DK

Screen Shot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

COMMODORE 64

GRAPHICS

PHIL CORNES

BOOK FOUR

Advanced sprite programming techniques plus a full colour design director

DK

Screen Shot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

COMMODORE 64 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 Screen-Shot style, the series continues to set new standards in the world of computer books.

BOOKS ABOUT THE COMMODORE 64

This is Book Four in a series of guides to programming the Commodore 64. It contains a complete sprite-programming course for the Commodore, 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 **ZX Spectrum+**

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

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

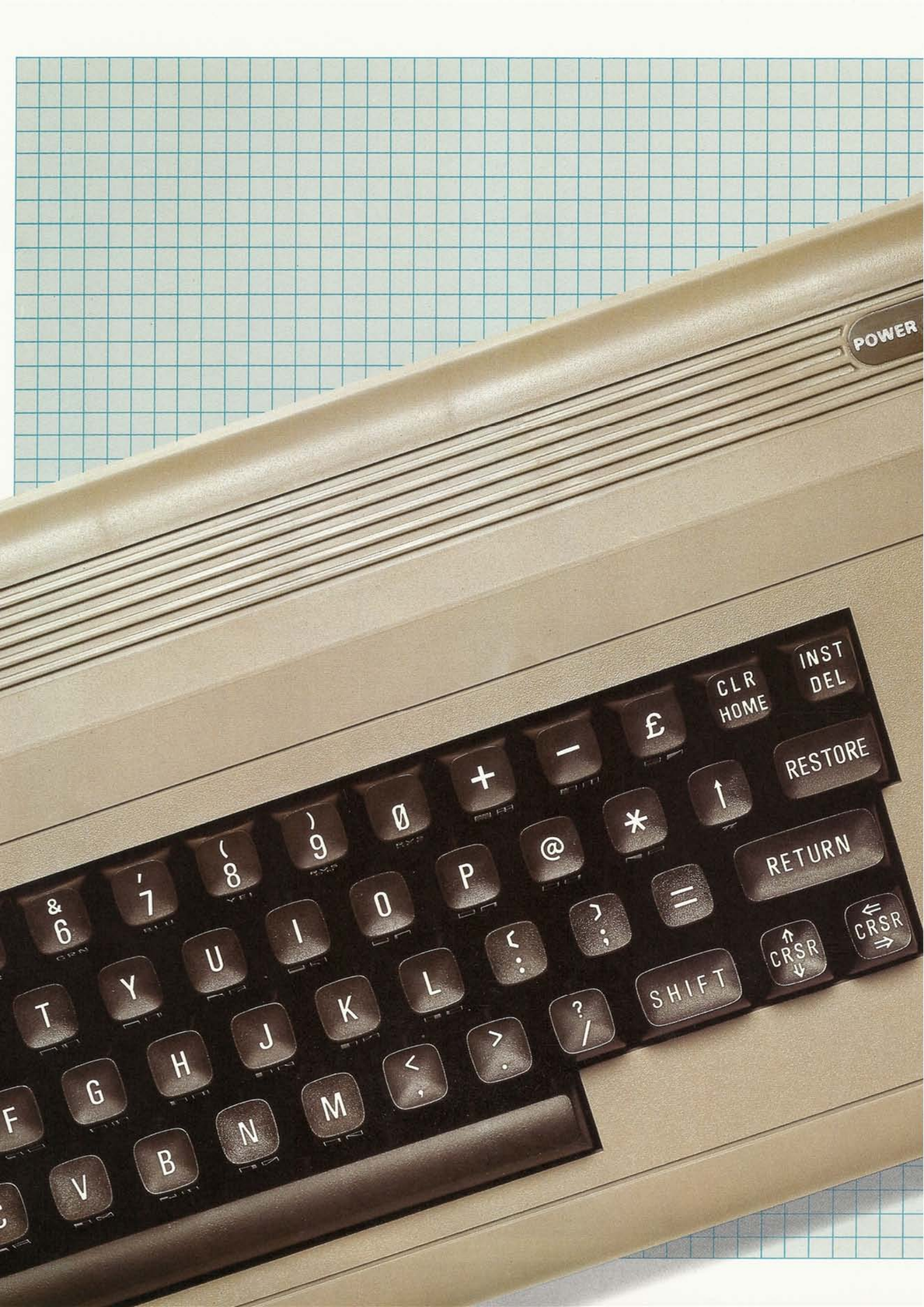
Step-by-Step Programming for the **Apple IIe**

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

PHIL CORNES

After taking a B.A. in Mathematics and Computing, Phil Cornes has been involved in system development of computer-based education at British Telecom's National Training College. He has been a part-time technical author since 1978, and has become a regular contributor to personal computer magazines such as *Personal Computer World, Computing Today* and *Electronics Today International*. He has written a book and a large number of articles on programming and using the Commodore 64.

BOOK FOUR


POWER
CLR HOME
INST DEL
RESTORE
RETURN
SHIFT
CRSR
CRSR

DK

Screen Shot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

COMMODORE 64 GRAPHICS

PHIL CORNES

GUILD PUBLISHING · LONDON

BOOK FOU

CONTENTS

The DK Screen-Shot Programming Series was conceived, edited and designed by Dorling Kindersley Limited.

Designer Hugh Schermuly
Photographer Vincent Oliver
Series Editor David Burnie
Series Art Editor Peter Luff
Managing Editor Alan Buckingham

Copyright © 1985 by Dorling Kindersley Limited, London

This edition published 1985 by Book Club Associates by arrangement with Dorling Kindersley Limited.

The term Commodore is a trade mark of Commodore Business Machines, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying recording, or otherwise, without the prior written permission of the copyright owner.

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

INTRODUCING SPRITES

Sprites are blocks of pixels that have a very special character. They move smoothly over the screen giving superb animation, and they can be stretched, overlapped or collided with each other. Most importantly, they can be displayed and moved either independently of anything else already on the screen, or they can be programmed to interact in different ways with what is already there.

The Commodore 64 is particularly good at producing sprites, allowing you to have up to eight on the screen at once. You can have up to 32 separate sprite designs simultaneously in memory, each of which can be called up onto the screen in a split-second.

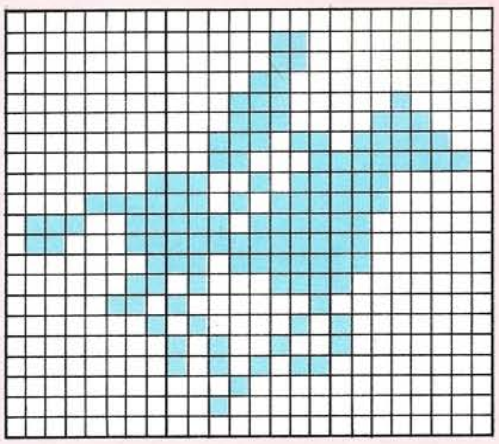
If you have read Books One and Two in this series, you will already know something about how to program sprites, and what they can do. If you have not, you will find all this information and much more in the following pages. This book shows you how to make the most of Commodore sprites. In it you will find out not only all about programming sprites, but also you will find a special sprite editor, which does most of the hard work for you. To help you produce the best designs, the second half of the book consists of a directory of sprites giving you over 200 ideas for sprite shapes.

However, before you launch into sprite programming, you will probably need to know a little more about how sprites are made up.

How sprites are made up and controlled

A sprite is a block of 504 pixels arranged in 21 rows each 24 pixels wide. You can see a typical sprite design below, as you would draw it up before incorporating it into a program. When it is displayed on the screen, all its characteristics—its shape, color and position—will be controlled by a single chip inside the Commodore, the Video Interface Circuit (VIC).

DESIGNING A SPRITE



Inside the VIC chip there are 47 special internal memory locations called registers. Of these, 34 are used to control all the actions of sprites. The numbers in these registers are specified with the POKE command. The key to successful sprite programming is understanding what numbers to POKE into which registers.

To program sprites, you need to give the computer two sets of instructions. Firstly, you need to set aside a section of memory for your sprites and then enter the sprite information to be stored there. Secondly, you need to retrieve the sprites from memory and put them on the screen. On these two pages you will find out how to complete the first part.

You can see how to set aside an area of memory in the panel below. The direct commands shown on the screen set aside a 16K block of memory from location 0 to 16383 for sprites, and move the BASIC storage area, which normally uses this part of memory, elsewhere.

IMPORTANT

The Commodore needs some special instructions before you can use the sprite programs in this book. After you turn on the computer you *must* key in these commands:

SPRITE MEMORY AREA COMMANDS

```
POKE 642,64
READY.
POKE 44,64
READY.
POKE 16384,0
READY.
NEW
READY.
```

The commands reorganize the Commodore's memory. There is no area permanently set aside in the Commodore for sprites or high-resolution graphics, so you need to tell the computer where to reserve space for them. The VIC chip can be switched so that it uses any one of four separate 16K areas available within the 64K RAM. The commands above make it use the first of these areas. **The programs in this book will not work if you forget to key in these commands.**

The commands must not be typed in as part of a program. If you try to enter them as a program, it may destroy itself.

Sprites and machine-code graphics

Although some of the programs in this book produce sprites with low-resolution (text mode) backgrounds, others feature complex high-resolution backgrounds instead. To create these backgrounds quickly, these programs use the machine-code graphics routines featured in Book Three in this series. If you want to run these programs, it is essential that you have the required machine-code in memory.

You can do this either by loading a copy of the routines from Book Three, or, if you haven't read Book Three, by following the instructions and keying in the machine-code routine listings shown on pages 60-61. The text with every program will state if any machine-code routines are needed.

Coding a sprite

After you have sketched out a sprite, you need to translate it into a sequence of numbers that the VIC chip can understand. For each sprite the computer has to store a pattern of 504 pixels, each either on or off. Information in the Commodore is stored in bytes—binary numbers — consisting of 8 separate bits (digits that are either 0 or 1). This means that, using one bit to specify whether a single pixel is on or off, each row of 24 pixels can be stored in 24 bits, or three bytes. The 21 rows in a sprite therefore need 21x3 or 63 bytes. However, because computers normally carry out all calculations in binary arithmetic which is based on powers of 2, it is simplest to add a 64th byte. This is included as padding, and is not actually used.

HOW A SPRITE IS CODED

Each horizontal 8-pixel block of a sprite is coded by a single byte. A whole 504-pixel sprite is coded by 63 bytes. A 64th byte, set to 0, is usually added to make the total a power of two.

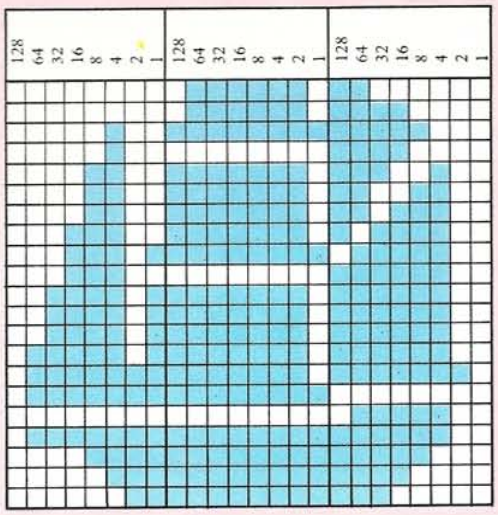
Byte	Byte	Byte
0	1	2
3	4	5
6	7	8
9	10	11
12	13	14
15	16	17
18	19	20
21	22	23
24	25	26
27	28	29
30	31	32
33	34	35
36	37	38
39	40	41
42	43	44
45	46	47
48	49	50
51	52	53
54	55	56
57	58	59
60	61	62

How to store sprites

The best way to store the pixel information for a sprite is as a series of DATA statements. What you do is to add together the "bit values" of each lit pixel in each 8-pixel row. These bit values increase in powers of 2 from the right-hand side of the row. The right-hand pixel has a value of 1 if lit, while the left-hand pixel has a value of 128 if lit. Unlit pixels all have a value of 0. Adding eight of these values together gives you a decimal number from 0-255, one of the 63 numbers used as DATA. The 64th DATA number, which has no effect, is set to 0.

Below you can see a sprite and the DATA that defines it. The program contains a loop which POKEs the 64 DATA numbers into memory, starting at location 2048 (you can find out how this location is specified on page 9). Once you know how to compile sprite DATA like this, you can then set about making the Commodore retrieve this information from memory and transform it into a sprite on the screen.

STORING A SPRITE



Bit values

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

SPRITE DATA PROGRAM

```
LIST

10 FOR C=0 TO 63
20 READ BYTE
30 POKE 2048+C,BYTE
40 NEXT C
500 DATA 0,126,192,0,126,240,4
510 DATA 255,248,4,0,240,12,254
520 DATA 228,12,254,204,12,254,220
530 DATA 28,254,188,29,255,124,28
540 DATA 0,252,61,254,252,61,254
550 DATA 252,61,254,252,125,254,252
560 DATA 125,254,254,127,255,0,0
570 DATA 0,124,127,255,252,15,255
580 DATA 248,7,255,240,3,255,224
590 DATA 0
READY.
```

SPRITE PROGRAMMING

When you program the computer to display a sprite, you must give it four types of instruction. You must turn on the sprite, you must tell it where to put the sprite on the screen and what color or colors to display it in, and finally you must tell it where to find the sprite's DATA.

The order doesn't really matter. Because all these instructions feature registers in the VIC chip, it is easiest to refer to them all in a shorthand way by letting a variable V represent the first register in the chip.

Turning sprites on and off

All eight sprites can be turned on or off by the separate bits that make up the byte in register V+21. Bit 0 controls sprite 0, bit 1 sprite 1, and so on up to bit 7 which controls sprite 7. Each bit that is set to 1 within the V+21 byte turns its respective sprite on, and each bit set to 0 turns its respective sprite off.

How to position sprites

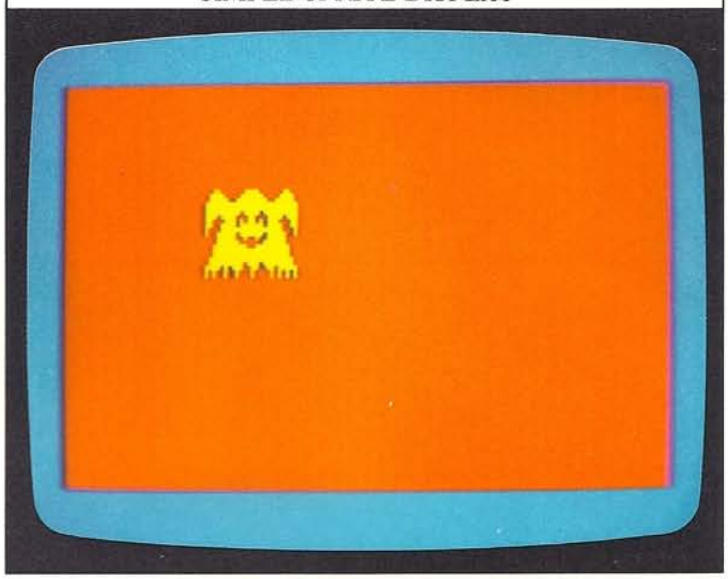
To specify the position of a sprite on the screen, you need to supply the VIC chip with a pair of coordinates. The coordinates refer to the top-left pixel in the sprite. Each sprite has a horizontal (X) position register and a vertical (Y) position register. If you call the first register in the VIC chip V, then the first sprite's position registers are V (horizontal) and V+1 (vertical). The second sprite is controlled by V+2 (horizontal) and V+3 (vertical) and so on as far as V+15. POKEing values into these locations will position a sprite when it is displayed. The range of values that can be used for the sprite positions are 0-255 for the vertical coordinate, and 0-511 for the horizontal coordinate.

You can see these registers being used in the program that follows. The sprite is positioned and turned on by line 10090.

SIMPLE SPRITE PROGRAM

```
10000 V=53248
10010 POKE 53280,6 : POKE 53281,2
10020 PRINT CHR$(147)
10030 FOR C=0 TO 63
10040 READ BYTE
10050 POKE 2048+C,BYTE
10060 NEXT C
10070 POKE 2040,32
10080 POKE V+39,7
10090 POKE V,100 : POKE V+1,100 : POKE V
+16,0 : POKE V+21,1
10100 POKE V+23,1 : POKE V+29,1
10110 GOTO 10110
15000 DATA 48,60,12,120,126,30,124
15010 DATA 255,62,255,255,255,255,255
15020 DATA 255,255,189,255,239,24,247
15030 DATA 199,90,227,207,126,243,143
15040 DATA 255,241,159,126,249,31,36
15060 DATA 248,31,129,248,63,231,252
15070 DATA 63,255,252,63,255,252,127
15080 DATA 255,254,127,221,254,245,204
15090 DATA 223,164,136,85,164,136,85
16000 DATA 0
READY.
```

SIMPLE SPRITE DISPLAY



In line 10090, the first two POKEs determine the position of the sprite. POKE V,100 sets the horizontal position of the sprite's top-left corner to 100 pixels from the left while POKE V+1,100 sets the vertical position to 100 pixels down from the top. The third statement, POKE V+16,0 can be used to modify the horizontal position. In this program, it is set to zero, and hence doesn't do anything noticeable. However on page 10 you will see why it is included here, and what happens when you use values other than zero with it. Lastly, POKE V+21,1 turns the sprite on. It doesn't matter where this is done in the program, but if you leave it out, the sprite will be put into memory but will fail to appear on the screen.

Setting sprite colors

The previous program also contains several POKE statements which refer to VIC memory locations that control color. The first two locations, V+32 and V+33, control the border and screen background colors respectively. The color of an individual sprite is controlled by VIC locations V+39 to V+46. The first location controls the color of sprite 0, the second sprite 1, and so on. The color code is POKEd into the appropriate address, thereby setting the color of the sprite. You will find a complete Commodore color combination chart on page 62. It is possible to use more than one color with sprites, although resolution drops when you do this. You can see some examples of multi-color sprites on page 21. Finally, line 10070 in the program tells the computer where to find the DATA.

The next program produces a more complex display by making eight sprites from the same DATA. In this program the eight colors are specified by the loop between lines 10070 and 10100.

MULTI-SPRITE PROGRAM

```
10000 V=53248 : POKE V+21,255 : POKE V+2
3,255 : POKE V+29,255
10010 POKE 53280,0 : POKE 53281,0
10020 PRINT CHR$(147)
10030 FOR C=0 TO 63
10040 READ B
10050 POKE 2048+C,B
10060 NEXT C
10070 FOR C=0 TO 7 : POKE 2040+C,32
10080 POKE V+39+C,C+1
10090 POKE V+2*C,RND(0)*207+48 : POKE V+
(2*C)+1,RND(0)*151+48
10100 NEXT C
10110 GOTO 10110
15000 DATA 8,16,128,4,9,0,2
15001 DATA 13,0,2,7,0,4,3
15002 DATA 226,8,3,220,16,7,248
15003 DATA 32,15,240,32,31,226,32
15004 DATA 31,244,32,63,250,32,63
15005 DATA 252,48,127,248,24,127,240
15006 DATA 8,127,240,12,127,240,6
15007 DATA 127,224,3,63,224,3,255
15008 DATA 192,1,255,128,0,31,224
15009 DATA 0
```

Specifying sprite DATA

You can define and store up to 32 sprite designs in memory at once, but you can only switch on and display a maximum of eight sprites at any one time. However, each of the eight sprites can be any of 32 designs. The way that the VIC chip knows which of the 32 sprite DATA areas are to be used for each of the eight sprites is by looking at the contents of memory locations 2040 to 2047. Location 2040 controls sprite 0, 2041 controls sprite 1 and so on, up to location 2047 which controls sprite 7. There are 256 different starting positions that can be specified for a block of sprite DATA. These starting positions begin at location 0 and are spaced at locations 64 bytes apart. So, for example, if you key in POKE 2040,40 the computer will make up sprite 0 from the DATA beginning at starting position 40, which is at memory address 40x64=2560.

The program on this page shows how you can alter the setting in location 2040 so that different sprites are all made up from the same DATA. The result is 8 sprite "clones" on the screen.

How to alter bits within a byte

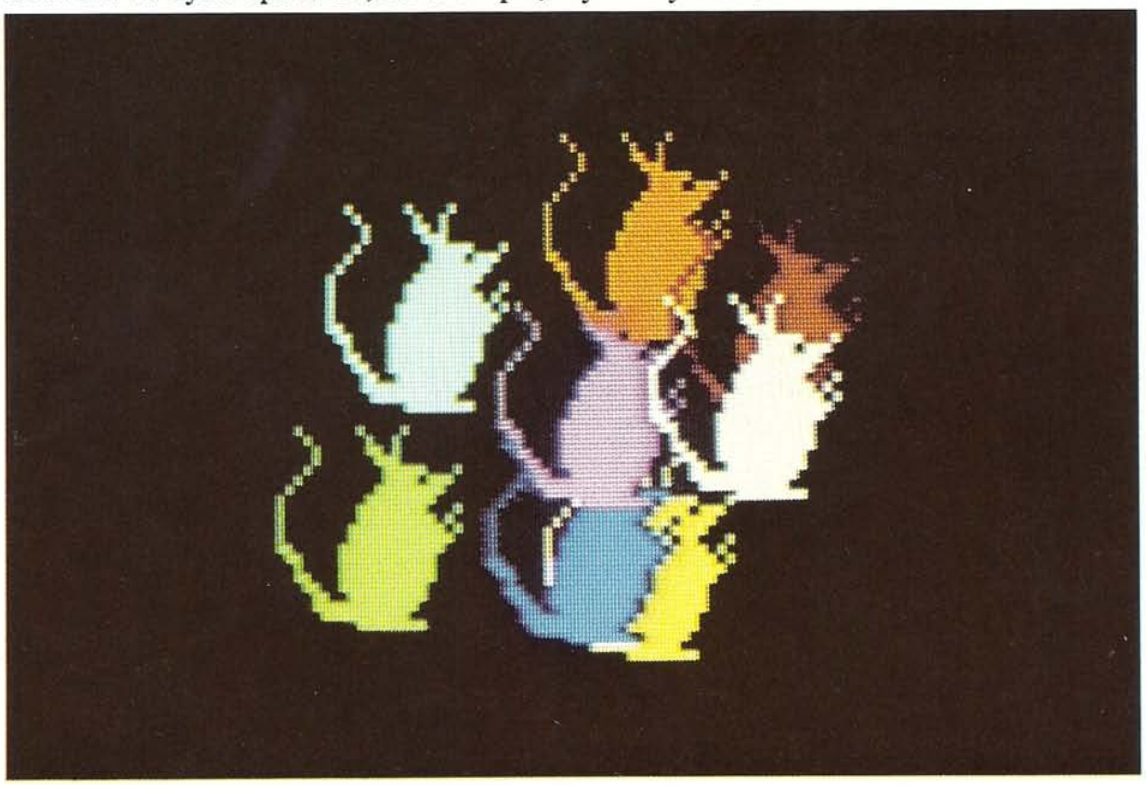
The technique for changing one bit only within a byte is called bit masking. This technique, which is described fully in Book Three, is achieved with the aid of the AND and OR logical operators. To set a single bit within a byte to 1, you use the OR function. To reset a bit to 0 you use the AND function. The table below shows the values to use with AND and OR to reset or set any desired bit within a byte. For example, to turn on just sprite 4 you would use the line:

```
POKE V+21,PEEK (V+21) OR 16
```

and to turn off sprite 6 you would use the line

```
POKE V+21, PEEK(V+21) AND 191
```

Sprite no.	To turn on	To turn off
0	OR 1	AND 254
1	OR 2	AND 253
2	OR 4	AND 251
3	OR 8	AND 47
4	OR 16	AND 239
5	OR 32	AND 223
6	OR 64	AND 191
7	OR 128	AND 127



MULTI-SPRITE PROGRAM

How the program works

This program makes the maximum number of sprites available on the screen at once out of just one set of sprite DATA. It does this by altering the sprite DATA pointers with a loop.

Line 10000 makes the variable V equal to the first register in the VIC chip, and turns on all eight sprites.

Lines 10030-10060 put identical DATA for eight sprites into memory.

Lines 10070-10100 set the sprite DATA pointers, color the sprites and then position them randomly within the visible screen boundaries.

Lines 15000-15009 contain the sprite DATA.

KEYBOARD ANIMATION

Once you know how to produce sprites on the screen, simple animation is quite easy. To move a sprite around the screen, all you need to do is repeatedly update its position. The easiest way to do this is by using a loop to increase or decrease the position registers. You can move a sprite one pixel at a time, giving really smooth animation, or, if you want it to move more quickly, you can use a larger increment.

The screen coordinates

If you look at the sprite coordinates grid on page 63, you can see that a fairly large proportion of the range is off-screen. This allows sprites to be moved smoothly onto and off the screen in any direction. Furthermore, you will see that the horizontal range, 0-511, is twice the vertical range, 0-255.

If you are familiar with binary arithmetic, you will know that it is not possible to specify a number up to 511 with one byte. A byte can only code a number as high as 255, so in addition to the normal horizontal VIC register, another register is required to hold a ninth bit which allows the horizontal coordinates to extend to 511. The ninth bit for each sprite is held in register V+16. Setting a sprite's bit to 1 in this byte increases the sprite's horizontal coordinate by 256

Moving sprites from the keyboard

Many games rely on moving sprites from the keyboard. The programs on these two pages show how you can do this for a single and a double sprite. They both work by checking for key-presses. The keys used by the programs are the two cursor keys in the bottom right-hand corner of the keyboard.

The first program is arranged so that sprite movement continues for as long as one of the cursor keys is held down. Line 90 is the one which alters the value in location V+16, allowing the sprite to move over the halfway point of the horizontal coordinate range.

The program has also been arranged so that the sprite does not move out of the visible screen boundaries. This is done by testing the value of the horizontal and vertical coordinates to see if they are equivalent to those of the screen boundaries. If they are, the program will not allow further movement in that direction.

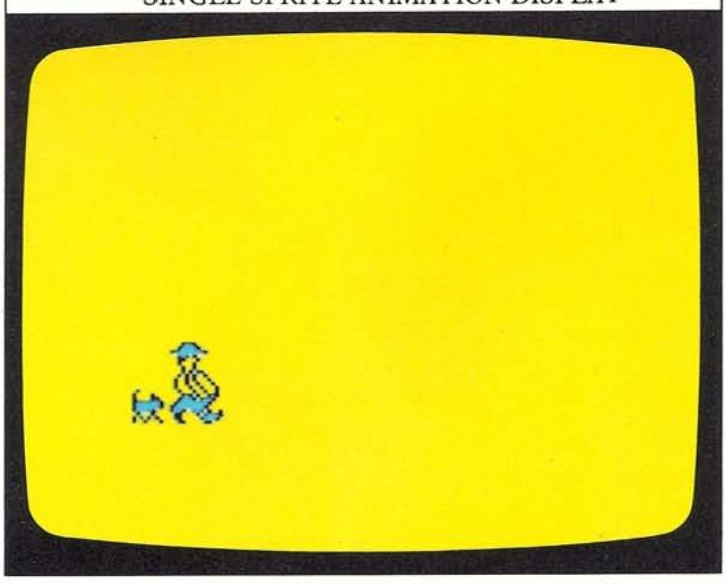
The limits tested for depend on the size of the sprite. Sprites can be expanded using a technique described on page 18, and in fact the ones featured on these two pages are fully expanded. Sprite expansion alters the distance from the top-left corner to the right and bottom edges. Normally this is 24 pixels, but it can be increased to 48, and this must be allowed for when testing for the screen boundary. In the following program, the test is carried out by lines 140 and 150.

SINGLE SPRITE ANIMATION PROGRAM

```
LIST -160

10 V=53248 : PRINT CHR$(147);CHR$(5)
20 POKE 53280,7 : POKE 53281,7
30 POKE V+23,1 : POKE V+29,1
40 FOR C=0 TO 63 : READ B
50 POKE 2048+C,B : NEXT C
60 POKE V+39,6 : POKE 2040,32
70 X=56 : Y=118 : POKE V+21,1
80 H=INT(X/256)+2 : L=X AND 255
90 POKE V,L : POKE V+1,Y : POKE V+16,H
100 GET A$ : A=ASC(A$+CHR$(0))
110 POKE 197,64
120 DX=((A=157)-(A=29))*4
130 DY=((A=145)-(A=17))*4
140 IF X+DX<300 AND X+DX>20 THEN X=X+DX
150 IF Y+DY<209 AND Y+DY>47 THEN Y=Y+DY
160 GOTO 80

READY.
```

```
LIST 500-

500 DATA 0,7,128,0,15,192,0
501 DATA 63,240,0,14,128,0,12
502 DATA 64,0,8,128,0,4,128
503 DATA 0,5,96,0,10,144,0
504 DATA 18,136,0,18,68,0,17
505 DATA 34,64,8,146,132,7,78
506 DATA 134,15,188,255,31,248,255
507 DATA 63,240,252,60,233,68,24
508 DATA 95,170,24,62,145,12,24
509 DATA 0
READY.
```

SINGLE SPRITE ANIMATION DISPLAY



How to animate a double sprite

Animating a double sprite under keyboard control uses the same techniques as single-sprite animation, but with some modifications. The program on this page lets you move a horizontal pair of sprites around the screen, again using the cursor keys.

Programming this kind of animation poses a problem when you want to move the pair across the mid-point of the horizontal coordinate range. The program has to be organized so that the pair can move across the screen together. This is simple enough over most of the screen, but when the top-left corner of the leading sprite reaches horizontal coordinate 255, the sprite's bit in location V+16 must then be set to 1 to allow it to move into the right-hand part of the horizontal coordinate range. However, for a period the trailing sprite's bit in location V+16 must remain at a zero value. Only when the top-left corner of the trailing sprite reaches the half-way point must the second sprite's bit change. The reverse applies when the sprites are moving in the opposite direction. In the program that follows, movement across the middle of the horizontal range is taken care of by lines 110 and 150 using two variables, HA and HB.

DOUBLE SPRITE KEYBOARD ANIMATION PROGRAM

How the program works

The position of the pair of sprites is controlled from the keyboard by the cursor keys. The sprites move 4 pixels in any direction in response to one cursor key-press.

Lines 40-60 put the first sprite into memory.

Lines 70-90 do the same for the second sprite.

Line 100 sets the initial positions and turns on the two sprites.

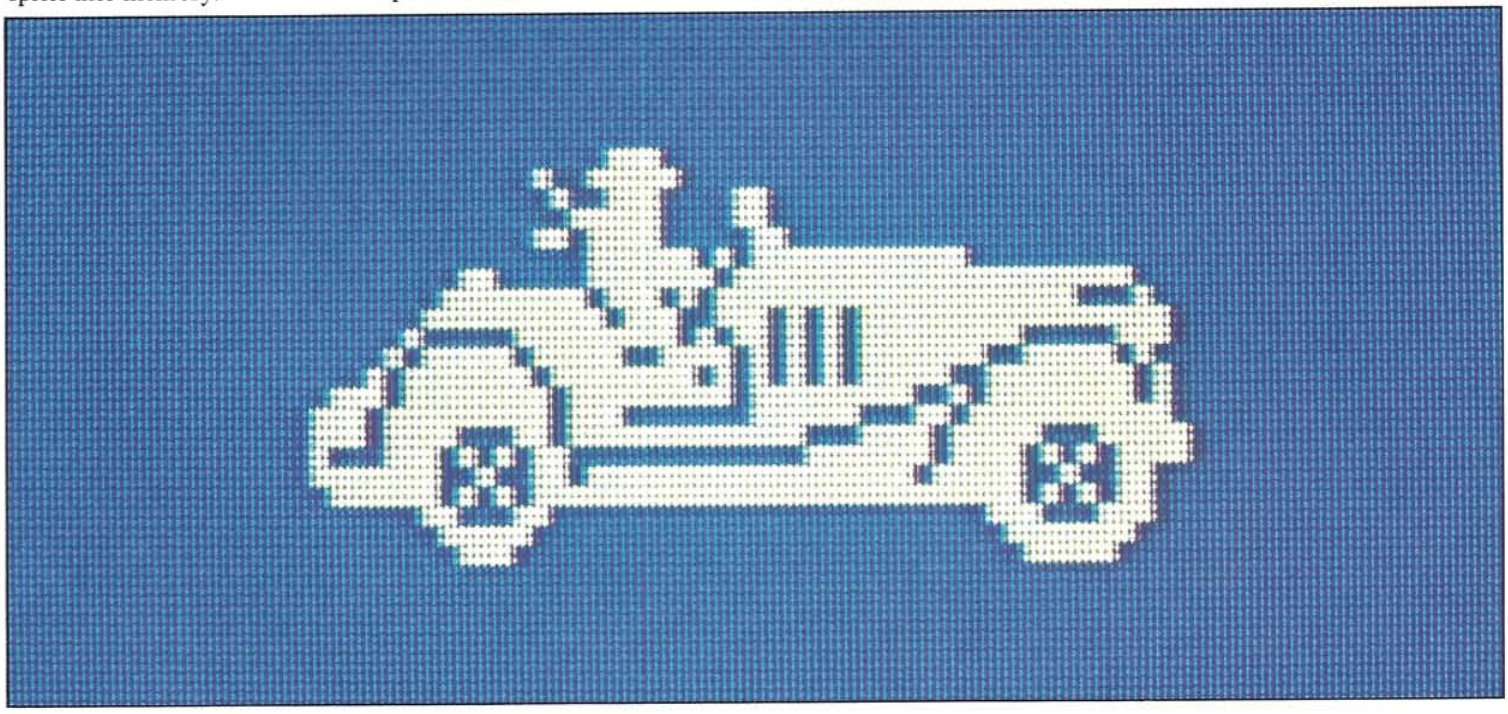
Lines 110-210 control movement by responding to the cursor keys. Both sprites automatically cross the horizontal mid-point.

