DK

ScreenShot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

AND

ZX SPECTRUM ZX SPECTRUM+

GRAPHICS

BOOK FOUR

Advanced sprite programming techniques plus a full-colour design directory

PIERS LETCHER

DK

Screen Shot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

ZX SPECTRUM ZX SPECTRUM+

GRAPHICS

THE DK SCREEN-SHOT PROGRAMMING SERIES

Books One and Two in the DK Screen-Shot Programming Series brought to home computer users a new and exciting way of learning how to program in BASIC. Following the success of this completely new concept in teach-yourself computing, the series now carries on to explore the speed and potential of machine-code graphics. Fully illustrated in the unique Screen-Shot style, the series continues to set new standards in the world of computer books.

BOOKS ABOUT THE ZX SPECTRUM+

This is Book Four in a series of guides to programming the ZX Spectrum+. It contains a complete machine-code sprite-programming course for the Spectrum+, and features its own sprite editor which enables you to design and store sprites directly from the keyboard. Together with its companion volumes, it builds up into a complete programming and graphics system.

ALSO AVAILABLE IN THE SERIES

Step-by-Step Programming for the **Commodore 64**

Step-by-Step Programming for the **BBC Micro**

Step-by-Step Programming for the **Acorn Electron**

Step-by-Step Programming for the **Apple IIe**

Step-by-Step Programming for the **Apple IIc**

PIERS LETCHER

After graduating with a degree in Computer Systems, Piers Letcher has worked in many areas of the computer industry, from programming and selling mainframes to designing and marketing educational software. He was Peripherals Editor of *Personal Computer News* until May 1984 and has since written a guide to peripherals and a number of other books for popular home micros.

BOOK FOUR


sinclair
ZX Spectrum +
TRUE VIDEO
INV VIDEO
BLUE DEF FN 1
RED FN 2
MGNTA LINE 3
GREEN OPEN # 4
CYAN CLOSE # 5
YELLW MOVE 6
WHITE ERASE 7
POINT 8
CAT 9
BLACK FORMAT 0
PEEK OUT POKE O
TAB PRINT P
CODE IN INPUT AT I
CHR$ IF OR U
STR$ RETURN AND Y
RND MERGE RAND T
INT VERIFY RUN R
TAN ATN REM E
COS ACS DRAW W
SIN ASN PLOT Q
DELETE
GRAPH
LEN SCRN$ LIST K
USR ATTR LET L
VAL VAL$ LOAD J
SQR CIRCLE GOSUB H
ABS GOTO THEN G
SGN FOR TO F
DATA DIM STEP D
RESTR SAVE NOT S
READ NEW STOP A
EXTEND MODE
EDIT
PI INVERS PAUSE M
INKY$ OVER NEXT N
BIN BRIGHT BORDER B
LLIST FLASH CLS V
LPRINT PAPER CONT C
EXP INK CLEAR X
LN BEEP COPY Z
CAPS LOCK
CAPS SHIFT
SYMBOL SHIFT

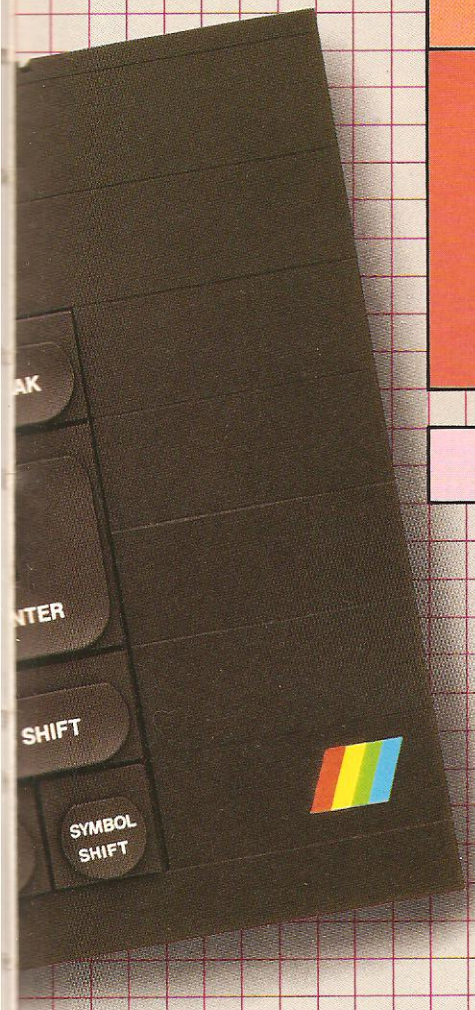
DK

Screen Shot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

ZX SPECTRUM ZX SPECTRUM+ GRAPHICS



PIERS LETCHER

DORLING KINDERSLEY·LONDON

BOOK FOUR

CONTENTS

The DK Screen-Shot Programming Series was conceived, edited and designed by Dorling Kindersley Limited, 9 Henrietta Street, Covent Garden, London WC2E 8PS.

Editor Michael Upshall
Designer Steve Wilson
Photographer Vincent Oliver
Series Editor David Burnie
Series Art Editor Peter Luff
Managing Editor Alan Buckingham

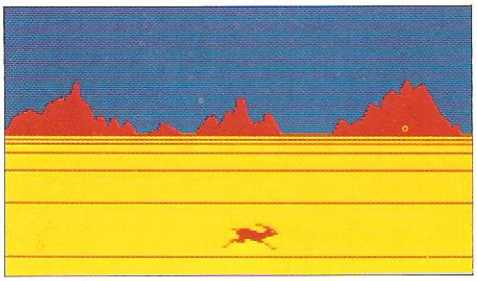
First published in Great Britain in 1985 by Dorling Kindersley Limited, 9 Henrietta Street, Covent Garden, London WC2E 8PS.
Second impression 1985

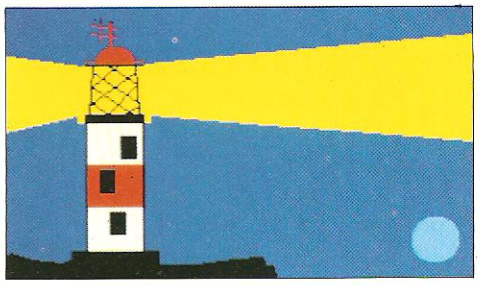
British Library Cataloguing in Publication Data

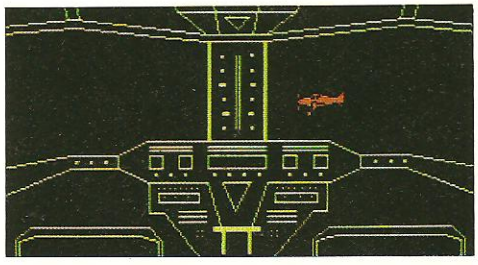
Letcher, Piers
Step-by-step programming ZX Spectrum and ZX Spectrum+ Graphics.
– (DK screen shot programming series) Bk. 4
1. Sinclair ZX Spectrum (Computer) - Programming
I. Title
001.64'2 QA76.8.S625

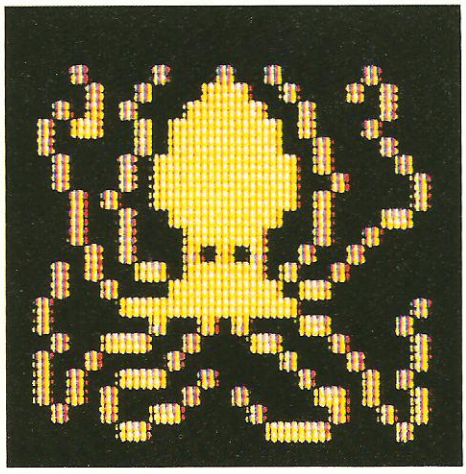
ISBN 0-86318-104-X

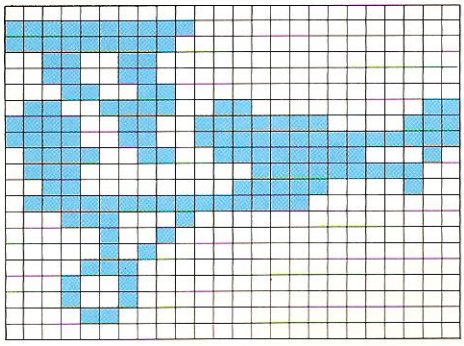
Typesetting by Gedset Limited, Cheltenham, England
Reproduction by Reprocolor Llovet S.A., Barcelona, Spain and F. E. Burman Limited, London
Printed and bound in Italy by A. Mondadori, Verona

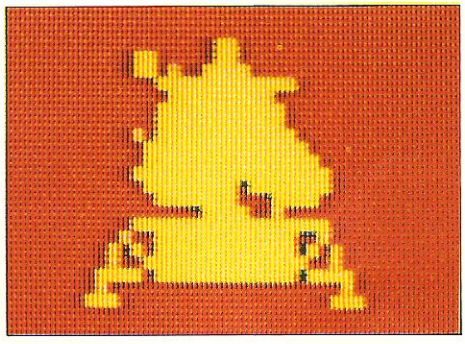


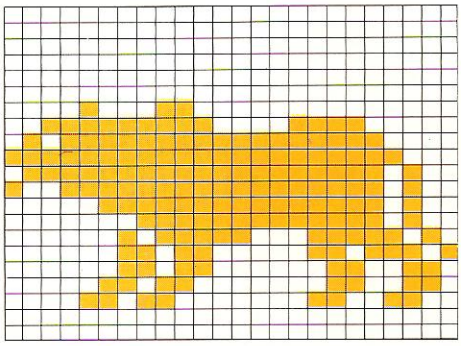


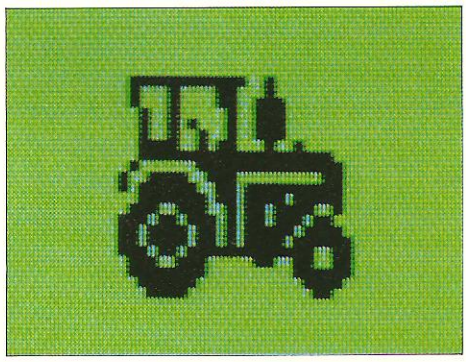


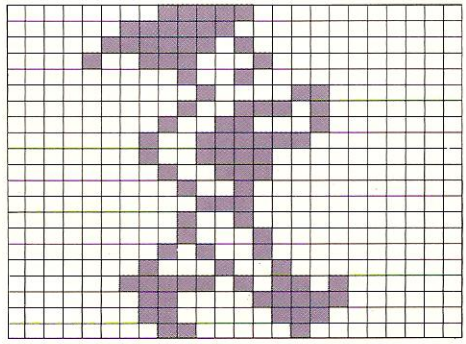


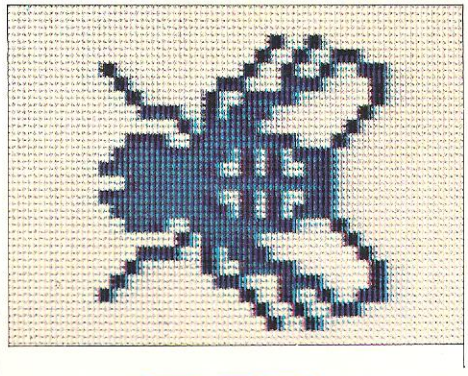


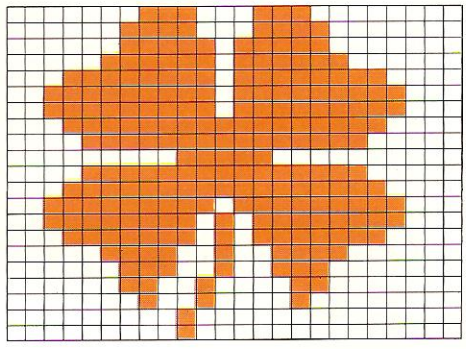












ABOUT THIS BOOK

The Sinclair Spectrum is one of the most popular microcomputers ever produced. One reason for its success has been its remarkable ability to produce graphic displays rivalling those produced by much larger computers designed only ten or fifteen years ago. However, graphics programming in BASIC under-utilizes the Spectrum. To produce the kind of displays seen in commercially available games, you need to use machine code as well as BASIC.

What is machine code?

The heart of the Spectrum, the Z80 central processor, cannot understand BASIC. A BASIC program must first be translated into a simpler language that the machine can understand (hence the term "machine code"). This code is in the form of binary 1s and 0s. Before the processor can execute a BASIC program line, all keywords and variables are first converted to machine-code instructions.

BASIC is an example of what is known as an "interpreted", as opposed to a "compiled", language — that is, it is executed by the central processor line by line rather than as a complete program. While an interpreted language is easier to use, it is also slower in execution. By writing programs in machine code, you can miss out the BASIC interpreter altogether. In addition, machine code allows you to utilize many features of your Spectrum which cannot be reached from BASIC, so that you can therefore achieve far more impressive results than would ever be possible from the simpler, but more restricted, BASIC. You can get an idea of how much faster machine code is by seeing the time taken for the programs in this book to run.

Disadvantages of machine code

Given all the advantages of machine code in both speed and flexibility, why not ignore BASIC and use machine code all the time? The answer is simply convenience. Using machine code is time-consuming, difficult and frustrating, and attempting to write your own code is only for the expert. When you see machine-code listings, they are usually in a "disassembled" form, that is, with some of the numbers translated into mnemonics such as LD for LOAD, and JP for JUMP. But a special disassembler program is required simply to give you a machine-code listing in this form, and these mnemonics are themselves far from simple. Using machine code even the simplest operations in BASIC, such as drawing a line on the screen, require many lines of programming. In addition, machine code has no error-trapping routines such as those in BASIC. If a mistake is made when keying in a BASIC program, the program will not be lost (although the program may refuse to RUN at some point); in machine code, without error-trapping routines, a mistake will probably cause the Spectrum to crash, with the result that both the program and its DATA are lost.

The solution

This book combines the advantages of machine code with the convenience and simplicity of BASIC. This is done by giving the machine code in the form of ready-made and tested routines, which you can then use in your BASIC programs. The machine code is shown as DATA statements in BASIC, which means it isn't necessary for you to understand anything about machine code to be able to use the routines. The DATA is given in the form of decimal numbers, rather than in binary or hexadecimal (to base 16), so that the machine code is in the form most convenient for you to key in.

The machine-code routines

The screen below shows an example of a machine code routine (the double vertical sprite routine, FNj, given on page 17).

DOUBLE VERTICAL SPRITE ROUTINE

```
7450 LET b=52100: LET l=225: LET
 z=0: RESTORE 7460
7451 FOR i=0 TO l-1: READ a
7452 POKE (b+i),a: LET z=z+a
7453 NEXT i
7454 LET z=INT (((z/l)-INT (z/l
))*l)
7455 READ a: IF a<>z THEN PRINT
"??": STOP

7460 DATA 0,42,11,92,17
7461 DATA 4,0,25,78,30
7462 DATA 8,25,70,25,126
7463 DATA 230,3,50,103,204
7464 DATA 25,126,50,104,204
7465 DATA 25,126,230,1,50
7466 DATA 105,204,25,126,230
7467 DATA 1,50,101,204,25
7468 DATA 126,17,63,0,33
7469 DATA 9,213,25,61,32

7470 DATA 252,175,50,102,204
7471 DATA 205,93,210,205,86
```

Each routine in the book is shown like this, in the form of a BASIC program. The machine code is contained as a series of DATA statements in lines 7460 onwards. At the beginning of the routine, in lines 7450 to 7455, there are a few lines of BASIC. This is a loader program; variable b tells the computer where in memory to begin loading the routine, and variable l the number of bytes in the routine. When the loader routine is RUN, this routine is placed in memory from address 52100 onwards, and has a total length of 225 bytes.

As shown here, of course, the routine is simply a list of numbers, and has no visible meaning. These numbers are the ready-tested and assembled machine code which has then been converted to a sequence of decimal numbers. Each number corresponds to a single instruction or item of DATA required by the routine;

hence, all the numbers have values between 0 and 255, the maximum range of a byte. All you need to know about the routine is what it does and what information it requires so that you can call it correctly from your BASIC program.

All the routines in the book are defined as functions. Each function is individually coded by the letters a to o; a complete list of functions is given on pages 62-63. Demonstration BASIC programs can be found on the same page as each routine; these give you an indication of the kind of displays which are possible using the machine code.

How to use the routines

To use any program in this book, simply key in a machine-code routine together with a BASIC program which demonstrates its use. You will find full details of how to do this on pages 8-9. When you RUN the program, you will begin to see the true power of your Spectrum.

As you progress through the book and the range of routines grows, the BASIC programs grow too by calling several routines to produce increasingly complex displays. By keying in each routine, and then SAVEing it onto cassette or Microdrive, you will have a sophisticated but flexible graphics capability at your fingertips.

The programs in use

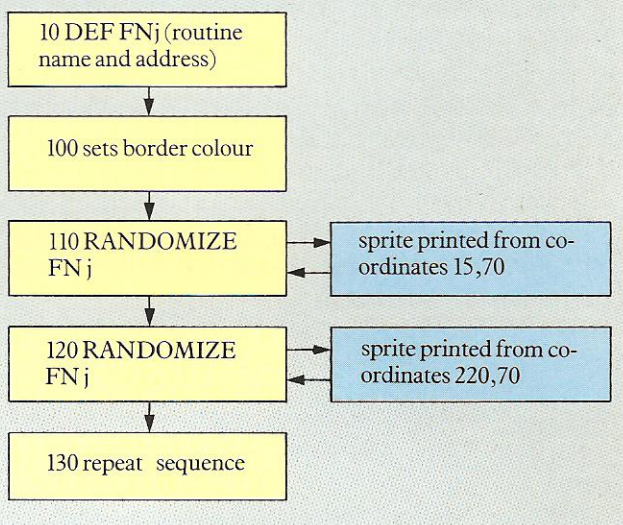
A typical program from this book (the unicycle program on page 23) contains two details which will be unfamiliar to BASIC programmers who have not used machine code before:

UNICYCLE PROGRAM

```
 10 DEF FN j(x,y,d,l,s,c,n)=USR
52100
100 BORDER 2
110 RANDOMIZE FN j(15,70,1,69,0
,0,1)
120 RANDOMIZE FN j(220,70,0,70,
0,0,3)
130 GO TO 110



0 OK, 0:1
```

First, you will see in line 10 a DEF FN statement, which is used to instruct the computer that a machine-code routine with two parameters (x and y) is located at address 52100 in memory. You will also notice two RANDOMIZE FN commands (lines 110 and 120). These are the calls to the double vertical sprite routine, and the numbers in brackets which follow them are the parameter values to be passed to the machine-code routine (in this case, the start co-ordinates of the sprite, its direction, how far it is to move and various other instructions). When RUNning, the program is carried out by the computer in this way:

HOW THE UNICYCLE PROGRAM WORKS



On the left side of this diagram is the main BASIC program, and on the right you can see the machine-code routines, called twice using a RANDOMIZE FN statement. You will see from the diagram that the machine-code is used here very much as a subroutine would be used in BASIC, with variables passed to the routines each time they are called.

What the routines do

The routines in this book free you from the limitations of programming in BASIC. By using the machine-code given here, you will be able to create and control sprites, to control them on the screen and to animate them, and to scroll both the entire screen and defined areas of it.

In addition, two of the later routines provide an introduction to one of the most exciting aspects of machine-code graphics: interrupt-driven routines, which operate independently of BASIC, and which enable you to program your Spectrum to carry out several tasks simultaneously.

Creating and editing sprites

To make sprites even easier to use, a directory of over 200 sprites is included from pages 36 to 61. These sprites can be keyed in and then edited with the sprite editor routine and program, given on pages 11 to 13. Using the sprite editor, you will find it easy to make your own versions of the sprites given in this book. Using single-key commands, for example, you can invert the sprites, make them face another direction, or turn them upside down.

USING THE MACHINE CODE

The machine-code routines in this book can easily be incorporated into your BASIC programs without you having to understand the intricacies of how they work. Simply choose a program from this book, and follow the steps given here.

1: CLEAR memory

As soon as you switch on your Spectrum, type CLEAR 49000. This command resets RAMTOP, the top of the area in memory free for BASIC programs, and ensures that BASIC programs cannot overlap with the machine code stored in memory from 49200 upwards. Now you can safely use NEW to delete BASIC programs without losing any of the machine code in memory.

Remember to use CLEAR before loading machine code, since this command erases whatever is in memory above the specfied address.

2: Load the machine code

Now type in whatever machine-code routines are required by the BASIC program. After keying in the routine, RUN the short BASIC program which accompanies it; this loads the code into memory. If you keyed in the DATA correctly, you will see an "OK" message on the screen; if not, you will see a couple of question marks. In this case, look again at what you have typed in to trace the mistake.

3: SAVE the routine

When you are sure you have keyed in the routine correctly, SAVE it onto cassette or Microdrive. Always SAVE machine code before using it, to minimize the risk of losing everything you have keyed in. When BASIC errors occur, an error message is usually produced but the program is not lost. Machine-code routines, however, do not generally have error-trapping facilities, and a fault in the code will as often as not cause the Spectrum to crash — deleting everything in memory.

The machine code can be SAVEd in two ways: either in the form of DATA statements like any other BASIC

EXPLANATION OF A MACHINE-CODE BOX

FNf — Routine title

SPRITE PRINT ROUTINE — Name of routine

Address in memory at which routine is located

Start address 54100 **Length** 75 bytes — Number of bytes in memory taken up by routine

Other routines called Sprite editor routines (FNa-FNe).

What it does Prints a single sprite on the screen at a specifed point. — Purpose of routine

Using the routine This routine displays any single sprite from the sprite buffer. The routine does not move the sprite. Note that if the sprite is too far to the right of the screen it will reappear one character below on the left-hand side of the screen, since the Spectrum PRINT routine is used in the transfer of memory to screen. — Points to note when using the routine

Number of parameters used by the routine, and letters used to describe these parameters

ROUTINE PARAMETERS — What the parameters do

DEF FNf(x,y,n)

x,y	specify print position (x<29, y<21)
n	specifies number of sprite (1-10)

Maximum and minimum values of parameter to ensure the routine does not plot off-screen points

BASIC loading routine for the machine-code DATA

ROUTINE LISTING

```
7900 LET b=54100: LET l=70: LET
z=0: RESTORE 7910
7901 FOR i=0 TO l-1: READ a
7902 POKE (b+i),a: LET z=z+a
7903 NEXT i
7904 LET z=INT ((z/l)-INT (z/l)
)*l)
7905 READ a: IF a<>z THEN PRINT
"??": STOP

7910 DATA 42,11,92,1,4
7911 DATA 0,9,86,1,8
7912 DATA 0,9,94,237,83
7913 DATA 148,211,9,126,50
7914 DATA 150,211,123,230,24
7915 DATA 246,64,103,123,230
7916 DATA 7,183,31,31,31
```

Start address for POKEing DATA

Number of machine-code bytes (without check digit)

POKEs byte value a into location (b+i)

Calculates check digit z

READs next DATA item, the routine check digit; if this is not the same as z, two question marks are PRINTed to show a mistake has been made

Start of machine-code DATA

listing, or, after you have loaded it into memory, as a block of code. To save machine code, type:

```
SAVE "routine name" CODE start address, length in bytes
```

The start address and length are given at the top of each machine-code box. The diagram on the facing page shows how this information is displayed.

4: LOAD a BASIC program

With the machine-code routine in memory, you can now use it in a BASIC program. DEF FN statements are used to tell the Spectrum the whereabouts of the routine in memory, and what information it requires.

Using functions

A machine-code routine can be called simply by specifying its start location, like this:

```
10 RANDOMIZE USR 54100
```

A line like this in a BASIC program, however, is not very informative. It tells you neither what the routine does, nor how many parameters the routine may require when called. This information could be POKEd into the appropriate memory locations – but the consequences of a mistake could be disastrous. Much more reliable is to pass information to the routines using a BASIC function. Functions on the Spectrum are identified by a single letter, and are followed by parameters in brackets. When you define the name and location of the function in your program, you must also specify the parameters, if any, which are to be passed to the routine. For example, the sprite print routine, FNf, requires three parameters:

```
10 DEF FN f(x,y,n)=USR 54100
```

Which letters are used after DEF FN is not important; their function is only to tell the computer the number of parameters which will follow the routine call in a BASIC program.

A machine-code function can be called from BASIC in two main ways, both of which require you to combine the keywords FN or USR with a BASIC keyword. The method used generally in this book is with the keyword RANDOMIZE. Thus,

```
20 RANDOMIZE FN f(10,10,1)
```

would display the first sprite from the sprite buffer in memory at co-ordinates 10,10. Note that using RANDOMIZE also resets the random number generator with a new seed; this may cause problems if you are also using a random function in your program. The second word you can use to call machine code is RESTORE. However, RESTORE also resets the pointer to DATA statements when you use it — which is of course the purpose of the RESTORE statement. If you opt to use RESTORE instead of RANDOMIZE then be especially careful if there are any READ or DATA statements in your program.

QUESTIONS AND ANSWERS

What if I make a mistake in keying in?
Don't panic! Nobody keys in long lists of numbers without making any mistakes. A check routine is included with each machine-code routine to warn you if you made any mistakes in keying in the DATA. This routine compares the DATA you have entered with a check number, which is placed by itself on the last DATA line of each routine.

After the loading program has POKEd the DATA numbers into memory, it looks to see if the check digit is the same as the one currently calculated. If the two numbers are different, the program prints two question marks to show an error has been made. If this happens, look through the numbers you have typed in to find the mistake. Having corrected the error, you may still find that the routine fails to load correctly; look to see if you have made more than one error.

Can I start anywhere in the book?
Yes, you can start on any page, but obviously when you key in a program it will not RUN unless the machine-code routine it calls is present in memory. Check before you begin if the program you want to RUN calls any machine-code routines you haven't already keyed in.
If you key in all the routines in this book as BASIC DATA statements, you will find there isn't room in memory to store them all. By loading each routine as machine code as soon as you have keyed it in, you can avoid this happening. In the form of machine code, you can, of course, use any of the Book Four routines together, as well as any routines from Book Three – the routines will not overlap in memory.

Can I adapt the BASIC programs?
Yes. You can edit the BASIC programs in any way you want to produce different displays, and you will find suggestions for variations throughout the book. One suggestion, though, if you are going to experiment with unusual or off-screen values for the machine-code parameters, is to SAVE what you have keyed in before experimenting. This will prevent you from losing hours of work at the keyboard!

Can I adapt the machine-code routines?
Yes, but at your own risk! Without a good understanding of machine code, it is highly unlikely that you will be able to alter any of the routines successfully. Much more probable is that the Spectrum would crash, with the result that both program and code are wiped from memory.

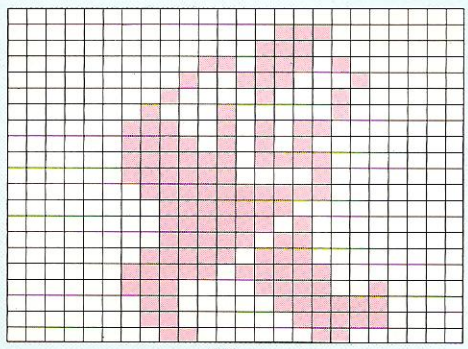
WHAT ARE SPRITES?

Most of the computer graphics you have created up to now have probably been stationary rather than moving, though you will have seen all sorts of moving graphics in arcade games and in commercial Spectrum software. Before you can create moving graphics for yourself, however, it is necessary to look at the ideas behind movement.

What is movement?

You tell something is moving if its position changes relative to something else. You know a train going past a window is moving, because the window is still. This book is about creating movement, and displaying moving objects. The objects to be moved are called sprites: objects which can move over a background without destroying it. The diagram below shows a single sprite..

EXAMPLE OF A SPRITE



Creating movement

The problem in creating movement is not so much in making something move as in making it move smoothly. You probably know that one way of getting something to move in BASIC on your Spectrum is to PRINT an object on the screen, wipe it off again, and PRINT it again in quick succession. This method has several disadvantages, the most important of which you will notice as soon as you try it out — the movement looks jerky. This is because there will be a space of one character between each position where the object is PRINTed. The other problem with BASIC is that it is simply not fast enough in operation to be used for smooth movement.

The jump of eight pixels between each position of the object is easily visible, and the obvious solution is to print the object every pixel rather than every character. This is easier said than done, however, since printing objects across pixel boundaries requires the step from BASIC to machine code. Using machine code will also give you the increase in speed which is necessary for implementing smooth movement.

To use movement effectively you must first make a few decisions on what you plan to move around the screen. Firstly, you must decide the size of object you want to move. The most obvious choice is a single character (8 by 8 pixels), but this looks very small in screen displays. On the other hand if you pick a size which is too large you will have the problem of trying to move thousands of pixels at the same time, resulting in a very jerky effect. The solution presented here is to use a shape 24 pixels wide by 21 pixels deep — or in character terms a little under three by three characters. To create a practical illusion of movement you must also make sure that the object to be moved does not destroy the background over which it passes.

Ways of implementing sprites

Some computers have sprites built into them as part of the machine hardware. The Commodore 64, for example, has a sophisticated chip dedicated to the implementation of sprites, since the method used to display them is so complicated. The chip works by saving the area of the screen underneath the sprite position (a block 24 by 21 pixels), printing the sprite, wiping off the sprite, printing the sprite one pixel further on and then replacing that part of the background which was uncovered by the movement. Though it would be possible to implement this process on the Spectrum, it would be very slow, or alternatively it would require a very long routine with many hundreds of bytes of DATA to be typed in.

An alternative method of implementing sprites uses a technique that you have probably used quite a lot already, which is to print onto the screen using Exclusive/OR plotting. If you do this with sprites you do not have to worry about the background, as it will remain unchanged, with the sprite appearing to move across the background without interference.

SPRITE SCREEN DISPLAY

THE SPRITE EDITOR 1

In order to use sprites, you need some means of creating them, and you need a location in memory where a number of them can be stored for future reference. This is the purpose of the sprite editor program. The program allows you to design and edit sprites on the screen, and stores them in memory for use by the sprite routines. Each sprite consists of 504 pixels, and is stored in 63 bytes of the Spectrum memory.

The sprite editor allows ten sprites to be defined at a time, and gives you the option of transferring sprites from one location in the sprite buffer to another. In addition, sprites can be flipped horizontally and vertically, and inverted (by switching the ink and paper attributes). The program also allows you to load in previously edited sprites from tape, and to save the current batch to tape for future use.

The sprite editor is a combination of BASIC and machine code. The code comprises six different routines, all linked together. The purpose of each routine is explained below.

The sprite editor routines

Routine FNa, at address 54200, is the base routine for the editor. It converts the large grid on the screen to the small display of the current sprite you see on the right. The current sprite is temporarily stored in a buffer, and the routine converts each dot in the buffer to a square on the screen.

When you want to save the current sprite, you have to decide which of the ten sprite positions you are saving to. A routine at 54317 transfers the sprite stored in the buffer to the appropriate location in the sprite table in memory. Routine FNb, at address 54353, carries out the reverse of this operation, transferring a sprite from its position in the sprite table to the sprite editor buffer. Routine FNa is then called again to display the sprite.

SPRITE EDITOR PROGRAM

```
  10 DEF FN a()=USR 54200
  20 DEF FN b()=USR 54353
  30 DEF FN c()=USR 54422
  40 DEF FN d()=USR 54436
  50 DEF FN e()=USR 54482
 100 LET a=22529: LET j=0: LET s
=a: LET o=s
 110 GO SUB 1200
 120 GO TO 560
 130 POKE o,233
 140 LET g$=INKEY$
 150 RANDOMIZE FN a()
 160 PAUSE 1.5
 170 LET o=s: LET j=PEEK o
 180 IF g$="I" THEN RANDOMIZE FN
 c(): RANDOMIZE FN a()
 190 IF g$="R" THEN RANDOMIZE FN
 d(): RANDOMIZE FN a()
 200 IF g$="U" THEN RANDOMIZE FN
 e(): RANDOMIZE FN a()
 210 IF g$="Z" THEN GO TO 480
 220 IF g$<"4" OR g$>"8" THEN GO

scroll?
```

```
 TO 240
 230 GO TO 370
 240 IF g$<"%" OR g$>"(" THEN GO
 TO 140
 250 IF g$="%" THEN LET s=s-1
 260 IF g$="&" THEN LET s=s+32
 270 IF g$="'" THEN LET s=s-32
 280 IF g$="(" THEN LET s=s+1
 290 GO SUB 1160
 300 IF l=0 THEN LET s=s+1: GO T
O 140
 310 IF l>24 THEN LET s=s-1: GO
TO 140
 320 IF s<a THEN LET s=s+32: GO
TO 140
 330 IF s>a-1+21*32 THEN LET s=s
-32: GO TO 140
 340 POKE o,0
 350 POKE s,130
 360 GO TO 140
 370 IF g$="5" THEN LET s=s-1
 380 IF g$="6" THEN LET s=s+32

scroll?
```

SPRITE EDITOR PROGRAM CONTD.

```
 390 IF g$="7" THEN LET s=s-32
 400 IF g$="8" THEN LET s=s+1
 410 GO SUB 1160
 420 IF l=0 THEN LET s=s+1: GO T
O 140
 430 IF l>24 THEN LET s=s-1: GO
TO 140
 440 IF s<a THEN LET s=s+32
 450 IF s>a-1+21*32 THEN LET s=s
-32
 460 POKE o,56
 470 POKE s,135: GO TO 140
 480 PRINT AT 1,26;"SAVE "
 490 PRINT AT 3,26;"a-j"
 500 LET g$=INKEY$
 510 IF g$<"a" OR g$>"j" THEN GO
 TO 500
 520 LET v=CODE g$-96: POKE 2330
0,v
 530 RANDOMIZE USR 54317
 540 BEEP .4,20
 550 GO SUB 1000

scroll?
```

```
 550 PRINT AT 1,26;"E a-j"
 570 PRINT AT 3,26;"1 SAVE"
 580 PRINT AT 5,26;"2 LOAD"
 590 LET g$=INKEY$
 600 IF g$="1" THEN GO TO 1100
 610 IF g$="2" THEN GO TO 1130
 620 IF g$<"a" OR g$>"j" THEN GO
 TO 590
 630 LET v=CODE g$-96: POKE 2330
0,v
 640 GO SUB 1000
 650 RANDOMIZE FN b()
 660 PRINT AT 7,26;g$;" ";v;" "
 670 BEEP .4,30
 680 GO TO 130
1000 PRINT AT 1,26;"          "
1010 PRINT AT 3,26;"          "
1020 PRINT AT 5,26;"          "
1030 RETURN
1100 INPUT "SAVE ";n$: IF n$=""
THEN GO TO 1100
1110 SAVE n$CODE 54600,699

scroll?
```

THE SPRITE EDITOR 2

The first two routines in the sprite editor enabled you to draw, load and save sprites. The remaining sprite routines have been written to give you the means of manipulating the sprite you have drawn. The effect they have on the sprite is shown in the displays here. Each routine is called by a keypress. Thus, when CAPS SHIFT and I are pressed together, a routine is called to invert every pixel of the sprite. To change the direction in which the sprite is facing, routine FNe at address 54436 is used, called by pressing CAPS SHIFT and R. Finally, to turn the current sprite upside down, routine FNf is used (called by pressing CAPS SHIFT and U).

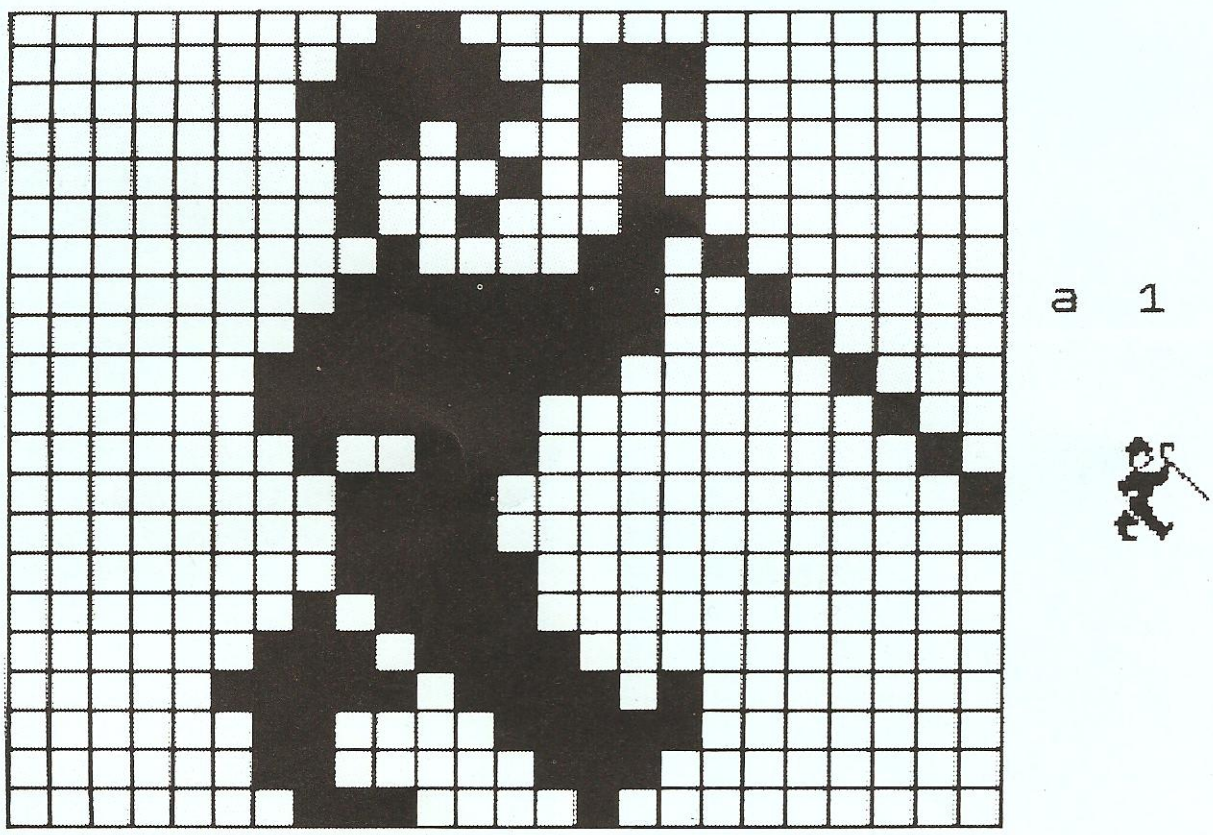
SPRITE EDITOR PROGRAM CONTD.

```
1120 GO TO 40
1130 INPUT "LOAD ";n$: LOAD n$ C
ODE : CLS
1140 GO TO 40
1150 STOP
1160 LET h=INT (s/32): LET l=s-h
*32
1170 RETURN
1200 FOR i=8 TO 200 STEP 8
1210 PLOT i,8: DRAW 0,167
1220 NEXT i
1230 FOR i=8 TO 175 STEP 8
1240 PLOT 8,i: DRAW 192,0
1250 NEXT i
1260 PLOT 8,175: DRAW 192,0
1270 RETURN



0 OK, 0:1
```

The BASIC editor program

The BASIC controlling program (shown on this page and on page 11) works as follows. Line 110 calls a subroutine at line 1200 to draw the grid on the screen (a simple series of lines). Line 120 calls a section of the program which prints the menu to the right of the grid. From this section blocks of the program at lines 1000, 1100 and 1130 are called as required. Lines 1000-1020 wipe the menu off the right-hand side of the screen. Lines 1100-1110 save the current sprite table to tape, and return control to the start of the program. A BEEP is heard whenever a sprite is saved in memory. Lines 1130-1140 load a new sprite table from tape into memory. If neither 1100 or 1130 is called, control returns to line 130.

Line 150 prints a sprite on the screen. Lines 180-210 accept a keypress and check if it is one of the functions I, R or U. If so, the relevant machine-code routine is called. If



FNa-e

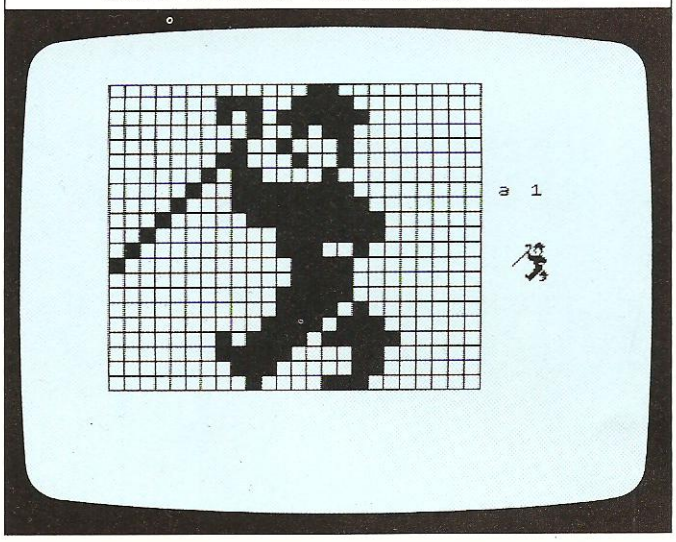
24 x 21 SPRITE EDITOR ROUTINES

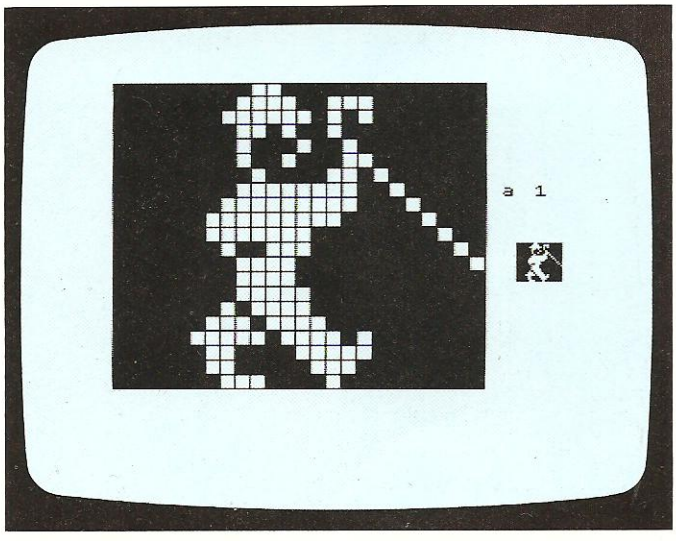
Start addresses 54200,54353,54422,54436,54482
Length 355 bytes
What they do The routines allow the user to design and edit up to ten sprites, and save them for use by later routines.

Using the routines The routines work in conjunction with a BASIC program shown on page 11. Editing the current sprite is controlled by the 5, 6, 7 and 8 keys; used with the shift key, pixels are inked in; used without, pixels are unset or, if set, changed to paper. The current sprite can be inverted (CAPS SHIFT and I), turned to face the other direction horizontally (CAPS SHIFT and R), or turned upside down (CAPS SHIFT and U).
Sprites are stored in a table from location 54600. Previously defined sprites may be loaded from tape, altered, and then saved as a new selection. When you save the sprite table, remember that all ten sprites are saved, not just the one you have most recently been editing.

the keypress is CAPS SHIFT and Z, control goes to the short sequence at 480 to display the menu again. Lines 250 to 280 do the work of editing the sprite. The sub-routine at line 1160 is called from this section. Notice how line 1160 divides the address to be saved into two parts, since 16 bit addresses will not fit into a single byte.

MANIPULATING THE CURRENT SPRITE





ROUTINE LISTING

```
7950 LET b=54200: LET l=350: LET
  z=0: RESTORE 7960
7951 FOR i=0 TO l-1: READ a
7952 POKE (b+i),a: LET z=z+a
7953 NEXT i
7954 LET z=INT (((z/l)-INT (z/l)
)*l)
7955 READ a: IF a<>z THEN PRINT
"??": STOP

7960 DATA 33,250,212,17,1
7961 DATA 88,14,3,213,237
7962 DATA 83,246,212,6,21
7963 DATA 237,91,246,212,197
7964 DATA 6,8,26,254,0
7965 DATA 40,4,254,130,32
7966 DATA 6,55,203,22,195
7967 DATA 224,211,167,203,22
7968 DATA 19,16,235,229,42
7969 DATA 246,212,1,32,0

7970 DATA 9,34,246,212,225
7971 DATA 193,35,16,212,209
7972 DATA 62,8,131,95,13
7973 DATA 32,197,33,250,212
7974 DATA 17,123,72,6,3
7975 DATA 213,197,14,3,6
7976 DATA 8,213,126,18,20
7977 DATA 35,16,250,209,62
7978 DATA 32,131,95,121,61
7979 DATA 254,1,32,5,6

7980 DATA 5,79,24,233,6
7981 DATA 8,79,254,0,32
7982 DATA 226,193,209,19,16
7983 DATA 215,201,205,64,212
7984 DATA 34,248,212,237,91
7985 DATA 248,212,33,250,212
7986 DATA 1,63,0,237,176
7987 DATA 201,58,4,91,71
7988 DATA 33,72,213,17,63
7989 DATA 0,167,237,82,25

7990 DATA 16,253,201,205,64
7991 DATA 212,34,248,212,17
7992 DATA 1,88,14,3,213
7993 DATA 237,83,246,212,6
7994 DATA 21,237,91,246,212
7995 DATA 197,6,8,126,167
7996 DATA 203,39,245,48,5
7997 DATA 62,0,195,120,212
7998 DATA 62,56,18,241,19
7999 DATA 16,239,229,42,246

8000 DATA 212,1,32,0,9
8001 DATA 34,246,212,225,193
8002 DATA 35,16,214,209,62
8003 DATA 8,131,95,13,32
8004 DATA 199,201,33,250,212
8005 DATA 6,63,62,255,174
8006 DATA 119,35,16,249,24
8007 DATA 39,33,250,212,6
8008 DATA 63,197,126,1,0
8009 DATA 8,167,203,31,203

8010 DATA 17,16,250,113,193
8011 DATA 35,16,239,6,21
8012 DATA 33,250,212,17,36
8013 DATA 213,78,26,119,121
8014 DATA 18,35,19,16,247
8015 DATA 33,250,212,205,87
8016 DATA 212,201,33,14,213
8017 DATA 17,250,212,6,3
8018 DATA 197,213,229,6,10
8019 DATA 78,26,119,121,18

8020 DATA 43,19,16,247,225
8021 DATA 209,1,21,0,84
8022 DATA 93,19,9,193,16
8023 DATA 230,24,213,0,0
8024 DATA 0,0,0,0,255
8025 DATA 255,255,0,0,190
8026 DATA 190,190,190,190,190
8027 DATA 190,190,190,190,128
8028 DATA 254,254,254,254,254
8029 DATA 254,254,254,0,0
8030 DATA 190,0,0,0,0
```

DISPLAYING SPRITES

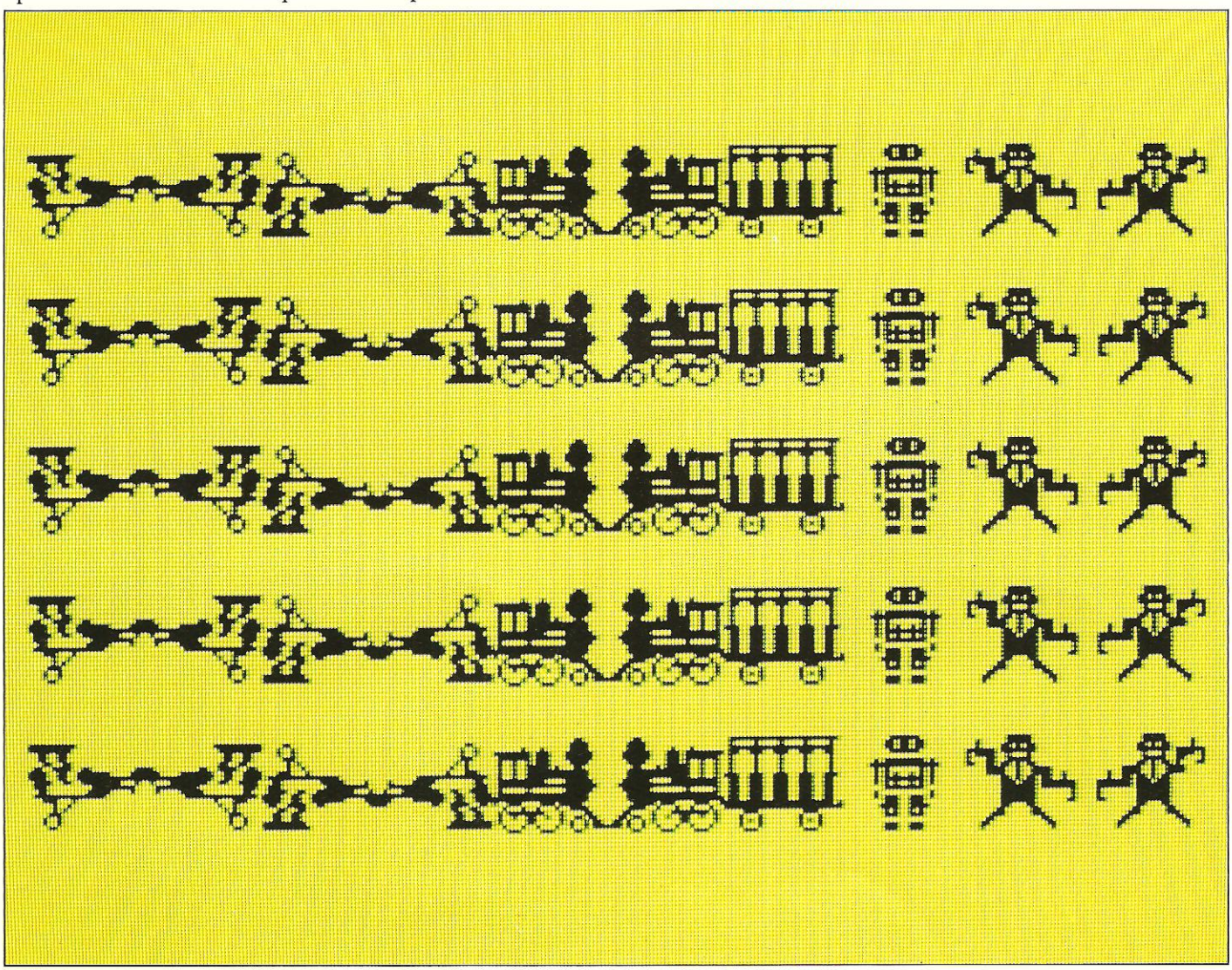
Once your sprites are defined it is very useful to be able to see what they look like on the screen. Obviously this can be done using the sprite-handling routine, but it is very hard to examine a sprite with a critical eye while it is moving across the screen! To avoid this problem, and also to give you the chance of looking at the shapes and designs of several sprites at once, the sprite print routine, FNf, has been provided.

