the generations. The DATA for the lines and positions is READ from Lines 210 to the end.

The Acorn program first sets up arrays for the generation number G(T), the x coordinate of the rabbits P(T) and the parent number of each rabbit PAR(T). Line 60 creates the UDGs and the next two lines print the bars for each generation. The loop in Lines 90 to 150 draws each pair of rabbits and the lines linking the generations.

As organisms multiply they also spread out and colonise more areas. A good example of this is the way bacteria divide and multiply. Computers can also be used to represent this and a well-known program is the game of Life. Although this can be written in BASIC it is impossibly slow. However, Spectrum users will already have a good example on their 'Horizons' introductory tape.

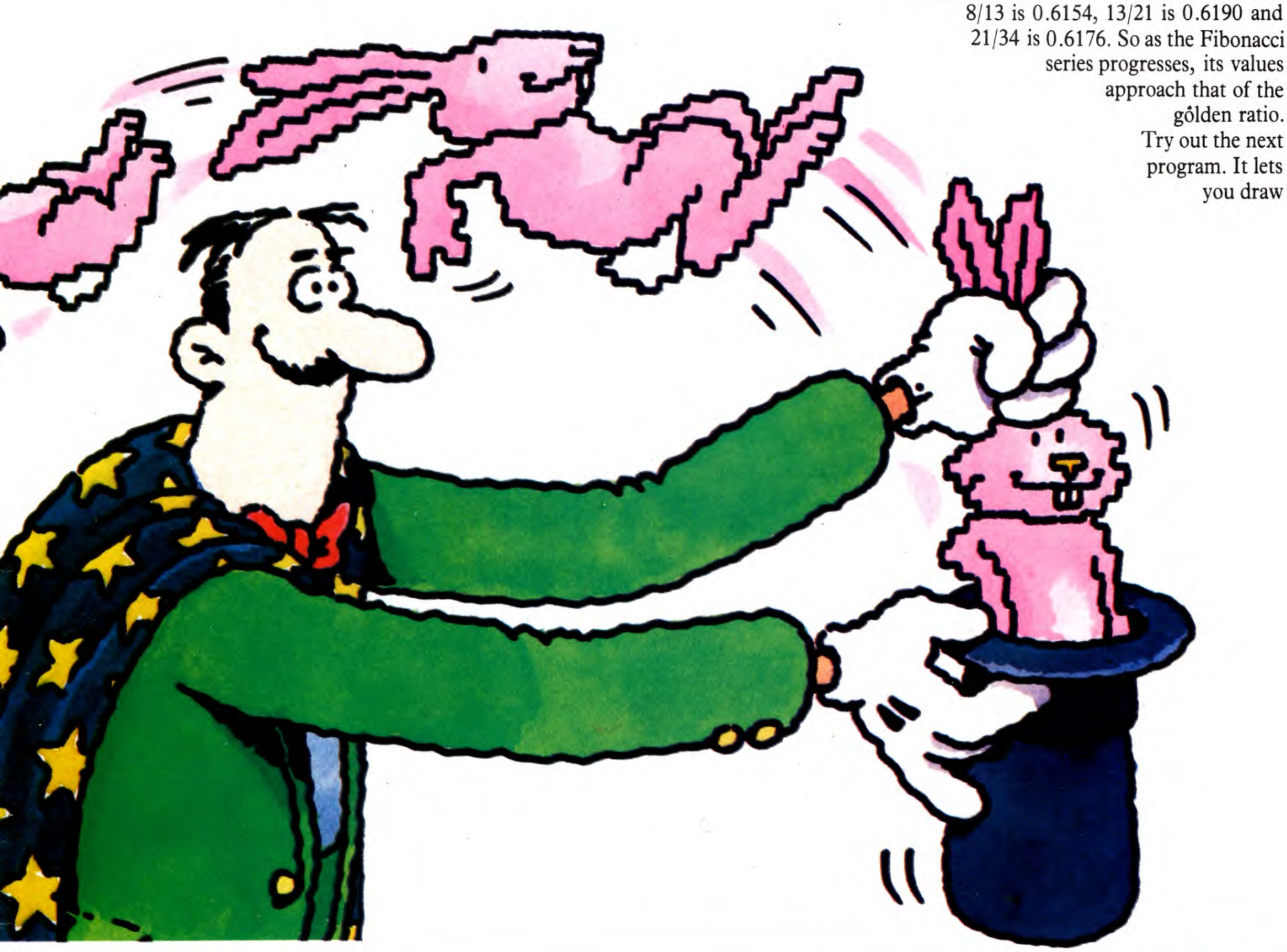
FIBONACCI NUMBERS

The series of numbers listed above have some interesting properties and are observed throughout nature and art. They are called the Fabonacci numbers after the thirteenth century Italian mathematician. Each number can be calculated by adding together the previous two numbers in the series.

One unusual property of these numbers is seen in taking any three of them in succession. Multiply the first and the last together and compare this to the square of the middle one. The results will always differ by one. For example with numbers 5, 8 and 13, 5 times 13 is 65 and 8 squared is 64.

Dividing each number by its right-hand neighbour results in a series of fractions or ratios and these are also found in nature and art. For example, it was discovered that not all rectangular shapes are equally pleasing to look at. Some are too narrow or too long or too fat. It is easily shown that the best-looking rectangle has a special ratio of width to length known as the golden ratio. This ratio was found to be equal to $(SQR(5)-1)/2$, which works out as 0.6180. If you work out any of the Fibonacci fractions you'll find that the larger numbers you use the nearer the result gets to the golden section. For example 8/13 is 0.6154, 13/21 is 0.6190 and 21/34 is 0.6176. So as the Fibonacci series progresses, its values approach that of the gôlden ratio.

Try out the next program. It lets you draw



different shaped rectangles so you can judge for yourself which proportions look best:

```
1Ø DIM F(12): DIM D(14)
2Ø LET D(1)=1: LET D(2)=1
3Ø FOR N=3 TO 14
4Ø LET D(N)=D(N−1)+D(N−2)
5Ø NEXT N
6Ø FOR N=1 TO 12
7Ø LET F(N)=D(N)/D(N+1)
8Ø NEXT N
9Ø BORDER Ø: INK 7: PAPER Ø: CLS
1ØØ LET A=15: LET B=8
11Ø LET X=2Ø: LET Y=17Ø
12Ø GOSUB 31Ø
13Ø PLOT Ø,13Ø: DRAW INK 2;255,Ø
14Ø PLOT Ø,128: DRAW INK 2;255,Ø
15Ø PLOT 8Ø,13Ø: DRAW INK 2;Ø,45
16Ø PLOT 82,13Ø: DRAW INK 2;Ø,45
17Ø PRINT AT 2,1; INK 3;"B";AT 4,5;"A"
18Ø INPUT "ENTER LENGTH OF SIDE A (MAX
   7Ø)";A
19Ø IF A<1 OR A>7Ø THEN GOTO 18Ø
2ØØ INPUT "ENTER LENGTH OF SIDE B (MAX
   4Ø)";B
21Ø IF B<1 OR B>4Ø THEN GOTO 2ØØ
22Ø LET X=128−(A*3/2): LET Y=12Ø
23Ø GOSUB 31Ø
24Ø PRINT AT Ø,11; INK 5;"SIDE
   A=";A;"□SIDE B=";B
25Ø FOR N=1 TO 12
26Ø IF A/B=F(N) OR B/A=F(N) THEN
   PRINT AT 2,11;"FIBONACCI RATIO"
27Ø NEXT N
28Ø PRINT AT 4,11; FLASH 1; INK 6;"ANY
   KEY TO CONTINUE"
29Ø PAUSE Ø
3ØØ RUN
31Ø PLOT X,Y: DRAW 3*A,Ø: DRAW Ø,−3*B
32Ø DRAW −3*A,Ø: DRAW Ø,3*B
33Ø RETURN
```

```
1Ø DIM F(12),D(14)
2Ø D(1)=1:D(2)=1
3Ø FOR N=3 TO 14
4Ø D(N)=D(N−1)+D(N−2)
5Ø NEXT N
6Ø FOR N=1 TO 12
7Ø F(N)=D(N)/D(N+1)
8Ø NEXT N
9Ø HIRES Ø,1:MULTI 2,4,5:COLOUR 1,1
1ØØ A=2Ø:B=8
11Ø X=2Ø:Y=25
12Ø C=2:GOSUB 31Ø
13Ø FOR Z=1 TO 3:LINE
   Ø,7Ø+Z,159,7Ø+Z,Z:NEXT Z
17Ø TEXT 37,1Ø,"A",3,1,8:TEXT
   8,4Ø,"B",3,1,8
175 POKE 198,Ø:WAIT 198,1:POKE 198,Ø
18Ø CSET(Ø):INPUT "♥ENTER LENGTH OF
   SIDE A (MAX 7Ø)";A
19Ø IF A<1 OR A>7Ø THEN 18Ø
2ØØ INPUT "♥ENTER LENGTH OF SIDE
   B(MAX 3Ø)";B
21Ø IF B<1 OR B>3Ø THEN 2ØØ
22Ø CSET (2):MULTI
   2,4,5:X=8Ø−(A*2)/2:Y=8Ø
23Ø C=1:GOSUB 31Ø
24Ø TEXT 85,5,"SIDE A ="+STR$(A),3,1,6
245 TEXT 85,15,"SIDE B ="+STR$(B),3,1,6
25Ø FOR N=1 TO 12
26Ø IF A/B=F(N) OR B/A=F(N) THEN TEXT
   6,6Ø,"FIBONACCI RATIO",3,1,1Ø
27Ø NEXT N
29Ø POKE 198,Ø:WAIT 198,1:POKE 198,Ø
3ØØ RUN
31Ø BLOCK X,Y,X+A*2,Y+B*4,C:RETURN
```

```
1Ø MODE 1: DIM F(12),D(14)
2Ø D(1)=1:D(2)=1
3Ø FOR N=3 TO 14
4Ø D(N)=D(N−1)+D(N−2)
5Ø NEXT
6Ø FOR N=1 TO 12
7Ø F(N)=D(N)/D(N+1)
8Ø NEXT
9Ø A=22Ø:B=13Ø:VDU29,64;85Ø;
1ØØ PROCREC:VDU26:PRINTTAB(Ø,3)"B"
   TAB(5,6)"A"
16Ø GCOL,1:MOVE Ø,76Ø:DRAW 128Ø,76Ø
17Ø MOVE 35Ø,76Ø:DRAW 35Ø,1Ø24
18Ø VDU28,Ø,31,39,3Ø
19Ø INPUT"LENGTH OF SIDE A (MAX
   25Ø)",A
2ØØ IF A<1 OR A>25Ø THEN 19Ø ELSE
   A=A*4
21Ø INPUT"LENGTH OF SIDE B (MAX
   15Ø)",B
22Ø IF B<1 OR B>15Ø THEN 21Ø ELSE
   B=B*4
23Ø VDU 12,29,64Ø−A/2;35Ø−B/2;:PROCREC
24Ø VDU26:PRINTTAB(15,1)"SIDE A=";A/4;
   TAB(26,1)"SIDEB=";B/4
25Ø FOR N=1 TO 12
26Ø IF A/B=F(N) OR B/A=F(N) THEN
   COLOURØ:COLOUR131:PRINTTAB(18,3)
   "FIBONACCI RATIO":COLOUR128
27Ø NEXT
28Ø COLOUR2:PRINTTAB(13,5)"PRESS ANY
   KEY TO CONTINUE"
29Ø G=GET
3ØØ RUN
31Ø DEF PROCREC
32Ø GCOLØ,3:MOVE Ø,Ø:DRAW A,Ø
33Ø DRAW A,B:DRAW Ø,B
34Ø DRAW Ø,Ø:ENDPROC
```

```
1Ø DIMF(11),D(13)
2Ø D(Ø)=1:D(1)=1
3Ø FOR N=2 TO 13
4Ø D(N)=D(N−1)+D(N−2)
5Ø NEXT
6Ø FOR N=Ø TO 11
7Ø F(N)=D(N)/D(N+1)
8Ø NEXT
9Ø PMODE3:PCLS:CLS
1ØØ A=15:B=8
11Ø X=2Ø:Y=22
12Ø GOSUB 31Ø
13Ø COLOR4:LINE(Ø,62)−(255,62),PSET
14Ø LINE(Ø,64)−(255,64),PSET
15Ø LINE(8Ø,62)−(8Ø,17),PSET
16Ø LINE(82,62)−(82,17),PSET
17Ø DRAW"BM9,38C2S8U4R2FGNLFGL2BM
   4Ø,58U3EFDNLD2"
175 FORK=1TO9ØØ:NEXT
18Ø INPUT "ENTER LENGTH OF SIDE A (MAX
   7Ø)□";A
19Ø IF A<1 OR A>7Ø THEN 18Ø
2ØØ INPUT"ENTER LENGTH OF SIDE B (MAX
   4Ø)□";B
21Ø IF B<1 OR B>4Ø THEN 2ØØ
22Ø X=128−A*3/2:Y=72
23Ø GOSUB 31Ø
25Ø FOR N=Ø TO 11
26Ø IF A/B=F(N) OR B/A=F(N) THEN
   DRAW"BM1Ø6,44C2S8NR2D2NRD2BR4U4
   BR2ND4RFGNLFGLBR4NU4R2U4L2BR
   4DND3F2NU3DBR2U3EFDNL2D2BR4L2
   U4R2BR4L2D4R2BR2U4BR8ND4R2D2L2F2
   BR2U3EFDNLD2BR3U4LR2BR2D4BR2R2U
   4L2D4"
27Ø NEXT
28Ø IF INKEY$="" THEN 28Ø
3ØØ RUN
31Ø SCREEN1,Ø:COLOUR3
32Ø LINE(X,Y)−(3*A+X,Y+3*B),PSET,BF
33Ø RETURN
```

The program asks you to enter the lengths of the sides of a rectangle. If you enter two adjacent numbers from the Fibonacci series the program tells you it's a Fibonacci ratio.

The numbers in the series are worked out by Lines 2Ø to 5Ø simply by starting with the two ones and then adding each number to the previous one. They are stored in an array in Lines 6Ø to 8Ø. An example rectangle is drawn on the screen by Lines 9Ø to 17Ø. The INPUT routine comes next then a check to see if the values entered are a Fibonacci ratio.

Examples of the Fibonacci fractions are also found in nature too. For instance, a spiral following leaves on a stem has gaps and turns in Fibonacci ratio. Count the number of turns the spiral makes starting with one leaf up to another leaf in the same position. Then count the number of gaps between these leaves. The ratio will ususally be 5/3 or 8/5.

CLIFFHANGER: PERILS AND PRIZES

Of course you'll need rewards—like cake and lemonade—for doing all this programming. But there are risks to be run too. Slimy snakes are out to get you!

After the first screen has been put up, the other screens are simply modifications to it. In the second level, Willie has to contend with potholes. These are simply created by overwriting the slope on the first screen with the background colour in the shape of the hole you want to create. Then the snakes are added by writing that background colour. The routine that makes the snake move will be added later.

The following program adds the holes, snakes and rewards:

```
      org 58455
elb   ld a,(57344)
      ld hl,57272
      ld b,a
      inc b
      ld de,8
ab    add hl,de
      djnz ab
      push hl
      pop bc
      ld hl,191
      ld a,58
      call print
      ld a,(57344)
      cp Ø
      jr z,ed
      push af
      call hls
      pop af
      cp 1
      jr z,ed
      call snp
ed    ret
hls   ld hl,457
      call hlp
      ld hl,4Ø1
      call hlp
      ld hl,314
      call hlp
      ret
hlp   ld b,4
hlq   push bc
      ld bc,15616
      ld a,45
      call print
      ld de,32
      add hl,de
      pop bc
      djnz hlq
      ret
snp   ld hl,457
      call snq
      ld hl,4Ø1
      call snq
      ld hl,314
      call snq
      ret
snq   ld a,4
      ld bc,57232
snr   push af
      ld a,43
      call print
      ld de,32
      add hl,de
      pop af
      dec a
      jr nz,snr
      ret
print org 58217
      *
```

REWARDING ROUTINES

The accumulator is loaded with the contents of memory location 57344. This location is going to be used to store what level you are on. The HL is loaded with the address of the beginning of the rewards in the data table.

The level number is then transferred from the accumulator to the B register and the B register is incremented. The number 8 is loaded into the DE register and that is added to the data pointer in HL.

The **ad** loop then continues to add 8 onto the HL register until B has counted down to Ø. The **djnz** operates on the B register remember, decrementing and jumping if the result is not zero. This process moves the data pointer along until it points at the beginning of the right reward. Each appears in a single eight-by-eight character square, so its data uses eight bytes in the data table.

The result of this multiple addition is left in the HL register. But you want it in the BC register when you call the **print** routine. And the easiest way to transfer it is to **push** it on the stack and **pop** it off into the other register.

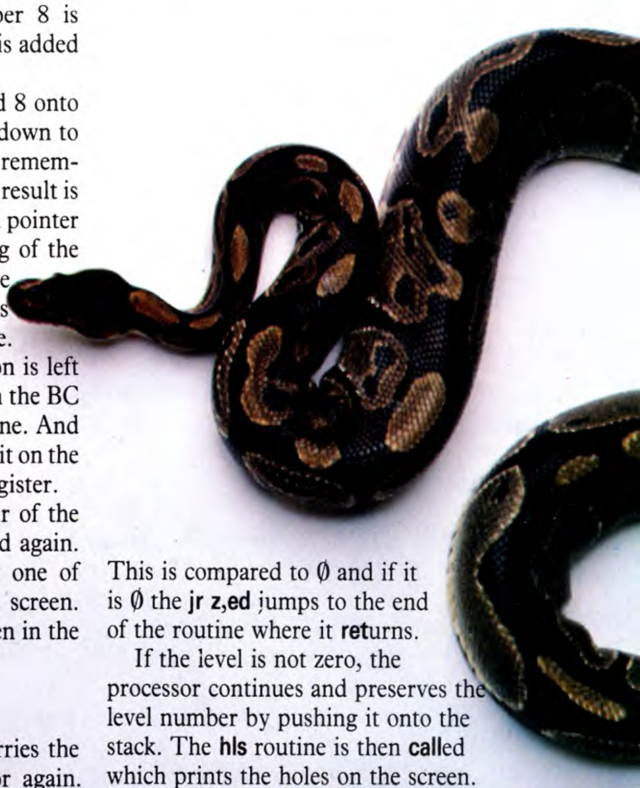
Loading A with 58 sets the colour of the reward and the **print** routine is **call**ed again. This was the routine given in part one of Cliffhanger that prints data on the screen. Here it prints the reward on the screen in the appropriate colour.

ON THE LEVEL

The contents of the location that carries the level is loaded into the accumulator again. This is compared to Ø and if it is Ø the **jr z,ed** jumps to the end of the routine where it **ret**urns.

If the level is not zero, the processor continues and preserves the level number by pushing it onto the stack. The **hls** routine is then **call**ed which prints the holes on the screen.

The 'CLIFFHANGER' listings published in this magazine and subsequent parts bear absolutely no resemblance to, and are in no way associated with, the computer game called 'CLIFF HANGER' released for the Commodore 64 and published by New Generation Software Limited.



The level is then recalled by **pop**ping it off the stack and compared to 1. If you are on level 1 there are potholes but no snakes. And if 1 is in the accumulator **cp 1** gives the result zero and sets the zero flag, so **jr z,ed** jumps to the end of the routine again and returns.

If not, the **snp** routine is **call**ed. This is the one that prints the snakes. Then the program returns. When the main driver program is added later, the processor will return there. But for now it will return to BASIC.