Lines 500-519 contain the sprite DATA.

DOUBLE SPRITE KEYBOARD ANIMATION PROGRAM

```
10 V=53248 : PRINT CHR$(147);CHR$(5)
20 POKE 53280,6 : POKE 53281,6
30 POKE V+23,3 : POKE V+29,3
40 FOR C=0 TO 63 : READ B
50 POKE 2048+C,B : NEXT C
60 POKE V+39,1 : POKE 2040,32
70 FOR C=0 TO 63 : READ B
80 POKE 2112+C,B : NEXT C
90 POKE V+40,1 : POKE 2041,33
100 X=56 : Y=118 : POKE V+21,3
110 HA=INT(X/256) : HB=INT((X+48)/256)
120 LA=X AND 255 : LB=(X+48) AND 255
130 POKE V,LA : POKE V+1,Y
140 POKE V+2,LB : POKE V+3,Y
150 POKE V+16,HA+2*HB
160 GET A$ : A=ASC(A$+CHR$(0))
170 POKE 197,64
180 DX=((A=157)-(A=29))*4
190 DY=((A=145)-(A=17))*4
200 IF X+DX<250 AND X+DX>20 THEN X=X+DX
210 IF Y+DY<209 AND Y+DY>47 THEN Y=Y+DY
220 GOTO 110

READY.
```

```
LIST 500-

500 DATA 0,0,224,0,9,240,0
501 DATA 4,225,0,3,225,0,13
502 DATA 224,0,1,250,0,193,253
503 DATA 1,254,235,3,255,115,4
504 DATA 31,245,11,239,158,23,247
505 DATA 250,119,251,254,239,251,128
506 DATA 239,27,255,142,172,0,254
507 DATA 77,255,126,175,255,3,24
508 DATA 0,1,240,0,0,224,0
509 DATA 0
510 DATA 0,0,0,0,0,0,128
511 DATA 0,0,128,0,0,192,0
512 DATA 0,255,240,0,255,255,248
513 DATA 255,255,196,171,255,254,171
514 DATA 254,14,171,249,244,171,231
515 DATA 250,255,151,250,252,111,254
516 DATA 227,223,31,31,225,175,255
517 DATA 190,76,255,254,172,0,7
518 DATA 28,0,3,248,0,1,240
519 DATA 0
READY.
```



SPRITE CARTOONS

You may be wondering what the point is of having facilities for storing DATA for up to 32 sprites in memory when you can only display eight sprites at a time on the screen. The main reason for this is that the VIC chip does not have the capacity for controlling any more information. However, this apparent limitation does allow you to produce some interesting effects in a display, one of the best of these being sprite cartoons.

Switching the DATA pointers

The VIC chip is able to switch its sprite DATA pointers from one area of the memory to another very quickly. You saw in the Multi-Sprite program on page 9 how separate sprite DATA pointers can be made to point to the same area of DATA, creating cloned sprites. In that case, the area pointed to by each pointer stayed the same. But you can control the area indicated by using a variable instead of a specific number. The following program does just this. It produces an effect opposite to that in the Multi-Sprite program — it makes the DATA pointers for just a single sprite change in a specified way to point to different areas of sprite DATA, creating a moving cartoon figure.

Multi-frame animation

The program on these two pages shows you how you can make a simple cartoon figure using five sets of sprite DATA. For a single sprite, you could have 32-frame animation, but the listing for this would be enormous, needing 2048 separate DATA numbers (although it wouldn't be too difficult to produce, as you will find out on the next page). Similarly, you could have four 8-frame cartoon figures or two 16-frame ones, or in fact any combination as long as when multiplied together the total number of sprites does not exceed 32.

The sprite in this program is a galloping horse which runs across the screen. It is programmed by storing five sets of DATA for the horse, showing it in different positions. The sprite DATA used at any point is linked to the sprite's position across the screen.

How to abbreviate sprite DATA

If each frame in a cartoon is significantly different from the first one, you need to specify a completely new set of 64 DATA numbers to code it. However, if you want to make a cartoon figure in which the top half, for example, stays the same, while only the bottom half changes, you can use some of the first sprite's DATA for all the sprites, thereby reducing the amount of program needed.

The way to do this is to split the DATA up into the part which is repeated and the parts which are different for each of the sprites. Make the program put the shared part of the DATA into the first part of all the DATA areas, and then put the rest of the DATA onto the end of the initial parts. This technique is easiest to use with sprites that are split horizontally. Because of the way sprite DATA is arranged, splitting sprites vertically is more tricky.

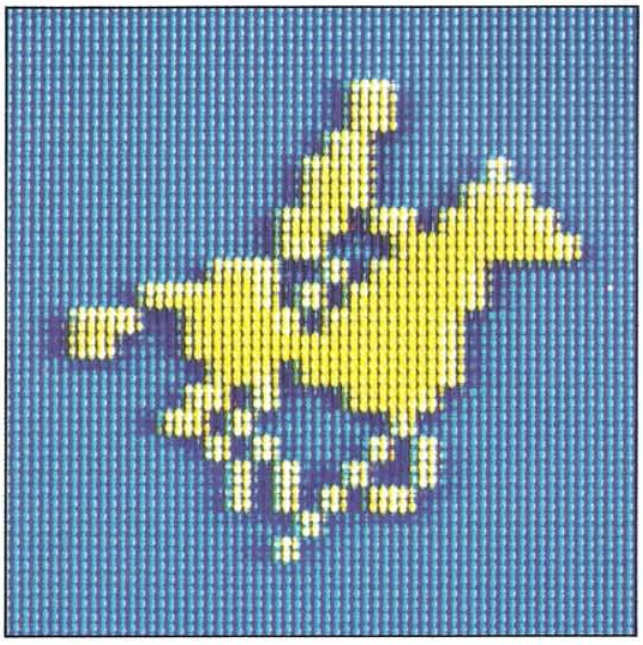
SPRITE CARTOON PROGRAM

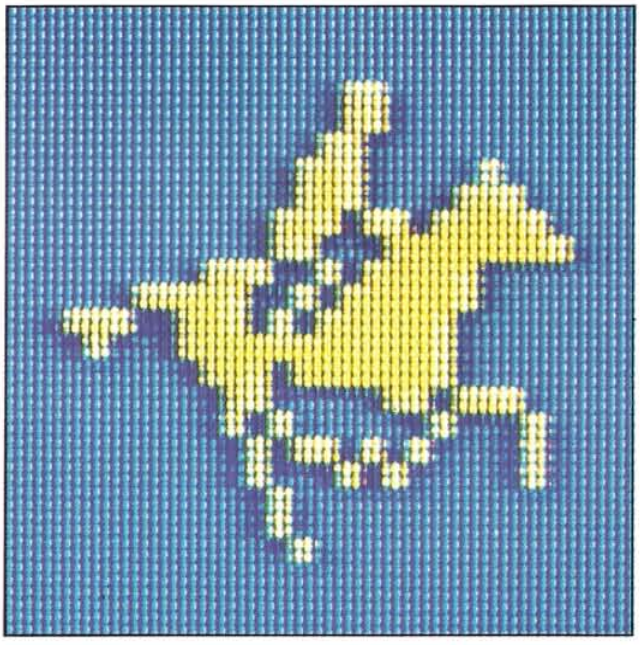
How the program works

The program creates five different sprites. It uses these five sets of sprite DATA in turn as horizontal coordinate of the single sprite increases on the screen.

Lines 10020-10030 put the DATA for the five frames of the cartoon into memory.

Lines 10070-10130 form a loop which selects sprite DATA according to the horizontal position with POKE 2040,S.





How to design the frames

Designing frames for this sort of animation can be quite difficult, particularly if you want to produce a uniformly smooth movement on the screen. If you want to try sprite cartooning yourself and you want to use a large number of frames, you will find it takes a long time to draw up all the separate designs and then key them in as DATA numbers. Fortunately, there is a much quicker and accurate alternative.

The Sprite Editor program which is developed on pages 24-32 lets you design frames in a cartoon sequence without having to calculate any numbers or draw anything on paper. It lets you create a sprite using a giant screen grid, and then store it as DATA numbers in program lines. Having designed your first sprite, you can then summon it to the grid again and gradually modify it, storing the stages as you do so. All these stages can automatically be made into DATA lines, so that by the end of the process you have a section of DATA that will produce the cartoon. With the Sprite Editor, the DATA is sure to be correct.

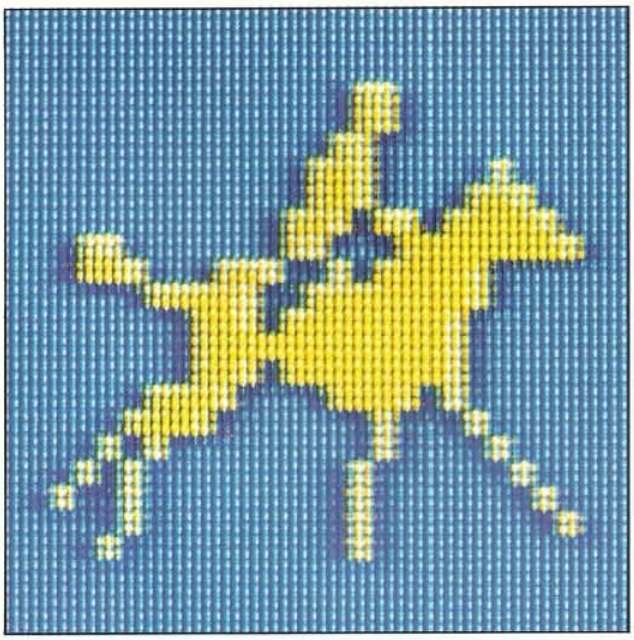
SPRITE CARTOON PROGRAM

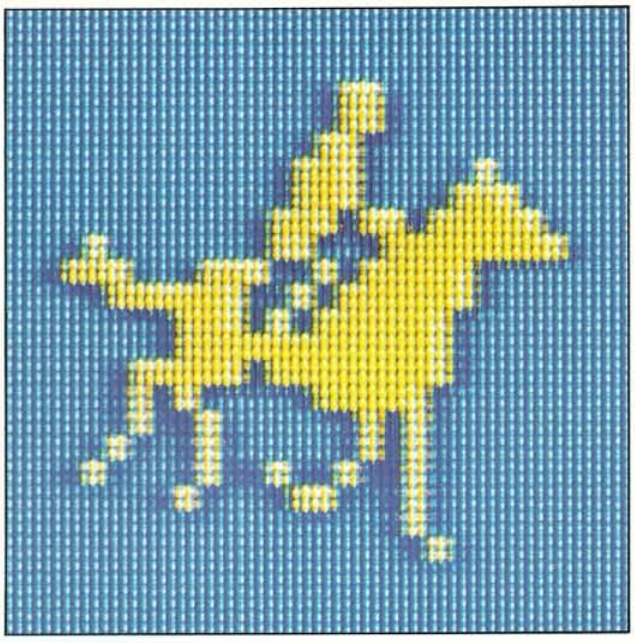
```
10000 V=53248 : PRINT CHR$(147)
10010 POKE 53280,6 : POKE 53281,6
10020 FOR C=0 TO 319 : READ BYTE
10030 POKE 2048+C,BYTE : NEXT C
10040 POKE V+29,1 : POKE V+23,1
10050 X=0 : Y=150 : S=32
10060 POKE V+21,1 : POKE V+39,7
10070 HI=INT(X/256) : LO=X AND 255
10080 POKE 2040,S : POKE V+16,HI
10090 POKE V,LO : POKE V+1,Y
10100 X=X+4 : IF X>350 THEN X=0
10110 S=S+1 : IF S>36 THEN S=32
10120 FOR T=1 TO 100 : NEXT T
10130 GOTO 10070
30000 DATA 0,0,0,0,6,0,0
30001 DATA 6,0,0,12,0,0,28
30002 DATA 16,0,28,56,0,59,124
30003 DATA 0,49,254,1,203,224,15
30004 DATA 215,224,115,239,192,99,255
30005 DATA 192,1,223,192,3,71,128
30006 DATA 6,129,0,1,66,128,0

READY.
```

SPRITE CARTOON PROGRAM (CONTD.)

```
30007 DATA 164,128,0,171,0,0,16
30008 DATA 0,0,32,0,0,0,0
30009 DATA 0
30010 DATA 0,0,0,0,6,0,0
30011 DATA 6,0,0,12,0,0,28
30012 DATA 16,0,28,56,0,59,124
30013 DATA 0,49,254,1,203,224,15
30014 DATA 151,224,115,175,192,35,255
30015 DATA 192,1,223,192,0,193,184
30016 DATA 0,160,68,0,90,132,0
30017 DATA 69,4,0,32,0,0,32
30018 DATA 0,0,16,0,0,0,0
30019 DATA 0
30020 DATA 0,0,0,0,6,0,0
30021 DATA 6,0,0,12,0,0,28
30022 DATA 16,0,28,56,0,59,124
30023 DATA 32,49,254,113,203,224,31
30024 DATA 151,224,3,175,192,7,255
30025 DATA 192,7,191,216,7,3,168
30026 DATA 13,0,16,25,0,24,16
30027 DATA 128,36,32,64,2,32,32
30028 DATA 0,32,16,0,0,0,0

READY.
```

```
LIST 30029-

30029 DATA 0
30030 DATA 0,0,0,0,6,0,0
30031 DATA 6,0,0,12,0,0,28
30032 DATA 16,0,28,56,0,59,124
30033 DATA 96,49,254,113,203,224,15
30034 DATA 159,224,3,191,192,3,255
30035 DATA 192,3,191,192,15,15,64
30036 DATA 30,2,32,40,2,16,80
30037 DATA 4,8,144,4,4,16,4
30038 DATA 2,32,4,0,0,0,0
30039 DATA 0
30040 DATA 0,0,0,0,6,0,0
30041 DATA 6,0,0,12,0,0,28
30042 DATA 16,0,28,56,0,59,124
30043 DATA 32,49,254,113,203,224,31
30044 DATA 151,224,3,175,192,3,255
30045 DATA 192,15,191,192,10,15,128
30046 DATA 9,2,128,17,2,128,33
30047 DATA 36,128,2,24,128,0,0
30048 DATA 128,0,0,64,0,0,0
30049 DATA 0
READY.
```





USING BACKGROUNDS

So far in this book, you have only seen sprites on otherwise blank screens. Adding backgrounds makes the displays much more interesting, particularly because sprites can be made to interact with them.

The high-resolution backgrounds shown in the following demonstration programs were created using the Graphics Editor program featured in Book Three. If you have read Book Three, you can use the Background Loader program below to call up any background you have designed with the Graphics Editor, and then you can display sprites on it.

BACKGROUND LOADER PROGRAM

```
LIST

10000 SYS A2 : PRINT CHR$(147)
10010 INPUT "1/2 ";A$
10020 PRINT : INPUT "LOAD ";FI$
10030 SYS A1 : SYS B1,16
10040 IF A$="2" THEN 10060
10050 OPEN 1,1,0,FI$ : GOTO 10070
10060 OPEN 1,8,2,"0:"+FI$+",S,R"
10070 FOR C=8192 TO 16191 : GET#1,A$
10080 POKE C,ASC(A$+CHR$(0)) : NEXT C
10090 FOR C=6144 TO 7167 : GET #1,A$
10100 POKE C,ASC(A$+CHR$(0)) : NEXT C
10110 CLOSE 1 : FOR C=1024 TO 2047
10120 POKE C,PEEK(5120+C) : NEXT C
READY.
```

To use a display file that you have stored on tape or disk using the Graphics Editor, you need to make sure that the background is displayed before the sprites. To do this, make sure that the Background Loader line numbers are lower than your sprite program (you may have to edit them). Also, you will need the routines in block A which appears on page 60. You must also use the restore routine in your sprite program to make sure that the DATA is READ from the right place, and take out any screen color or clearing commands. You can see two examples of the sort of displays you can produce by this method below. If you haven't used the Graphics Editor, you will find that the effects described next can be seen with low-resolution (text mode) backgrounds as well.

Setting priorities

There are two different ways in which objects can interact on the screen. A sprite can either pass in front of or behind another sprite, or it can pass in front of or behind a background object.

BACKGROUND PRIORITIES PROGRAM

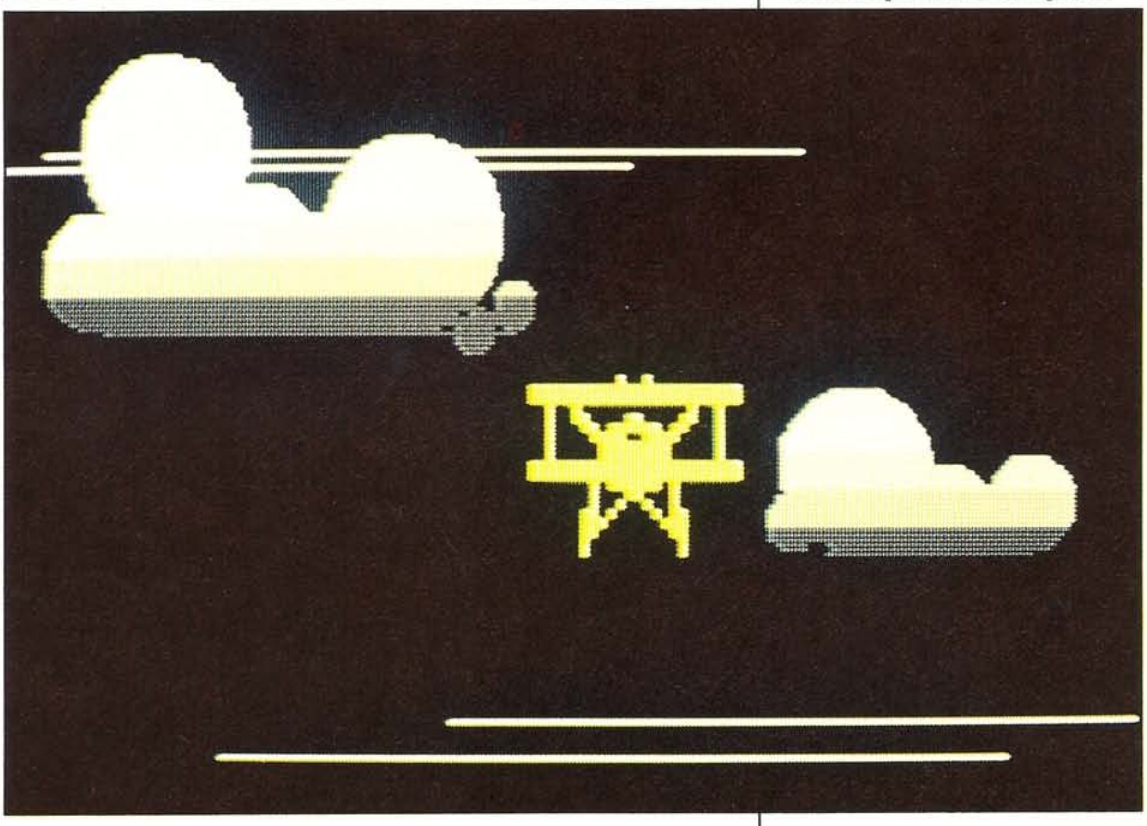
How the program works

The program creates a pair of sprites. After the trailing sprite has reached horizontal coordinate 190, the priorities of both sprites are changed so that they pass in front of the background.

Line 30040 sets the initial background priorities.

Lines 30080-30100 position the sprites.

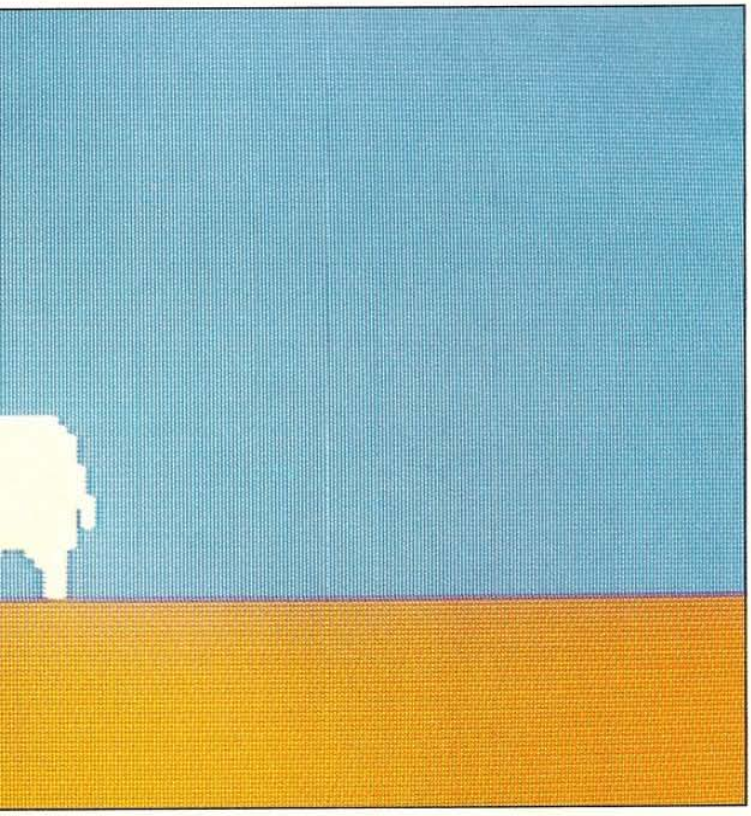
Line 30110 resets the priorities with POKE V+27,0.



In the first case, the rule is that a low-numbered sprite always passes in front of a high-numbered one. So, for example, sprite 3 always passes in front of sprite 5, while the same sprite always passes behind sprite 1. Sprite-background priorities are slightly different because each sprite can be set individually to move in front or behind lit background pixels. This is controlled by the contents of VIC register V+27, where bit 0 is used for sprite 0, bit 1 for sprite 1 and so on. If a bit is set to 1 within this register, then its associated sprite will move behind background objects, while if the same bit is reset to 0, the sprite will pass in front. The following program, which uses a Graphics Editor background, shows both kinds of background priority.

BACKGROUND PRIORITIES PROGRAM

```
LIST 30010-30160

30010 FOR C=0 TO 127 : READ BYTE
30020 POKE 2048+C,BYTE : NEXT C
30030 X0=350 : Y0=134 : X1=374 : Y1=134
30040 POKE V+21,3 : POKE V+27,3
30050 POKE 2040,32 : POKE 2041,33
30060 POKE V+39,0 : POKE V+40,0
30070 HI=INT(X0/256)+2*INT(X1/256)
30080 POKE V+16,HI
30090 POKE V+2,X1 AND 255 : POKE V+3,Y1
30100 POKE V,X0 AND 255 : POKE V+1,Y0
30110 IF X1<190 THEN POKE V+27,0
30120 X0=X0-1 : X1=X1-1
30130 IF X1>24 THEN 30070
30140 IF X1>1 THEN X0=0 : GOTO 30070
30150 POKE V+21,0 : GOTO 30150
30160 POKE V+21,0 : GOTO 30160
```

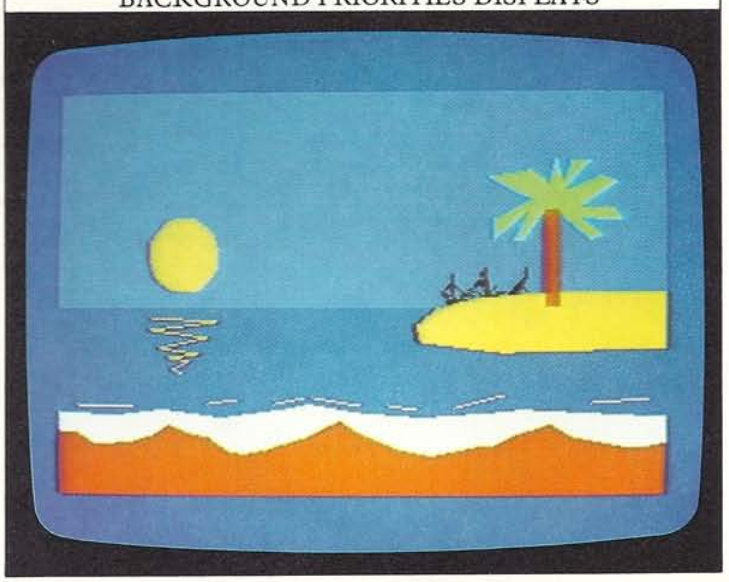


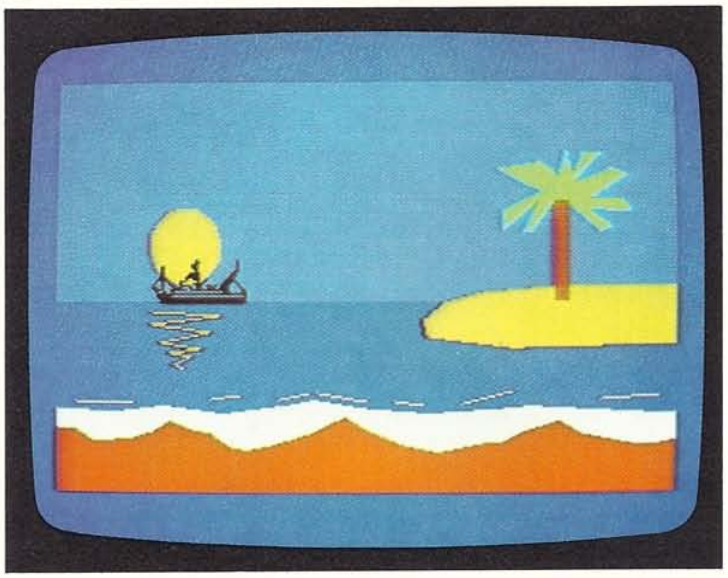
BACKGROUND PRIORITIES PROGRAM (CONTD.)

```
LIST 40000-

40000 DATA 0,0,2,0,0,3,0
40001 DATA 0,10,4,0,10,4,0
40002 DATA 6,4,0,2,4,0,23
40003 DATA 4,0,63,14,0,85,21
40004 DATA 0,69,36,131,129,68,66
40005 DATA 5,132,33,225,228,30,17
40006 DATA 92,42,143,34,32,0,31
40007 DATA 255,255,29,64,0,15,255
40008 DATA 255,7,213,251,7,255,255
40009 DATA 0
40010 DATA 0,0,16,0,0,16,0
40011 DATA 0,16,0,0,16,0,0
40012 DATA 16,0,0,48,0,0,96
40013 DATA 0,0,224,0,1,192,16
40014 DATA 3,192,32,1,160,64,15
40015 DATA 16,141,158,8,133,30,4
40016 DATA 255,254,3,0,7,3,255
40017 DATA 255,255,0,0,1,255,255
40018 DATA 255,234,175,255,255,255,255
40019 DATA 0
READY.
```

You can see in the following displays what happens as the ship gradually moves across the background, changing priorities after a particular point is reached.

BACKGROUND PRIORITIES DISPLAYS





DETECTING COLLISIONS

The Commodore's VIC chip allows collision detection with simple programming. Whenever a lit pixel is about to be plotted on the screen in a position already occupied by another one, the VIC chip signals that a collision has occurred.

Eight sprite-sprite collision detectors are contained in VIC register V+30 in the usual 1-bit-per-sprite arrangement. Similarly, eight sprite-background collision detectors are contained in VIC register V+31. When a collision takes place, the VIC chip records it by setting the appropriate bit in register V+30 or V+31 to 1. To check for a collision, you just need to PEEK the contents of the appropriate VIC register to see if the sprite's bit is 1. If it is, the bit will then be reset to 0.

The following program shows collision detection in action. In it, one sprite slowly creeps up on another. When they meet the stationary sprite shows a sudden reaction triggered by the collision.

SPRITE COLLISION PROGRAM

```
10 V=53248 : PRINT CHR$(147) : POKE 5328
0,4 : POKE 53281,4
20 POKE V+23,3 : POKE V+29,3
30 FOR C=0 TO 63 : READ BYTE : POKE 2048
+C,BYTE : NEXT C
40 FOR C=0 TO 63 : READ BYTE : POKE 2048
+64+C,BYTE : NEXT C
50 POKE 2040,32 : POKE 2041,33
60 POKE V+39,0 : POKE V+40,7
70 X0=300 : Y0=150 : X1=150 : Y1=150
80 DX=-1 : CD=PEEK(V+30)
90 POKE V+2,X1 : POKE V+3,Y1
100 HI=INT(X0/256) : LO=X0 AND 255
110 POKE V+21,3 : POKE V+16,HI
120 POKE V,LO : POKE V+1,Y0
130 CD=PEEK(V+30) : X0=X0+DX
140 IF CD=0 THEN 100
150 LO=X1 AND 255
160 POKE V+2,LO : POKE V+3,Y0
170 X1=X1+DX*15
180 IF X1<24 THEN 180
190 GOTO 150
```

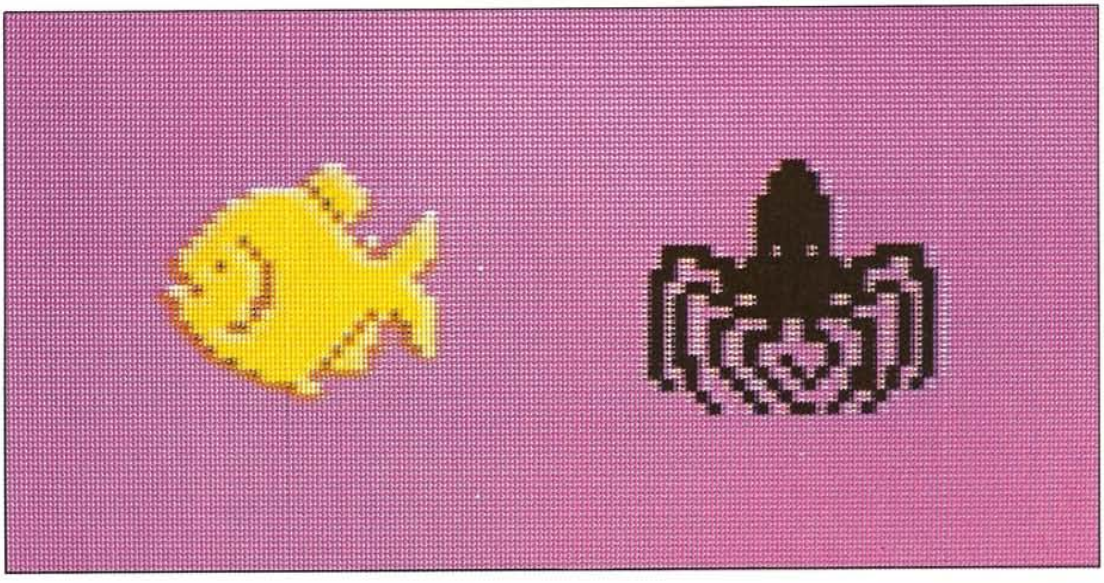
The octopus moves towards the fish until its collision detector in line 140 signals that it has touched another sprite. At this point, the loop that moves the octopus is interrupted and instead the fish moves to the left. It moves much faster than the octopus because its horizontal coordinate is changed by 15 pixels each move.

Detecting collisions in a game

The program opposite uses a technique you saw earlier for moving a sprite under keyboard control. The program draws a random maze out of colored blocks. You have to guide the farmer, who starts on the left, through the maze to reach the pig on the right. The object of the game is to make the farmer and the pig collide. The keys for moving the farmer up and down are S and X, while the comma and full point keys move him left and right. The program uses sprite-background and sprite-sprite collision detection.

SPRITE COLLISION PROGRAM (CONTD.)

```
LIST 15000-

15000 DATA 0,24,0,0,60,0,0
15001 DATA 60,0,0,126,0,0,126
15002 DATA 0,0,126,0,0,126,0
15003 DATA 120,90,30,78,255,114,195
15004 DATA 255,195,155,255,217,166,126
15005 DATA 101,161,255,133,166,36,101
15006 DATA 164,230,37,164,129,37,180
15007 DATA 145,37,146,74,105,81,36
15008 DATA 106,8,144,144,4,77,32
15009 DATA 0
15010 DATA 0,6,0,0,15,0,1
15011 DATA 247,128,7,251,128,15,253
15012 DATA 2,31,254,6,61,255,78
15013 DATA 126,255,158,119,127,252,63
15014 DATA 127,248,223,127,252,126,127
15015 DATA 254,126,255,174,57,255,70
15016 DATA 31,254,198,15,253,130,3
15017 DATA 251,128,0,240,0,0,24
15018 DATA 0,0,0,0,0,0,0
15019 DATA 0
READY.
```



SPRITE MAZE PROGRAM

How the program works

The program creates a random maze and two sprites, one of which you can control from the keyboard. It detects collisions between the two sprites and between the moving sprite and the background.

Line 230 tests to see if the sprite-sprite collision register has recorded any collisions. If it has, the game is brought to an end after the score appears.