This routine prints a sprite from the sprite buffer onto the screen at a specified position. Since any of the sprites can be printed at any position on the screen you can use the routine to preview the sprites you have designed, and it is at this stage that you can decide the best sort of starting and finishing positions for the sprites when they come to be used.

You can also use the routine to preview the effects of animation by calling the routine repeatedly to print sprites in an animation sequence on top of one another.

SPRITE DISPLAY PROGRAM

```
  10 DEF FN f(x,y,n)=USR 54100
 100 BORDER 4: PAPER 4: INK 0: C
LS
 110 FOR n=1 TO 10
 120 FOR k=1 TO 5
 130 RANDOMIZE FN f(n*3-3,(k*4)-
3,n)
 140 NEXT k
 150 NEXT n
 160 PAUSE 0
 500 CLS
 510 RANDOMIZE FN f(10,10,9)
 520 PAUSE 10
 530 RANDOMIZE FN f(10,10,10)
 540 PAUSE 10
 550 GO TO 510


0 OK, 0:1
```



This will give you an idea of how effective the animation is going to be, as well as allowing you to make any changes to the sprites before you work out the animation in detail.

How sprites are stored

Before examining the machine-code routine it is important to understand how sprites are stored in memory. As you know, a sprite table which stores ten sprites can be found at location 54600 onwards in memory. A 24 by 21 sprite has 504 elements (24x21), and since any of these elements can have an ink attribute either set or not set, this means that 504 bits of information need to be stored away for each sprite. Each sprite requires 63 bytes of memory (since there are eight bits in a byte) and, as all ten sprites are stored one after the other, they take up a total of 630 bytes.

Each sprite is arranged in memory as shown in the diagram below. Numbers correspond to the byte numbers (0-63), and the 1s and 0s represent the individual bits.

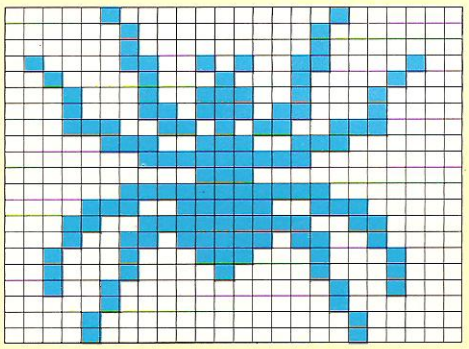
HOW SPRITES ARE STORED IN BINARY

Sprites are stored as binary numbers in 63 bytes (504 bits) of memory. The Is and Os correspond to set and unset pixels respectively on the sprite diagram shown underneath.

how memory bits are set

```
1   00000010000000000000100000000
25  00000010000000000001000000000
49  00000010000000000001000000000
73  01000011001010011000001000
97  00100001000100010000100000
121 00010001001110001000100000
145 00010001101110110001000000
160 00011100110101100111000000
193 00000111001110011100000000
217 00000000111111111000000000
241 00000000001110000000000000
265 00000001111111111110000000
289 00001110011111100111000000
313 00011001111111111001100000
337 00010011011111011001000000
361 00100010001110001000100000
385 00100010000010000100010000
409 00100100000000000010010000
433 00000010000000000010000000
457 00001000000000000001000000
481 00001000000000000001000000
```

sprite diagram



The BASIC program

The program on this page shows the contents of a sprite buffer, containing ten different sprites, displayed on the screen. The program calls the sprite print routine five times in a loop, to display the sprites down the screen as well as across it.

FNf

SPRITE PRINT ROUTINE

Start address 54100 **Length** 75 bytes
Other routines called Sprite editor routines (FNa-FNe).
What it does Prints a single sprite on the screen at a specifed point.

Using the routine This routine displays any single sprite from the sprite buffer. The routine does not move the sprite. Note that if the sprite is too far to the right of the screen it will reappear one character below on the left-hand side of the screen, since the Spectrum PRINT routine is used in the transfer of memory to screen.

ROUTINE PARAMETERS

DEF FNf(x,y,n)

x,y	specify print position (x<29, y<21)
n	specifies number of sprite (1-10)

ROUTINE LISTING

```
7900 LET b=54100: LET l=70: LET
z=0: RESTORE 7910
7901 FOR i=0 TO l-1: READ a
7902 POKE (b+i),a: LET z=z+a
7903 NEXT i
7904 LET z=INT (((z/l)-INT (z/l)
)*l)
7905 READ a: IF a<>z THEN PRINT
"??": STOP

7910 DATA 42,11,92,1,4
7911 DATA 0,9,86,1,8
7912 DATA 0,9,94,237,83
7913 DATA 148,211,9,126,50
7914 DATA 150,211,123,230,24
7915 DATA 246,64,103,123,230
7916 DATA 7,183,31,31,31
7917 DATA 31,130,111,34,146
7918 DATA 211,58,150,211,71
7919 DATA 17,63,0,33,9
7920 DATA 213,25,16,253,237
7921 DATA 91,146,211,195,1
7922 DATA 212,201,174,72,13
7923 DATA 14,3,0,0,0
7924 DATA 60,0,0,0,0
```

How the program works

Line 10 defines the sprite print routine, which has three parameters: the x,y screen co-ordinates at which the sprite is to be printed, and the number of the sprite (from the ten sprites stored in the sprite table). Lines 90 and 180 set up loops to print the sprites. Line 190 prints the sprite, using a combination of the two loop variables to define the x,y co-ordinate.

Lines 300 to 330 show how animation can be achieved using a print routine. A left and right mirror image of a single sprite are printed one after the other to produce a simple animation effect. You could produce more complex animation in this way; the only limitation to the method is the amount of time it takes to define sprites, although the animation in this program can also be speeded up substantially by leaving out the PAUSE statements. Of course, the other drawback is that you cannot move the sprite around the screen using this method.

MOVING SPRITES 1

Now that you have routines to create sprites and display them on the screen, you need a routine which makes the sprites change position. The routines introduced here enable you to get your sprites moving on the screen.

The master sprite routine

This routine enables you to move simple (that is, not animated) sprites. The routine has no title since it is always used together with the sprite-controlling routines in this book; by itself, the routine does nothing. When you use a sprite, it is this routine which causes the sprite to move across the screen in the required way. The main job of the routine is to print a sprite on the screen using Exclusive/OR printing, wipe it off, and print it again one pixel away until the program or routine asks the sprite to stop moving.

The routine has been programmed to work whether it has been called by a routine which is working within BASIC (a normal routine) or by one working independently of BASIC (an interrupt-driven routine).

The sprite-handling routine

This routine, FNg, allows you to control the movement of sprites. The routine works in conjunction with the master sprite routine, and both must be present in memory for sprites to move on the screen. The routine has a range of parameters to control exactly how the sprite will appear on the screen.

How to use the routines

Having produced some sprites, it is a simple matter to display them against a background, and the ideal way of creating backgrounds is to use a graphics editor program such as that in Book Three. All the background displays in this book were created using the graphics editor. The programs in this book do not themselves create backgrounds; you must add these yourself.

Interrupts

Most machine-code routines are called (as you will have seen) from BASIC, and are then executed in much the

MASTER SPRITE ROUTINE

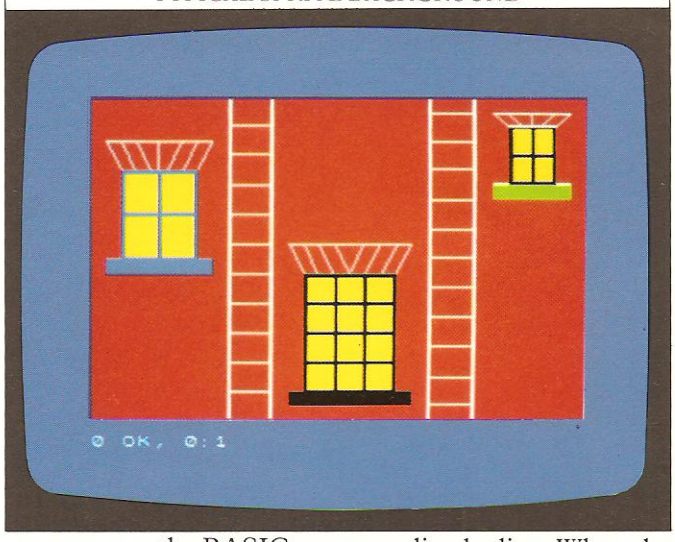
Start address 53700 **Length** 365 bytes
What it does Used in conjunction with the sprite-handling routine (FNg), this routine takes a sprite from the sprite table at location 54600 and moves it across the screen.

Using the routine This routine must always be used together with the other sprite routines given in this book; if used by itself, you will not see anything happen on the screen. Whenever sprites are used, the master sprite routine is called by the other routines to do the work of moving the sprite.

ROUTINE LISTING

```
7800 LET b=53700: LET l=360: LET
  z=0: RESTORE 7810
7801 FOR i=0 TO l-1: READ a
7802 POKE (b+i),a: LET z=z+a
7803 NEXT i
7804 LET z=INT (((z/l)-INT (z/l)
)*l)
7805 READ a: IF a<>z THEN PRINT
"??": STOP

7810 DATA 229,213,197,245,229
7811 DATA 221,225,120,205,177
7812 DATA 34,40,82,237,67
7813 DATA 15,210,50,229,209
7814 DATA 62,21,8,221,94
7815 DATA 0,221,86,21,221
7816 DATA 78,42,6,1,175
7817 DATA 203,27,203,26,203
7818 DATA 25,31,16,247,35
7819 DATA 35,35,174,119,43

7820 DATA 126,169,119,43,126
7821 DATA 170,119,43,126,171
7822 DATA 119,8,61,40,25
7823 DATA 8,221,35,36,124
7824 DATA 230,7,32,205,1
7825 DATA 49,0,8,95,8
7826 DATA 62,21,147,128,71
7827 DATA 205,177,34,24,189
7828 DATA 241,193,209,225,201
7829 DATA 237,67,81,210,62

7830 DATA 21,8,221,94,0
7831 DATA 221,86,21,221,78
7832 DATA 42,35,35,126,169
7833 DATA 119,43,126,170,119
7834 DATA 43,126,171,119,8
7835 DATA 61,40,218,8,221
7836 DATA 35,36,124,230,7
7837 DATA 32,221,8,95,8
7838 DATA 1,208,46,62,21
7839 DATA 147,128,71,205,177

7840 DATA 34,24,205,229,213
7841 DATA 197,229,221,225,151
7842 DATA 50,41,211,120,205
7843 DATA 177,34,202,216,210
7844 DATA 237,67,194,210,50
7845 DATA 130,210,62,21,8
7846 DATA 221,94,0,221,86
7847 DATA 21,221,78,42,6
7848 DATA 1,175,203,27,203
7849 DATA 26,203,25,31,16

7850 DATA 247,35,35,35,71
7851 DATA 126,160,196,33,211
7852 DATA 126,168,119,43,126
7853 DATA 161,196,33,211,126
7854 DATA 169,119,43,126,162
7855 DATA 196,33,211,126,170
7856 DATA 119,43,126,163,196
7857 DATA 33,211,126,171,119
7858 DATA 8,61,40,25,8
7859 DATA 221,35,36,124,230

7860 DATA 7,32,183,1,49
7861 DATA 0,8,95,8,62
7862 DATA 21,147,128,71,205
7863 DATA 177,34,24,167,193
7864 DATA 209,225,58,41,211
7865 DATA 201,237,67,21,211
7866 DATA 62,21,8,221,94
7867 DATA 0,221,86,21,221
7868 DATA 78,42,35,35,126
7869 DATA 161,196,33,211,126

7870 DATA 169,119,43,126,162
7871 DATA 196,33,211,126,170
7872 DATA 119,43,126,163,196
7873 DATA 33,211,126,171,119
7874 DATA 8,61,40,201,8
7875 DATA 221,35,36,124,230
7876 DATA 7,32,206,8,95
7877 DATA 8,1,208,46,62
7878 DATA 21,147,128,71,205
7879 DATA 177,34,24,190,245
7880 DATA 62,1,50,41,211
7881 DATA 241,201,1,0,0
7882 DATA 83,0,0,0,0
```

TYPICAL SPRITE BACKGROUND



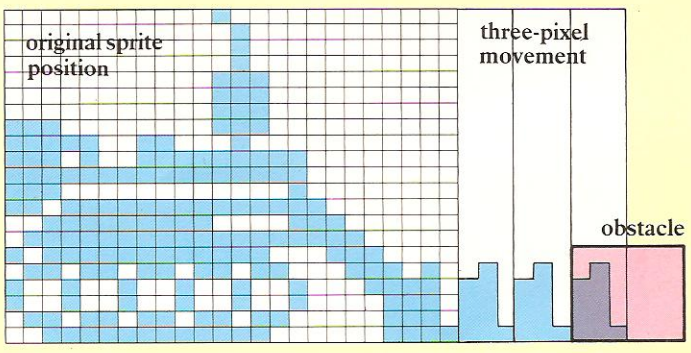
same way as the BASIC program, line by line. When the machine-code routine has finished running control is returned to the BASIC program. However, there is a much more sophisticated method of using machine code, which is to make a routine run independently of BASIC.

This can be achieved by taking advantage of the fact that at regular intervals the Spectrum interrupts the running of any BASIC program or machine-code routine which is being executed. It does this so that it can print information on the screen, and perform various other household chores, like memory management. These interrupts occur at least fifty times a second, so quickly in fact that if you carry out a machine-code routine as well as a BASIC program using interrupts it will appear that both things are happening simultaneously.

Two of these "interrupt-driven" routines have been provided in this book. One is the keyboard-controlled sprite, and the other is the interrupt-driven window given on pages 32-33.

HOW A SPRITE MOVES

Sprites move in blocks of three pixels. On reaching an obstacle, and with the collision flag set to 1, a sprite continues to move for three pixels before stopping.



FNg

SPRITE-HANDLING ROUTINE

Start address 53500 **Length** 170 bytes
Other routines called Master sprite routine.
What it does Prints and moves a single sprite on the screen.

Using the routine The screen area is measured in pixel co-ordinates from the top left-hand corner, rather than from the bottom of the screen.
Sprites move in multiples of three pixels, so a value of 60 for l moves the sprite 180 pixels. If the collision detection flag is set to 0, the sprite will pass over obstacles (any pixels with INK set). If c is set to 1, the sprite will stop when it hits an object (after an overlap of three pixels). You can find out the precise position where a sprite stopped by PEEKing locations 53498 and 53499 for the y and x co-ordinates respectively.

ROUTINE PARAMETERS

DEF FNg(x,y,d,l,s,c,n)

x,y	specify top left-hand corner of sprite (x<232, y<155)
d	direction of travel of the sprite (0=left, 1=right, 2=up, 3=down)
l	distance moved (vertical<=51, horizontal<=77)
s	switch (s=1 to disappear, s=0 to remain on screen)
c	collision detection flag (1=stop, 0=continue)
n	specifies number of sprite (1-10)

ROUTINE LISTING

```
7750 LET b=53500: LET l=165: LET
 z=0: RESTORE 7760
7751 FOR i=0 TO l-1: READ a
7752 POKE (b+i),a: LET z=z+a
7753 NEXT i
7754 LET z=INT (((z/l)-INT (z/l)
)*l)
7755 READ a: IF a<>z THEN PRINT
"??": STOP

7760 DATA 42,11,92,17,4
7761 DATA 0,25,78,30,8
7762 DATA 25,70,25,126,230
7763 DATA 3,50,158,209,25
7764 DATA 126,50,159,209,25
7765 DATA 126,230,1,50,160
7766 DATA 209,25,126,230,1
7767 DATA 50,161,209,25,126
7768 DATA 17,63,0,33,9
7769 DATA 213,25,61,32,252

7770 DATA 205,93,210,254,0
7771 DATA 40,19,58,161,209
7772 DATA 254,0,40,12,58
7773 DATA 160,209,198,1,230
7774 DATA 1,50,160,209,24
7775 DATA 71,118,205,196,209
7776 DATA 58,158,209,254,0
7777 DATA 40,19,254,1,40
7778 DATA 26,254,2,40,33
7779 DATA 5,5,5,120,230

7780 DATA 252,40,44,195,132
7781 DATA 209,13,13,13,121
7782 DATA 230,252,40,33,195
7783 DATA 132,209,12,12,12
7784 DATA 121,214,231,48,22
7785 DATA 195,132,209,4,4
7786 DATA 4,120,214,150,48
7787 DATA 11,58,159,209,61
7788 DATA 50,159,209,254,0
7789 DATA 32,159,58,160,209
7790 DATA 237,67,250,208,254
7791 DATA 0,200,118,205,93
7792 DATA 210,201,1,47,0
7793 DATA 66,0,0,0,0
```

MOVING SPRITES 2

The sprite-handling routine has many user-controlled features built into it, as you can see from the long list of parameters which are passed to it. It is a good idea to become familiar with these, as otherwise you will under-utilize the potential of this very powerful routine. The train program, given here, is a good example of how the parameters are used.

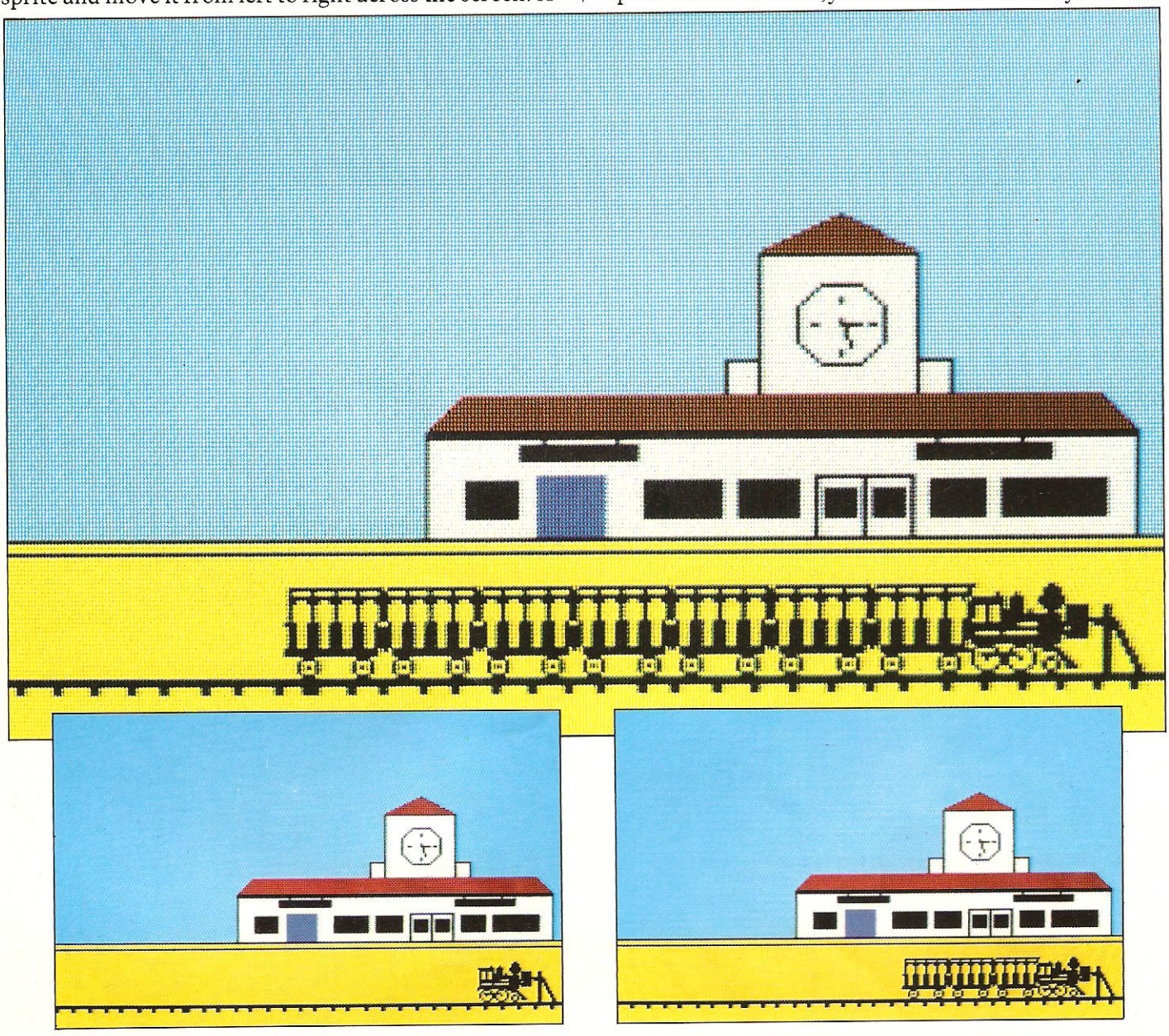
The train program

The large display on this page shows a train and some carriages being shunted along a railway line. All the movement in this program is controlled by the sprite-handling routine. Lines 110 and 120 take the train sprite and move it from left to right across the screen. A mirror image of the train is then selected and driven back to the start position, where it remains on the screen. Lines 130-150 set up a loop which calls carriages one by one. Since this contains the collision flag set to 1 each carriage moves across the screen until it encounters an obstacle—the previous carriage — when it stops and remain on the screen.

More about the parameters

It is worth looking at the impressive range of parameters available with the sprite-handling routine in a little more detail.

The first feature which you will use, of course, is the specification of the x,y co-ordinate at which you want



TRAIN PROGRAM

```
 10 DEF FN g(x,y,d,l,s,c,n)=USR
53500
 100 BORDER 4
 110 RANDOMIZE FN g(205,139,0,68
,0,0,6)
 120 RANDOMIZE FN g(1,139,1,70,1
,0,5)
 130 FOR x=1 TO 7
 140 RANDOMIZE FN g(1,139,1,70,1
,1,7)
 150 NEXT x
 160 PAUSE 0
 170 GO TO 110


0 OK, 0:1
```

the sprite to start. Normally you can specify this simply by looking at the screen, but more sophisticated methods are available for determining the start point.

In the program example a train appears from the left and crosses to the right-hand side. It travels a distance of 65 (that is, 195 pixels) and so you know that you can start the second train off from a point 195 pixels to the right of where the first started. But what if you didn't know where the first train had started or stopped? Since the routine stores the last pixel position of the sprite at two pointers in memory, the new sprite could then start from position (PEEK(53499),(PEEK(53498)).

There is a case for making the distance moved by the sprite (parameter l) as short as possible with each call to the routine, despite this producing some flicker because of the volume of transfers from BASIC to machine code and back. This is because while you are in BASIC you can keep a check on the current screen, the position of the sprite and so on, but while the sprite is moving you do not have any control over it. This, of course, is an advantage of interrupt-driven routines which allow you to monitor the progress of other things on the screen while a sprite or window is moving.

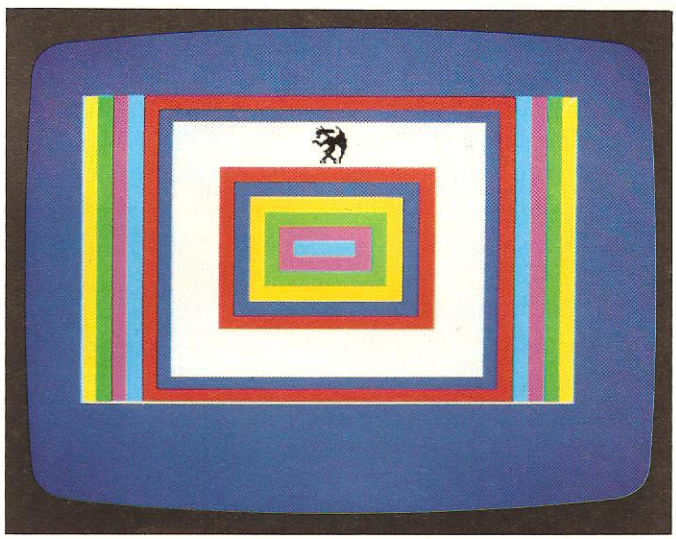
The value you give the switch, s, depends on what you want the sprite to do after you have finished using it. In the program the switch is off for the first train and on for the second: after the first train has come on and travelled across the screen it should disappear before the second returns. On the other hand the second train must remain on the screen after its journey as, if it doesn't, the carriage will have nothing to run into.

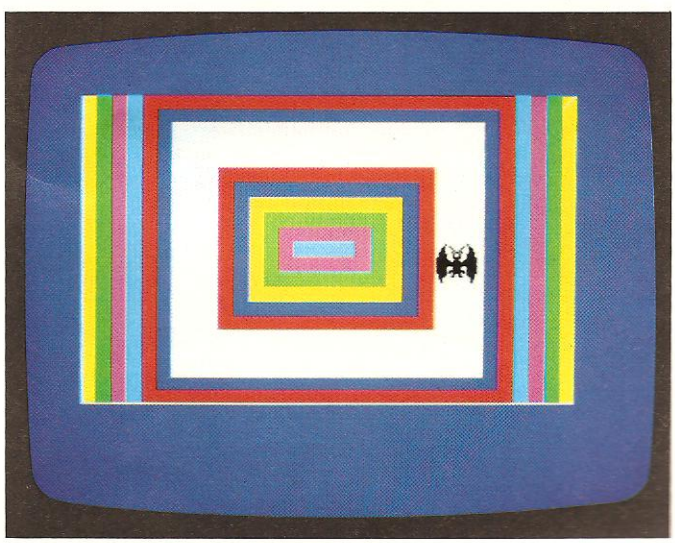
The bat program

The final program on this page shows how the sprite-handling routine can be called a number of times in a loop to move an object in four different directions around the screen. Although you see the bat moving continuously on the screen, the program listing reveals that four different routine calls are being used, one for each direction that the sprite is being moved, and three different sprite shapes are used to give various views of the bat's body as it moves around the loop.

BAT PROGRAM

```
 10 DEF FN g(x,y,d,l,s,c,n)=USR
53500
 100 BORDER 1
 110 RANDOMIZE FN g(48,140,1,40,
0,0,1)
 120 RANDOMIZE FN g(183,140,3,34
,0,0,3)
 130 RANDOMIZE FN g(177,25,0,40,
0,0,2)
 140 RANDOMIZE FN g(48,25,2,40,0
,0,3)
 150 GO TO 20


0 OK, 0:1
```





KEYBOARD-CONTROLLED SPRITES

The keyboard-controlled sprite routine, FNh, enables you to control the movement of sprites using the cursor keys. The only difficulty with the routine lies not so much in using it as in switching it off. Because the routine is interrupt-driven it continues to operate after any BASIC program has finished. Even as you edit a program you will find that the sprite is still moving on the screen. A subroutine is required to switch it off:

```
2000 DEF FN z(s)=USR 53100
2010 RANDOMIZE FN z(0)
2020 RETURN
```

This redefines the routine with just one parameter, the switch. If the switch is given a value of 0, the routine will be switched off. The maze program on page 20 gives you a chance to try using the routine; you can create your own maze with the Book Three graphics editor.

Controlling the routine

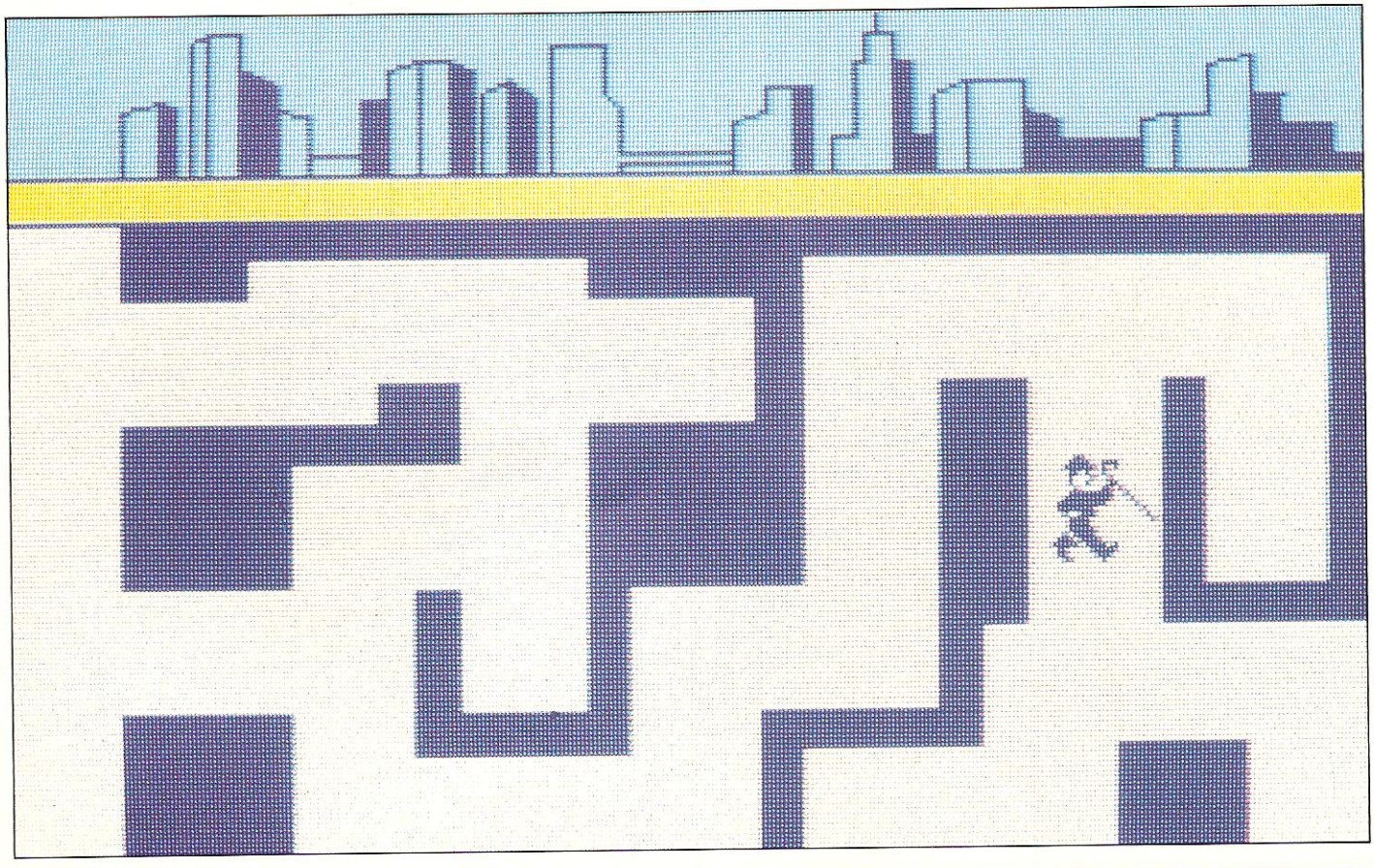
Although keyboard-controlled, the routine is quite hard to control from within a program. Since the sprite routine runs outside BASIC, it is only when the routine stops that you can check its position. One solution is to leave the collision detection off but to set up your own collision detection instead. You can do this by switching the sprite off, and looking at the last x,y co-ordinate (stored at 53099, 53098). Then use the POINT command to find if pixels at these co-ordinates are set:

x,y	x+24,y
x,y+21	x+24,y+21

If this is so, a collision has occurred.

MAZE PROGRAM

```
  10 DEF FN h(s,x,y,c,n)=USR 531
00
  20 DEF FN z(s)=USR  53100
 100 BORDER 2
 110 RANDOMIZE FN h(1,1,50,0,3)
 120 GO TO 120
 130 STOP
1000 RANDOMIZE FN z(0)


0 OK, 0:1
```



FNh

KEYBOARD-CONTROLLED SPRITE ROUTINE

Start address 53100 **Length** 250 bytes
Other routines called Master sprite routine.
What it does Puts a sprite on the screen and allows it to be controlled by the cursor keys.

Using the routine The routine is interrupt-driven, so it will continue to respond to keyboard presses until switched off by calling the routine with the switch parameter (s) set to 0.
If not switched off, the routine continues to operate even when you stop the program and attempt to, say, edit — you will see a sprite moving whenever you use the cursor keys. It is advisable to switch the routine off at the end of any BASIC program which calls the keyboard-controlled sprite routine. This can be done by defining the function a second time using a function title not used elsewhere in the program, say FNz. This function is defined as having a single parameter only, s, which means that the function can then be called with this parameter only, set to zero.
The collision detection parameter, c, can be used to detect if the sprite passes over any pixels with set INK attributes.

ROUTINE PARAMETERS

DEF FN h(s,x,y,c,n)	
s	switch (1=on, 0=off)
x,y	start position of sprite (0<=x<=231, 0<=y<154)
c	collision detection (1=on, 0=off)
n	number of sprite (1-10)

The obstacle program

The final program again has the aim of avoiding obstacles. Lines 10 and 20 define the routine twice as before. Line 110 switches the keyboard sprite on. Lines 120-140 print a series of random graphics blocks on the screen. The aim of the game is to avoid these blocks. Line 500 is a continuous loop which keeps the program RUNning.

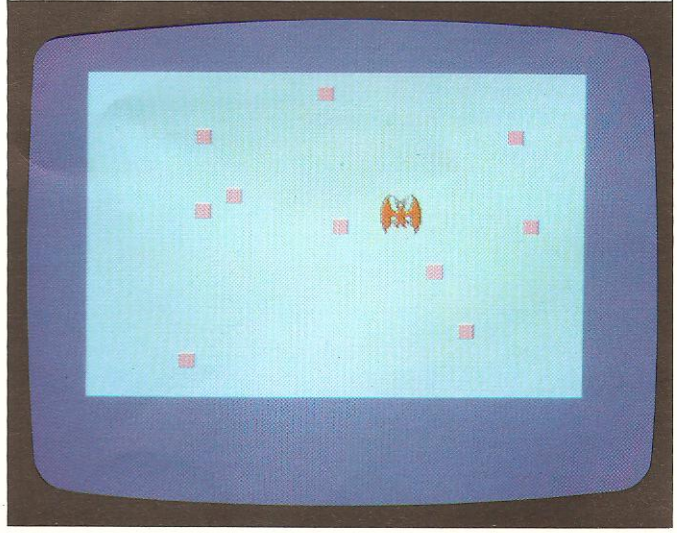
ROUTINE LISTING

```
7650 LET b=53100: LET l=245: LET
 z=0: RESTORE 7660
7651 FOR i=0 TO l-1: READ a
7652 POKE (b+i),a: LET z=z+a
7653 NEXT i
7654 LET z=INT (((z/l)-INT (z/l)
)*l)
7655 READ a: IF a<>z THEN PRINT
"??": STOP

7660 DATA 42,11,92,17,4
7661 DATA 0,25,126,230,1
7662 DATA 50,95,208,32,11
7663 DATA 42,96,208,237,75
7664 DATA 106,207,205,196,209
7665 DATA 201,30,8,25,78
7666 DATA 25,70,25,126,230
7667 DATA 1,50,93,208,25
7668 DATA 126,33,9,213,17
7669 DATA 63,0,25,61,32

7670 DATA 252,34,96,208,205
7671 DATA 93,210,237,67,106
7672 DATA 207,243,175,71,50
7673 DATA 94,208,62,207,33
7674 DATA 2,206,119,35,16
7675 DATA 252,62,206,237,71
7676 DATA 237,94,251,201,0
7677 DATA 0,0,0,0,0
7678 DATA 0,0,0,0,0
7679 DATA 0,0,0,0,255

7680 DATA 243,245,197,213,229
7681 DATA 8,245,8,221,229
7682 DATA 58,95,208,254,0
7683 DATA 40,24,205,7,208
7684 DATA 58,94,208,254,0
7685 DATA 40,14,58,93,208
7686 DATA 254,0,40,7,205
7687 DATA 123,207,175,50,95
7688 DATA 208,195,252,207,221
7689 DATA 225,8,241,8,225

7690 DATA 209,193,241,251,201
7691 DATA 237,75,106,207,42
7692 DATA 96,208,62,239,219
7693 DATA 254,203,103,40,44
7694 DATA 203,95,40,31,203
7695 DATA 87,40,18,62,247
7696 DATA 219,254,203,103,40
7697 DATA 1,201,13,13,13
7698 DATA 121,230,252,200,24
7699 DATA 25,12,12,12,62

7700 DATA 231,145,216,24,16
7701 DATA 5,5,5,120,230
7702 DATA 252,200,24,7,4
7703 DATA 4,4,62,153,144
7704 DATA 216,197,237,75,106
7705 DATA 207,205,196,209,193
7706 DATA 205,93,210,50,94
7707 DATA 208,237,67,106,207
7708 DATA 201,1,0,0,9
7709 DATA 26,0,0,0,0
```

OBSTACLE PROGRAM

```
  10 DEF FN h(s,x,y,c,n)=USR 531
00
  20 DEF FN z(s)=USR 53100
 100 BORDER 1: PAPER 5: INK 2: C
LS
 110 RESTORE FN h(1,10,10,1,3)
 120 FOR x=1 TO 10
 130 PRINT INK 3;AT (INT (RND*20
)),(INT (RND*32));"■"
 140 NEXT x
 150 GO TO 150
 499 STOP
 500 RANDOMIZE FN z(0)



0 OK, 0:1
```

OBSTACLE DISPLAY



DOUBLE-SIZED SPRITES

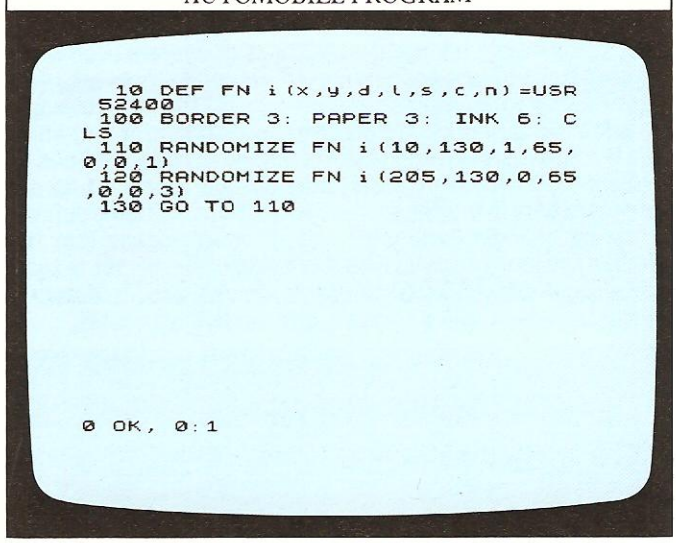
You have already seen what can be done with a sprite 24 by 21 pixels (504 pixels in all), but there are occasions when you would like to use larger sprites still. This makes great demands upon your Spectrum, but the two routines that are provided here (FNi and FNj) each give you the power to move over 1000 pixels at once. These routines provide you with double horizontal and vertical sprites respectively. In each case sprites from the sprite table are attached to one another and are then moved together in exactly the same way as a single sprite — though naturally not quite as quickly.

The double sprite programs

Both the demonstration programs are straightforward. In the first program, a double horizontal sprite has been used, and this demonstrates the effectiveness of quite large moving objects — it would take 18 user-defined graphics to define the area of the car, let alone move it! The program enables you to see the great improvement in sprite visibility that can be obtained by doubling its size.

The two sprites which make up the car are stored as the first and second of the sprite table — the routine simply takes the sprite specified by the n parameter, together with the following sprite from the sprite table that is stored in memory.

AUTOMOBILE PROGRAM



FNi

DOUBLE HORIZONTAL SPRITE ROUTINE

Start address 52400 **Length** 235 bytes
Other routines called Sprite editor routines (FNa-FNe).
What it does Displays and moves two sprites together horizontally on the screen.

Using the routine The routine takes sprites n (left) and n+1 (right) from the sprite buffer. Parameters are as for the sprite-handling routine (FNg). Bytes 52398 and 52399 specify the y and x co-ordinates of the sprite's final position after calling the routine.

ROUTINE PARAMETERS

DEF FNi(x,y,d,l,s,c,n)

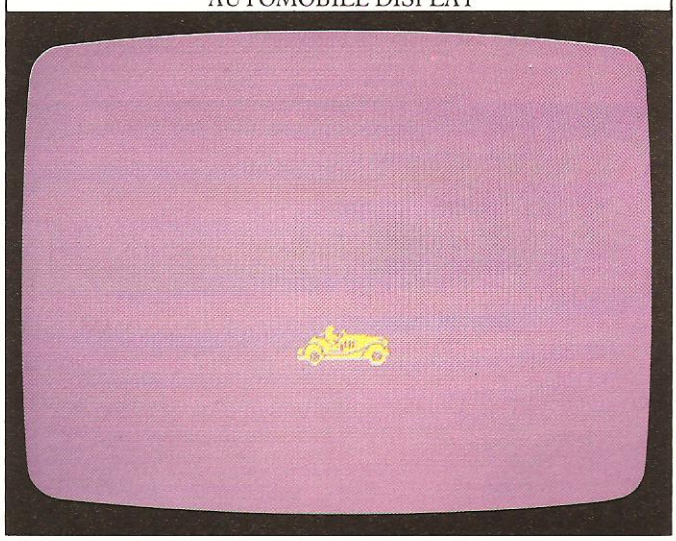
x,y	start co-ordinates (0<=x<=231,0<=y<=154)
d	specifies direction of travel (0=left, 1=right, 2=up, 3=down)
l	distance moved (vert<=51, horiz<=77)
s	switch (1=on, 0=off)
c	flag for collision detection (1=on, 0=off)
n	number of first sprite (1-10)

ROUTINE LISTING

```
7550 LET b=52400: LET l=230: LET z=0: RESTORE 7560
7551 FOR i=0 TO l-1: READ a
7552 POKE (b+i),a: LET z=z+a
7553 NEXT i
7554 LET z=INT (((z/l)-INT (z/l))*l)
7555 READ a: IF a<>z THEN PRINT "??": STOP
```

```
7560 DATA 42,11,92,17,4
7561 DATA 0,25,78,30,8
7562 DATA 25,70,25,126,230
7563 DATA 3,50,148,205,25
7564 DATA 126,50,149,205,25
7565 DATA 126,230,1,50,150
7566 DATA 205,25,126,230,1
7567 DATA 50,146,205,25,126
7568 DATA 17,63,0,33,9
7569 DATA 213,25,61,32,252

7570 DATA 175,50,147,205,205
7571 DATA 93,210,205,137,205
7572 DATA 197,229,17,63,0
7573 DATA 25,62,24,129,79
7574 DATA 205,93,210,205,137
7575 DATA 205,225,193,197,229
7576 DATA 6,1,118,16,253
7577 DATA 225,193,0,0,118
7578 DATA 205,196,209,197,229
7579 DATA 17,63,0,25,62

7580 DATA 24,129,79,205,196
7581 DATA 209,225,193,58,148
7582 DATA 205,254,0,40,19
7583 DATA 254,1,40,26,254
7584 DATA 2,40,33,5,5
7585 DATA 5,120,230,252,40
7586 DATA 59,195,84,205,13
7587 DATA 13,13,121,230,252
7588 DATA 40,48,195,84,205
7589 DATA 12,12,12,121,214

7590 DATA 207,48,37,195,84
7591 DATA 205,4,4,4,120
7592 DATA 214,150,48,26,58
7593 DATA 147,205,254,0,40
7594 DATA 7,58,146,205,254
7595 DATA 0,32,12,58,149
7596 DATA 205,61,50,149,205
7597 DATA 254,0,194,230,204
7598 DATA 58,150,205,237,67
7599 DATA 174,204,254,0,200

7600 DATA 118,205,93,210,17
7601 DATA 63,0,25,62,24
7602 DATA 129,79,118,205,93
7603 DATA 210,201,254,0,200
7604 DATA 62,1,50,147,205
7605 DATA 201,1,0,1,0
7606 DATA 53,0,0,0,0
```

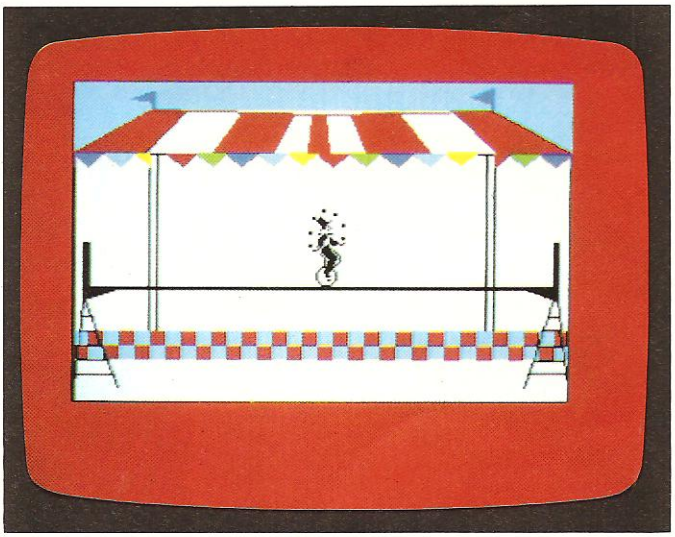
AUTOMOBILE DISPLAY



The other program shows a double vertical sprite — a unicyclist — against a circus background, drawn using the graphics editor from Book Three.