WHOLLY HOLES

The next subroutine is wholly devoted to making holes. This begins with the label **hls** which is called by the main routine when holes need to be dug.

It consists of three little modules each two instructions long. Each module begins with a **ld,hl** instruction. This loads the HL register with the print position of the top of each hole. Then the **hlp** routine is **call**ed. This actually prints the hole on the screen, over the slope generated in the last part of Cliffhanger.

There are three modules because there are three holes, each with its own print position, which is loaded into HL afresh each time, but each is created in exactly the same way by the **hlp** routine.

DIGGING THE HOLES

B is loaded with 4—the hole is to be four character squares deep. This is stored by **push**ing it onto the stack. The BC register pair is then loaded with the location of an empty space. This is, in fact, part of the data to the top of the screen.

A is loaded with 45 to give the sky colour and the **print** routine is **call**ed yet again. This prints the first square of the sky colour over the slope, effectively taking the first spadeful of soil out of the hole.

Loading DE with 32 and adding it to HL, moves the HL screen pointer down one line. The pointer is then **pop**ped off the stack again and the **djnz** decrements and jumps back if it hasn't become zero. So the processor goes round this loop four times, each time printing one more square of sky, taking one more spadeful of soil out of the hole.

THE SNAKE ADDER

The rest of the routine adds the snakes. The first seven instructions are like those at the beginning of the hole-printing routine. They load the HL register with the print position of the snake, then the subroutine that actually prints the snake is called.

Again this is like the hole-print routine—after all, the snakes fit in the holes.

This time the counter is kept in the accumulator because you need to keep track of the data pointer. Before, with the holes, the data pointer was loaded up again each time with the address of a space. But for the snake, the data pointer has to count along the snake data in the data table. The **print** routine automatically updates the pointer, moving it onto the next byte of data. And as the data pointer is held in the BC register you don't want to keep on having to store its value—on the stack or elsewhere. Swapping between them, especially when using the A register only adds one extra instruction. The **djnz** instruction in the hole-digging routine works on the B register, so if you are using a counter in A it must be replaced with **dec a** and **jr nz,snr**.

TESTING

To test the routine POKE 57344 with the level number Ø to 3 and call the routine each time with a RANDOMIZE USR 58455 to check that you are getting the holes on level 1, the snakes on levels 2 and 3 and a different reward each time.

The following routine determines which sprites appear at which level and initializes the start position of the man and the boulder. And it sets up the sea.

```
         ORG    24576
         JMP    FF
GG       LDA    $CØØØ
         CMP    #1
         BNE    AA
         LDA    #71
         STA    $DØ15
         RTS
AA       CMP    #2
         BNE    BB
         LDA    #125
         STA    $DØ15
RET      LDX    #3
         LDY    #Ø
LOOP     LDA    #15
         STA    $DØ2A,Y
         LDA    #236
         STA    $Ø7FB,Y
         INY
         DEX
         BNE    LOOP
         RTS
BB       CMP    #3
         BNE    CC
         LDA    #127
         STA    $DØ15
         JMP    RET
CC       CMP    #4
         BNE    DD
         LDA    #125
EE       STA    $DØ15
         LDX    #3
         LDY    #Ø
LOOPA    LDA    #5
         STA    $DØ2A,Y
         LDA    #234
         STA    $Ø7FB,Y
         INY
         DEX
         BNE    LOOPA
         RTS
DD       LDA    #127
         JMP    EE
FF       LDA    #7
         STA    $CØØE
         LDA    #232
```

```
STA     $C00D
LDA     $C002
STA     $C002
STA     $C00C
LDA     #2
STA     $C012
LDA     #33
STA     $C011
LDA     #18
STA     $D000
LDA     #161
STA     $D001
LDA     #66
STA     $D010
LDA     #74
STA     $D00D
LDA     #64
STA     $D00C
LDA     $D01E
LDA     $D01F
JMP     GG
```

JUMP ABOUT

This routine starts with a jump to a label halfway through the routine itself. And the processor only jumps back to the beginning again when it has completed the second half. While this is not the best way to write programs, it is a useful device to switch two bits of programming round without rewriting all the source code.

The first part of the program determines which sprites are used at which level. The second half deals with the sea.

WHICH SPRITE?

During the course of writing the game a table of variables is constructed. This starts at 49,152 and is used for temporary storage of the parameters that change during the game.

The variable stored in 49,152 itself specifies the level that the player has reached and thus determines which screen and which sprites are required.

The contents of 49,152 are loaded into A and compared with 1. If the game is not on level one, the BNE instruction branches forward over the next subroutine. If it is on level one, A is loaded with the number 71 which is stored in 53,269. This is the sprite display enable location. The sprites are switched on by a 1 in the bit pattern. So the sprites zero, one, two and six are switched on. Then the processor returns.

The next subroutine starts off in the same way, only this time it checks to see if the game is on level two. If it is, sprites three, four and five are switched on, too, but sprite two—which corresponds to the boulder—is switched off.

Then X is loaded with 3 as a loop counter, and Y is loaded with Ø as an offset which will be incremented each time the processor goes round the loop. So 15—the background colour, is loaded into the colour locations for sprites three, four and five. The corresponding sprite pointers are then set. The subroutine digs the holes.

The next subroutine checks for level three. If the player is on level three, the boulder sprite is switched back on—an extra two is added and stored in the sprite enable location. But the holes still have to be dug so the processor jumps back to the RET routine above.

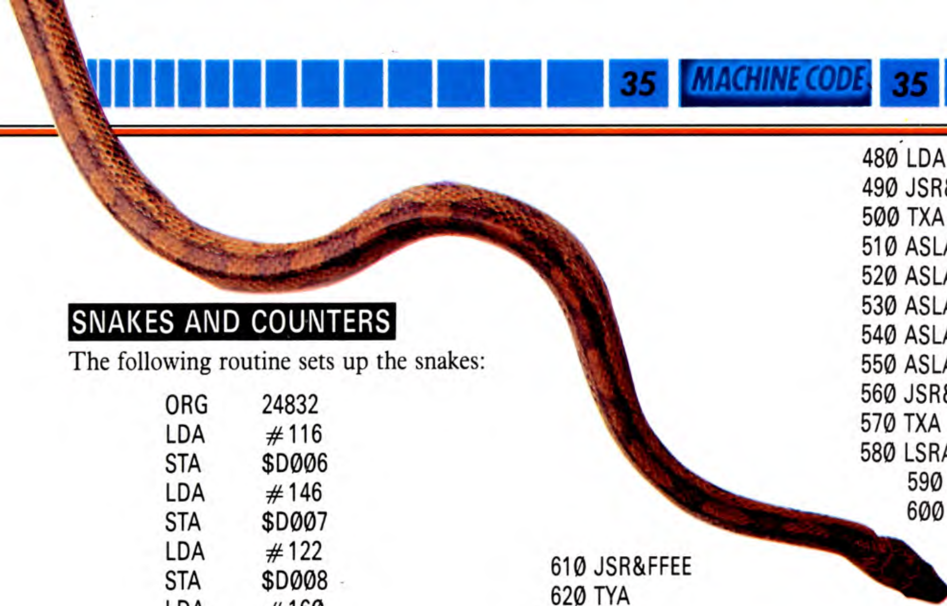
On level three, the boulder sprite is turned back off again and the new loop—LOOPA—is performed. This is the same as the hole digging routine above, only the colour is set to 5—the snake colour—and the sprite pointer is directed by the snake data instead.

And on level five, the boulder sprite is switched back on again and the snake-print routine is called.

SEA SET

The variables in 49,165 and 49,166 determine the state of the sea. The delay that determines how quickly the sea rises is in 49,154—this can be varied during the game to make Willie's scramble more desperate and the game harder.

The X and Y positions of the sprites which make up the sea are then set and the sprite collision flags are cleared.



SNAKES AND COUNTERS

The following routine sets up the snakes:

```
        ORG     24832
        LDA     #116
        STA     $DØØ6
        LDA     #146
        STA     $DØØ7
        LDA     #122
        STA     $DØØ8
        LDA     #16Ø
        STA     $DØØ9
        LDA     #13Ø
        STA     $DØØA
        LDX     #3
        LDY     #Ø
        LDA     #2Ø
LOOP    STA     $C364,Y
        SBC     #5
        INY
        DEX
        BNE     LOOP
        RTS
```

The X and Y coordinates of the three holes and snakes are loaded into the accumulator and stored in memory locations 53,254 to 53,259. These are the locations which control the X and Y positions of sprites three, four and five.

The loop staggers the tongues so that they don't all flick out at the same time.

```
3Ø FORPASS=ØTO3STEP3
4Ø RESTORE
1ØØ DATA4,16,9,13,16,11
11Ø FORA%=&1928TO&192D:READ?A%:
   NEXT
16Ø DATA17,8,32,8,1Ø,32,8,1Ø,32,8,1Ø,168,8,
   1Ø,169
17Ø FORA%=&192ETO&193C:READ?A%:
   NEXT
22Ø P%=&193D
23Ø [OPTPASS
24Ø .Holes
25Ø LDY#Ø
26Ø .Lb1
27Ø LDA#31
28Ø JSR&FFEE
29Ø LDA&1928,Y
3ØØ JSR&FFEE
31Ø LDA&1929,Y
32Ø JSR&FFEE
33Ø INY
34Ø INY
35Ø LDX#Ø
36Ø .Lb2
37Ø LDA&192E,X
38Ø JSR&18Ø3
39Ø INX
4ØØ CPX#15
41Ø BNELb2
42Ø CPY#6
43Ø BNELb1
44Ø RTS
45Ø .Tab
46Ø LDA#25
47Ø JSR&FFEE
48Ø LDA#4
49Ø JSR&FFEE
5ØØ TXA
51Ø ASLA
52Ø ASLA
53Ø ASLA
54Ø ASLA
55Ø ASLA
56Ø JSR&FFEE
57Ø TXA
58Ø LSRA
59Ø LSRA
6ØØ LSRA
61Ø JSR&FFEE
62Ø TYA
63Ø ASLA
64Ø ASLA
65Ø ASLA
66Ø ASLA
67Ø SEC
68Ø SBC #4
69Ø PHP
7ØØ JSR&FFEE
71Ø TYA
72Ø LSRA
73Ø LSRA
74Ø LSRA
75Ø LSRA
76Ø PLP
77Ø SBC#Ø
78Ø JSR&FFEE
79Ø RTS
8ØØ ]
85Ø DATA5,18,Ø,4,141
86Ø DATA8,1Ø,17Ø,8,1Ø
87Ø DATA171,8,1Ø,172,8
88Ø DATA1Ø,173,8,11,11
89Ø DATA11,18,Ø,2,174
9ØØ DATA8,1Ø,175,8,1Ø
91Ø DATA176,8,1Ø,177,4
92Ø FORA%=&1996TO&19B8:READ?A%:
   NEXT
97Ø P%=&19B9
98Ø [OPTPASS
99Ø .Snakes
1ØØØ JSRHoles
1Ø1Ø LDY#Ø
1Ø2Ø .Lb3
1Ø3Ø TYA
1Ø4Ø PHA
1Ø5Ø LDA&1928,Y
1Ø6Ø ASLA
1Ø7Ø TAX
1Ø8Ø LDA#32
1Ø9Ø SEC
11ØØ SBC&1929,Y
111Ø ASLA
112Ø TAY
113Ø JSRTab
114Ø PLA
115Ø TAY
116Ø LDX#Ø
117Ø .Lb4
118Ø LDA&1996,X
119Ø JSR&18Ø3,X
12ØØ INX
121Ø CPX#35
122Ø BNELb4
123Ø INY
124Ø INY
125Ø CPY#6
126Ø BNELb3
127Ø RTS
128Ø ]NEXT
```

DIGGING THE DATA

The first block of DATA in Line 1ØØ gives the position of the holes and snakes. Line 11Ø reads this into a data table where the machine code program can access it.

And the next section of DATA in Line 16Ø contains the details of the potholes themselves.

PICK OF THE POTS

To print anything on the screen you have to move the cursor into position first. This is done by the instructions in Lines 27Ø to 32Ø. As before, loading the accumulator with 31 and jumping to the subroutine at FFEE gives the equivalent of a VDU 31. That routine is then primed to accept two more parameters—the first it will take to be the X coordinate of the proposed cursor position and the second will be taken as the Y coordinate.

Indirect indexed addressing is used to pick up the appropriate values of the coordinates from the data table constructed by the BASIC program in Lines 1ØØ and 11Ø. The base addresses used are those of the first and second byte of the data table. The offset Y is set to Ø by Line 25Ø on the first pass. With the loop, opened by the label .Lb1 in Line 26Ø, it is incremented by the two INYs in Lines 33Ø and 34Ø to pick up the next two coordinates for the next hole.

IN THE HOLE

This time X is used as the index register and is initialized to Ø in Line 35Ø. Line 37Ø picks up the bytes of the data table that refer to the shape of the hole. These are output to the screen by jumping to the subroutine which starts at &18Ø3. This is the UDG print routine given in the last part of Cliffhanger—the one that prints out a character from the character set if its number is less than 128 and a UDG if the number is greater than 127. X is then incremented and compared to 15. The processor branches back to output the next byte of data, if the end of the hole data has not been reached.

If you think that this is the only part of the program that does not call &FFEE, you'd be wrong. The subroutine at &18Ø3 uses the &FFEE routine to output to the screen once it has decided whether an ASCII character or a UDG is required. That said, it is clear that the data in the hole data table is designed to drive the &FFEE routine in the normal way.

The leading 17 gives a VDU 17—or COLOUR—command when output through &FFEE. So the 8 following it gives colour 8. This has been redefined as logical colour 6 in the last part of Cliffhanger. This gives foreground cyan—you are going to overwrite the green slope foreground with the background cyan colour.

The next byte is 32, ASCII for a space. Then the 8 moves the cursor back onto the space that has just been printed and the 10 moves it down to the character square below. This is done three times to print three squares. Then UDG 168 is printed out, the cursor is moved back and down again, and UDG 169 is printed. These two UDGs shape the bottom of the hole.

MOVING

So far you have been shifting the text cursor around the screen before you print. But in some circumstances wou will need to move the graphics cursor too. The routine to do this is Lines 450 to 780.

This routine is almost exactly the same as the DRAW routine given in part one of Cliffhanger. Here, though, after 25 has been output to the routine at &FFEE to give a PLOT, a 4 is output there too. This gives a PLOT 4, or a MOVE. But the X and Y coordinates are encoded into and decoded out of a single data byte in exactly the same way using repeated arithmetic shifts left and logical shifts right.

The SBC#4 in Line 680 simply MOVEs down one extra pixel. The PHP pushes the process register—that is, the flags—onto the stack, the register is restored after the high byte of the coordinate has been decoded and zero is subtracted using an SBC#0. Notice that this is a subtract with carry and the carry flag is in the same condition as it was immediately after the SBC#4 in Line 680. This adjusts the high byte if subtracting the 4 has taken the low byte through zero.

SNAKE BYTES

Lines 850 to 910 carry the data for the snake and Line 920 READs it into a data table.

The instruction in Line 1000 calls the routine that prints the holes starting in Line 240 above. Then the index register Y is initialized to 0, the value of Y is transferred into A and A is pushed onto the stack to save it.

The snake/hole position is then loaded up from the data table constructed by Lines 100 and 110. The position in that data table are encoded for the text screen, so they have to be re-evaluated for the graphics screen. To give the X coordinate, the byte from the data table is arithmetically shifted left—to multiply it by two—and transferred into the X register.

Then the next data byte is subtracted from 32, multiplied by two by an arithmetic shift left and transferred into the Y register. The tab routine is then called. This is the routine given above which MOVEs the graphics cursor into the appropriate position.

The loop counter is then pulled back off the stack and put into the Y register and X is initialized to 0 before the next loop is entered. Then the appropriate byte of the snake table is loaded up and the UDG print routine at &1803 is called. X is incremented and compared to 35, and the processor branches back until all 35 bytes of snake have been output.

Y is incremented twice to move the routine along to the next two bytes of the snake/hole position data table. It is then compared to 6 and the processor branches back until all 6 bytes of position data have been used and the three snakes have been printed in the three holes.

At the moment these snakes do not move. You will get them wiggling in a later part.

TESTING

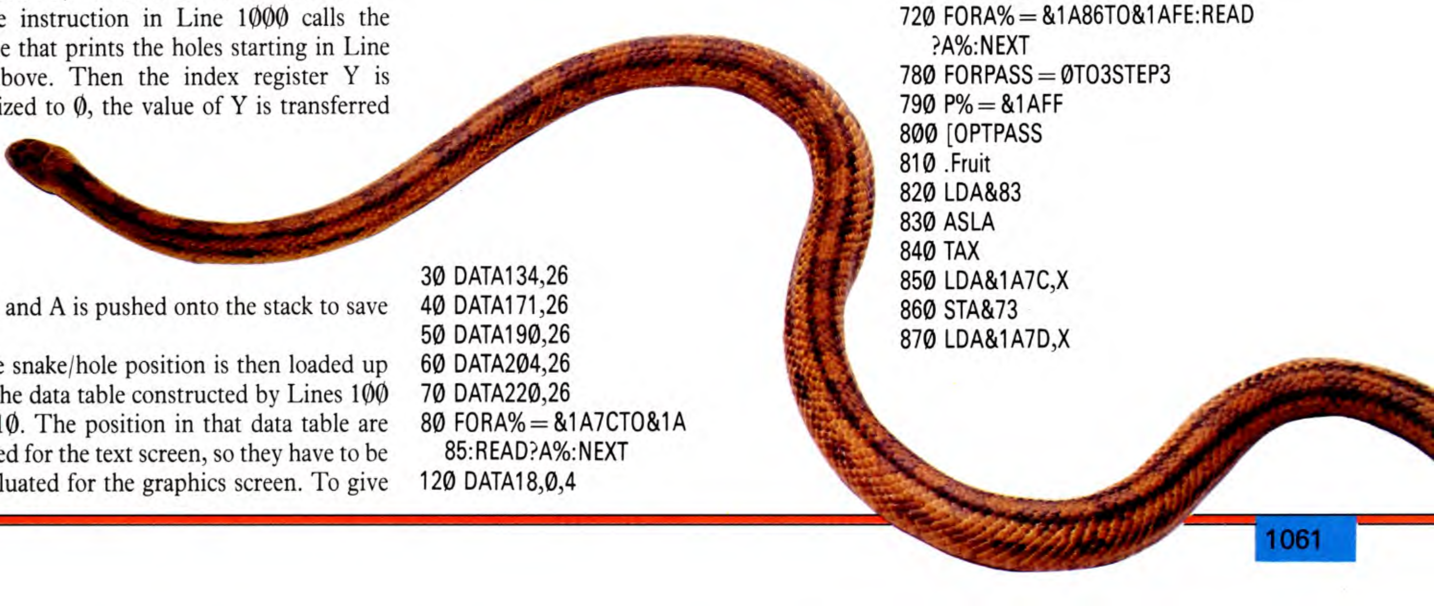
As routine given in former parts of Cliffhanger are called from the routines given here, you must have them in memory before you CALL any part of this program. Then key in the following:

```
PAGE=&2000
NEW
MODE2:CALL&182D:CALL&1855:CALL&1894:
   CALL&19B9
```

THE GOODIES

The following program prints up the items of Willies picnic on the screen. Don't forget to type PAGE=&3000 and NEW before you key it in:

```
30 DATA134,26
40 DATA171,26
50 DATA190,26
60 DATA204,26
70 DATA220,26
80 FORA%=&1A7CTO&1A
   85:READ?A%:NEXT
120 DATA18,0,4
130 DATA178,178,10
140 DATA8,8,179
150 DATA179,11,8
160 DATA8,18,0
170 DATA8,180,181
180 DATA10,8,8
190 DATA182,183,11
200 DATA8,8,18
210 DATA0,2,184
220 DATA185,10,8
230 DATA8,186,187
240 DATA255
280 DATA18,0,1
290 DATA188,188,10
300 DATA8,8,189
310 DATA190,11,8
320 DATA8,18,0
330 DATA3,191,192
340 DATA255
380 DATA18,0,3
390 DATA9,193,8
400 DATA10,194,8
410 DATA18,0,1
420 DATA195
430 DATA255
470 DATA18,0,4
480 DATA196,197,8
490 DATA8,18,0
500 DATA2,198,199
510 DATA8,10,200
520 DATA255
560 DATA18,0,4
570 DATA201,202,8
580 DATA8,10,203
590 DATA8,11,18
600 DATA0,1,204
610 DATA205,8,8
620 DATA10,206,207
630 DATA8,8,11
640 DATA18,0,8
650 DATA208,209,8
660 DATA8,10,210
670 DATA211
680 DATA255
720 FORA%=&1A86TO&1AFE:READ
   ?A%:NEXT
780 FORPASS=0TO3STEP3
790 P%=&1AFF
800 [OPTPASS
810 .Fruit
820 LDA&83
830 ASLA
840 TAX
850 LDA&1A7C,X
860 STA&73
870 LDA&1A7D,X
```



```
880 STA&74
890 LDX#36
900 LDY#60
910 JSR&1964
920 LDA#5
930 JSR&FFEE
940 LDY#0
950 .Lb1
960 LDA(&73),Y
970 JSR&1803
980 INY
990 LDA(&73),Y
1000 CMP#&FF
1010 BNELb1
1020 LDA#4
1030 JSR&FFEE
1040 RTS
1050 ]NEXT
```

To run it, you'll need the other programs given so far in memory, then key in:

```
PAGE=&2000
NEW
MODE2:CALL&182D:CALL&1855:CALL&1894:
   CALL&19B9
```

Then POKE in the level number using ?&83= followed by a number between 0 and 4. Then:

```
CALL&1AFF
```

PICNIC DATA

The DATA in Lines 30 to 70 contain the pointers to blocks of data which define each of Willie's missing picnic goodies. The DATA for the sandwich is in Lines 120 to 230. The DATA for the apple is in Lines 380 to 330. The DATA for the lemonade is in Lines 380 to 420. The DATA for the ice-cream is in Lines 470 to 510. And the DATA for the cake is in Lines 560 to 670. Each block of data ends with 255, so that the processor can recognise the end.