Line 240 tests to see if the moving sprite has touched the maze. If it has, the movement controls are reversed.

All the time you are trying to move through the maze, the program keeps a record of how long you take. To make things more interesting, you have to try to get through the maze without letting the farmer touch the sides. This is where the collision detection comes in. Every time he does touch, you get a two-second time penalty and the controls then reverse the direction in which you travel.

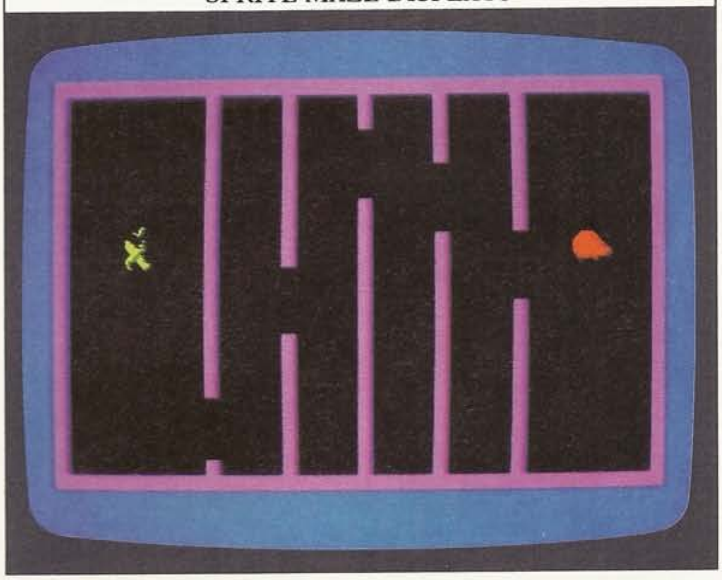
SPRITE MAZE PROGRAM

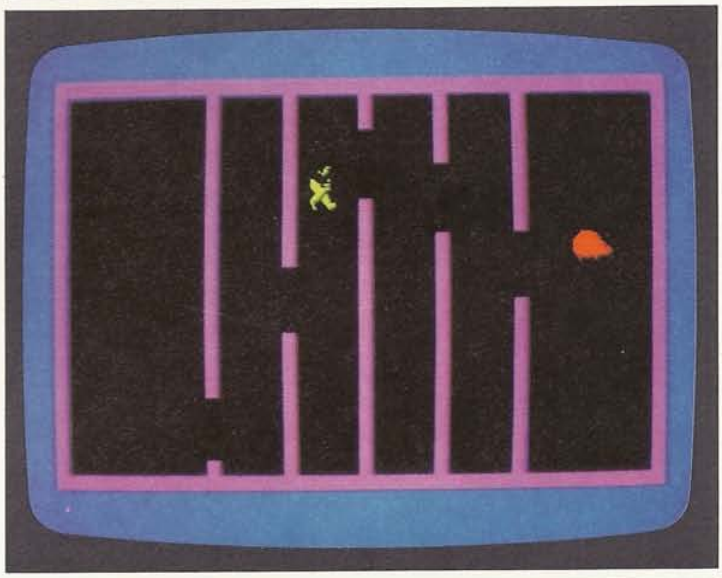
```
10 T=0 : V=53248 : PRINT CHR$(147)
20 POKE 53280,6 : POKE 53281,0
30 FOR X=0 TO 39 : Y=0 : GOSUB 120
40 Y=24 : GOSUB 120 : NEXT X
50 FOR Y=0 TO 24 : X=0 : GOSUB 120
60 X=39 : GOSUB 120 : NEXT Y
70 FOR X=10 TO 30 STEP 5
80 R=INT(RND(0)*18+1)
90 FOR Y=1 TO R : GOSUB 120 : NEXT Y
100 FOR Y=R+5 TO 23 : GOSUB 120
110 NEXT Y : NEXT X : GOTO 140
120 POKE 55296+40*Y+X,4
130 POKE 1024+40*Y+X,224 : RETURN
140 RESTORE : FOR C=0 TO 127 : READ B
150 POKE 2048+C,B : NEXT C
160 POKE V+2,36 : POKE V+3,118
170 X=56 : Y=118 : RH=4 : RV=4
180 POKE V+39,7 : POKE V+40,10
190 POKE 2040,32 : POKE 2041,33
200 T=0 : TI$="000000"
210 SC=PEEK(V+30) : BC=PEEK(V+31)
220 GOSUB 340

READY.
```

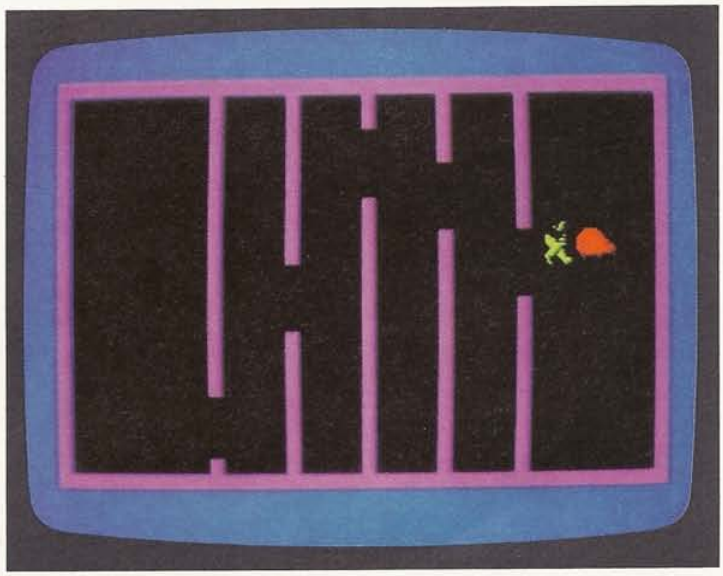
```
230 IF PEEK(V+30)<>0 THEN 370
240 IF PEEK(V+31)=0 THEN 290
250 IF DX<>0 THEN RH=-RH
260 IF DY<>0 THEN RV=-RV
270 X=X-DX : Y=Y-DY : T=T+2
280 GOSUB 340 :GOTO 210
290 GET A$ : A=ASC(A$+CHR$(0))
300 POKE 197,64
310 DX=((A=44)-(A=46))*RH
320 DY=((A=83)-(A=88))*RV
330 X=X+DX : Y=Y+DY : GOTO 220
340 H=INT(X/256)+2 : L=X AND 255
350 POKE V,L : POKE V+1,Y
360 POKE V+16,H : POKE V+21,3 : RETURN
370 PRINT CHR$(147);CHR$(5)
380 POKE 214,8 : PRINT : POKE 211,10
390 PRINT "***"; INT(T+TI/60);"***"
400 PRINT TAB(10);"RUN (0/1)";
410 GET A$ : IF A$<>"" THEN 410
420 POKE V+21,0 : INPUT A$
430 IF A$="1" THEN 10

READY.
```

```
LIST 500-

500 DATA 0,0,0,0,3,128,0
501 DATA 7,0,0,55,64,0,78
502 DATA 32,0,132,64,1,16,64
503 DATA 3,149,128,3,213,0,3
504 DATA 247,128,3,240,128,1,255
505 DATA 128,0,254,0,0,125,0
506 DATA 0,251,0,1,247,0,3
507 DATA 239,128,3,135,144,1,131
508 DATA 240,1,129,240,0,192,192
509 DATA 0
510 DATA 0,0,0,0,0,0,0
511 DATA 0,0,64,0,0,163,252
512 DATA 0,71,254,0,175,255,24
513 DATA 31,255,144,63,255,160,63
514 DATA 255,224,63,255,240,63,255
515 DATA 218,63,255,254,63,255,254
516 DATA 31,255,254,15,255,248,13
517 DATA 247,6,6,6,0,6,4
518 DATA 0,2,4,0,1,2,0
519 DATA 0
READY.
```

SPRITE MAZE DISPLAYS







SPRITE GAMES 1

The next six pages will give you two examples of sprite games, using the sprite facilities you have seen so far, and also some that might be new to you. The program on this page is a fairly short listing which uses sprites and low-resolution graphics. Pages 20-23 feature a longer game which uses more sprites, this time displayed on a high-resolution background. This is created using machine-code graphics routines.

Expanding your sprites

The program on this page uses sprites that are horizontally expanded. You may have noticed that some of the sprites in earlier programs were larger than the normal size. This effect is easy to achieve. A standard sprite can cover an area on the screen up to 24x21 pixels, but with the Commodore's built-in sprite expansion facility, this can be increased by a factor of 2 in either or both the horizontal and vertical directions. This means that a fully expanded sprite can cover 48x42 pixels on the screen, four times the unexpanded area.

The expansion of all sprites is controlled by two separate VIC registers, one for expansion in the horizontal direction and one for expansion in the vertical direction. Register V+29 controls horizontal expansion and register V+23 controls vertical expansion. With these two registers, expansions for each sprite can be controlled individually with the usual 1-bit-per-sprite arrangement. Any bit that is set to 1 in registers V+23 or V+29 signifies that the associated sprite is expanded either horizontally, vertically or both, and any bit reset to 0 indicates that the sprite is of normal size. By using these registers, you can stretch sprites in different directions to create complex mobile shapes with a limited amount of DATA.

For example, to expand just sprite 6 vertically, you would apply the technique shown on page 9 and arrive at the following line of programming:

```
POKE V+23,PEEK(V+23) OR 64
```

When you use the sprite expansion facility, all that changes is the size that each of the normal sprite's 24x21 pixels is plotted on the screen. There is no increase in resolution so that the amount of DATA required to specify a sprite does not increase. This means that the sprites will be larger but coarser.

Programming a darts game

The following program uses sprites to simulate a game of darts. The three darts are sprites made from the same set of DATA, and each one is expanded horizontally. The program uses animation to make the darts move, both under keyboard control and direct program control, and it also checks the positions of the sprites after throwing to calculate your score. To try out the game, key in the three screens of listing that follow.

SPRITE DARTS PROGRAM

```
10 V=53248 : RESTORE : PRINT CHR$(147)
20 POKE 53280,0 : POKE 53281,0
30 POKE 214,6 : PRINT
40 FOR C=1 TO 5 : READ K,S
50 PRINT TAB(4);CHR$(5);S;CHR$(K);
60 PRINT TAB(8);CHR$(18);"  ";
70 PRINT TAB(8);CHR$(18);"  "
80 NEXT C : SC=0 : PRINT CHR$(5)
90 FOR C=0 TO 63 : READ B
100 POKE 2048+C,B : NEXT C
110 FOR C=0 TO 3 : POKE 2040+C,32
120 POKE V+39+C,4 : POKE V+2*C,100
130 POKE V+2*C+1,0 : NEXT C
140 POKE V+29,15 : POKE V+21,15
150 POKE 214,2 : PRINT : POKE 211,30
160 PRINT SC : POKE 650,0
170 FOR D=0 TO 2 : Y0=200
180 X0=280+INT(RND(0)*11)
190 HI=INT(X0/256) : LO=X0 AND 255
200 POKE V,LO : POKE V+16,HI
210 POKE V+1,Y0 : POKE 197,64
220 GET A$ : IF A$="" THEN 220

READY.
```

```
LIST 230-410

230 A=ASC(A$+CHR$(0))
240 IF A=145 AND Y0>100 THEN Y0=Y0-1
250 IF A=17 AND Y0<200 THEN Y0=Y0+1
260 IF A<49 OR A>57 THEN 190
270 N=3.3+0.1*(A-48) : X1=X0-100
280 FOR T=1 TO X1/5 STEP X1/180
290 X=X0-5*T : Y=INT(Y0-N*T+0.1*T↑2)
300 HI=INT(X/256) : LO=X AND 255
310 POKE V,LO : POKE V+16,HI
320 POKE V+1,Y : NEXT T
330 POKE V+2*D+3,Y : POKE V+1,0
340 IF Y>95 AND Y<112 THEN SC=SC+5
350 IF Y>111 AND Y<128 THEN SC=SC+10
360 IF Y>127 AND Y<144 THEN SC=SC+20
370 IF Y>143 AND Y<160 THEN SC=SC+10
380 IF Y>159 AND Y<176 THEN SC=SC+5
390 POKE 214,2 : PRINT : POKE 211,30
400 PRINT SC : NEXT D
410 FOR T=1 TO 2000 : NEXT T : GOTO 10

READY.
```

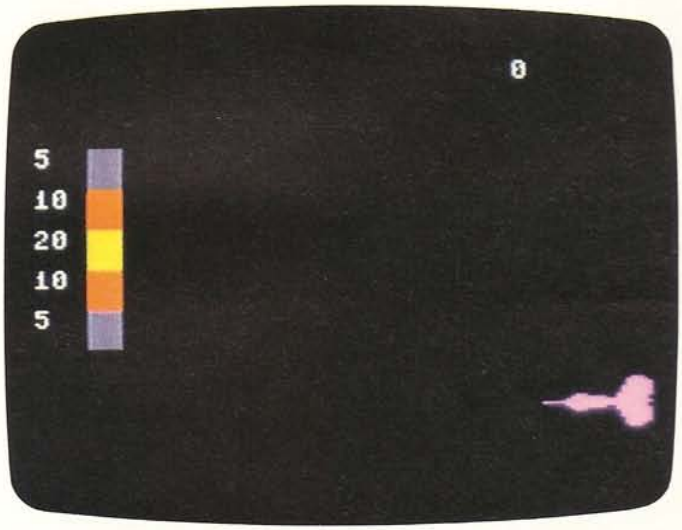
```
LIST 500-

500 DATA 152,5,129,10,158
510 DATA 20,129,10,152,5
600 DATA 0,0,28,0,0,62,0
601 DATA 0,63,0,0,63,0,0
602 DATA 127,0,0,127,0,0,127
603 DATA 1,192,255,3,252,254,7
604 DATA 255,254,255,255,255,7,255
605 DATA 254,3,252,254,1,192,255
606 DATA 0,0,127,0,0,127,0
607 DATA 0,127,0,0,63,0,0
608 DATA 63,0,0,62,0,0,28
609 DATA 0
READY.
```

To play the game, you need to move one dart at a time to the position you want to throw it from, and then tell the computer how fast you want to throw it. You move the darts into the throwing position with the up and down cursor keys, and then you select a throwing speed with keys 1-9. The number you use determines the strength with which the dart is launched (1 is slowest, 9 fastest).

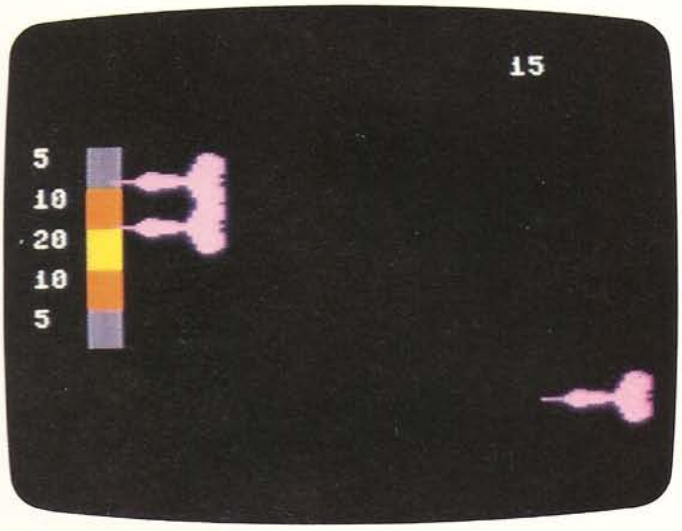
Controlling position and speed

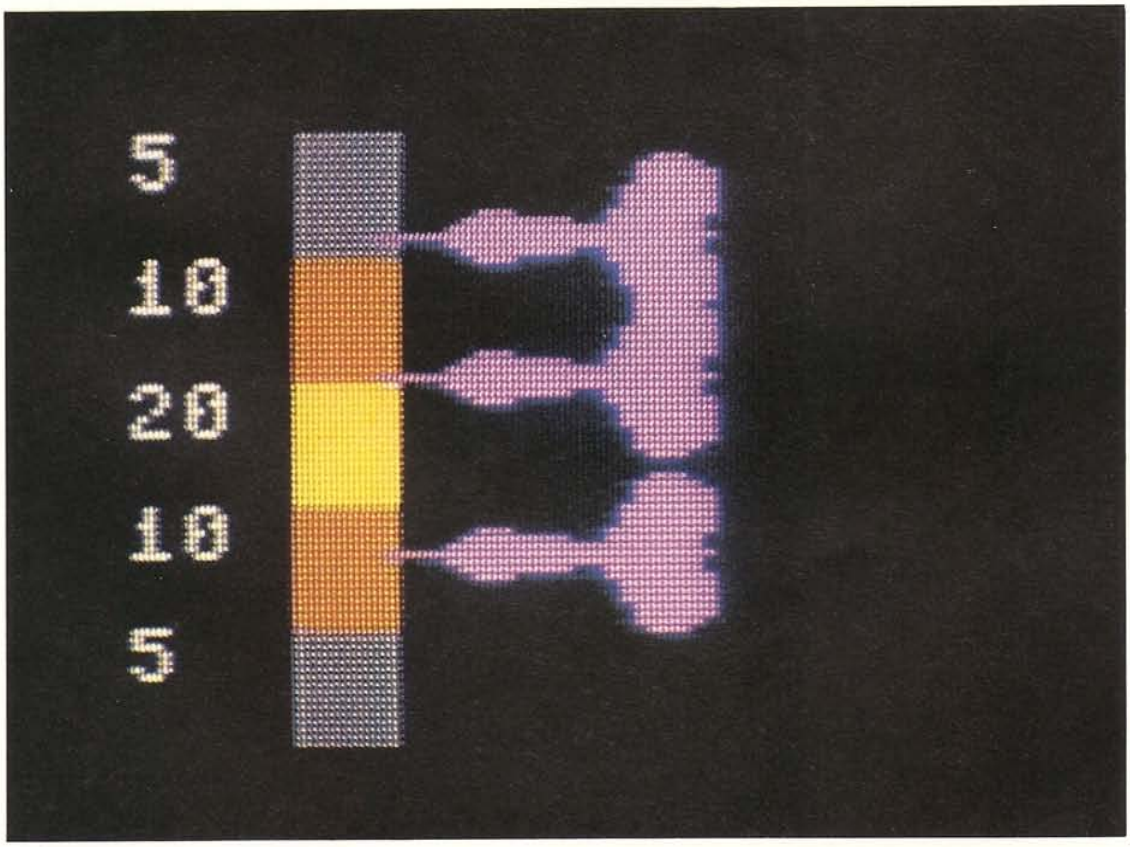
The Sprite Darts program is written so that you can control two characteristics of a sprite—its position and the speed with which it moves. Your input from the keyboard is monitored by line 220. The program then gives the variable A the character code of the key you have pressed, checks this number, and then either moves the sprite vertically, launches it, or ignores your input and starts again. The speed and trajectory of the sprite is calculated by lines 260-320. If you key in a number, it is used to produce a variable N with which the computer fixes each dart's vertical coordinates as it moves across the screen. When a dart eventually hits the target, its vertical position is used to decide what figure should be added to the total score. You can alter the basic speed of the darts and the range of scores by changing the values in these lines. You can also make the score accumulate for a set number of games.

SPRITE DARTS DISPLAY



SPRITE DARTS DISPLAY





SPRITE DARTS PROGRAM

How the program works

Three darts are produced in sequence from the same sprite DATA. You can throw them one by one at the target, controlling both throwing position and speed.

Lines 30-80 PRINT the dart board.

Lines 90-100 produce the darts.

Line 220 checks for key-presses.

Lines 240-250 control the darts.

Lines 260-300 work out the speed and trajectory for each dart.

Lines 340-400 calculate the score, which is displayed by line 160.

SPRITE GAMES 2

The program on the next four pages produces a game which uses single and multi-color sprites to simulate a fruit machine (one-armed bandit). The display uses a high-resolution background. To enable the computer to draw the background, you will first need to key in some of the machine-code routines shown on pages 60-61 (these are the same as the routines in Book Three). The routines you need to key in are listed in the panel below. Make sure that you have them in memory before you start keying in the program itself.

FRUIT MACHINE PROGRAM PART 1

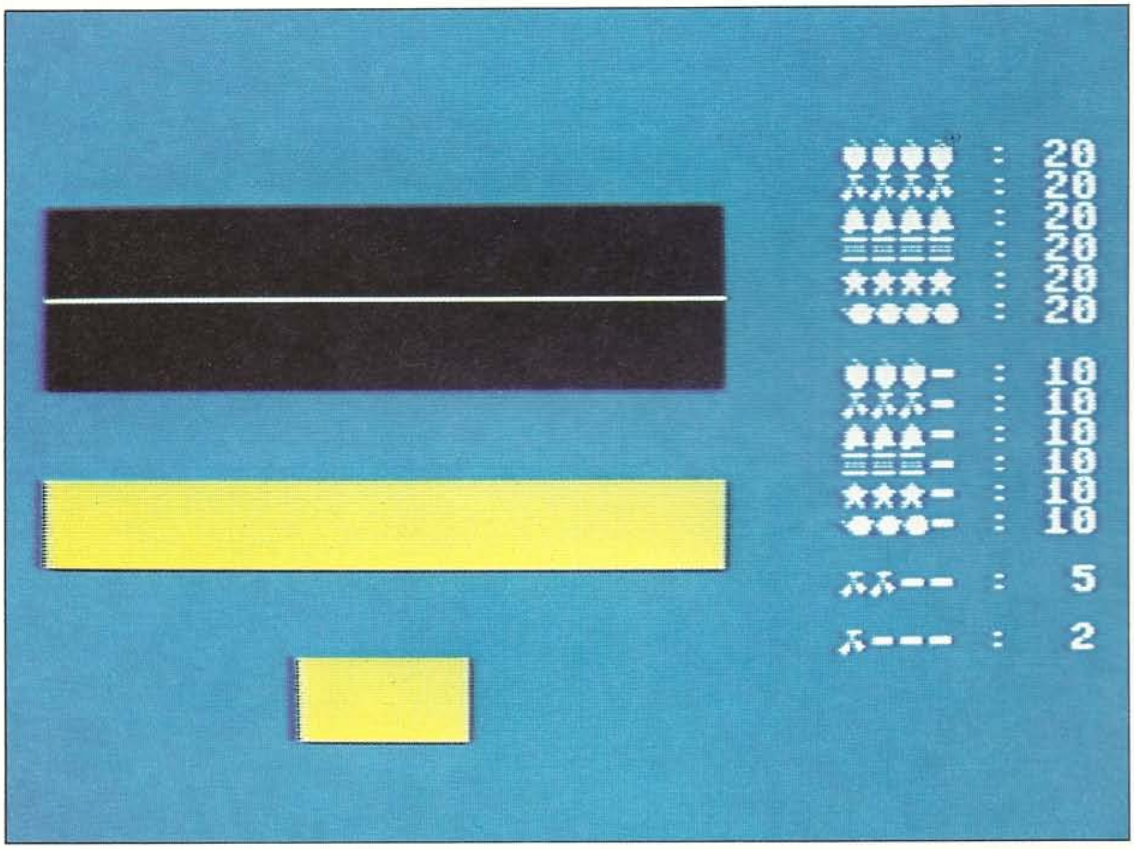
```
LIST 10000-10160

10000 SYS A1 : SYS B1,22
10010 POKE 53280,6 : SYS H1
10020 SYS A3,15000 : FOR C=1 TO 7
10030 READ P,Q,R,S,T,U,V,W
10040 SYS I1,83+C,P,Q,R,S,T,U,V,W
10050 NEXT C : V=53248
10060 LX=16 : LY=48 : UX=200 : UY=88
10070 C=16 : GOSUB 20000
10080 LX=16 : LY=120 : UX=200 : UY=136
10090 C=103 : GOSUB 20000
10100 LX=88 : LY=168 : UX=128 : UY=184
10110 C=103 : GOSUB 20000
10120 FOR C=0 TO 16 : READ A$
10130 SYS H2,240,8*C+32,A$
10140 NEXT C : SYS C1,16,72
10150 SYS D1,207,72 : SYS H1
10160 GOTO 10160

READY.
```

FRUIT MACHINE PROGRAM PART 1 (CONTD.)

```
LIST 15000-

15000 DATA 16,8,126,126,126,126,60,24
15010 DATA 0,60,16,16,44,110,228,64
15020 DATA 0,16,56,124,124,124,254,96
15030 DATA 0,254,0,84,84,0,254,0
15040 DATA 0,16,56,254,124,56,108,68
15050 DATA 0,0,30,191,127,63,30,0
15060 DATA 0,0,0,126,126,0,0,0
15100 DATA "TTTT : 20","UUUU : 20"
15110 DATA "VVVV : 20","WWWW : 20"
15120 DATA "XXXX : 20","YYYY : 20"," "
15130 DATA "TTTZ : 10","UUUZ : 10"
15140 DATA "VVVZ : 10","WWWZ : 10"
15150 DATA "XXXZ : 10","YYYZ : 10"," "
15160 DATA "UUZZ : 5"," "
15170 DATA "UZZZ : 2"
20000 FOR X=LX TO UX STEP 8
20010 FOR Y=LY TO UY STEP 8
20020 SYS B2,X,Y,C
20030 NEXT Y : NEXT X : RETURN
READY.
```

The game is written so that you can key in parts of it and test them as you go along. Part 1 of the program produces the outline display shown below. At this stage, there are no sprites and no scoring mechanism. When you have keyed this in and made sure that your copy works, store it on tape or disk for safety.

When you have a working copy of part 1, add part 2 to it. This contains the DATA for the six single or multi-color sprites shown at the bottom of the next page. Multi-color sprites need the same amount of DATA but



FRUIT MACHINE PROGRAM Part 1

How the program works

Part 1 of the program draws the fruit machine display with high-resolution graphics.

Lines 10020-10050 define seven characters.

Lines 10060-10110 use the block-color routine to create the display.

Lines 10120-10140 READ DATA to display the score table characters.

ROUTINES USED BY THIS PROGRAM

A High-resolution Restore
B Clear-and-color Block-color
C Plot
D Draw
H ROM-copy Text
I Define-character

use it in a different way, so that some pixel information is transferred to coding extra colors. You can see how to program your own multi-color sprites on page 62. The combined parts 1 and 2 won't actually show the sprites — that happens after you have keyed in part 3.

FRUIT MACHINE PROGRAM PART 2

```
LIST -15219

15200 DATA 0,64,0,0,16,0,0
15201 DATA 16,0,9,85,128,38,102
15202 DATA 96,42,102,160,174,170,232
15203 DATA 170,234,168,170,171,168,186
15204 DATA 186,168,171,171,184,170,186
15205 DATA 168,174,170,232,170,238,168
15206 DATA 186,170,184,42,187,160,43
15207 DATA 170,160,10,171,128,10,234
15208 DATA 128,2,170,0,0,168,0
15209 DATA 0
15210 DATA 0,5,64,1,85,64,1
15211 DATA 84,0,0,20,0,0,20
15212 DATA 0,0,20,0,0,17,0
15213 DATA 0,17,0,0,17,0,0
15214 DATA 17,0,0,64,64,0,64
15215 DATA 64,0,66,96,1,10,168
15216 DATA 9,138,184,42,170,184,42
15217 DATA 234,168,42,234,168,42,162
15218 DATA 160,42,160,128,10,128,0
15219 DATA 0

READY.
```

```
LIST 15220-15239

15220 DATA 0,24,0,0,24,0,0
15221 DATA 60,0,0,255,0,1,255
15222 DATA 128,1,255,128,3,255,192
15223 DATA 3,255,192,3,255,192,7
15224 DATA 255,224,7,255,224,7,255
15225 DATA 224,7,255,224,7,255,224
15226 DATA 11,255,208,28,126,56,63
15227 DATA 129,252,31,255,248,39,255
15228 DATA 224,56,255,0,0,0,0
15229 DATA 0
15230 DATA 0,0,0,255,255,255,255
15231 DATA 255,255,255,255,255,0,0
15232 DATA 0,120,24,120,100,36,100
15233 DATA 102,102,102,102,102,102,120
15234 DATA 126,120,100,102,120,102,102
15235 DATA 108,102,102,102,100,102,102
15236 DATA 120,102,102,0,0,0,255
15237 DATA 255,255,255,255,255,255,255
15238 DATA 255,0,0,0,0,0,0
15239 DATA 0

READY.
```

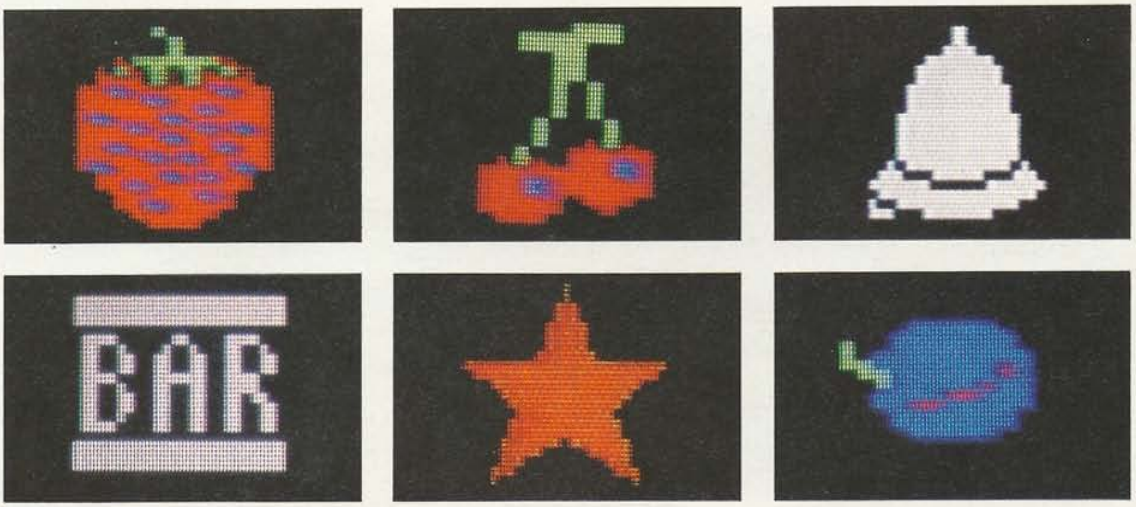
FRUIT MACHINE PROGRAM PART 2 (CONTD.)

```
LIST 15240-

15240 DATA 0,8,0,0,8,0,0
15241 DATA 28,0,0,28,0,0,62
15242 DATA 0,0,62,0,0,62,0
15243 DATA 0,127,0,127,255,255,63
15244 DATA 255,254,15,255,248,7,255
15245 DATA 240,3,255,224,1,255,192
15246 DATA 3,255,224,3,255,224,7
15247 DATA 247,240,7,193,240,7,0
15248 DATA 112,14,0,56,8,0,8
15249 DATA 0
15250 DATA 0,0,0,0,0,0,0
15251 DATA 0,0,0,0,0,0,255
15252 DATA 192,3,255,240,67,255,240
15253 DATA 79,255,252,95,255,236,23
15254 DATA 255,236,7,255,188,15,250
15255 DATA 252,15,175,252,3,255,240
15256 DATA 3,255,240,0,255,192,0
15257 DATA 0,0,0,0,0,0,0
15258 DATA 0,0,0,0,0,0,0
15259 DATA 0
READY.
```

Multi-color sprites

In the multi-color mode, the pixels that make up a sprite are programmed in horizontal pairs. Each of these pairs can be displayed in one of four colors. Because you cannot program each pixel in a pair separately, this means that there are effectively half the normal number of pixels, with each being twice as wide as usual. In multicolor sprites, two bits are used to code each pixel pair, instead of one bit being used to code one pixel. This gives a total of four different bit combinations for each pair. As well as specifying whether the pixel pair is turned on or off, there is enough information available to specify two extra states as well, and this extra capacity is used to code two supplementary colors. To produce multi-color sprites, first turn the multi-color facility on with POKE V+28 followed by a number from 0-255, to determine how many sprites are to be multi-color. You can now specify two extra colors with location V+37 and V+38. All you now have to do is key in the DATA. You can see how to do that on page 62.

Multi-color sprites
The fruit machine uses a total of six sprites – three are normal one-color sprites, while the rest are multi-color. The program so far includes only the DATA from which the sprites are made – the conclusion of the listing overleaf creates the sprites and turns on the multi-color mode.



SPRITE GAMES 3

You have already put together most of the programming required for the visual side of the game. Now you can add the final part of the program. Part 3 of the listing is concerned with the logic that actually drives the fruit machine, selecting the sprites to be displayed and working out your score. You should type part 3 onto the end of the program from pages 20-21.

To play the game, wait until the fruit machine has been set up complete with its row of four sprites and then just press RETURN. If the boxes under the display windows contain the word HOLD, then before the game has started you can select any or all of the reels to be held so that they do not change when the game is played. To set reels to be held, just press the number keys 1-4 corresponding to the reels. If you make a mistake, pressing the same keys again will cancel the HOLD on the selected reels. Now press RETURN again and the game will start as different sprites are displayed in the windows.