UNICYCLE PROGRAM

```
 10 DEF FN j(x,y,d,l,s,c,n)=USR
52100
 100 BORDER 2
 110 RANDOMIZE FN j(15,70,1,69,0
,0,1)
 120 RANDOMIZE FN j(220,70,0,70,
0,0,3)
 130 GO TO 110




0 OK, 0:1
```



FNj

DOUBLE VERTICAL SPRITE ROUTINE

Start address 52100 **Length** 230 bytes
What it does Displays a double vertical sprite.

Using the routine Used in the same way as the horizontal sprite routine, but final sprite position now specified by 52098 and 52099.

ROUTINE PARAMETERS

DEF FNj(x,y,d,l,s,c,n)

x,y	start co-ordinates (0<=x<=231,0<=y<=154)
d	specifies direction of travel (0=left, 1=right, 2=up, 3=down)
l	distance moved (vertical<=51, horizontal<=77)
s	switch (1=on, 0=off)
c	flag for collision detection (1=on, 0=off)
n	number of first sprite(1-10)

ROUTINE LISTING

```
7450 LET b=52100: LET l=225: LET
 z=0: RESTORE 7460
7451 FOR i=0 TO l-1: READ a
7452 POKE (b+i),a: LET z=z+a
7453 NEXT i
7454 LET z=INT (((z/l)-INT (z/l)
)*l)
7455 READ a: IF a<>z THEN PRINT
"??": STOP

7460 DATA 0,42,11,92,17
7461 DATA 4,0,25,78,30
7462 DATA 8,25,70,25,126
7463 DATA 230,3,50,103,204
7464 DATA 25,126,50,104,204
7465 DATA 25,126,230,1,50
7466 DATA 105,204,25,126,230
7467 DATA 1,50,101,204,25
7468 DATA 126,17,63,0,33
7469 DATA 9,213,25,61,32

7470 DATA 252,175,50,102,204
7471 DATA 205,93,210,205,86
7472 DATA 204,197,229,17,63
7473 DATA 0,25,62,21,128
7474 DATA 71,205,93,210,205
7475 DATA 86,204,225,193,197
7476 DATA 229,6,1,118,16
7477 DATA 253,225,193,0,0
7478 DATA 118,205,196,209,197
7479 DATA 229,17,63,0,25

7480 DATA 62,21,128,71,205
7481 DATA 196,209,225,193,58
7482 DATA 103,204,254,0,40
7483 DATA 19,254,1,40,26
7484 DATA 254,2,40,33,5
7485 DATA 5,5,120,230,252
7486 DATA 40,51,195,41,204
7487 DATA 13,13,13,121,230
7488 DATA 252,40,40,195,41
7489 DATA 204,12,12,12,121

7490 DATA 214,231,48,29,195
7491 DATA 41,204,4,4,4
7492 DATA 120,214,132,48,18
7493 DATA 58,102,204,254,0
7494 DATA 32,11,58,104,204
7495 DATA 61,50,104,204,254
7496 DATA 0,32,128,58,105
7497 DATA 204,237,67,202,91
7498 DATA 254,0,200,118,205
7499 DATA 93,210,17,63,0

7500 DATA 25,62,21,128,71
7501 DATA 118,205,93,210,201
7502 DATA 254,0,200,58,101
7503 DATA 204,254,0,200,62
7504 DATA 1,50,102,204,201
7505 DATA 20,0,0,0,0
```

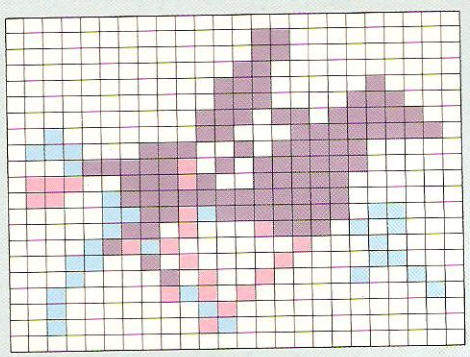
ANIMATION 1

So far the movement you have seen has been restricted to moving fixed sprites on the screen. This is all very well if your sprite is an aeroplane flying across the sky, or a car driving along, because these do not change shape as they move. However, if you want to have a moving person or a flying bird then something more sophisticated is required: the sprites need to be animated while they are moving on the screen.

Animation, like most techniques in computer graphics, relies on tricking the eye. It is achieved by displaying several stationary images in succession to create the illusion of movement. Animation can be achieved without any relative movement, as in an animated figure of a man jumping on the spot, for example. You will immediately notice when you start to animate your designs how only a small change in the sprite shape is required to give an effective result. By looking at the display in the figure above, for example, you will see how only the legs and tail of a horse need to be moved to give an animated effect; the horse's body remains unchanged. If the body was moved when the horse was animated, the effect would look very jerky.

ACHIEVING SMOOTH ANIMATION

The body of the horse sprite stays in the same position in successive frames (shown in red and blue); only the legs and tails of the horse are moved.

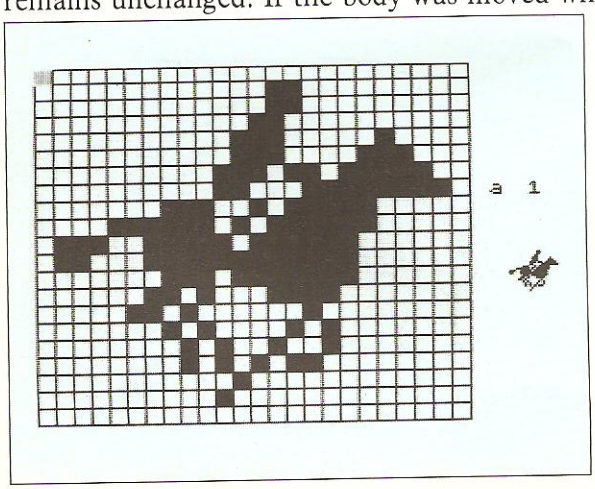


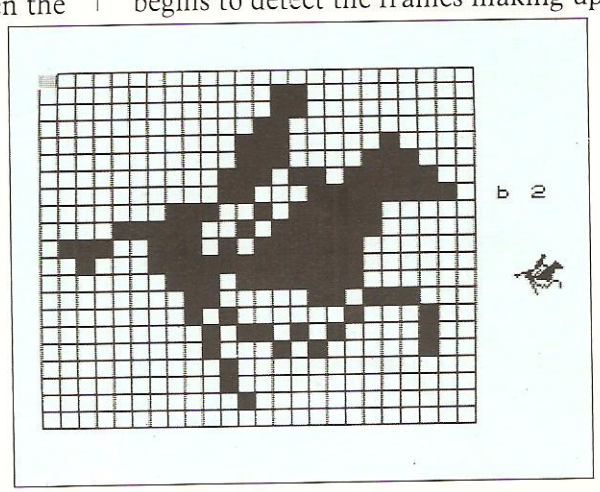
ANIMATION PROGRAM

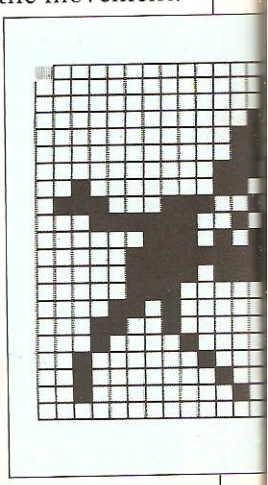
```
  10 DEF FN k(x,y,d,t,s,f,c,v,n)
=USR 51700
 100 BORDER 3: PAPER 5: INK 0: C
LS
 110 RANDOMIZE FN k(10,10,1,65,0
,5,0,50,1)
 120 RANDOMIZE FN k(205,10,0,65,
0,5,0,50,6)
 130 GO TO 100


0 OK, 0:1
```

In many cases you only need to take two frames and switch between them to produce convincing animation, but the animation routine, FNk, allows you to have as many frames as you like, within the limits of the sprite table — that is, up to a maximum of ten. Using ten frames, each of which differs slightly, you can create very effective illusions: unless you think about what is happening you would never realize that the sprites are being printed as stationary images.

The illusion produced by animation is also used in the cinema, where stationary images are shown one after the other at 14 frames per second. When watching a film you don't think of the picture as stationary, because the images are changing too fast for your eye to register. Since the animation routine allows you to vary the speed at which images are replaced on the screen, you can experiment by giving different values to the v parameter to see how slow the animation can be before your eye begins to detect the frames making up the movement.







FNk

SPRITE ANIMATION ROUTINE

Start address 51700 **Length** 275 bytes
Other routines called Master sprite routine.
What it does Uses a sequence of sprites from the sprite butter to give the effect of animation.

Using the routine Most of the routine parameters are as before, with some additional ones added to give you increased control of the animated sprites. Thus, the frame parameter specifies how many frames are used in the animation, and the velocity parameter determines how quickly the routine should move from one frame to the next. Do not give this parameter a value of 0.

ROUTINE PARAMETERS

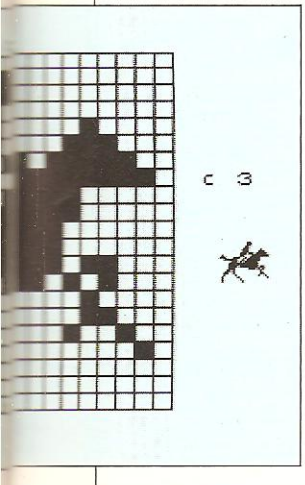
DEF FNk(x,y,d,l,s,f,c,v,n)	
x,y	sprite start co-ordinate (0<x<256, 0<y<176)
d	specifies direction of travel (0=left, 1=right, 2=up, 3=down)
l	distance moved (vertical<=51, horizontal<=77)
s	switch (1=switch on, 0=switch off)
f	number of frames used in the animation (1-10)
c	flag for collision detection (1=on, 0=off)
v	velocity of animation (1<=v<=255, the slowest)
n	number of first sprite (1-10)

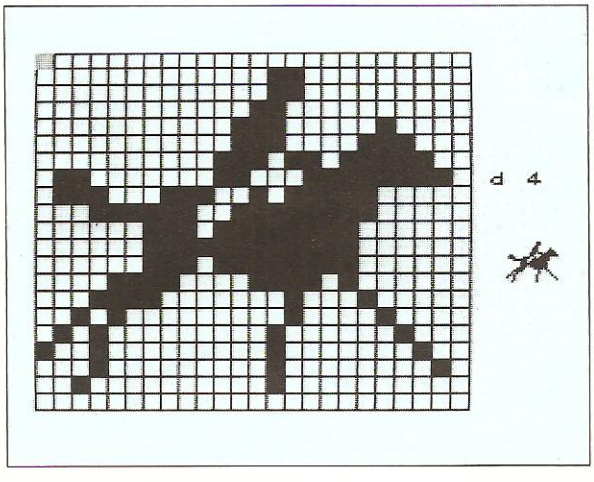
The animation program

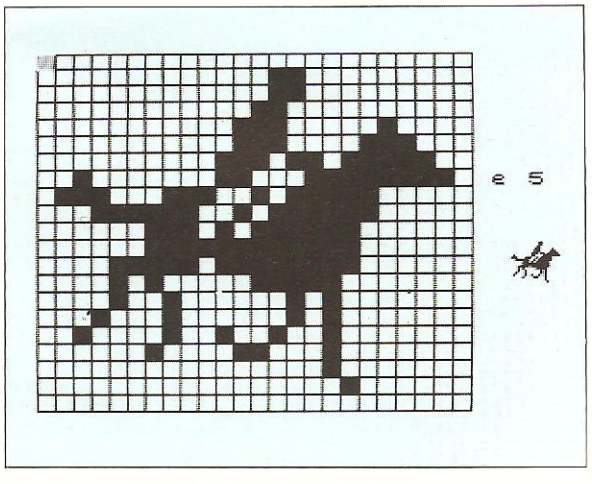
This program has not been displayed for the obvious reason that it is difficult to capture the effect of movement in a still photograph. The program takes five sprites and displays them in turn to give a very smooth effect of movement. The five sprites are shown along the bottom of the page as they appear on the sprite editor. Notice how similar each horse is to the next; only minor changes are needed for the animation. For an even better result, increase the number of frames.

ROUTINE LISTING

```
7350 LET b=51700: LET l=270: LET
 z=0: RESTORE 7360
7351 FOR i=0 TO l-1: READ a
7352 POKE (b+i),a: LET z=z+a
7353 NEXT i
7354 LET z=INT (((z/l)-INT (z/l)
)*l)
7355 READ a: IF a<>z THEN PRINT
"??": STOP

7360 DATA 42,11,92,17,4
7361 DATA 0,25,78,30,8
7362 DATA 25,70,25,126,230
7363 DATA 3,50,209,202,25
7364 DATA 126,50,211,202,25
7365 DATA 126,230,1,50,212
7366 DATA 202,25,126,50,216
7367 DATA 202,50,215,202,25
7368 DATA 126,230,1,50,210
7369 DATA 202,25,126,50,217

7370 DATA 202,25,126,17,63
7371 DATA 0,33,202,212,25
7372 DATA 61,32,252,34,213
7373 DATA 202,58,215,202,61
7374 DATA 50,215,202,254,0
7375 DATA 32,9,58,216,202
7376 DATA 50,215,202,42,213
7377 DATA 202,17,63,0,25
7378 DATA 205,93,210,254,0
7379 DATA 40,19,58,210,202

7380 DATA 254,0,40,12,58
7381 DATA 212,202,198,1,230
7382 DATA 1,50,212,202,24
7383 DATA 90,229,213,197,58
7384 DATA 217,202,71,17,0
7385 DATA 0,33,0,0,237
7386 DATA 176,193,209,225,118
7387 DATA 205,196,209,58,209
7388 DATA 202,254,0,40,19
7389 DATA 254,1,40,26,254

7390 DATA 2,40,33,5,5
7391 DATA 5,120,230,252,40
7392 DATA 45,195,182,202,13
7393 DATA 13,13,121,230,252
7394 DATA 40,34,195,182,202
7395 DATA 12,12,12,121,214
7396 DATA 231,48,23,195,182
7397 DATA 202,4,4,4,120
7398 DATA 214,150,48,12,58
7399 DATA 211,202,61,50,211

7400 DATA 202,254,0,194,54
7401 DATA 202,58,212,202,237
7402 DATA 67,242,201,254,0
7403 DATA 200,118,205,93,210
7404 DATA 201,1,1,7,0
7405 DATA 9,213,1,3,40
7406 DATA 231,48,22,195,232
7407 DATA 202,4,4,4,120
7408 DATA 214,150,48,11,58
7409 DATA 3,203,61,50,3
7410 DATA 203,254,0,32,155
7411 DATA 58,4,203,237,67
7412 DATA 86,202,254,0,200
7413 DATA 118,205,93,210,201
7414 DATA 63,0,0,0,0
```







ANIMATION 2

How does the animation routine work? The answer is simple for you, but not quite so simple for the programmer. What happens with ordinary sprites is that they are picked out of memory, printed on the screen and moved across it, according to the parameters passed to the routine. With animation the first sprite is picked out of memory, displayed on the screen, and moved one position (three pixels in the case of the sprites in this book). The sprite is then deleted from the screen, and the next sprite in the sequence (the equivalent of the next frame in a film, or next drawing in an animated cartoon) receives exactly the same treatment. It is printed on the screen, moved three pixels and then wiped off again.

The sequence continues until all the frames in the sequence have been displayed, and then starts again. Some applications are ideally suited to two frames (like birds flying), but obviously the more frames in the sequence, the smoother the animation will be. For this reason it is best to choose to animate things which have a regular and repetitive movement.

Transferring animation to the screen

Sooner or later you will want to start designing your own animated characters. The technique recommended is to begin by reproducing the movement you see, as simply as possible. Remember that your eye will help by persuading you that things resemble real life, even though they are not. Secondly, remember that the effect you are aiming for is a flowing movement. To achieve this it is necessary to make sure that the last frame in the sequence runs into the first, so that the sequence is circular — after all this is the way it will be projected onto the screen. One of the commonest errors made in animation is to have an open-ended sequence, so that when it is shown repeatedly on the screen the effect is smooth during the sequence, but with a jerk at the end as the sequence restarts.

You are fortunate enough to have the sprite editor and the sprite print routine to help you with your designs. You will find that designing on the screen is much easier than sitting down and working out an image with a pencil and a grid. As a further aid, several animated sequences have been provided in the sprite design directory later in the book. Using the sprite editor, you can start a new sprite design from a previous one and so create smooth and flowing animation sequences with little difficulty.

Trying out the animation

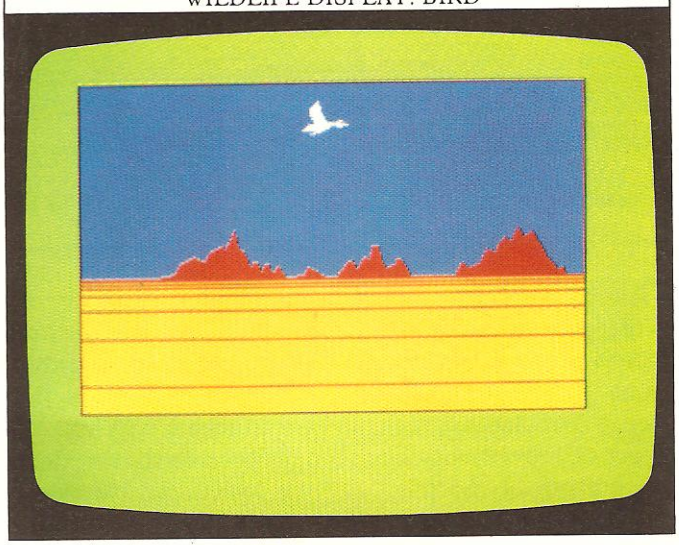
When you have designed an animation sequence you can use the sprite print routine to preview your designs. You saw this idea used with a robot earlier in the book, but the point of using the print routine here is twofold.

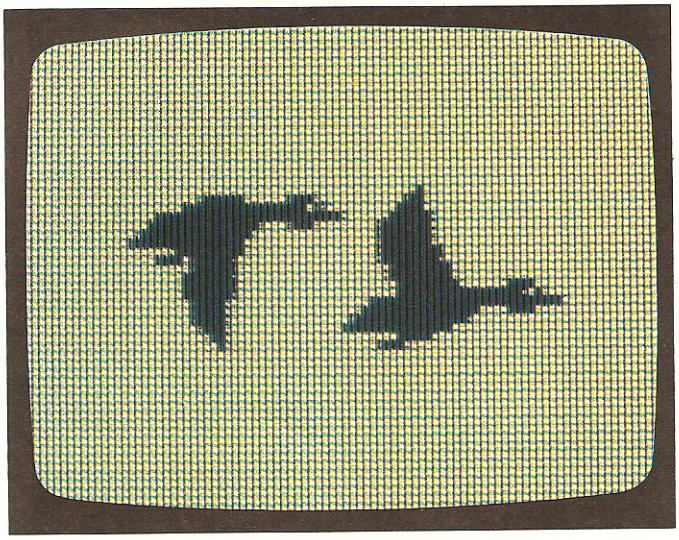
The first point is that it is much easier to be analytical and critical about a sprite design if it is not moving across the screen. If you print your sequence one after the other onto the same screen position in a continuous loop you will get some idea about where changes need to be made (if any) before the sequence is animated properly.

Secondly, you will be able to judge more accurately the sort of speed at which the sequence should be animated. Put a PAUSE statement between each print statement in the loop, starting with values of about 20. You will then be able to see how much the animation routine needs to be slowed down to be most effective. In practice you should multiply the results you get from tests with the PAUSE statements by a factor of between five and ten (that is, PAUSE 20 converts to a value for v of 100), simply to compensate for the fact that machine code is so much faster than BASIC.

It is a good idea to make use of the sprite print routine for one more reason. Once a sprite is being animated with

WILDLIFE DISPLAY: BIRD





the animation routine it is impossible to stop it until the distance l has been covered. This can take quite a while if you have long pauses between each frame of the sequence. However, with the sprite print routine you can break into the program between single frames at any point since the Spectrum is returning to BASIC between each call to the print routine.

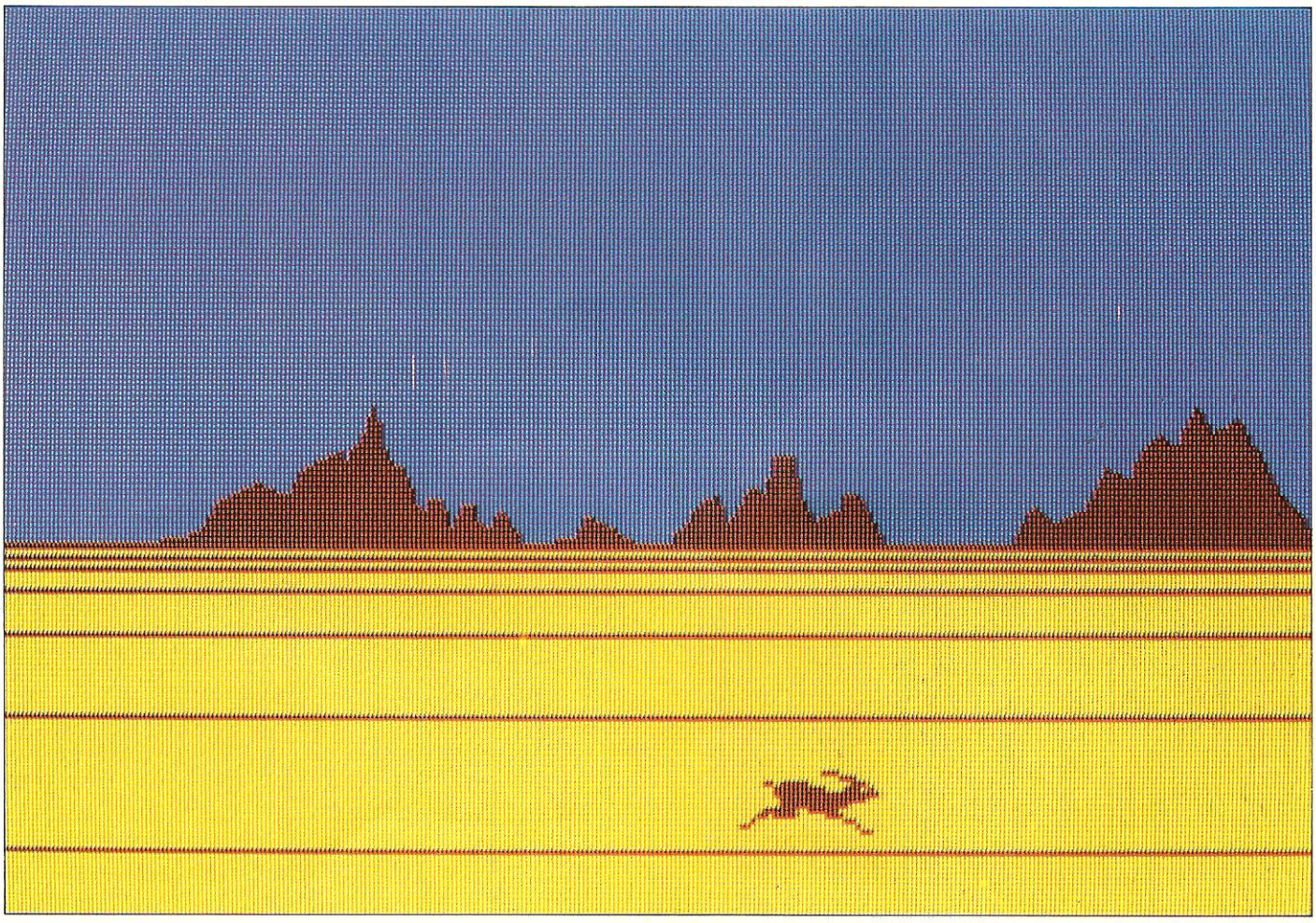
WILDLIFE PROGRAM

```
  10 DEF FN k(x,y,d,l,s,f,c,v,n)
=USR 51700
 100 BORDER 5
 110 RANDOMIZE FN k(10,10,1,65,0
,2,0,50,1)
 120 RANDOMIZE FN k(205,10,0,65,
0,2,0,50,3)
 130 PAUSE 50
 140 RANDOMIZE FN k(10,140,1,65,
0,2,0,100,5)
 150 RANDOMIZE FN k(205,140,0,65
,0,2,0,100,7)
 160 GO TO 100



0 OK, 0:1
```

Once you are completely satisfied with the edited results and the way that they work with the sprite print routine then you should be sure to save the sprites generated to tape, as a separate file. The reason for this is that after all the effort you have spent getting the sprite right it would be a pity to lose them through a clerical error — or a momentary power cut.

The wildlife program

The displays on this page are both produced by a single program. Both displays use two states of animation, and both move the animated sprite across the screen and back again. The movement of the hare (shown in the large display below) is simplified to two states — the elongated position, and the familiar crouching pose. Alternating between these two provides a reasonable approximation of movement. The two bird sprite states, shown in detail in the close-up photograph, show how both the wings and body of the bird are made to move. You may find the bird's movement rather jerky. One way of overcoming this, without increasing the number of frames, would be to reduce the displacement of the body by keeping it more central, rather than rising and falling within the 24 by 21 frame with each animation state.



SCREEN SCROLLING

All the movement you have seen so far relies on the idea of you being stationary and something moving past you. But how can you create the illusion of moving past something which is stationary? The most effective way of doing this is by scrolling the whole screen. There are several ways of doing this; the simplest can be seen whenever you use the BASIC command LIST, which scrolls the screen upwards one character at a time.

The scroll routines

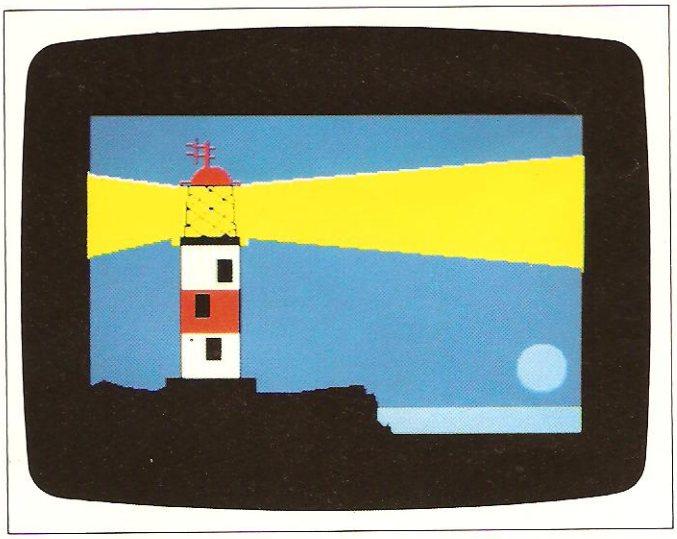
To create a more effective and gentle illusion of movement you need a smoother scroll, which moves the screen one pixel at a time. The two routines given here, FNl and FNm, allow you to scroll the screen a pixel at a time in either a horizontal or a vertical direction.

Note that when using the two scrolling routines on this page, it is inadvisable to use an interrupt-driven routine (such as the keyboard-controlled sprite routine, FNh) at the same time. This is because the scrolling routines are shifting the co-ordinates of screen memory back and forth as the screen is scrolled, and an image is displayed on the screen by interrupts being used to "refresh" the screen at regular intervals. Obviously, this can cause problems if the screen display moves its location in memory when the screen refresh routine looks for it.

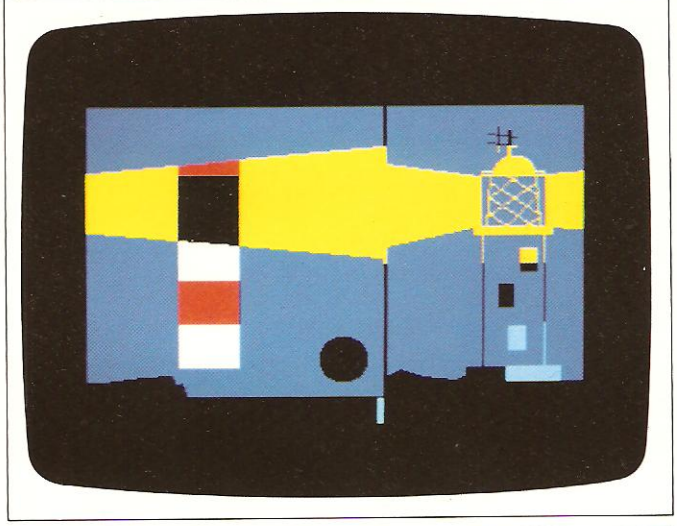
A similar difficulty arises when using PAUSE, since this statement is also based on interrupts. Rather than use PAUSE between two calls to a scroll routine, you are advised to use a FOR...NEXT loop for a delaying effect.

SCREEN SCROLLING PROGRAM

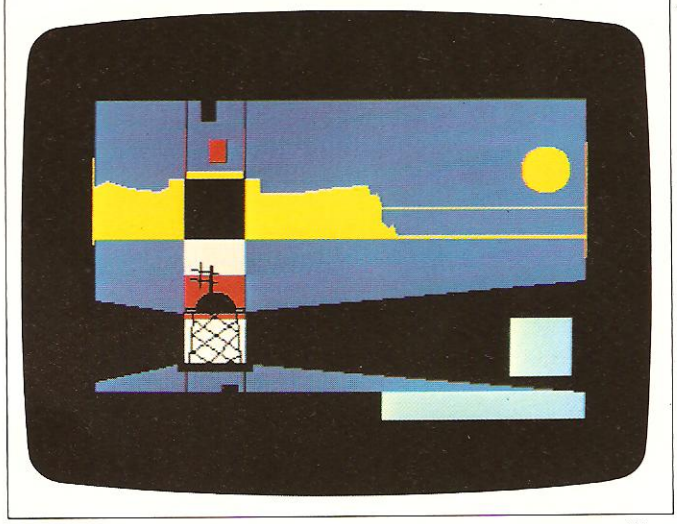
```
 10 LOAD ""SCREEN$
 20 DEF FN l(l,d)=USR 51500
 30 DEF FN m(l,d)=USR 50900
100 RANDOMIZE FN l(100,1)
110 RANDOMIZE FN l(100,0)
120 RANDOMIZE FN m(100,1)
130 RANDOMIZE FN m(100,0)

0 OK, 0:1
```



SCROLLING THE SCREEN HORIZONTALLY



SCROLLING THE SCREEN VERTICALLY



Other scrolling effects

Several effects can be linked to scrolling the screen. Both of the scroll routines have a wraparound effect, which means that whatever goes off one edge of the

FNI

HORIZONTAL SCROLL ROUTINE

Start address 51500 **Length** 190 bytes
What it does Scrolls the screen left or right a specified distance.

Using the routine Parameters d and l set the direction and length of scroll. The routine has a wraparound effect, so that whatever disappears off the left of the screen reappears on the right, and vice versa.

ROUTINE PARAMETERS

DEF FNI (l,d)

l	specifies length of scroll (0–255)
d	specifies direction of scroll (0=left, 1=right)

ROUTINE LISTING

```
7300 LET b=51500: LET l=185: LET
 z=0: RESTORE 7310
7301 FOR i=0 TO l-1: READ a
7302 POKE (b+i),a: LET z=z+a
7303 NEXT i
7304 LET z=INT (((z/l)-INT (z/l)
)*l)
7305 READ a: IF a<>z THEN PRINT
"??": STOP

7310 DATA 243,237,115,141,201
7311 DATA 42,11,92,17,4
7312 DATA 0,25,70,30,8
7313 DATA 25,126,254,0,32
7314 DATA 38,14,191,197,33
7315 DATA 0,64,229,6,32
7316 DATA 167,203,30,35,16
7317 DATA 251,225,48,4,62
7318 DATA 128,182,119,17,32
7319 DATA 0,25,13,32,233

7320 DATA 193,16,224,237,123
7321 DATA 141,201,251,201,14
7322 DATA 191,197,33,31,64
7323 DATA 229,6,32,167,203
7324 DATA 22,43,16,251,225
7325 DATA 48,4,62,1,182
7326 DATA 119,17,32,0,25
7327 DATA 13,32,233,193,16
7328 DATA 224,201,237,123,141
7329 DATA 201,251,0,0,0

7330 DATA 42,11,92,17,4
7331 DATA 0,25,70,30,8
7332 DATA 25,126,254,0,32
7333 DATA 33,14,191,197,33
7334 DATA 0,64,229,6,32
7335 DATA 167,203,30,35,16
7336 DATA 251,225,48,4,62
7337 DATA 128,182,119,17,32
7338 DATA 0,25,13,32,233
7339 DATA 193,16,224,201,14

7340 DATA 191,197,33,31,64
7341 DATA 229,6,32,167,203
7342 DATA 22,43,16,251,225
7343 DATA 48,4,62,1,182
7344 DATA 119,17,32,0,25
7345 DATA 13,32,233,193,16
7346 DATA 224,201,0,0,0
7347 DATA 61,0,0,0,0
```

screen then reappears on the opposite edge. Other routines allow you to leave something stationary on the screen while scrolling the rest — as used in popular games like Defender and Pole Position. To do this requires a much longer routine than those given here.

However, another type of scrolling effect has been included in this book. This is a partial screen scroll, in which the vertical dimension of the area scrolled is

FNm

VERTICAL SCROLL ROUTINE

Start address 50900 **Length** 215 bytes
What it does Scrolls the screen a specified distance up or down.

Using the routine Parameters are the same as for the horizontal scroll routine. As before, a wraparound effect occurs with the routine, but in this case when scrolling off the top and bottom of the screen.

ROUTINE PARAMETERS

DEF FNm(l,d)

l	specifies length of scroll (0–175)
d	specifies direction of scroll (0=up, 1=down)

ROUTINE LISTING

```
7200 LET b=50900: LET l=210: LET
 z=0: RESTORE 7210
7201 FOR i=0 TO l-1: READ a
7202 POKE (b+i),a: LET z=z+a
7203 NEXT i
7204 LET z=INT (((z/l)-INT (z/l)
)*l)
7205 READ a: IF a<>z THEN PRINT
"??": STOP

7210 DATA 243,42,11,92,17
7211 DATA 4,0,25,70,30
7212 DATA 8,25,126,230,1
7213 DATA 40,91,197,1,32
7214 DATA 0,33,0,64,17
7215 DATA 162,199,126,18,35
7216 DATA 19,16,250,1,175
7217 DATA 0,33,0,64,17
7218 DATA 0,65,213,229,6
7219 DATA 32,26,119,35,19

7220 DATA 16,250,225,209,36
7221 DATA 62,7,164,32,10
7222 DATA 62,32,133,111,40
7223 DATA 4,124,214,8,103
7224 DATA 20,62,7,162,32
7225 DATA 10,62,32,131,95
7226 DATA 40,4,122,214,8
7227 DATA 87,13,32,209,6
7228 DATA 32,33,162,199,17
7229 DATA 160,87,126,18,35

7230 DATA 19,16,250,193,16
7231 DATA 167,251,201,197,33
7232 DATA 160,87,17,162,199
7233 DATA 6,32,126,18,35
7234 DATA 19,16,250,1,175
7235 DATA 0,33,160,86,17
7236 DATA 160,87,213,229,6
7237 DATA 32,126,18,35,19
7238 DATA 16,250,225,209,37
7239 DATA 124,230,7,254,7

7240 DATA 32,12,125,214,32
7241 DATA 111,254,224,40,4
7242 DATA 62,8,132,103,21
7243 DATA 122,230,7,254,7
7244 DATA 32,12,123,214,32
7245 DATA 95,254,224,40,4
7246 DATA 62,8,130,87,13
7247 DATA 32,201,17,0,64
7248 DATA 33,162,199,6,32
7249 DATA 126,18,35,19,16
7250 DATA 250,193,16,160,251
7251 DATA 201,0,0,0,0
7252 DATA 47,0,0,0,0
```

restricted to the size of the sprites used in this book. This partial screen scroll routine (given on page 30) produces what is in effect a window.

The program on this page shows scrolling at work. You will notice that the routine has the effect of moving ink attributes while leaving coloured areas unchanged.

WINDOWS 1

As you saw on pages 28-29, it is often more useful to be able to scroll parts of the screen than to scroll all of it. The window routine given here, FNn, enables you to do this. With the routine, you can define an area of the screen three characters deep of any width, and then move a sprite across it. The routine enables you to make sprites appear to move behind objects on the screen, since they appear from outside the window and then disappear the other side — rather like looking out of a window and watching a train go past.

Repeating sprites

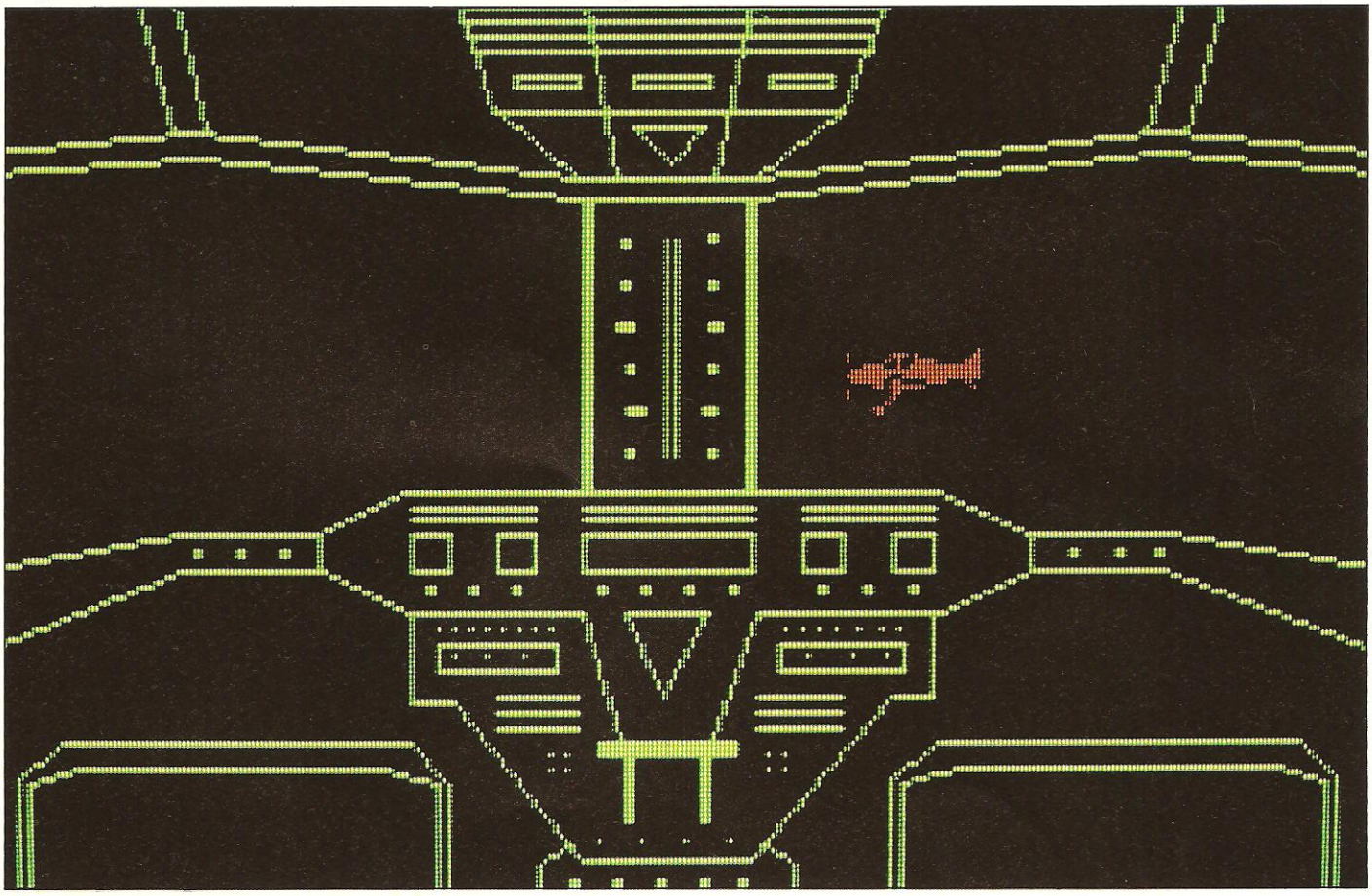
One additional feature of the window routine is that you can repeat the sprite to give the effect of a chain of sprites passing through the window. Alternatively, quite spectacular effects can be achieved by having a variety of sprites set up in the sprite table, like the sequence produced by the cockpit program shown on this page.

Differences between sprites and windows

In many cases, you may find it more useful in your programs to use this window routine than a standard sprite routine. However, when using the routine, it is important to remember that the routine is essentially a scrolling, rather than a sprite routine, and that everything contained in the window will be scrolled by the routine, even if it was only part of the original background. To display a sprite moving against a static background, or to move a sprite vertically, you must use one of the sprite routines.

COCKPIT PROGRAM

```
  10 DEF FN n(x,y,t,n,d,r)=USR 4
9600
 100 FOR x=1 TO 10
 110 IF x<>2 AND x<>5 AND x<>7 T
HEN RANDOMIZE FN n(18,8,13,x,1,1
): RANDOMIZE FN n(1,8,13,x,1,1):
 PAUSE 50
 120 IF x=2 OR x=5 OR x>=7 THEN
RANDOMIZE FN n(1,8,13,x,0,1): RA
NDOMIZE FN n(18,8,13,x,0,1): PAU
SE 50
 130 NEXT x
 140 GO TO 100



0 OK, 0:1
```



FNn

WINDOW ROUTINE

Start address 49600 **Length** 290 bytes
What it does Moves one or several sprites across a window of specified dimension.

Using the routine The routine carries out a partial screen scroll: everything contained within the area defined by the parameters is scrolled.
Note that the start co-ordinates x,y represent the top left hand corner of the sprite if the sprite moves right, but the top right-hand corner of the sprite if the sprite moves left. The repeat flag, r, can be used to repeat a sequence of sprites moving across the window. Repeated sprites can be seen in the window game program on pages 32-33.

ROUTINE PARAMETERS

DEF FNn(x,y,l,n,d,r)	
x,y	start co-ordinates (0<=x<=31, 0<=y<=21)
l	width of the window (0<=l<=31)
n	number of sprite to be scrolled (n=1-10)
d	specifies direction of scroll (0=right, 1=left)
r	repeat flag (1=switch off repeat, 0=repeat)

ROUTINE LISTING

```
7100 LET b=49600: LET l=285: LET
  z=0: RESTORE 7110
7101 FOR i=0 TO l-1: READ a
7102 POKE (b+i),a: LET z=z+a
7103 NEXT i
7104 LET z=INT (((z/l)-INT (z/l)
)*l)
7105 READ a: IF a<>z THEN PRINT
"??": STOP

7110 DATA 42,11,92,1,4
7111 DATA 0,9,86,14,8
7112 DATA 9,94,9,126,50
7113 DATA 218,194,9,126,50
7114 DATA 223,194,9,126,230
7115 DATA 1,50,222,194,9
7116 DATA 126,230,1,50,221
7117 DATA 194,62,1,50,219
7118 DATA 194,50,224,194,123
7119 DATA 230,24,246,64,103

7120 DATA 123,230,7,183,31
7121 DATA 31,31,31,130,111
7122 DATA 58,222,194,254,0
7123 DATA 40,6,58,218,194
7124 DATA 133,61,111,58,218
7125 DATA 194,60,60,60,71
7126 DATA 197,229,205,102,194
7127 DATA 225,6,8,229,197
7128 DATA 17,225,194,58,222
7129 DATA 194,254,0,40,3

7130 DATA 17,11,195,6,3
7131 DATA 229,197,120,254,1
7132 DATA 32,4,6,5,24
7133 DATA 2,6,8,197,213
7134 DATA 229,58,222,194,254
7135 DATA 0,32,5,205,160
7136 DATA 194,24,3,205,189
7137 DATA 194,225,209,193,36
7138 DATA 19,16,231,193,225
7139 DATA 62,32,133,111,48

7140 DATA 4,62,8,132,103
7141 DATA 16,204,193,225,16
7142 DATA 183,193,118,16,171
7143 DATA 201,58,224,194,61
7144 DATA 50,224,194,192,62
7145 DATA 3,50,224,194,58
7146 DATA 223,194,17,63,0
7147 DATA 71,33,72,213,167
7148 DATA 237,82,25,16,253
7149 DATA 67,17,225,194,58

7150 DATA 219,194,254,0,40
7151 DATA 1,126,18,35,19
7152 DATA 16,243,58,221,194
7153 DATA 254,0,200,62,0
7154 DATA 50,219,194,201,167
7155 DATA 6,3,213,26,31
7156 DATA 18,245,62,21,131
7157 DATA 95,48,1,20,241
7158 DATA 16,242,209,58,218
7159 DATA 194,71,203,30,35

7160 DATA 16,251,201,167,213
7161 DATA 6,3,26,23,18
7162 DATA 245,123,214,21,95
7163 DATA 48,1,21,241,16
7164 DATA 242,209,58,218,194
7165 DATA 71,203,22,43,16
7166 DATA 251,201,10,0,0
7167 DATA 43,0,0,0,0
```

The cockpit program

The large display on this page shows windows at work. From the interior of an aircraft cockpit, various air- and spacecraft can be seen flying across in front of the plane; the display shows one of these. What is particularly effective is the way the sprites disappear behind the centre pillar and reappear in the other window — an effect which could not be achieved with the sprite routines. This program, then, shows the real difference between windows and sprites — using windows you can make the sprites look as if they are behind a solid object rather than in front of it.

How the program works

Line 10 of the program defines the window routine, together with its assortment of parameters. As you can see, three characters wide is a very effective width for the routine. The program actually uses two windows, three characters apart, with a barrier (the window frame) in the middle of the screen. The eye, however, is tricked into believing that the sprites are moving along behind the windows when they are in fact disappearing, stopping, and then reappearing as a result of a second window call. Line 100 sets up a loop so that all ten sprites can be made to appear across the window, one by one.

Line 120 deals with the sprites that are left facing — that is, all of the sprites which appear to fly from right to left. This works by testing for the second, fifth and seventh sprites which are right facing. If any other sprite is to be used then it is first made to go across the right-hand window, immediately followed by a call to make it go across the left-hand window. A PAUSE statement has been added to give you time to think about what just flew in front of your eyes!

Line 130 behaves in the same way, but handles the right facing sprites. Sprites 2, 5, and 7 and upwards appear to face to the right, so these are made to fly across the left-hand window first, immediately followed by a second call to make them fly across the right-hand one. Line 140 ends the loop, and finally line 150 starts the whole process over again.

WINDOWS 2

The window routine given on this page has all the features of the window routine, FNn, but with the added feature of being interrupt-driven. Once switched on, it will keep going until switched off again, regardless of just about any BASIC command.

Your first priority with this routine must be to have a way of switching it off. As for keyboard-controlled sprites, this is done by a subroutine:

```
2000 DEF FN z(s)=USR 53100
2010 DEF FN x(s)=USR 49200
2020 RANDOMIZE FN z(1)
2030 RANDOMIZE FN z(0)
2040 RANDOMIZE FN x(0)
2050 RETURN
```

This subroutine is slightly more complicated than you would expect, since it is used both to set up the routine and to switch it off again. The routine also switches on the keyboard-controlled sprite routine, since this sets up the interrupt vector table required for the interrupt-driven window routine to work correctly.