Each of these goodies appears on a different screen. The sandwich appears on screen one, the apple on screen two, the lemonade on screen three, the ice-cream on screen four and the cake on screen five.

SCREENING THE GOODIES

Zero-page memory location &83 is used to store the number of the level the game has reached, and thus the screen that is required. The LDA&83 in Line 820 loads the level number into the accumulator and the ASLA in Line 830 multiplies it by two—the DATA pointers take up two bytes, so you have to count along the data pointer table two at a time.

This number is then used as an index, so it is transferred into the X register and used as an offset in the indexed instructions that load up the data pointers in Lines 850 and 870. The two bytes of the appropriate data pointer are stored in &73 and &74.

The X and Y coordinates of the position the goodies are going to be printed in are loaded into the X and Y registers. Then the processor jumps to the tab subroutine given in the first program in this part of Cliffhanger which positions the graphics cursor.

Five is loaded into the accumulator and the &FFEE routine is called. This directs all printing to the graphics cursor instead of the text cursor.

The Y index register is set to 0 and the first byte of data for the appropriate goody is loaded up by using the indirect indexed instruction LDA(&73),Y. The pointer in &73 and &74 points to the beginning of the block of data for the appropriate goody, remember. Then the processor jumps to the UDG print routine at &1803 to print out the first part of the goody on the screen.

The Y index is then incremented and the next byte is loaded up. This is compared to &FF—or 255—to see if the end of that block of data has been reached. If it hasn't the processor branches back and prints out the next byte. If it has, the processor drops out of the loop and proceeds to the next instruction.

The accumulator is loaded with 4 and the &FFEE routine is called again. This directs any subsequent screen output to the text cursor.

The following prints the holes and the snakes on the screen for the Dragon and the Tandy when you have progressed to the appropriate level:

```
        ORG     19289
ELB     LDA     18238
        LDB     #16
        MUL
        ADDD    #18142
        TFR     D,U
        LDX     #2782
        JSR     CHARPR
        LDA     18238
        BEQ     ED
        PSHS    A
        JSR     HOLES
        PULS    A
        CMPA    #1
        BEQ     ED
        JSR     SNAKE
ED      RTS
HOLES   LDX     #5095
        LDU     #3071
        JSR     HOLPR
        LDX     #4591
        LDU     #3071
        JSR     HOLPR
        LDX     #3833
        LDU     #3071
        JSR     HOLPR
        RTS
SNAKE   LDX     #5095
        LDU     #18078
        JSR     HOLPR
        LDX     #4591
        LDU     #18078
        JSR     HOLPR
        LDX     #3833
        LDU     #18078
        JSR     HOLPR
        RTS
HOLPR   LDB     #4
HOLPRI  PSHS    U,X,B
        JSR     CHARPR
        PULS    B,X,U
        LEAX    256,X
        LEAU    16,U
        DECB
        BNE     HOLPRI
        RTS
CHARPR  LDB     #2
CHARI   PSHS    B,X
        LDB     #8
CHARZ   PULU    A
        STA     ,X
        LEAX    32,X
        DECB
        BNE     CHARZ
        PULS    X,B
        LEAX    1,X
        DECB
        BNE     CHARI
        RTS
```

TO TEST THIS PROGRAM

As this program adds graphics to the scenery you've already got you'll need the rest of the game in memory before you execute it. Then RUN the following BASIC program to test it.

```
5 PCLEAR4: CLEAR 2ØØ,16999
1Ø EXEC 19ØØØ: EXEC191Ø9
2Ø POKE 18238,Ø
3Ø EXEC 19289
4Ø GOTO 4Ø
```

Line 1Ø executes the bit of the game you've keyed in from previous issues of Cliffhanger. Line 2Ø sets the level to Ø and Line 3Ø executes the new part of Cliffhanger given above. Line 4Ø is just an infinite loop to hold the display on the screen.

To make sure this new part of the program is working properly, you'll have to BREAK this program and edit Line 2Ø, POKEing new levels into memory location 18,238. The Ø here will just give you the slope—the obstacle on level one of Cliffhanger, the boulders, is given later. A one will give you the potholes. A two will give you the holes and the snakes. Level three also features boulders for extra difficulty, but not yet.

Try testing all four levels, though, to make sure that you have got all Willie's picnic things drawn properly.

ON THE LEVEL

Memory location 18,238 is going to be used to store the level of difficulty you're on. Thus its contents are loaded into the accumulator. 16 is put into the B register and the instruction MUL multiplies the contents of those two registers together. The result is put in D.

Each reward is made up of 16 bytes of data, so to count along the data table to the start of the appropriate reward for the level attained, you have to multiply the level number by 16.

The number 18,142 is added to the result in the D register, so that its contents now point to the start address of the data you want. This pointer is then transferred into U, the user stack pointer. This effectively turns the appropriate section of data into the user stack.

The X register is then loaded with 2,814, which is the screen position where you want the reward printed. And the processor jumps to CHARPR which prints the reward.

LDA 18238 loads the level number into the accumulator again. The processor then branches to the label ED, which marks the RTS at the end of the program, then the contents of 18238 are zero. On level one—or level Ø in computer parlance—you needn't go any further with this routine.

If you're on level two, though, the processor pushes the level number onto the hardware stack to preserve it. Then it jumps to the subroutine which digs the holes.

The level number is pulled back off the hardware stack into the accumulator and compared with one. If it is one—and you're on level two—the JSR SNAKE instruction, which is the routine that prints the snakes, is skipped and the processor returns.

HOLES AND SNAKES

The hole routine and the snake routine work exactly the same. They each have three cycles of three instructions.

In each cycle the X register is loaded with the screen position of the top of the hole and the U register is loaded with the address of the beginning of the appropriate data. Then the processor jumps to the HOLPR subroutine.

Obviously all the holes are the same, so the data pointer is always initialized at 3,Ø71. And all the snakes—though the graphic printed is actually a snake in a hole—are one snake in the data table and start at 18,Ø78.

And obviously the snakes in their holes have to be printed at the same place at the holes they replace. The top of first hole/snake-hole is at 5,Ø95, the second is at 4,591 and the third is at 3,Ø71.

Both snakes and snake-holes are printed over the original slope drawn in an earlier part of Cliffhanger. This means that the rest of the screen can be left as it is.

DIGGING IT

Both the HOLES and the SNAKES routines call the HOLPR routine. And it is this routine that digs out the potholes and the snake-pits, spadeful by spadeful from the top.

B is loaded with four—the holes are going to be four character squares deep. U, X and B are pushed onto the stack to preserve them while the processor jumps to the CHARPR subroutine. This is the most basic routine of all. It just prints the character pointed to by the data pointer onto the screen in the appropriate place.

So if you are on level two and are printing holes, the data pointer in U will point to the empty hole data and the routine will print a block of the background colour at the screen position in X. If you are at level three or four, U points to the snake-pit data and this is printed as a block at the screen position in X.

Once the character block has been printed B, X and U are pulled off the stack again, then X is loaded with X + 256 which moves it onto the beginning of the next character block and U is loaded with U + 16, which moves the data pointer onto the graphic. Then B, the spadeful counter, is decremented and the processor loops back if B hasn't counted down to zero. If it has and all three spadefuls have been dug out, the processor returns.

Each character block that is printed on the screen is 16 bits—that is, two bytes wide—and eight bytes deep. So the B counter is loaded with 2 to count across the hole. This is saved by pushing it onto the hardware stack along with the print position in X. B is then reloaded with eight to count down the hole.

The data is then retrieved by pulling it off the user stack into the accumulator. And this is stored at the position pointed to by the X register. This actually prints the hole or snake-pit data on the screen.

X is then reloaded with X + 32, to move the screen pointer down one line of pixels on the screen. The B counter is then decremented and the processor branches back to print the next byte of data on the screen until the B counter has counted down from eight to zero.

X and B are then pulled of the stack again. X is loaded with X + 1 which moves the screen pointer onto the next character square to the right and B is decremented. If it hasn't counted down from two to zero, it branches back to dig out the next column of the hole or snake-pit. And when both columns have been dug, the processor returns.

PLANNING THE FUTURE

Here is the final part of the calendar and diary program. It adds the printout routines and lets you SAVE and LOAD the diary lists either to tape or to disk

If you SAVEd the last two parts of the program LOAD them back in now and enter the remaining lines given here. You'll then have a complete program and can start to use it to keep track of future appointments and events.

The instructions for using the program were given with the last two parts so look back at them to see what to do. Extra instructions for SAVEing and LOADing the data are given below.

The program is actually very straightforward to use. The main menu lets you choose exactly what you want the program to do, and each option is well-prompted so you know what to enter. The entries are error-trapped so you needn't worry if you accidentally type in the wrong type of data—the program will simply ask you to enter it again.

SAVING THE LISTS

When you've entered all of the data you should save the lists using option 4 on the Commodore or option 8 on the others. The program is designed to save the lists to tape. Changes to the program to allow it to work with a disk drive are given separately after the main program.