FRUIT MACHINE PROGRAM PART 3

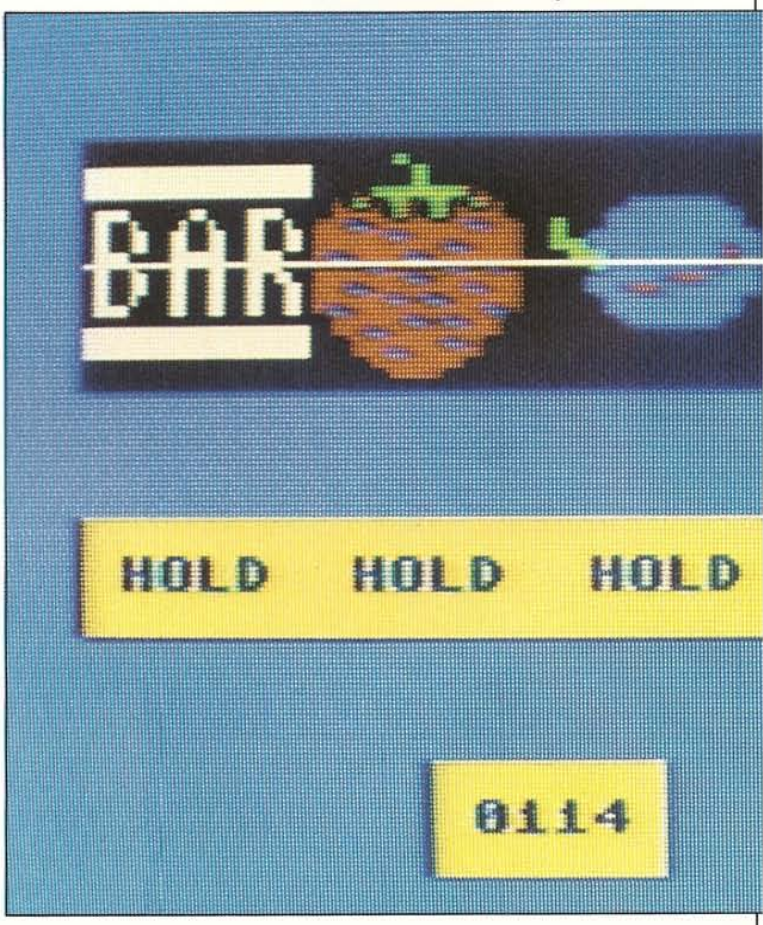
```
10160 FOR C=0 TO 383 : READ B
10170 POKE 2048+C,B : NEXT C
10180 POKE V+37,5 : POKE V+38,6
10190 FOR C=0 TO 3 : POKE V+2*C+1,102
10200 POKE V+2*C,48*C+40 : NEXT C
10210 POKE V+16,0 : POKE V+21,15
10220 POKE V+23,15 : POKE V+29,15
10230 POKE V+27,15 : FOR C=0 TO 5
10240 READ S(C,0),S(C,1),S(C,2)
10250 NEXT C : FOR C=0 TO 3
10260 R(C)=INT(RND(0)*6) : H(C)=0
10270 GOSUB 21000 : NEXT C
10280 CR=100 : SYS H2,96,176,"0100"
10290 IF INT(RND(0)*10)>5 THEN 10380
10300 SYS H2,24,128,"HOLD  HOLD  HOLD  H
OLD"
10310 GET A$ : IF A$="" THEN 10310
10320 IF A$=CHR$(13) THEN 10390
10330 IF A$<"1" OR A$>"4" THEN 10310
10340 A=ASC(A$)-49 : H(A)=1-H(A)
10350 IF H(A)=1 THEN SYS H2,48*A+31,128,
"****"
```

```
10360 IF H(A)=0 THEN SYS H2,48*A+31,128,
"HOLD"
10370 GOTO 10310
10380 GET A$ : IF A$<>CHR$(13) THEN 1038
0
10390 CR=CR-1 : IF CR<0 THEN 10590
10400 CR$=LEFT$(LEFT$("0000",4-INT(LOG(C
R+0.1)*0.4344+1))+MID$(STR$(CR),2),4)
10410 SYS H2,96,176,CR$ : Z=0
10420 FOR K=0 TO RND(0)*8+4
10430 FOR C=Z TO 3
10440 IF H(C)=1 THEN 10480
10450 R(C)=R(C)+1
10460 IF R(C)>5 THEN R(C)=0
10470 GOSUB 21000
10480 NEXT C : NEXT K
10490 Z=Z+1 : IF Z<4 THEN 10420
10500 IF R(0)=R(1) AND R(2)=R(3) AND R(1
)=R(2) THEN CR=CR+20 : GOTO 10540
10510 IF R(0)=R(1) AND R(1)=R(2) THEN CR
=CR+10 : GOTO 10540
```

FRUIT MACHINE PROGRAM PART 3 (CONTD.)

```
LIST 10520-

10520 IF R(0)=1 AND R(1)=1 THEN CR=CR+5
 : GOTO 10540
10530 IF R(0)=1 THEN CR=CR+2
10540 CR$=LEFT$(LEFT$("0000",4-INT(LOG(C
R+0.1)*0.4344+1))+MID$(STR$(CR),2),4)
10550 SYS H2,96,176,CR$
10560 SYS H2,24,128,"
                    "
10570 FOR C=0 TO 3 : H(C)=0 : NEXT C
10580 GOTO 10290
10590 SYS H2,72,16,"**********"
10600 GOTO 10600
15300 DATA 32,2,1,33,2,1
15310 DATA 34,12,0,35,1,0
15320 DATA 36,9,0,37,2,1
21000 POKE V+39+C,S(R(C),1)
21010 IF S(R(C),2)=1 THEN POKE V+28,PEEK
(V+28) OR 2↑C
21020 IF S(R(C),2)=0 THEN POKE V+28,PEEK
(V+28) AND (255-2↑C)
21030 POKE 2040+C,S(R(C),0) : RETURN
READY.
```

When you run the program, part 3 decides at random whether the hold facility is available or not. If it is available, you can specify from the keyboard which of the reels are to be held. It also ensures that the sprites in the marked windows stay unaltered while the others change. When you next press RETURN, the sprites in the windows that are not held will change at random, while the value of the credit window is reduced by one.



Each time you press RETURN, the sprites are altered and the new combination of four sprites in the display windows is checked for a winning line. If a win has occurred, the value in the credit window is increased by the amount shown in the score table. Finally, after each game, the value in the credit window is checked for a zero reading. If this value is detected, it means that you have run out of credit and the game has ended. Provided you are still in credit, this whole procedure will be repeated once again.

In this program, all six sprites are enlarged in both directions, and some of the sprites appear in the multi-color mode. The multi-colors are specified by line 10180.

FRUIT MACHINE PROGRAM

How the program works The program sets up the machine and selects four of its six sprites at random. These are displayed. Pressing the RETURN key then alters the sequence of sprites and your remaining credit.

Lines 10160-10170 POKE the DATA for six sprites into memory.

Line 10180 sets up two multi-colors.

Lines 10250-10260 select a random number from 0-5 four times.

Line 10300 produces the HOLD line.

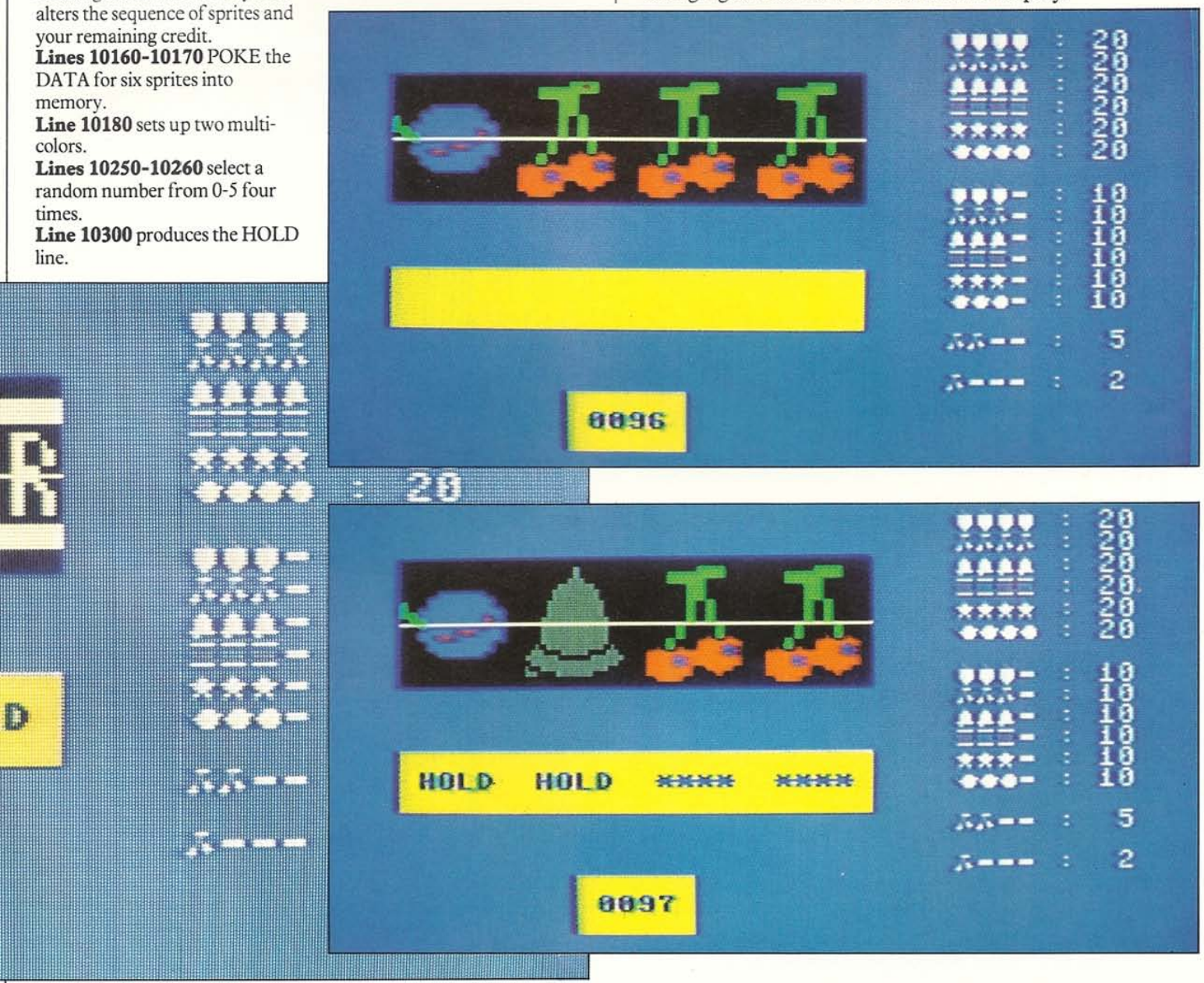
Line 10350 indicates which windows are being held.

Line 10410 indicates your current credit.

Lines 21000-21030 select and color the sprites.

Setting the odds

The program you have now completed produces a completely fair game — the odds of any winning combination coming up are just those dictated by change. If you want to, you can alter them so that the machine is more mean. There are a number of ways of doing this. The display selection uses the RND function which for practical purposes completely random. But it isn't difficult to make the computer reject some results which are potential wins. If the program detects a score-line, for example, you can make it carry out another sprite alteration so that the line loses, or you can make it reject one or more of the sprites a certain proportion of the time. This will give you results very like those with real machines, where sophisticated programming ensures that, on average, the machine wins. Your commodore fruit machine can be biased in just the same way! You can also alter the odds of a player winning by changing the likelihood of the HOLD option being activiated, thereby changing the amount of skill needed to play.



SPRITE EDITOR 1

Creating and displaying sprites on the Commodore is straightforward but at the same time quite time-consuming. The individual programming steps involved are simple, but together they produce listings that are difficult to adapt or debug. If you do program a sprite and then want to modify its design in some way, you will soon find yourself trying to unscramble a block of DATA, which is never an easy process.

Over the next seven pages, you can develop a program which provides an answer to these problems. It's a powerful Sprite Editor, a program which enables you to create sprites on a giant-sized grid, flip them horizontally or vertically, merge them with other sprites, store them on tape or disk and even convert them automatically to DATA which you can then use in your own programs.

In order to speed up its running time, the Sprite Editor uses some machine-code graphics routines. Like the programs in Book Three, the Sprite Editor is designed to be added to the end of these routines. Before you can run the Sprite Editor, you will need to load or key in the routine blocks on pages 60-61. **If you do not have these machine-code graphics routines in memory before you start adding the Sprite Editor listing, the editor will not work.**

How to key in the editor

The Sprite Editor is arranged so that you can build it up in small stages, testing each one as you go to make sure that it is correct. Before you start keying in part 1, make sure that you have moved the BASIC storage area (see page 6), and that you also have the machine-code graphics routines in memory. Finally, if you know how to use the merge routine from Book Three, remember not to use it here as the program's lines are not always added in strict numerical order.

Clearing memory and creating the grid

The first part of the Sprite Editor clears out a block of memory large enough to store 32 sprites. This clearing out process only takes place when you first run the program each time you switch on.

Next, the editor sets up a sprite bank, showing images of all the sprites currently in its 32-sprite block. It also produces a large grid which you will be able to use for designing your sprites when you have added some more parts of the listing.

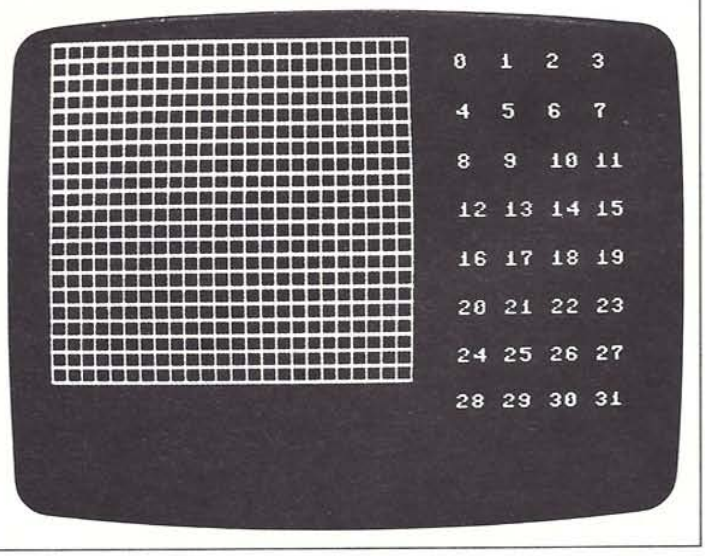
When you run this part of the program for the first time, all the sprites will be undefined. Rather than just showing a blank sprite bank, the editor displays sprite numbers from 0-31 in the 32 separate positions.

SPRITE EDITOR PART 1

```
10000 SYS A1 : SYS B1,16 : POKE 53280,0
10010 IF PEEK(52222)=165 THEN 10100
10020 SYS H1
10030 POKE 52222,165
10040 SYS H2,40,40,"**********"
10050 FOR C=7 TO 0 STEP -1
10060 SYS H2,120,40,STR$(C)
10070 FOR K=0 TO 255
10080 POKE 2048+C*256+K,0
10090 NEXT K : NEXT C : SYS B1,16
10100 SYS H2,40,40,"********SPRITE"
10110 FOR CY=0 TO 7 : FOR CX=0 TO 3
10120 C=CY*4+CX
10130 SYS H2,152,40,STR$(C)+"  "
10140 IF PEEK(2111+64*C)=0 THEN 10220
10150 FOR K=0 TO 62 : KY=INT(K/3)
10160 KX=K-KY*3
10170 X=216+CX*24+KX*8
10180 Y=8+CY*24+KY
10190 B=8192+INT(Y/8)*320+INT(X/8)*8+(Y
AND 7)

READY.
```

```
LIST 10200-

10200 POKE B,PEEK(2048+64*C+K)
10210 NEXT K : GOTO 10240
10220 SX=216+CX*24 : SY=16+CY*24
10230 SYS H2,SX,SY,STR$(C)
10240 NEXT CX : NEXT CY
10250 SYS H2,40,40,"                    "
10260 FOR X=8 TO 200 STEP 8
10270 SYS C1,X,8 : SYS D1,X,176
10280 NEXT X
10290 FOR Y=8 TO 176 STEP 8
10300 SYS C1,8,Y : SYS D1,200,Y
10310 NEXT Y : POKE 650,128
10320 GOTO 10320
READY.
```

PART 1 DISPLAY



The "current sprite" and the cursor

Part 2 of the program sets the current sprite number to 0. What this means is that the sprite which you can create on the large grid will be stored as the first sprite in the 32-sprite bank. Most of the work of part 2 is carried out by three subroutines at lines 20000, 22000 and 24000. These highlight a position in the sprite bank, copy the design onto the sprite grid and produce the cursor.

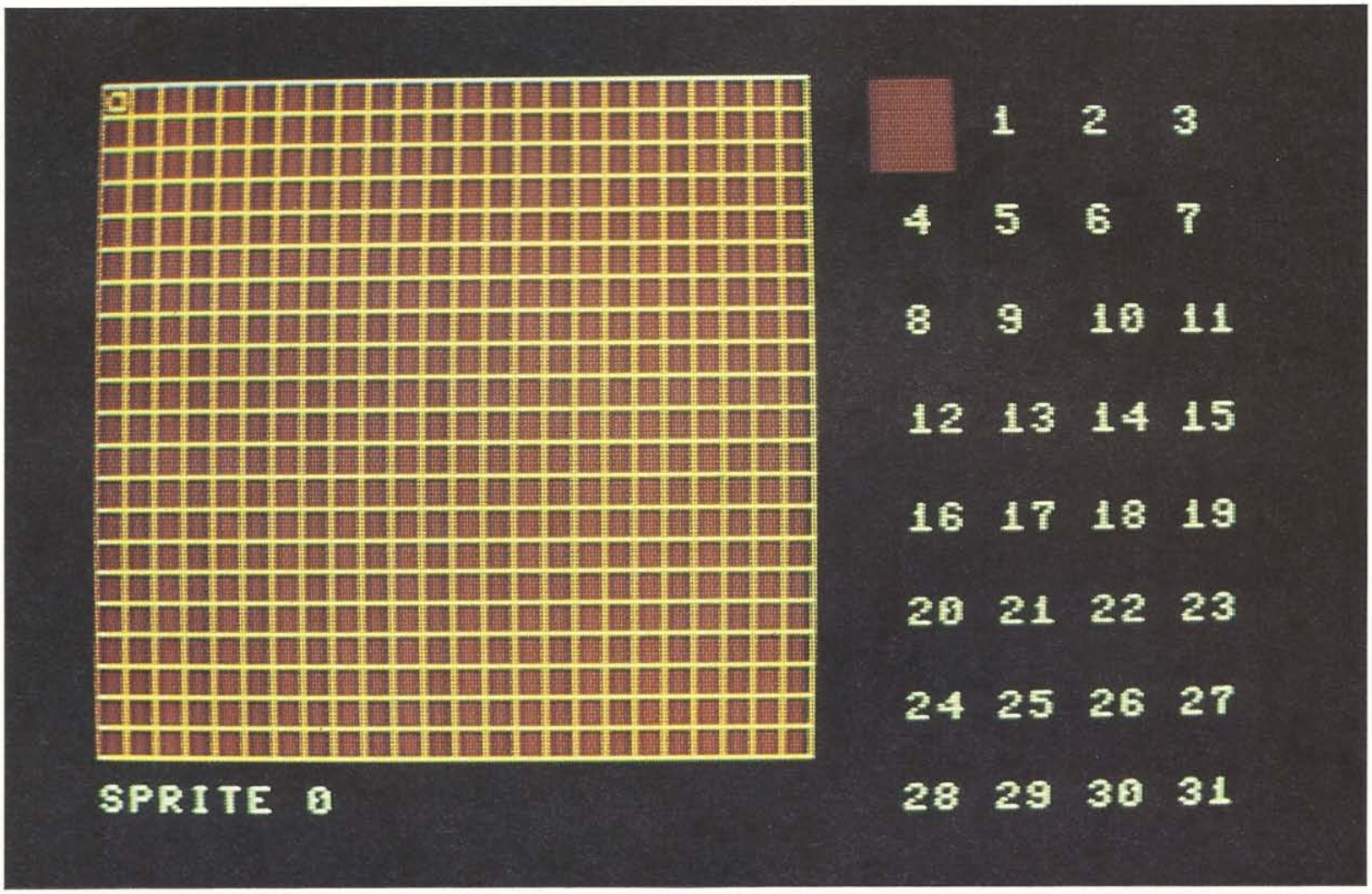
SPRITE EDITOR PART 2

```
10320 SN=0 : GOSUB 20000
10330 GOSUB 24000 : Z=2
10340 SYS H2,8,184,"SPRITE"+STR$(SN)+"
"
10350 CX=0 : CY=0 : GOSUB 22000
10360 GOTO 10360
20000 C=121
20010 SY=INT(SN/4) : SX=SN-SY*4
20020 LX=216+24*SX : LY=8+24*SY
20030 IF PEEK(2111+64*SN)<>0 THEN 20050
20040 SYS H2,LX,LY+8," "
20050 FOR X=LX TO LX+16 STEP 8
20060 FOR Y=LY TO LY+16 STEP 8
20070 SYS B2,X,Y,C
20080 NEXT Y : NEXT X : RETURN
22000 SYS F1,1
22010 LX=8+8*CX : LY=8+8*CY
22020 SYS C1,LX+2,LY+2
22030 SYS D1,LX+6,LY+2
22040 SYS D1,LX+6,LY+6
22050 SYS D1,LX+2,LY+6
22060 SYS D1,LX+2,LY+2

READY.
```

SPRITE EDITOR PART 2 (CONTD.)

```
LIST 22070-

22070 SYS F1,0 : RETURN
24000 IF PEEK(2111+64*SN)=0 THEN 24100
24010 B=2048+64*SN-1
24020 FOR Y=0 TO 20
24030 FOR X=0 TO 16 STEP 8 : B=B+1
24040 FOR W=0 TO 7 : M=2↑(7-W)
24050 LX=8+8*(X+W) : LY=8+8*Y
24060 IF (PEEK(B) AND M)=0 THEN 24080
24070 SYS B2,LX,LY,24 : GOTO 24090
24080 SYS B2,LX,LY,25
24090 NEXT W : NEXT X : NEXT Y : RETURN
24100 FOR X=8 TO 192 STEP 8
24110 FOR Y=8 TO 168 STEP 8
24120 SYS B2,X,Y,25
24130 NEXT Y : NEXT X : RETURN
READY.
```

When you run parts 1 and 2 for the first time, the program will simply highlight the first space in the sprite bank and produce a cursor in the top left of the grid. Later, when you run the complete program, the design held in memory for sprite 0 will appear in position 0 in the sprite bank, and the whole sprite will appear greatly enlarged on the grid. The cursor shows the place where sprite editing instructions will be carried out. At this stage, you cannot move it.



SPRITE EDITOR 2

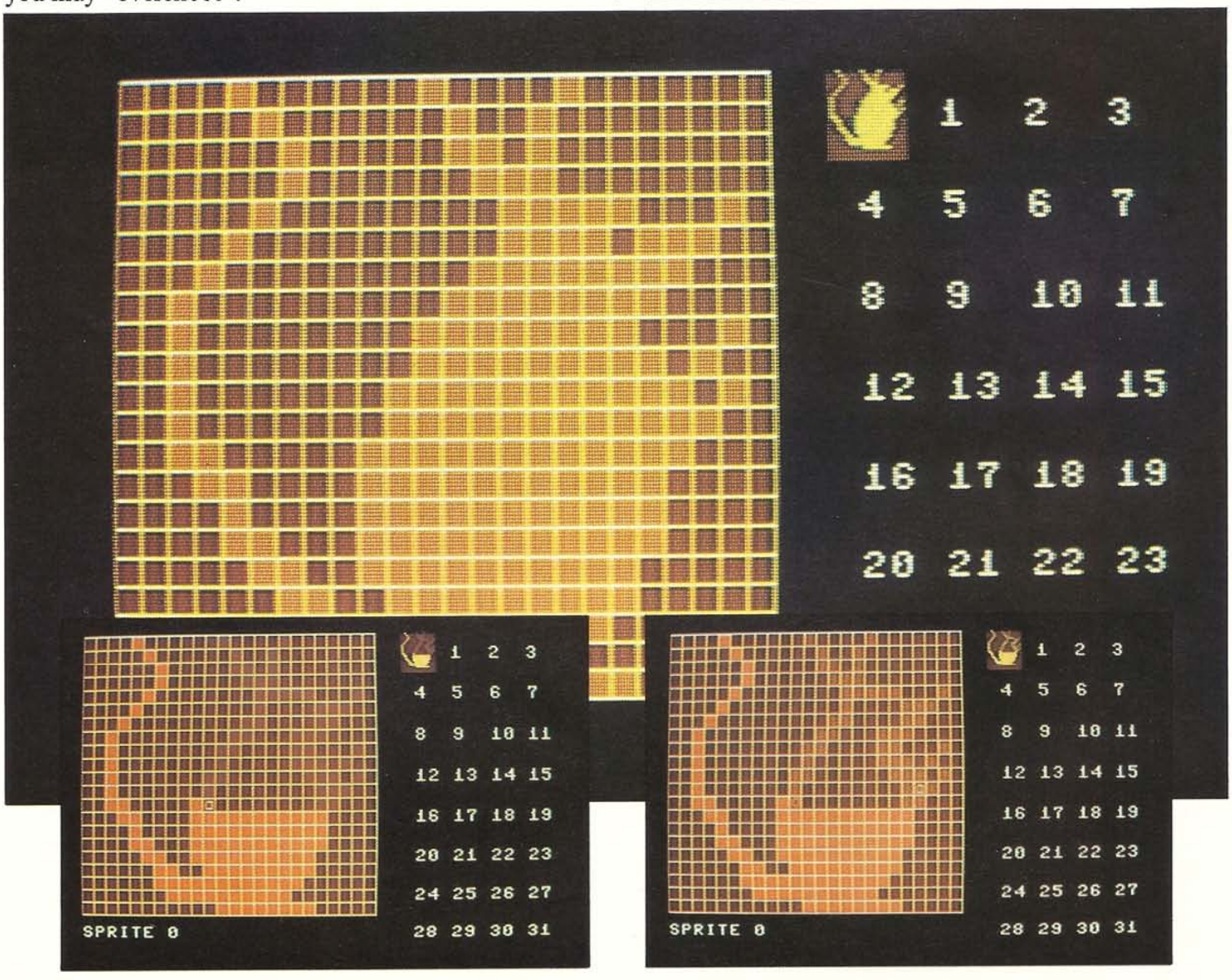
With these preliminaries completed, you can add part 3 of the program. This allows you to move the cursor around the sprite grid and make it leave a trail of lit or unlit pixels.

The cursor on the grid is moved with the usual cursor keys, but in addition, it responds to three other keys. If you press the + key, the cursor will leave a trail of lit pixels after it when it is moved. Pressing the – key makes the cursor leave a trail of unlit pixels, while pressing the * key lets the cursor move over anything already on the grid without changing it. These commands make use of a new subroutine at line 23000. As you draw out your design, it will also appear in the highlighted square in the sprite bank.

When you try out parts 1-3 you will find that key autorepeat is activated. When you are designing sprites, be careful not to hold any cursor keys down for too long or you may "overshoot".

SPRITE EDITOR PART 3

```
10360 GET A$ : IF A$="" THEN 10360
10370 IF A$<>"+" THEN 10400
10380 POKE 2111+64*SN,1
10390 Z=1 : GOTO 10360
10400 IF A$<>"-" THEN 10420
10410 Z=0 : GOTO 10360
10420 IF A$<>"*" THEN 10440
10430 Z=2 : GOTO 10360
10440 IF A$<>CHR$(145) THEN 10490
10450 IF CY=0 THEN 10360
10460 GOSUB 22000 : GOSUB 23000
10470 CY=CY-1 : GOSUB 22000
10480 GOTO 10360
10490 IF A$<>CHR$(17) THEN 10540
10500 IF CY=20 THEN 10360
10510 GOSUB 22000 : GOSUB 23000
10520 CY=CY+1 : GOSUB 22000
10530 GOTO 10360
10540 IF A$<>CHR$(157) THEN 10590
10550 IF CX=0 THEN 10360
10560 GOSUB 22000 : GOSUB 23000
10570 CX=CX-1 : GOSUB 22000
10580 GOTO 10360
```



SPRITE EDITOR PART 3 (CONTD.)

```
10590 IF A$<>CHR$(29) THEN 10640
10600 IF CX=23 THEN 10360
10610 GOSUB 22000 : GOSUB 23000
10620 CX=CX+1 : GOSUB 22000
10630 GOTO 10360
10640 GOTO 10360
23000 IF Z=2 THEN RETURN
23010 LX=8+8*CX : LY=8+8*CY
23020 B=2048+64*SN+INT(CX/8)+CY*3
23030 M=2↑(7-(CX AND 7))
23040 IF Z=0 THEN 23080
23050 SYS B2,LX,LY,24
23060 POKE B,PEEK(B) OR M
23070 GOTO 23100
23080 SYS B2,LX,LY,25
23090 POKE B,PEEK(B) AND (255-M)
23100 SY=INT(SN/4) : SX=SN-SY*4
23110 LX=216+SX*24+CX
23120 LY=8+SY*24+CY
23130 SYS C1,LX,LY
23140 IF Z=1 THEN RETURN
23150 SYS F1,1 : SYS C1,LX,LY
23160 SYS F1,0 : RETURN
READY.
```

When you have keyed in and tested parts 1 to 3 of the sprite editor, store a copy on tape or disk so that what you have written so far is safe.

How to clear the current sprite

Now that you can create and edit a sprite, you can add some new commands to make the editor more useful. With a copy of the program so far (parts 1-3) in memory, add part 4.

When you run parts 1-4, you should find that you can now clear the current sprite, making the grid blank again. This clears three things—the main grid, the display in the sprite bank and the area of memory occupied by the current sprite. This is activated by pressing the C key. Because this is something that could easily be done by accident, the program has a security device. It asks you to confirm that you really do want to clear the sprite. You can do this by keying in the number 1 in answer to the question-mark that is displayed. The sprite will then be cleared.

SPRITE EDITOR PART 4

```
LIST

10640 IF A$<>"C" THEN 10790
10650 SYS H2,88,184,"**??**"
10660 GET A$ : IF A$="" THEN 10660
10670 IF A$<>"1" AND A$<>"0" THEN 10660
10680 SYS H2,88,184,"          "
10690 IF A$="0" THEN 10360
10700 FOR C=0 TO 63
10710 POKE 2048+SN*64+C,0
10720 NEXT C : GOSUB 22000
10730 SY=INT(SN/4) : SX=SN-SY*4
10740 LX=216+24*SX : LY=8+24*SY
10750 FOR X=LX TO LX+16 STEP 8
10760 FOR Y=LY TO LY+16 STEP 8
10770 SYS H2,X,Y," " : NEXT Y
10780 NEXT X : GOTO 10330
10790 GOTO 10360
READY.
```

How to change the current sprite

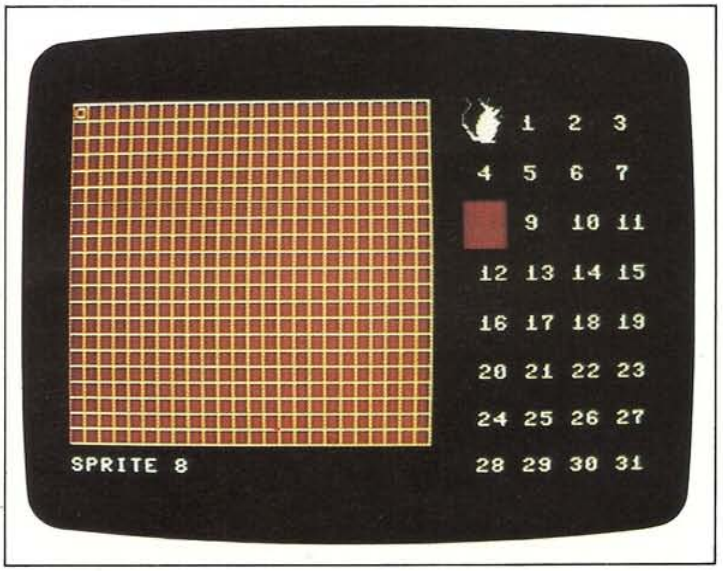
So far, all your designs have been stored as sprite number 0. The next part of the program lets you change the current sprite number so that the program stores the current sprite design and then lets you move on from there to create another one.

After you have added part 5 of the program, pressing the W key will produce the message SPRITE#?? on the screen. This is asking for the new current sprite number. If you key in a number from 0-31 and press RETURN, the computer will highlight that sprite in the sprite bank and draw it on the grid. If the number you enter is not within the correct range, your entry will be ignored and you will be re-prompted to enter a valid number.

When you change current sprites, the previous current sprite will no longer be highlighted. The subroutine at line 21000 is used by the program to move the highlight in the sprite bank. You can see in the display below the effect of designing sprite 0, and then changing the current sprite number to 8. You can change the current sprite number as frequently as you want.