WINDOW GAME PROGRAM

```
 10 DEF FN h(s,x,y,c,n)=USR 531
00
 20 DEF FN g(x,y,d,l,s,c,n)=USR
 53500
 30 DEF FN o(s,x,y,l,n,d,r)=USR
 49200
 40 DEF FN z(s)=USR 49200
 50 DEF FN y(s)=USR 53100
100 BORDER 1: PAPER 7: CLS
110 LET sc=0: LET psc=0: LET g=
-1: LET ng=0
120 FOR i=1 TO 21
130 PRINT INK 6;AT i,0;"▀▀▀▀▀"
140 PRINT INK 4;AT i,28;"▀▀▀▀▀"
150 PRINT INK 2;AT 1,i;"▀▀▀▀▀
▀▀▀"
160 NEXT i
170 PRINT INK 1;AT 5,4;"▀  ▀▀▀▀
 ▀▀▀▀▀▀  ▀▀▀▀▀  ▀▀"
180 GO SUB 500
190 RANDOMIZE FN o(1,4,2,24,3,0
,0)

scroll?
```

```
200 RANDOMIZE FN h(1,32,150,0,1
0)
210 IF INKEY$=" " THEN GO TO 23
0
220 GO TO 210
230 LET xp=PEEK 53098
240 LET yp=PEEK 53099
250 IF xp>=32 AND xp<=44 THEN L
ET sc=sc+2: LET ng=1
260 IF xp>=80 AND xp<=92 THEN L
ET sc=sc+2: LET ng=2
270 IF xp>=142 AND xp<=152 THEN
 LET sc=sc+2: LET ng=3
280 IF xp>=190 AND xp<=200 THEN
 LET sc=sc+2: LET ng=4
290 LET sc=sc-1
300 IF sc+1=psc THEN LET ng=g+1
310 IF ng=g THEN LET sc=sc-1: G
O TO 340
320 RANDOMIZE FN g(xp,yp-20,3,I
NT ((yp/3)),0,1,1)
330 IF INKEY$<>"" THEN GO TO 33

scroll?
```

WINDOW GAME PROGRAM CONTD.

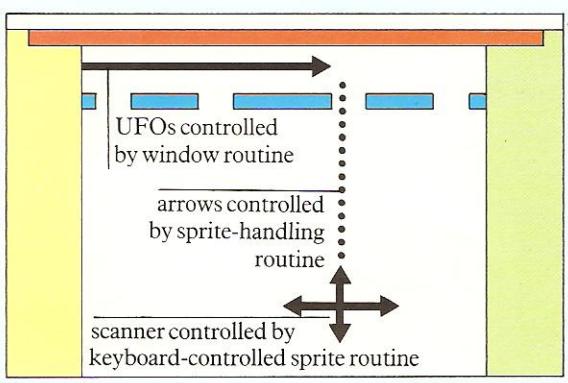
```
0
 340 IF sc<0 THEN LET sc=0
 350 LET g=ng: LET psc=sc
 360 GO SUB 500
 370 GO TO 210
 380 STOP
 500 PRINT PAPER 6; INK 0;AT 0,1
5;sc;
 510 RETURN
1000 RANDOMIZE FN y(1)
1010 RANDOMIZE FN y(0)
1020 RANDOMIZE FN z(0)




0 OK, 0:1
```

The interrupt-driven routine can be used together with the other routines in this book to set up and run sophisticated games. A simple game is given here; you will be able to improve it with a little effort.

The window game

This game is based on the idea of shooting at moving objects at the top of the screen. The game uses three machine-code routines. An interrupt-driven window is used to make flying saucers scroll across the top of the screen, the keyboard-controlled sprite routine is used to control your scanner, and the sprite-handling routine is used to send arrows towards the saucers.

WINDOW GAME SPRITE MOVEMENT



One of the best parts about writing a computer game is in deciding the scenario for the game. In this case you could imagine, for example, you are trapped on earth with only a scanner satellite, and your crew aboard it

FNo

INTERRUPT-DRIVEN WINDOW ROUTINE

Start address 49200 **Length** 315 bytes
Other routines called Keyboard-controlled sprite routine (FNh).
What it does Moves either one or several sprites across a window, and continues to operate whatever is happening in BASIC.

Using the routine To use this routine you must first switch on the keyboard routine, since this routine sets up tables of interrupts where needed. The keyboard sprite routine can then be switched off again, unless it is called in the program elsewhere. Use s in the same way as it is used in the keyboard controlled sprite routine.
Note that the start co-ordinates x,y of the window routine represent the top left-hand corner of the sprite if the sprite moves right and the top right-hand corner if it moves left.

ROUTINE PARAMETERS

DEF FNo(s,x,y,l,n,d,r)

s	stop flag (1=stop, 0=normal)
x,y	start co-ordinates (0<=x<=31, 0<=y<=21)
l	width of window (0<=l<=31)
n	number of sprite to be scrolled (n=1-10)
d	specifies direction of scroll (0=right, 1=left)
r	repeat flag (1=switch off repeat, 0=repeat)

have only a bow and arrow with which to repel invaders. You can also use other sprites with the same background to change the feel of the game. Using different sprites, you could convert this game to a fairground shooting gallery, for example.

Several refinements could be made to the program. One sensible one would be to stop the keyboard sprite from going too high up the screen. This could be done by testing the current y co-ordinate of the sprite (stored at location 53098), and starting the sprite again at the bottom of the screen if a given limit is exceeded.

How the program works

Lines 10-20 define the three machine-code routines. Two of these routines are interrupt-driven and will therefore have to be switched off at some point. Lines 40 and 50 are here for this reason. These extra function definitions enable you to switch the interrupt-driven routines off and on by calling the routine with just one parameter, without bothering about specifying all the other parameters normally required.

Line 110 sets up some variables. The current score is held as variable sc, and the previous score (before the current arrow was fired) as psc. Variables g and ng are used to record the position of the keyboard sprite under the gaps, with ng defining the gap under which the sprite is resting, and g the previous gap. These variables can have values between 1 and 4 (since there are four gaps through which you can fire), and these variables are used to restrict your firing. The program compares variables g and ng in line 310, and if they are the same, the fire sequence is bypassed. At the beginning of the game, it is important that g and ng have different values, so that the fire sequence is not disallowed initially. Thus, g is set at first to -1, a value which ng can never have.

ROUTINE LISTING

```
7000 LET b=49200: LET l=310: LET
 z=0: RESTORE 7010
7001 FOR i=0 TO l-1: READ a
7002 POKE (b+i),a: LET z=z+a
7003 NEXT i
7004 LET z=INT (((z/l)-INT (z/l)
)*l)
7005 READ a: IF a<>z THEN PRINT
"??": STOP

7010 DATA 243,42,11,92,1
7011 DATA 4,0,9,126,254
7012 DATA 0,32,9,17,252
7013 DATA 207,237,83,250,207
7014 DATA 251,201,17,139,192
7015 DATA 237,83,250,207,14
7016 DATA 8,9,86,9,94
7017 DATA 9,126,50,100,193
7018 DATA 9,126,50,108,193
7019 DATA 9,126,230,1,50

7020 DATA 107,193,9,126,230
7021 DATA 1,50,106,193,62
7022 DATA 1,50,104,193,50
7023 DATA 109,193,50,101,193
7024 DATA 123,230,24,246,64
7025 DATA 103,123,230,7,183
7026 DATA 31,31,31,31,130
7027 DATA 111,34,102,193,251
7028 DATA 201,58,107,193,254
7029 DATA 0,40,6,58,100

7030 DATA 193,133,61,111,58
7031 DATA 101,193,71,16,5
7032 DATA 205,240,192,6,8
7033 DATA 120,50,101,193,17
7034 DATA 110,193,42,102,193
7035 DATA 58,107,193,254,0
7036 DATA 40,3,17,152,193
7037 DATA 6,3,229,197,120
7038 DATA 254,1,32,4,6
7039 DATA 5,24,2,6,8

7040 DATA 197,213,229,58,107
7041 DATA 193,254,0,32,5
7042 DATA 205,42,193,24,3
7043 DATA 205,71,193,225,209
7044 DATA 193,36,19,16,231
7045 DATA 193,225,62,32,133
7046 DATA 111,48,4,62,8
7047 DATA 132,103,16,204,195
7048 DATA 252,207,58,109,193
7049 DATA 61,50,109,193,192

7050 DATA 62,3,50,109,193
7051 DATA 58,108,193,17,63
7052 DATA 0,71,33,72,213
7053 DATA 167,237,82,25,16
7054 DATA 253,67,17,110,193
7055 DATA 58,104,193,254,0
7056 DATA 40,1,126,18,35
7057 DATA 19,16,243,58,106
7058 DATA 193,254,0,200,62
7059 DATA 0,50,104,193,201

7060 DATA 167,6,3,213,26
7061 DATA 31,18,245,62,21
7062 DATA 131,95,48,1,20
7063 DATA 241,16,242,209,58
7064 DATA 100,193,71,203,30
7065 DATA 35,16,251,201,167
7066 DATA 213,6,3,26,23
7067 DATA 18,245,123,214,21
7068 DATA 95,48,1,21,241
7069 DATA 16,242,209,58,100
7070 DATA 193,71,203,22,43
7071 DATA 16,251,201,5,4
7072 DATA 154,0,0,0,0
```

WINDOWS 3

Lines 120 to 170 of the program draw the background, with a loop used to create the sides and top, and line 170 creating the gaps through which you can fire. Line 180 calls the score subroutine at line 500.

Line 190 starts off the interrupt-driven window, though it does not begin working until the interrupt vector table has been set up by the keyboard-controlled sprite routine. Because this is the very next statement you do not see any delay. but it is important to remember that the interrupt-driven window will not function unless this table is in place. The window routine will continue to scroll a sprite across the window until something occurs to stop it.

Lines 210-220 form the main program loop, which simply waits for the space key (the signal to fire) to be pressed, since everything else at this stage is interrupt-controlled. The controlling keys for the keyboard sprite must of course be ignored, as their function is handled by the machine-code routine.

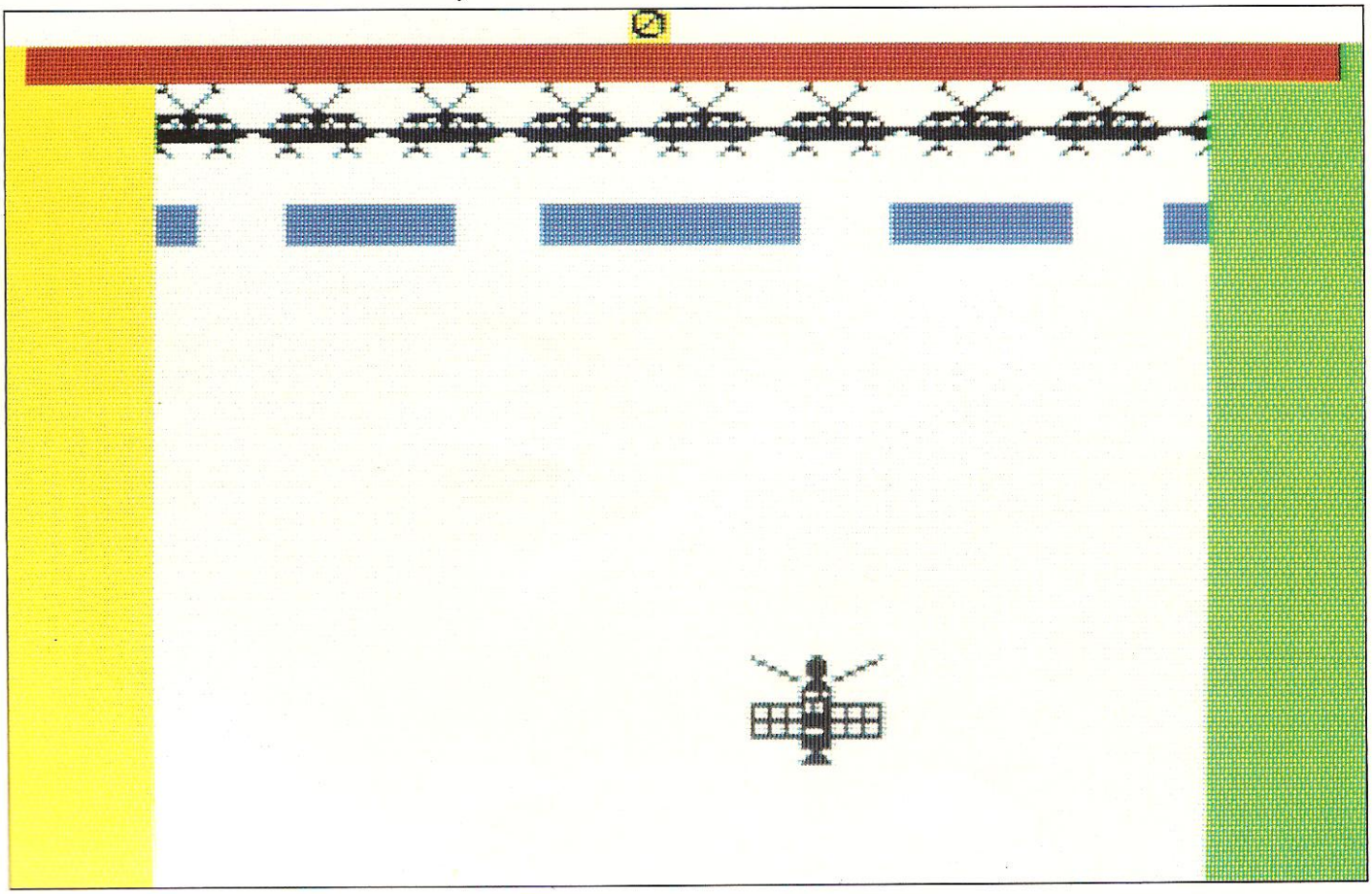
Lines 230 and 240 find out the current position of the keyboard-controlled sprite. Given the position of the sprite, and the gaps in the barrier, the program can decide whether a shot fired would go through a gap.

Lines 250-280 work in the same way for each of the four gaps. First a test is made to see whether the keyboard sprite is under the current gap. If it is then the gap flag (ng) is set to the value of the current gap. In addition the current score is incremented by two — not one as you might expect. Line 290 normalizes the score by taking one away again. Note that if the keyboard sprite is not under one of the gaps then the score is now one less than it was before the shot was fired.

Line 300 looks at the past score compared with the present score plus 1, and makes the gap variables different from their previous values, if the sprite was not under a gap. Line 310 deducts another one from the score if the gap is the same as the last gap which was fired at. It also skips the arrow-firing sequence, since arrows are only fired if the gap is a new one.

Line 280 fires an arrow by calling the sprite routine. Line 340 makes sure that the score cannot become negative, however bad the player. Line 350 adjusts the values of the previous score and the old gap value, ready for the next time round the loop. Line 360 prints the new score.

Finally, lines 1000-1020 switch off the interrupt-driven routines. To break out of the program, key GO TO 1000 to stop the routines.



USING THE SPRITE DIRECTORY

While using this book, you will have found that producing good sprite designs is not always easy. To overcome this problem, you can turn to the sprite directory, which contains over 200 sprite designs. These sprites can either be copied directly in your own programs, or used as a model on which to base your own ideas. Each entry in the directory includes the DATA for the sprite and shows the sprite on the screen.

Keying in the sprites

A sprite from the directory can be keyed in by using the following loading program (add DATA from line 100):

```
10 INPUT "Sprite number (1-10)",a
20 IF a<0 OR a>10 THEN GO TO 10
30 LET b=54600+((a-1)*63)
40 FOR i=b TO b+20
50 READ n : POKE i,n
60 READ n : POKE i+21,n
70 READ n : POKE i+42,n
80 NEXT i
100 DATA 0,0,0,0,etc
```

The routine loads the sprite in the DATA statements into the sprite location specified by you at the start of this BASIC program. The routine requires you to key in all 63 sprite DATA items, even if some of them are zero. The sprite table can easily be corrupted by wrong DATA being entered, but the easiest way to correct this is by editing the sprite image with the sprite editor.

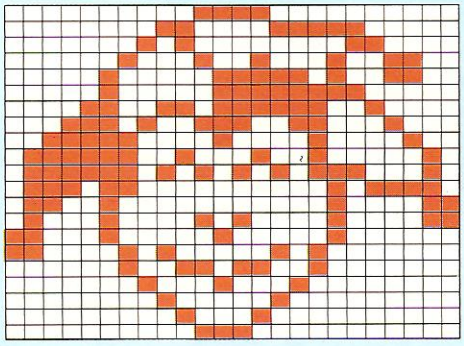
What the directory contains

The directory groups various kinds of sprite under theme headings. Most of the sprites are designed to be used individually, but the directory also contains some double sprites, which are used in conjunction with each other. In addition, a number of animation sequences of sprites in two or three different positions are included. The sprites can be entered either by keying in the DATA numbers provided, or simply by copying the drawing using the sprite editor. The sprite editor can also be used to increase the number of frames in an animation sequence, up to a maximum of ten designs (the maximum number of sprites that can be held in the buffer at one time).

EXAMPLES OF SPRITES

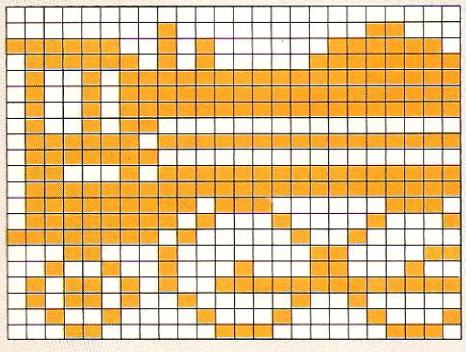
JOKER

```
0,60,0,0,195,224,1
64,48,2,36,76,4,31
76,4,31,224,12,223,144
29,57,136,62,16,132,126
68,130,126,170,226,118,0
94,100,0,67,68,40,67
196,16,64,194,130,128,2
108,128,1,17,0,0,130
0,0,68,0,0,56,0
```



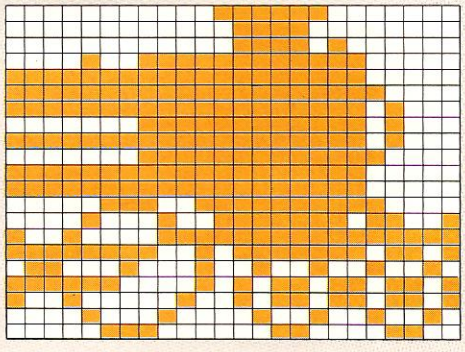
PACIFIC-TYPE LOCO

```
0,0,0,0,128,24,252
160,126,126,160,255,75,255
255,75,255,255,75,255,255
75,0,0,126,255,255,127
0,0,98,255,255,127,255
255,127,220,14,127,162,17
255,65,32,24,128,195,36
136,196,90,159,255,90,65
32,36,34,17,24,28,14
```



PACIFIC-TYPE LOCO

```
0,31,128,0,15,0,0
15,64,9,255,224,255,255
224,255,255,240,255,255,232
1,255,232,255,255,232,1
255,240,255,255,224,255,255
224,7,63,240,8,159,217
144,255,255,255,47,237,98
36,146,252,43,109,144,75
109,8,132,146,7,3,12
```



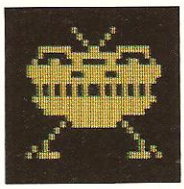
HOW SPRITES ARE SHOWN

The diagrams shown illustrate how sprites are displayed in the directory. Each sprite is shown in the top left-hand corner as it appears on the screen, and in the large display below as it can be keyed in on a grid. The 63 DATA numbers which are POKEd into memory are shown on the top right of each sprite display.

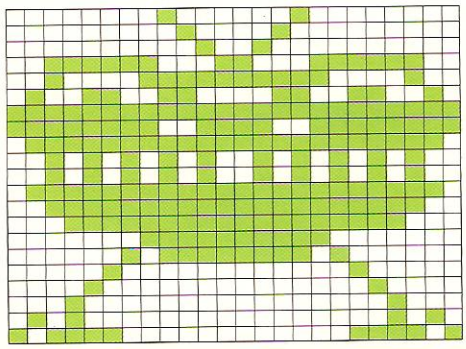
Note that double sprites are shown as two separate sprites; obviously, the sprites will appear joined when used with the double horizontal sprite routine (FNi).

ALIENS

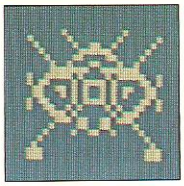
BUG



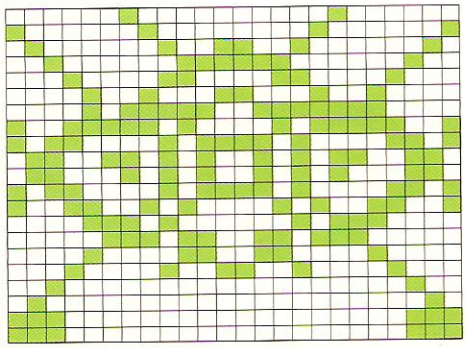
0,129,0,0,66,0,0
36,0,63,24,252,33,255
132,76,195,50,255,255,255
255,60,255,127,255,254,42
165,84,42,165,84,127,255
254,63,255,252,31,255,248
1,255,128,2,255,64,4
0,32,8,0,16,16,0
8,80,0,10,188,0,61



BUG

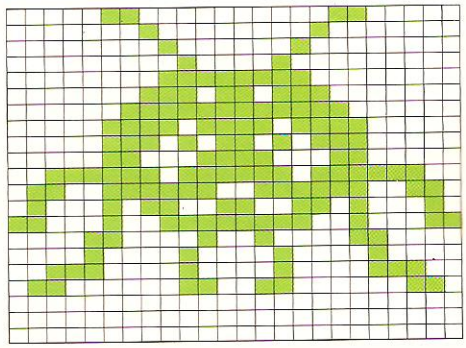


2,0,64,129,0,129,64
153,2,32,126,4,16,195
8,9,24,144,15,231,240
159,66,249,176,189,13,102
165,102,98,165,70,176,189
13,153,66,153,14,129,112
7,102,224,8,102,16,16
153,8,32,0,4,64,0
2,224,0,7,224,0,7

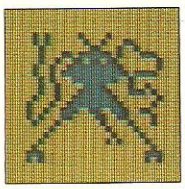


MICRO-MITE

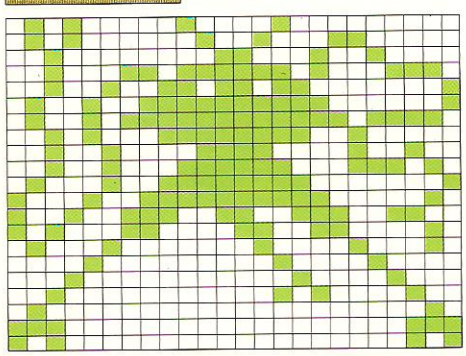
6,0,96,1,0,128,0
129,0,0,66,0,0,255
0,1,219,128,3,255,192
7,189,224,7,90,224,6
60,96,63,126,252,103,231
230,69,153,162,196,255,35
12,36,48,8,66,16,56
66,28,96,102,6,0,0
0,0,0,0,0,0,0



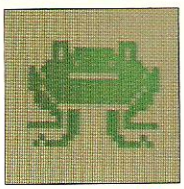
TRIPOD



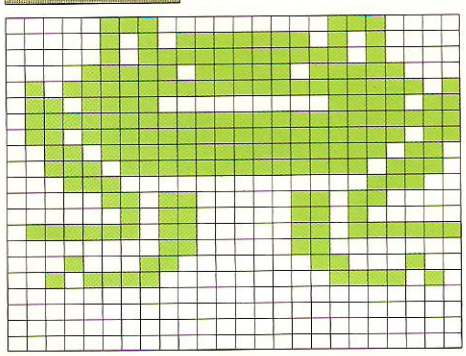
80,66,0,80,36,96,32
24,144,35,126,142,36,219
1,73,255,129,75,255,222
40,255,32,36,60,36,36
126,58,68,255,1,137,255
130,147,153,204,163,12,194
68,4,34,8,2,16,16
1,8,32,2,132,64,0
2,224,0,7,160,0,5



HOPPER

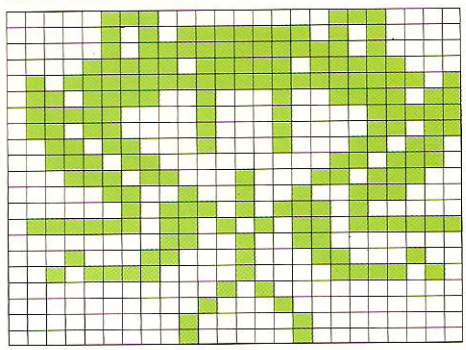


7,0,112,5,127,80,5
255,208,15,190,248,95,255
253,111,0,123,127,255,255
111,255,251,55,255,246,59
255,238,28,0,28,6,193
176,2,193,160,126,193,191
0,193,128,17,128,196,47
0,122,0,0,0,0,0
0,0,0,0,0,0,0

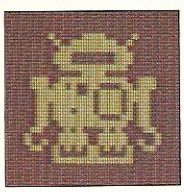


HOPPER

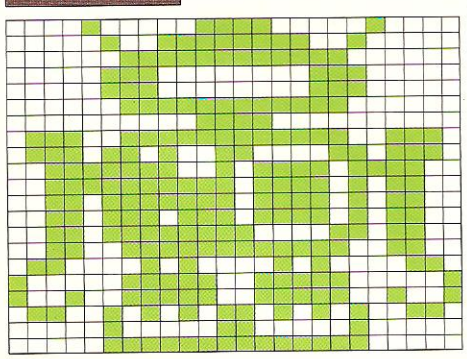
7,0,112,5,127,80,5
255,208,15,190,248,95,255
253,111,34,123,124,34,31
108,34,27,54,34,54,59
0,110,28,200,220,6,73
48,2,235,160,126,213,191
0,201,128,17,136,196,47
8,122,0,20,0,0,34
0,0,65,0,0,65,0



ROBOT

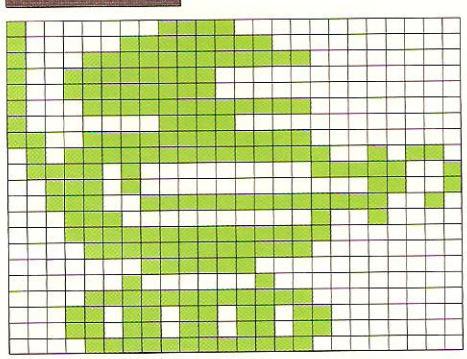


9,60,16,4,255,32,3
255,192,7,0,224,7,0
224,3,255,192,0,60,0
115,255,206,118,152,110,63
151,188,63,247,188,55,247
172,55,119,172,55,248,108
51,255,204,97,66,134,131
231,193,147,231,201,101,90
166,4,24,32,7,255,224

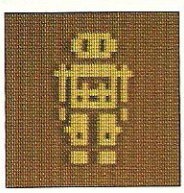


ROBOT

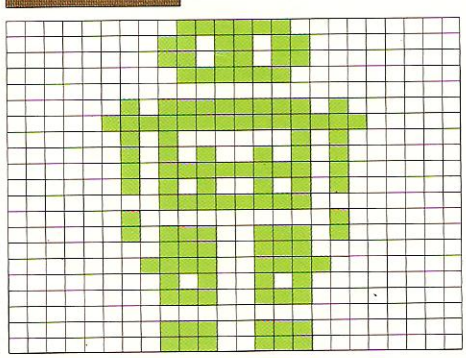
128,240,0,131,252,0,143
255,0,159,224,0,159,224
0,143,255,0,128,240,0
207,255,0,95,255,150,122
0,249,58,0,25,57,255
242,28,0,16,31,255,240
15,255,224,1,254,0,2
1,0,15,255,224,27,109
176,27,109,176,15,255,224



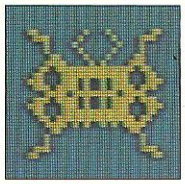
ROBOT



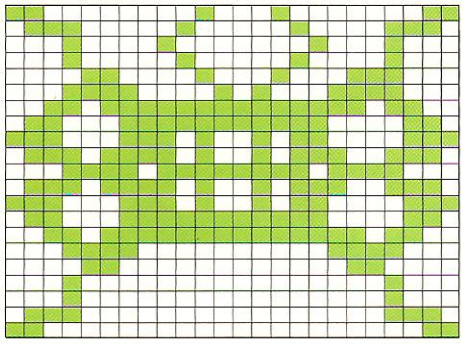
0,126,0,0,219,0,0
219,0,0,126,0,0,0
0,2,255,64,7,255,224
2,129,64,2,165,64,2
255,64,2,165,64,0,255
0,2,0,64,2,231,64
0,231,0,1,231,128,0
165,0,0,231,0,0,0
0,0,231,0,0,231,0



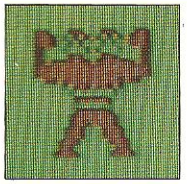
SQUAROID



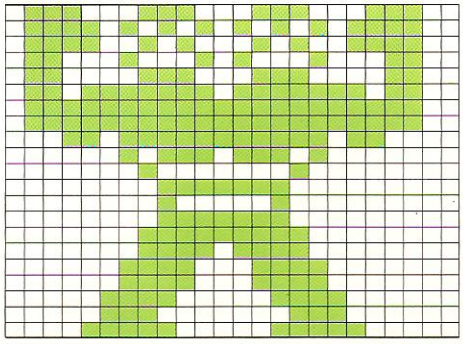
192,34,3,112,65,14,16
128,136,16,65,8,12,34
24,30,20,56,55,255,236
99,255,198,227,165,199,119
165,238,30,255,120,119,165
238,227,165,199,99,255,198
55,255,236,30,165,120,12
0,48,16,0,8,16,0
8,112,0,14,192,0,3



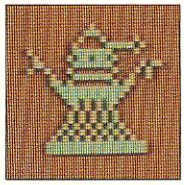
HUMANOID



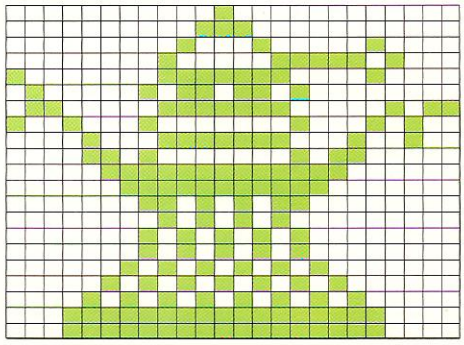
113,199,28,122,170,188,121
69,60,98,40,140,97,199
12,111,239,236,127,255,252
127,255,252,27,255,176,2
238,128,1,1,0,0,254
0,0,130,0,0,254,0
1,255,0,1,239,0,3
199,128,3,199,128,7,131
192,7,1,192,15,131,224



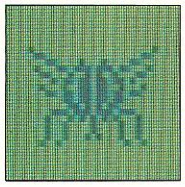
"DALEK"



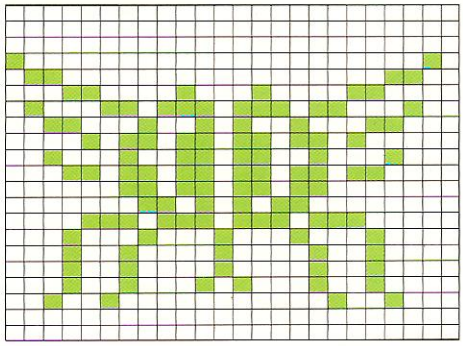
0,16,0,0,56,0,0
68,16,0,131,232,128,254
16,65,109,0,96,254,11
145,1,20,8,254,36,13
1,96,7,255,192,3,255
128,1,171,0,0,170,0
1,85,0,1,85,0,2
170,128,5,85,64,10,170
160,31,255,248,31,255,248



SPACE-FLY

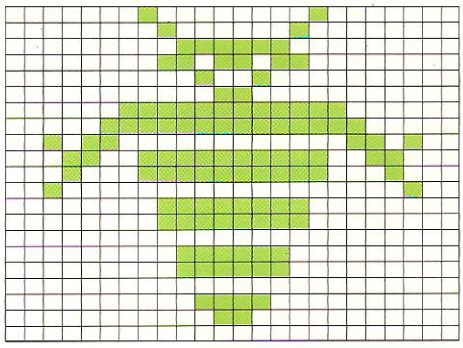


0,0,0,0,0,0,0
0,0,128,0,2,96,0
12,24,68,48,70,238,196
49,41,24,11,109,160,34
108,136,27,109,176,3,109
128,1,171,0,14,238,224
17,85,16,18,16,144,18
16,144,18,40,144,36,0
72,0,0,0,0,0,0



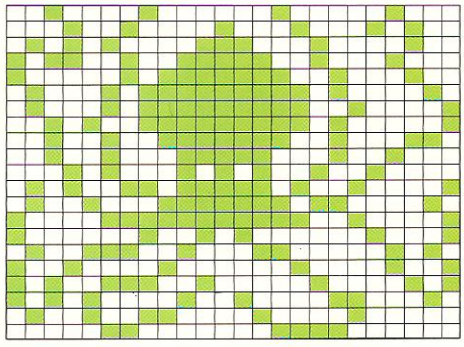
INSECTOID

1,0,128,0,129,0,0
126,0,0,219,0,0,36
0,0,24,0,3,255,192
7,255,224,44,0,52,25
255,152,17,255,136,32,0
4,0,255,0,0,255,0
0,0,0,0,126,0,0
126,0,0,0,0,0,60
0,0,24,0,0,0,0

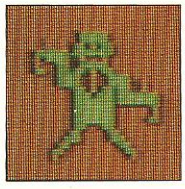


SEA MONSTER

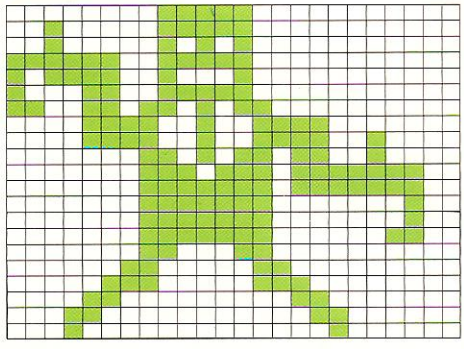
33,18,8,18,57,20,84
124,132,178,254,136,130,254
68,69,255,66,69,255,33
41,255,65,36,254,70,36
124,72,20,84,144,83,125
32,72,254,192,135,255,1
129,40,194,78,70,33,144
129,17,167,32,138,168,72
105,73,132,37,6,3,194



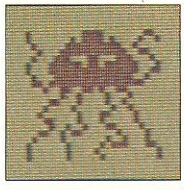
ANDROID



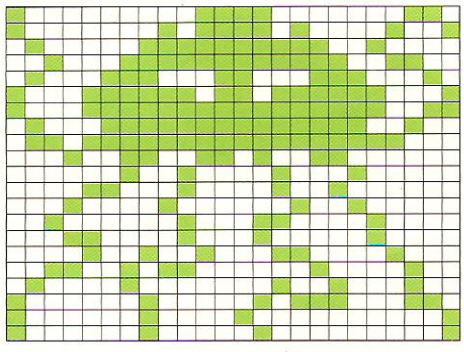
0,248,0,32,248,0,33
172,0,253,252,0,188,136
0,140,248,0,205,172,0
15,39,128,15,39,144,1
173,144,1,221,252,1,253
252,1,252,4,1,252,4
1,252,12,1,140,0,3
6,0,7,7,0,12,1
128,24,0,192,16,0,64



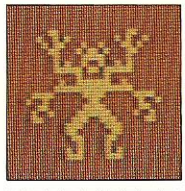
JELLY MONSTER



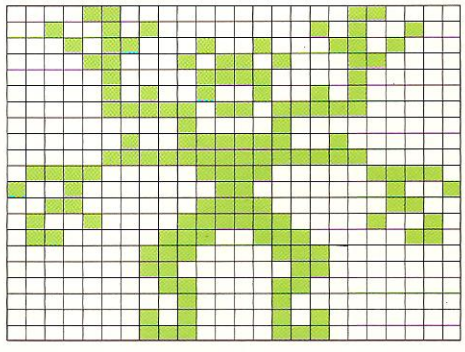
128,60,6,128,189,9,65
255,144,35,255,204,71,24
226,143,219,241,143,255,243
71,255,228,111,255,248,19
52,192,2,66,32,12,66
64,16,129,32,32,70,16
24,68,16,9,35,8,50
192,132,66,2,68,129,5
198,129,8,1,129,4,1



HYDRA MAN

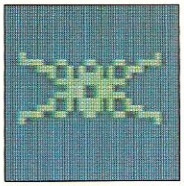


12,0,48,37,0,164,22
36,104,12,90,48,4,60
32,5,36,160,7,219,224
0,66,0,2,126,64,7
255,224,120,60,30,144,24
9,48,60,12,72,126,18
96,231,6,1,195,128,1
129,128,0,195,0,0,66
0,1,66,128,1,195,128

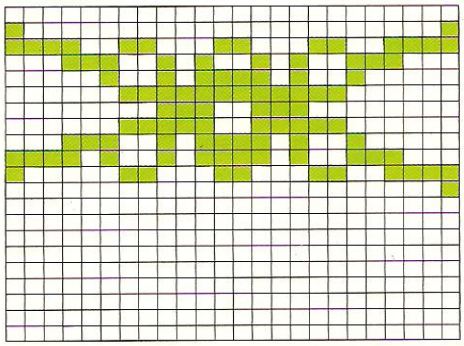


SPACECRAFT

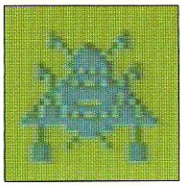
PLANETARY PROBE



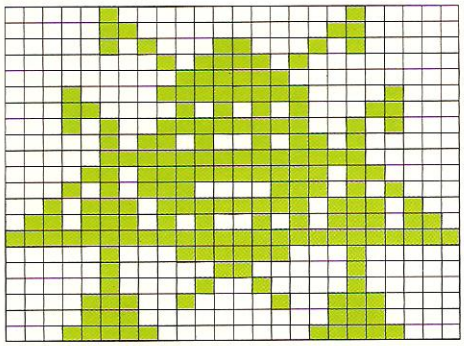
```
0,0,0,128,0,1,243
24,207,28,165,56,4,189
32,3,255,192,0,231,0
3,231,192,4,189,32,28
189,56,243,24,207,128,0
1,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
```



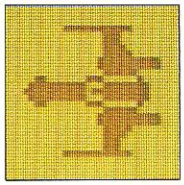
LANDER



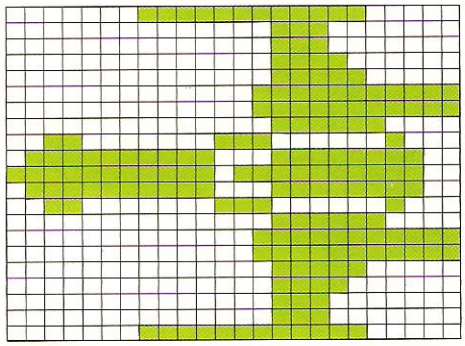
```
4,0,32,6,0,96,5
24,160,0,189,0,0,126
0,16,255,8,24,85,24
20,255,40,5,165,160,7
255,224,13,255,176,21,195
168,46,255,116,110,165,118
255,255,255,16,255,8,16
60,8,16,66,8,56,129
28,56,0,28,56,0,28
```



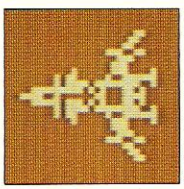
SPACE FIGHTER



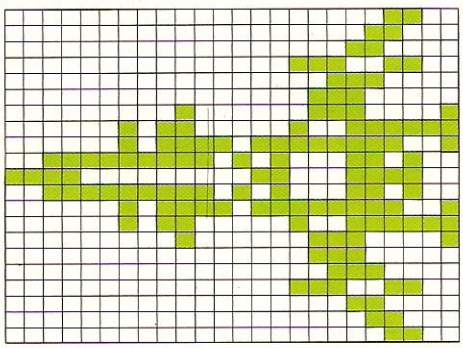
```
1,255,224,0,3,128,0
3,128,0,3,192,0,3
224,0,7,255,0,7,255
0,3,240,48,28,8,127
227,252,255,239,252,127,227
252,48,28,8,0,3,240
0,7,255,0,7,255,0
3,224,0,3,192,0,3
128,0,3,128,1,255,224
```



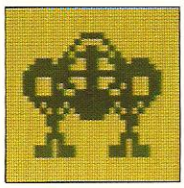
SPACE FIGHTER



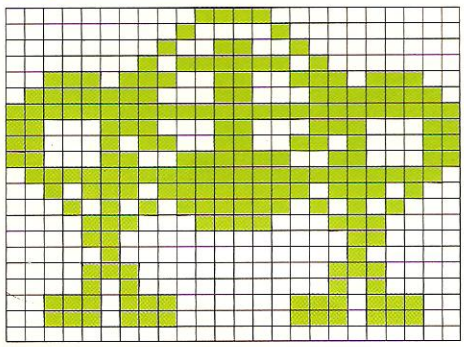
```
0,0,12,0,0,24,0
0,48,0,1,204,0,0
112,0,0,224,0,64,225
2,193,63,2,247,249,63
234,52,242,6,60,62,234
52,2,247,249,2,193,63
0,64,225,0,0,224,0
0,112,0,1,204,0,0
48,0,0,24,0,0,12
```



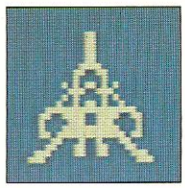
EXCURSION VEHICLE



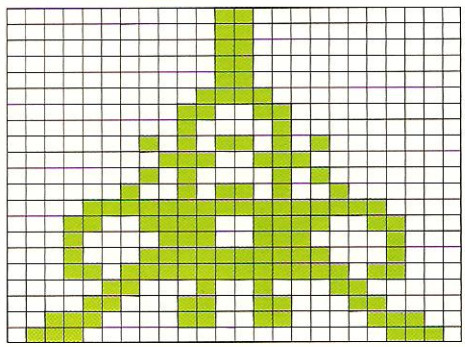
```
0,60,0,0,66,0,0
153,0,1,126,128,59,153
220,127,24,254,255,255,255
199,24,227,197,153,163,198
126,99,127,255,254,46,255
116,21,126,168,14,60,112
12,0,48,4,0,32,14
0,112,10,0,80,59,129
220,59,129,220,0,0,0
```



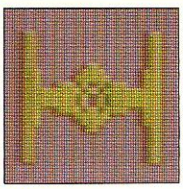
EXCURSION VEHICLE



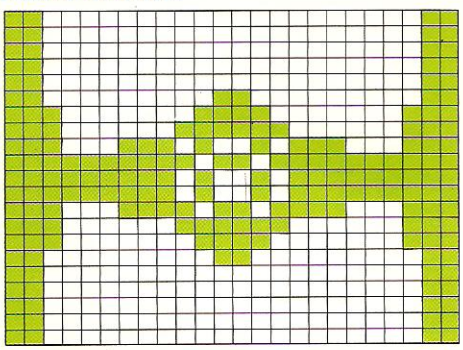
```
0,24,0,0,24,0,0
24,0,0,24,0,0,24
0,0,60,0,0,102,0
0,66,0,1,90,128,0
231,0,1,66,128,2,90
64,15,255,240,25,231,152
16,231,8,17,255,136,27
255,216,6,36,96,12,102
48,56,102,28,112,0,14
```



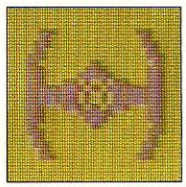
INTERGALACTIC CRUISER



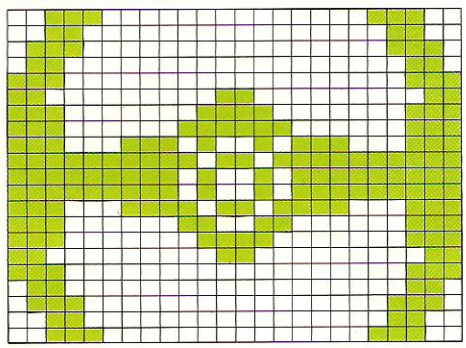
```
192,0,3,192,0,3,192
0,3,192,0,3,192,0
3,192,24,3,224,60,7
224,126,7,227,165,199,255
219,255,255,165,255,255,165
255,227,219,199,224,126,7
224,60,7,192,24,3,192
0,3,192,0,3,192,0
3,192,0,3,192,0,3
```



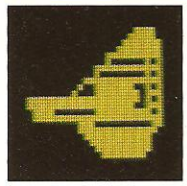
INTERGALACTIC CRUISER



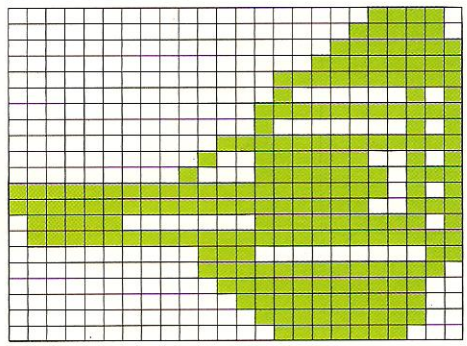
```
56,0,28,48,0,12,112
0,14,96,0,6,224,0
7,192,24,3,224,60,7
224,126,7,227,165,199,255
219,255,255,165,255,255,165
255,227,219,199,224,126,7
224,60,7,192,24,3,224
0,7,96,0,6,112,0
14,48,0,12,56,0,28
```



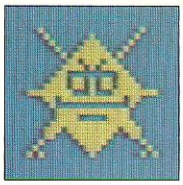
COMMAND SHIP



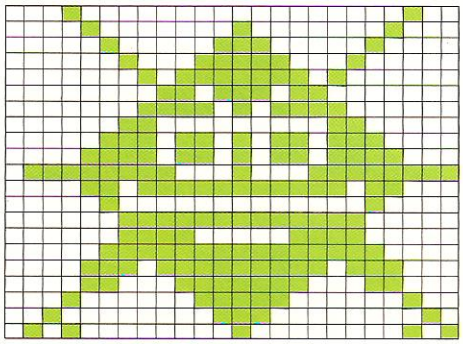
```
0,0,30,0,0,62,0
0,127,0,0,255,0,3
255,0,2,5,0,15,255
0,8,5,0,63,255,0
79,229,0,143,247,255,255
245,255,255,231,120,15,253
127,255,255,0,112,3,0
127,254,0,63,252,0,31
252,0,15,252,0,7,248
```



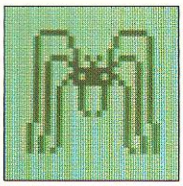
TRIBAL SPECTRE



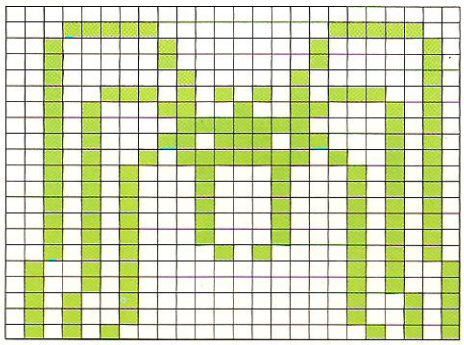
16,0,4,8,8,8,4
28,16,2,62,32,1,127
64,0,221,128,1,235,192
3,0,96,7,107,112,15
107,120,127,136,255,13,255
216,4,0,16,3,255,224
3,193,224,7,255,240,14
255,184,12,127,24,16,62
4,32,28,2,80,8,5



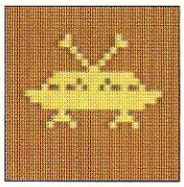
SPOOKY SPIDER



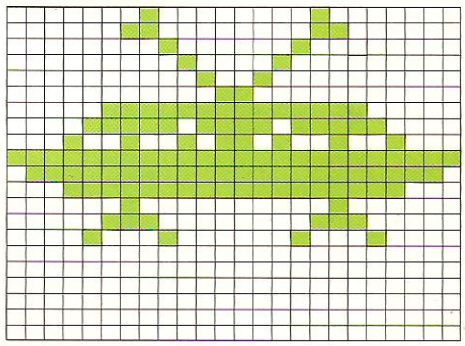
0,0,0,31,0,124,32
128,130,32,128,130,32,65
2,39,85,114,40,201,138
40,127,10,40,221,130,41
127,74,42,34,42,42,34
42,42,34,42,42,34,42
42,20,42,42,0,42,74
0,41,74,0,41,82,0
37,82,0,37,84,0,21



SPYING SAUCER

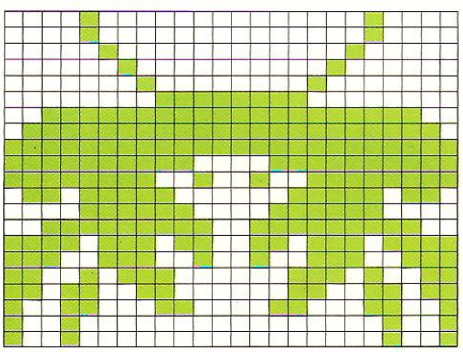


1,0,128,3,0,192,0
129,0,0,66,0,0,36
0,0,24,0,7,255,224
15,189,240,18,90,72,255
255,255,127,255,254,31,255
248,2,0,64,7,0,224
8,129,16,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0