Next time you use the program answer Y to the question 'have you any existing lists' and the data will be loaded back in again. It can then be altered, deleted and updated, viewed or printed out, and then the new list saved. The data is saved in a file called 'DIARY' so save the main program under a different name, CALENDAR, say.

```
1760 LET M4=Ø: LET A4=Ø
1770 INPUT "YEAR:";YR: IF YR<1753 OR
     YR>29999 THEN GOTO 1770
1780 GOSUB 640
1790 GOSUB 2480
1800 CLS
1810 POKE 23692,255
1820 PRINT #P;"YEAR□";YR
1830 PRINT #P: LET KB=Ø: GOSUB 1920:
     PRINT #P
1840 GOSUB 2460
1850 FOR z= 1 to 12
1860 LET MO=z
1870 PRINT #P;M$(MO*9-8 TO MO*9)
1880 LET T2=5: LET S2=Ø: GOSUB 2020
1890 IF P=2 THEN IF INKEY$="" THEN
     GOTO 1890
1900 NEXT z
1910 RETURN
1920 LET X2=Ø: LET C2=Ø: LET D2=Ø
1940 IF P=3 THEN LET KB=KB+1
1950 PRINT #P;Z$( TO X2);
1960 FOR d=1 TO 7
1970 INK 4: IF d=1 THEN INK 2
1980 PRINT #P;Z$( TO KB);s$((d-1)
     *3+1 TO (d-1)*3+3);
1990 NEXT d
2000 INK 7
2010 RETURN
2020 PRINT
2030 IF P=3 THEN PRINT AT 10,4; FLASH
     1;"OUTPUT GOING TO PRINTER"
2040 LET M5=Ø: LET XP=Ø: LET X2=Ø:
     LET W2=Ø: LET A$="": LET D$=""
2050 IF S2=1 THEN LET A$="□": LET
     W2=4
2060 IF S2=Ø THEN LET X2=7: LET W2=3
2080 LET DA=1
2090 LET KB=MO: GOSUB 270: LET
     M5=KB
2100 GOSUB 560: LET K2=7: LET XP=FN
     M(KB)
2110 PRINT #P;Z$( TO XP*W2);
2120 LET DA=Ø
2130 PRINT #P;Z$( TO X2);
2140 LET DA=DA+1: LET D$=A$+(STR$
     (DA))+"□": IF LEN D$<W2 THEN LET
     D$=D$+Z$( TO W2-LEN D$)
2150 IF A$="" THEN PRINT #P;D$;: GOTO
     2170
2160 LET KB=T2: GOSUB 350: PRINT #P;
     INK KB;D$;
2170 LET XP=XP+1
2180 IF NOT (XP>6 OR DA=M5) THEN
     GOTO 2140
2190 LET XP=Ø: PRINT #P: IF S2=1 THEN
     PRINT #P
2200 IF DA<>M5 THEN GOTO 2130
2210 IF MO=ME THEN PRINT #P;: PRINT
     #P;"Easter Sunday□";M$(ME*9-8
     TO ME*9);DE
2220 IF P=3 THEN PRINT AT 10,Ø;z$: PRINT
     AT 10,13;"READY"
2230 RETURN
2240 GOSUB 2510
2250 LET T2=Ø: LET MX=Ø: LET N2=Ø:
     LET A$="": LET CL=Ø: LET M9=MO:
```

```
   LET Y9=YR
2260 GOSUB 2480: CLS
2270 GOSUB 2570: PRINT #P
2280 PRINT #P;"DAY□□□□□□ENTRY":
   PRINT AT 0,18;"Any key for"'TAB 18;"
   next entry"
2290 PRINT #P
2300 GOSUB 2460
2310 IF MO=ME THEN PRINT #P;INK 5;
   DE;"□□Easter Sunday"
2320 PRINT #P
2330 FOR t=1 TO 4
2340 LET MX=Q(t)
2350 IF MX=0 THEN GOTO 2410
2360 FOR N=1 TO MX
```

```
2370 LET K$=L$(t, N)
2380 LET KB=t
2390 LET K2=3: GOSUB 470: IF K2=1
   THEN PRINT #P;INKZ(t); K$(2 TO 3);
   "□□□□□"; K$(10 TO )
2400 NEXT N
2410 IF INKEY$="" THEN GOTO 2410
2420 NEXT t
2430 FOR I=1 TO 100: NEXT I
2440 IF INKEY$="" THEN GOTO 2440
2450 LET MO=M9: LET YR=Y9: RETURN
2460 IF INKEY$="" THEN GOTO 2460
2470 RETURN
2480 PRINT : PRINT "WOULD YOU LIKE A
   PRINTOUT (Y/N)?": LET K$="yn":
   GOSUB 1480
2490 LET P=2: IF KB=1 THEN LET P=3
2500 RETURN
2510 INPUT "MONTH ?";MO
2520 IF MO<1 OR MO>12 THEN GOTO
   2510
2530 INPUT "YEAR ?";YR
2540 IF YR<1735 OR YR>29999 THEN
   GOTO 2530
2550 GOSUB 640
2560 RETURN
2570 PRINT #P; PAPER 1; INK 7;
   M$(MO*9-8 TO MO*9);"□";YR
2580 RETURN
```

```
1740 FL=0
1750 FOR N1=1 TO 4:CD(N1)=ASC
   (MID$(DL$(C,LX),N1,1))
1760 NEXT N1:CD(4)=CD(4)+(ASC
   (MID$(DL$(C,LX),5,1))*256)
1770 IF CD(4)>YY THEN FL=0: RETURN
1780 IF CD(1)=1 AND M>=CD(3) THEN
   FL=1:RETURN
1790 IF CD(1)=2 AND M>=CD(3) AND
   M-CD(3)-INT((M-CD(3))/3)
   *3=0 THEN FL=1:RETURN
1800 IF CD(1)=3 AND CD(3)=M THEN
   FL=1:RETURN
1810 IF CD(1)=4 AND CD(3)=M AND
   CD(4)=YY THEN FL=1
1820 RETURN
1830 IF K=6 THEN M=M-1:RETURN
1840 IF K=5 THEN M=M+1:RETURN
1850 IF Z=1 THEN Z=0:RETURN
1860 IF Z=0 THEN Z=1:RETURN
1870 IF PP=0 THEN PRINT"■"
1880 PRINT"□DIARY FOR□";
   MID$(MN$,M*9-8,9):PRINT
1885 IF M<>EM THEN 1910
1890 PRINT"□DATE
   □□:";STR$(ED)
1900 PRINT"□EVENT□:□EASTER
   SUNDAY■
1910 ML=VAL(MID$(ML$,2*M-1,2)):
   IF M=2 THEN ML=ML+LY
1920 C=CP:IFC>3THENC=0
1930 PRINTTAB(20-(LEN(ST$(C))
   *.5))"■"CHR$(CL(C));ST$(C)
1940 MX=VAL(DL$(C,0))
1950 IF MX=0 GOTO 1990
1960 FOR CD=1 TO ML:FOR LX=1 TO
   MX:CD(2)=ASC(MID$(DL$(C,LX),
   2,1))
1970 IF CD=CD(2) THEN GOSUB 2010
1980 NEXT LX,CD
1990 RETURN
2010 GOSUB1740
2020 IF FL=0 THEN RETURN
2030 A$=STR$(CD):IF LEN(A$)<2 THEN
   A$=A$+"□"
2040 B$=DL$(C,LX)
2050 PRINTA$;CHR$(CL(C));
   RIGHT$(B$,LEN(B$)-4)
2060 RETURN
2070 PRINT"■■■■■■■
   HAVE YOU A DIARY LIST SAVED (Y/N)?
   ■■"
2080 PRINT"■■■■■□□□□
   □□□□□□□□□□□□□□□
   □□□□□□□□□□□□□
   □□□□■"
2090 GET A$:IF A$="" GOTO 2090
2100 IF A$="N" THEN RETURN
2110 IF A$="Y" GOTO 2130
2120 GOTO 2090
2130 GOSUB 3000:OPEN1,DV,0,NM$
2140 FOR C=0 TO 3:INPUT#1,
   DL$(C,0):MX=VAL(DL$(C,0)):
   IF MX=0 GOTO 2160
2150 FOR N=1 TO MX:INPUT
   #1,DL$(C,N):NEXT N
2160 NEXT C:CLOSE1:RETURN
2170 PRINT"■■□□UPDATED DIARY
   LIST TO BE SAVED (Y/N)?□"
2180 GET A$:IF A$="" GOTO 2180
2190 IF A$="N" THEN RETURN
2200 IF A$<>"Y" GOTO 2180
2210 GOSUB 3000:OPEN1,DV,1,NM$
2220 FOR C=0 TO 3
2230 MX=VAL(DL$(C,0))
2240 FOR N=0 TO MX
2250 PRINT #1,CHR$(34)+DL$(C,N)+
   CHR$(34)
2260 NEXT N,C:CLOSE 1:RETURN
2270 PP=1:OPEN 1,4:CMD1:RETURN
2280 IF PP=1 THEN PRINT#1,
   "□":CLOSE 1
2290 PP=0:RETURN
3000 NM$="": INPUT "FILE NAME";
   NM$:IFNM$="" THEN 3000
3010 DV=1:INPUT "(D)ISK OR (T)APE";
   DV$:IF DV$="D" THEN DV=8
3020 PRINTTAB(13)"■■■PRESS ANY
   KEY":POKE 198,0:WAIT 198,1:
   POKE 198,0
3030 PRINT "■":RETURN
```

```
2260 DEF PROCprintdays(s%)
2270 LOCALd,c%,x%
2280 IF s%=0 x%=7
2290 PRINTSPC(x%);
2300 FOR d=0 TO 6
2310 IF d=0 c%=129 ELSE c%=134
2320 IF P%=2 c%=32
2330 PRINTCHR$c%+STRING$(s%,
   "□")+MID$(DayName$,d*3+1,3);
2340 NEXT
2350 ENDPROC
2360 DEF PROCprintmonth(type%,s%)
2370 LOCALmax%,xpos%,x%,w%,
   a$,d$
2380 IF s%=1 a$="□":w%=4
2390 IF s%=0 x%=7:w%=3
2400 Day%=1
2410 max%=FNmonthL(Month%)
2420 xpos%=FNdayNo MOD7
2430 PRINTSPC(xpos%*(w%+1));
2440 Day%=0
2450 REPEAT
2460 PRINTSPC(x%);
2470 REPEAT
2480 Day%=Day%+1:d$=a$+STR$
   (Day%)+"□":IF LENd$<w%
d$=d$+"□"
2490 PRINTCHR$(FNmarker
   (type%));d$;
2500 xpos%=xpos%+1
2510 UNTIL xpos%>6 OR Day%=max%
2520 xpos%=0:PRINT:IFs%=1 PRINT
2530 UNTIL Day%=max%
2540 IF Month%=Meast%:PRINT'
   CHR$131;"Easter Sunday□";
   MID$(MonthName$,Meast%*
   9-8,9);Deast%
2550 ENDPROC
2560 DEF PROCdiary
2570 LOCALt%,max%,n%,a$,col%,
   month%,year%
2580 VDU 14
2590 PROCmydate:PROCprinter:CLS
2600 PROCmyheader:PRINT
2610 PRINT"DAY";CHR$134;
   SPC(5)"ENTRY"
2620 PROCspacebar
2630 IF Month%=Meast% PRINT;
   Deast%;CHR$134;"Easter Sunday":
   PRINT
2640 FOR t%=0 TO 3
2650 col%=129+t%:IF t%=3 col%=133
2660 IF t%=2 THEN col%=132
2670 IF P%=2 col%=32
2680 max%=VAL(List$(t%,0))
2690 IF max%=0 GOTO2740
2700 FOR n%=1 TO max%
2710 a$=List$(t%,n%)
2720 IF FNcheck(a$)=1 PRINTSTR$
```

```
    (ASC(MID$(a$,2,1)));"□□";
    CHR$col%;RIGHT$(a$,LENa$-4)
2730 NEXT
2740 PRINT:IF P%=Ø a$=GET$
2750 NEXT
2760 VDU3:a$=GET$
2770 VDU 15
2780 ENDPROC
2790 DEF PROCspacebar
2800 VDU3,31,4,24,132,157,135:
    PRINT"Any Key to continue
    □□";:VDU156,28,Ø,23,39,5,P%
2810 ENDPROC
2820 DEF PROCprinter
2830 PRINT'"Would you like a printout
    (Y/N)?":IF FNget("YN")=1 P%=2 ELSE
    P%=Ø
2840 ENDPROC
2850 DEF PROCmydate
2860 Month%=FNnoIn(1,12,"Month:")
2870 Year%=FNnoIn(1753,3299,
    "□Year:")
2880 PROCeaster
2890 ENDPROC
2900 DEF PROCmyheader
2910 PRINTF$;MID$(MonthName$,
    Month%*9-8,9);"□";Year%;
    TAB(3Ø)CHR$156
2920 VDUP%:PRINTF$;MID$(Month
    Name$,Month%*9-8,9);"□";
    Year%;TAB(3Ø)CHR$156
2930 ENDPROC
```

```
2240 'PRINTMONTH -T2 -S2
2250 M5=Ø:XP=Ø:X2=Ø:W2=Ø:
    A2$="":D2$=""
2260 IF S2=1 THEN A2$="□":W2=4
2270 IF S2=Ø THEN X2=7:W2=3
2280 IF P=2 THEN A2$=A2$+"□":
    W2=W2+1
2290 DA=1
2300 KB=MO:GOSUB 23Ø:M5=KB
2310 GOSUB 56Ø:K2=7:XP=FNM(KB)
2320 PRINT#-P,STRING$(XP*(W2),32);
2330 DA=Ø
2340 REM
2350 PRINT#-P,STRING$(X2,"□");
2360 REM
2370 DA=DA+1:D2$=A2$+MID$
    (STR$(DA),2)+"□":IF LEN (D2$)<W2
    THEN D2$=D2$+"□"
2380 IF A2$="" THEN PRINT#-P,D2$;:
    GOTO 24ØØ
2390 KB=T2:GOSUB 31Ø:MID$
    (D2$,1,1)=CHR$(KB):PRINT#-P,D2$;
2400 XP=XP+1
2410 IF NOT (XP>6 OR DA=M5) THEN
    236Ø
2420 XP=Ø:PRINT#-P:IF S2=1 THEN
    PRINT#-P
2430 IF DA<>M5 THEN 234Ø
2440 IF MO=ME THEN PRINT#-P:PRINT
    #-P,"EASTER SUNDAY□";MID$(MN$,
    ME*9-8,9);DE
2450 RETURN
2460 'DIARY ROUTINE
2470 T2=Ø:MX=Ø:N2=Ø:A$="":
    CL=Ø:M9=MO:Y9=YR
2480 GOSUB 275Ø:GOSUB 272Ø:CLS
2490 GOSUB 282Ø:PRINT#-P
2500 PRINT#-P,"DAY□□□□□
    ENTRY"
2510 GOSUB 266Ø
2520 IF MO=ME THEN PRINT#-P,DE;
    "EASTER SUNDAY":PRINT#-P
2530 FOR T2=Ø TO 3
2540 CL=159+16*T2
2550 IF P=2 THEN CL=32
2560 MX=VAL(LI$(T2,Ø))
2570 IF MX=Ø THEN 262Ø
2580 FOR N2=1 TO MX
2590 A$=LI$(T2,N2)
2600 KB$=A$:GOSUB 47Ø: IF K2=1 THEN
    PRINT#-P,MID$(STR$(ASC(MID$
    (A$,2,1))),2);TAB(3);CHR$(CL);RIGHT$
    (A$,LEN(A$)-4)
2610 NEXT: PRINT#-1
2620 IF P=Ø ANDINKEY$=""THEN 262Ø
2630 NEXT
2640 IF INKEY$=""THEN 264Ø
2650 MO=M9:YR=Y9:RETURN
2660 'WAIT FOR KEY
2670 P1=PEEK(136):P2=PEEK(137)
2680 PRINT@48Ø,"ANY KEY TO CONTINUE";
2690 IF INKEY$="" THEN 269Ø
2700 PRINT@48Ø,STRING$(3Ø,32);:
    POKE 136,P1:POKE 137,P2
2710 RETURN
2720 'OFFER PRINTOUT
2730 PRINT:PRINT"WOULD YOU LIKE A
    PRINTOUT (Y/N)":KB$="YN":
    GOSUB 159Ø:IF KB=1 THEN P=2
2740 RETURN
2750 'MY DATE ROUTINE
2760 INPUT "MONTH:";MO
2770 IF MO<1 OR MO>12 THEN 276Ø
2780 INPUT "□YEAR:";YR
2790 IF YR<1753 OR YR>29999 THEN 278Ø
2800 GOSUB 65Ø
2810 RETURN
2820 'MY HEADER ROUTINE
2830 PRINTMID$(MN$,MO*9-8,9);"□";YR
2840 IF P=2 THEN PRINT#-2,MID$
    (MN$,MO*9-8,9);"□□";YR
2850 RETURN
```

CHANGES FOR DISK

Here are the changes to make to the main program to use it with a disk drive, (the Commodore program can already be used with disk or tape):

```
1610 CLS : INPUT "Which drive ? ";drv
1620 INPUT "Enter today's date ";b$
1625 SAVE *"M";1;"C."+B$ DATA Q()
1630 OPEN #4;"m";1;"D."+b$
1660 FOR M=1 TO Q(N): PRINT
    #4;L$(N,M): NEXT M
1675 CLOSE #4
1690 CLS : INPUT "Which drive ?";drv
1700 CAT drv: INPUT "Enter name of file
    counter (pre-fixed C.) ";b$
1705 LOAD *"m";drv;b$ DATA Q()
1707 INPUT "Enter name of file required
    (prefixed D.) ";b$: OPEN #4;"m";drv;b$
1730 FOR M=1 TO Q(N): INPUT
    #4;L$(N,M);: NEXT M
1745 CLOSE #4
```

Simply delete Line 22

Delete Line 189Ø then add:

```
1750 CREATE"DIARY"
1780 FWRITE"DIARY";LI$(N,Ø)
1810 FOR J=1 TO 4:FWRITE"DIARY";STR$
    (ASC(MID$(LI$(N,P),J,1))):NEXTJ
1820 FWRITE"DIARY";MID$(LI$(N,P),5)
1850 CLOSE
1910 FLREAD"DIARY";LI$(N,Ø)
1950 FORJ=1 TO4:FREAD"DIARY";NN$:
    PRINTNN$:LI$(N,P)=LI$(N,P)+CHR$
    (VAL(NN$)):NEXTJ
1960 FLREAD"DIARY";NN$:LI$(N,P)=LI$
    (N,P)+NN$
1990 CLOSE
```

CHANGES FOR THE ELECTRON

As the program for the BBC is written in Mode 7, and uses teletext control codes for colour and double-height characters you'll need to make a few changes before it will RUN on the Electron. You'll need to delete Lines 1Ø8Ø, 1Ø9Ø, 111Ø and 113Ø and alter the following:

```
10 MODE6
140 MODE6:C%=FNmenu:MODE6
1120 PRINT:PRINT
1150 PRINT
1440 PRINT F$+Type$(t%)
2150 PRINT
2485 JM%=FNmarker(type%):IF
    JM%<>135 EOR type%<>5
    THEN COLOURØ:COLOUR129 ELSE
    COLOUR1:COLOUR128
2490 PRINTCHR$(JM%);d$;
2535 COLOUR1:COLOUR128
2910 PRINT
```

TELETEXT SCREENS ON THE BBC

Find out how to make the most of the Teletext mode on the BBC and how to put together the graphics shapes to create a complete and colourful screen picture

The real advantage of using the Teletext mode on the BBC is that it uses so little memory. Computers that have good graphics facilities usually need a lot of memory for the screen display. In the case of the BBC this varies from 8K to 20K. But the Teletext mode—mode 7—takes only 1K! In fact it uses exactly 1000 bytes because the screen is divided into 25 lines of 40 characters and the characters, whether text or graphics, take up only one byte each. The graphics are quite 'chunky', but with a little skill you can create quite complex and detailed pictures.

The reason the characters take up so little memory is because they are generated by a special Teletext chip. So all the information about the way the character is drawn on the screen is stored in this chip rather than in the computer's RAM. All the RAM needs to do is to store the character's ASCII code which is then sent on to the Teletext chip.

The Teletext letters, numbers, punctuation and most of the symbols have the same ASCII codes as characters in the other modes. But whereas the other modes leave 128 ASCII codes undefined (so they can be used for UDGs), the Teletext mode uses some of these codes for graphics characters such as the shape and others as extra control codes to change colour. In fact it uses the set of 256 possible ASCII codes very efficiently to form a combined coloured text and graphics system used on television's Ceefax, 4-Tel and Oracle as well as electronic networks.

1. The diagram, right, shows how to work out the code number for any desired shape. The screen above shows the entire range of shapes and codes

GRAPHICS AND ASCII CODES

The first program prints out all the graphics characters along with their ASCII codes, so you can see what's available:

```
10 MODE7
20 @%=3
30 PRINT'"ASCII Code followed by graphics
   shape"
40 PRINT
50 FOR N=0 TO 15
60 FOR NN=0 TO 5
70 C=160+N*6+NN
80 PRINTCHR$134,C,CHR$147+CHR$C;
90 NEXT
100 PRINT
110 NEXT
120 @%=&90A
130 END
```

Line 20 sets the field width to three columns so all the codes and graphics fit closely together on the screen. Lines 50 to 70 step through the relevant codes. The calculation is arranged to print out 16 rows of 6 columns. The codes go from 160 to 255.

If you look at Line 80 you'll see there are four items being printed. The first is a control code (134) to print out cyan coloured text, the second prints out the code number, the third sets it in yellow graphics (code 147) and the last prints the graphics or text character corresponding to the code number. Here are the essentials of a Teletext mode line. It is multi-coloured with mixed text and graphics, plus control codes (that print as spaces) and yet it uses only a few bytes of memory.

COLOUR CODES

In the last program the control codes 134 and 147 produced the colours in the display. There are separate codes to colour text and graphics and the whole list of these codes is shown below:

Colour	Graphics	Text
red	145	129
green	146	130
yellow	147	131
blue	148	132
magenta	149	133
cyan	150	134
white	151	135

Try substituting different codes in the program, replacing the numbers 134 and 147 with any of the numbers above and see what effect they have.

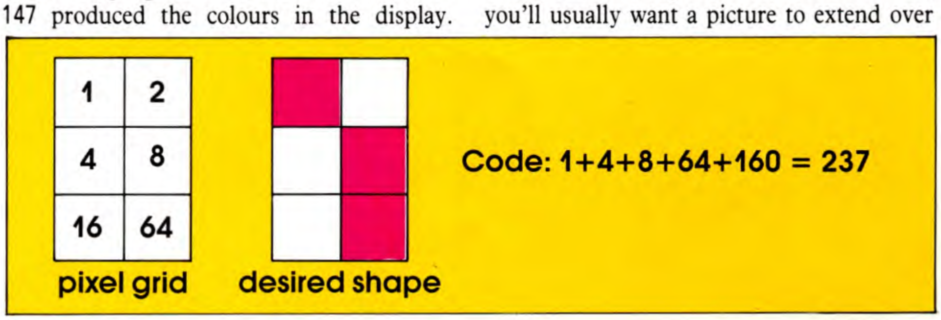
GRAPHICS SHAPES

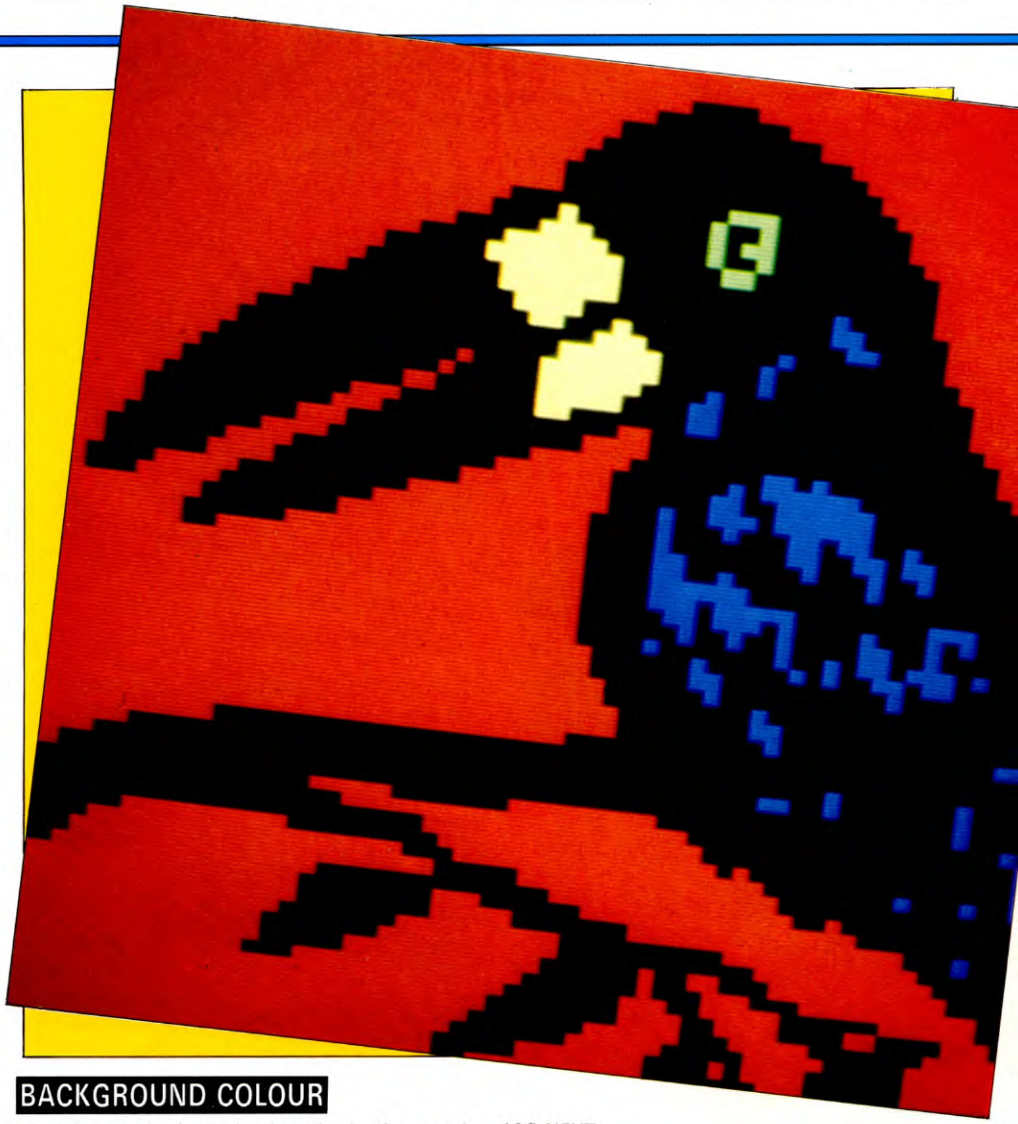
If you have a close look at the graphics shapes you'll see they are based on a 2 by 3 grid. This gives an overall graphics resolution of 2 pixels by 40 columns (80 across) and 3 pixels by 25 lines (75 deep). These 'chunky' graphics are suitable for nearly all applications except for line drawings.

As there are 64 different combinations of pixels on a 2 by 3 grid it can be rather difficult to find out the code of a particular shape from the list in the manual. Fortunately, it is simple to calculate the code for any symbol. It is simply a matter of using the master grid pattern where each cell of the grid represents a number. Just compare the shape to the grid and add up the numbers, plus a final 160. Have a look at the diagram below to see what the numbers are, and how to add them up.