SPRITE EDITOR PART 5

```
10790 IF A$<>"W" THEN 10900
10800 SYS H2,88,184,"SPRITE #??"
10810 N=0
10820 GET A$ : IF A$="" THEN 10820
10830 A=ASC(A$) : IF A=13 THEN 10860
10840 IF A<48 OR A>57 THEN 10820
10850 N=N*10+A-48 : GOTO 10820
10860 SYS H2,88,184,"          "
10870 IF N<0 OR N>31 THEN 10800
10880 GOSUB 22000 : GOSUB 21000 : SN=N
10890 GOSUB 20000 : GOTO 10330
10900 GOTO 10360
21000 C=16
21010 FOR X=8 TO 192 STEP 8
21020 FOR Y=0 TO 168 STEP 8
21030 SYS B2,X,Y,16
21040 NEXT Y : NEXT X
21050 SY=INT(SN/4) : SX=SN-SY*4
21060 LX=216+24*SX : LY=8+24*SY
21070 IF PEEK(2111+64*SN)<>0 THEN 20050
21080 SYS H2,LX,LY+8,STR$(SN)
21090 GOTO 20050
READY.
```



SPRITE EDITOR 3

Now you can move on to some of the more advanced features of the editor. Part 6 allows you to turn the current sprite upside down by pressing H. Doing this requires the computer to carry out a considerable amount of calculating, so be prepared to wait some seconds before the inverted sprite appears.

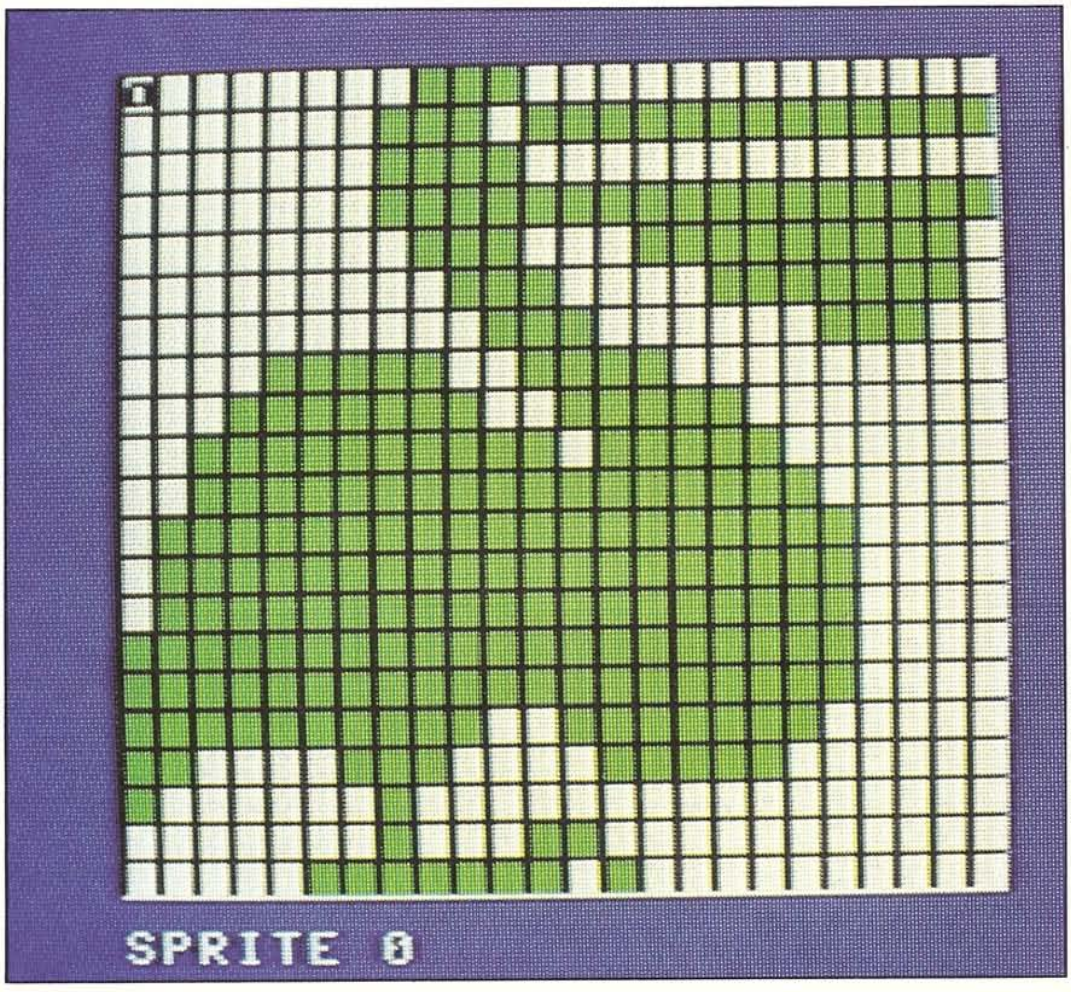
SPRITE EDITOR PART 6

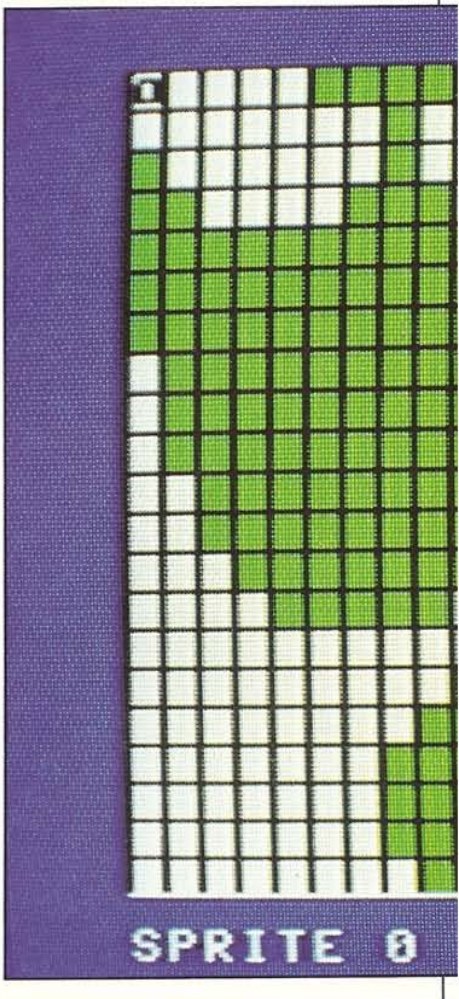
```
LIST

10900 IF A$<>"H" THEN 11070
10910 IF PEEK(2111+64*SN)=0 THEN 10360
10920 GOSUB 21000
10930 FOR X=0 TO 2 : FOR Y=0 TO 9
10940 BP=2048+64*SN+Y*3+X
10950 BQ=2048+64*SN+(20-Y)*3+X
10960 T=PEEK(BP) : POKE BP,PEEK(BQ)
10970 POKE BQ,T : NEXT Y : NEXT X
10980 FOR K=0 TO 62 : KY=INT(K/3)
10990 KX=K-KY*3 : SY=INT(SN/4)
11000 SX=SN-SY*4 : Y=8+SY*24+KY
11010 X=216+SX*24+KX*8
11020 B=8192+INT(Y/8)*320+INT(X/8)*8+(Y
AND 7)
11030 POKE B,PEEK(2048+64*SN+K)
11040 NEXT K
11050 GOSUB 24000
11060 GOSUB 20000 : GOTO 10360
11070 GOTO 10360
READY.
```

Part 7 of the program is complementary to the last one—a reflection in a vertical plane. Because of the way sprites are stored, you will find that this command takes a little longer to be carried out than the inversion. It's activated by pressing the V key. You can see parts 6 and 7 in action below.

SPRITE EDITOR PART 7

```
LIST

11070 IF A$<>"V" THEN 11190
11080 IF PEEK(2111+64*SN)=0 THEN 10360
11090 GOSUB 21000
11100 FOR Y=0 TO 20 : X=2048+64*SN+Y*3
11110 X0=PEEK(X) : X1=PEEK(X+1)
11120 X2=PEEK(X+2)
11130 N=X1 : GOSUB 25000 : X1=N
11140 N=X0 : GOSUB 25000 : T=N
11150 N=X2 : GOSUB 25000 : X0=N
11160 X2=T : POKE X+2,X2
11170 POKE X+1,X1 : POKE X,X0
11180 NEXT Y : GOTO 10980
11190 GOTO 10360
25000 NN=0 : FOR KK=1 TO 8
25010 N=N/2 : NN=NN*2
25020 IF N<>INT(N) THEN NN=NN+1
25030 N=INT(N) : NEXT KK
25040 N=NN : RETURN
READY.
```



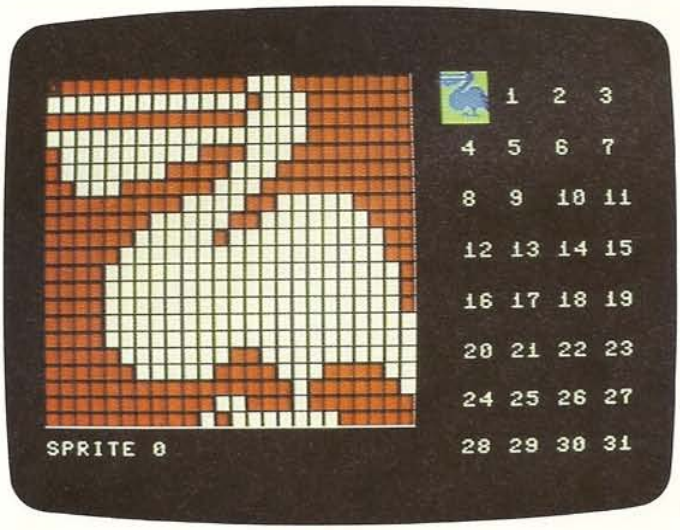


How to change colors

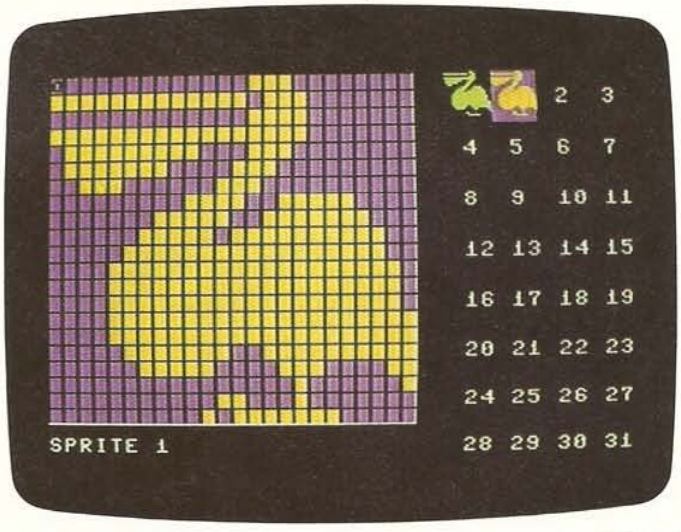
Although the Sprite Editor colors are initially set, you can change them if you want to. The coloring is quite complex because some parts of the display, like the sprite shown in the main grid, can be produced by more than one section of the program. Each section has its own color controls. For example, the main sprite is drawn by one subroutine when it is taken directly from memory, but it is drawn by another when you use the cursor. In each case, there are separate color controls.

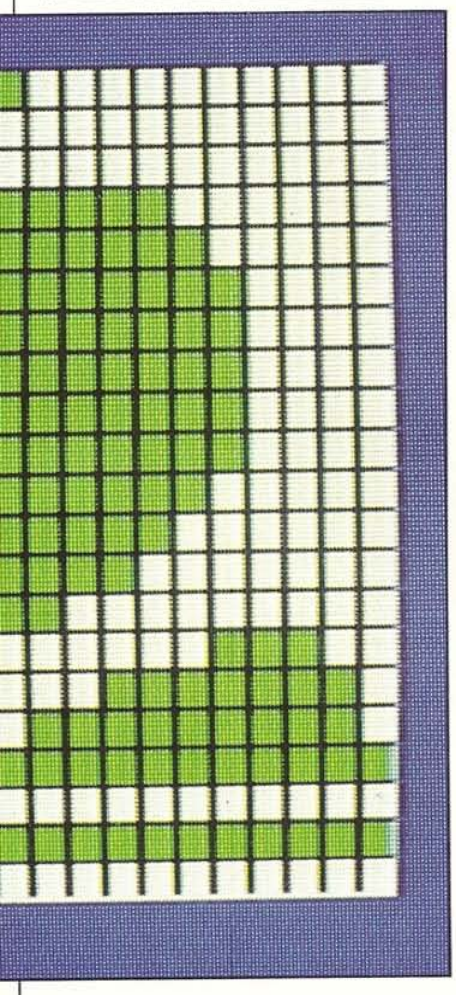
All this gives you the opportunity to produce a large number of different color combinations. If you want to test some out, try experimenting with the block-color settings in lines 21030, 23050, 23080, 24070, 24080 and 24120. You can also change colors in the sprite bank by altering the value of C in lines 20000 and 21000. The two displays below show just two alternative color combinations which you can create.

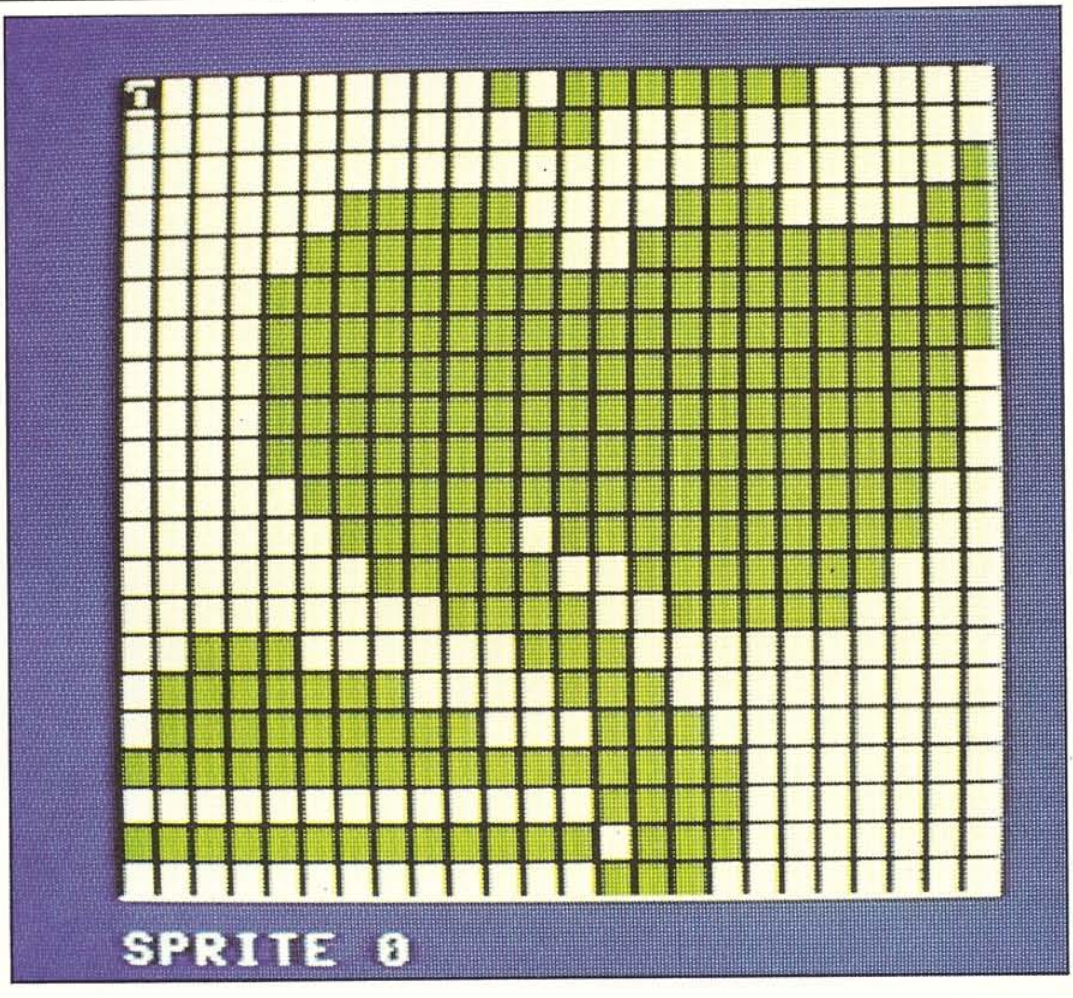
ALTERED COLOR DISPLAY



ALTERED COLOR DISPLAY







SPRITE EDITOR 4

Part 8 of the editor allows you to make more use of the inversion and reflection facilities on pages 28-29. It programs a new function which is activated by the M key. This allows you to merge the pixel pattern of any specified sprite in the sprite bank with that of the current sprite. This means that you can design one sprite and then add another to it.

Merging sprites is useful in two ways. You can use it to create symmetrical sprites by designing just one half, reflecting it, and then merging the two halves. You can also use it in cartooning if you want to add a detail to an initial sprite design.

SPRITE EDITOR PART 8

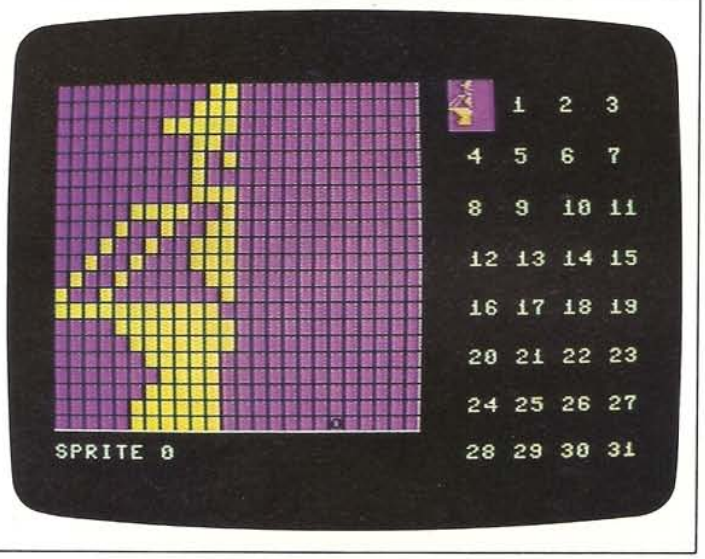
```
LIST

11190 IF A$<>"M" THEN 11330
11200 SYS H2,88,184,"SPRITE #??"
11210 N=0
11220 GET A$ : IF A$="" THEN 11220
11230 A=ASC(A$) : IF A=13 THEN 11260
11240 IF A<48 OR A>57 THEN 11220
11250 N=N*10 +A-48 : GOTO 11220
11260 SYS H2,88,184,"
11270 IF N<0 OR N>31 THEN 11220
11280 GOSUB 21000
11290 FOR K=0 TO 63
11300 BP=2048+64*SN+K : BQ=2048+64*N+K
11310 POKE BP,PEEK(BP) OR PEEK(BQ)
11320 NEXT K : GOTO 10980
11330 GOTO 10360
READY.
```

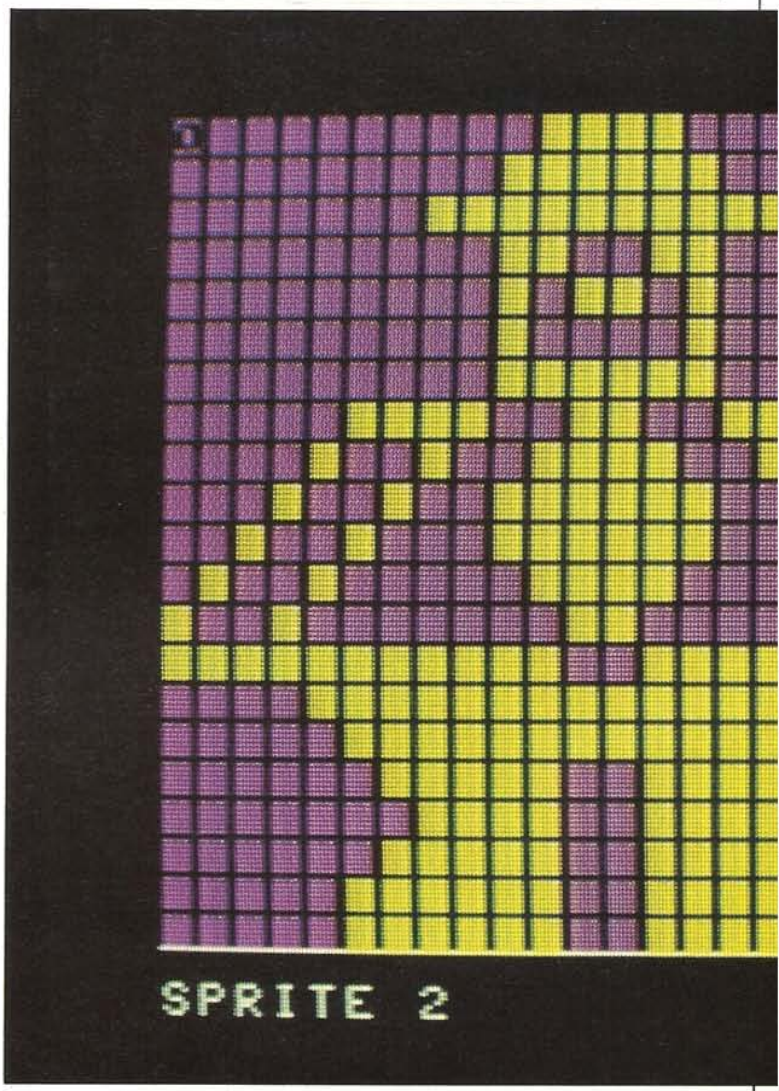
When you press the M key, the editor will ask for the number of the sprite to be merged with the current sprite, by giving the message SPRITE#??. In response to this prompt, enter a number in the range from 0-31 followed by RETURN (if your entry is out of range, it will be ignored and you will be prompted again). There will then be a pause as the computer combines the two sprites in its memory, and then the current working sprite will gradually be overprinted to give the merged design. This is displayed both on the large grid and in the sprite bank. The sprite merged with the current sprite is left unaltered.

Making symmetrical sprites

One particularly effective use of the merge facility is in the creation of symmetrical sprites. Suppose you want to create a symmetrical shape. The easiest way to do this is to create half of it, say the left half, then change the current sprite number to an undefined sprite. You then merge the left-hand display with this "empty" sprite to give another copy of the left-hand display. Now, using the V key, you can reflect this second left-hand half to give the right-hand half. Having done this, you can then merge the right-hand half with the original left-hand half to give the complete symmetrical design which is certain to be accurate.

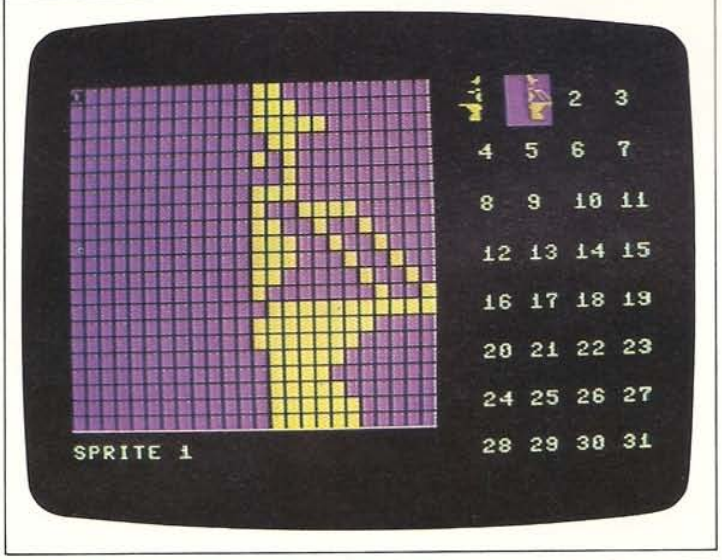
REFLECTING AND MERGING A SPRITE





The displays on these two pages show the sequence in action. The first small display on the left is the original left-hand half (the only part which you actually have to design). The small display on the right is the reflected half, and the large display below them is the result of merging the two.

REFLECTING AND MERGING A SPRITE



Automatic sprite DATA

So far the editor enables you to create and modify sprites, but it doesn't help you to produce sprite DATA which could be used in your own programs. This is the job of part 9 of the listing.

When you have added part 9 to parts 1-8, you will find that pressing the D key switches the Commodore back to low resolution and PRINTs a series of DATA statements. These are all the DATA for the current sprite, automatically presented as numbered program lines. At this point, the editor comes to a halt with the cursor positioned at the beginning of the pre-PRINTed word RUN.

To store the DATA as part of the Sprite Editor, just move the cursor to the first DATA line and press RETURN, repeating this for each line. To store it as a separate program, type NEW over the place of RUN and then enter the lines as before. With the DATA in memory, you can now use the numbers with any other program. Alternatively, if you do not want to store the DATA, just press RETURN to restart the editor.

SPRITE EDITOR PART 9

```
LIST

11330 IF A$<>"D" THEN 11500
11340 SYS A2
11350 PRINT CHR$(147)
11360 POKE 214,2
11370 PRINT : POKE 211,0
11380 PRINT "RUN"
11390 POKE 214,5 : PRINT : POKE 211,0
11400 FOR K=0 TO 8
11410 PRINT 15000+10*SN+K;"DATA ";
11420 FOR C=0 TO 6
11430 N=PEEK(2048+64*SN+7*K+C)
11440 PRINT MID$(STR$(N),2);
11450 IF C<>6 THEN PRINT ",";
11460 NEXT C : PRINT : NEXT K
11470 PRINT 15009+10*SN;"DATA 0"
11480 POKE 214,0 : PRINT : POKE 211,0
11490 END
11500 GOTO 10360
READY.
```

```
READY.
RUN

30020 DATA 0,60,0,0,126,0,1
30021 DATA 255,128,0,102,0,0,90
30022 DATA 0,0,66,0,0,126,0
30023 DATA 7,153,224,0,60,144,18
30024 DATA 126,72,36,126,36,72,60
30025 DATA 18,144,24,9,255,231,255
30026 DATA 15,255,240,7,255,224,3
30027 DATA 231,192,1,231,128,3,231
30028 DATA 192,7,231,224,7,231,224
30029 DATA 0
```

SPRITE EDITOR 5

The final Sprite Editor facility lets you SAVE and LOAD the memory copy of a 32-sprite bank onto tape or disk. The SAVE and LOAD commands are activated using the S and L keys respectively. In both cases, you will be asked to specify tape or disk and then to specify a filename. Do this by answering the first question with 1 (for tape) or 2 (for disk), and then by keying in a filename in response to the second question $=?.

SPRITE EDITOR PART 10

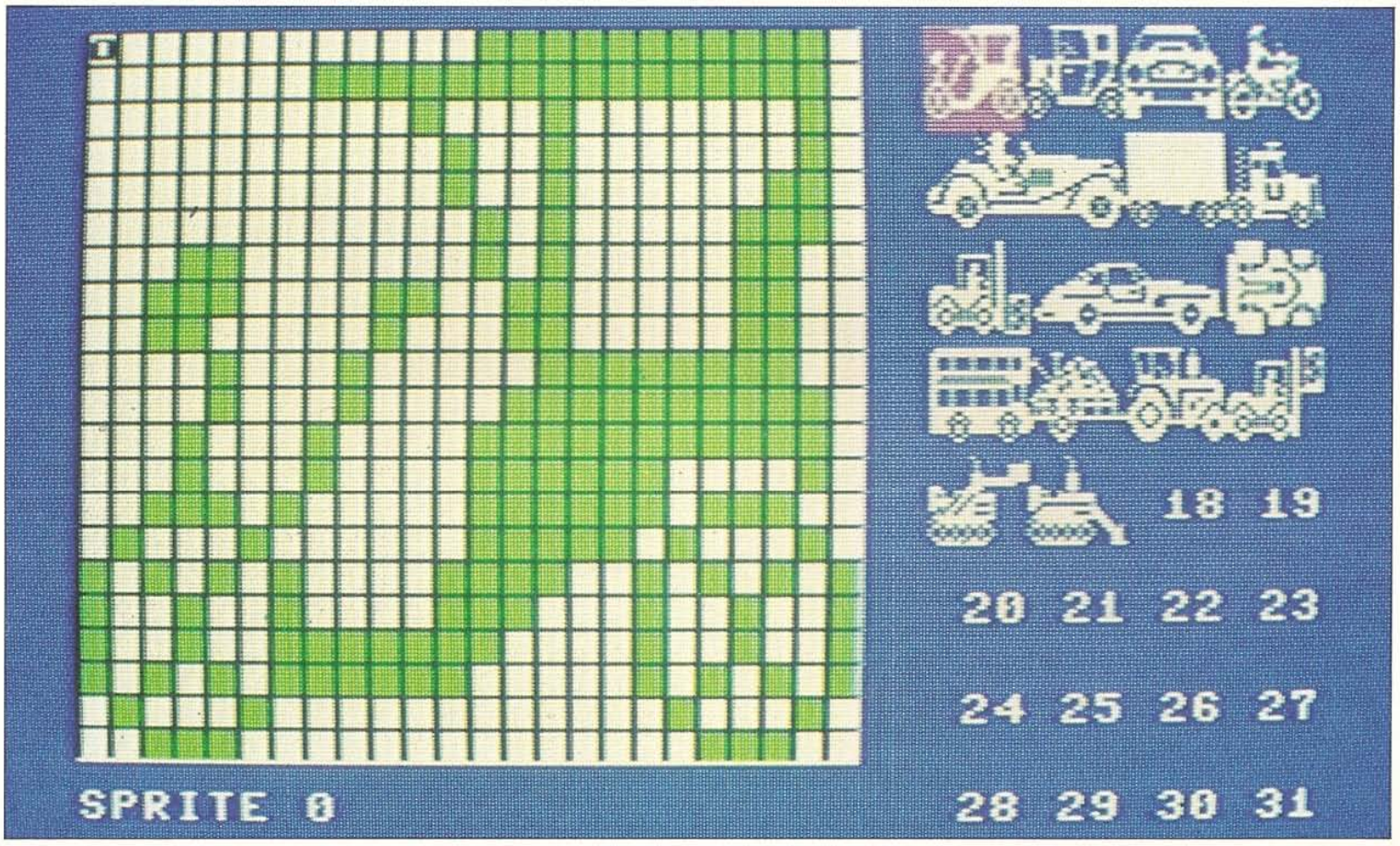
```
11500 IF A$<>"S" AND A$<>"L" THEN 10360
11510 SYS A2 : PRINT CHR$(147)
11520 PRINT "1/2 ";
11530 INPUT B$ : DEV=0
11540 PRINT
11550 IF B$="1" THEN DEV=1
11560 PRINT "$= ";
11570 INPUT F$
11580 IF A$="S" THEN 11660
11590 IF DEV=0 THEN 11610
11600 OPEN 1,1,0,F$ : GOTO 11620
11610 OPEN 1,8,2,"0:"+F$+",S,R"
11620 FOR C=2048 TO 4095
11630 GET#1,A$
11640 POKE C,ASC(A$+CHR$(0))
11650 NEXT C : CLOSE 1 : GOTO 10000
11660 IF DEV=0 THEN 11680
11670 OPEN 1,1,1,F$ : GOTO 11690
11680 OPEN 1,8,2,"0:"+F$+",S,W"
11690 FOR C=2048 TO 4095
11700 PRINT#1,CHR$(PEEK(C));
11710 NEXT C : CLOSE 1 : GOTO 10000
READY.
```

Two extra lines send DATA to a Commodore printer:

```
11375 OPEN 4,4 : CMD 4
11445 CLOSE 4
```

SPRITE EDITOR CONTROL KEYS

- ↑ Moves cursor up
- ↓ Moves cursor down
- ← Moves cursor left
- → Moves cursor right
- \+ Switches on design mode
- \- Switches off design mode
- * Switches on neutral mode
- C Clears current sprite
- W Changes current sprite number
- H Reflects current sprite in horizontal plane
- V Reflects current sprite in vertical plane
- M Merges current sprite with another
- D Converts current sprite to DATA
- S Saves current sprite bank
- L Loads named sprite bank



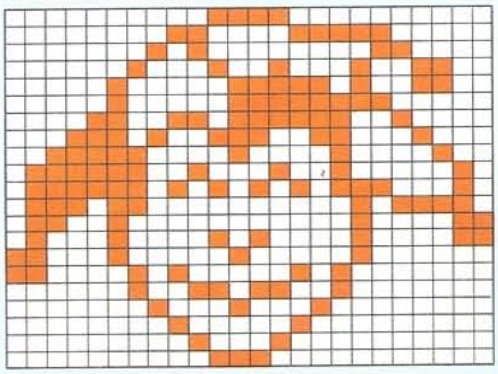
USING THE SPRITE DIRECTORY

During the course of this book, you have probably found that producing good sprite designs is not always easy. To get around this problem, you can turn to the Sprite Directory, a bank of over 200 sprite designs which makes up the next section of this book. The sprites in the Directory have been specially created for use in games and other programs and they have been designed so that you can either copy them directly, or you can use them as a basic idea which you can then develop. The Directory shows you what each sprite looks like on the screen, and what DATA numbers are needed to code it.