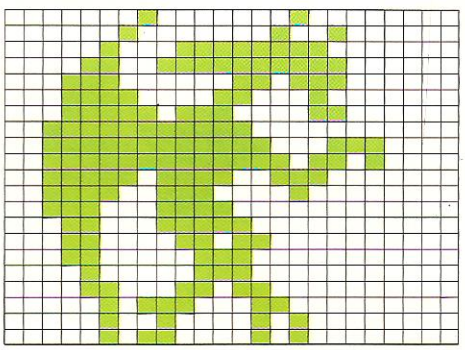
SPACE CRAB

8,0,16,8,0,16,4
0,32,2,0,64,1,0
128,0,255,0,63,255,252
127,255,254,255,255,255,255
195,255,255,36,255,111,129
246,15,195,240,63,231,252
247,102,239,231,36,231,203
129,211,217,129,155,152,195
25,144,0,9,144,0,9

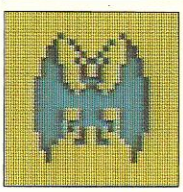


VAMPIRE

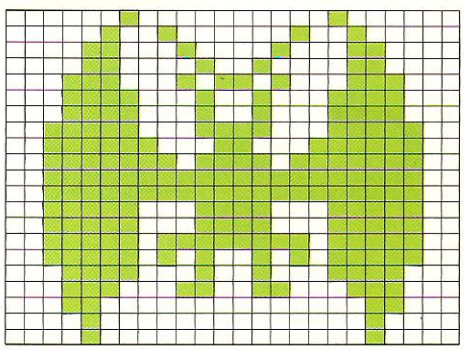
1,1,0,2,2,64,4
127,128,12,255,192,28,13
192,30,24,128,31,112,192
63,248,0,63,252,112,63
254,208,62,243,128,60,225
0,56,240,0,24,248,0
24,108,0,24,56,0,8
56,0,12,108,0,5,198
0,4,133,0,5,133,0



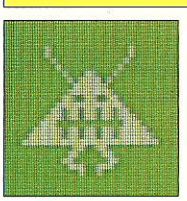
VAMPIRE



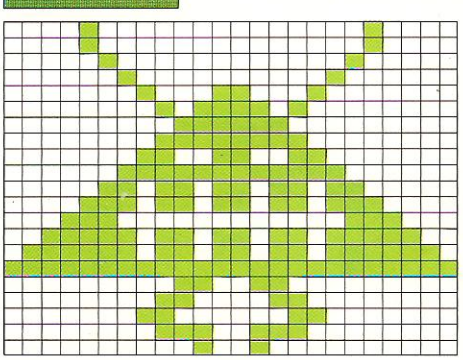
2,0,64,4,129,32,12
66,48,12,36,48,28,90
56,30,36,120,30,36,120
62,60,124,63,24,252,63
189,252,63,255,252,63,255
252,62,60,124,62,60,124
62,255,124,62,165,124,30
36,120,28,102,56,24,0
24,8,0,16,8,0,16



GHOUL

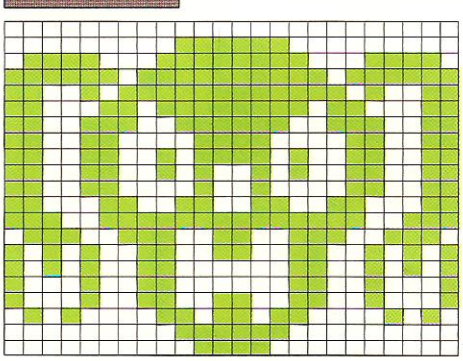


8,0,16,8,0,16,4
0,32,2,0,64,1,24
128,0,189,0,0,126,0
0,255,0,1,153,128,3
255,192,6,219,96,14,219
112,28,0,56,62,219,124
126,219,126,255,255,255,0
102,0,0,195,0,1,129
128,0,231,0,0,36,0

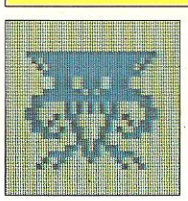


GHOUL

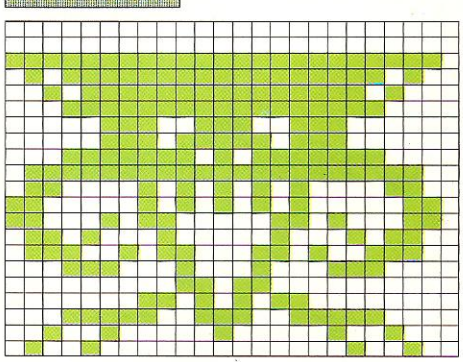
0,0,0,0,126,0,120
255,30,253,255,191,203,255
211,198,255,99,206,126,115
204,60,51,204,189,51,204
165,51,204,36,51,198,102
99,231,255,231,115,153,206
169,153,149,169,129,149,137
129,145,137,153,145,80,219
10,0,126,0,0,60,0



GHOUL

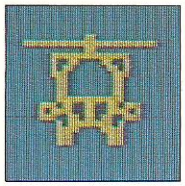


0,0,0,0,0,0,255
255,254,95,255,244,47,255
232,31,255,240,7,57,192
7,57,192,31,215,240,127
255,252,97,85,12,193,85
6,197,131,70,104,130,44
114,198,156,28,68,112,0
108,0,1,171,0,15,57
224,18,16,144,36,0,72

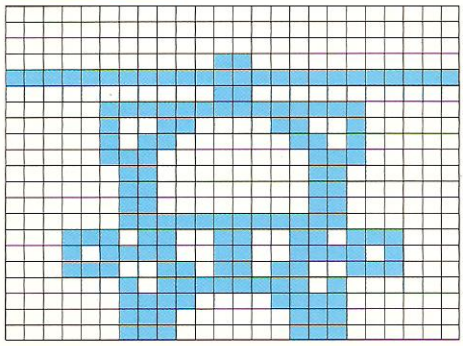


AIRCRAFT

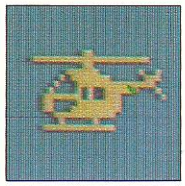
HELICOPTER



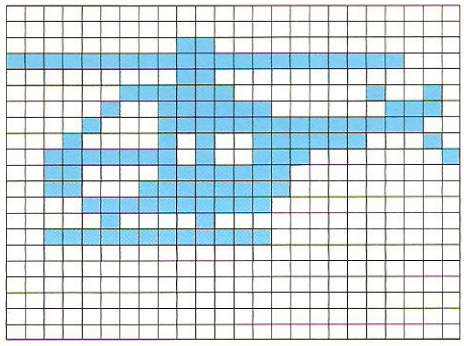
0,0,0,0,0,0,0
0,0,0,24,0,255,255
255,0,24,0,7,255,224
4,195,32,5,129,160,7
0,224,3,0,192,3,0
192,3,0,192,3,255,192
29,153,184,23,153,232,28
219,56,2,255,64,3,195
192,3,129,192,3,129,192



HELICOPTER

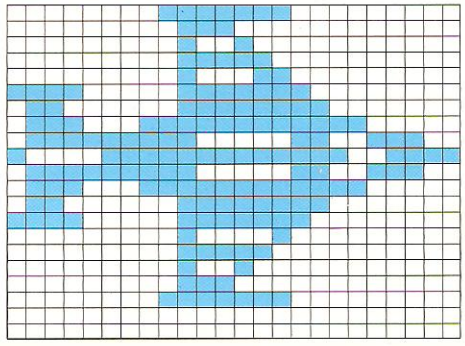


0,0,0,0,0,0,0
96,0,255,255,248,0,96
0,3,240,18,5,252,14
8,255,252,16,167,226,63
167,1,48,255,0,49,255
0,15,254,0,4,16,0
63,254,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0



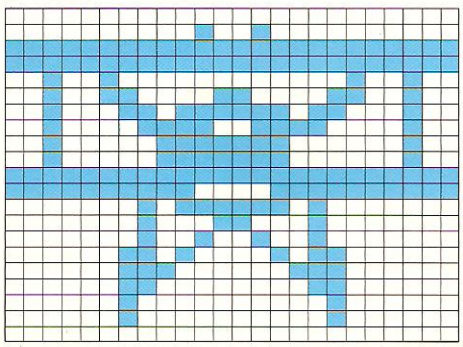
HIGH-ALTITUDE JET

0,254,0,0,112,0,0
72,0,0,124,0,0,66
0,240,127,0,96,127,128
113,255,224,127,193,156,135
255,207,127,193,156,113,255
224,96,127,128,240,127,0
0,66,0,0,124,0,0
72,0,0,112,0,0,254
0,0,0,0,0,0,0



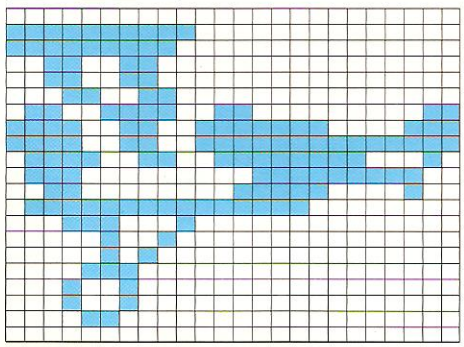
BIPLANE

0,36,0,255,255,255,255
255,255,36,0,36,38,24
100,35,126,196,33,231,132
33,255,4,33,255,4,255
255,255,255,195,255,1,126
128,1,24,128,1,36,128
3,66,192,3,129,192,3
0,192,2,0,64,2,0
64,0,0,0,0,0,0

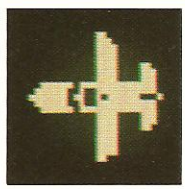


BIPLANE

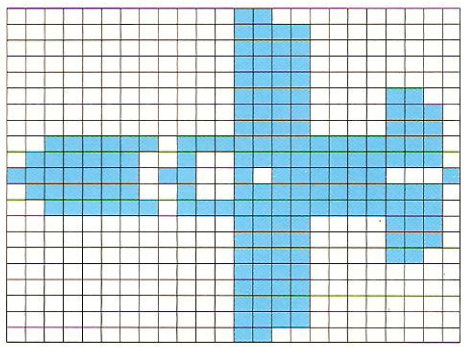
0,0,0,255,192,0,255
128,0,51,0,0,51,0
0,25,128,0,103,152,3
243,63,135,243,63,255,121
159,226,112,7,252,48,15
132,127,255,0,28,64,0
4,128,0,5,0,0,14
0,0,18,0,0,18,0
0,12,0,0,0,0,0



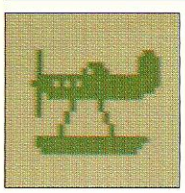
RECONNAISSANCE PLANE



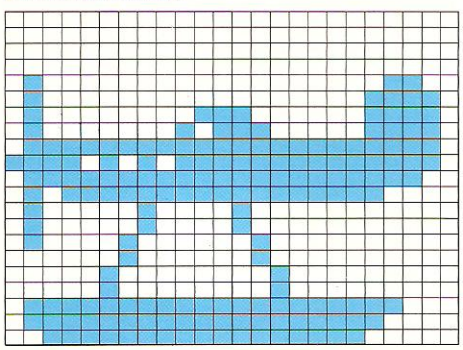
0,12,0,0,14,0,0
15,0,0,15,0,0,15
0,0,15,12,0,15,14
0,15,14,63,127,254,126
79,254,254,203,241,126,79
254,63,127,254,0,15,14
0,15,14,0,15,12,0
15,0,0,15,0,0,15
0,0,15,0,0,12,0



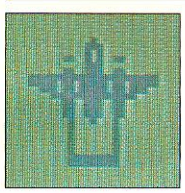
SEAPLANE



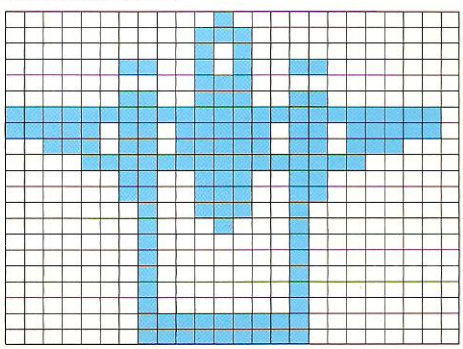
0,0,0,0,0,0,0
0,0,0,0,0,64,0
12,64,0,30,64,56,30
64,92,30,127,255,254,245
127,254,127,255,252,95,255
240,65,8,0,65,8,0
66,4,0,2,4,0,4
2,0,4,2,0,127,255
248,127,255,240,63,255,224



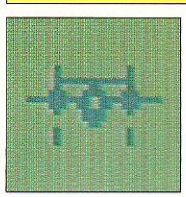
SEAPLANE



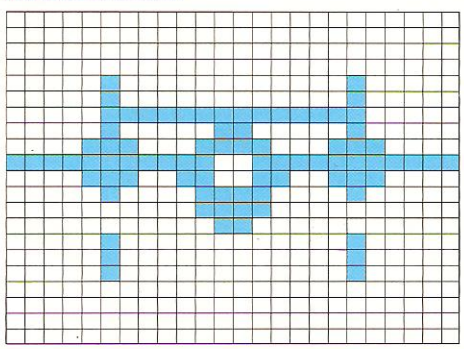
0,16,0,0,56,0,0
40,0,3,41,128,0,56
0,3,57,128,255,255,254
251,125,190,59,125,184,15
255,224,3,57,128,3,57
128,3,57,128,1,17,0
1,1,0,1,1,0,1
1,0,1,1,0,1,1
0,1,255,0,1,255,0



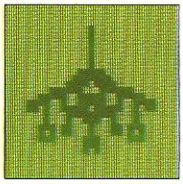
SEAPLANE



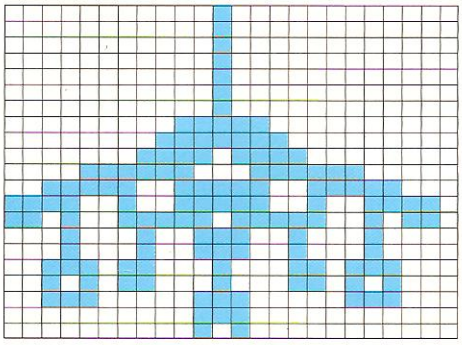
0,0,0,0,0,0,0
0,0,0,0,0,4,0
32,4,0,32,7,255,224
4,24,32,14,60,112,255
231,255,14,102,112,4,60
32,0,60,0,0,24,0
4,0,32,4,0,32,4
0,32,0,0,0,0,0
0,0,0,0,0,0,0



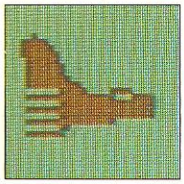
HIGH-ALTITUDE JET



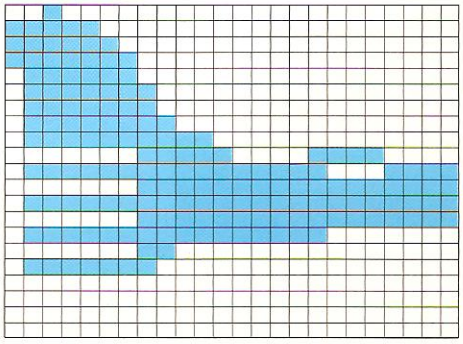
```
0,8,0,0,8,0,0
8,0,0,8,0,0,8
0,0,8,0,0,8,0
0,62,0,0,127,0,0
247,128,7,227,240,31,62
124,121,62,79,105,247,203
8,255,136,8,156,136,29
136,220,21,136,212,28,28
28,0,28,0,0,20,0
```



STUNT PLANE

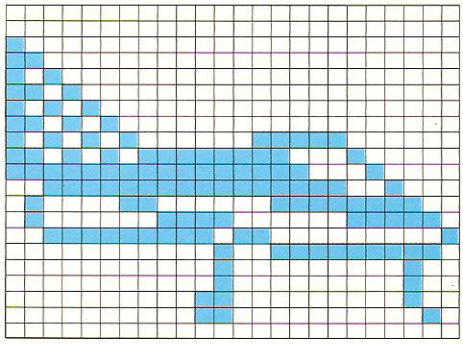


```
32,0,0,248,0,0,252
0,0,254,0,0,126,0
0,127,0,0,127,0,0
127,128,0,127,224,0,1
240,240,127,255,143,1,255
255,127,255,255,1,255,255
127,255,128,1,128,0,127
0,0,0,0,0,0,0
0,0,0,0,0,0,0
```

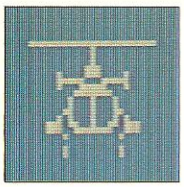


JET TRAINER

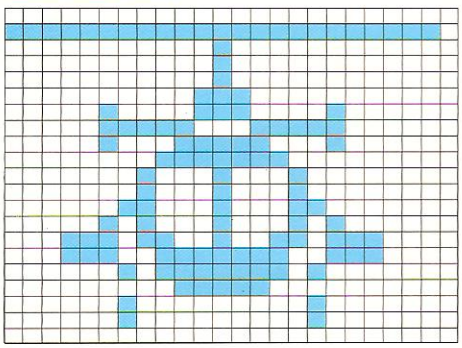
```
0,0,0,0,0,0,128
0,0,192,0,0,160,0
0,208,0,0,168,0,0
212,0,0,170,7,192,213
254,32,255,255,16,63,255
200,67,3,252,64,240,62
63,236,1,0,19,254,0
16,4,0,16,4,0,48
4,0,48,2,0,0,0
```



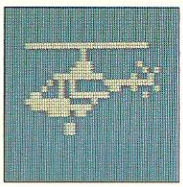
HELICOPTER



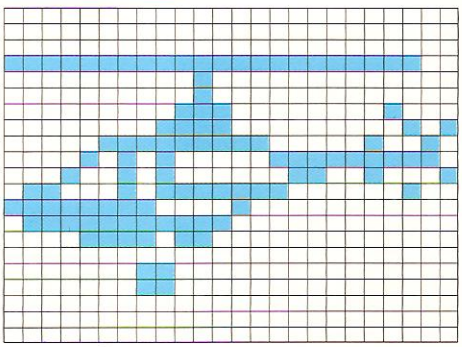
```
0,0,0,255,255,254,0
16,0,0,16,0,0,16
0,0,56,0,4,56,64
7,199,192,4,124,64,0
254,0,1,17,0,1,17
0,3,17,128,5,17,64
29,147,112,28,254,112,2
0,128,0,0,0,2,0
128,2,0,128,0,0,0
```



HELICOPTER

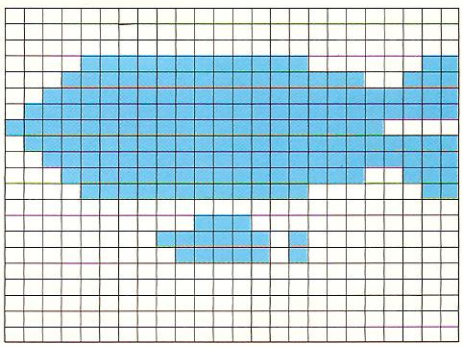


```
0,0,0,0,0,0,0
0,0,255,255,252,0,32
0,0,32,0,0,112,8
0,240,5,7,252,18,10
131,254,18,129,145,98,254
4,255,8,0,127,240,0
15,96,0,0,0,0,1
128,0,1,128,0,0,0
0,0,0,0,0,0,0
```

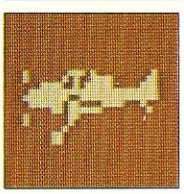


AIRSHIP

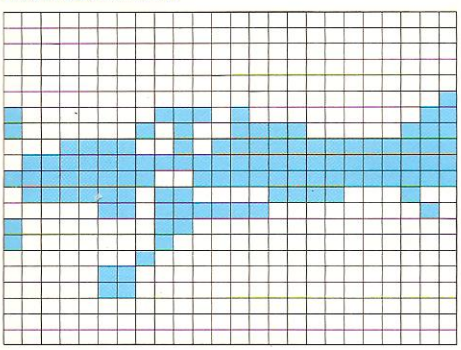
```
0,0,0,0,0,0,0
0,0,0,0,0,15,255
3,31,255,231,63,255,255
127,255,255,255,255,240,127
255,255,63,255,255,31,255
231,15,255,3,0,0,0
0,56,0,0,253,0,0
125,0,0,0,0,0,0
0,0,0,0,0,0,0
```



MONOPLANE

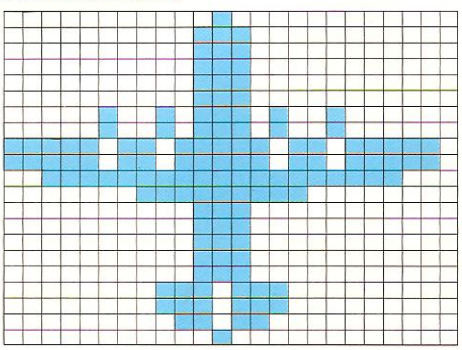


```
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,128,232,1
129,79,3,30,95,7,127
255,255,255,63,255,126,195
196,6,252,2,128,192,0
128,128,0,1,0,0,6
0,0,6,0,0,0,0
0,0,0,0,0,0,0
```

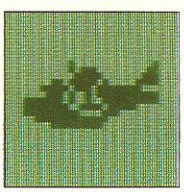


TRANSPORTER

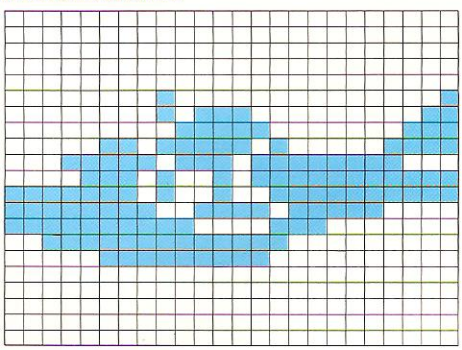
```
0,16,0,0,56,0,0
56,0,0,56,0,0,56
0,0,56,0,4,186,64
4,186,64,251,125,190,251
125,190,63,255,248,1,255
0,0,56,0,0,56,0
0,56,0,0,56,0,0
56,0,0,238,0,0,238
0,0,108,0,0,16,0
```



TRANSPORTER

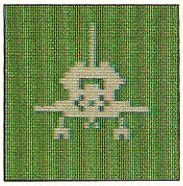


```
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,128,1,0,184,3
0,124,7,6,250,15,31
119,248,14,151,255,254,147
255,255,61,240,255,195,128
63,254,0,7,252,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
```

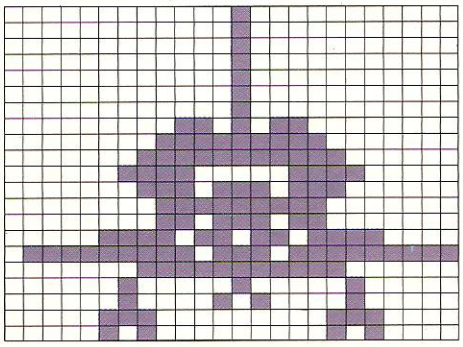


SPACECRAFT

SHUTTLE

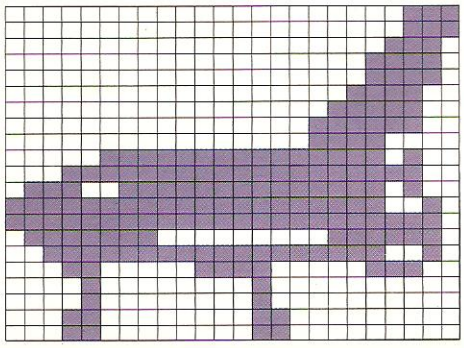


```
0,8,0,0,8,0,0
8,0,0,8,0,0,8
0,0,8,0,0,8,0
0,107,0,0,255,128,1
255,192,3,193,224,1,221
192,0,255,128,0,221,128
7,182,240,127,221,255,1
255,192,2,8,32,2,20
32,7,0,112,5,0,80
```



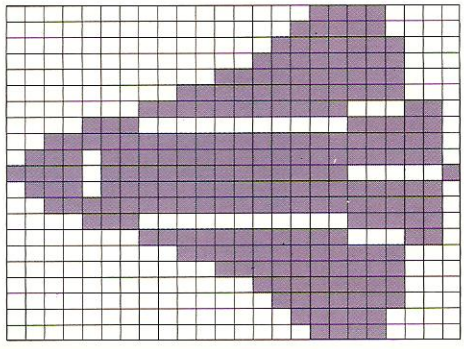
SHUTTLE

```
0,0,7,0,0,7,0
0,14,0,0,30,0,0
30,0,0,60,0,0,124
0,0,252,0,0,240,7
255,244,31,255,252,115,255
244,255,255,242,255,255,254
127,0,126,63,255,242,24
60,28,8,4,0,8,4
0,24,6,0,24,6,0
```

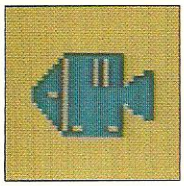


SHUTTLE

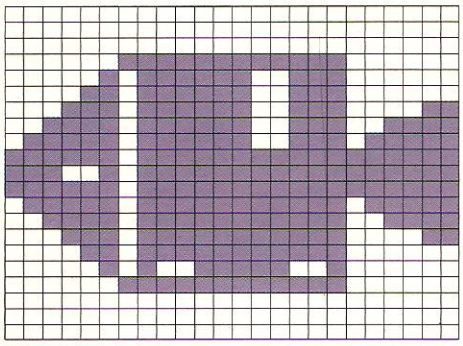
```
0,0,240,0,1,240,0
3,248,0,7,248,0,31
248,0,127,248,1,255,198
14,0,62,63,255,254,119
255,254,247,255,193,119,255
254,63,255,254,14,0,62
1,255,198,0,127,248,0
31,248,0,7,248,0,3
248,0,1,240,0,0,240
```



LUNAR MODULE

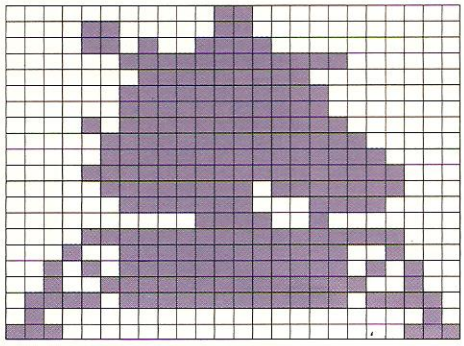


```
0,0,0,0,0,0,0
0,0,3,255,192,5,245
192,13,245,192,29,245,195
61,245,207,125,245,223,253
255,255,229,255,255,253,255
255,125,255,223,61,255,207
29,255,195,13,255,192,5
62,64,3,255,192,0,0
0,0,0,0,0,0,0
```



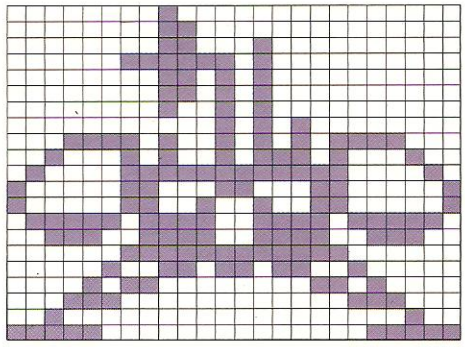
LUNAR LANDER

```
0,24,0,12,24,0,13
60,0,3,255,192,1,255
0,3,255,128,3,255,128
11,255,192,7,255,224,7
255,240,15,255,248,7,251
248,3,248,240,0,60,128
15,255,248,19,255,200,43
255,212,39,255,228,123,255
222,64,0,2,224,0,7
```

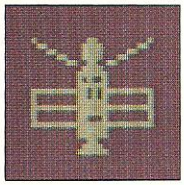


VIKING

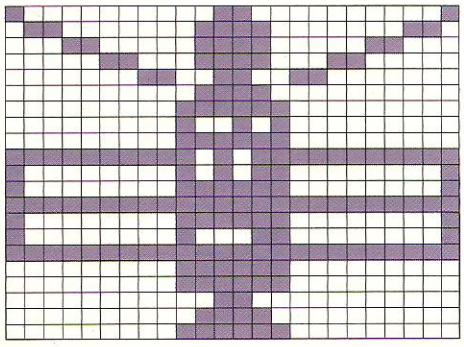
```
0,128,0,0,192,0,0
196,0,3,244,0,0,212
0,0,212,0,0,148,0
0,21,0,30,149,120,34
151,68,67,255,194,131,24
193,131,36,193,125,231,190
57,255,156,3,255,192,4
219,32,11,0,208,28,0
56,32,0,4,248,0,31
```



SKYLAB

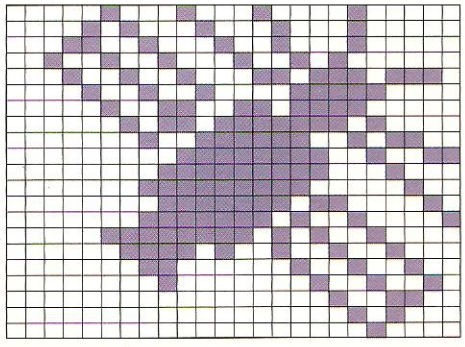


```
128,24,1,96,60,6,24
60,24,6,60,96,1,153
128,0,60,0,0,126,0
0,74,0,0,126,0,255
215,255,128,86,1,128,126
1,255,255,255,128,126,1
128,70,1,255,255,255,0
126,0,0,126,0,0,24
0,0,60,0,0,126,0
```



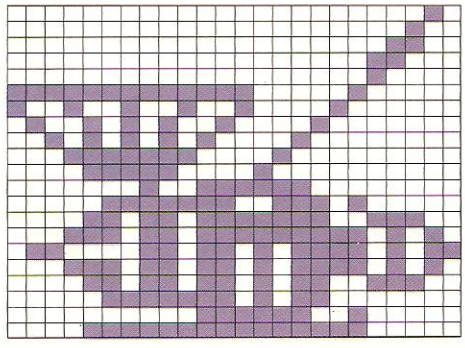
SKYLAB

```
4,68,32,10,34,32,25
18,32,36,137,64,18,70
238,9,33,240,4,205,224
2,223,208,1,63,44,0
127,147,0,255,136,1,255
4,0,254,194,1,253,33
7,250,144,3,210,72,1
129,36,0,128,146,0,0
76,0,0,40,0,0,16
```

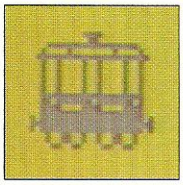


VENERA

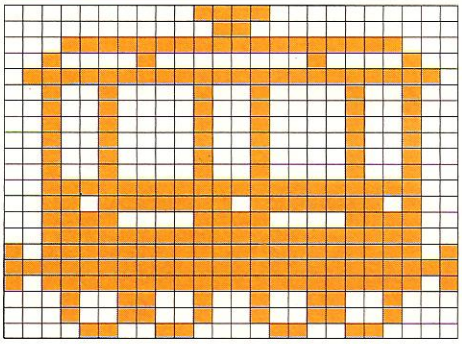
```
0,0,2,0,0,4,0
0,12,0,0,24,0,0
16,255,248,32,146,72,64
74,144,128,42,161,0,31
194,0,15,132,0,1,63
0,7,243,160,13,191,156
29,191,234,97,181,235,29
181,234,13,181,156,7,245
128,0,63,32,15,255,224
```



CARRIAGE

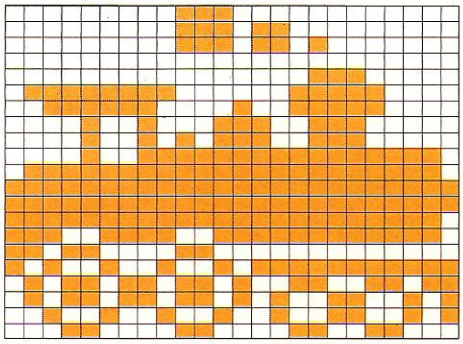


0,60,0,0,24,0,31
255,248,33,0,132,127,255
254,36,36,36,36,36,36
36,36,36,36,36,36,36
36,36,36,36,36,63,255
252,55,247,244,60,60,60
63,255,252,191,255,253,255
255,255,191,255,253,19,36
200,19,36,200,12,195,48

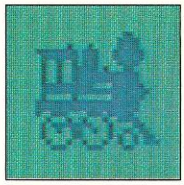


4-4-0 LOCO

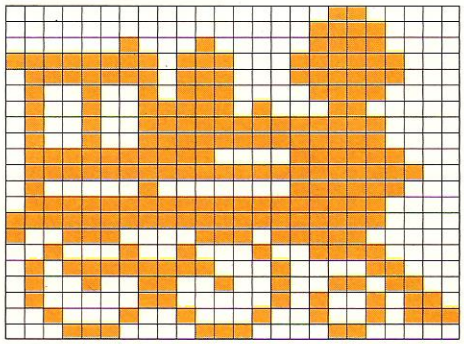
0,112,0,0,118,0,0
118,128,0,0,0,0,0
224,127,1,240,63,9,240
9,28,224,9,92,224,9
255,254,127,255,254,255,255
255,255,255,255,255,255,254
231,159,254,216,96,0,164
151,255,91,105,155,91,106
101,36,146,101,24,97,152



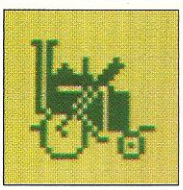
U.S. LOCO



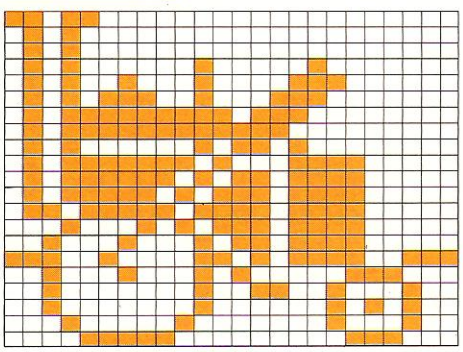
0,0,96,0,0,240,2
32,240,255,113,248,127,113
248,73,112,240,73,244,96
73,255,240,73,255,240,127
225,248,127,255,252,65,1
248,255,255,240,255,255,240
156,59,224,34,68,240,95
226,24,73,146,108,65,130
150,34,68,151,28,56,96



ROCKET

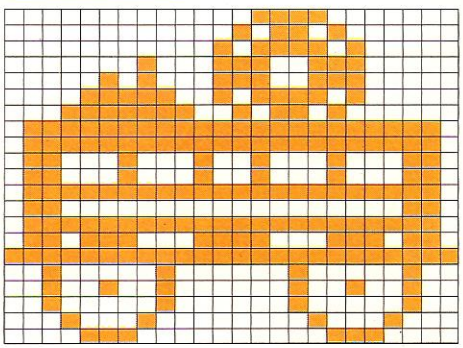


216,0,0,80,0,0,80
0,0,80,32,128,82,33
192,87,35,128,95,247,0
95,254,0,88,41,0,95
213,224,95,173,224,95,93
224,111,189,224,17,93,224
34,45,224,228,53,231,34
40,56,32,38,68,32,32
84,16,64,68,15,128,56

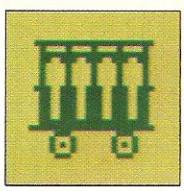


TENDER

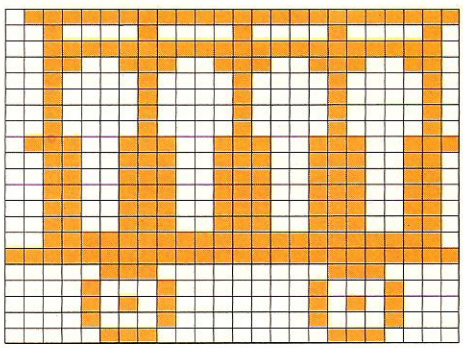
0,7,128,0,11,64,0
23,160,1,28,224,5,28
224,15,151,160,31,203,64
127,255,254,127,255,254,98
36,70,98,36,70,127,255
254,96,0,6,127,255,254
110,60,118,255,255,255,32
129,4,36,129,36,32,129
4,17,0,136,14,0,112



CARRIAGE

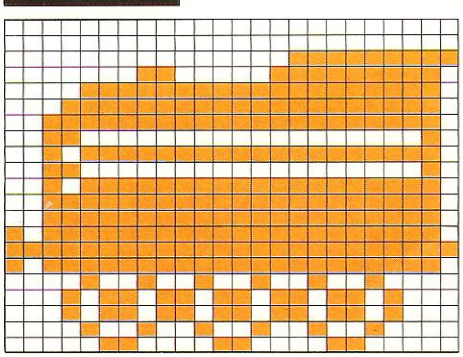


127,255,255,33,8,66,63
255,254,51,156,230,33,8
66,33,8,66,33,8,66
33,8,66,115,156,231,51
156,230,51,156,230,51,156
230,51,156,230,51,156,230
63,255,254,255,255,255,7
0,112,8,128,136,10,128
168,8,128,136,7,0,112



TENDER

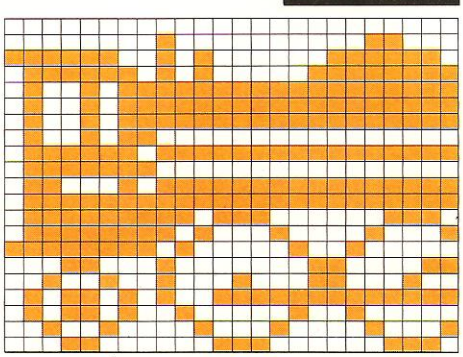
0,0,0,0,0,0,0
1,255,1,131,254,15,255
254,31,255,254,63,255,254
48,0,2,47,255,254,48
0,2,47,255,254,63,255
254,63,255,254,191,255,254
255,255,255,191,255,254,9
36,144,22,219,104,22,219
104,9,36,144,6,24,96

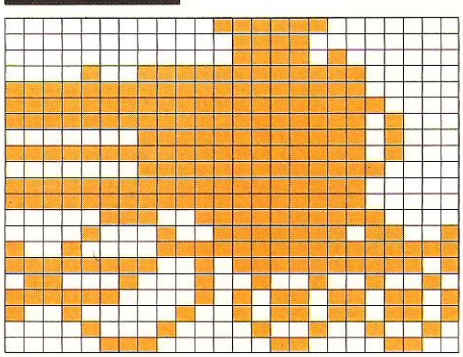


PACIFIC-TYPE LOCO

0,0,0,0,128,24,252
160,126,126,160,255,75,255
255,75,255,255,75,255,255
75,0,0,126,255,255,127
0,0,98,255,255,127,255
255,127,220,14,127,162,17
255,65,32,24,128,195,36
136,196,90,159,255,90,65
32,36,34,17,24,28,14

0,31,128,0,15,0,0
15,64,9,255,224,255,255
224,255,255,240,255,255,232
1,255,232,255,255,232,1
255,240,255,255,224,255,255
224,7,63,240,8,159,217
144,255,255,255,47,237,98
36,146,252,43,109,144,75
109,8,132,146,7,3,12

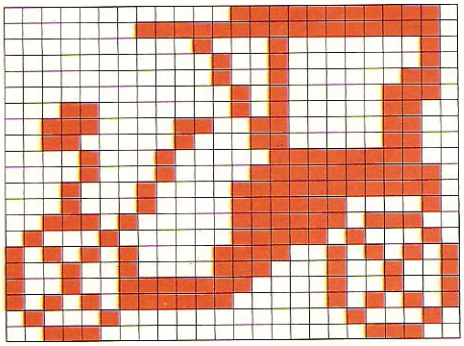




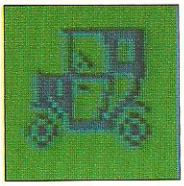
CARS, TRUCKS AND MOTORBIKES

VETERAN

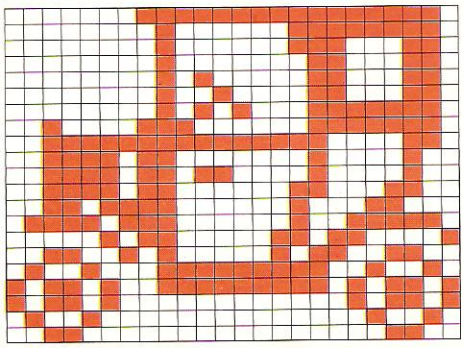
0,15,254,1,255,254,0
34,2,0,18,2,0,18
6,0,10,14,24,10,12
56,102,12,48,70,12,8
131,252,8,135,254,17,15
254,17,15,194,58,15,220
68,15,162,170,30,85,146
28,73,147,248,73,171,240
85,68,0,34,56,0,28



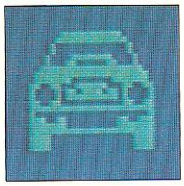
VETERAN



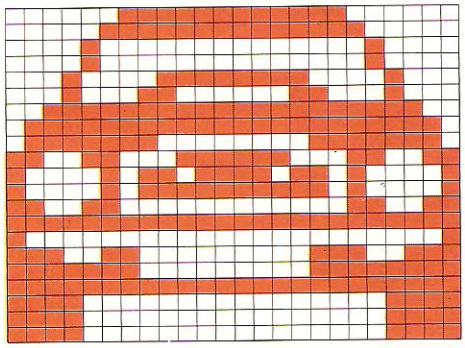
1,255,254,0,129,254,0
129,198,0,128,198,0,160
198,0,144,198,0,168,254
32,192,254,63,255,134,61
128,132,61,176,132,61,129
140,55,129,24,123,129,62
5,131,97,50,130,204,72
255,146,180,255,173,180,0
45,72,0,18,48,0,12



SALOON

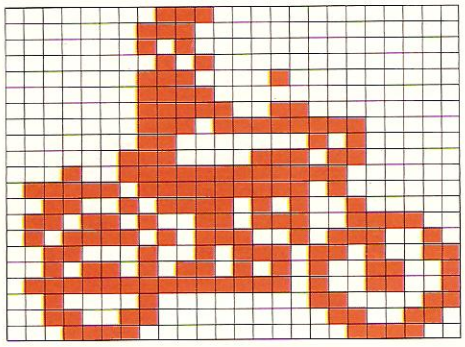


3,255,192,7,255,224,8
0,16,8,0,16,16,255
8,17,129,136,63,255,252
127,255,254,127,0,254,204
126,51,133,189,161,133,255
161,204,0,51,255,255,255
128,0,1,230,0,103,254
0,127,255,255,255,240,0
15,240,0,15,240,0,15



MOTORBIKE

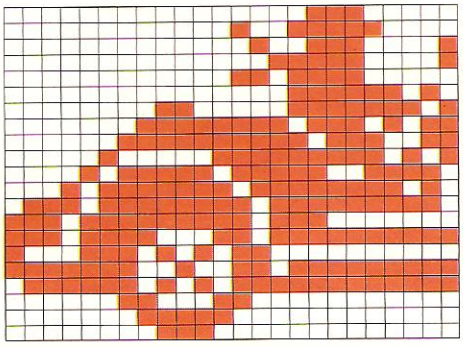
0,224,0,1,192,0,1
64,0,0,160,0,1,194
0,1,224,0,1,243,0
1,223,224,1,128,160,3
135,96,19,255,192,125,255
64,50,245,32,109,55,60
94,185,114,49,46,211,33
106,153,127,254,153,33,0
195,51,0,102,30,0,60

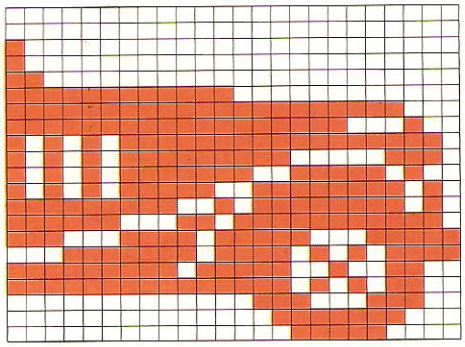


CLASSIC TOURER

0,0,224,0,9,240,0
4,225,0,3,225,0,13
224,0,1,250,0,193,253
1,254,235,3,255,115,4
31,245,11,239,158,23,247
250,119,251,254,239,251,128
239,27,255,142,172,0,254
77,255,126,175,255,3,24
0,1,240,0,0,224,0

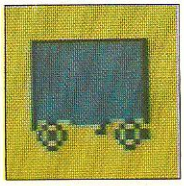
0,0,0,0,0,0,128
0,0,128,0,0,192,0
0,255,240,0,255,255,248
255,255,196,171,255,254,171
254,14,171,249,244,171,231
250,255,151,250,252,111,254
227,223,31,31,222,175,255
190,76,255,254,172,0,7
28,0,3,248,0,1,240

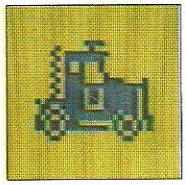




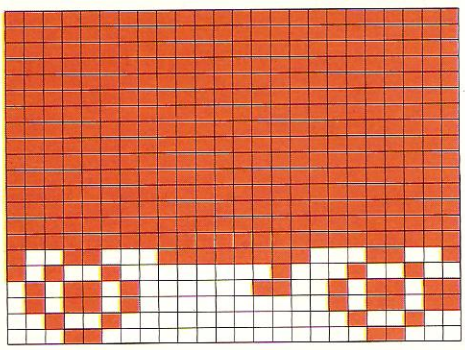
TRUCK

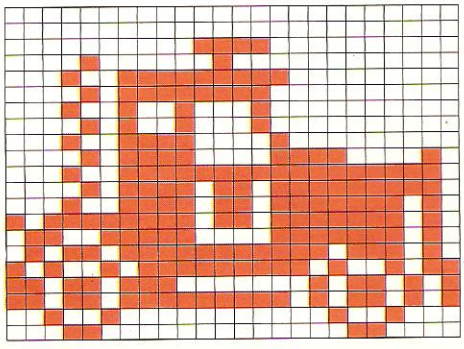
255,255,255,255,255,255,255
255,255,255,255,255,255,255
255,255,255,255,255,255,255
255,255,255,255,255,255,255
255,255,255,255,255,255,255
255,255,255,255,255,255,255
255,255,255,152,255,25,164
2,36,90,6,91,90,0
91,36,0,36,24,0,24



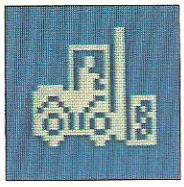


0,0,0,0,0,0,0
60,0,8,16,0,19,254
0,11,252,0,18,68,0
10,68,0,19,194,0,11
195,194,19,255,254,11,219
250,19,219,250,191,219,250
103,195,250,91,255,186,165
255,75,219,128,181,219,254
181,36,0,72,24,0,48

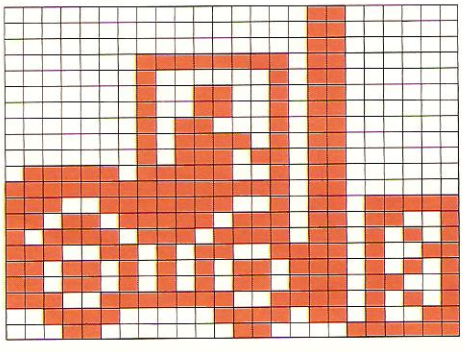




FORKLIFT

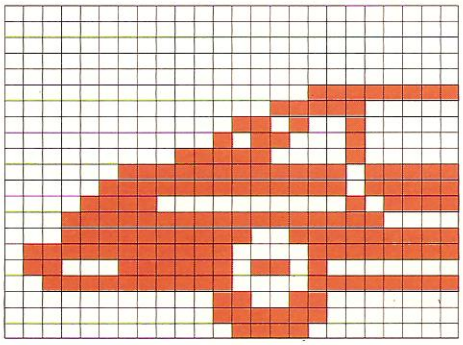


0,0,192,0,0,192,0
0,192,1,254,192,1,2
192,1,50,192,1,50,192
1,98,192,1,122,192,1
118,192,125,114,192,255,254
192,231,242,223,219,238,217
189,222,213,230,179,211,218
173,213,218,173,217,231,243
213,60,30,211,24,12,63