PICTURES IN A STRING

To print a shape on the screen just type PRINT CHR$ followed by its number. However, you'll usually want a picture to extend over





several lines on the screen and there is a bit of skill involved in positioning the characters in the correct place. Type in and RUN the next program to see one way in which this can be done:

```
10 MODE7
20 PRINTCHR$145+CHR$160+CHR$240+
   CHR$252+CHR$252+CHR$180
30 PRINTCHR$145+CHR$232+CHR$255+
   CHR$255+CHR$241+CHR$255
40 PRINTCHR$145+CHR$234+CHR$255+
   CHR$255+CHR$239+CHR$239
50 PRINTCHR$145+CHR$170+CHR$255+
   CHR$255+CHR$240+CHR$252+CHR$
   172+CHR$247+CHR$243
60 PRINTCHR$145+CHR$160+CHR$163+
   CHR$175+CHR$167+CHR$163
```

In this program each line is printed out separately, one underneath the other. However, this is not a very useful way of storing a picture. It is far better to join it into a single string as shown in the next few lines:

```
70 A$=CHR$145+CHR$160+CHR$240+
   CHR$252+CHR$252+CHR$180
80 B$=CHR$145+CHR$232+CHR$255+
   CHR$255+CHR$241+CHR$255
90 C$=CHR$145+CHR$234+CHR$255+
   CHR$255+CHR$239+CHR$239
100 D$=CHR$145+CHR$170+CHR$
   255+CHR$255+CHR$240+CHR$252+
   CHR$172+CHR$247+CHR$243
110 E$=CHR$145+CHR$160+CHR$
   163+CHR$175+CHR$167+CHR$163
120 NL$=CHR$10+STRING$(6,CHR$8)
130 X$=A$+NL$+B$+NL$+C$+
   NL$+D$+NL$+STRING$(3,CHR$8)+
   E$
140 PRINTTAB(10,15)X$
150 PRINTTAB(25,7)X$
```

This time each line is assigned to a single string and these are all added together in Line 130. Notice that ordinary control codes are used in this line to move the cursor down one line (CHR$ 10) and backspace (CHR$ 8). The whole picture can then be printed anywhere on the screen using PRINTTAB as shown in Lines 140 and 150.

BACKGROUND COLOUR

Now have a look at altering the background colour. The control code to do this is 157, and you have to precede it with the colour you would like. For example, the two codes CHR$131+CHR$157 would make the rest of the line it is printed on have a yellow background. You need to print the codes at the start of every line you want to change but this can easily be carried out using a FOR ... NEXT loop. Add the next few lines to the program to create a yellow area between rows 12 to 20 and columns 12 to the end of the line:

```
160 CLS
170 FOR Y=12 TO 20
180 PRINTTAB(12,Y)CHR$131+CHR$157
190 NEXT
```

This can be repeated anywhere on the screen, including colouring the whole screen as shown by Lines 200 to 220:

```
200 FOR Y=0 TO 24
210 PRINTTAB(0,Y)CHR$130+CHR$157
220 NEXT
230 FOR Y=3 TO 8
240 PRINTTAB(8,Y)CHR$132+CHR$157
250 PRINTTAB(25,Y)CHR$130+CHR$157
260 NEXT
```

Lines 230 to 260 print codes for blue and background in positions 8 and 9 of each line, followed by codes for green and background at positions 25 and 26. This causes a blue box to be printed inside the green background. The cursor ends up on a line underneath the blue box.

SCREEN LAYOUT

Now that there are a few things on the screen, have a close look at just one line. Line 5 (the sixth line on the screen because the initial line is 0) is 40 columns long, numbered from 0 to 39. Fig. 2 shows the codes that are on the screen. The first appears as a space, and the other five are hidden in the line. In fact one of the things to be careful about when using Teletext is not to alter or remove any codes inadvertently, such as might happen if you print any text on the screen. The default colours are always white text on a black background.

SPECIAL EFFECTS

There are a series of other control codes in addition to the colour codes, that control

2. The diagram shows all the control codes in line 5 of the screen picture above. Only the first one appears as a gap, the other five are hidden

some of the special effects. They are listed below:

Code	Effect
136	Makes rest of line flash (foreground only)
137	Turns flash off
140	Gives normal height text or graphics
141	Gives double height text or graphics
152	Hides rest of line until colour change
153	Joins graphics pixels
154	Separates graphics pixels
156	Turns background to black
157	Turns background coloured
158	Holds graphics
159	Releases graphics

There are other code numbers between 128 and 159 but they have no effect.

HOLD GRAPHICS

The next few program lines demonstrate the difference between hold graphics and release graphics. Key in Line 270 first, adding it to the program already in memory:

```
270 PRINTTAB(12,3)X$
```

This prints the graphics picture created earlier. To liven it up a little it could perhaps have a white tongue, easily managed by putting CHR$151 for white graphics in the correct position.

The next line does this for you:

```
280 PRINTTAB(15,6)CHR$151
```

But the control character leaves a hole in the picture. This is where the hold graphics code 158 comes in. If this code is present at some earlier position on the line (and not cancelled by a following release graphics code) then, whenever there would normally be a hole, the previous character is copied into the hole. See it in action in the next line:

```
290 PRINTTAB(11,6)CHR$158
```

A picture can often look more effective if separated graphics are used, causing each individual pixel to be surrounded by background colour. The next few lines show how the little alien looks when printed in separated graphics:

```
300 PRINTTAB(18,12)X$
310 FOR Y=12 TO 18:PRINTTAB(17,Y)CHR$
    154:NEXT
```

The effect can be turned off on any line by using code 153.

HIDDEN TEXT

Another special effect is the ability to hide something printed on the screen. An apparently blank screen could thus be full of information! Try the next section of program, as before, add it to the program already in the computer:

```
320 PRINTTAB(26,13)CHR$132+"□Hello"
330 PRINTTAB(27,13)CHR$152
```

In Line 330, code 152 is put on the screen just before the message and it will cause invisible printing until the end of the line or a colour change. To counteract it, some other code has to be inserted—such as a code that does nothing, like 155:

```
340 PRINTTAB(27,13)CHR$155
```

DOUBLE HEIGHT

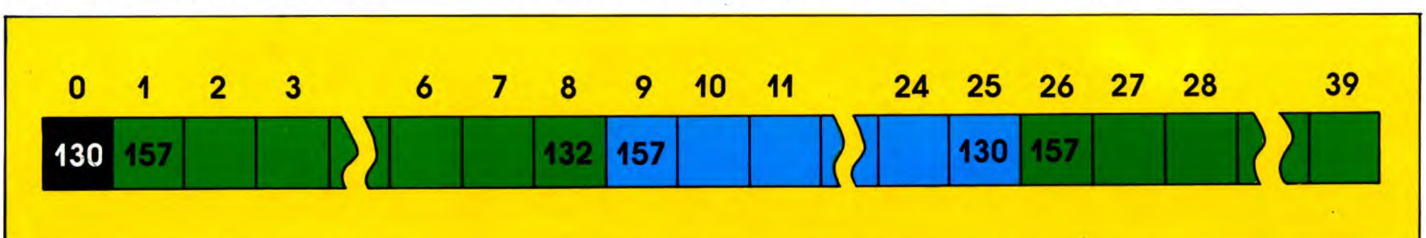
Double height printing is particularly useful for headings. The code to use is 141, but the point to note is that the code along with the text has to be repeated twice, one on each of the two lines needed for the double height. Lines 350 and 360 print the word 'Title' in double height characters, and in the colour magenta:

```
350 PRINTTAB(4,1)CHR$133+CHR$141+
    "Title"
360 PRINTTAB(4,2)CHR$133+CHR$141+
    "Title"
```

FLASHING

A final effect is the flash code 136. Putting this code in front of either text or graphics causes it to flash on and off until cancelled by code 137. Add these last three lines to the program:

```
370 PRINTTAB(3,1)CHR$136
380 PRINTTAB(3,2)CHR$136
390 PRINTTAB(4,23)
```



A TELETEXT PAGE

Now that the basic theory has been covered have a look at preparing a Teletext page either as a title page for a game or program, or for transmission along a telephone line to another computer.

DIRECT ENTRY GRAPHICS

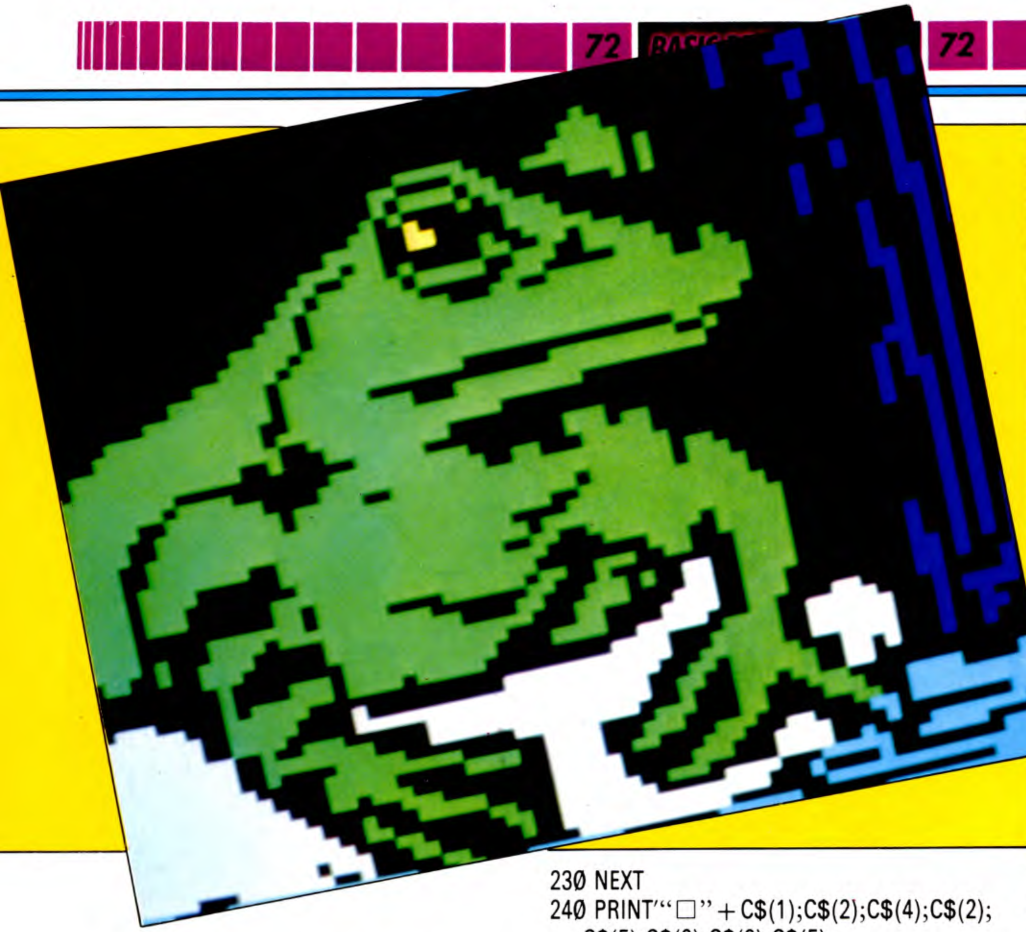
There are several methods of entering graphics symbols other than using CHR$ for each character. One is to use the keyboard keys direct. Try this:

```
PRINT CHR$149 + "abc"
```

You will notice that instead of the letters "abc" you get graphics characters. What is happening is that the control code 149 puts the whole of the line into magenta graphics mode, so that the computer assumes that graphics characters are following. It does not test it. Consequently, because the difference between the ordinary ASCII letters and the Teletext graphics is 128, instead of printing "a" which has ASCII value 64 it actually prints the Teletext character 64 + 128. So you can fool the computer into accepting ordinary keyboard input which it turns into a picture for you. This method does not always succeed since one character does not perform this trick: the #, but it can be entered using CHR$. Type in the following lines to see how the method works, it prints the word INPUT in large letters at the top of the screen.

```
10 MODE7
20 A$=CHR$132+CHR$157+CHR$151
30 PRINTA$+",,$,,$□□(,,(,,,,□□,,$(,,(,
   ,,,,"
40 PRINTA$+"cs!cscp□""s3""s3£cq□cs!"
   "s3b3cs£s"
50 PRINTA$+"(,□(,□,$□,$□,$□□,$(,
   □□,$□□(,□□"
60 PRINTA$+"bs□bs□""sØs1□sqpr£□bs
   □□s1□□bs□□"
70 PRINTA$+"(,□(,□□(,,$□,,,□□
   □(,□□,$□□(,□□"
80 PRINTA$+"rsØrsØ□□cs1"+CHR$
   224+"sq□□□□□cqr3□□□rsØ
   □"
90 PRINTA$+",,$,,$□□□,$(,,
   □□□□□□□,,□□□□,,$"
```

The listing looks quite strange but is actually simple. What is happening is that Line 2Ø is a string to make the background colour blue, and set the rest of the line to white graphics. It is put in front of each of the strings that follow (Lines 3Ø to 9Ø) so that the letters are translated as graphics characters. Occasionally, because of the awkward



character, the full version of the character has to be input with CHR$ as shown in Line 80. Also, a double quote mark has to be entered twice for the program to recognize it.

If you do not get a nice recognizable blue and white picture when you RUN this it is because you have mis-typed the rather incomprehensible strings, which is very easy to do.

DATA STATEMENT PICTURES

A surer method of entering graphics characters is to use the totals obtained by adding up the pixel numbers, as explained earlier and putting the numbers into DATA statements. You then let the computer add on the 160 and form the strings itself. Type in and RUN the next section:

```
100 DIM C$(10)
110 FOR N=1 TO 5
120 COL=145+N
130 B$=CHR$10+STRING$(4,CHR$8)+
   CHR$(COL)
140 C$(N)=CHR$(COL)
150 FOR L=1 TO 4
160 FOR P=1 TO 3
170 READ C
180 C$(N)=C$(N)+CHR$(C+160)
190 NEXT
200 IF L<4 C$(N)=C$(N)+B$
210 NEXT
220 C$(N)=C$(N)+CHR$11+CHR$11
230 NEXT
240 PRINT'"□"+C$(1);C$(2);C$(4);C$(2);
   C$(5);C$(2);C$(3);C$(5)
250 DATA 88,92,92,0,95,0,0,95,0,0,11,4
260 DATA 0,0,0,88,92,92,95,12,15,11,12,12
270 DATA 0,0,0,84,0,88,66,79,17,7,0,11
280 DATA 80,16,0,74,21,0,74,21,0,2,13,0
290 DATA 64,0,0,78,31,0,74,21,0,2,13,0
```

These lines print the word 'teletext' at an angle on the screen. The word is made up of five sections, each made of four rows of three columns plus a graphics control character at the start of each row. The shape of the graphics really needs to be planned out on graph paper or on a grid to make up the shape of the letters.

The actual strings for each section are stored in the string array by Line 100. Lines 120 to 220 read the DATA for each section and convert it into a string. B$ helps by controlling the position of the cursor, taking it down one line and back-spacing four places, and then adding on the graphics colour control code COL. COL is altered for each section so the letters are printed in different colours. This is repeated for each picture by the loop in Lines 110 and 230.

All that remains is to print out the sections in the right order. Line 220 adds two cursor-up controls—CHR$ 11— to the end of each string. The Line 240 prints out each of the strings. An initial space is put in for a control code to be added later (see below). All in all, this is a rather complicated procedure—but it does save a lot of work in turning the pixel patterns into a string.

The next three lines add the extra control code at the start of each line to complete this part of the picture:

```
300 FOR Y=8 TO 20
310 PRINTTAB(0,Y)CHR$154
320 NEXT
```

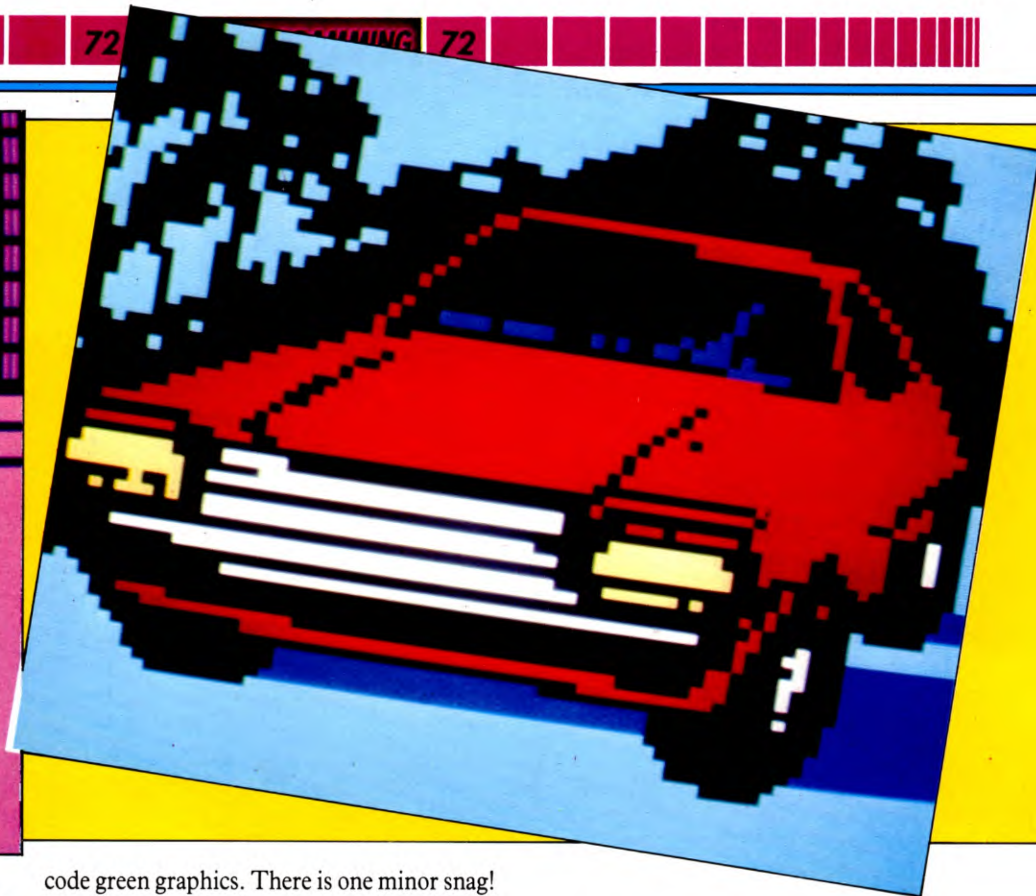
HIDDEN MESSAGES

First add these lines to the program. They print the word 'Message' in double height characters on a coloured background:

```
330 PRINTTAB(0,21)CHR$129+CHR$157+
   CHR$135+CHR$141+"Message:"
340 PRINTCHR$132+CHR$157+CHR$
   135+CHR$141+"Message:"+CHR$11;
```

The last part of the Teletext screen layout shows yet another method of putting the codes on the screen using the VDU command. Again, the actual codes are contained in the DATA statement:

```
350 FOR P=1 TO 25
360 READ C
370 VDU C,10,8,C,11
380 NEXT
390 PRINT'
400 DATA 145,158,232,233,161,146,247,229,
   236,236,147,228,239,238,229,174,149,152,
   201,207,213,177,135,136,63
```



The code C is READ and then put on the screen twice, one above the other. It prints out a secret message. See if you can decode it.