DIRECTORY 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
0
```



Single and double sprites

The Directory contains two types of sprites mixed together under theme headings. There are single sprites, ones which are designed to be used individually, and double sprites, pairs that are designed to be used together.

To put any of these sprites into memory, you can either key in the DATA shown, using it as part of a program, or you can use the Sprite Editor and key in the design directly. This second method lets you produce the DATA automatically. Once you have a sprite design in memory, you can recall it and get it moving on the screen.

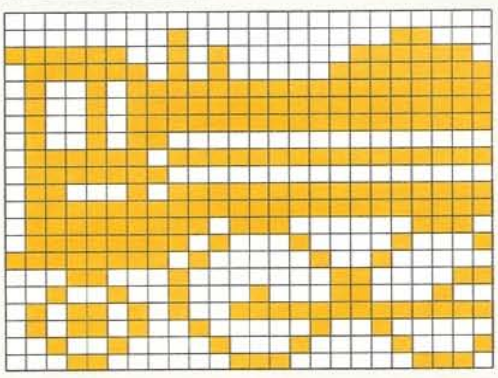
You can see how to animate single sprites on pages 10. If you want to animate double sprites, you will need to program their coordinates so that they move in step. You can see how to do this on page 11. If you want to use any of these designs as multi-color sprites, you will need to use the mutli-color table on page 62 and the sprite grid on page 63.

Cartoon sprites

The Directory contains a number of cartoon sequences, single sprites shown in three different positions. These are designed so that you can use them in sprite cartoons. Again, you can either key in the DATA numbers shown, or you can use the Sprite Directory to put the design into DATA numbers. If you want to increase the number of frames in one of these sequences, use the Sprite Editor and adapt the designs so that you have intermediate stages. This will give smoother cartooning, although the figure's speed across the screen will be reduced. You can have a maximum of 33 frames with single sprite.

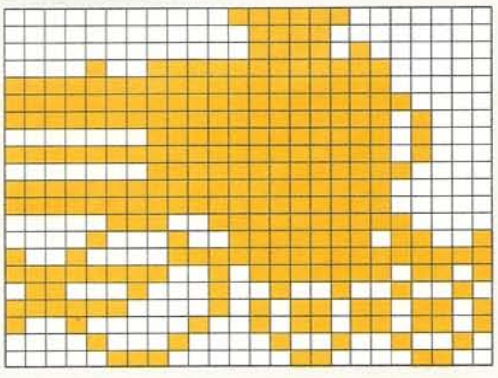
PACIFIC-TYPE LOCO

```
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
```



PACIFIC-TYPE LOCO

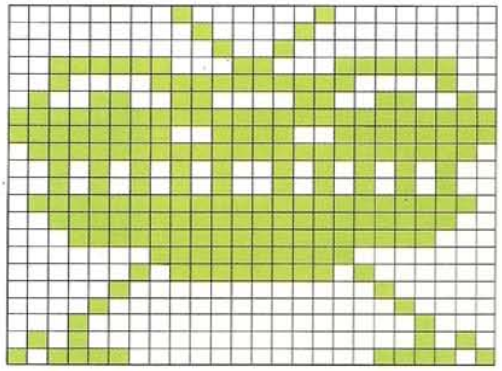
```
0,31,128,0,15,0,0
15,64,9,255,224,255,255
224,255,255,240,255,255,232
1,255,232,255,255,232,1
255,240,255,255,224,255,255
224,7,63,240,8,159,217
144,255,255,255,47,237,98
36,146,252,43,109,144,75
109,8,132,146,7,3,12
0
```



DOUBLE SPRITES To use a double sprite, you need to set the sprite positioning controls so that the sprites are adjacent on the screen. When you are doing this, it is important to bear in mind whether or not you have expanded the sprites.
Unexpanded sprites will be adjacent if you set their positions 24 pixels apart.
Expanded sprites need to be 48 pixels apart in order not to overlap.

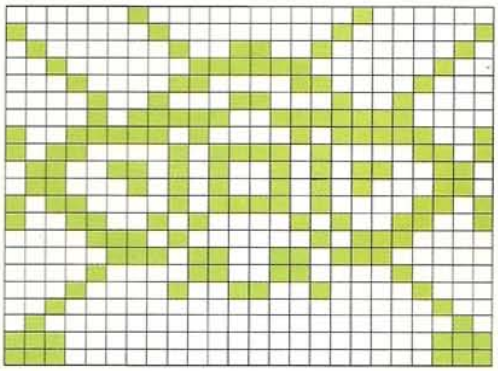
BUG

```
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
0
```



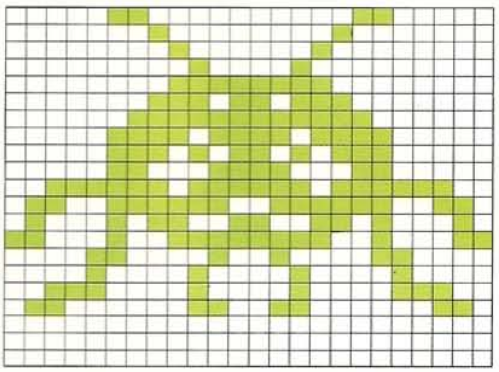
BUG

```
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
0
```



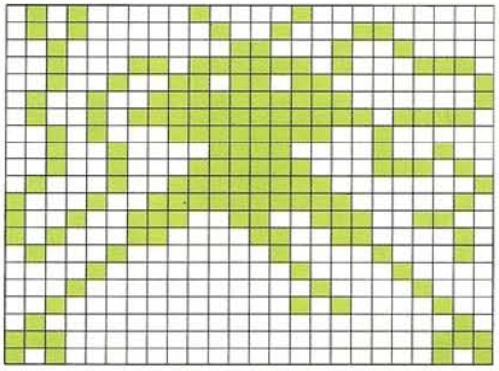
MICRO-MITE

```
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
0
```



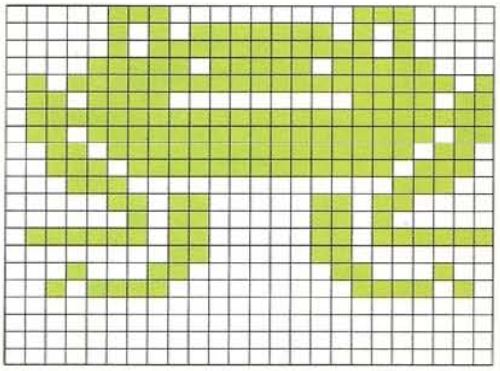
TRIPOD

```
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
0
```



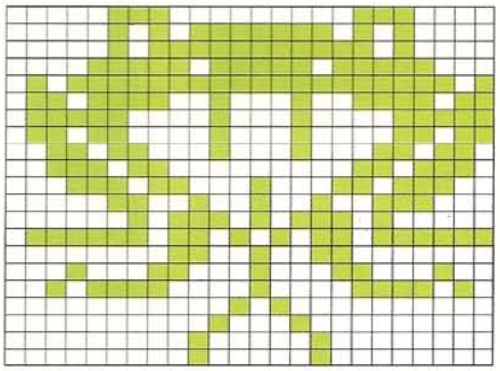
HOPPER

```
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
0
```



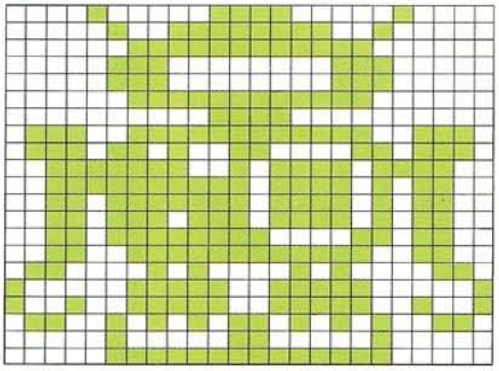
HOPPER

```
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
0
```



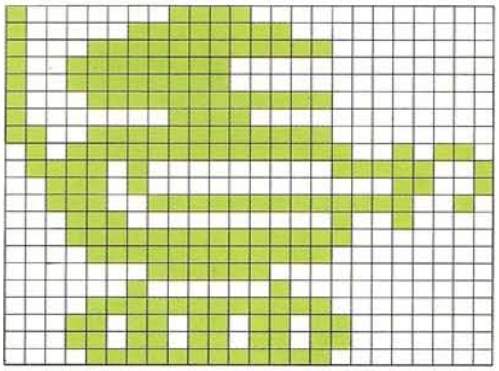
ROBOT

```
3,60,16,4,255,32,3
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
0
```

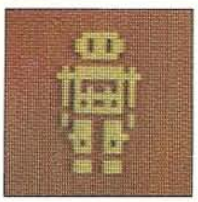


ROBOT

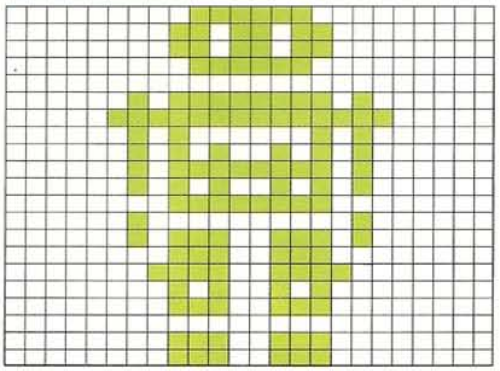
```
128,240,0,131,252,0,143
255,0,159,224,0,159,224
0,143,255,0,128,240,0
207,255,0,95,255,150,122
0,249,58,0,25,57,255
242,28,0,16,31,255,240
15,255,224,1,254,0,2
1,0,15,255,224,27,109
176,27,109,176,15,255,224
0
```



ROBOT

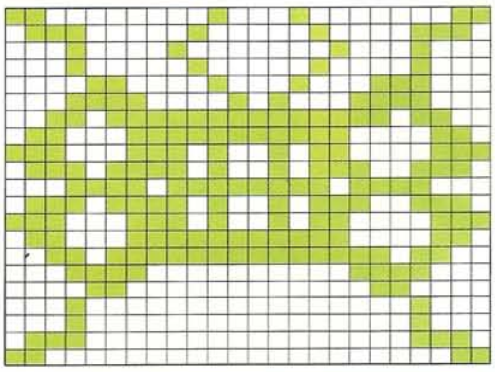


```
0,126,0,0,219,0,0
219,0,0,126,0,0,0
0,2,255,64,7,255,224
2,129,64,2,165,64,2
255,64,2,165,64,0,255
0,2,0,64,2,231,64
0,231,0,1,231,128,0
165,0,0,231,0,0,0
0,0,231,0,0,231,0
0
```



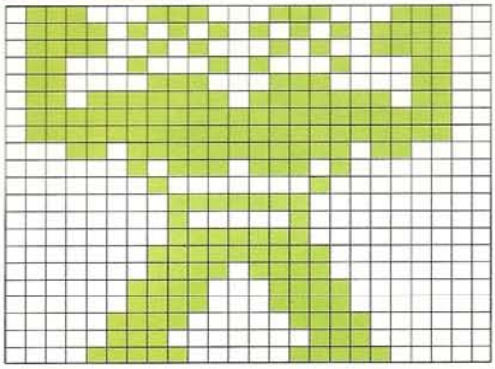
SQUAROID

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
0



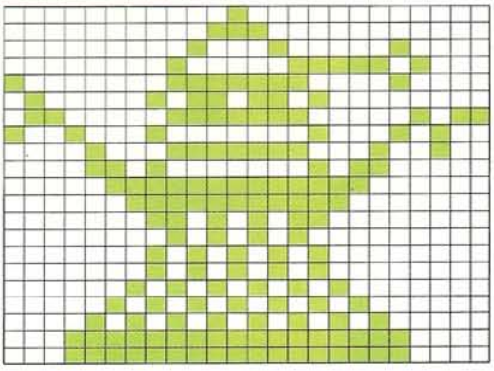
HUMANOID

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
0



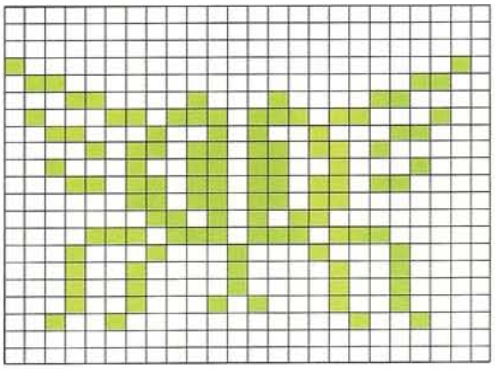
"DALEK"

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
0



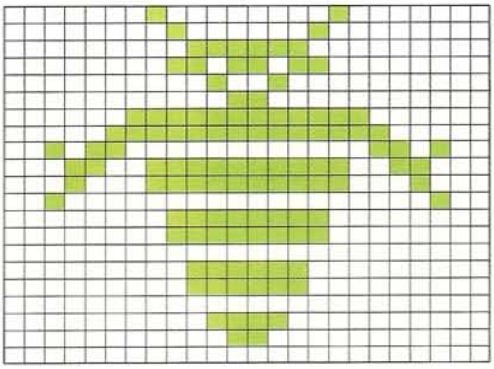
SPACE-FLY

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
0



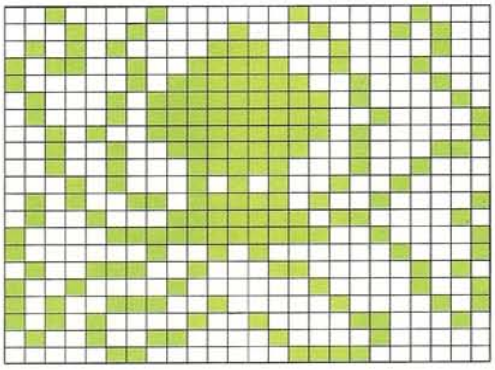
INSECTOID

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
0



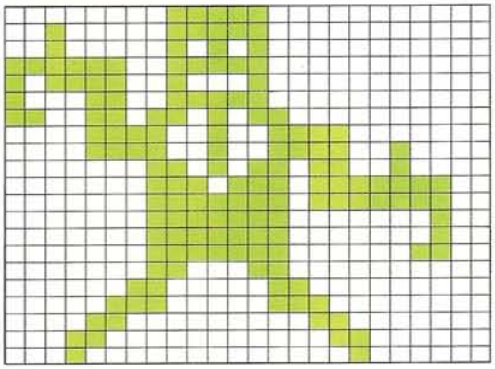
SEA MONSTER

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
0



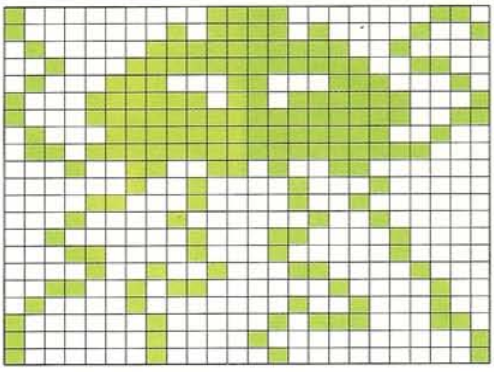
ANDROID

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
0



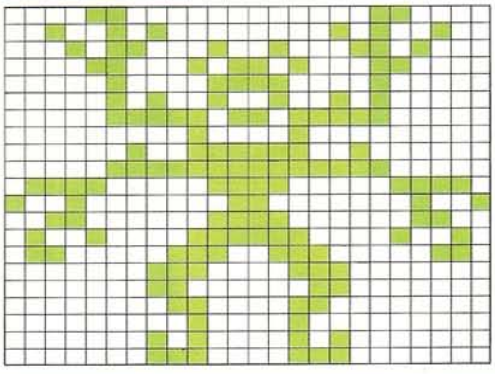
JELLY MONSTER

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
0



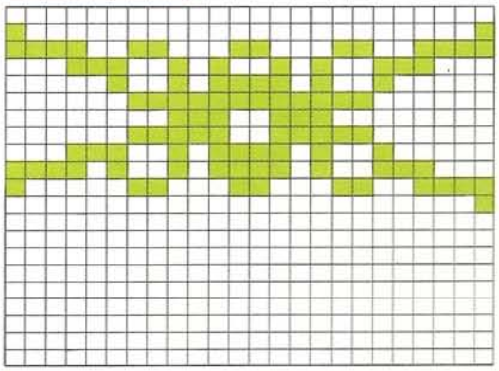
HYDRA MAN

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
0

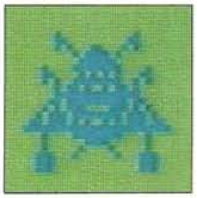


PLANETARY PROBE

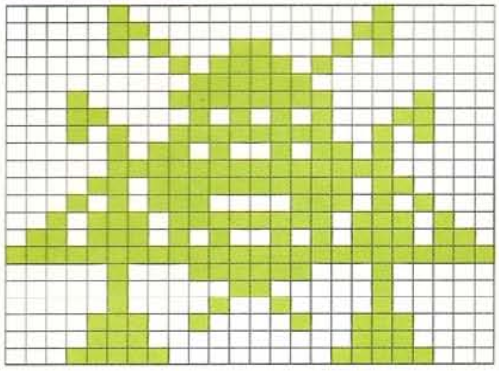
```
0,0,0,128,0,1,243
24,207,28,165,56,4,189
32,3,255,192,0,231,0
3,231,192,4,189,32,28
189,56,243,24,207,128,0
1,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0
```



LANDER

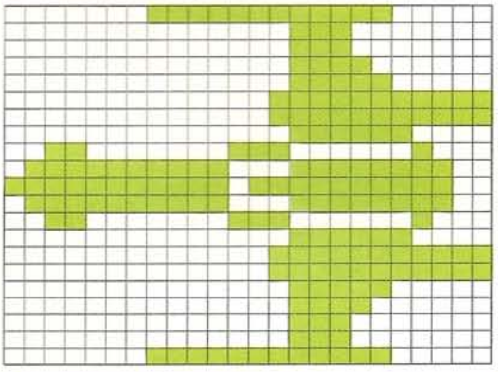


```
4,0,32,6,0,96,5
24,160,0,189,0,0,126
0,16,255,8,24,85,24
20,255,40,5,165,160,7
255,224,13,255,176,21,195
168,46,255,116,110,165,118
255,255,255,16,255,8,16
60,8,16,66,8,56,129
28,56,0,28,56,0,28
0
```



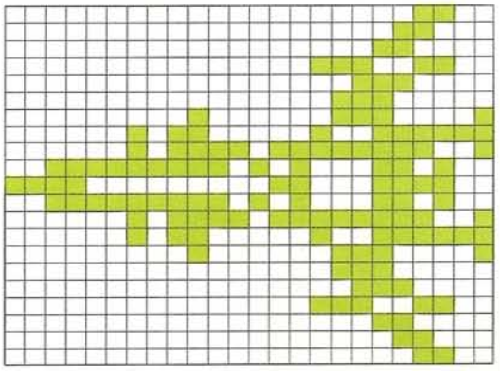
SPACE FIGHTER

```
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
0
```



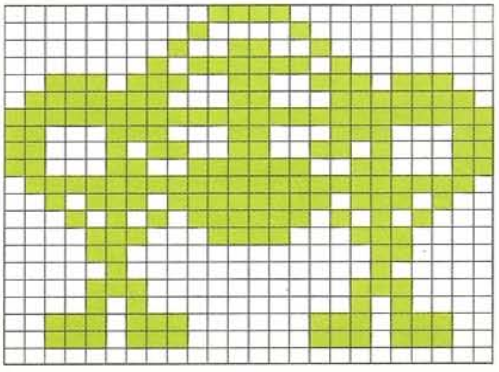
SPACE FIGHTER

```
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
0
```



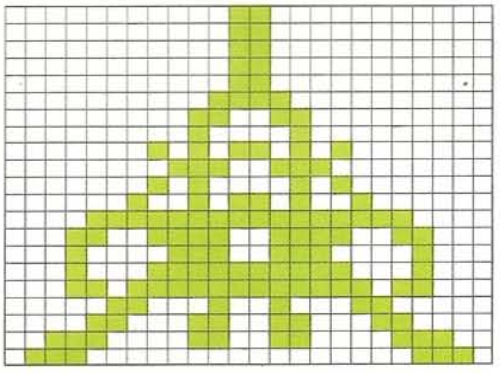
EXCURSION VEHICLE

```
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
0
```



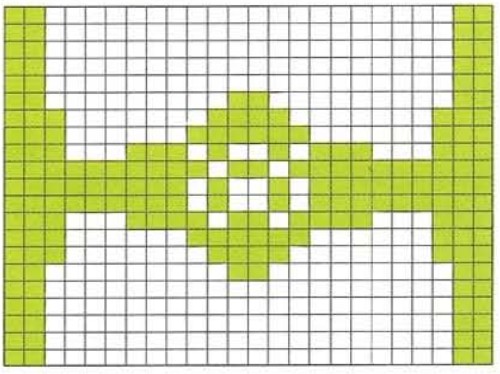
EXCURSION VEHICLE

```
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
0
```



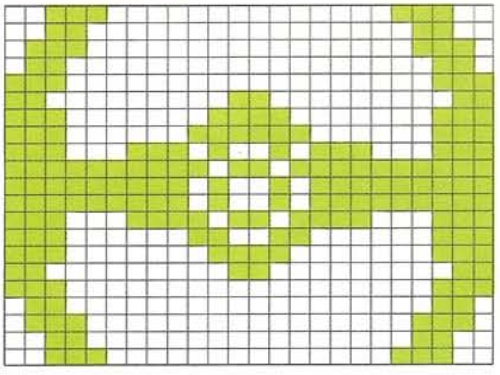
INTERGALACTIC CRUISER

```
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
0
```



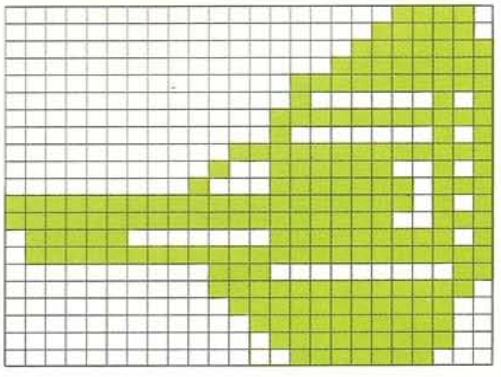
INTERGALACTIC CRUISER

```
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
0
```



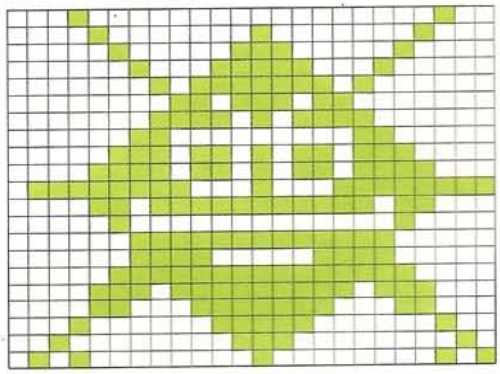
COMMAND SHIP

```
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
0
```



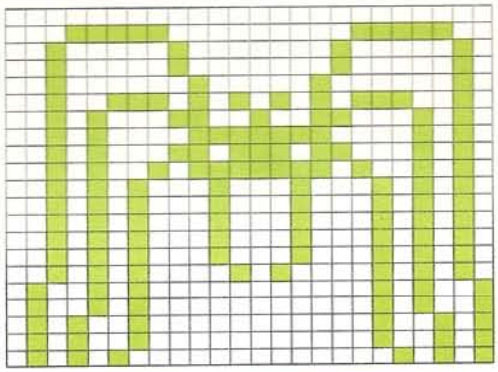
TRIBAL SPECTER

```
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
0
```



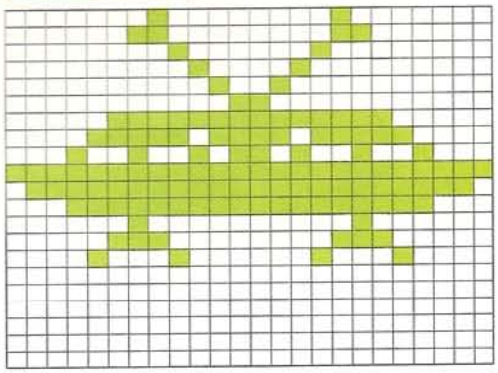
SPOOKY SPIDER

```
0,0,0,31,0,124,32
128,130,32,128,130,32,65
2,39,85,114,40,201,138
40,127,10,40,221,138,41
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
0
```



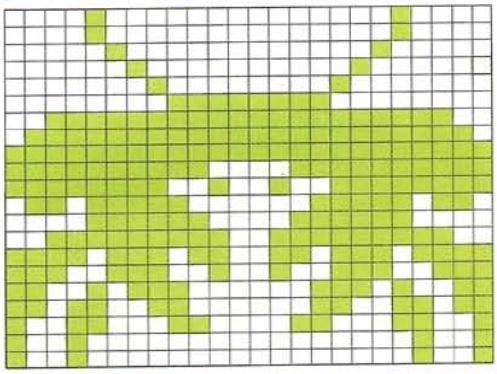
SPYING SAUCER

```
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
0
```



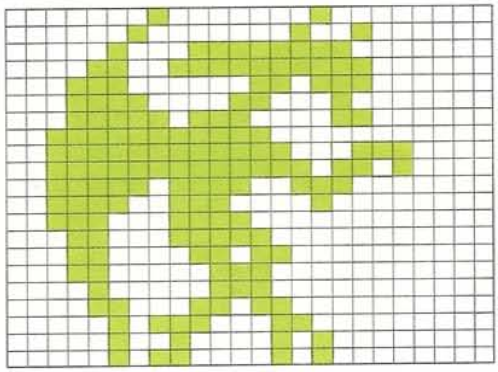
SPACE CRAB

```
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
0
```



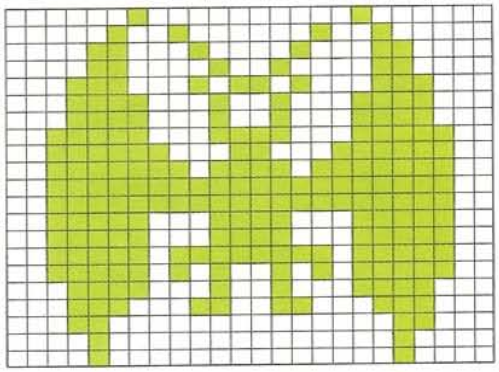
VAMPIRE

```
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
0
```



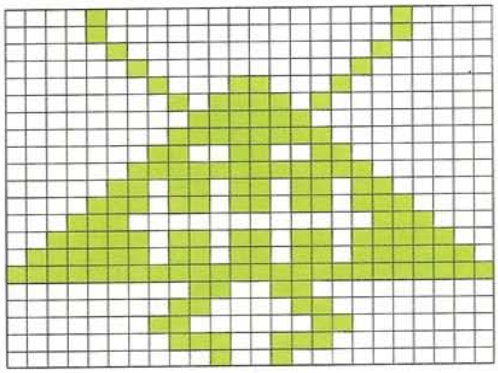
VAMPIRE

```
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
0
```



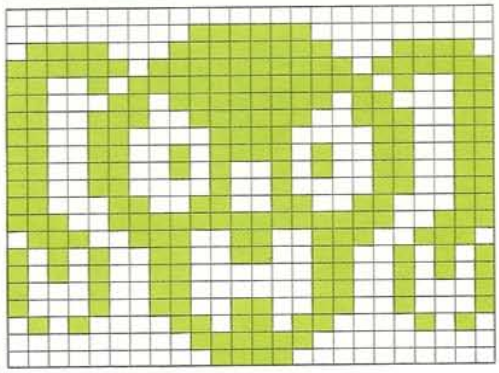
GHOUL

```
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
0
```



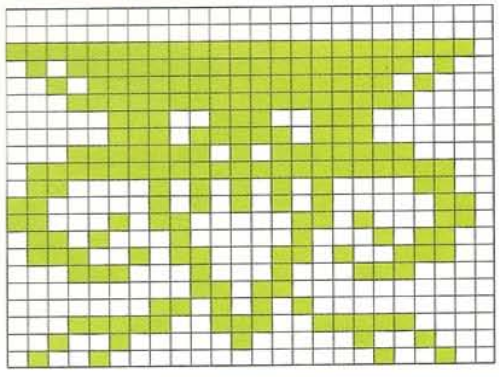
GHOUL

```
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
0
```



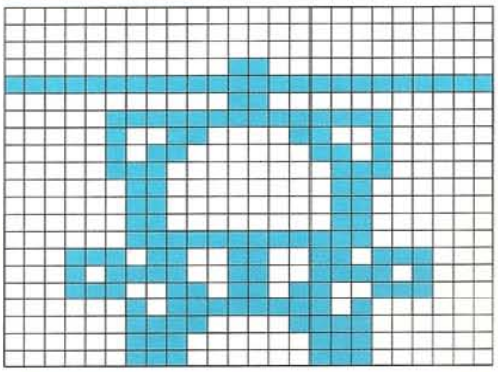
GHOUL

```
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
0
```



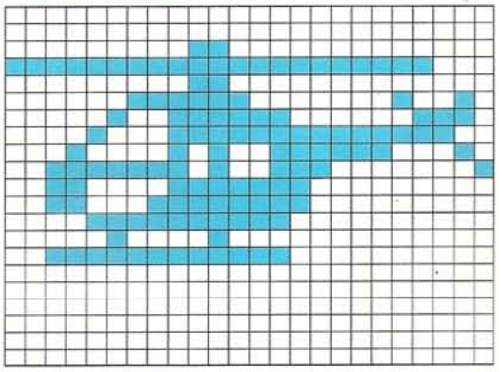
HELICOPTER

```
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
0
```



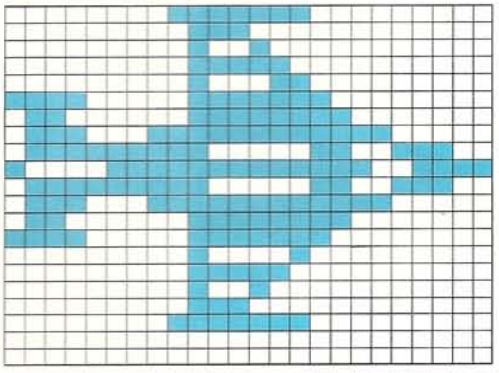
HELICOPTER

```
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
0
```



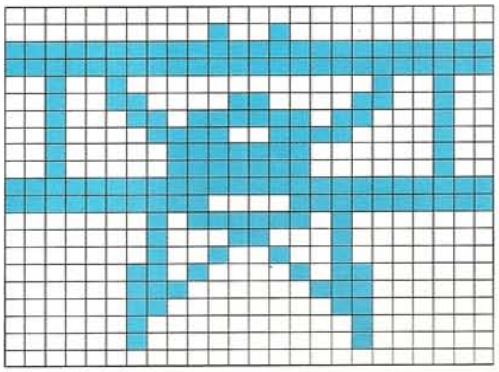
HIGH-ALTITUDE JET

```
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
0
```



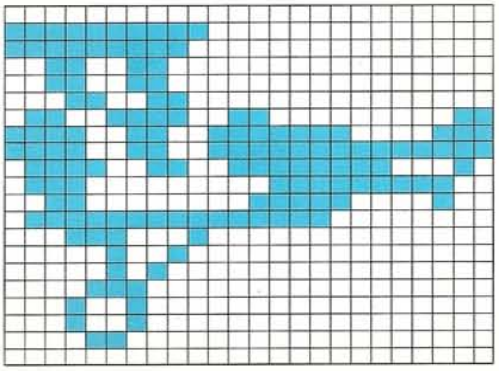
BIPLANE

```
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
0
```



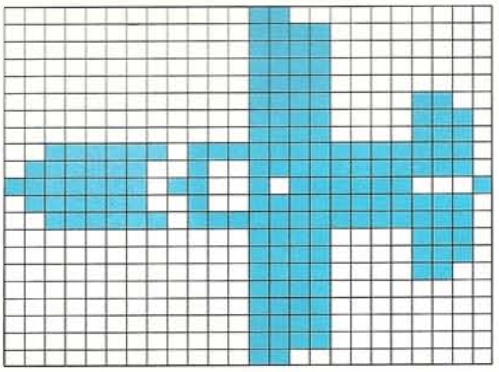
BIPLANE

```
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
0
```



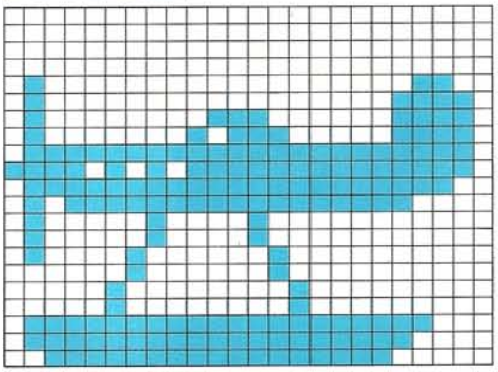
RECONNAISSANCE PLANE

```
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
0
```



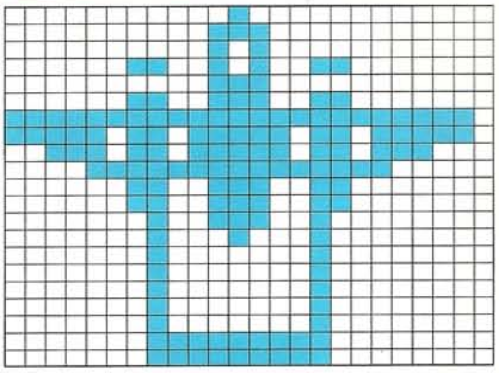
SEAPLANE