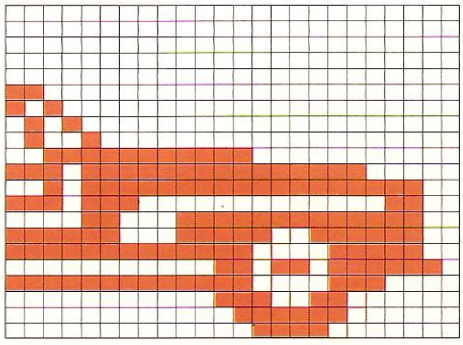


SPORTS SALOON

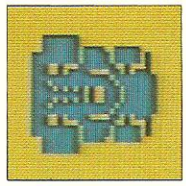
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,255,0,3,224
0,13,32,0,22,32,0
124,32,3,255,223,7,255
223,15,0,47,31,255,240
31,249,255,127,240,255,99
246,128,63,240,255,0,25
128,0,31,128,0,15,0



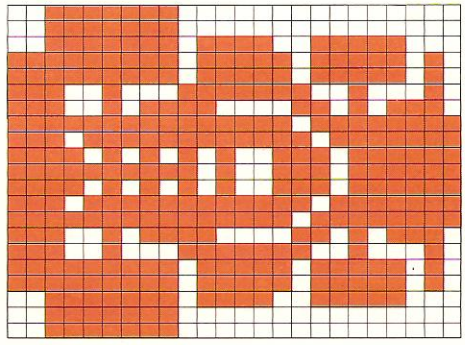
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,192,0,0,160,0,0
80,0,0,72,0,0,63
248,0,239,255,224,47,255
252,238,0,12,12,127,252
252,124,252,255,248,124,0
27,126,255,248,96,0,28
192,0,15,192,0,7,128



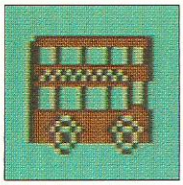
FORMULA 1



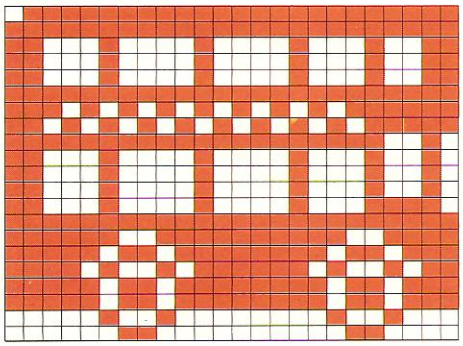
63,128,0,63,128,0,63
188,248,255,190,250,255,190
250,228,127,34,229,225,174
255,254,255,239,255,127,245
83,191,255,211,191,245,83
191,239,255,127,255,254,255
229,225,174,228,127,34,255
190,250,255,190,250,63,188
248,63,128,0,63,128,0



LONDON BUS

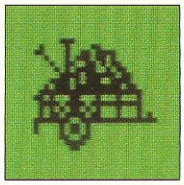


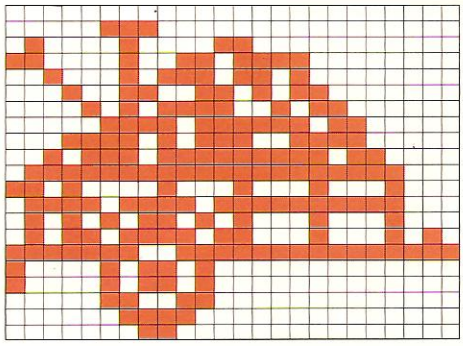
127,255,255,255,255,255,196
33,19,196,33,19,196,33
19,255,255,255,234,170,191
213,85,95,255,255,242,196
33,18,196,33,18,196,33
18,196,33,30,255,255,255
252,255,207,251,127,183,244
191,75,251,127,183,251,127
183,4,128,72,3,0,48



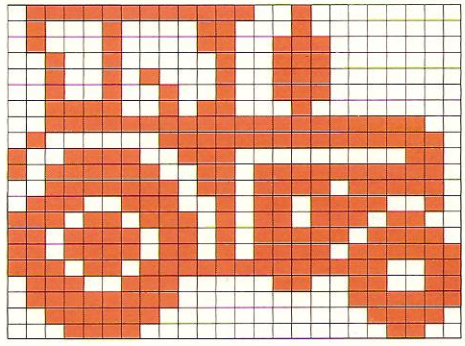
TRACTOR

0,0,0,7,0,0,66
24,0,194,111,0,34,237
0,19,53,128,10,242,192
7,191,96,30,185,224,46
255,176,127,255,248,209,136
136,123,223,248,85,168,136
91,216,138,254,127,255,133
160,0,133,160,0,6,96
0,3,192,0,1,128,0

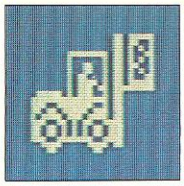




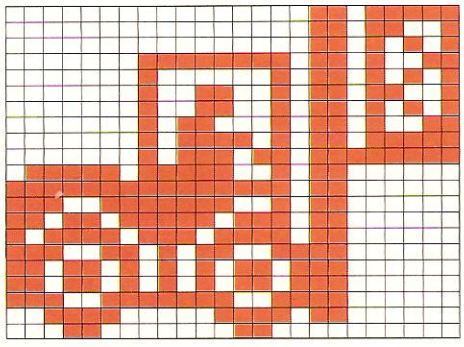
127,249,0,68,49,0,68
51,128,36,19,128,37,19
128,36,147,128,60,177,32
63,255,252,64,255,254,158
112,6,191,55,254,63,183
190,115,150,242,237,214,236
222,215,222,222,215,191,237
255,243,115,131,51,127,128
63,63,0,30,30,0,12



FORKLIFT

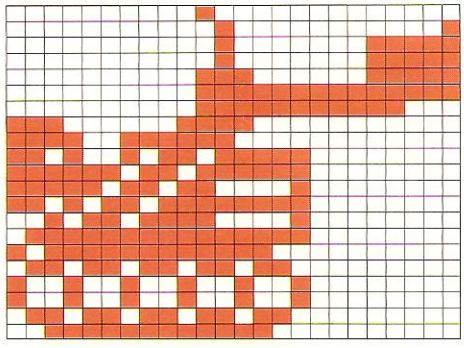


0,0,223,0,0,217,0
0,213,1,254,211,1,2
213,1,50,217,1,50,213
1,98,211,1,122,223,1
118,255,125,114,192,255,254
192,231,242,192,219,238,192
189,222,192,230,179,192,218
173,192,218,173,192,231,243
192,60,30,192,24,12,0



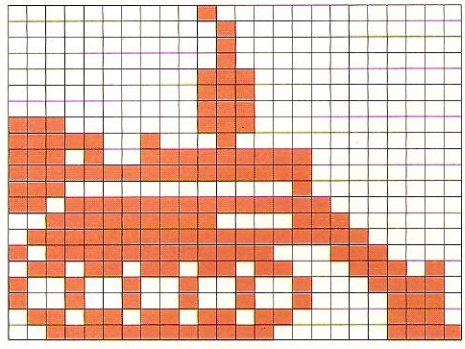
BULLDOZER

0,32,0,0,16,1,0
16,31,0,16,31,0,56
31,0,63,255,0,63,248
224,120,0,243,240,0,210
223,0,253,191,0,251,97
0,214,255,0,45,225,0
127,255,0,191,252,0,109
182,0,146,73,0,146,73
0,109,182,0,63,252,0



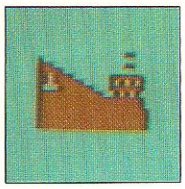
BULLDOZER

0,64,0,0,32,0,0
32,0,0,32,0,0,112
0,0,112,0,0,112,0
224,112,0,249,32,0,233
255,0,255,255,0,224,1
0,223,255,128,63,225,192
127,255,224,191,252,112,109
182,58,146,73,30,146,73
14,109,182,14,63,252,15

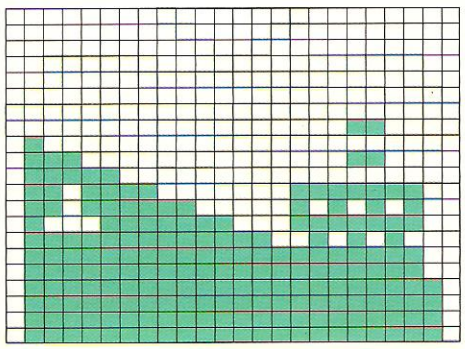


SHIPS AND BOATS

FREIGHTER

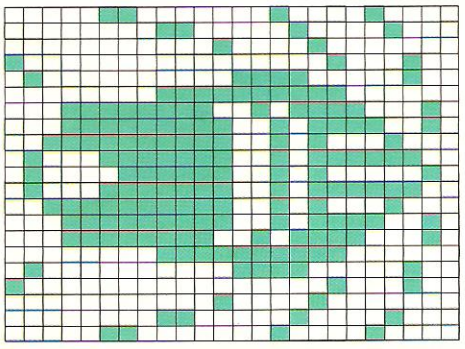


```
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,48,64,0,0,112
0,48,124,0,0,111,1
252,111,193,84,71,241,252
127,252,168,127,255,252,127
255,252,127,255,254,127,255
254,127,255,254,127,255,254
```



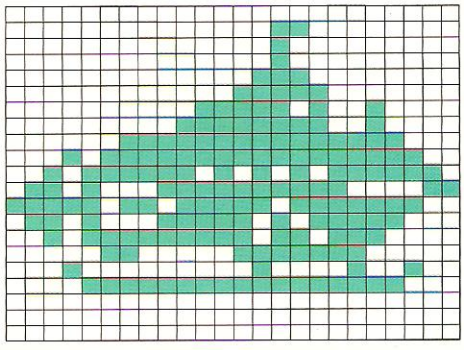
SPEEDBOAT (FROM ABOVE)

```
6,2,16,0,193,8,0
16,128,128,0,68,64,15
4,0,255,192,31,229,226
31,242,242,63,242,136,71
242,252,67,242,130,71,242
252,63,242,136,31,242,242
31,229,226,0,255,192,64
15,4,128,0,68,0,16
128,0,193,8,6,2,16
```

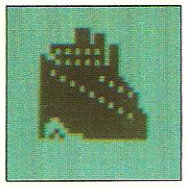


SUBMERSIBLE

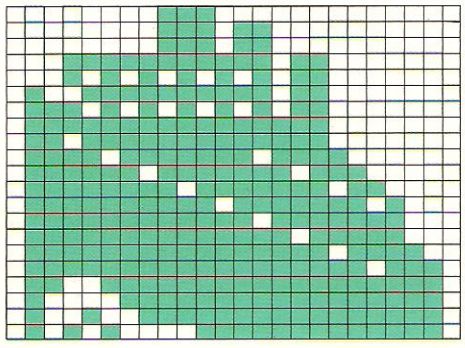
```
0,0,0,0,3,0,0
2,0,0,2,0,0,7
0,0,15,128,0,62,144
0,127,144,3,255,248,23
255,252,47,213,126,104,255
195,251,125,252,104,250,248
47,251,112,6,15,224,16
4,36,15,255,248,0,0
0,0,0,0,0,0,0
```



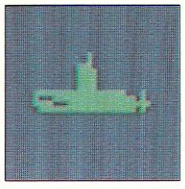
LINER



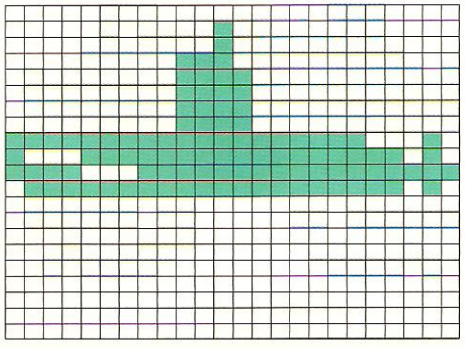
```
0,224,0,0,236,0,0
236,0,31,253,128,21,85
128,79,253,128,117,85,128
127,255,128,111,255,128,123
251,192,126,254,224,127,191
176,127,239,232,127,251,248
127,254,248,127,255,188,127
255,238,103,255,254,67,255
254,73,255,254,84,63,254
```



SUBMARINE

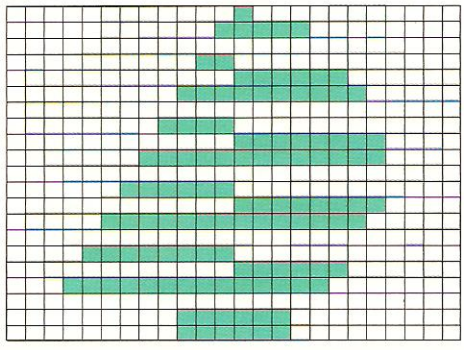


```
0,0,0,0,16,0,0
16,0,0,112,0,0,120
0,0,120,0,0,120,0
0,120,0,255,255,226,143
255,250,241,255,255,127,255
250,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
```

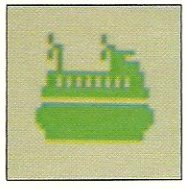


YACHT

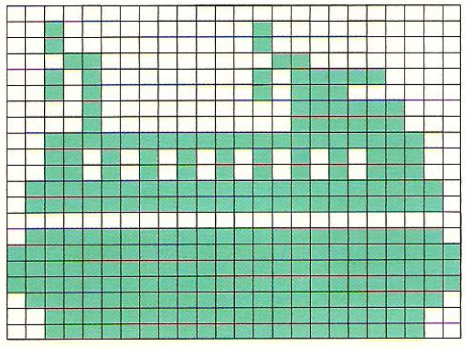
```
0,8,0,0,31,0,0
0,0,0,48,0,0,15
192,0,127,224,0,0,0
0,240,0,0,15,240,1
255,240,0,0,0,3,240
0,0,15,240,7,255,224
0,0,0,15,240,0,0
15,192,31,255,128,0,0
0,0,126,0,0,126,0
```



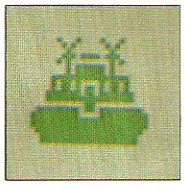
HOVERCRAFT



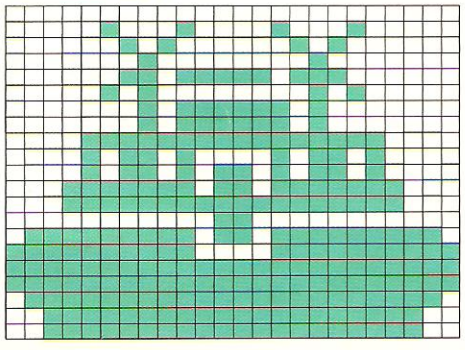
```
0,0,0,32,4,0,32
4,0,24,3,0,40,5
240,40,5,192,8,1,248
8,1,248,63,255,252,53
85,92,53,85,92,127,255
254,127,255,254,0,0,0
127,255,254,255,255,255,255
255,255,255,255,255,127,255
254,63,255,252,63,255,252
```



HOVERCRAFT

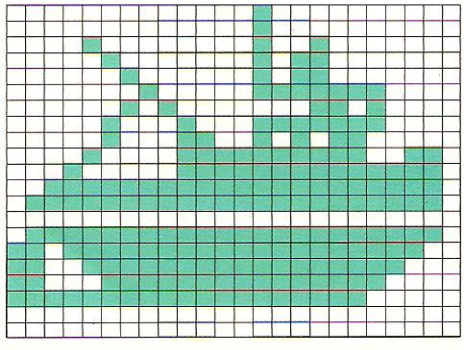


```
0,0,0,4,66,32,2
129,64,1,0,128,3,189
192,5,0,160,1,126,128
1,126,128,15,255,240,11
66,208,11,90,208,31,219
248,31,255,248,0,24,0
127,219,254,255,195,255,255
255,255,255,255,255,127,255
254,63,255,252,63,255,252
```

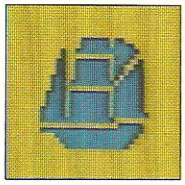


TUGBOAT

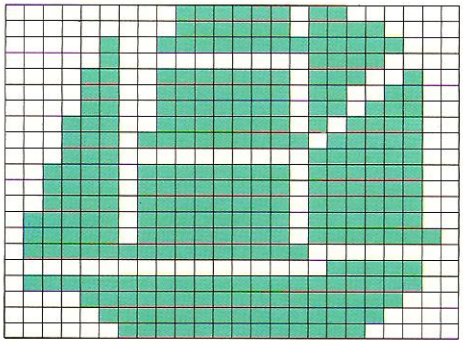
```
0,4,0,0,4,0,8
4,128,4,5,128,2,5
128,1,7,240,2,128,208
4,71,240,4,101,96,8
127,230,16,127,254,63,255
254,127,255,254,0,0,0
127,255,254,223,255,252,207
255,248,199,255,240,255,255
224,0,0,0,0,0,0
```



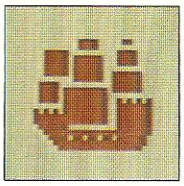
TALL SHIP



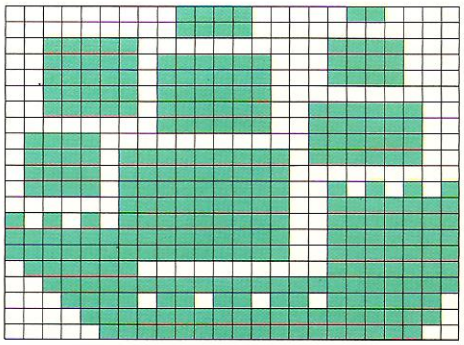
0,126,192,0,126,240,4
255,248,4,0,240,12,254
228,12,254,204,12,254,220
28,254,188,29,255,124,28
0,252,61,254,252,61,254
252,61,254,252,125,254,252
125,254,254,127,255,0,0
0,124,127,255,252,15,255
248,7,255,240,3,255,224



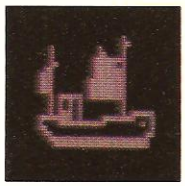
MAN O' WAR



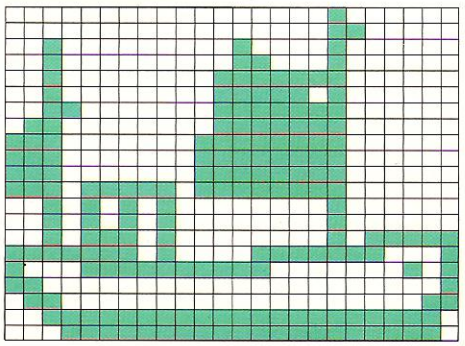
0,120,48,0,120,0,62
0,120,62,252,120,62,252
120,62,252,0,62,252,252
0,252,252,120,0,252,123
254,252,123,254,0,123,254
85,3,254,127,171,254,127
255,254,127,255,254,127,126
0,127,63,255,255,30,219
126,15,255,254,7,255,252



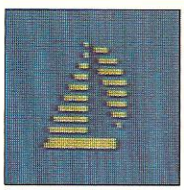
FISHING SMACK



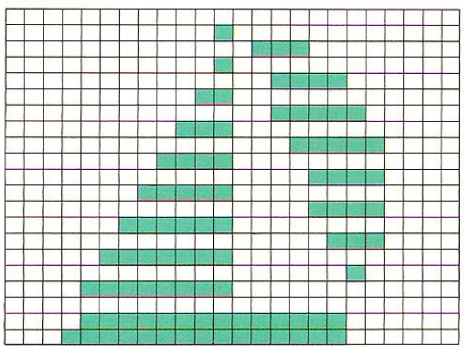
0,0,64,0,0,96,32
8,64,32,12,64,32,31
192,48,31,64,96,31,192
224,31,192,224,63,192,224
63,192,224,63,192,47,191
192,42,128,64,42,128,64
46,128,95,254,135,241,143
252,5,192,0,1,96,0
3,63,255,254,31,255,254



YACHT



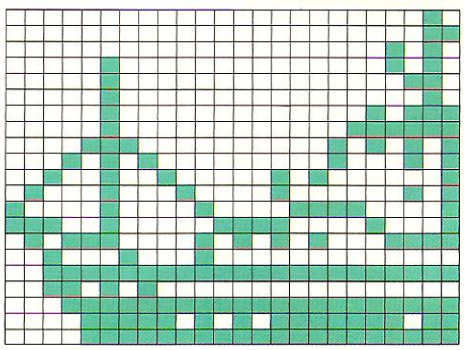
0,0,0,0,16,0,0
7,0,0,16,0,0,3
192,0,48,0,0,3,224
0,112,0,0,1,240,0
240,0,0,0,240,1,240
0,0,0,240,3,240,0
0,0,112,7,240,0,0
0,32,15,240,0,0,0
0,15,255,192,31,255,192

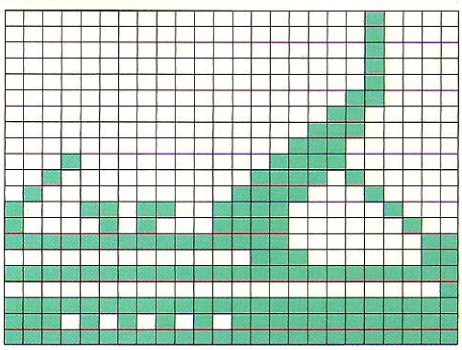


STERN TRAWLER

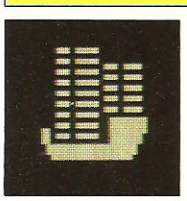
0,0,2,0,0,3,0
0,10,4,0,10,4,0
6,4,0,2,4,0,23
4,0,63,14,0,85,21
0,69,36,131,129,68,66
5,132,33,225,228,30,17
92,42,143,34,32,0,31
255,255,29,64,0,15,255
255,7,213,251,7,255,255

0,0,16,0,0,16,0
0,16,0,0,16,0,0
16,0,0,48,0,0,96
0,0,224,0,1,192,16
3,192,32,7,160,64,15
16,141,158,8,133,30,4
255,254,3,0,7,3,255
255,255,0,0,1,255,255
255,234,175,255,255,255,255

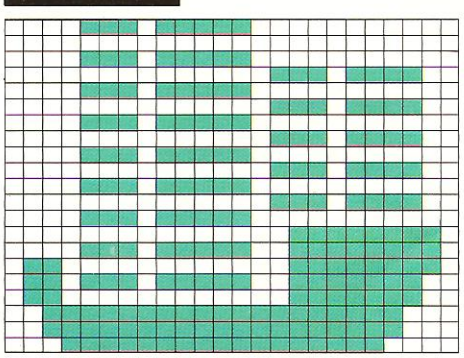




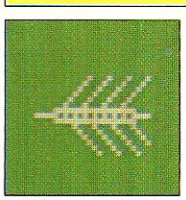
JUNK



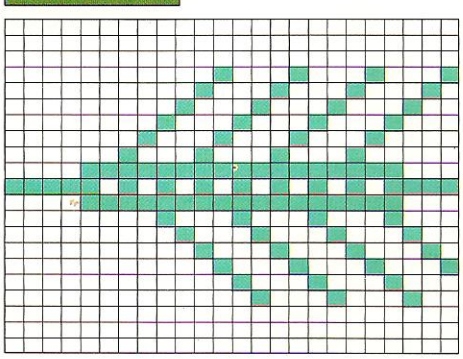
14,248,0,0,0,0,14
248,0,0,3,188,14,248
0,0,3,188,14,248,0
0,3,188,14,248,0,0
3,188,14,248,0,0,3
188,14,248,0,0,1,254
14,249,254,96,1,254,110
249,252,96,1,252,63,255
248,63,255,248,31,255,240



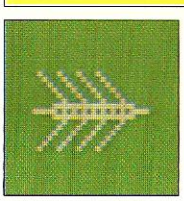
ROWING EIGHT



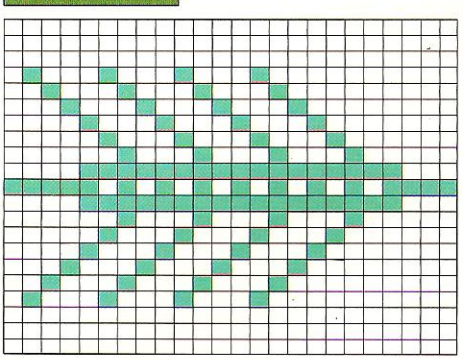
0,0,0,0,0,0,0
0,0,0,17,17,0,34
34,0,68,68,0,136,136
1,17,16,2,34,32,15
255,248,250,170,175,15,255
248,0,136,136,0,68,68
0,34,34,0,17,17,0
8,136,0,4,68,0,0
0,0,0,0,0,0,0



ROWING EIGHT

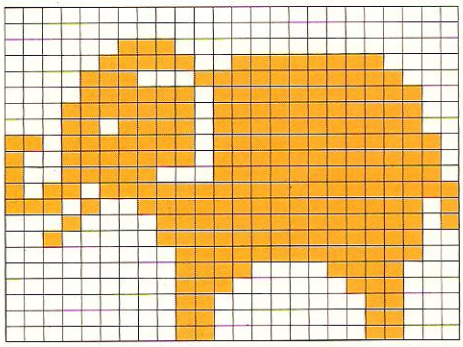


0,0,0,0,0,0,0
0,0,68,68,0,34,34
0,17,17,0,8,136,128
4,68,64,2,34,32,15
255,248,250,170,175,15,255
248,2,34,32,4,68,64
8,136,128,17,17,0,34
34,0,68,68,0,0,0
0,0,0,0,0,0,0



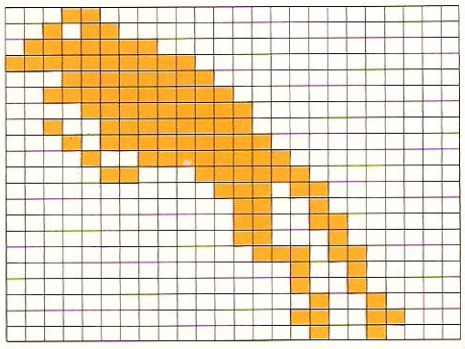
ELEPHANT

0,0,0,0,0,0,3
128,0,7,207,240,14,63
248,15,223,252,31,223,252
27,223,252,223,223,254,159
223,254,159,31,254,183,255
255,233,255,253,16,255,253
32,255,252,0,127,252,0
120,120,0,112,56,0,96
24,0,96,24,0,96,24



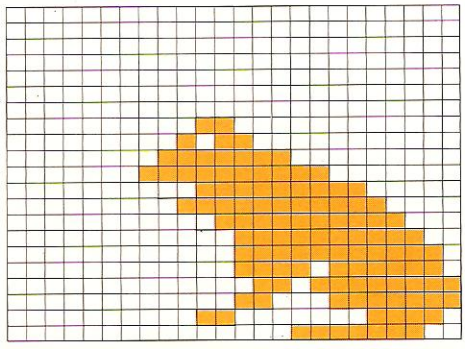
FROG

24,0,0,44,0,0,127
0,0,255,192,0,31,224
0,63,240,0,15,248,0
39,248,0,23,252,0,9
254,0,6,63,0,0,29
128,0,12,128,0,12,192
0,12,64,0,7,96,0
1,32,0,1,32,0,0
144,0,1,152,0,0,144



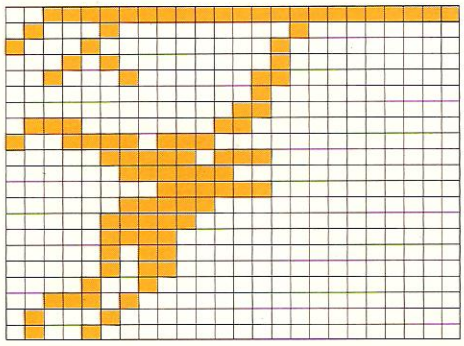
FROG

0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,48,0,0,88,0,0
254,0,1,255,128,0,31
224,0,127,240,0,31,248
0,15,252,0,15,254,0
7,126,0,6,255,0,12
127,0,48,30,0,1,252



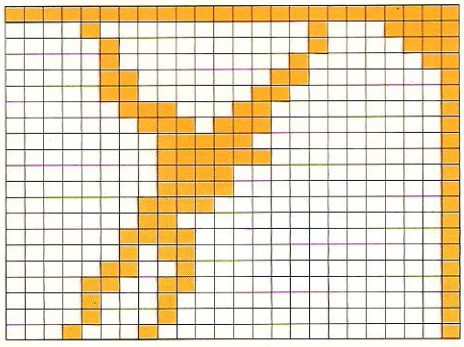
GIBBON

63,255,255,68,1,0,
2,0,20,2,0,34,6
0,0,4,0,0,12,0
112,24,0,158,224,0,7
220,0,3,248,0,0,252
0,3,224,0,7,192,0
7,128,0,5,128,0,5
128,0,9,0,0,58,0
0,68,0,0,72,0,0



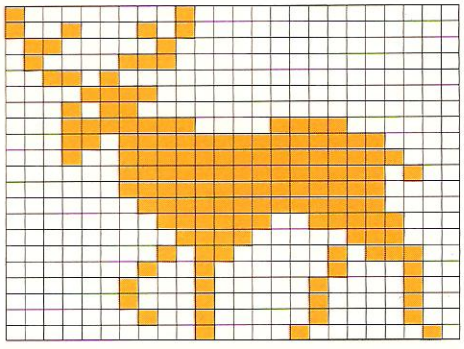
GIBBON

255,255,255,8,0,143,
0,135,4,1,3,6,3
1,6,6,1,3,140,1
1,220,1,0,248,1,0
252,1,0,240,1,0,240
1,1,224,1,1,128,1
3,192,1,2,192,1,6
64,1,12,192,1,8,128
1,8,128,1,17,0,1



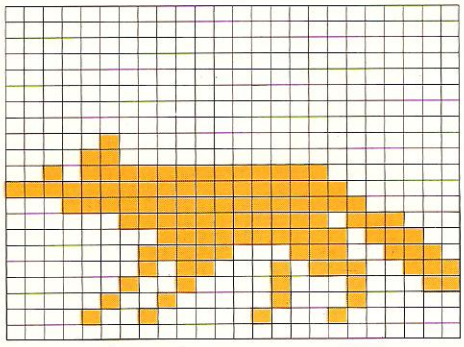
MOOSE

128,64,0,145,64,0,96
128,0,67,128,0,52,0
0,15,0,0,22,0,0
31,195,192,27,255,240,19
255,248,3,255,244,3,255
240,1,255,240,0,252,248
0,120,56,0,176,92,1
32,68,2,32,132,2,32
132,1,32,132,0,33,2



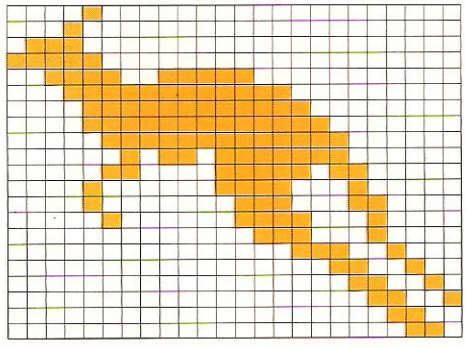
FOX

0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,4,0,0,12
0,0,47,255,128,255,255
192,31,255,224,3,255,184
0,255,220,1,241,206,1
98,231,2,66,35,4,130
32,9,4,64,0,0,0



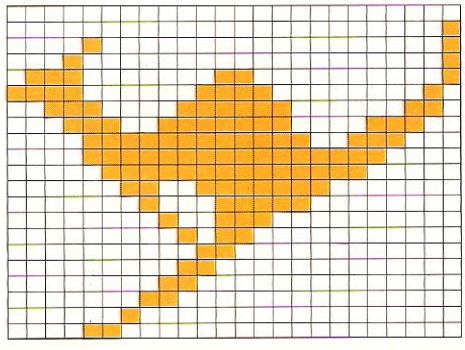
KANGAROO

8,0,0,16,0,0,112
0,0,248,0,0,60,248
0,31,254,0,7,255,0
7,255,128,3,255,192,1
111,192,3,15,96,4,14
32,4,14,48,2,15,16
0,7,16,0,1,200,0
0,100,0,0,50,0,0
25,0,0,12,0,0,6

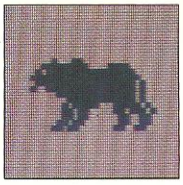


KANGAROO

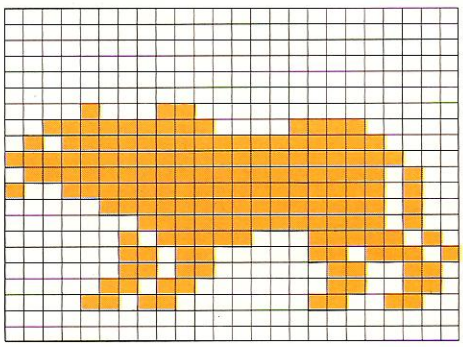
0,0,0,0,0,1,8
0,1,16,0,1,112,24
1,240,60,2,56,126,6
28,254,28,15,255,56,15
255,240,7,255,224,3,63
128,1,31,0,0,156,0
0,88,0,0,48,0,0
96,0,0,192,0,1,128
0,2,0,0,12,0,0



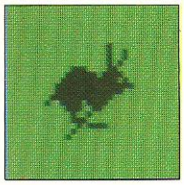
TIGER



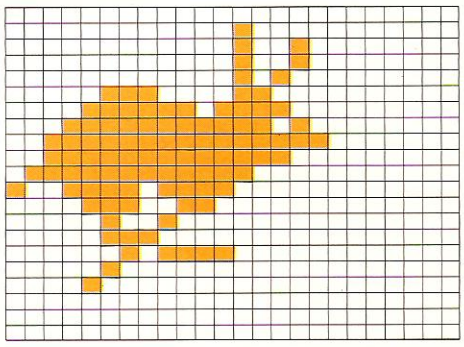
0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0
0,0,0,0,8,192,0
63,225,224,95,255,240,255
255,248,127,255,244,159,25
244,7,255,244,1,255,244
2,248,250,3,96,221,3
96,102,6,192,102,13,128
204,0,0,0,0,0,0



RABBIT

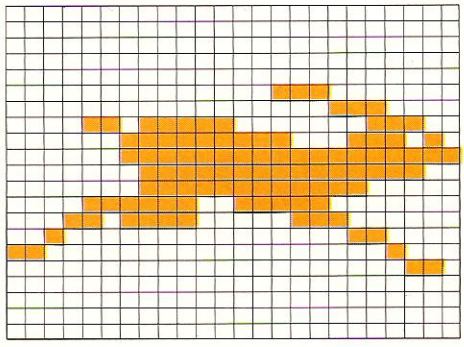


0,0,0,0,8,0,0
9,0,0,9,0,0,10
0,7,12,0,15,222,0
31,253,0,63,255,128,63
254,0,127,248,0,158,240
0,14,96,0,4,128,0
7,0,0,2,240,0,4
0,0,8,0,0,0,0
0,0,0,0,0,0,0



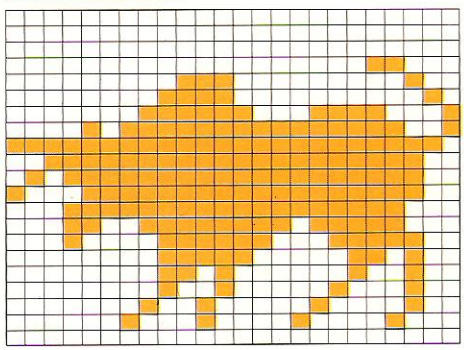
RABBIT

0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,3,128,0,0,112
13,240,28,3,254,58,3
255,255,1,255,252,1,255
224,15,207,128,27,193,192
32,0,48,192,0,8,0
0,6,0,0,0,0,0
0,0,0,0,0,0,0



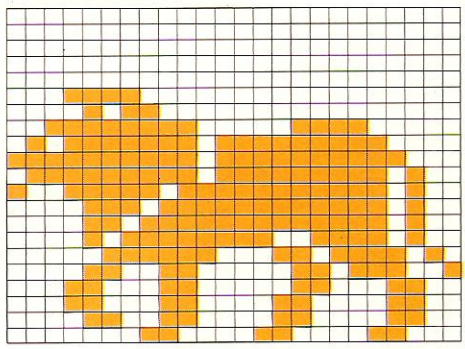
BUFFALO

0,0,0,0,0,0,0
0,0,0,0,24,0,112
4,0,240,2,1,248,241
11,255,249,255,255,254,63
255,252,95,255,252,143,255
252,31,255,248,25,254,248
16,252,124,0,248,52,0
208,36,0,144,68,1,32
132,2,33,8,0,0,0



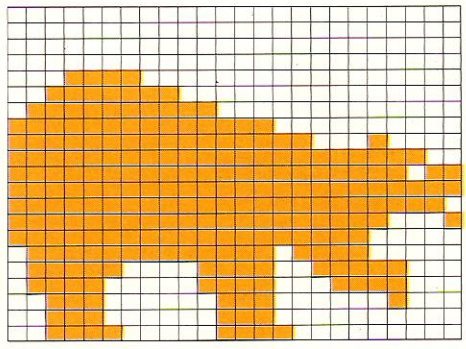
LION

0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,30,0,0,11,0,0
63,193,224,95,223,240,255
223,248,127,223,244,159,12
244,14,255,244,13,255,244
11,252,244,7,227,122,12
193,185,24,193,140,24,195
12,12,195,12,1,134,24



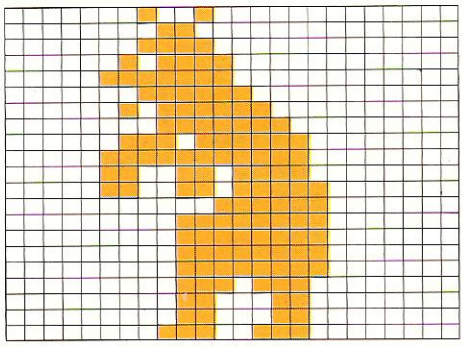
POLAR BEAR

0,0,0,0,0,0,0
0,0,0,0,0,30,0
0,63,128,0,127,224,0
255,252,0,255,255,16,255
255,252,255,255,251,255,25
255,255,255,254,255,255,24
255,255,224,127,253,224,12
60,248,120,60,120,56,28
8,56,28,0,60,30,0



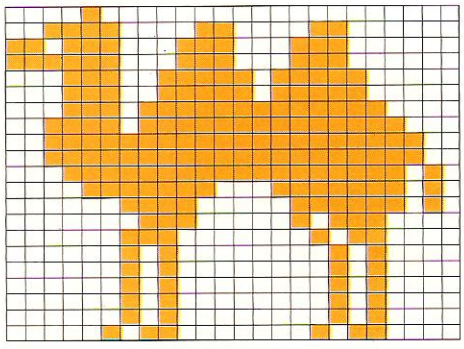
BROWN BEAR

1,32,0,0,192,0,1
224,0,3,240,0,7,240
0,1,248,0,2,252,0
0,254,0,3,191,0,7
255,0,6,111,0,6,111
128,0,31,128,0,127,128
0,127,128,0,127,128,0
127,128,0,119,0,0,99
0,0,98,0,0,231,0

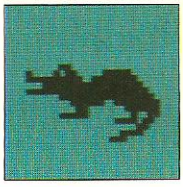


CAMEL

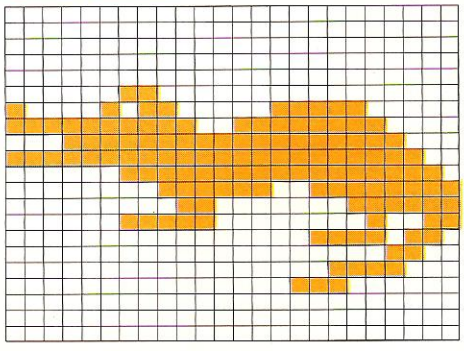
8,0,0,60,48,192,220
113,192,252,113,192,28,251
224,28,251,224,29,255,240
61,255,248,63,255,252,63
255,252,31,255,250,15,227
250,3,193,250,1,193,112
3,128,176,2,128,80,2
128,80,2,128,80,2,128
80,2,128,80,5,128,176



CROCODILE

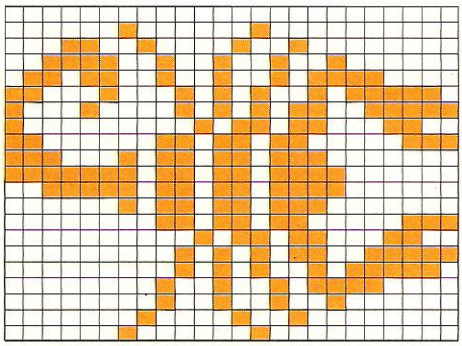


0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,3,0,0,133,131,224
255,207,240,31,255,248,25
255,252,3,255,254,0,252
255,0,96,63,3,224,19
0,0,246,0,0,12,0
0,120,0,1,192,0,0
0,0,0,0,0,0,0



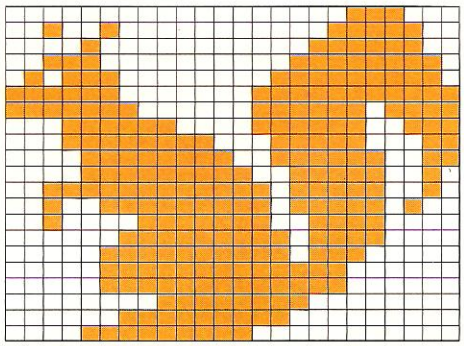
SCORPION

0,0,0,2,4,0,25
9,0,60,146,64,108,146
240,196,84,190,136,85,159
128,45,140,192,219,7,226
219,128,126,219,192,62,219
192,2,219,128,0,219,7
0,45,140,0,85,159,0
84,190,0,146,240,0,146
64,1,9,0,2,4,0

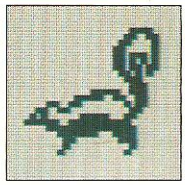


SQUIRREL

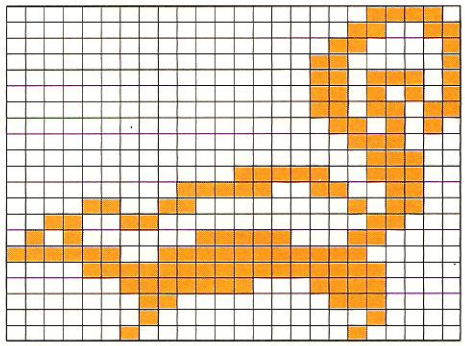
0,0,56,36,0,124,8
0,252,56,1,254,92,3
254,252,7,255,254,7,239
31,135,231,31,227,227,15
243,243,15,249,242,63,253
242,39,253,244,33,252,240
3,254,240,7,254,224,7
255,192,7,255,192,0,255
0,3,254,0,15,248,0



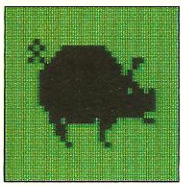
SKUNK



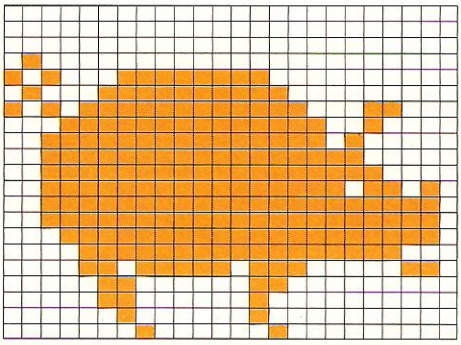
0,0,30,0,0,51,0
0,97,0,0,193,0,0
221,0,0,213,0,0,219
0,0,106,0,0,56,0
0,28,0,15,156,0,124
204,12,192,44,51,0,24
80,62,48,252,255,224,15
255,240,3,248,240,1,128
112,1,0,16,2,0,32



PIG

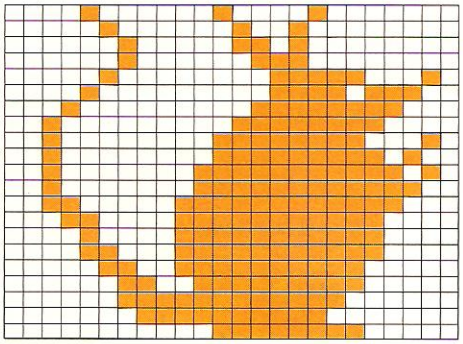


0,0,0,0,0,0,0
0,0,64,0,0,163,252
0,71,254,0,175,255,24
31,255,144,63,255,160,63
255,224,63,255,240,63,255
218,63,255,254,63,255,254
31,255,254,15,255,248,13
247,6,6,6,0,6,4
0,2,4,0,1,2,0

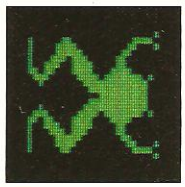


MOUSE

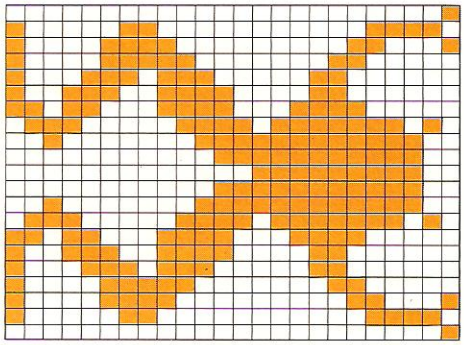
8,16,128,4,9,0,2
13,0,2,7,0,4,3
226,8,3,220,16,7,248
32,15,240,32,31,226,32
31,244,32,63,250,32,63
252,48,127,248,24,127,240
8,127,240,12,127,240,6
127,224,3,63,224,3,255
192,1,255,128,0,31,224



FROG

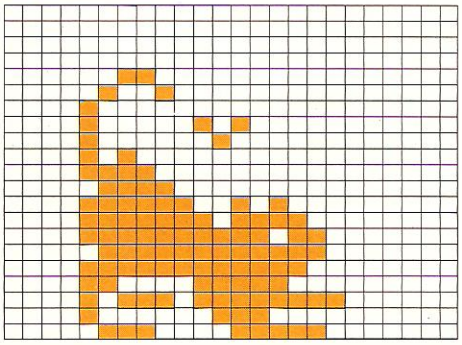


0,0,1,131,0,30,1
128,49,135,192,96,1
192,156,240,192,216,2
112,123,250,32,63,252,0
31,252,0,7,252,0,31
252,32,63,252,112,123,25
216,241,224,156,240,192,
224,192,135,192,96,135,1
49,131,0,30,0,0,1

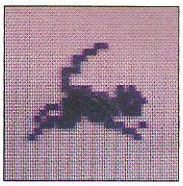


CAT

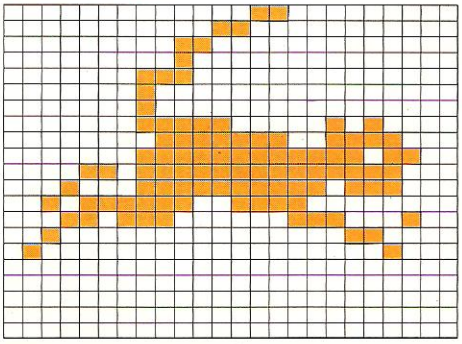
0,0,0,0,0,0,0
0,0,0,0,0,3,0
0,4,128,0,8,64,0
8,40,0,8,16,0,10
0,0,15,0,0,7,128
0,15,202,0,15,239,0
15,253,128,7,255,128,15
255,0,12,62,0,11,191
192,8,28,0,7,15,128



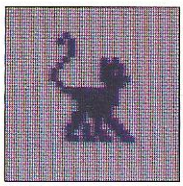
CAT



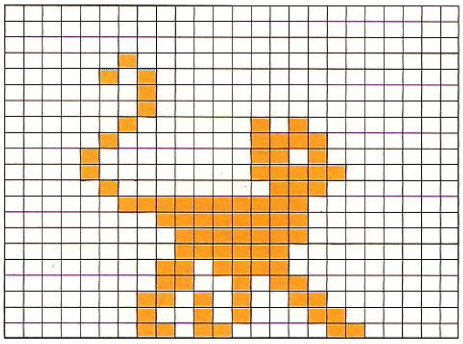
0,6,0,0,24,0,0
96,0,0,64,0,1,192
0,1,0,0,1,0,0
1,240,80,0,254,120,1
255,236,13,255,248,17,255
184,47,207,0,27,129,196
32,0,48,64,0,8,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0