TRANSMISSION OF TELETEXT

Now that you have a teletext page, all that is needed is a modem, a telephone, a small amount of software and something at the other end of the line to receive it. This multi-coloured complex picture could then be sent in just 1000 bytes of information; a remarkable achievement for such a simple code.

DIRECT ENTRY CONTROL CODES

The final part of this article shows a shorter method of putting the control codes onto the screen. So far you've used CHR$ and VDU for the codes and direct keyboard entry for the actual characters themselves. But the control codes 128 to 159 can also be entered directly. One minor problem is that all you can see after their entry is another space on the screen! It is only the effect upon the following characters on the line that will show if you have chosen the correct keyboard sequence.

The method is to use the red function keys in combination with the SHIFT and CTRL keys. The function keys produce a value that is added to 128 if the SHIFT key is pressed at the same time; added to 140 with the CTRL key and added to 150 if both SHIFT and CTRL keys are pressed simultaneously. Thus if f6 and CTRL are pressed together, then code 140 + 6 is sent to the screen which is Teletext code green graphics. There is one minor snag! These work perfectly well until both SHIFT and CTRL are used together, when nothing will happen. This particular combination needs setting up each time the computer is turned on or after BREAK has been pressed. The relevant command is to type in *FX228,15Ø and *FX227,14Ø. This sets the base number for the SHIFT CTRL option to 15Ø. The entire list of possibilities is shown below.

Now try creating a picture by typing in both the control codes and graphics characters in direct mode. It is a difficult thing to do unless you plan it out very carefully first.

Direct entry control codes

Function Key No.	SHIFT	CTRL	SHIFT CTRL
fØ	Nothing	Normal Height	Graphics Cyan
f1	Text Red	Double Height	Graphics White
f2	Text Green	Nothing	Conceal Display
f3	Text Yellow	Nothing	Joined Pixels
f4	Text Blue	Nothing	Separated Pixels
f5	Text Magenta	Graphics Red	Nothing
f6	Text Cyan	Graphics Green	Black Background
f7	Text White	Graphics Yellow	New Background
f8	Flash	Graphics Blue	Hold Graphics
f9	Steady	Graphics Magenta	Release Graphics

THE FRUITS OF YOUR LABOURS

Release the hold on your Fruit Machine and try to hit the jackpot with the second half of the program. The payout's 20 times as much as your stake!

In this second part of the Fruit Machine article, you'll complete the program, and the machine will be ready to play.

HOLD ON TO YOUR HATS

```
190 LET HOLD=Ø
200 LET TOTAL=TOTAL-1Ø: GOSUB 75Ø:
    IF TOTAL<Ø THEN GOTO 77Ø: LET
    NUDGE=Ø: PRINT AT 13,26; INK
    2:"■■■■■"
210 IF HFLAG=Ø THEN LET HOLD=Ø
220 FOR I=1 TO 3: FOR J=1 TO 12: BEEP
    .ØØ1,6Ø
230 IF HOLD=Ø THEN PRINT AT7,1Ø;A$(J);
    AT 7,15;B$(J);AT 7,2Ø;C$(J);AT 1Ø,1Ø;A$
    (J+1);AT 1Ø,15;B$(J+1);AT 1Ø,2Ø;C$
    (J+1);AT 13,1Ø;A$(J+2);AT 13,15;B$
    (J+2);AT 13,2Ø;C$(J+2): NEXT J: NEXT I
240 IF HOLD=1 THEN PRINT AT 7,15;B$(J);
    AT 7,2Ø;C$(J);AT 1Ø,15;B$(J+1);AT 1Ø,
    2Ø;C$(J+1);AT 13,15;B$(J+2);AT
    13,2Ø;C$(J+2):NEXTJ: NEXTI
250 IF HOLD=4 THEN PRINT AT 7,2Ø; A$
    (J); AT 1Ø,2Ø; B$(J); AT 13,2Ø; C$(J):
    NEXT J: NEXT I
260 IF HOLD=6 THEN PRINT AT 7,15;B$(J);
    AT 1Ø,15;B$(J+1);AT 13,15;B$(J+2):
    NEXT J: NEXT I
270 IF HOLD=2 THEN PRINT AT 7,1Ø;A$(J);
    AT 7,2Ø;C$(J);AT 1Ø,1Ø;A$(J+1);AT 1Ø,
    2Ø;C$(J+1);AT 13,1Ø;A$(J+2);AT
    13,2Ø;C$(J+2): NEXTJ: NEXT I
280 IF HOLD=5 THEN PRINT AT 7,1Ø;A$
    (J);AT 1Ø,1Ø;A$(J+1);AT 13,1Ø;
    A$(J+2): NEXT J: NEXT I
290 IF HOLD=3 THEN PRINT AT 7,1Ø;A$(J);
    AT 7,15;B$(J);AT 1Ø,1Ø;A$(J+1);AT 1Ø,
    15;B$(J+1);AT 13,1Ø;A$(J+2);AT 13,
    15;B$(J+2): NEXT J: NEXT I
300 IF HOLD<>1 AND HOLD<>4 AND
    HOLD<>6 THEN LET M=INT (RND*12):
    IF M=Ø THEN LET M=1
310 IF HOLD<>2 AND HOLD<>5 AND
    HOLD<>4 THEN LET K=INT (RND*12):
    IF K=Ø THEN LET K=1
320 IF HOLD<>3 AND HOLD<>5 AND
    HOLD<>6 THEN LET L=INT (RND*12):
    IF L=Ø THEN LET L=1
330 LET HOLD=Ø
340 PRINT AT 7,1Ø;A$(M);AT 7,15;B$(K);AT
    7,2Ø;C$(L);AT 1Ø,1Ø;A$(M+1);AT 1Ø,15;
    B$(K+1);AT 1Ø,2Ø;C$(L+1);AT
    13,1Ø;A$(M+2);AT 13,15;B$(K+2);AT
    13,2Ø;C$(L+2)
```

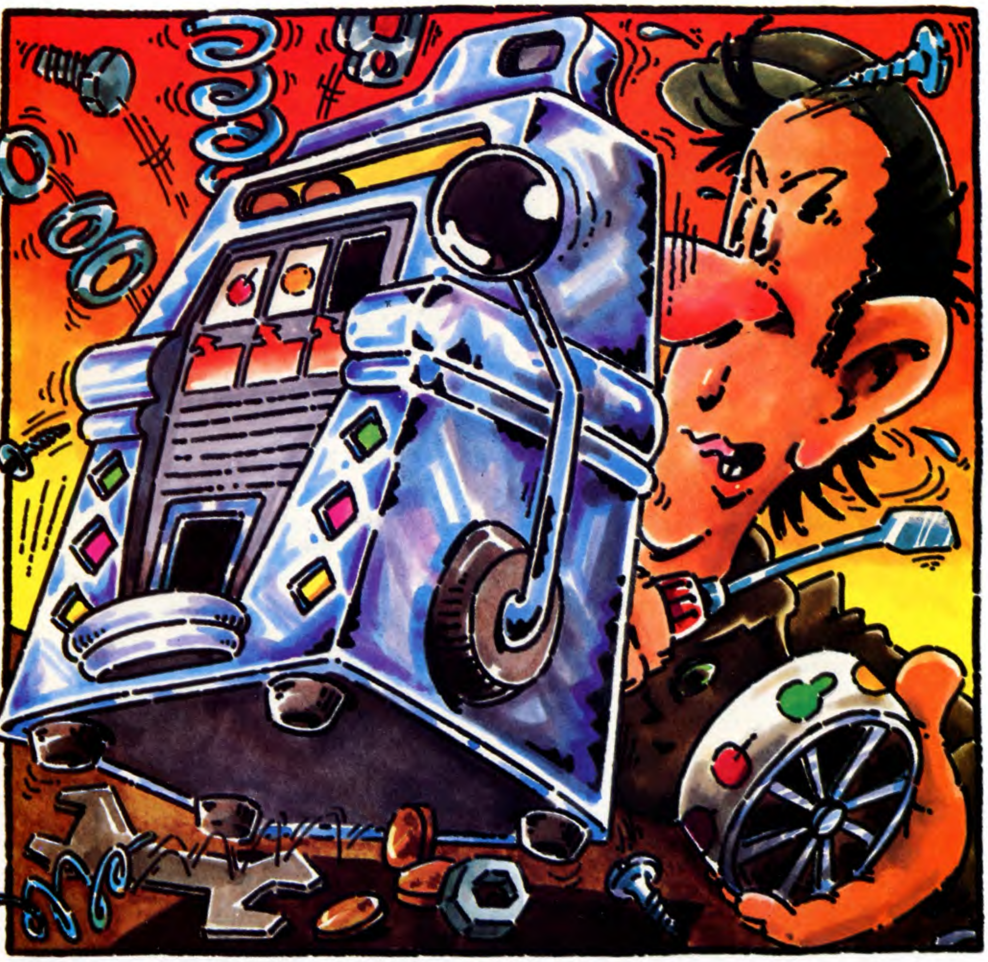
Line 19Ø sets the HOLD variable to zero. The routine looks for the value of HOLD—determined by the key pressed by the player—and spins the free reels.

After the reels have been spun, HOLD is reset to zero in Line 33Ø—the hold buttons are cleared—and the reels are PRINTed in their stopped position in Line 34Ø.

REEL FRUIT

```
350 GOSUB 6ØØ
600 LET T$=A$(M)+B$(K)+C$(L)
610 LET M$=A$(M+1)+B$(K+1)+
    C$(L+1)
620 LET L$=A$(M+2)+B$(K+2)+C$(L+2)
630 IF M$ ( TO 4)=C$(1) AND M$ ( TO 4)
    =M$ (5 TO 8) AND M$ ( TO 4)=M$
    (9 TO ) THEN LET TOTAL=TOTAL+1ØØ:
    GOSUB 75Ø: GOSUB 76Ø: GOTO 38Ø
640 IF M$ ( TO 4)=C$ (3) AND M$ ( TO 4)
    =M$ (5 TO 8) AND M$ ( TO 4)=M$
    (9 TO ) THEN LET TOTAL=TOTAL+1ØØ:
    GOSUB 75Ø: GOSUB 76Ø: GOTO 38Ø
650 IF M$ ( TO 4)=C$ (9) AND M$ ( TO
    4)=M$ (5 TO 8) AND M$ ( TO 4)=M$
    (9 TO ) THEN LET TOTAL=TOTAL+5Ø:
    GOSUB 75Ø: GOSUB 76Ø: GOTO 38Ø
660 IF M$ ( TO 4)=C$(2) AND M$ ( TO
    4)=M$ (5 TO 8) AND M$ ( TO 4)=M$
    (9 TO ) THEN LET TOTAL=TOTAL+5Ø:
    GOSUB 76Ø: GOSUB 75Ø: GOTO 38Ø
670 IF M$ ( TO 4)=C$ (6) AND M$( TO
    4)=M$ (5 TO 8) AND M$ ( TO 4)=M$
    (9 TO ) THEN LET TOTAL=TOTAL+5Ø:
    GOSUB 75Ø: GOSUB 76Ø: GOTO 38Ø
680 IF M$ ( TO 4)=C$(5) AND M$ ( TO
    4)=M$ (5 TO 8) AND M$ ( TO 4)=M$
    (9 TO ) THEN LET TOTAL=TOTAL+5Ø:
    GOSUB 75Ø: GOSUB 76Ø: GOTO 38Ø
690 IF M$ ( TO 4)=C$ (4) AND M$ ( TO
    4)= M$ (5 TO 8) AND M$ ( TO 4)=M$
    (9 TO ) THEN LET TOTAL=TOTAL+5ØØ:
    GOTO 82Ø
700 IF M$ ( TO 4)=C$ (5) AND M$ ( TO
    4)= M$ (5 TO 8) AND M$ (9 TO )
    <>C$ (5) THEN LET
    TOTAL=TOTAL+2Ø: GOSUB 75Ø:
    GOSUB 76Ø: GOTO 38Ø
710 IF M$ ( TO 4)=C$(5) AND M$(5 TO
    8)<>C$(5) THEN LET
    TOTAL=TOTAL+1Ø: GOSUB 75Ø:
    GOSUB 76Ø: GOTO 38Ø
720 IF M$ ( TO 4)=C$(4) AND M$ ( TO
    4)=M$(5 TO 8) AND M$( TO
    4)<>M$(9 TO ) THEN GOSUB 76Ø:
    GOTO 38Ø
730 IF M$ ( TO 4)=C$(4) AND M$ ( TO
    4)<>M$(5 TO 8) THEN GOSUB 76Ø:
    GOTO 38Ø
740 IF M$(5 TO 8)=C$(4) AND M$(5 TO
    8)<>M$(9 TO ) THEN GOSUB 76Ø:
    GOTO 38Ø
750 LET DD=INT (TOTAL/1ØØ): LET
    CC=TOTAL-(DD*1ØØ): PRINT INK 2;
    PAPER 6; AT 17,Ø; "£□"; AT 18Ø; "P□";
    PAPER 7; BRIGHT 1; AT 17,1; "□";
    DD;AT 18,1;"□"; CC: RETURN
760 LET HFLAG=Ø: LET HOLD=Ø: LET
    NUDGE=Ø: FOR I=9 TO 19 STEP 5:
    PRINT AT 16,1) INK 2; "■■■■":
```

- COMPLETE YOUR FRUIT MACHINE
- THE MAIN LOOP
- SPINNING THE REELS
- CHECKING FOR WINNING LINES
- ADJUSTING THE CREDIT
- FLASHING HOLDS, NUDGES AND GAMBLES
- PRESSING THE BUTTONS
- SOUND EFFECTS



```
    NEXT I: PRINT AT 13,26; INK 2;
    "■■■■■": RETURN
```

Line 35Ø jumps to the subroutine starting at Line 6ØØ, which puts the three rows that are being displayed on the reels into three strings—T$, M$ and L$. The middle string, M$, is the line from which the score is calculated in the subroutine starting at Line 63Ø. The routine checks if there are any winning lines, and adds the winnings to the player's total.

GIVING IT THE NUDGE

```
36Ø IF M<7 OR K=L OR L>2 THEN LET
    NUDGE =1: PRINT BRIGHT 1; PAPER 7;
    INK 2;AT 13,26;"NUDGE"
37Ø LET HFLAG=INT (RND+.5): IF
    HFLAG=1 THEN FOR I=9 TO 19 STEP 5:
    PRINT AT 16,I; INK 6; BRIGHT 1;"HOLD":
    NEXT I
38Ø LET I$=INKEY$: IF I$="" THEN GOTO
    38Ø
39Ø IF I$="□" THEN FOR I=9 TO 19
    STEP 5: PRINT INK 2;AT 16,1;
    "■■■■": NEXT I: GOTO 2ØØ
4ØØ IF I$="E" AND NUDGE=1 THEN
    GOSUB 52Ø: LET NUDGE=Ø: PRINT AT
    13,26; INK 2; "■■■■■": BEEP .1,
    3Ø: GOSUB 6ØØ: LET RN=INT (RND*1Ø):
    IF INT (RN/2)=RN/2 AND HFLAG<>1
    THEN LET NUDGE=1: PRINT AT 13,26;
    IN7; BRIGHT 1; "NUDGE": GOTO 38Ø
41Ø IF I$="Q" AND NUDGE=1 THEN
    GOSUB 48Ø: LET NUDGE=Ø: PRINT AT
    13,26; INK 2; "■■■■■": BEEP .1,
    3Ø: GOSUB 6ØØ: LET RN=INT (RND*1Ø):
    IF INT (RN/2)=RN/2 AND RN<3 THEN
    LET NUDGE=1: PRINT AT 13,26; INK 7;
    BRIGHT 1;"NUDGE": GOTO 38Ø
42Ø IF I$="W" AND NUDGE=1 THEN
    GOSUB 5ØØ: LET NUDGE=Ø: PRINT AT
    13,26; INK 2; "■■■■■": BEEP .1,
    3Ø: GOSUB 6ØØ: GOTO 38Ø
43Ø IF I$="D" AND NUDGE=1 THEN
    GOSUB 54Ø. LET NUDGE=Ø: PRINT AT
    13,26; INK 2; "■■■■■": BEEP .1,
    3Ø: GOSUB 6ØØ LET RN=INT (RND*1Ø):
    IF INT (RN/2)<>RN/2 THEN LET
    NUDGE=1: PRINT AT 13,26; INK 7;
    BRIGHT 1;"NUDGE": GOTO 38Ø
44Ø IF I$="S" AND NUDGE=1 THEN
    GOSUB 58Ø: LET NUDGE=Ø: PRINT AT
    13,26; INK 2;'■■■■■": BEEP .1,
    3Ø: GOSUB 6ØØ: LET RN=INT (RND*1Ø):
    IF INT (RN/2)=RN/2 THEN LET
    NUDGE=1: PRINT AT 13,26; INK 7;
    BRIGHT 1;"NUDGE": GOTO 38Ø
45Ø IF I$="A" AND NUDGE=1 THEN
```

```
GOSUB 560: LET NUDGE=0: PRINT AT
   13,26; INK 2; "■■■■■": BEEP .1,
   30: GOSUB 600: LET RN=INT (RND*10):
   IF INT (RN/2)<>RN/2 AND RN>6
   THEN LET NUDGE=1: PRINT AT 13,26;
   INK 7; BRIGHT 1;"NUDGE": GOTO 380
460 IF HFLAG=1 AND I$="1" OR I$="2"
   OR I$="3" OR I$="4" OR I$="5" OR
   I$="6" THEN LET HOLD=VAL I$: FOR
   I=9 TO 19 STEP 5: PRINT AT 16,I; INK 2;
   "■■■■■": NEXT I: GOTO 200
470 GOTO 380
480 LET M=M+1: IF M>12 THEN LET
   M=M-12
490 PRINT AT 7,10;A$(M);AT
   10,10;A$(M+1);AT 13,10;A$(M+2):
   RETURN
500 LET K=K+1: IF K>12 THEN LET
   K=K-12
510 PRINT AT 7,15;B$(K);AT
   10,15;B$(K+1);AT 13,15;B$(K+2):
   RETURN
520 LET L=L+1: IF L>12 THEN LET
   L=L-12
530 PRINT AT 7,20;C$(L);AT
   10,20;C$(L+1);AT 13,20;C$(L+2):
   RETURN
540 LET L=L-1: IF L<1 THEN LET
   L=L+12
550 PRINT AT 7,20;C$(L);AT
   10,20;C$(L+1);AT 13,20;C$(L+2):
   RETURN
560 LET M=M-1: IF M<1 THEN LET
   M=M+12
570 PRINT AT 7,10;A$(M);AT
   10,10;A$(M+1);AT 13,10;A$(M+2):
   RETURN
580 LET K=K-1: IF K<1 THEN LET
   K=K+12
590 PRINT AT 7,15;B$(K);AT
   10,15;B$(K+1);AT 13,15;B$(K+2):
   RETURN
```

The nudge routine is very similar to the hold routine. Reels are moved up or down according to the keyboard input from the player.. The instructions as to which keys to press are displayed on the screen. On each nudge, a random number is generated to determine if another nudge is to be offered.

THE JACKPOT

```
770 CLS : PRINT AT 10,0;"□□□□□□
   □□□□□GAME OVER□□□□□□□
   □□□□□□□□YOU HAVE LOST ALL
   YOUR MONEY": BEEP 1,-20
780 PRINT '"□□DO YOU WANT ANOTHER
   GAME ? PRESS ""Y"" OR ""N"""
790 IF INKEY$="" THEN GOTO 790
800 LET I$=INKEY$: IF I$="Y" OR
   I$="y" THEN RUN
810 STOP
820 CLS : PRINT AT 10,0;"□□□□□□
   □□CONGRATULATIONS□□□□□□□
   □□□□YOU HAVE JUST WON THE
   JACKPOT": PRINT "□□□□□YOU
   ARE RICHER BY £50": FOR J=1 TO 3:
   FOR I=1 TO 10: BEEP .01,5*I: NEXT I:
   NEXT J
830 GOTO 780
```

These routines are very simple. The 'another go?' routine—Lines 770 to 810—is called when the player runs out of money.