```
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
0
```



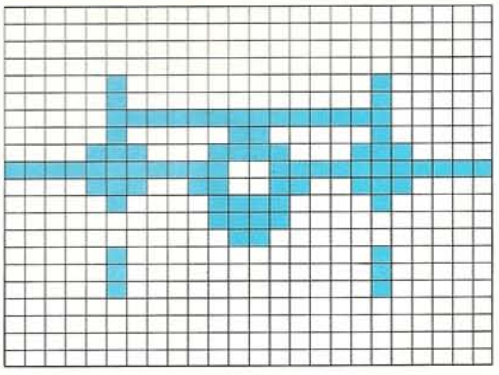
SEAPLANE

```
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
0
```



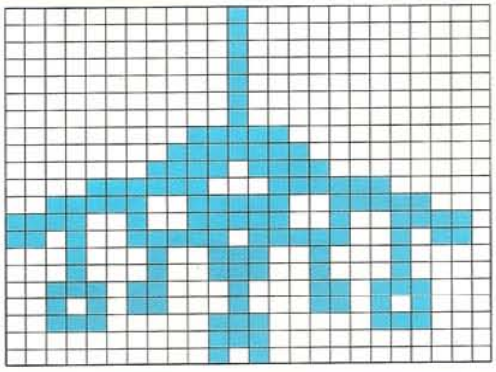
SEAPLANE

```
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
0
```



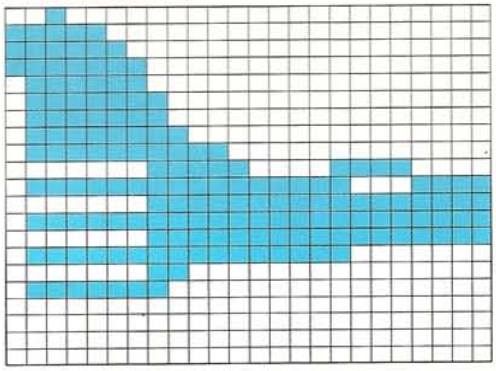
HIGH-ALTITUDE JET

```
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
0
```



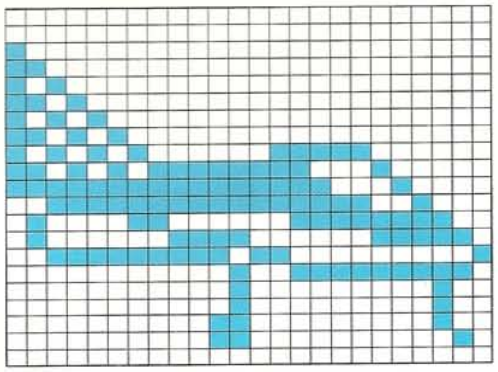
STUNT PLANE

```
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
0
```



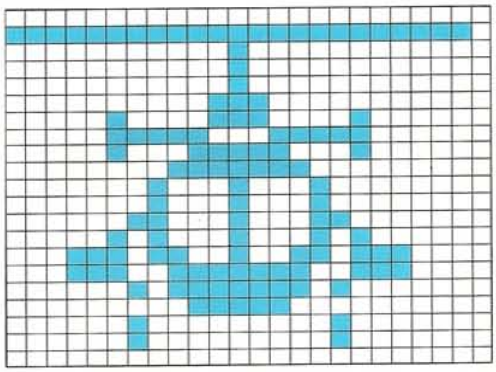
JET TRAINER

```
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
0
```



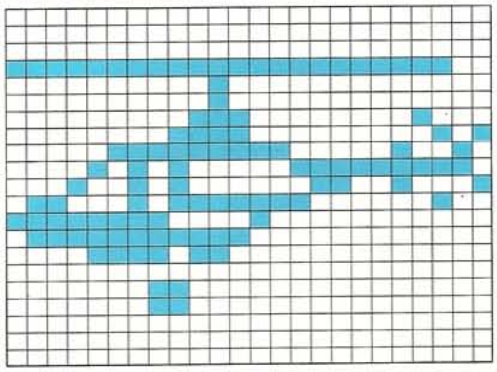
HELICOPTER

```
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
0
```



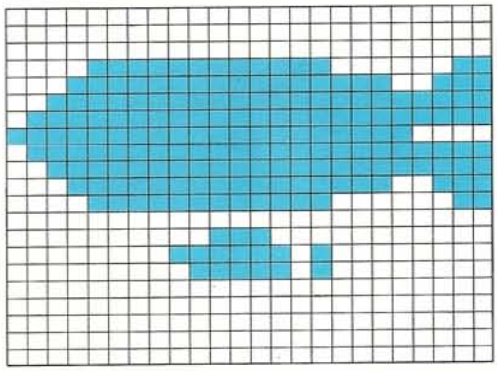
HELICOPTER

```
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
0
```



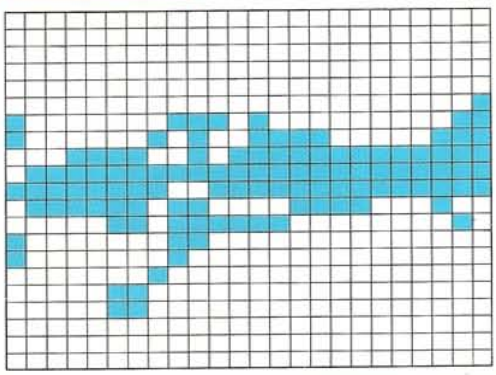
AIRSHIP

```
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
0
```



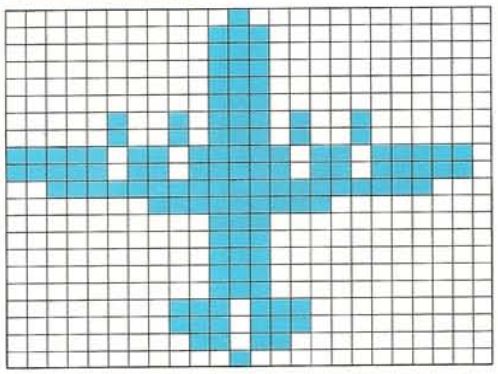
MONOPLANE

```
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
195,6,252,2,128,192,0
128,128,0,1,0,0,6
0,0,6,0,0,0,0
0,0,0,0,0,0,0
0
```



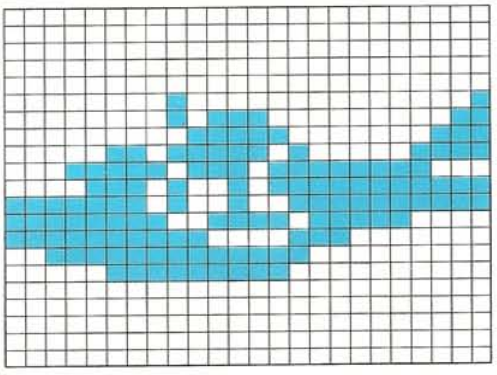
TRANSPORTER

```
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
0
```



TRANSPORTER

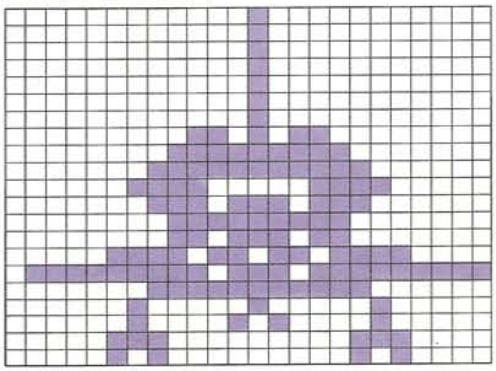
```
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,128,1,0,184,3
0,124,7,6,250,15,31
119,248,14,151,255,254,147
255,255,61,240,255,195,128
63,254,0,7,252,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0
```



SPACECRAFT

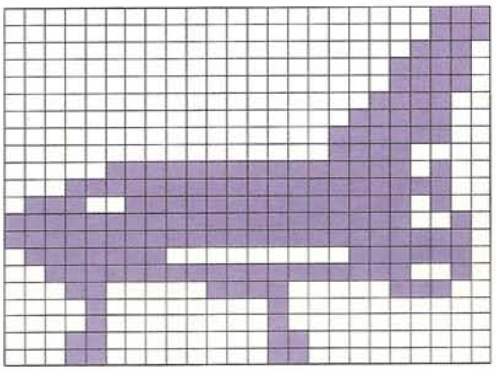
SHUTTLE

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
0



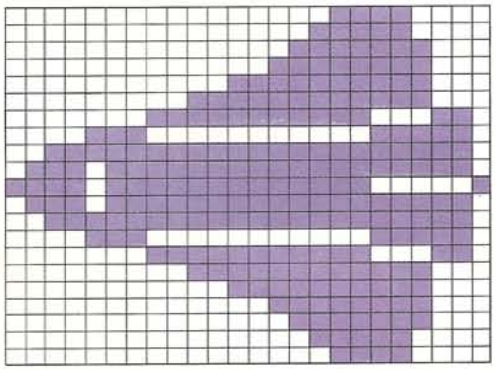
SHUTTLE

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
0



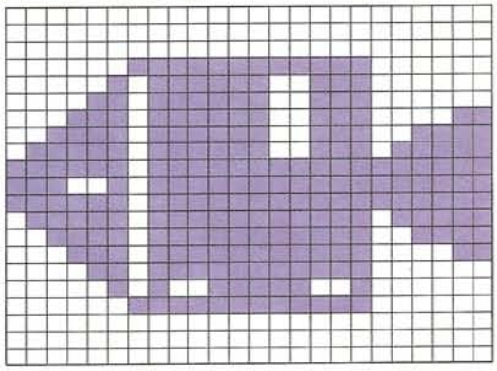
SHUTTLE

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
0



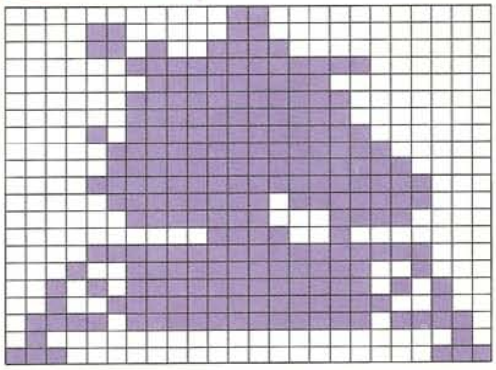
LUNAR MODULE

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
0



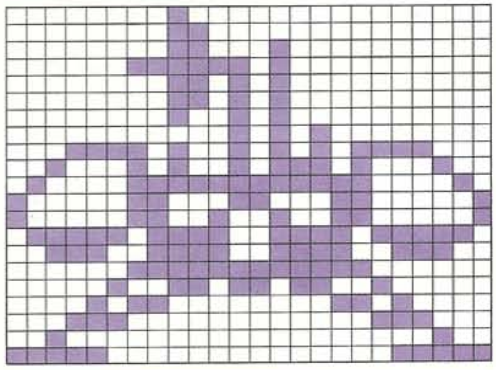
LUNAR LANDER

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
0



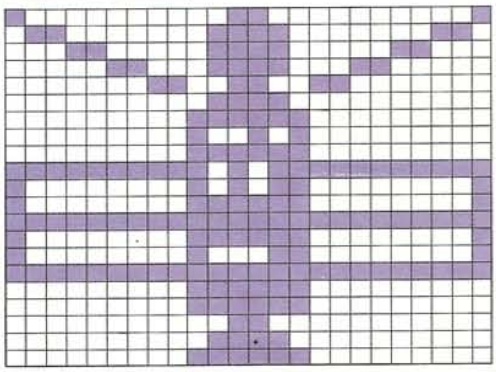
VIKING

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
0



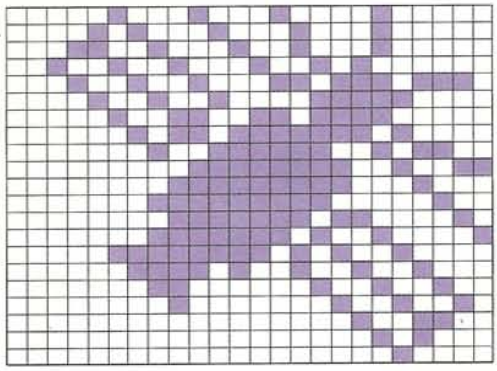
SKYLAB

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
0



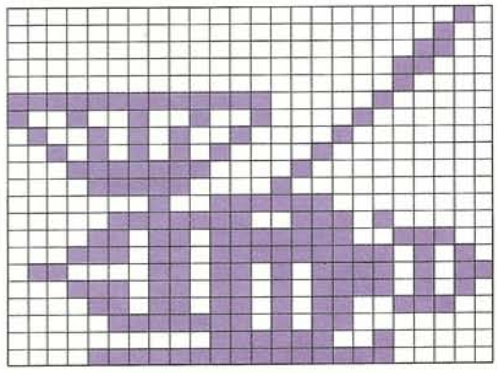
SKYLAB

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
0



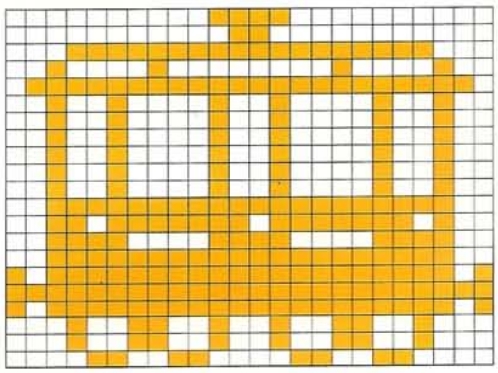
VENERA

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
0



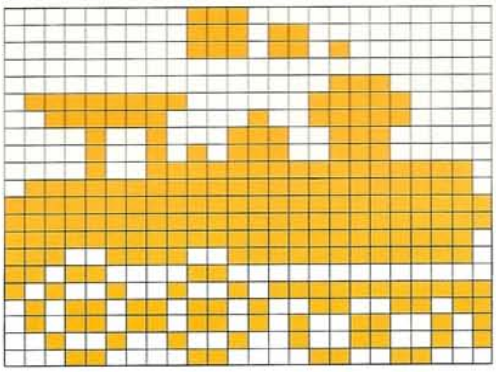
CARRIAGE

```
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
0
```



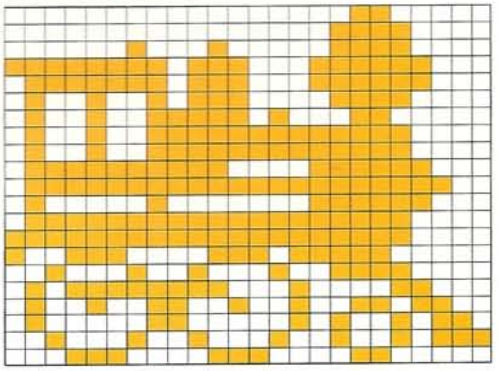
4-4-0 LOCO

```
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
0
```



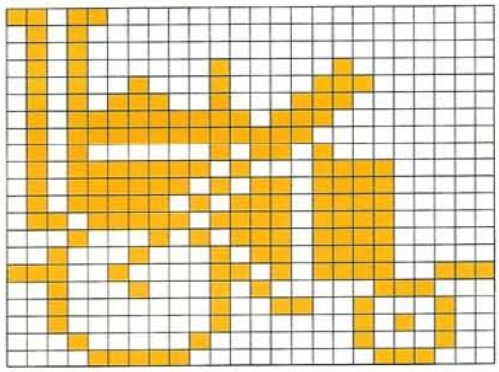
U.S. LOCO

```
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
0
```



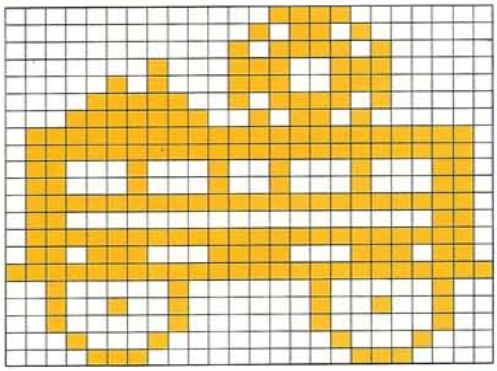
ROCKET

```
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
0
```



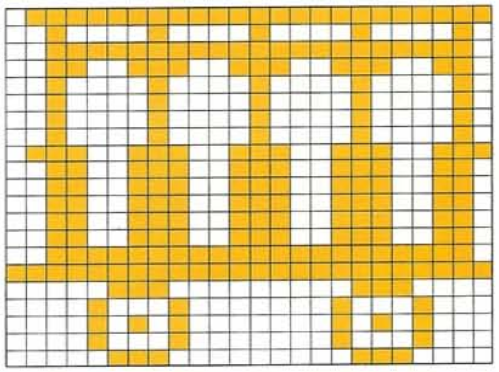
TENDER

```
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
0
```



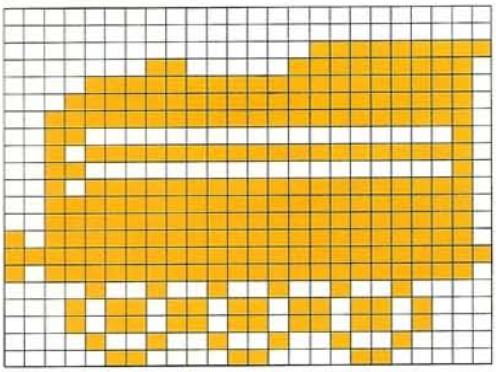
CARRIAGE

```
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
0
```



TENDER

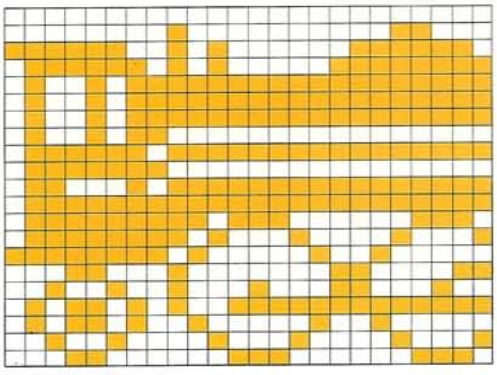
```
0,0,0,0,0,0,0
1,255,1,131,254,15,255
254,31,255,254,63,255,254
48,0,2,47,255,254,48
0,2,47,255,254,63,255
254,63,255,254,191,255,254
255,255,255,191,255,254,9
36,144,22,219,104,22,219
104,9,36,144,6,24,96
0
```

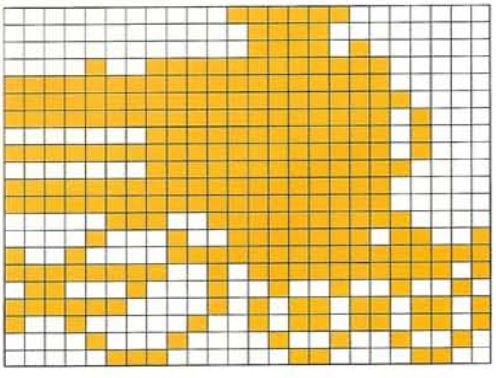


PACIFIC-TYPE LOCO

```
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
0
```

```
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
0
```

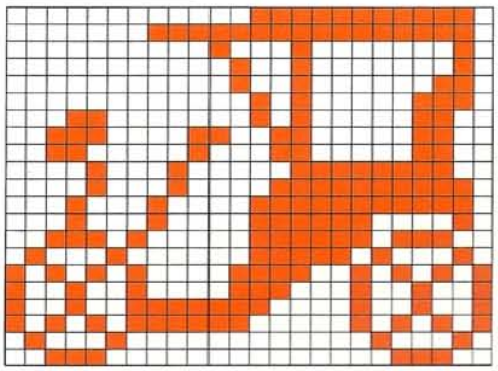




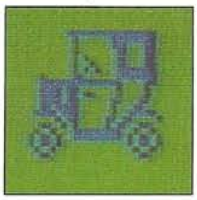
CARS, TRUCKS AND MOTORBIKES

VETERAN

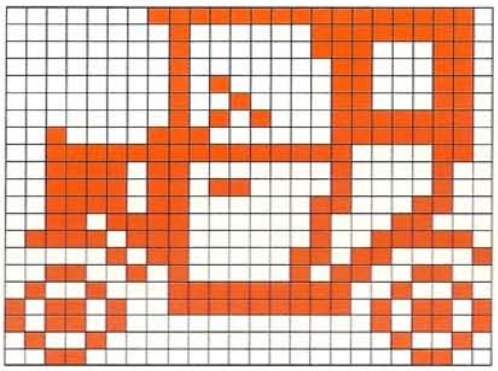
0,15,254,1,255,254,0
34,2,0,18,2,0,18
6,0,10,14,24,10,12
56,102,12,48,70,12,8
131,252,8,135,254,17,15
254,17,15,194,58,15,220
68,15,162,170,30,85,146
28,73,147,248,73,171,240
85,68,0,34,56,0,28
0



VETERAN

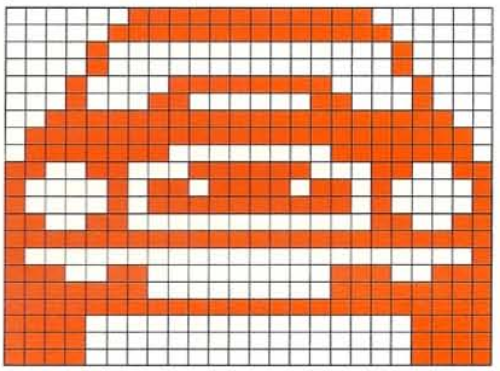


1,255,254,0,129,254,0
129,198,0,128,198,0,160
198,0,144,198,0,168,254
32,192,254,63,255,134,61
128,132,61,176,132,61,129
140,55,129,24,123,129,62
5,131,97,50,130,204,72
255,146,180,255,173,180,0
45,72,0,18,48,0,12
0



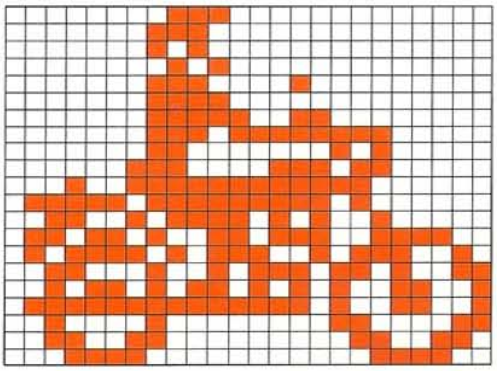
SALOON

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
0



MOTORBIKE

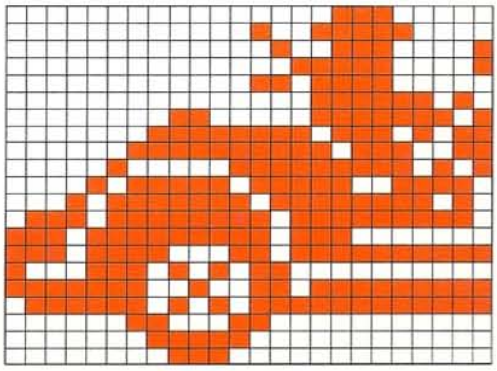
0,224,0,1,192,0,1
64,0,0,160,0,1,194
0,1,224,0,1,243,0
1,223,224,1,128,160,3
135,96,19,255,192,125,255
64,50,245,32,109,55,60
94,185,114,49,46,211,33
106,153,127,254,153,33,0
195,51,0,102,30,0,60
0

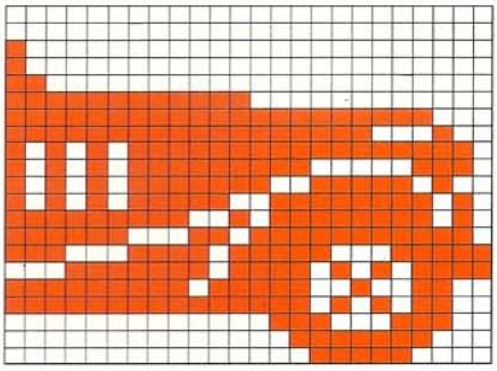


CLASSIC TOURER

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

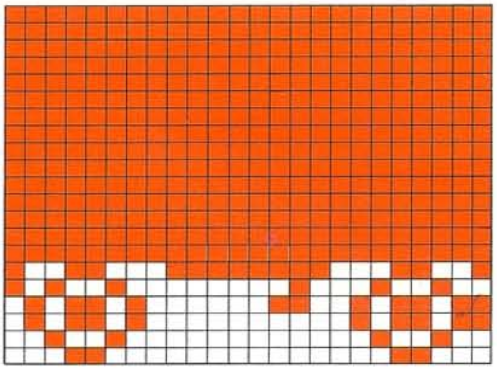
0,0,0,0,0,0,128
0,0,128,0,0,192,0
0,255,240,0,255,255,248
255,255,196,171,255,254,171
254,14,171,249,244,171,231
250,255,151,250,252,111,254
227,223,31,31,222,175,255
190,76,255,254,172,0,7
28,0,3,248,0,1,240
0





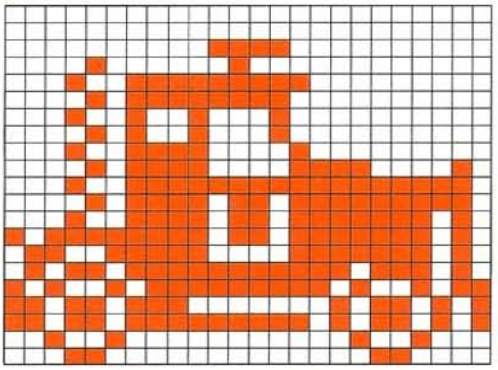
TRUCK

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

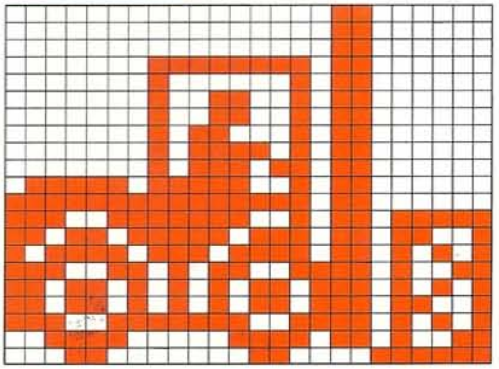


FORKLIFT

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
0



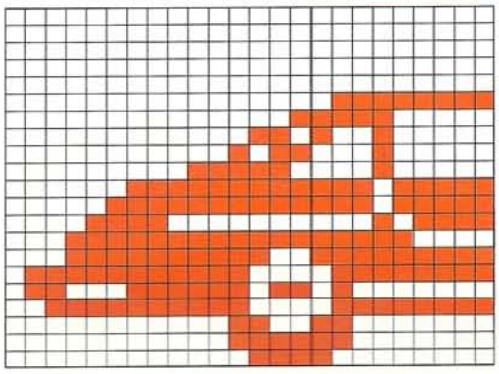
0,0,192,0,0,192,0
0,192,1,254,192,1,2
192,1,50,192,1,50,192
1,98,192,1,122,192,1
118,192,125,114,192,255,254
192,231,242,223,219,238,217
189,222,213,230,179,211,218
173,213,218,173,217,231,243
213,60,30,211,24,12,63
0

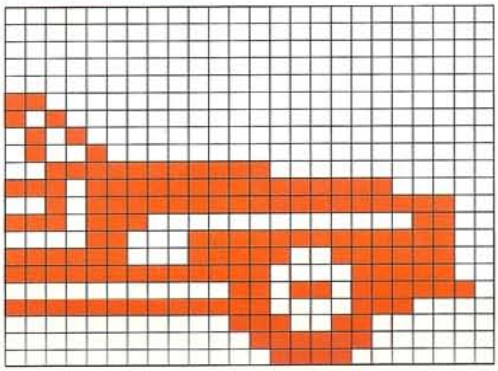


SPORTS SALOON

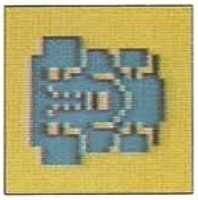
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,255,0,3,224
0,13,32,0,22,32,0
124,32,3,255,223,7,255
223,15,0,47,31,255,240
31,249,255,127,240,255,99
246,128,63,240,255,0,25
128,0,31,128,0,15,0
0

0,0,0,0,0,0,0
0,0,0,0,0,0,0
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
0

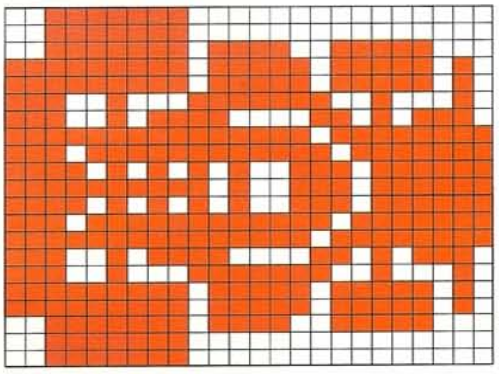




FORMULA 1

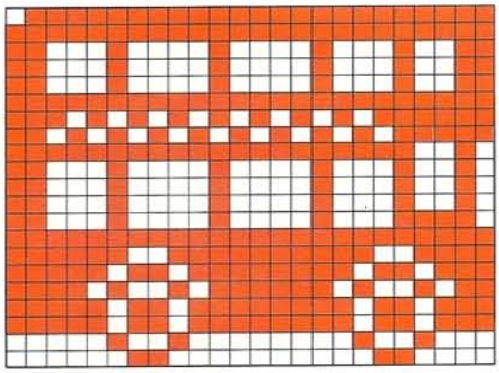


63,128,0,63,128,0,63
188,248,255,190,250,255,190
250,228,127,34,229,225,174
255,254,255,239,255,127,245
83,191,255,211,191,245,83
191,239,255,127,255,254,255
229,225,174,228,127,34,255
190,250,255,190,250,63,188
248,63,128,0,63,128,0
0



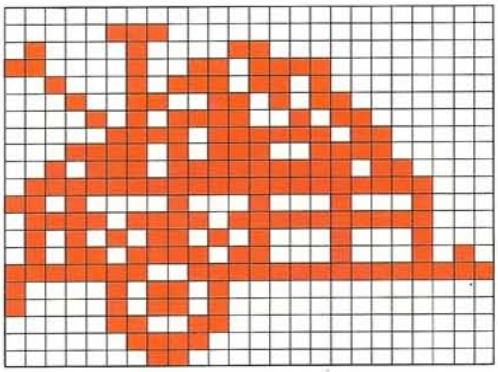
LONDON BUS

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
0

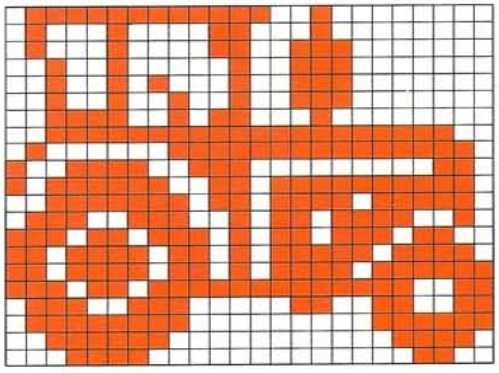


TRACTOR

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
0

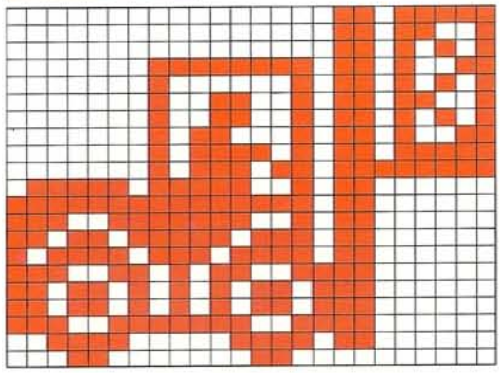


127,249,0,68,49,0,68
51,128,36,19,128,37,19
128,36,147,128,60,177,32
63,255,252,64,255,254,158
112,6,191,55,254,63,183
190,115,150,242,237,214,236
222,215,222,222,215,191,237
255,243,115,131,51,127,128
63,63,0,30,30,0,12
0



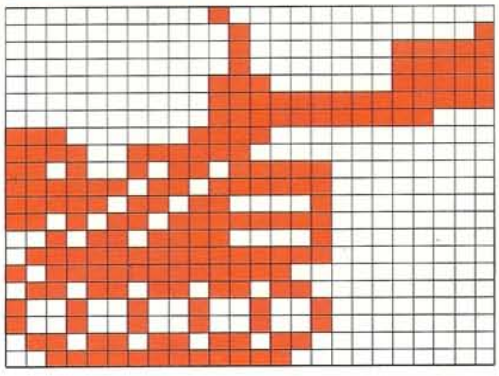
FORKLIFT

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
0



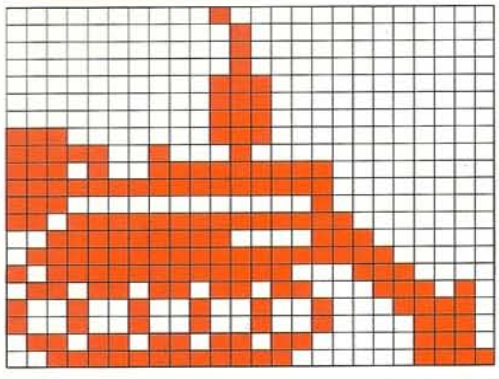
BULLDOZER

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
0



BULLDOZER

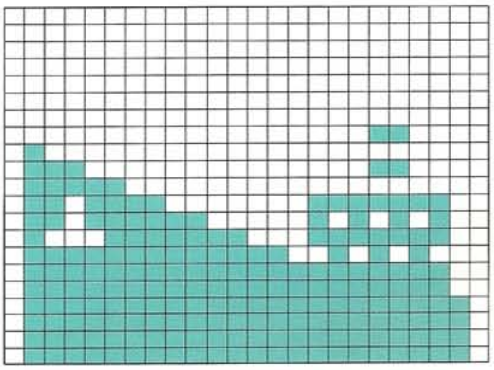
0,64,0,0,32,0,0
32,0,0,32,0,0,112
0,0,112,0,0,112,0
224,112,0,249,32,0,233
255,0,255,255,0,224,1
0,223,255,128,63,225,192
127,255,224,191,252,112,109
182,58,146,73,30,146,73
14,109,182,14,63,252,15
0



SHIPS AND BOATS

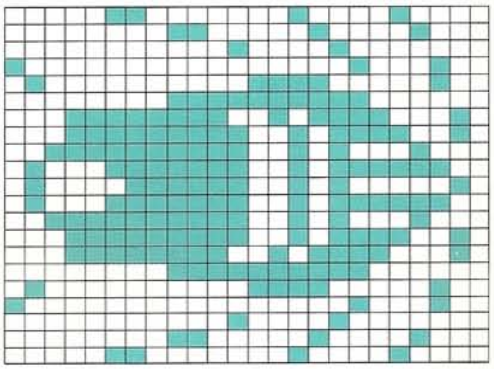
FREIGHTER