CAT

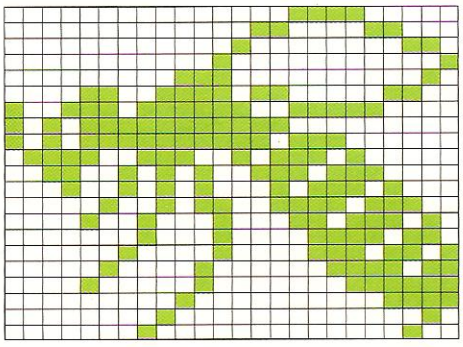


0,0,0,0,0,0,0
0,0,2,0,0,5,0
0,1,0,0,1,0,0
2,5,0,4,7,128,8
6,128,8,7,192,4,3
128,3,255,0,0,255,0
0,127,0,0,126,0,0
222,0,0,203,0,1,169
128,1,44,192,1,182,96

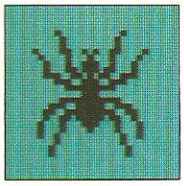


WASP

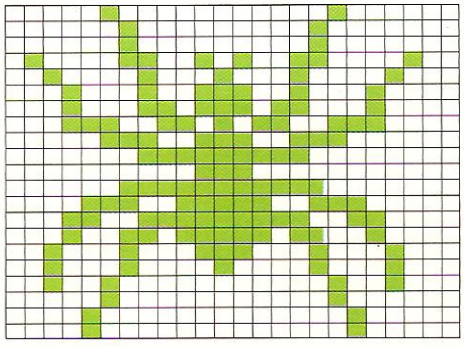
0,1,224,0,6,24,0
8,6,0,16,1,0,112
2,12,246,12,157,249,240
175,254,0,175,253,192,121
203,160,50,167,112,18,162
232,4,179,220,9,17,186
1,16,246,2,16,237,4
32,123,8,32,63,0,64
15,0,128,2,1,0,4



SPIDER

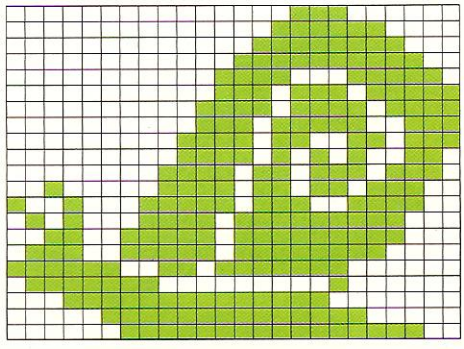


4,0,64,2,0,128,2
0,128,67,41,132,33,17
8,17,57,16,17,187,16
28,214,112,7,57,192,1
255,0,0,56,0,3,255
128,14,124,224,25,255,48
19,125,144,34,56,136,34
16,136,36,0,72,4,0
64,8,0,32,8,0,32



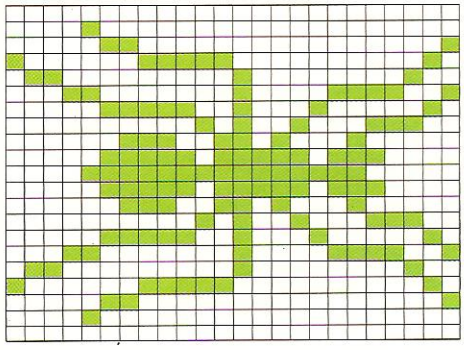
SNAIL

0,1,192,0,3,240,0
7,248,0,15,252,0,30
62,0,29,223,0,61,239
0,59,231,0,123,55,0
122,166,0,246,238,32,246
220,145,247,60,81,239,248
51,239,248,115,239,240,252
223,192,127,0,64,31,255
128,7,255,224,3,255,252



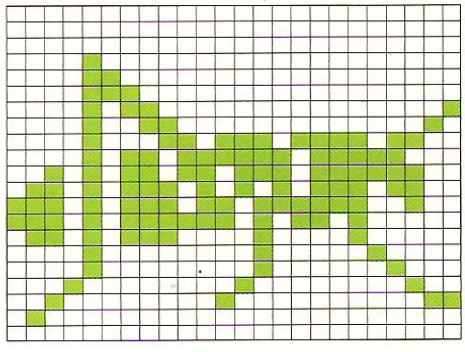
ANT

0,0,0,8,0,0,6
0,1,129,240,2,96,8
4,24,8,121,7,200,130
0,41,28,3,154,32,7
223,112,15,255,224,7,223
112,3,154,32,0,41,28
7,200,130,24,8,121,96
8,4,129,240,2,6,0
1,8,0,0,0,0,0

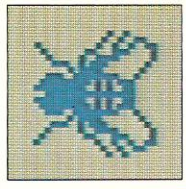


CRICKET

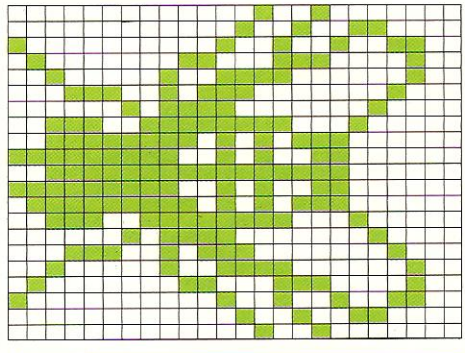
0,0,0,0,0,0,0
0,0,8,0,0,12,0
0,14,0,0,11,0,1
9,128,2,8,221,236,11
110,184,43,110,188,107,186
236,235,183,228,235,245,16
107,132,32,8,4,16,8
4,8,16,8,4,32,16
3,64,0,0,0,0,0



FLY

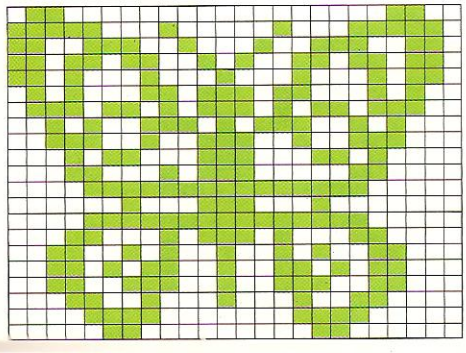


0,4,128,0,9,48,128
18,76,64,99,132,32,158
4,28,184,8,2,240,16
57,254,32,127,213,192,255
148,192,63,255,192,255,148
192,127,213,192,57,254,32
2,240,16,28,184,8,32
158,4,64,99,132,128,18
76,0,9,48,0,4,128



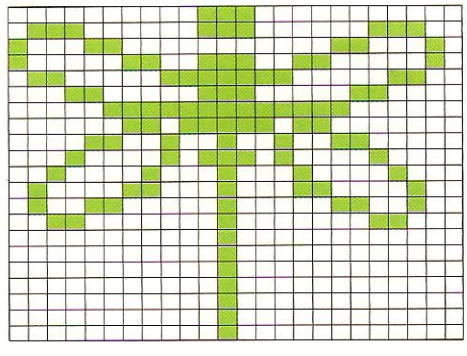
BUTTERFLY

96,0,12,248,130,62,206
68,230,219,41,182,209,17
22,81,187,20,118,186,220
41,215,40,57,57,56,22
56,208,24,186,48,15,255
224,2,56,128,15,255,224
24,186,48,50,146,152,53
147,88,51,147,152,25,17
48,15,1,224,6,0,192

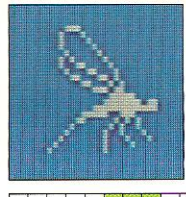


DRAGONFLY

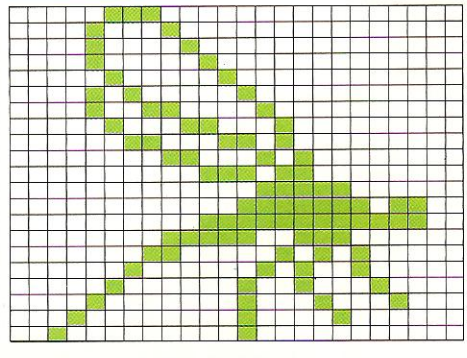
0,56,0,112,56,28,140
0,98,131,57,130,96,254
12,24,56,48,7,255,192
3,125,128,12,146,96,16
186,16,33,17,8,67,17
132,76,16,100,48,16,24
0,16,0,0,16,0,0
16,0,0,16,0,0,16
0,0,16,0,0,16,0



DRAGONFLY



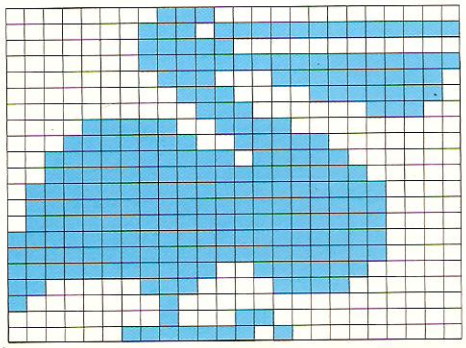
7,0,0,8,128,0,8
64,0,8,32,0,4,16
0,10,8,0,9,132,0
4,98,0,3,26,0,0
197,0,0,59,0,0,7
192,0,15,236,0,63,252
0,249,192,1,130,160,2
5,32,4,9,16,8,8
136,16,8,64,32,8,0



BIRDS

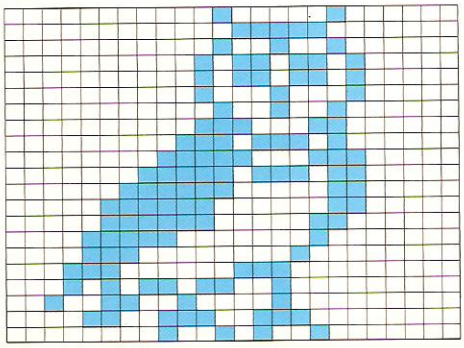
PELICAN

15,231,240,30,0,120,31
231,248,92,0,58,92,0
58,92,66,50,76,255,34
167,255,229,147,255,201,143
255,241,71,255,226,63,255
252,7,255,224,63,255,252
67,255,194,141,255,177,144
126,9,160,0,5,160,0
5,16,0,8,8,0,16



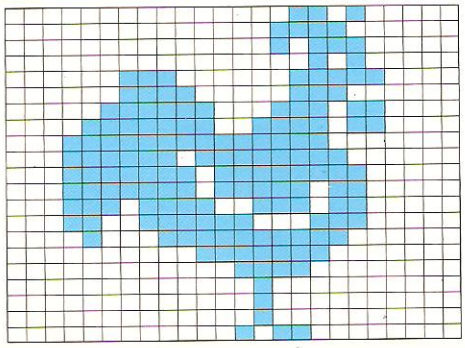
OWL

0,0,120,0,0,246,0
1,255,0,1,254,0,3
254,0,3,243,0,7,241
0,15,225,0,63,225,31
255,225,127,255,242,255,255
242,255,255,244,255,255,240
255,255,240,255,253,224,255
249,216,249,243,216,252,3
220,126,3,140,31,143,135



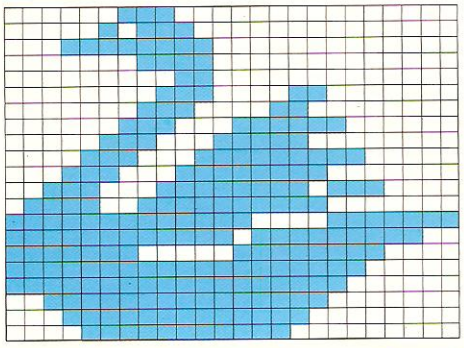
COCKEREL

0,0,0,0,0,0,0
0,55,0,0,127,0,0
126,0,0,252,0,1,248
0,3,240,0,15,240,0
127,240,3,255,240,31,255
248,63,255,248,127,255,248
255,255,248,255,255,244,253
255,244,252,56,246,127,0
238,63,129,199,31,199,129



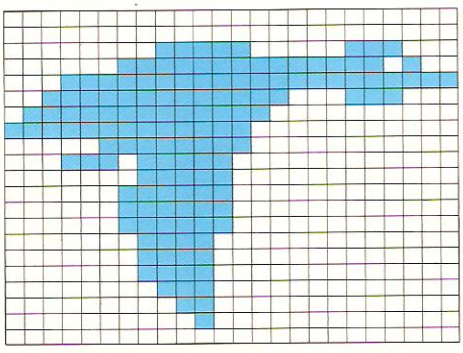
SWAN

0,0,2,0,0,14,0
0,62,0,0,126,0,0
254,0,1,254,0,15,255
0,255,255,3,255,255,13
250,191,63,250,191,127,250
191,255,58,191,252,254,191
115,255,255,3,255,254,0
31,240,0,3,240;0,0
240,0,0,48,0,0,0



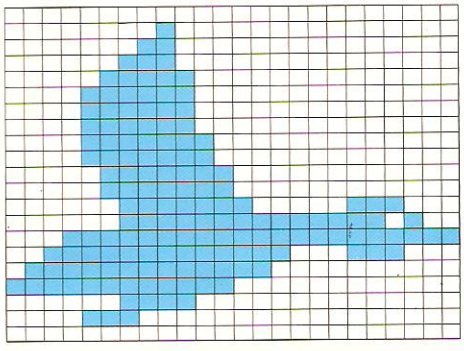
DUCK

0,0,0,0,0,0,0
0,3,0,0,6,0,0
12,0,0,28,240,8,56
255,152,120,255,252,240,255
255,224,255,255,224,255,254
240,255,236,112,254,4,56
224,0,12,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0



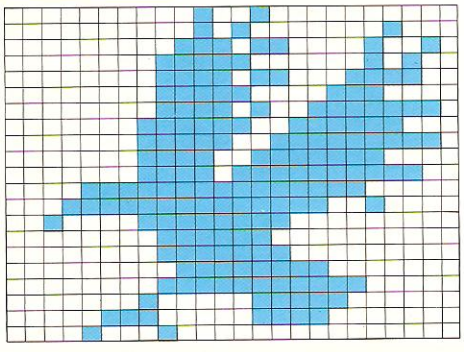
DUCK

0,24,0,0,60,0,0
60,0,0,126,0,0,126
0,0,126,0,0,126,0
120,90,30,78,255,114,195
255,195,155,255,217,166,126
101,161,255,133,166,36,101
164,230,37,164,129,37,180
145,37,146,74,105,81,36
106,8,144,144,4,77,32



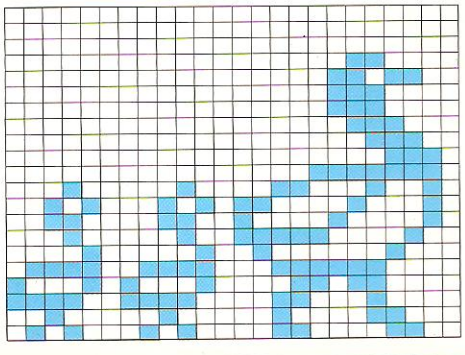
EAGLE

0,4,192,0,9,0,0
18,96,0,148,158,1,85
48,2,85,62,98,86,113
224,46,198,240,127,152,127
255,224,127,255,192,127,255
224,240,127,152,224,46,198
98,86,113,2,85,62,1
85,48,0,148,158,0,18
96,0,9,0,0,4,192



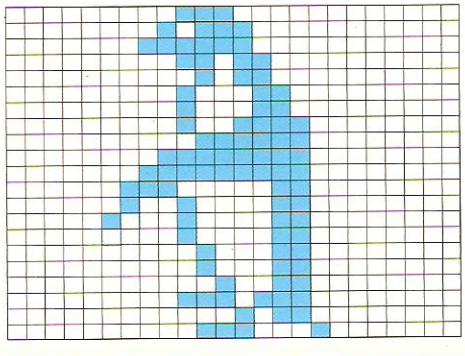
DUCK AND DUCKLINGS

0,6,0,0,15,0,1
247,128,7,251,128,15,253
2,31,254,6,61,255,78
126,255,158,119,127,252,63
127,248,223,127,252,126,127
254,126,255,174,57,255,70
31,254,198,15,253,130,3
251,128,0,240,0,0,24
0,0,0,0,0,0,0

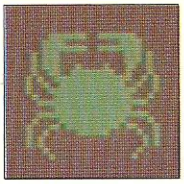


PENGUIN

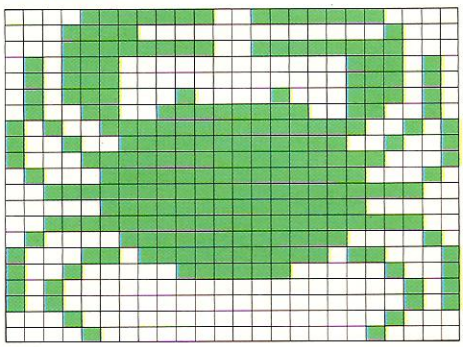
3,224,0,6,28,0,11
254,0,23,253,0,47,15
0,94,2,130,124,1,65
248,1,194,216,0,196,232
0,164,244,0,228,122,0
230,63,0,162,31,193,194
15,113,194,7,131,134,3
243,140,0,3,24,0,3
176,0,1,224,0,0,128



CRAB

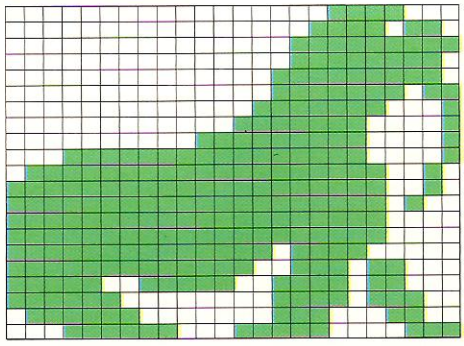


0,224,0,1,223,255,1
240,0,1,255,255,0,227
254,0,112,254,0,56,28
15,158,0,31,207,128,63
247,192,63,255,224,127,255
240,127,255,240,127,255,240
255,255,240,255,255,240,255
207,224,195,135,192,129,0
0,1,24,0,7,244,0

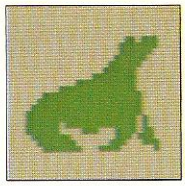


WALRUS

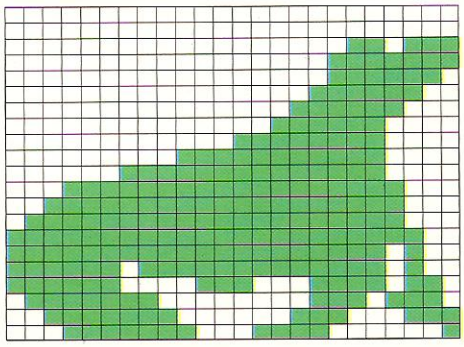
0,16,64,0,15,128,0
18,64,0,45,160,0,45
160,0,34,32,0,18,64
0,56,192,0,119,64,0
248,224,1,247,96,3,240
96,7,224,96,7,224,192
15,128,128,14,1,0,28
14,0,21,178,0,38,66
0,12,231,0,0,148,128



SEAL

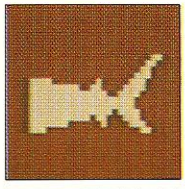


0,1,160,0,3,192,0
2,224,0,0,160,3,129
240,3,193,192,7,227,176
15,231,176,31,255,128,31
191,192,31,223,224,31,223
96,29,231,96,25,248,224
8,255,192,0,63,128,0
15,0,0,4,0,0,4
0,0,4,0,0,11,0

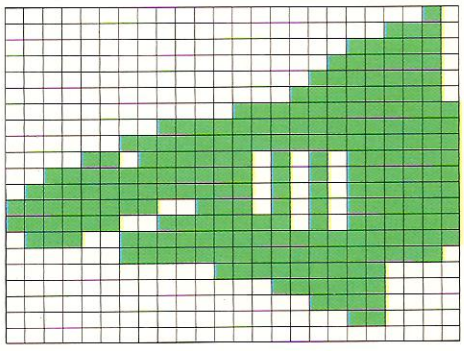


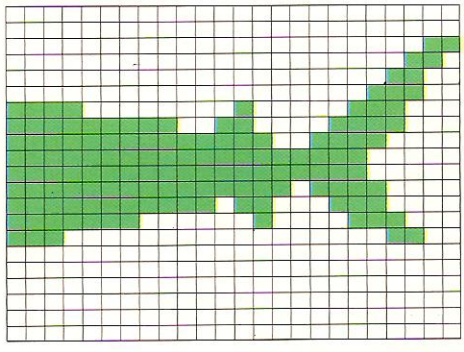
SHARK

3,128,0,5,192,0,31
224,0,1,224,0,0,96
0,0,225,128,1,195,0
3,143,192,7,31,0,14
63,224,28,127,128,56,255
112,121,255,128,255,248,127
255,247,252,254,15,240,255
255,224,255,255,192,63,255
128,31,255,0,15,254,0



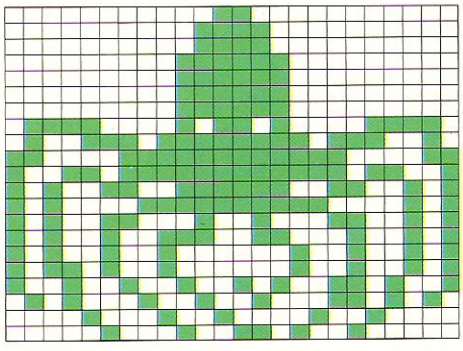
0,0,0,0,0,0,0
248,56,3,255,244,31,255
255,63,254,56,127,252,0
255,248,0,7,248,0,29
240,0,1,240,0,3,240
0,3,240,0,3,240,0
1,224,0,1,224,0,1
224,0,0,224,0,0,96
0,0,32,0,0,0,0





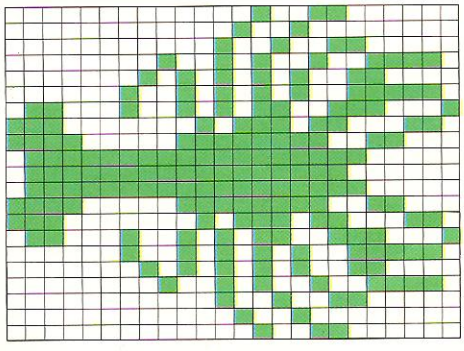
OCTOPUS

0,0,0,0,128,0,1
128,0,3,128,0,7,192
0,15,192,0,15,192,0
15,192,0,15,224,0,15
224,0,7,240,0,7,240
0,3,248,56,3,255,244
31,255,255,63,254,56,127
252,0,255,240,0,3,192
0,14,0,0,0,0,0

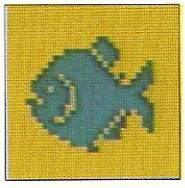


LOBSTER

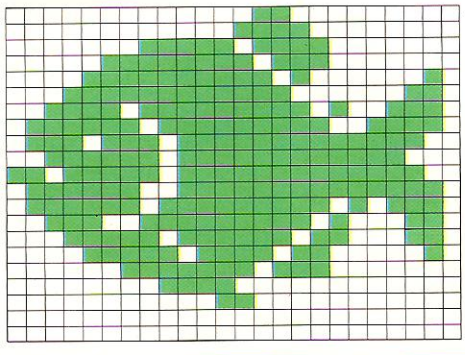
0,36,0,0,40,8,0
118,18,0,248,20,0,246
60,0,248,112,0,244,254
1,249,248,1,243,254,1
231,240,1,239,252,15,255
224,31,255,144,33,254,0
0,252,0,0,254,0,0
127,128,0,255,224,1,7
192,7,7,128,8,128,0



PIRANHA

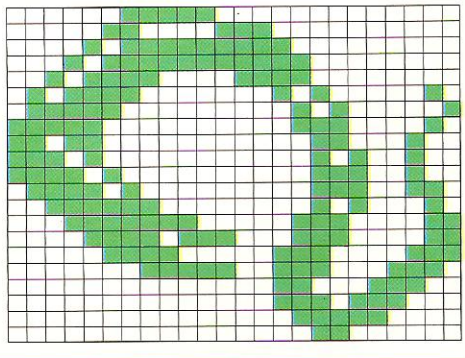


0,0,0,0,0,0,0
0,0,0,0,48,0,0
108,0,0,112,0,0,112
0,0,56,0,0,28,0
0,14,0,0,254,16,67
18,40,172,34,48,200,66
16,79,132,8,36,8,121
227,240,138,33,176,243,195
16,32,130,24,81,67,12



MORAY EEL

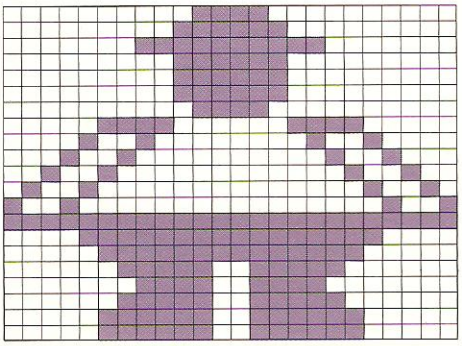
0,112,0,0,184,0,1
248,0,0,108,0,0,44
0,0,70,0,0,70,0
0,79,0,0,63,0,0
255,0,1,255,0,3,131
0,2,67,0,4,67,0
0,67,0,0,35,0,0
35,0,0,19,0,0,119
0,0,11,0,0,56,128



CHARACTERS

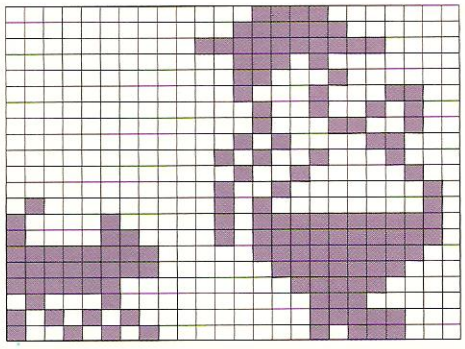
SHERRIFF

```
0,60,0,0,126,0,1
255,128,0,126,0,0,126
0,0,126,0,0,60,0
7,129,224,9,0,144,18
0,72,36,0,36,72,0
18,144,0,9,255,255,255
15,255,240,7,255,224,3
231,192,1,231,128,3,231
192,7,231,224,7,231,224
```



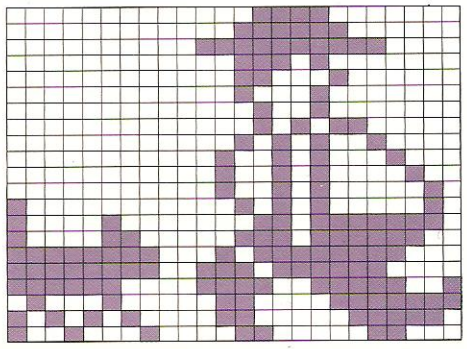
SHERRIFF

```
0,7,128,0,15,192,0
63,240,0,14,128,0,12
64,0,8,140,0,4,148
0,4,100,0,10,24,0
20,136,0,9,4,0,18
2,64,20,2,132,23,254
134,20,30,255,7,252,255
3,248,252,1,240,68,0
224,170,0,248,145,0,184
```



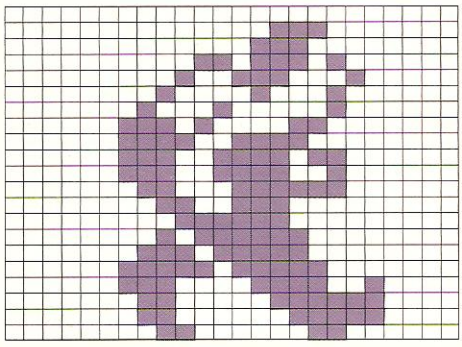
SHERRIFF

```
0,7,128,0,15,192,0
63,240,0,14,128,0,12
64,0,8,128,0,4,128
0,5,96,0,10,144,0
18,136,0,18,132,0,18
130,128,10,130,132,6,254
134,14,252,255,23,248,255
59,240,252,60,249,68,24
127,170,24,62,145,12,24
```



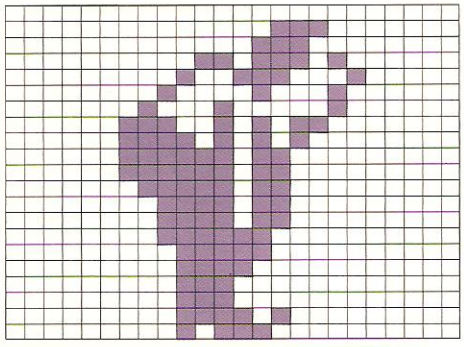
HUNCHBACK

```
0,0,0,0,3,128,0
7,0,0,55,64,0,78
32,0,132,64,1,16,64
3,161,128,3,207,0,3
158,192,3,158,64,1,143
192,0,206,0,0,126,0
0,191,0,1,223,0,3
239,128,3,135,144,1,131
240,1,129,240,0,192,192
```



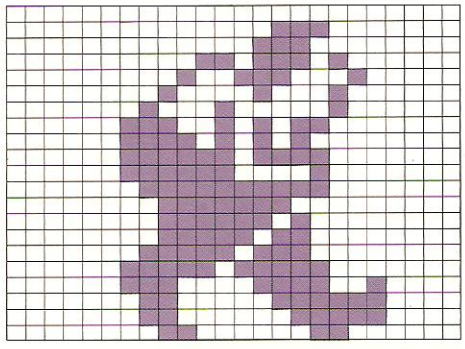
HUNCHBACK

```
0,0,0,0,3,128,0
7,0,0,55,64,0,78
32,0,132,64,1,16,64
3,149,128,3,213,0,3
246,0,3,246,0,1,246
0,0,246,0,0,246,0
0,252,0,0,124,0,0
120,0,0,112,0,0,112
0,0,122,0,0,92,0
```



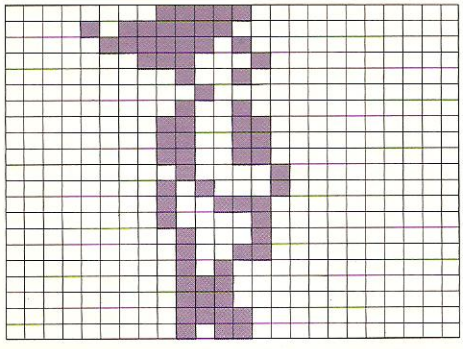
HUNCHBACK

```
0,0,0,0,3,128,0
7,0,0,55,64,0,78
32,0,132,64,1,16,64
3,149,128,3,213,0,3
247,128,3,240,128,1,255
128,0,254,0,0,125,0
0,251,0,1,247,0,3
239,128,3,135,144,1,131
240,1,129,240,0,192,192
```

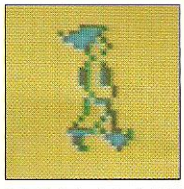


DWARF

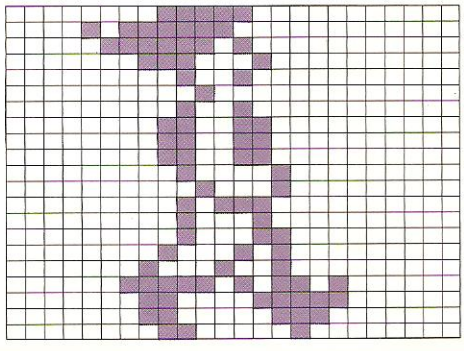
```
0,248,0,11,240,0,7
232,0,1,196,0,0,72
0,0,32,0,0,72,0
0,204,0,0,204,0,0
204,0,0,66,0,0,162
0,0,156,0,0,132,0
0,68,0,0,76,0,0
80,0,0,112,0,0,112
0,0,120,0,0,88,0
```



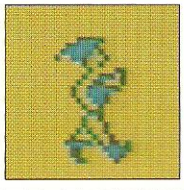
DWARF



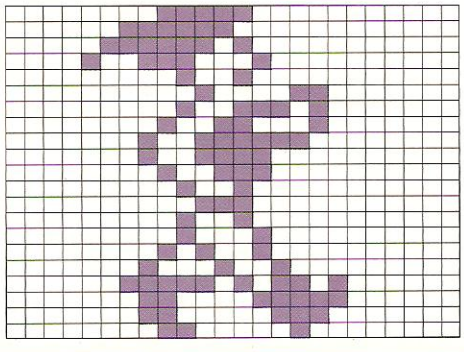
```
0,248,0,11,240,0,7
232,0,1,196,0,0,72
0,0,32,0,0,72,0
0,204,0,0,204,0,0
204,0,0,66,0,0,34
0,0,92,0,0,68,0
0,70,0,0,138,0,1
18,0,3,235,64,1,135
192,1,131,128,0,193,0
```



DWARF

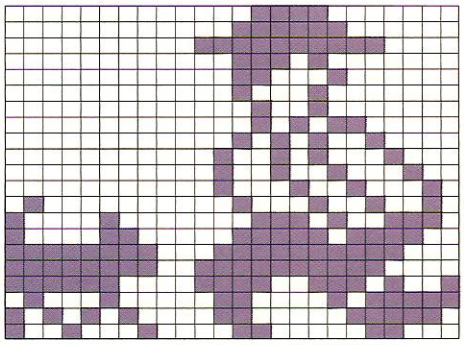


```
0,248,0,3,240,0,7
232,0,9,196,0,0,72
0,0,33,128,0,78,128
0,152,128,1,63,0,1
60,0,0,156,0,0,72
0,0,56,0,0,72,0
0,68,0,0,164,0,1
18,0,3,235,64,1,135
192,1,131,128,0,193,0
```



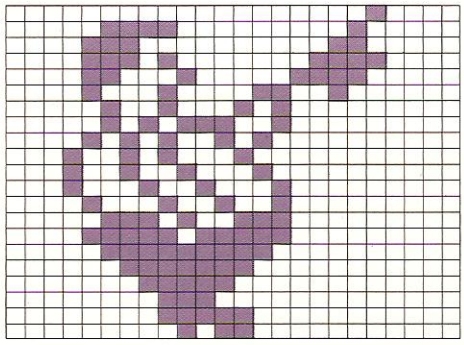
SHERRIFF

```
0,7,128,0,15,192,0
63,240,0,14,128,0,12
64,0,8,128,0,4,128
0,5,96,0,10,144,0
18,136,0,18,68,0,17
34,64,8,146,132,7,78
134,15,188,255,31,248,255
63,240,252,60,233,68,24
95,170,24,62,145,12,24
```



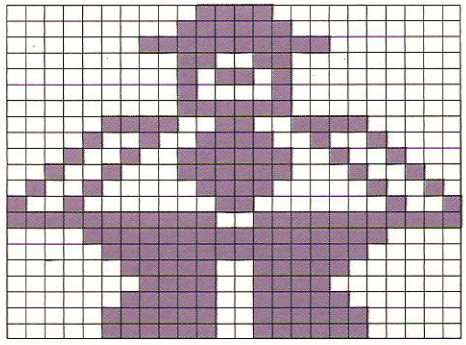
SHERRIFF

```
0,0,16,7,128,32,14
0,96,14,128,240,12,65
224,8,134,64,4,138,0
5,114,0,10,148,0,18
136,0,18,68,0,17,34
0,8,154,0,7,78,0
15,190,0,7,252,0,3
248,0,1,240,0,0,224
0,0,248,0,0,184,0
```



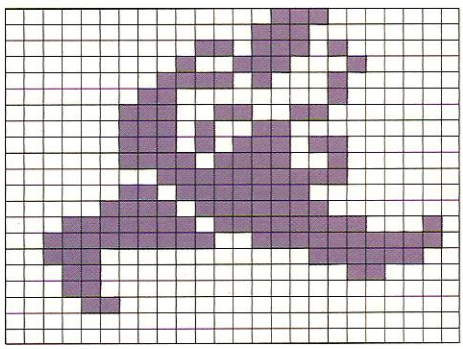
SHERRIFF

```
0,60,0,0,126,0,1
255,128,0,102,0,0,90
0,0,66,0,0,126,0
7,153,224,9,60,144,18
126,72,36,126,36,72,60
18,144,24,9,255,231,255
15,255,240,7,255,224,3
231,192,1,231,128,3,231
192,7,231,224,7,231,224
```

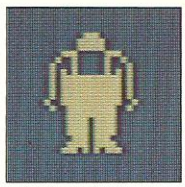


HUNCHBACK

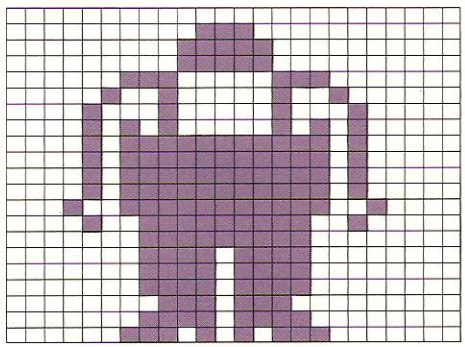
```
0,3,128,0,7,0,0
55,64,0,78,32,0,244
64,1,128,64,3,153,128
3,166,0,3,174,192,3
222,64,0,255,192,3,127
0,7,191,128,31,223,226
63,231,254,56,0,252,24
0,48,24,0,0,12,0
0,0,0,0,0,0,0
```



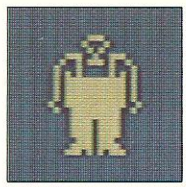
HUNCHBACK



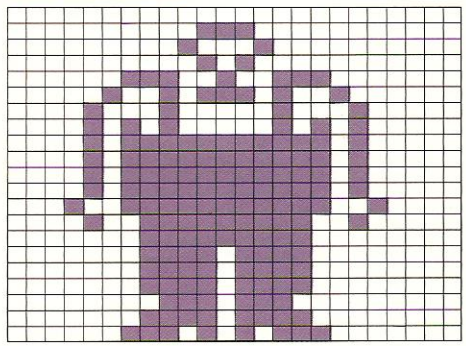
```
0,0,0,0,56,0,0
124,0,0,124,0,3,131
128,4,130,64,8,130,32
10,130,160,11,255,160,11
255,160,11,255,160,11,255
160,19,255,144,9,255,32
1,255,0,1,239,0,1
239,0,1,239,0,0,238
0,1,239,0,3,171,128
```



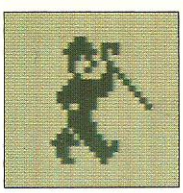
HUNCHBACK



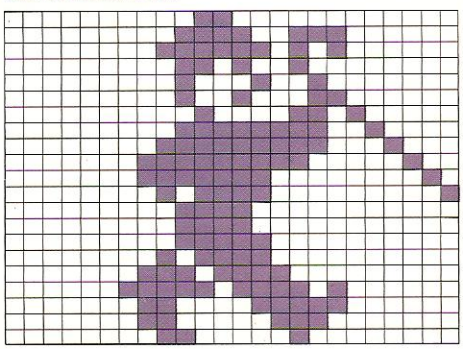
```
0,0,0,0,56,0,0
68,0,0,40,0,3,147
128,4,186,64,8,130,32
10,130,160,11,255,160,11
255,160,11,255,160,11,255
160,19,255,144,9,255,32
1,255,0,1,239,0,1
239,0,1,239,0,0,238
0,1,239,0,3,171,128
```



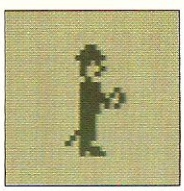
CHARLIE CHAPLIN



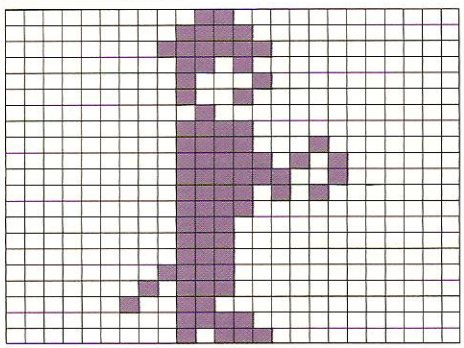
```
0,48,0,0,121,192,0
253,64,0,105,0,0,68
128,0,72,192,0,33,160
0,127,144,0,255,136,1
255,4,1,252,2,0,156
1,0,120,0,0,120,0
0,124,0,0,188,0,1
222,0,3,239,64,1,135
192,1,131,128,0,193,0
```



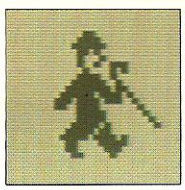
CHARLIE CHAPLIN



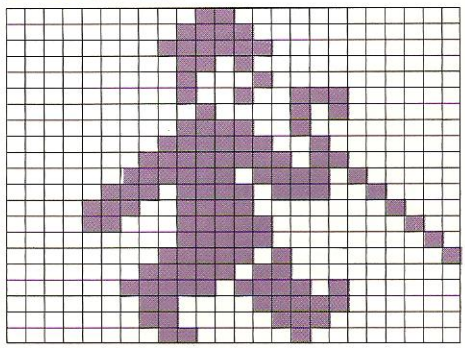
```
0,48,0,0,120,0,0
252,0,0,104,0,0,68
0,0,72,0,0,32,0
0,120,0,0,120,128,0
125,64,0,126,64,0,122
128,0,120,0,0,112,0
0,112,0,0,112,0,0
240,0,1,112,0,2,96
0,0,120,0,0,92,0
```



CHARLIE CHAPLIN

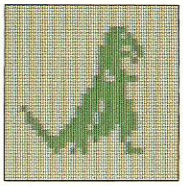


```
0,48,0,0,120,0,0
252,0,0,104,0,0,68
0,0,73,192,0,33,64
0,121,0,0,252,128,1
254,192,3,247,160,7,123
144,6,120,8,0,124,4
0,124,2,0,250,1,1
246,0,3,239,64,1,135
192,1,131,128,0,193,0
```

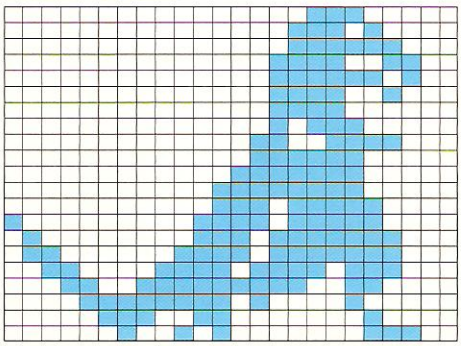


DINOSAURS

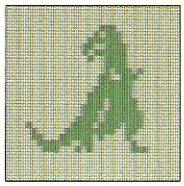
TYRANNOSAURUS



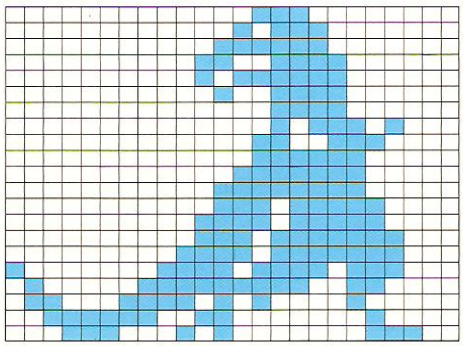
0,0,224,0,1,208,0
1,248,0,3,204,0,3
244,0,3,200,0,3,192
0,2,224,0,6,120,0
7,224,0,15,224,0,31
224,0,29,240,128,61,240
64,59,248,96,123,248,48
247,184,25,255,48,15,244
32,7,48,16,2,80,28



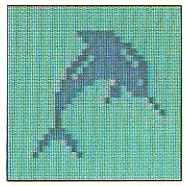
TYRANNOSAURUS



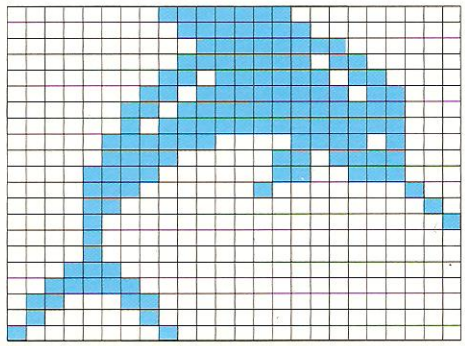
0,7,0,0,11,128,0
31,128,0,51,192,0,47
192,0,19,192,0,3,192
0,2,232,0,6,112,0
7,224,0,15,224,0,31
224,0,29,240,0,61,240
0,59,248,0,123,248,128
247,184,65,255,48,99,244
32,63,48,16,28,80,28



ICHTHYOSAURUS

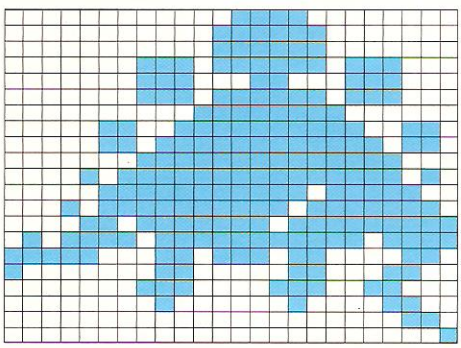


0,254,0,0,127,0,0
63,192,0,127,96,0,223
176,1,191,216,3,127,248
2,255,248,7,225,232,7
131,120,15,3,56,14,4
4,12,0,2,8,0,1
8,0,0,8,0,0,28
0,0,62,0,0,99,0
0,65,0,0,128,128,0



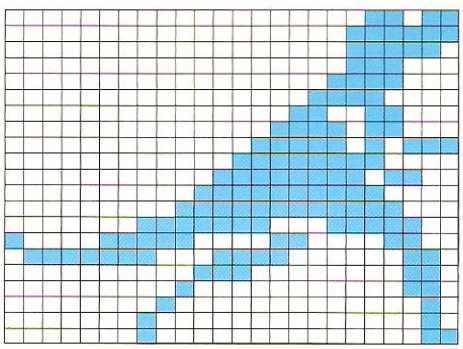
STEGOSAURUS

0,15,0,0,31,128,0
31,128,1,223,184,1,198
56,1,223,184,0,63,192
6,127,230,6,255,246,1
255,248,11,255,250,7,255
124,23,254,254,13,253,255
94,255,239,248,221,134,128
193,140,1,195,152,0,129
12,0,0,6,0,0,1



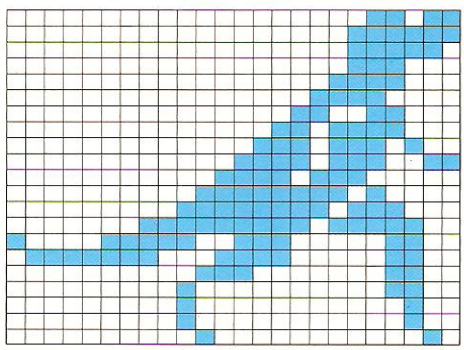
ALLOSAURUS

0,0,27,0,0,63,0
0,62,0,0,56,0,0
120,0,0,240,0,1,152
0,3,216,0,7,215,0
15,240,0,31,236,0,63
240,0,127,248,1,252,124
135,195,12,127,30,6,0
56,6,0,64,6,0,128
2,1,0,2,1,0,3



ALLOSAURUS

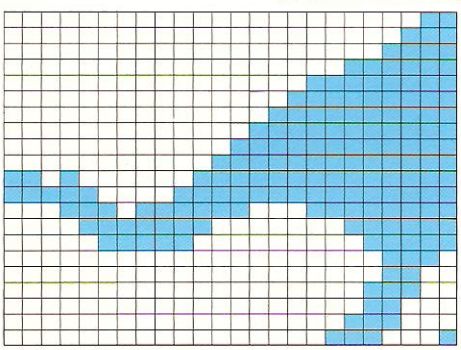
0,0,27,0,0,63,0
0,62,0,0,56,0,0
120,0,0,240,0,1,152
0,3,120,0,7,116,0
15,115,0,30,224,0,63
208,0,127,184,1,255,124
135,206,12,127,30,12,0
56,12,0,64,12,0,64
4,0,64,4,0,32,2