The Jackpot routine—Lines 820 and 830—tells the player the good news and adds £50 to the money total, before ending the game—the bank is bust. The player is now given the opportunity of another go.

THE MAIN LOOP

```
310 GOSUB1000:GOSUB2000:GOSUB3000:
   IFM>0THEN310
315 GOSUB4000:GOTO70
4000 POKE53270,PEEK(53270)AND239:POKE
   53280,2:POKE53281,8:POKE53272,21
4005PRINT"■■■";CHR$(14),"■■■
   ■□H□□AD□□UCK!"
4010 PRINT"■■■■■■■■■■
   □OU HAVE RUN OUT OF MONEY!"
4015 PRINT"■□RESS THE ■PACE-BAR
   FOR ANOTHER GAME..."
4020 GETA$:IFA$<>CHR$(32)THEN4020
4025 XX=1:GOTO70
```

Line 310 calls subroutines which spin the reels, check for winning lines, and check the player's key presses. The loop is repeated until the player runs out of money.

If the player does run out of money, Line 315 calls the subroutine at Line 4000 which announces the unwelcome news, and gives the option of another go.

SPINNING THE REELS

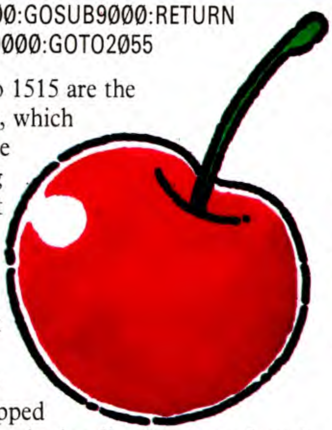
```
1000 PRINT"■■■■■■■■■■
   ■■■■■■■■■■■■■■
   □□□□■■■□□□□■
   ■■□□□□■■";:M=M-10
1005 GOSUB9500
1010 TI$="000000":DM=0:DR=100:IF
   NOTH%THEN1035
1015 IFH%THENGOSUB5500
1020 IFI%THENGOSUB6500
1025 IFJ%THENGOSUB7500
1030 IFTI$<"000002"ANDH%THENFOR
   Z=1TO50:NEXTZ:GOTO1015
1035 TI$="000000":IFNOTI%THEN1055
1040 IFI%THENGOSUB6500
1045 IFJ%THENGOSUB7500
1050 IFTI$<"000002"ANDI%THENFOR
   Z=1TO100:NEXTZ:GOTO1040
1055 TI$="000000":IFNOTJ%THEN1075
1060 IFJ%THENGOSUB7500
1065 IFTI$<"000002"ANDJ%THENFOR
   Z=1TO150:NEXTZ:GOTO1060
1075 H%=-1:I%=-1:J%=-1:RETURN
```

Lines 1000 to 1075 spin the reels which the player has chosen not to hold. H5, I5 and J% are the hold flags for the three reels.

NO HOLDS BARRED

```
1500 IFD%=0THENRETURN
1505 FORY=10TOD%STEP10:M=M+10
1510 GOSUB9500
1515 DR=40:DM=40:GOSUB8000:FOR
   T=1TO100:NEXTT,Y:RETURN
2000 D%=9:A=PEEK(1313):B=PEEK
   (1320):C=PEEK(1327):IFA=BANDB=C
   ANDC=11THEND%=0
2005 IFA=BANDB=CANDC=17THEN
   D%=1
2010 IFA=BANDB=CANDC=23THEN
   D%=2
2015 IFA=BANDB=CANDC=29THEN
   D%=3
2020 IFA=BANDB=CANDC=37THEN
   D%=4
2025 IFA=BANDB=CANDC=43THEN
   D%=5:GOTO2035
2030 IFA=BANDB=43THEND%=6
2035 IFA=BANDB=CANDC=59THEN
   D%=6:GOTO2050
2040 IFA=BANDB=59THEND%=7:GOTO
   2050
2045 IFA=59THEND%=8
2050 IFD%=9ORD%=0THEND%=W%
   (D%):GOSUB1500:RETURN
2055 FORA=1172+(D%*40)TO1176+
   (D%*40):POKEA+54272,14:NEXTA:
   LL=INT(RND(1)*2)
2060 FORA=1132+(LL*80)+(D%*40)TO
   1136+(LL*80)+(D%*40):POKE
   A+54272,12:NEXTA:CL=12
2065 GETA$:IFA$=CHR$(13)THEND%=W%
   (D%):GOSUB1500:GOSUB9000:RETURN
2070 IFA$=CHR$(32)THEN2090
2075 FORA=1212-(LL*80)+(D%*40)TO
   1216-(LL*80)+(D%*40):POKEA+
   54272,CL:NEXTA
2080CL=27-CL:FORA=1132+(LL*80)+
   (D%*40)TO1136+(LL*80)+(D%*40):
   POKEA+54272,CL
2085 NEXTA:GOTO2065
2090 FORA=1172+(D%*40)TO1176+
   (D%*40):POKEA+54272,15:NEXTA
2095 IF(CL=15ANDLL=0)OR(CL=12AND
   LL=1)THEND%=W%(D%+1):GOSUB
   1500:GOSUB9000:RETURN
2100 D%=D%-1:IFD%=0THEND%=200:
```

```
   GOSUB1500:GOSUB9000:RETURN
2105 GOSUB9000:GOTO2055
```



Lines 15ØØ to 1515 are the credit routine, which increments the M% according to the amount the player has won.

Line 2ØØØ to 21Ø5 check for winning lines once the reels have stopped spinning. If a winning line has been found, the routine at Line 15ØØ is called.

Lines 2Ø55 to 2Ø85 control the display to the right of the reels which shows the amount the player stands to win.

HOLD, NUDGES AND GAMBLES

```
3000 IFRND(1)<.25THENGOSUB4500
3005 IFRND(1)<.4THENCL$="◆▦":
   SS=1:H%=-1:I%=-1:J%=-1:
   GOTO3020
3010 GETA$:IFA$<>CHR$(32)THEN3010
3015 RETURN
3020 PRINT"◧▣▣▣▣▣▣▣▣▣▣▣
   ▣▣▣▣▣▮▮▮▮▮▮▮▮▮";
   MID$(CL$,SS,1);:IFH%THENPRINT"□⊟
   ◲◫●▮▮";
3025 IFI%THENPRINT"◧▣▣▣▣▣▣▣
   ▣▣▣▣▣▣▣▣","▮▮▮▮
   □□◲◫●▮▮";
3030 IFJ%THENPRINT"◧▣▣▣▣▣▣
   ▣▣▣▣▣▣▣▣▣",,"▮□⊟
   ◲◫●◧";
3035 GETA$:IFA$=CHR$(32)THENRETURN
3040 IFA$<"1"ORA$>"4"THEN
   SS=3-SS:GOTO3020
3045 ONVAL(A$)GOTO3065,3050,3055,3060
3050 H%=0:PRINT"◧▣▣▣▣▣▣▣
   ▣▣▣▣▣▣▣▣▣▮▮▮▮▮▮
   ▮▮♣□⊟◲◫●◧";:GOTO3020
3055 I%=0:PRINT"◧▣▣▣▣▣▣▣
   ▣▣▣▣▣▣▣▣","▮▮▮▮
   ♣□⊟◲◫●◧";:GOTO3020
3060 J%=0:PRINT"◧▣▣▣▣▣▣▣
   ▣▣▣▣▣▣▣▣",,"▮♣□
   ⊟◲◫●◧";:GOTO3020
3065 H%=-1:I%=-1:J%=-1:GOTO
   3020
4500 FORA=1760TO1772STEP3:FORB=ATO
   A+1:POKEB+54272,9:NEXTB
4505 GETA$:IFA$=CHR$(32)THEN
   N%=(A-1757)/3:GOTO4515
4510 FORB=ATOA+1:POKEB+54272,10:
   NEXTB,A:GOTO4500
4515 FORZ=N%TO1STEP-1:KK=
   ((Z-1)*3)+1760:POKEKK+54272,9:
   POKEKK+54273,9
4520 GETA$:IFA$>"9"OR(A$<"5"AND
   A$<>"Ø")THEN4520
4525 IFA$="Ø"THENA$="10"
4530 ONVAL(A$)-4GOSUB5000,6000,
   7000,5500,6500,7500
4535 DR=50:DM=40:GOSUB8000:GOSUB
   2000:KK=((Z-1)*3)+1760
4540 IFD%=0ANDZ=1THEN4550
4545 IFD%=0THENPOKEKK+54272,10:POKE
   KK+54273,10:NEXTZ
4550 FORA=1760TO1772STEP3:FORB=ATO
   A+1:POKEB+54272,10:NEXTB,A:
   RETURN
5000 PRINT"◧▣▣▣▣▣▮▮▮
   ▮▮▮▮▮";F$(R1%(P%));
   "▣▣▮▮▮▮▮▮";
5005 IFP%=15THENP%=-1
5010 P%=P%+1:PRINTF$
   (R1%(P%));"▣
   ▣▮▮▮▮▮▮";
5015 IFP%
   =15THEN
   PRINTF$
   (R1%(0));:
   RETURN
5020PRINTF$
   (R1%(P%+1));:
   RETURN
5500 IFP%<2THEN
   PRINT"◧▣▣▣▣▣▮▮▮▮
   ▮▮▮";F$(R1%(14+P%));
   "▣▣▮▮▮▮▮▮";:
   GOTO5510
5505 PRINT"◧▣▣▣▣▣▮▮▮▮▮
   ▮▮▮";F$(R1%(P%-2));"▣▣▮▮
   ▮▮▮▮";
5510 IFP%=0THENP%=16
5515 P%=P%-1:PRINTF$(R1%(P%));"▣
   ▣▮▮▮▮▮▮";
5520 IFP%=15THENPRINTF$(R1%(0));:
   RETURN
5525 PRINTF$(R1%(P%+1));:RETURN
6000 PRINT"◧▣▣▣▣","▮▮▮▮
   ▮";F$(R2%(Q%));"▣▣▮▮▮▮▮▮";
6005 IFQ%=15THENQ%=-1
6010 Q%=Q%+1:PRINTF$(R2%(Q%));"▣
   ▣▮▮▮▮▮▮";
6015 IFQ%=15THENPRINTF$(R2%(0));:
   RETURN
6020 PRINTF$(R2%(Q%+1));:RETURN
6500 IFQ%<2THENPRINT"◧▣▣▣
   ▣","▮▮▮▮▮▮";F$(R2%
   (14+Q%));"▣▣▮▮▮▮▮▮";:GOTO6510
6505 PRINT"◧▣▣▣▣","▮▮▮▮
   ▮";F$(R2%(Q%-2));"▣▣▮▮▮
   ▮▮";
6510 IFQ%=0THENQ%=16
6515 Q%=Q%-1:PRINTF$(R2%(Q%));"▣
   ▣▮▮▮▮▮▮";
6520 IFQ%=15THENPRINTF$(R2%(0));:
   RETURN
6525 PRINTF$(R2%(Q%+1));:RETURN
7000 PRINT"◧▣▣▣▣",,"▮▮";F$
   (R3%(R%));"▣▣▮▮▮▮▮▮";
7005 IFR%=15THENR%=-1
7010 R%=R%+1:PRINTF$(R3%(R%));
   "▣▣▮▮▮▮▮▮";
7015 IFR%=15THENPRINTF$(R3%(0));:
   RETURN
7020 PRINTF$(R3%(R%+1));:RETURN
7500 IFR%<2THENPRINT"◧▣▣▣
   ▣",,"▮▮";F$(R3%(14+R%));"▣
   ▣▮▮▮▮▮▮";:GOTO7510
7505 PRINT"◧▣▣▣▣",,"▮▮";F$
   (R3%(R%-2));"▣▣▮▮▮▮▮▮";
7510 IFR%=0THENR%=16
7515 R%=R%-1:PRINTF$(R3%(R%));"▣
   ▣▮▮▮▮▮▮";
7520 IFR%=15THENPRINTF$(R3%(0));:
   RETURN
7525 PRINTF$(R3%(R%+1));:RETURN
```

Line 3ØØØ chooses a random number which determines if the player is to be offered nudges, while Line 3ØØ5 sets the hold flags if the correct random number is chosen. Lines 3Ø2Ø to 3Ø65 flash the gamble and hold lights, and read the player's input.

Lines 45ØØ to 455Ø are the nudge routine. The reel moving subroutines are called according to the player's choice(s). Lines 5ØØØ to 52Ø5 make the first reel move up, while Lines 55ØØ to 5525 make the first reel move down. Lines 6ØØØ to 6Ø25, and Lines 65ØØ to 6525 do the same for the second reel, and Lines 7ØØØ to 7Ø25 and Lines 75ØØ to 7525 control the third reel.

OTHERS

```
8000 S=54272:POKES,150:POKES+1,
   50+DM:POKES+5,0:POKES+6,240:
   POKES+24,15
8005 POKES+4,17:FORDD=1TODR:NEXT
   DD:POKES+24,0:DM=DM+10:
   RETURN
9000 FORA=1172TO1532STEP40:FORB=A
   TOA+4:POKEB+54272,15:NEXTB,A:
   RETURN
9500 PRINT"◧▣▣▣▣▣▣▣▣▣▣
   ▣▣▣▣▣▣▣▣▣▣▣▣",,
   "▮▮▮▮▮▮▮▮▮●▮▮▮▮▮▮▮
   ▮▮◘";:IN=INT(M/100)
9505 RM=M-(IN*100):PRINTMID$(STR$
```

```
(IN),2);"□";MID$(STR$(RM)+"Ø",2,
2);:RETURN
```

Lines 8ØØØ and 8ØØ5 are a sound effect used when the reels stop—see page 232. Lines 9ØØØ to 95Ø5 clear the gamble display to the right of the reels so that it is no longer lit, ready for your next turn on Superfruit, the game for the push-button gambler.

THE MAIN LOOP

```
360 REPEAT:PROCspin:PROCcheck:PROCkeys:
UNTILM%=Ø:PROCend:RUN
```

Line 36Ø is the heart of the Fruit Machine, and is REPEATed UNTIL the memory held drops to zero. PROCspin, PROCcheck and PROCkeys are called repeatedly.

SPINNING THE REELS

```
37Ø DEFPROCspin:VDU5:GCOLØ,4:FORA%=Ø
TO2:MOVE192+A%*32Ø,384:PROChbox:
NEXT
38Ø VDU4,26:M%=M%-1Ø:COLOUR13Ø:
COLOUR6:PRINTTAB(12,3Ø);M%/1ØØ;
```

```
  "□":COLOUR128:GCOLØ,128
390 TIME=Ø
400 REPEAT
410 IFH%PROCreel1
420 IFI%PROCreel2
430 IFJ%PROCreel3
440 UNTILTIME>100+RND(5Ø):IFH%
  SOUND&11,1,5Ø,2
450 REPEAT
460 IFI%PROCreel2
470 IFJ%PROCreel3
480 UNTILTIME>200+RND(5Ø):IFI%
  SOUND&11,1,7Ø,2
490 REPEAT
500 IFJ%PROCreel3
510 UNTILTIME>300+RND(5Ø):IFJ%
  SOUND&11,1,9Ø,2
520 H%=TRUE:I%=TRUE:J%=TRUE:
  ENDPROC
```

Line 37Ø puts all the hold boxes in steady blue, and Line 38Ø takes 10p for a spin, and PRINTs the credit remaining.

The reels are spun by Lines 39Ø to 51Ø. The duration of the spin varies according to the number of reels that are being held. Line 52Ø resets the hold flags.

NO HOLDS BARRED

```
530 DEFPROCcred(D%):IFD%=ØENDPROC:
  ELSEVDU4,26:COLOUR13Ø:COLOUR6:FOR
  A%=1ØTOD%STEP1Ø:M%=M%+1Ø:
  PRINTTAB(12,3Ø);M%/1ØØ
540 SOUND&11,1,15Ø,1:FORB%=ØTO1ØØØ:
  NEXT:SOUND&11,Ø,Ø,1:NEXT:ENDPROC
550 DEFPROChbox:PLOT1,256,Ø:PLOT1,Ø,64:
  PLOT1,-256,Ø:PLOT1,Ø,-64:PLOTØ,8,
  48:PRINT"HOLD":ENDPROC
560DEFPROCreel1:VDU4,28,3,15,6,4,31,3,11,
  1Ø,1Ø,1Ø,5:P%=(P%+1)MOD16:MOVE
  256,576:PRINTF$(R1%((P%+1)MOD16));
  CHR$4:ENDPROC
570 DEFPROCreel2:VDU4,28,8,15,11,4,31,3,
  11,1Ø,1Ø,1Ø,5:Q%=(Q%+1)MOD16:
  MOVE576,576:PRINTF$(R2%((Q%+1)
  MOD16));CHR$4:ENDPROC
580 DEFPROCreel3:VDU4,28,13,15,16,4,31,3,
  11,1Ø,1Ø,1Ø,5:R%=(R%+1)MOD16:
  MOVE896,576:PRINTF$(R3%((R%+1)
  MOD16));CHR$4:ENDPROC
```

Lines 53Ø and 54Ø add money to the total and make a blip each time a 'coin' is dropped in.

Line 55Ø is a PROCedure to draw a hold box. PLOT1—draw relative—is used so the same routine can be called for all three boxes.

PROCreel starts at Line 56Ø, and starts by defining a text window for the first reel. The pointer, P%, is adjusted also. Lines 57Ø and 58Ø are similar lines for the second and third reels.

WINNING

```
590 DEFFNA:IF(R1%(P%)=R2%(Q%))AND
  (R2%(Q%)=R3%(R%)):=R1%(P%)
600 IFR1%(P%)=R2%(Q%)AND(R1%(P%)=
  6 ORR1%(P%)=5):=1+R1%(P%)
610 IFR1%(P%)=6:=8
620 :=9
630 DEFPROCcheck
640 C%=FNA
650 IFC%=9 ORC%=ØPROCcred(W%(C%)):
  ENDPROC
660 VDU28,11,26,19,23,4
670 COLOUR13Ø:COLOUR13:PRINTTAB(Ø,Ø);
  "£";W%(C%+1)/1ØØ:COLOUR6:PRINTTAB
  (2,1);"£";W%(C%)/1ØØ:COLOUR14:
  PRINTTAB(4,2);"£";W%(C%-1)/1ØØ;
680 A%=GET:IFA%=13CLS:PROCcred(W%
  (C%)):ENDPROC
690 IFA%<>32GOTO68Ø
700 CLS:IFRND(2)=1PROCcred(W%
  (C%+1)):ENDPROC
710 C%=C%-1:IFC%=ØPROCcred(2ØØ):
  ENDPROC
720 GOTO67Ø
```

FNA in Lines 59Ø to 62Ø reads a winning line and points to the amount won in W%. The PROCedure in Lines 63Ø to 65Ø checks if the payout is Ø or £2, and credits the player.

Line 66Ø defines a text window, and Line 67Ø displays the gamble choices. The following lines check if RETURN is pressed, clear the text window, and increase the pointer to W% if the gamble has been won.