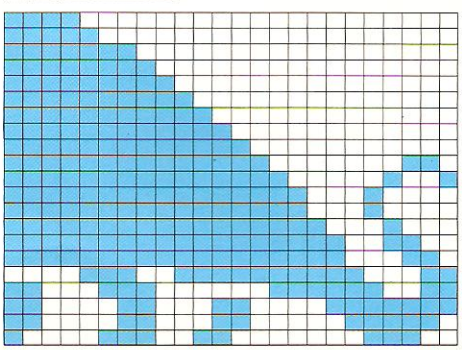


BRONTOSAURUS

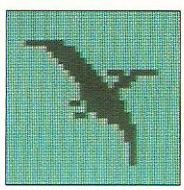
0,0,3,0,0,7,0
0,15,0,0,63,0,0
255,0,1,255,0,3,255
0,7,255,0,15,255,0
31,255,208,63,255,248,127
255,29,248,255,15,224,63
7,192,31,0,0,14,0
0,12,0,0,28,0,0
24,0,0,56,0,0,113

240,0,0,248,0,0,254
0,0,255,0,0,255,128
0,255,192,0,255,224,0
255,240,0,255,248,0,255
252,6,255,254,9,255,255
16,255,255,16,255,255,136
255,255,204,255,255,198,14
127,227,199,3,243,195,24
127,195,24,62,135,48,28

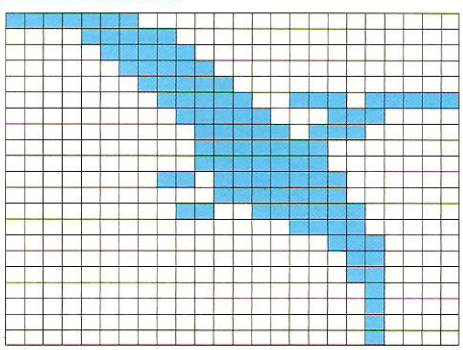




PTERANODON

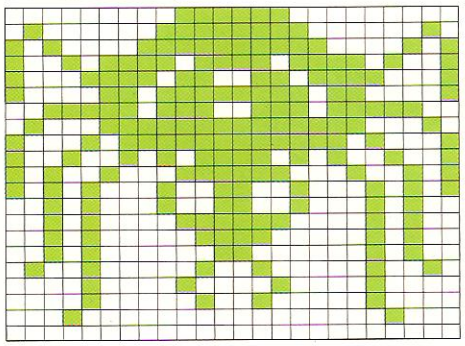


254,0,0,15,128,0,7
192,0,3,224,0,1,240
0,0,249,223,0,124,112
0,62,224,0,63,128,0
63,128,0,223,192,0,31
192,0,103,224,0,1,224
0,0,112,0,0,48,0
0,48,0,0,16,0,0
16,0,0,16,0,0,16

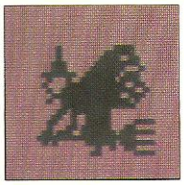


SPIDER

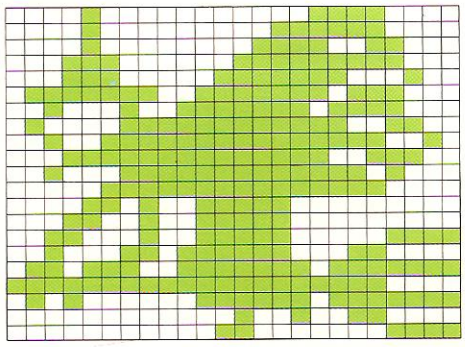
0,126,0,64,255,2,161
255,133,151,189,233,143,102
241,134,255,97,62,231,124
71,255,226,139,255,209,146
189,73,164,126,37,168,153
21,40,153,20,40,126,20
40,60,20,40,24,20,72
36,18,8,66,16,8,36
16,16,0,8,0,0,0



WITCH

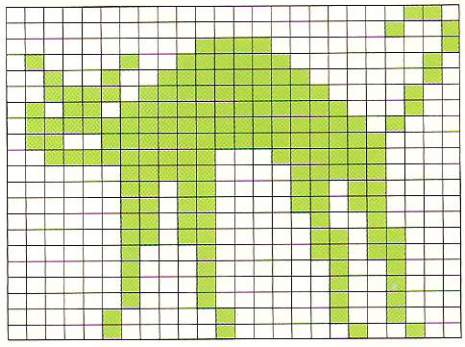


8,0,240,8,3,240,8
7,152,28,15,248,28,31
196,127,63,184,34,127,120
67,255,204,39,255,242,31
255,156,39,255,232,7,255
240,13,254,0,25,62,0
49,62,15,35,126,48,110
127,127,255 255,240,80,63
127,0,9,48,0,24,15



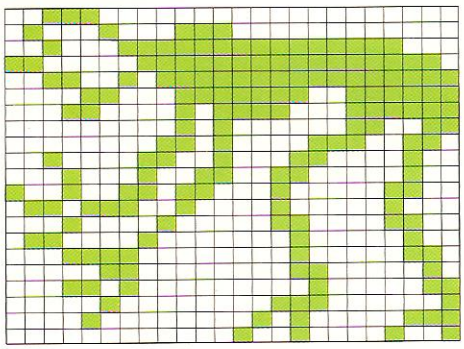
BLACK CAT

0,0,6,0,0,9,0
60,1,66,126,2,36,255
2,61,255,132,91,255,196
127,255,232,103,255,240,63
227,240,3,225,240,3,97
176,3,97,176,3,96,144
3,96,216,2,32,216,2
32,72,2,32,72,2,32
72,4,16,72,4,16,36



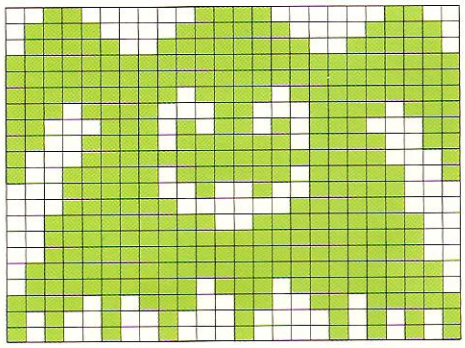
SPECTRE

56,0,0,68,112,0,35
255,248,193,255,254,50,255
255,12,127,63,24,94,63
0,80,51,0,208,230,33
145,132,17,51,4,147,98
12,110,66,8,16,131,12
97,1,4,30,1,6,6
1,130,26,0,131,4,3
129,8,5,7,0,9,9



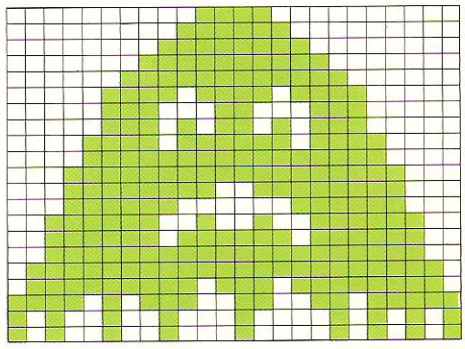
SPOOK

48,60,12,120,126,30,124
255,62,255,255,255,255,255
255,255,189,255,239,24,247
199,90,227,207,126,243,143
255,241,159,126,249,31,36
248,31,129,248,63,231,252
63,255,252,63,255,252,127
255,254,127,221,254,245,204
223,164,136,85,164,136,85



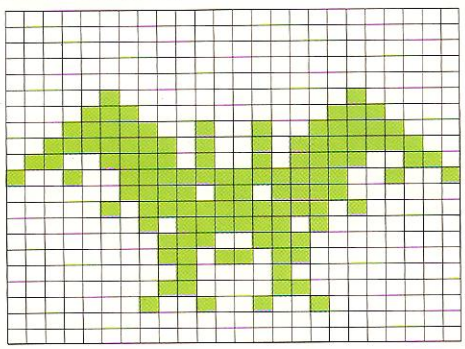
SPOOK

0,60,0,0,126,0,0
255,0,1,255,128,3,255
192,7,189,224,7,24,224
7,90,224,15,126,240,15
255,240,31,255,248,31,231
248,31,129,248,63,36,252
63,126,252,63,255,252,127
255,254,127,221,254,245,204
223,164,136,85,164,136,85



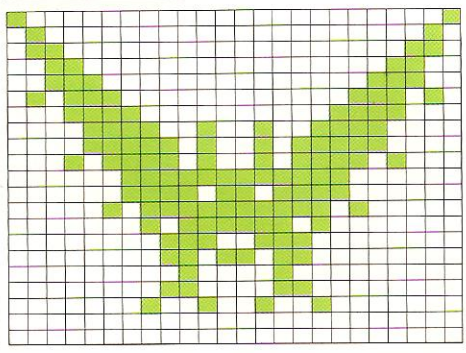
BAT

0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,4,0,32,14,0,112
31,36,248,63,165,252,103
165,230,147,255,201,3,219
192,5,255,160,1,126,128
0,231,0,0,219,0,0
66,0,0,195,0,1,36
128,0,0,0,0,0,0

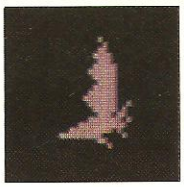


BAT

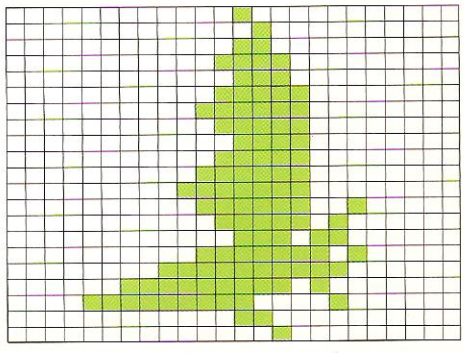
128,0,1,64,0,2,96
0,6,48,0,12,56,0
28,92,0,58,30,0,120
15,36,240,15,165,240,23
165,232,3,255,192,3,219
192,5,255,160,1,126,128
0,231,0,0,219,0,0
66,0,0,195,0,1,36
128,0,0,0,0,0,0



BAT



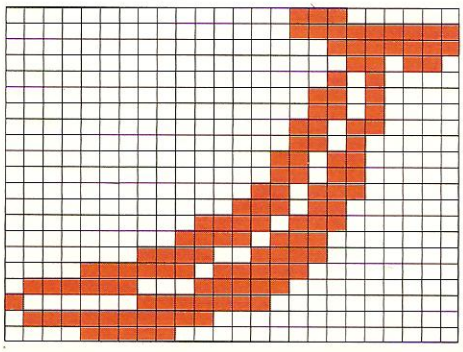
0,8,0,0,4,0,0
12,0,0,30,0,0,14
0,0,31,0,0,63,0
0,31,0,0,15,0,0
31,0,0,63,0,0,127
0,0,63,32,0,31,64
0,14,192,0,63,160,0
255,192,3,254,128,15,248
64,0,12,0,0,2,0



GAMES SYMBOLS

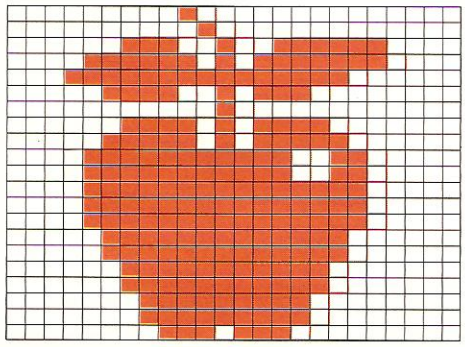
BANANA

0,1,192,0,1,255,0
0,127,0,0,54,0,0
80,0,0,208,0,0,208
0,1,176,0,1,160,0
3,96,0,3,96,0,6
224,0,14,192,0,61,192
0,123,128,1,231,128,15
223,0,126,62,0,129,252
0,127,224,0,15,128,0

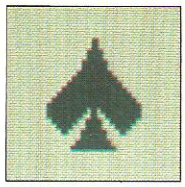


APPLE

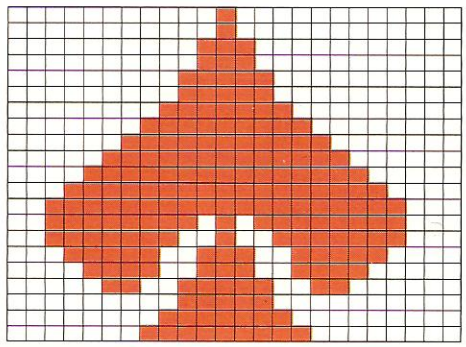
0,64,0,0,32,0,3
147,224,15,215,240,31,255
192,7,151,0,0,16,0
3,215,128,7,215,192,15
254,96,15,254,96,15,255
224,15,255,224,15,255,224
7,255,192,7,255,192,3
255,128,3,255,128,1,255
0,1,255,0,0,238,0



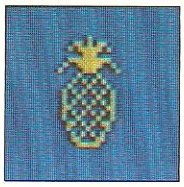
SPADE



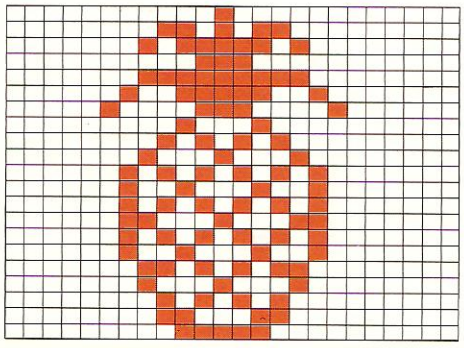
0,16,0,0,16,0,0
56,0,0,56,0,0,124
0,0,124,0,0,254,0
1,255,0,3,255,128,7
255,192,15,255,224,31,255
240,63,255,248,63,215,248
63,147,248,31,57,240,14
56,224,4,124,64,0,124
0,0,254,0,1,255,0



PINEAPPLE

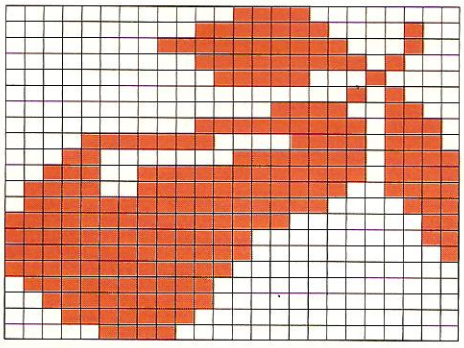


0,16,0,0,214,0,1
125,0,0,56,0,1,255
0,2,124,128,4,56,64
0,68,0,0,170,0,1
17,0,2,170,128,2,68
128,2,170,128,3,17,128
2,170,128,2,68,128,1
171,0,1,17,0,0,170
0,0,198,0,0,124,0

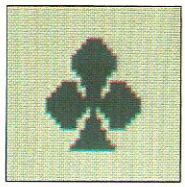


PEAR

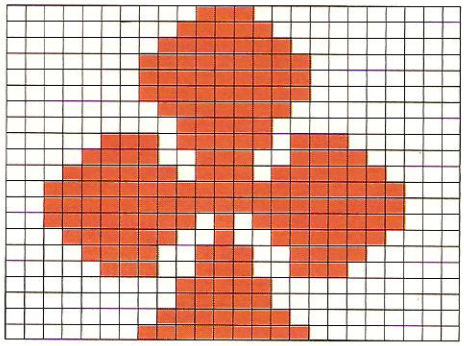
0,15,0,0,63,132,0
255,196,0,63,232,0,31
16,0,0,40,0,3,204
0,63,238,15,225,238,24
15,239,57,255,239,121,255
207,127,255,135,255,254,3
255,248,3,255,248,1,255
240,0,127,224,0,63,224
0,31,192,0,7,128,0



CLUB

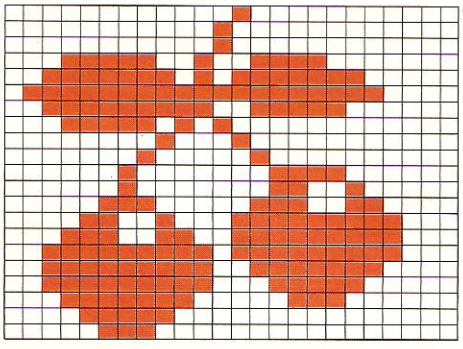


0,56,0,0,124,0,0
254,0,1,255,0,1,255
0,1,255,0,0,254,0
0,124,0,7,125,192,15
57,224,31,187,240,63,255
248,63,255,248,63,215,248
31,147,240,15,57,224,7
57,192,0,124,0,0,124
0,0,254,0,1,255,0

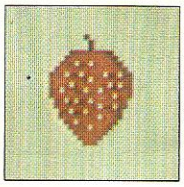


CHERRIES

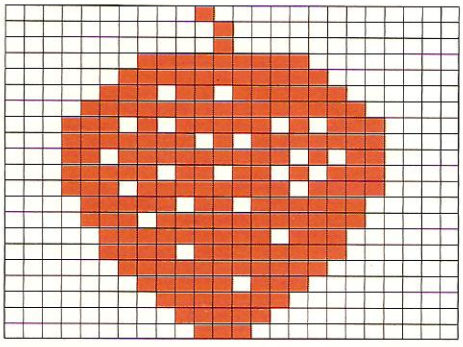
0,8,0,0,16,0,0
16,0,31,39,128,63,175
224,127,255,240,63,167,192
31,80,0,0,136,0,1
4,0,2,3,192,2,3
32,7,7,48,12,143,248
28,207,248,63,239,248,63
231,240,63,227,224,31,193
192,15,128,0,7,0,0



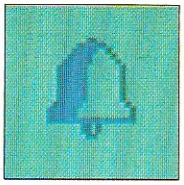
STRAWBERRY



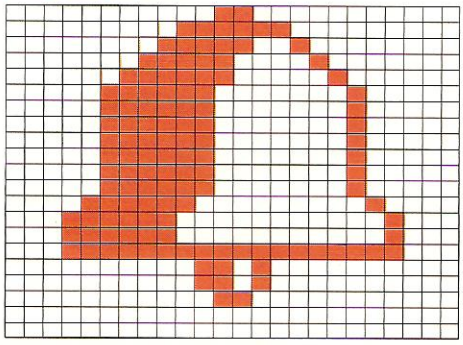
0,32,0,0,16,0,0
16,0,1,255,0,3,255
128,7,111,192,15,255,224
29,189,176,31,239,240,30
254,176,27,219,240,31,255
112,13,111,224,15,251,96
6,223,192,7,251,192,3
127,128,1,239,0,0,254
0,0,124,0,0,56,0



BELL

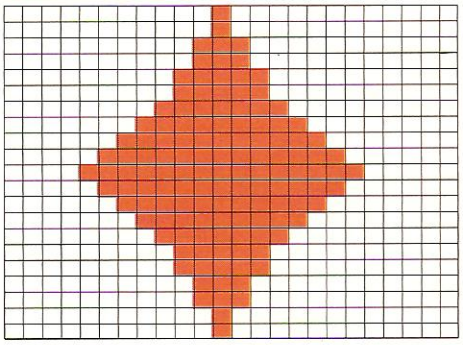


0,24,0,0,126,0,0
249,0,1,240,128,3,240
64,7,224,32,7,224,32
7,224,32,7,224,32,7
224,32,7,224,32,7,224
32,15,192,16,31,128,8
31,128,8,31,255,248,0
52,0,0,52,0,0,24
0,0,0,0,0,0,0



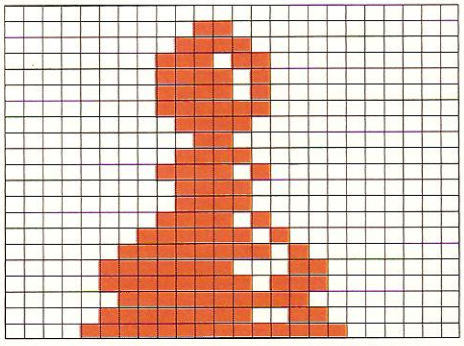
DIAMOND

```
0,16,0,0,16,0,0
56,0,0,56,0,0,124
0,0,124,0,0,254,0
1,255,0,3,255,128,7
255,192,15,255,224,7,255
192,3,255,128,1,255,0
0,254,0,0,124,0,0
124,0,0,56,0,0,56
0,0,16,0,0,16,0
```



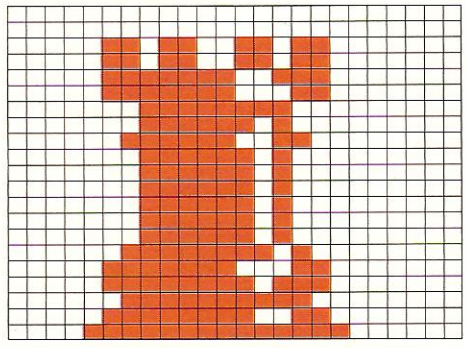
PAWN

```
0,0,0,0,48,0,0
120,0,0,236,0,0,244
0,0,244,0,0,252,0
0,120,0,0,48,0,0
120,0,0,244,0,0,120
0,0,120,0,0,244,0
1,250,0,3,253,0,7
241,128,7,251,128,3,255
0,7,249,128,15,255,192
```



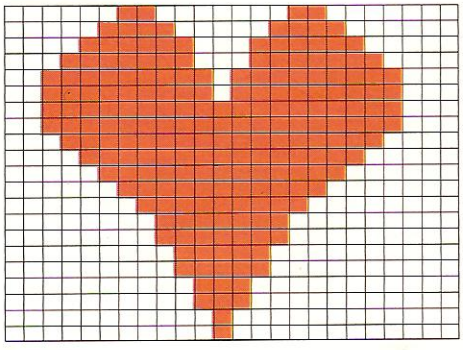
ROOK

```
0,0,0,0,0,0,6
205,128,6,205,128,7,243
128,7,241,128,1,254,0
1,250,0,3,243,0,1
250,0,1,250,0,1,250
0,1,250,0,1,250,0
1,250,0,3,253,0,7
241,128,7,251,128,3,255
0,7,249,128,15,255,192
```



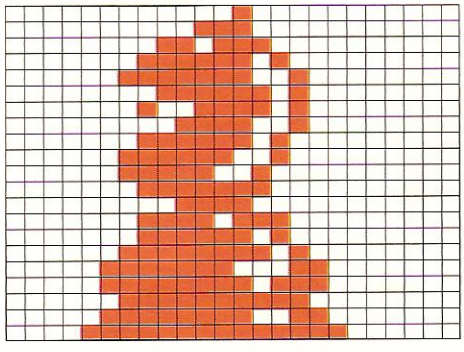
HEART

```
3,1,128,7,131,192,15
199,224,31,199,240,63,239
248,63,239,248,63,255,248
63,255,248,31,255,240,15
255,224,7,255,192,3,255
128,1,255,0,0,254,0
0,254,0,0,124,0,0
124,0,0,56,0,0,56
0,0,16,0,0,16,0
```



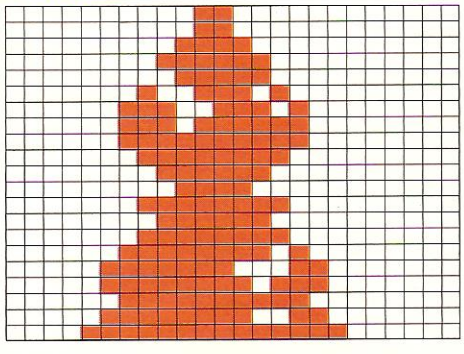
KNIGHT

```
0,8,0,0,56,0,0
220,0,1,250,0,3,249
0,0,253,0,1,61,0
0,121,0,1,250,0,3
242,0,3,228,0,1,252
0,0,120,0,1,238,0
1,250,0,3,253,0,7
241,128,7,251,128,3,255
0,7,249,128,15,255,192
```



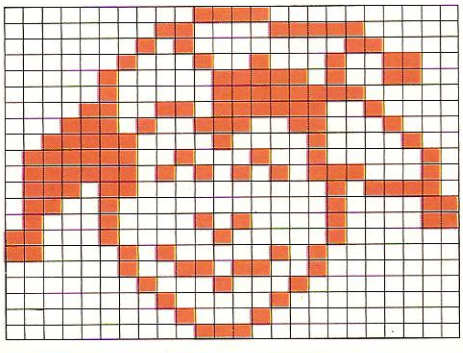
BISHOP

```
0,48,0,0,48,0,0
120,0,0,252,0,0,124
0,1,58,0,3,157,0
3,191,0,3,255,0,1
254,0,0,252,0,0,120
0,1,254,0,0,252,0
1,254,0,3,253,0,7
241,128,7,251,128,3,255
0,7,249,128,15,255,192
```



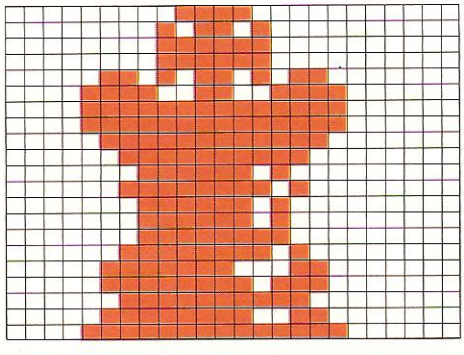
JOKER

```
0,60,0,0,195,224,1
64,48,2,36,76,4,31
76,4,31,224,12,223,144
29,57,136,62,16,132,126
68,130,126,170,226,118,0
94,100,0,67,68,40,67
196,16,64,194,130,128,2
108,128,1,17,0,0,130
0,0,68,0,0,56,0
```

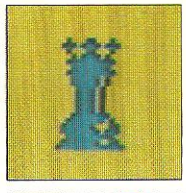


KING

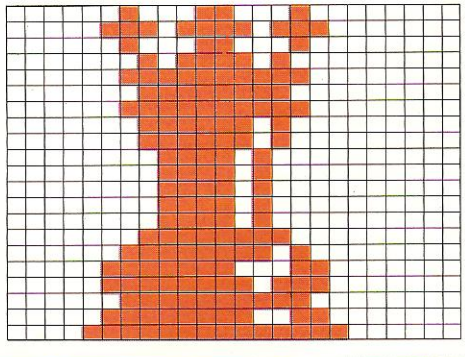
```
0,120,0,0,180,0,0
252,0,0,252,0,6,181
128,15,51,192,15,255,192
15,255,192,7,255,128,3
255,0,1,250,0,3,253
0,1,250,0,1,250,0
1,250,0,3,253,0,7
241,128,7,251,128,3,255
0,7,249,128,15,255,192
```



QUEEN



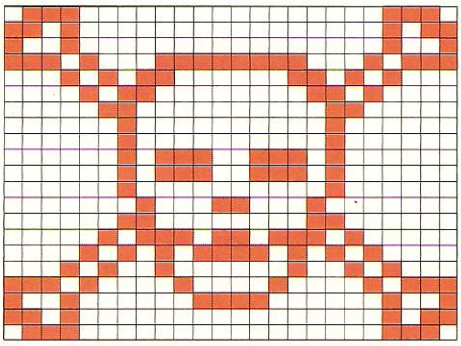
```
2,49,0,7,123,128,2
49,0,1,122,0,3,255
0,1,254,0,3,255,0
1,250,0,1,250,0,0
244,0,0,244,0,0,244
0,0,244,0,0,244,0
1,254,0,3,253,0,7
241,128,7,251,128,3,255
0,7,249,128,15,255,192
```



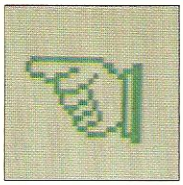
GAMES SYMBOLS

SKULL AND CROSSBONES

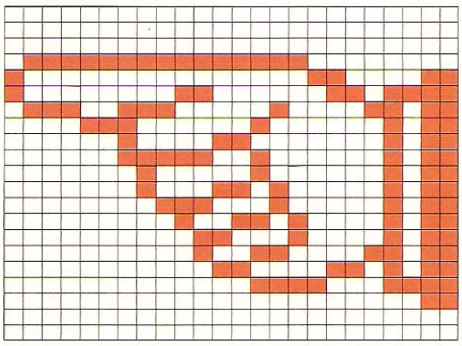
```
112,0,14,208,0,11,144
0,9,232,255,23,21,129
168,11,0,208,6,0,96
2,0,64,2,0,64,2
231,64,2,231,64,2,0
64,1,24,128,3,0,192
5,219,160,10,189,80,20
129,40,232,66,23,144,60
9,208,0,11,112,0,14
```



POINTING HAND

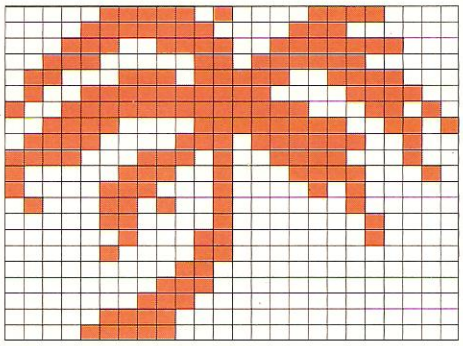


```
0,0,0,0,0,0,0
0,0,127,255,0,128,0
195,128,96,111,115,130,27
14,4,11,2,24,11,3
228,11,1,4,11,1,26
11,0,226,11,0,77,11
0,49,11,0,38,27,0
24,107,0,7,143,0,0
3,0,0,0,0,0,0
```



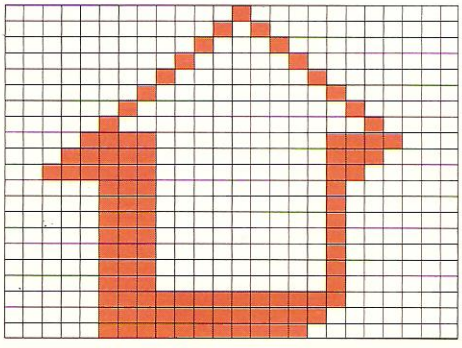
PALM TREE

```
7,208,240,15,225,192,28
243,248,56,55,224,99,190
120,79,252,60,159,255,26
60,63,205,120,119,228,113
211,196,195,145,226,135,16
160,134,144,32,70,16,16
2,48,16,4,48,0,0
96,0,0,224,0,1,192
0,7,128,0,15,128,0
```



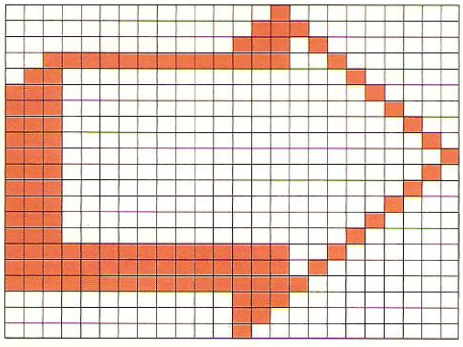
ARROW

```
0,8,0,0,20,0,0
34,0,0,65,0,0,128
128,1,0,64,2,0,32
4,0,16,15,0,120,31
0,112,63,0,96,7,0
64,7,0,64,7,0,64
7,0,64,7,0,64,7
0,64,7,0,64,7,255
192,7,255,128,7,255,0
```

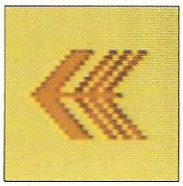


ARROW

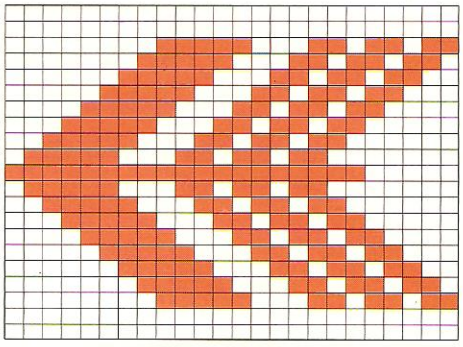
```
0,2,0,0,7,0,0
14,128,63,254,64,96,0
32,224,0,16,224,0,8
224,0,4,224,0,2,224
0,1,224,0,2,224,0
4,224,0,8,224,0,16
224,0,32,255,254,64,255
254,128,255,255,0,0,14
0,0,12,0,0,8,0
```



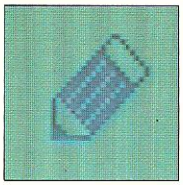
ARROW



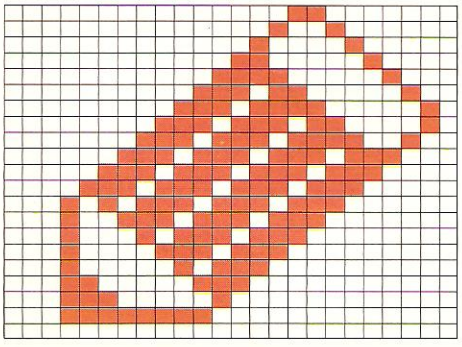
```
0,0,0,0,0,0,0
248,219,1,241,182,3,227
108,7,198,216,15,141,176
31,27,96,62,54,192,124
109,128,255,255,224,124,109
128,62,54,192,31,27,96
15,141,176,7,198,216,3
227,108,1,241,182,0,248
219,0,0,0,0,0,0
```



PENCIL

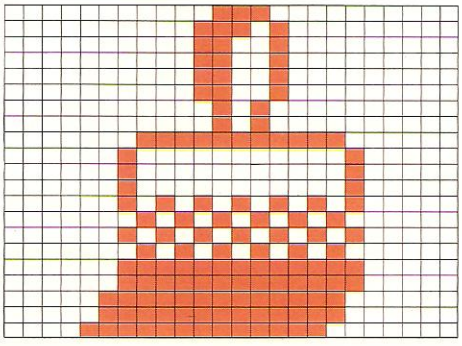


```
0,1,128,0,2,64,0
4,32,0,12,16,0,30
8,0,59,4,0,119,130
0,238,194,1,221,228,3
187,184,7,119,112,14,238
224,21,221,192,19,187,128
17,119,0,16,238,0,16
92,0,24,56,0,28,16
0,31,224,0,0,0,0
```



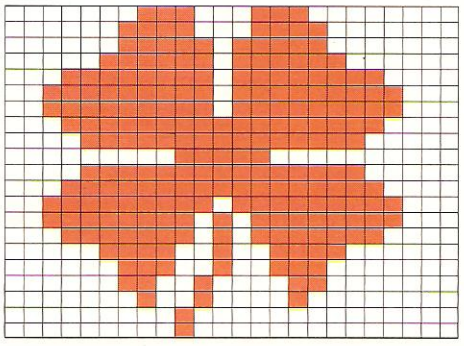
PAINTBRUSH

```
0,28,0,0,58,0,0
50,0,0,50,0,0,50
0,0,50,0,0,20,0
0,20,0,1,255,192,2
0,32,2,0,32,2,0
32,2,170,160,1,85,64
2,170,160,1,85,64,3
255,224,3,255,224,7,255
192,7,255,192,15,255,128
```



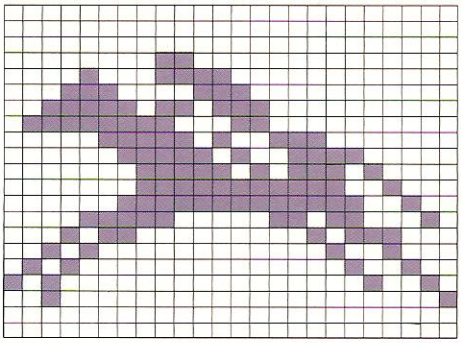
SHAMROCK

```
1,199,0,3,199,128,3
239,128,7,239,192,31,239
240,63,239,248,63,239,248
31,255,240,15,255,224,0
124,0,15,255,224,31,255
240,63,239,248,63,215,248
31,215,240,7,147,192,3
147,128,3,33,128,1,33
0,0,64,0,0,64,0
```



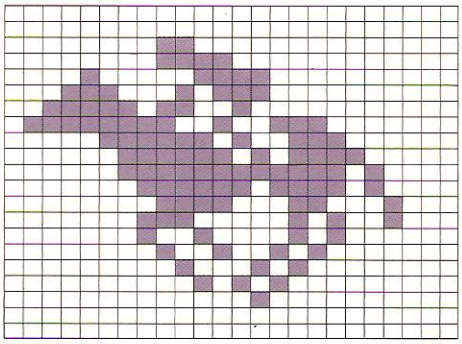
HORSE AND JOCKEY

0,0,0,0,0,0,0
0,0,0,192,0,8,240
0,28,56,0,62,220,0
127,140,0,7,211,128,3
233,224,1,245,208,1,255
200,3,255,228,15,131,98
22,0,184,40,0,72,80
0,36,160,0,18,32,0
1,0,0,0,0,0,0



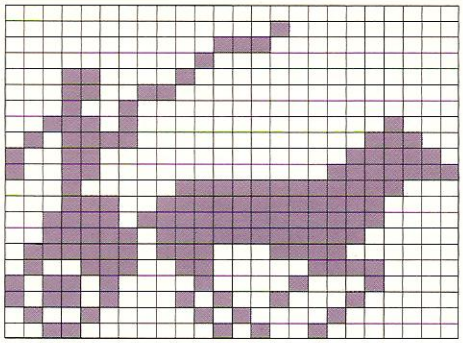
HORSE AND JOCKEY

0,0,0,0,0,0,0
192,0,0,240,0,8,60
0,28,92,0,62,200,0
127,147,128,7,233,192,3
245,224,3,255,208,1,251
208,0,241,136,1,193,72
1,33,64,0,146,128,0
69,0,0,42,0,0,4
0,0,0,0,0,0,0



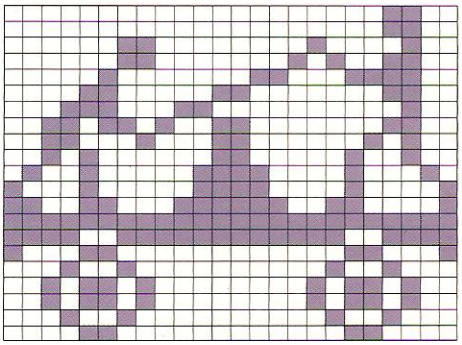
CHARIOT

0,0,0,0,2,0,0
28,0,0,32,0,24,64
0,25,128,0,18,0,0
58,0,8,94,0,28,152
0,62,152,0,127,16,127
248,28,255,240,61,255,240
62,255,224,126,227,192,78
96,240,180,33,16,180,82
32,72,40,64,48,20,128



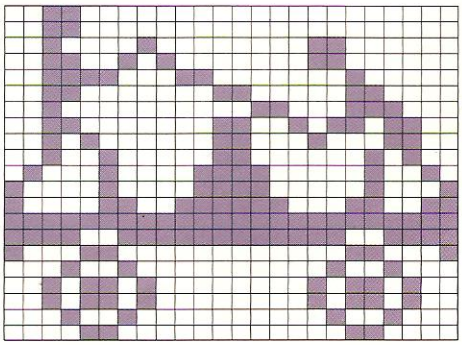
TROLLEY

0,0,12,0,0,12,3
0,132,3,1,76,4,6
60,12,24,12,28,96,4
54,152,12,49,24,20,40
24,36,72,60,34,136,60
33,204,126,97,255,255,255
243,255,207,140,0,49,18
0,72,45,0,180,45,0
180,18,0,72,12,0,48



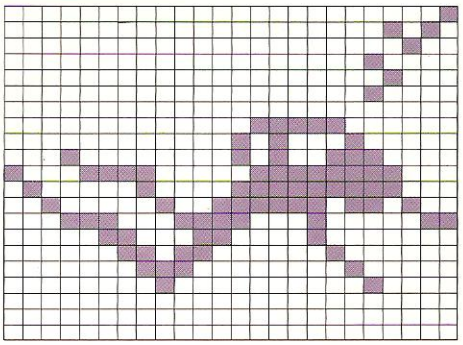
TROLLEY

48,0,0,48,0,0,33
0,192,50,128,192,60,96
32,48,24,48,32,6,56
48,25,108,40,24,140,36
24,20,68,60,18,132,60
17,134,126,51,255,255,255
243,255,207,140,0,49,18
0,72,45,0,180,45,0
180,18,0,72,12,0,48



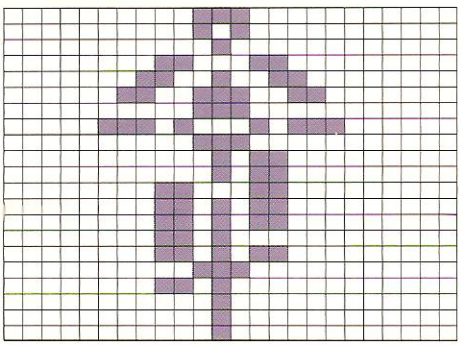
FROGMAN

0,0,1,0,0,10,0
0,4,0,0,16,0,0
8,0,0,16,0,0,0
0,7,192,0,10,32,16
10,112,143,7,248,65,15
248,32,159,180,28,120,131
6,48,128,3,96,64,1
192,32,0,128,16,0,0
0,0,0,0,0,0,0



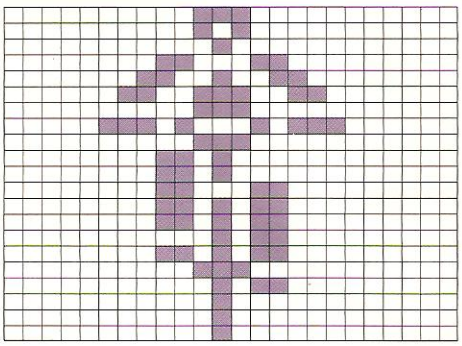
BMX RIDER

0,56,0,0,40,0,0
16,0,0,198,0,1,147
0,3,57,128,0,56,0
7,69,192,0,56,0,0
22,0,0,22,0,0,198
0,0,214,0,0,214,0
0,208,0,0,214,0,0
56,0,0,208,0,0,16
0,0,16,0,0,16,0

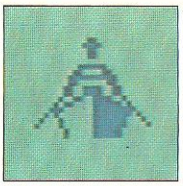


BMX RIDER

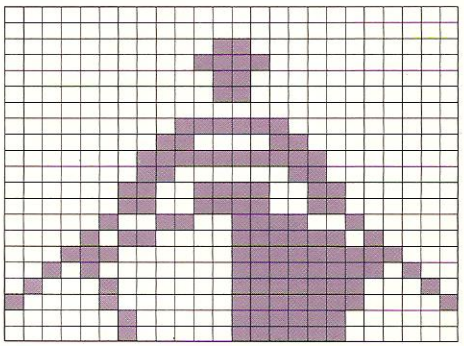
0,56,0,0,40,0,0
16,0,0,198,0,1,147
0,3,57,128,0,56,0
7,69,192,0,56,0,0
208,0,0,208,0,0,198
0,0,214,0,0,214,0
0,22,0,0,214,0,0
56,0,0,22,0,0,16
0,0,16,0,0,16,0



MAN IN BOAT



0,0,0,0,0,0,0
24,0,0,60,0,0,24
0,0,24,0,0,0,0
0,126,0,0,195,0,1
255,128,1,129,128,3,60
192,3,60,192,4,207,32
11,15,208,28,15,248,40
15,244,68,15,226,132,15
225,2,15,192,2,15,192



ROUTINES CHECKLIST

The table shown below gives a summary of all the machine-code routines used in this book. This table does not explain every detail of using each routine; it is intended only as an aid when using the routines in your programs. If you have not used a routine before, it is recommended that you read the introduction to the routine on the appropriate page of the book before using it in your program.

page	routine	parameters			co-ordinates
	sprite buffer				
13	24 x 21 sprite editor	Fna()–e()			character
15	sprite print	FNf(x,y,n)	x,y	print position	–
16	master sprite	– – –			
17	sprite handling	FNg(x,y,d,l,s,c,n)	x,y	start co-ordinates	pixel
			d	direction	–
			l	distance to be moved	pixel x 3
			s	switch	–
			c	collision flag	–
			n	sprite number	–
21	keyboard-controlled sprite	FNh(s,x,y,c,n)	s	switch	–
			x,y	start co-ordinates	pixel
			c	collision flag	–
			n	sprite number	–
	interrupt vector table				
22	double horizontal sprite	FNi(x,y,d,l,s,c,n)	x,y	start co-ordinates	pixel
			d	direction	–
			l	distance to be moved	pixel x 3
			s	switch	–
			c	collision flag	–
			n	sprite number	–
23	double vertical sprite	FNj(x,y,d,l,s,c,n)	x,y	start co-ordinates	pixel
			d	direction	–
			l	distance to be moved	pixel x 3
			s	switch	–
			c	collision flag	–
			n	sprite number	–
25	sprite animation	FNk(x,y,d,l,s,f,c,v,n)	x,y	start co-ordinates	pixel
			d	direction	–
			l	distance to be moved	pixel x 3
			s	switch	–
			f	number of frames	–
			c	collision flag	–
			v	animation speed	–
			n	sprite number	–
29	horizontal scroll	FN1(l,d)	l	length of scroll	pixel
			d	direction	–
29	vertical scroll	FNm(l,d)	l	length of scroll	pixel
			d	direction	–
31	window	FNn(x,y,l,n,d,r)	x,y	start co-ordinates	character
			l	width of window	character
			n	sprite number	–
			d	direction	–
			r	repeat flag	–
33	interrupt-driven window	FNo(s,x,y,l,n,d,r)	s	switch	–
			x,y	start co-ordinates	character
			l	width of window	character
			n	sprite number	–
			d	direction	–
			r	repeat flag	–

Before using a routine, you must first define it in your program using DEF FN followed by the correct number of parameters. Parameters passed to machine-code routines must always be whole numbers; if a parameter value is calculated by your program, then put an INT statement in front of it to ensure a whole-number value is passed to the routine.

ranges	bytes	address	check
	700	54600	
	355	54200	190
0-28 and 0-20	75	54100	60
1-10	365	53700	83
0-231 and 0-154 0-3 0–51 (vertical) 0-77 (horizontal) 0-1 0-1 1-10	170	53500	66
0-1 0-231 and 0-154 0-1 1-10	250	53100	26
	256	52736	
0-231 and 0-154 0-3 0-51 (vertical) 0-77 (horizontal) 0-1 0-1 1-10	235	52400	53
0-231 and 0-154 0-3 0-51 (vertical) 0-77 (horizontal) 0-1 0-1 1-10	230	52100	20
0-231 and 0-154 0-3 0-51 (vertical) 0-77 (horizontal) 0-1 1-10 0-1 1-255 1-10	275	51700	63
0-255 0-1	190	51500	61
0-175 0-1	215	50900	47
0-31 and 0-21 0-31 1-10 0-1 0-1	290	49600	43
0-1 0-31 and 0-21 0-31 1-10 0-1 0-1	315	49200	154

MEMORY MAP

This chart shows how the Spectrum memory is organized when all the routines are present in memory. RAMTOP can be set to 49000 using a CLEAR command.

routine	title	address
	lowest book 3 routine	55500
	sprite buffer (700 bytes)	54600
FNa – FNe	24 x 21 sprite editor	54200
FNf	sprite print	54100
	master sprite	53700
FNg	sprite-handling	53500
FNh	keyboard-controlled sprite	53100
	interrupt vector table	52736
FNi	double horizontal sprite	52400
FNj	double vertical sprite	52100
FNk	sprite animation	51700
FNl	horizontal scroll	51500
FNm	vertical scroll	50900
FNn	window	49600
FNo	interrupt-driven window	49200
	RAMTOP (after CLEAR 49000)	49000

INDEX

Main entries are given in **bold** type

Acknowledgments
A number of people helped and encouraged me with this book. Thanks to Alan and Michael at Dorling Kindersley, to Jacqui Lyons for her representation and to Andy Werbinski for reluctant assistance. I am particularly grateful, as always, to my parents, and to Martine.

Piers Letcher
Spring 1985

DK

ScreenShot

PROGRAMMING SERIES

The bestselling teach-yourself programming course now offers the first complete full-colour book on creating sprites on the ZX Spectrum.

Illustrated with over 300 screen-shot photographs, it contains programs for single and double sprites, for animation, and for setting overlaps and detecting collisions, and includes an easy-to-use sprite generator with which you can design and save your own sprites. In addition, there is a full-colour design directory containing over 200 original sprite designs complete with all the data needed to program them.

Together, Books Three and Four in this series form a complete, self-contained graphics system for Spectrum-owners.

All the programs in this book run on both 48K ZX Spectrum and ZX Spectrum+ machines.

"Far better than anything else reviewed on these pages . . . Outstandingly good"
BIG K

"As good as anything else that is available, and far better than most"
COMPUTING TODAY

"Excellent . . . As a series they could form the best 'basic introduction' to programming I've seen"
POPULAR COMPUTING WEEKLY

DORLING KINDERSLEY

ISBN 0-86318-104-X

£5.95

9 780863 181047