NUDGES, HOLDS AND SPINS

```
730 DEFPROCkeys
740 IFRND(4)=1PROCnudges
750 IFRND(5)>2REPEATUNTILGET=32:
  ENDPROC
760 GCOLØ,15:VDU5:FORA%=ØTO2:MOVE
  192+A%*32Ø,384:PROChbox:NEXT:
  H%=TRUE:I%=TRUE:J%=TRUE:*FX15,1
770 A%=GET:IFA%=32VDU4:ENDPROC
780 A%=A%-48:IFA%<1ORA%>4GOTO
  77Ø
790 ONA%GOTO76Ø,8ØØ,81Ø,82Ø
800 H%=FALSE:GCOLØ,4:MOVE192,384:
  PROChbox:GOTO77Ø
810 I%=FALSE:GCOLØ,4:MOVE512,384:
  PROChbox:GOTO77Ø
820 J%=FALSE=GCOLØ,4=MOVE832,384:
  PROChbox:GOTO77Ø
830 DEFPROCnudges:Z%=8:GCOLØ,128:
  COLOUR128
840 VDU19,Z%,8;Ø;:SOUND&11,1,
  2Ø+16*Z%,1:TIME=Ø:REPEATUNTIL
  TIME>5ORINKEY(-99):IFINKEY(-99)
  N%=Z%-7:GOTO87Ø
850 VDU19,Z%,1;Ø;:Z%=Z%+1:IFZ%=13
  Z%=8
860 GOTO84Ø
870 B%=GET-52:IFB%=-4B%=6:ELSE
  IFB%<1ORB%>6GOTO87Ø
880 ONB%GOTO89Ø,9ØØ,91Ø,92Ø,93Ø,94Ø
890 PROCreel1:GOTO95Ø
900 PROCreel2:GOTO95Ø
910 PROCreel3:GOTO95Ø
920 PROCrd1:GOTO95Ø
930 PROCrd2:GOTO95Ø
940 PROCrd3:GOTO95Ø
950 SOUND&11,1,2Ø,2:VDU19,N%+7,1;Ø;
960 IFFNA<9PROCcheck:N%=Ø:ENDPROC
970 IFN%=1N%=Ø:ENDPROC
980 N%=N%-1:VDU19,N%+7,8;Ø;:GOTO
  87Ø
990DEFPROCrd1:VDU4,28,3,15,6,4,3Ø,11,11,
  11,11,5:P%=(P%+15)MOD16:MOVE256,
  864:PRINTF$(R1%((P%+15)MOD16));
  CHR$4:ENDPROC
1000 DEFPROCrd2:VDU4,28,8,15,11,4,3Ø,11,
  11,11,11,5:Q%=(Q%+15)MOD16:MOVE
  576,864:PRINTF$(R2%((Q%+15)MOD
  16));CHR$4:ENDPROC
1010DEFPROCrd3:VDU4,28,13,15,16,4,3Ø,11,
  11,11,11,5:R%=(R%+15)MOD16:MOVE
  896,864:PRINTF$(R3%((R%+15)MOD
  16));CHR$4:ENDPROC
1020 DEFPROCend:COLOUR13Ø:COLOUR3:
  VDU26,12:PRINTTAB(Ø,8);"You ran out of
  money""""""Press the space-bar""" □□□
  for another go"
1030 REPEATUNTILGET=32:ENDPROC
```

Line 74Ø gives a random chance of getting nudges—PROCnudge looks after the nudges. Similarly, Line 75Ø gives a random chance of holds. Line 76Ø flashes the hold boxes and resets the hold flags. The reels are spun by Line 77Ø if SPACE is pressed.

The hold flags are set in Lines 8ØØ to 82Ø, and the appropriate hold box(es) drawn. Line 83Ø is the start of the nudge routine. Lines 84Ø to 86Ø flash the nudge lights until SPACE is pressed. N% is set to the number of nudges.

If a win is possible, Line 96Ø goes into the payout/gamble routine, and finish the nudge routine. The nudge routine also finishes if Line 97Ø finds that the nudges have been exhausted. If nudges still remain, N% is decremented, and the next light is flashed.

Line 99Ø defines a text window for reel one, and scrolls it down three lines to move the fruit off the bottom. The pointer is decremented, and the next fruit is displayed on the reel. The second and third reels are spun by Lines 1ØØØ and 1Ø1Ø.

If you have a disk drive, type *TAPE, then FOR A% = Ø TO & 18ØØ:? (&EØØ + A%) = ? (PAGE + A%): NEXT: PAGE = & EØØ RETURN OLD RETURN before you run it.

THE MAIN LOOP

```
350 M=100:H=-1:I=-1:J=-1:
    P=RND(16)-1:Q=RND(16)-1:
    R=RND(16)-1
360 SCREEN1:GOSUB1000:GOSUB2000:
    GOSUB2500:IF M>Ø THEN 360
370 CLS:PRINT@101,"YOU RAN OUT OF
    MONEY"
380 PRINT@417,"PRESS SPACE FOR
    ANOTHER GO"
390 IF INKEY$<>"□" THEN 390
    ELSERUN
```

Line 35Ø sets the money held equal to $1, sets the hold flags to −1, and chooses positions for the three reels. Line 36Ø is the heart of the program, calling subroutines which spin the reels, PUT the fruit on screen, enable the player to have the option to gamble or take the win, or hold.

If the player's money drops below zero, Line 37Ø displays YOU RAN OUT OF MONEY.

PUTTING FRUIT ON SCREEN

```
500 ON CH+1 GOTO 540,530,560,550,570,
    510,520
510 PUT(XX,YY)-(XX+31,YY+15),B,PSET:
    RETURN
520 PUT(XX,YY)-(XX+31,YY+15),C,PSET:
    RETURN
530 PUT(XX,YY)-(XX+31,YY+15),A,PSET:
    RETURN
540 PUT(XX,YY)-(XX+31,YY+15),BR,
    PSET:RETURN
550 PUT(XX,YY)-(XX+31,YY+15),S,PSET:
    RETURN
560 PUT(XX,YY)-(XX+31,YY+15),PL,
    PSET:RETURN
570 PUT(XX,YY)-(XX+31,YY+15),P,PSET:
    RETURN
1000 M=M-10:FORL=1 TO
    RND(3)+RND(3)
1010 IF H GOSUB 1520
1020 IF I GOSUB 1530
1030 IF J GOSUB 1540
1040 NEXT:IF H THEN SOUND 100,1
1050 FOR L=1 TO RND(3)+RND(3)
1060 IF I GOSUB 1530
1070 IF J GOSUB 1540
1080 NEXT:IF I THEN SOUND 120,1
1090 FORL=1 TO RND(3)+RND(3)
1100 IF J GOSUB 1540
1110 NEXT:IF J THEN SOUND 140,1
1120 H=-1: I=-1: J=-1: RETURN
1500 CLS:IF D=Ø THENRETURN ELSEPRINT
    @166,USING"CREDIT□=□$$###.
    ##";M/100:FORA=10TOD STEP10:
    M=M+10:PRINT@166,USING"CREDIT
    □=□$$###.##";M/100
1510 SOUND200,1:FORB=ØTO400:NEXT
    B,A:RETURN
1520 P=(P-1)AND15:XX=48:YY=28:FOR
    G=P-1TOP+1:CH=R1(15ANDG):
    GOSUB500:YY=YY+32:NEXT:RETURN
1530 Q=(Q-1)AND15:XX=112:YY=28:
    FORG=Q-1 TO Q+1:CH=R2(15AND
    G):GOSUB500:YY=YY+32:NEXT:
    RETURN
1540 R=(R-1)AND15:XX=176:YY=28:
    FORG=R-1 TO R+1:CH=R3(15AND
    G):GOSUB500:YY=YY+32:NEXT:
    RETURN
1550 C=9:IF (R1(P)=R2(Q))AND
    (R2(Q)=R3(R)) THEN C=R1(P):RETURN
1560 IF R1(P)=R2(Q) AND (R1(P)=6 OR
    R1(P)=5) THEN C=1+R1(P):RETURN
1570 IF R1(P)=6 THEN C=8
1580 RETURN
```

Lines 5ØØ to 57Ø PUT the correct fruit on screen. Lines 1ØØØ to 112Ø spin the reels that are not being held. The reels are spun by calling Lines 152Ø to 154Ø. Lines 155Ø to 158Ø check if the reels end up displaying a winning combination, and Line 15ØØ adds the winnings to the money total.

GAMBLES, HOLDS AND NUDGES

```
2000 GOSUB1550
2010 IF C=9 OR C=Ø THEND=W(C):
    GOSUB1500:RETURN
2020 CLS9-C:PRINT@265,"gamble";:PRINT
    @278,USING"$$#.##□";W(C)/100;
2030 PLAY"L4T20B":PRINT@212,USING"$$
    #.##□";W(C-1)/100;:PRINT@340,
    STRING$(7,271-C*16);
2040PLAY"T20C":PRINT@212,STRING$(7,
    271-C*16);:PRINT@340,USING"$$#.
    ##□";W(C+1)/100;
2050 R$=INKEY$:IF R$<>"□" AND
    R$<>CHR$(13) THEN 2030
2060 IF R$=CHR$(13) THEN CLS:D=W(C):
    GOSUB1500:RETURN
2070 IF RND(2)=1 THENCLS:D=W(C+1):
    GOSUB1500:RETURN
2080 C=C-1:IF C=Ø THEND=200:
    GOSUB1500:RETURN
2090 GOTO2020
2500 IFRND(4)=1 GOSUB3060:GOTO2550
2510 IFRND(5)<3 THEN 2560
2520 FORK=1TO2000:NEXT:SCREEN1,Ø
2530 A$=INKEY$:IF A$<>"□" AND
    A$<>"C" THEN 2530
2540 IF A$="□" THEN RETURN
2550 CLS:PRINT@166,USING"CREDIT
    □=□$$###.##";M/100:GOTO
    2520
2560 SCREEN1,Ø:H=-1:I=-1:J=-1
2570 IF H THEN PUT(38,122)-(91,143),H,
    NOT
2580 IF I THEN PUT(102,122)-(155,143),H,
    NOT
2590 IF J THEN PUT(166,122)-(219,143),H,
    NOT
2600 R$=INKEY$:IF R$="□" THENFOR
    K=ØTO2:PUT(38+64*K,122)-
    (91+64*K,143),H,PSET:NEXT:RETURN
3000 IF R$<"1" OR R$>"4" THEN 2570
3010 ON VAL(R$) GOTO 3020,3030,3040,
    3050
3020 H=-1:I=-1:J=-1:GOTO2570
3030 H=Ø:PUT(38,122)-(91,143),H,PSET:
    GOTO2570
3040 I=Ø:PUT(102,122)-(155,143),H,PSET:
    GOTO2570
3050 J=Ø:PUT(166,122)-(219,143),H,
    PSET:GOTO2570
3060 SCREEN1,Ø:COLOR4,2:PUT(159,156)-
    (224,170),H,NOT:PLAY"L40T10"
3070 K=1
3080 LINE(10+K*16,158)-(21+K*16,169),
    PRESET,BF
3090 IF INKEY$="□" THEN 3120
3100 K=K+1:PLAYSTR$(K*2):IF K<6
    THEN 3080
3110 FORK=1TO5:LINE(10+K*16,158)-
    (21+K*16,169),PSET,BF:NEXT:GOTO3070
3120 N=K:PUT(159,156)-(224,170),H,NOT
3130 R$=INKEY$:IF (R$<"5" OR R$>
    "9") AND R$<>"Ø" THEN 3130
3140 IF R$="Ø" THEN R$="10"
3150 ON VAL(R$)-4 GOTO 3160,3170,3180,
    3190,3200,3210
3160 P=P+2:GOSUB1520:GOTO3220
3170 Q=Q+2:GOSUB1530:GOTO3220
3180 R=R+2:GOSUB1540:GOTO3220
3190 GOSUB1520:GOTO3220
3200 GOSUB1530:GOTO3220
3210 GOSUB1540
3220 SOUND40,1:GOSUB1550:IF C<9
    GOSUB2010:N=Ø:GOTO3250
3230 IF N=1 THEN N=Ø:GOTO3250
3240 LINE(10+N*16,158)-(21+N*16,169),
    PSET,BF:N=N-1:GOTO3130
3250 FORK=1TO5:LINE(10+K*16,158)-
    (21+K*16,169),PSET,BF:NEXT:RETURN
```

Lines 2Ø1Ø to 2Ø5Ø are the gamble routine. The player has the choice of collecting the money or gambling it against a larger win.

The hold routine is to be found between Lines 253Ø to 3Ø5Ø, and is called from Line 251Ø. Flags are set in the hold routine according to the player's key presses.

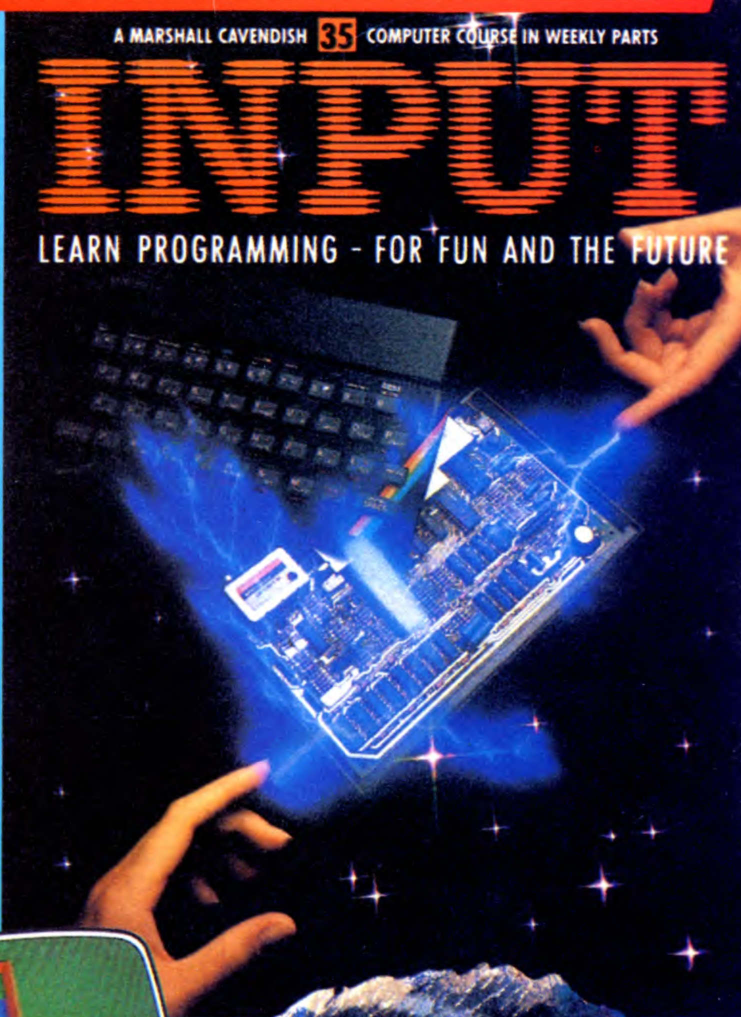
Lines 3Ø6Ø to 325Ø are the nudge routine. The player is given nudges if the random number chosen in Line 25ØØ is one. The routine reads the player's keyboard input, and moves the reels accordingly.

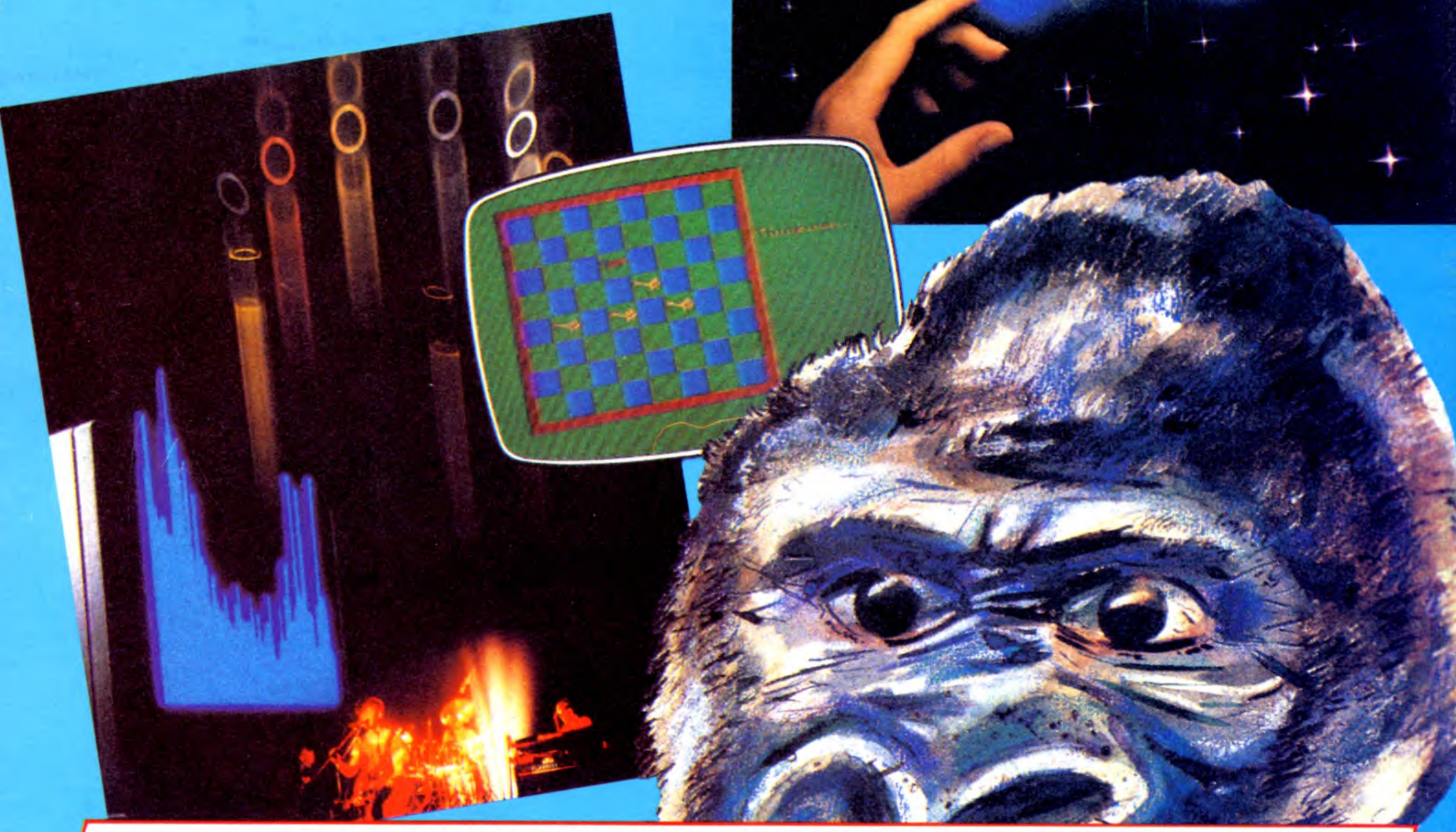
CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

COMING IN ISSUE 35...

- You can almost hear the cogs grating as you start teaching your computer to think in the FOX and GEESE game
- Willie's getting impatient after all these weeks of waiting, so start him off on his picnic in CLIFFHANGER
- HOW BASIC PROGRAMS ARE STORED reveals the inner workings of microcomputers
- INPUT's SOUND ANALYZER is an excellent introduction to up-to-the-minute digital recording
- Visit the shrink's and expand your ego with the DRAWING AID program. Draw or inspect fine detail on screen by simple key presses
- Impatient astronauts can enter the complete LUNAR LANDER game





ASK YOUR NEWSAGENT FOR INPUT