```
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
0
```



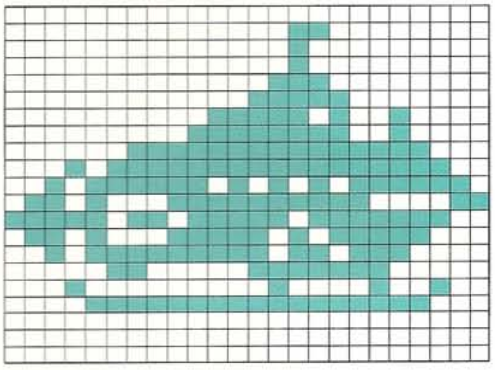
SPEEDBOAT (FROM ABOVE)

```
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
0
```



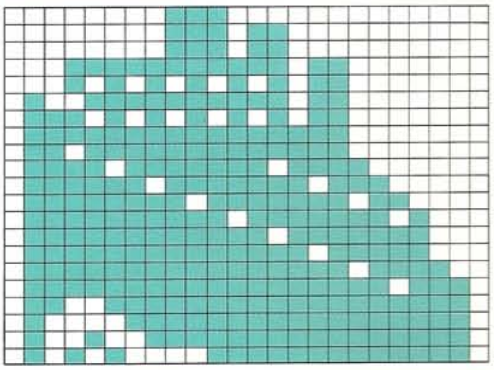
SUBMERSIBLE

```
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
0
```



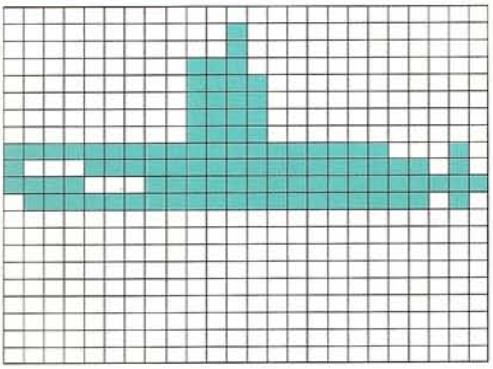
LINER

```
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
0
```



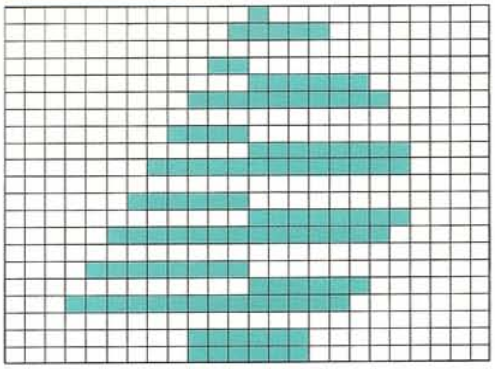
SUBMARINE

```
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
0
```



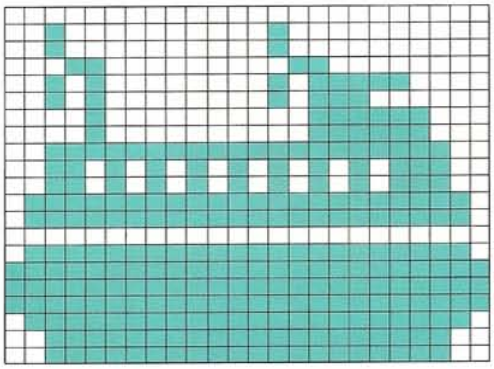
YACHT

```
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
0
```



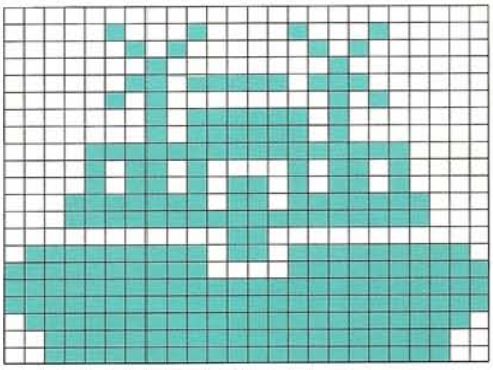
HOVERCRAFT

```
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
0
```



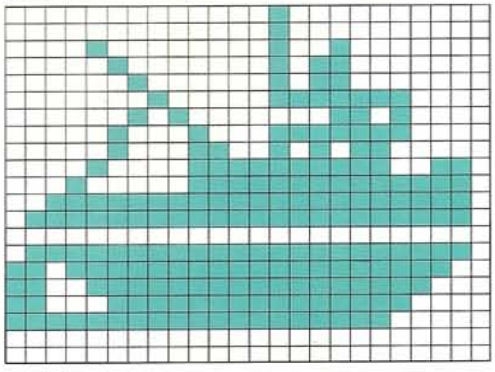
HOVERCRAFT

```
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
0
```



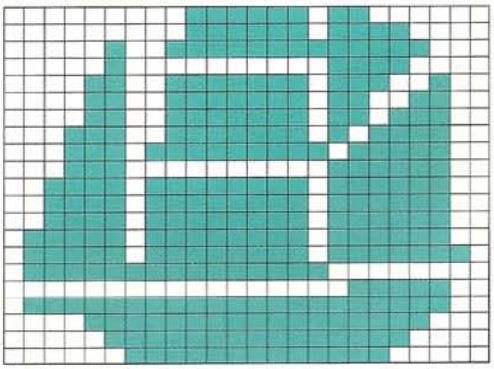
TUGBOAT

```
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
0
```



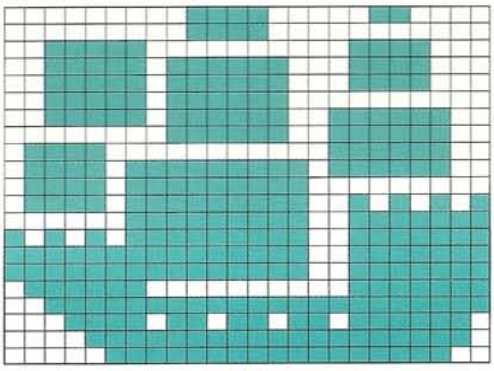
TALL SHIP

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
0



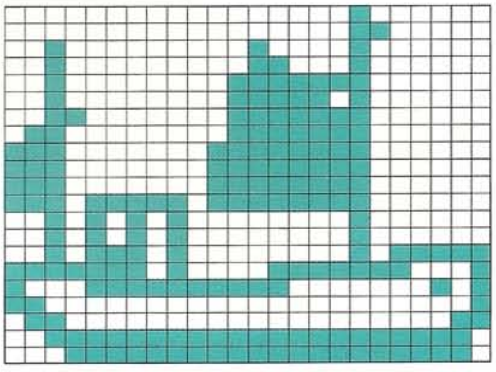
MAN O' WAR

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
0



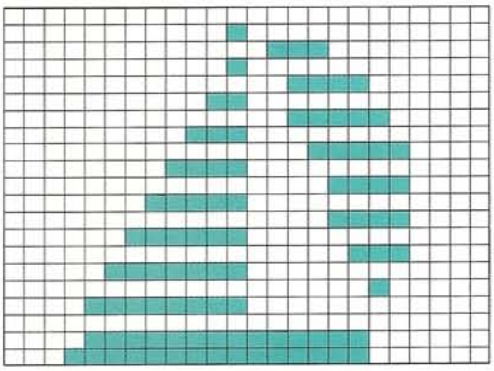
FISHING SMACK

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
0



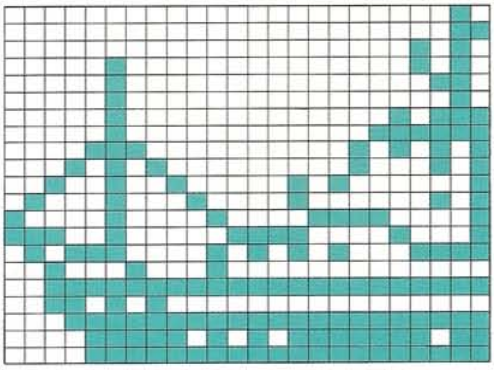
YACHT

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
0

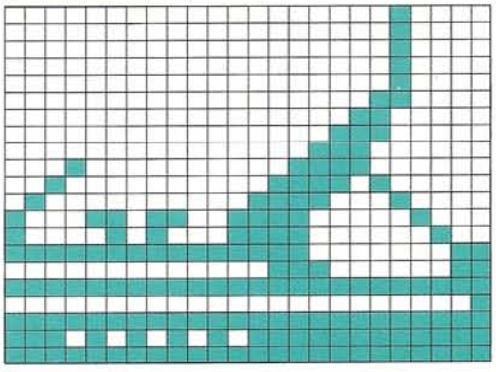


STERN TRAWLER

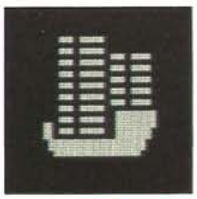
0,0,2,0,0,3,0
0,10,4,0,10,4,0
6,4,0,2,4,0,23
4,0,63,14,0,85,21
0,69,36,131,129,68,66
5,132,33,225,228,30,17
92,42,143,34,32,0,31
255,255,29,64,0,15,255
255,7,213,251,7,255,255
0



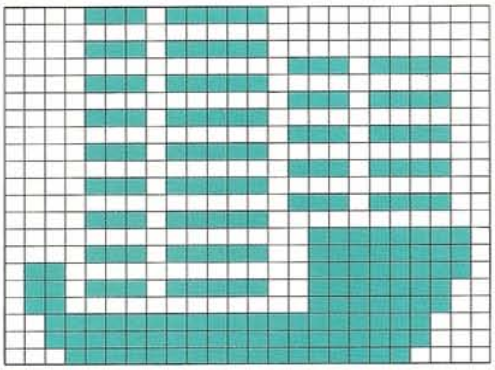
0,0,16,0,0,16,0
0,16,0,0,16,0,0
16,0,0,48,0,0,96
0,0,224,0,1,192,16
3,192,32,7,160,64,15
16,141,158,8,133,30,4
255,254,3,0,7,3,255
255,255,0,0,1,255,255
255,234,175,255,255,255,255
0



JUNK

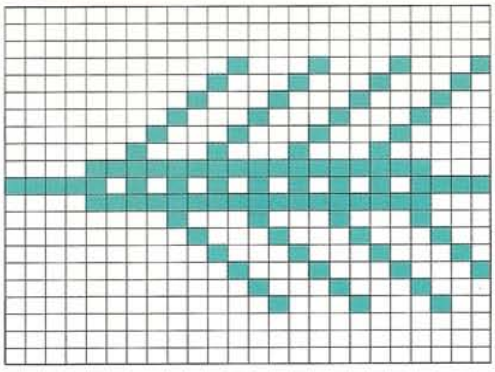


14,248,0,0,0,0,14
248,0,0,3,188,14,248
0,0,3,188,14,248,0
0,3,188,14,248,0,0
3,188,14,248,0,0,3
188,14,248,0,0,1,254
14,249,254,96,1,254,110
249,252,96,1,252,63,255
248,63,255,248,31,255,240
0



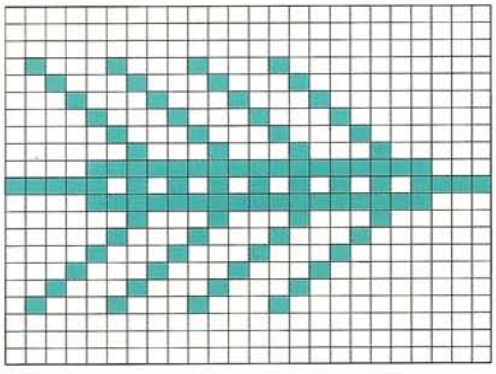
ROWING EIGHT

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
0



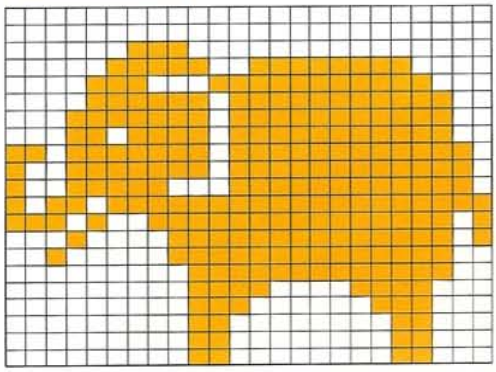
ROWING EIGHT

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
0



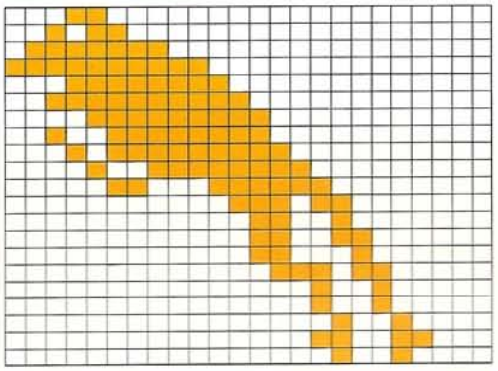
ELEPHANT

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
0



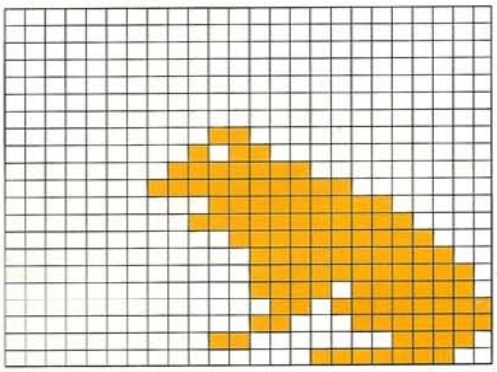
FROG

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
0



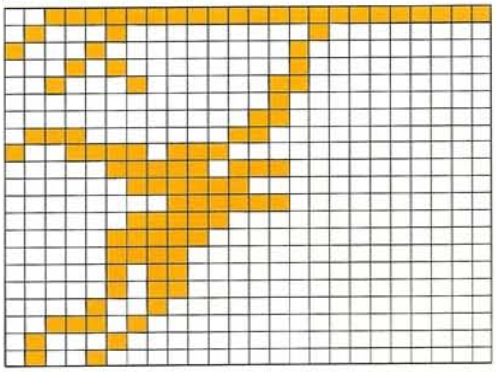
FROG

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
0



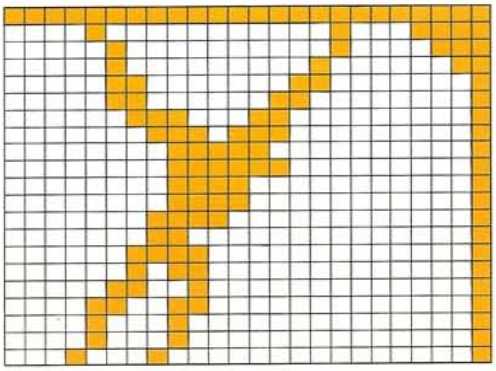
GIBBON

63,255,255,68,1,0,136
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
0



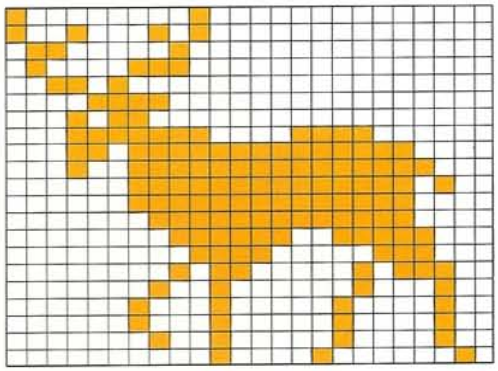
GIBBON

255,255,255,8,0,143,4
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
0



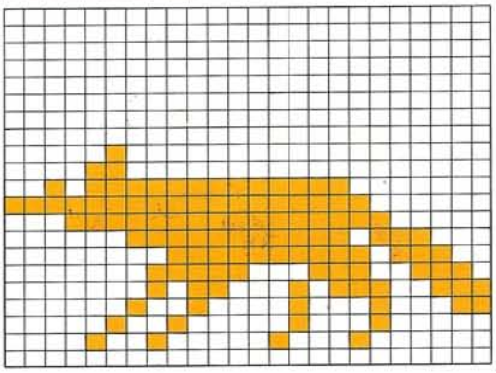
MOOSE

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
0



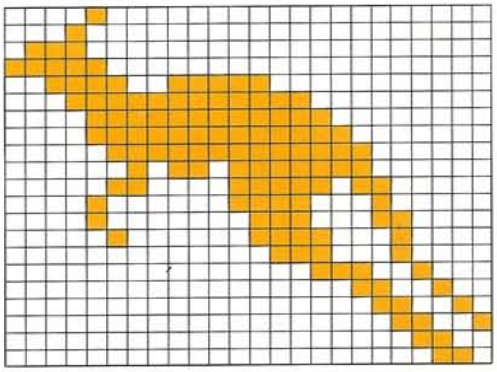
FOX

0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,4,0,0,12
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
0



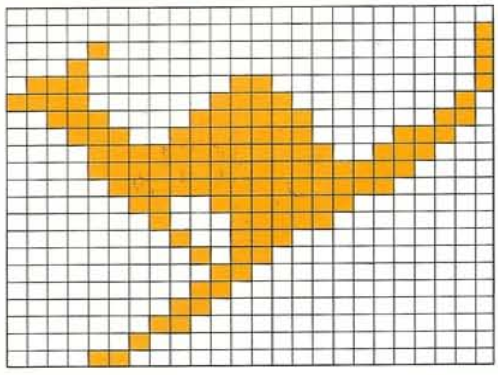
KANGAROO

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
0



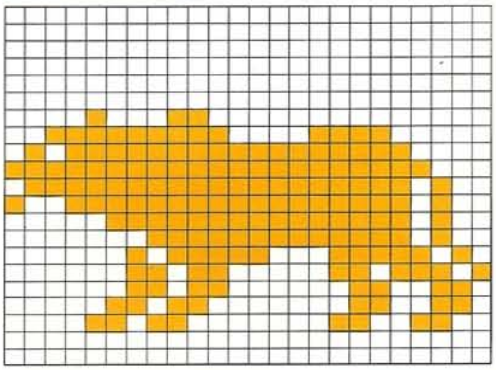
KANGAROO

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
0



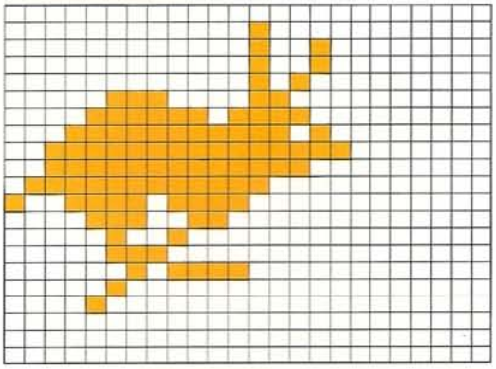
TIGER

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
255,248,127,255,244,159,255
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
0



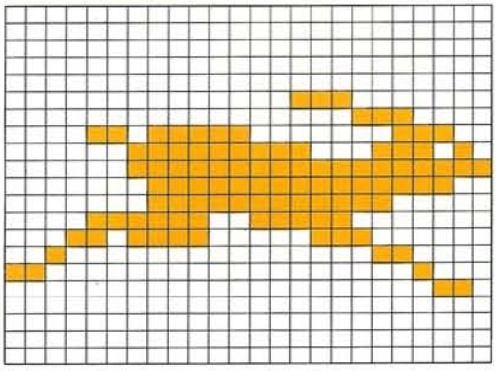
RABBIT

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
0



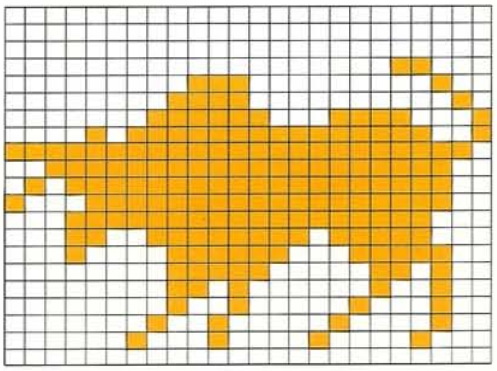
RABBIT

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
0



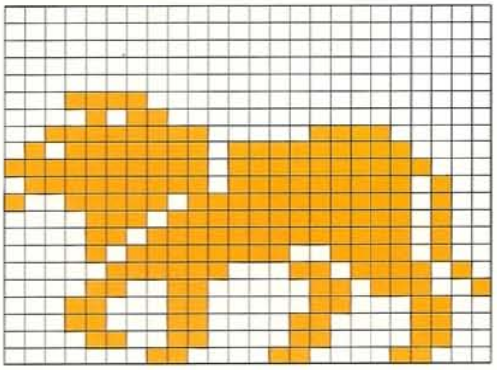
BUFFALO

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
0



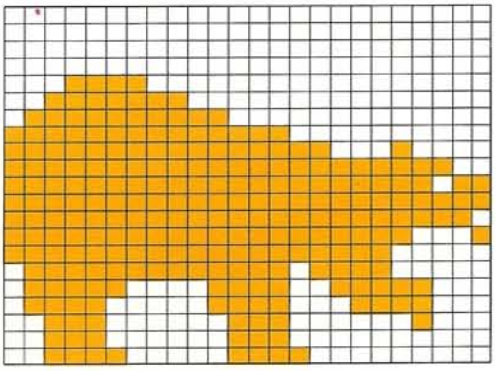
LION

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
223,248,127,223,244,159,127
244,14,255,244,13,255,244
11,252,244,7,227,122,12
193,105,24,193,140,24,195
12,12,195,12,1,134,24
0



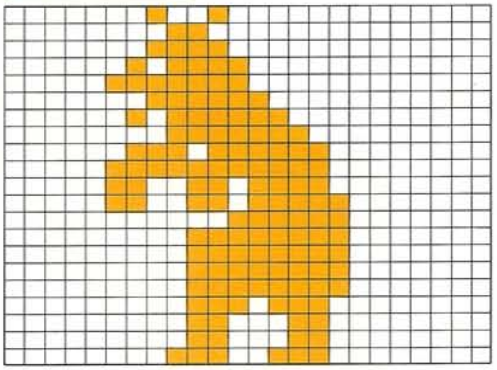
POLAR BEAR

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
255,252,255,255,251,255,255
255,255,255,254,255,255,241
255,255,224,127,253,224,124
60,248,120,60,120,56,28
8,56,28,0,60,30,0
0



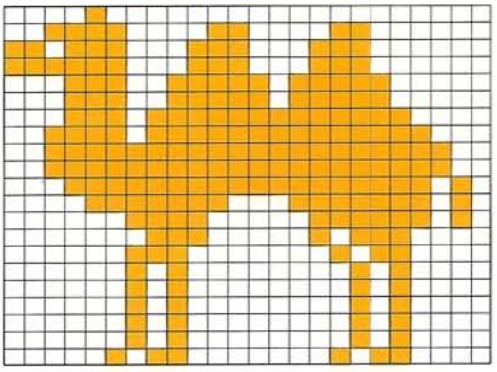
BROWN BEAR

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
0,0,99,0,0,231,0
0



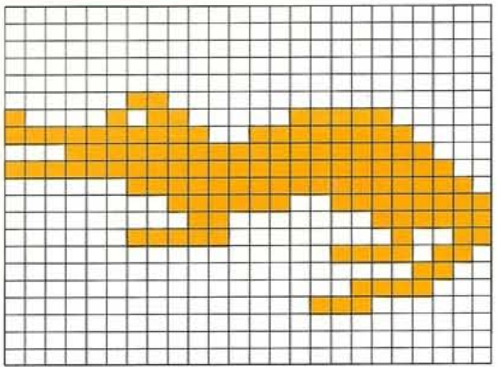
CAMEL

8,0,0,60,48,192,220
113,192,252,113,192,28,251
224,28,251,224,29,255,240
61,255,240,63,255,252,63
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
0



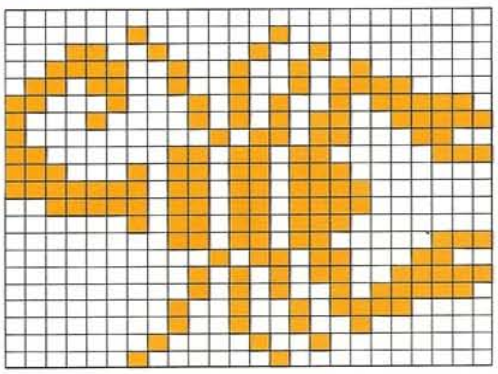
CROCODILE

0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,3,0,0,133,131,224
255,207,240,31,255,248,255
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
0



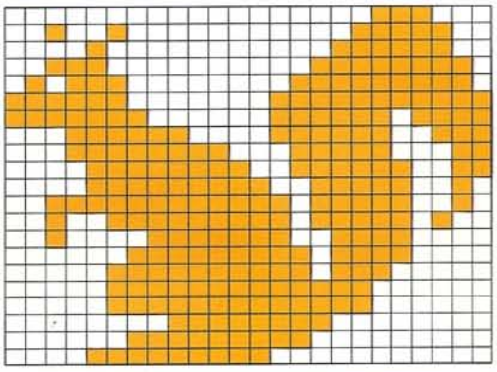
SCORPION

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
0



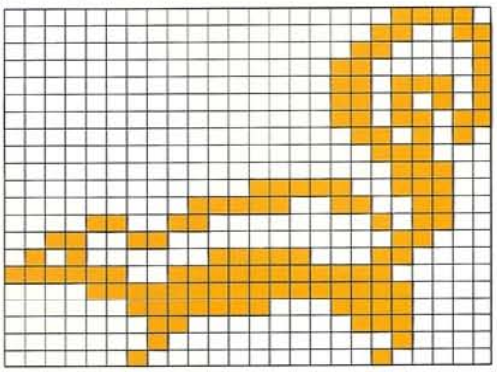
SQUIRREL

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
0



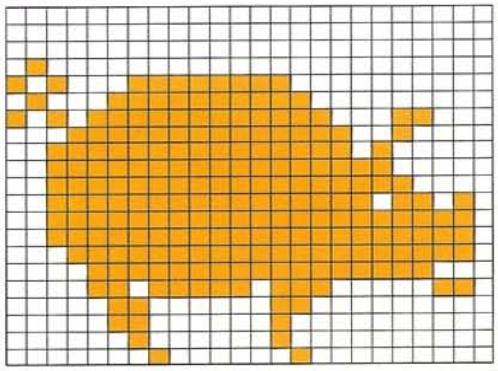
SKUNK

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
0



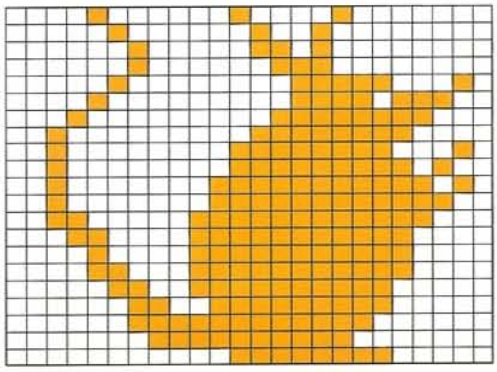
PIG

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
0



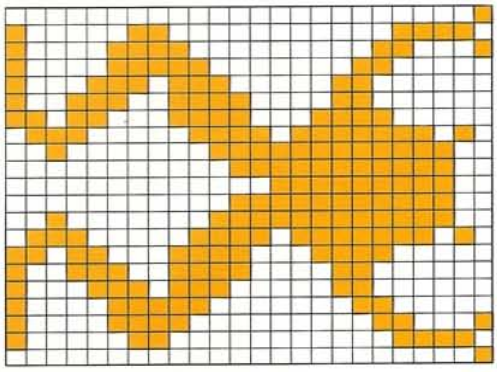
MOUSE

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
0



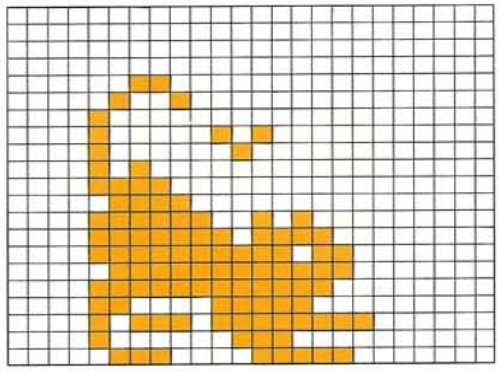
FROG

0,0,1,131,0,30,135
128,49,135,192,96,142,224
192,156,240,192,216,241,224
112,123,250,32,63,252,0
31,252,0,7,252,0,31
252,32,63,252,112,123,250
216,241,224,156,240,192,142
224,192,135,192,96,135,128
49,131,0,30,0,0,1
0

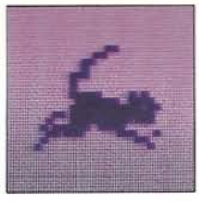


CAT

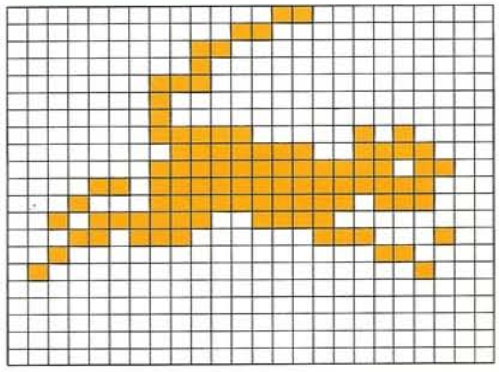
0,0,0,0,0,0,0
0,0,0,0,0,3,0
0,4,128,0,8,64,0
8,40,0,8,16,0,10
0,0,15,0,0,7,128
0,15,202,0,15,239,0
15,253,128,7,255,128,15
255,0,12,62,0,11,191
192,8,28,0,7,15,128
0



CAT

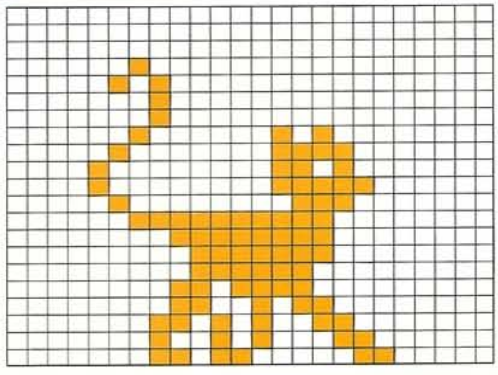


0,6,0,0,24,0,0
96,0,0,64,0,1,192
0,1,0,0,1,0,0
1,240,80,0,254,120,1
255,236,13,255,248,17,255
184,47,207,0,27,129,196
32,0,48,64,0,8,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0



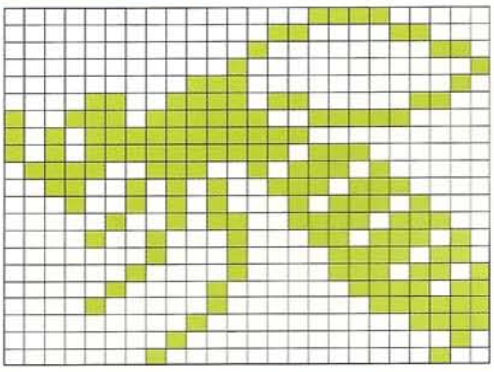
CAT

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
0



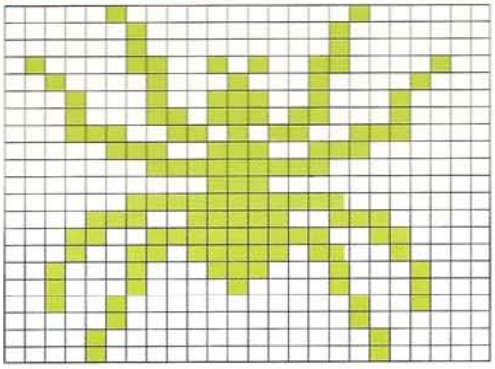
WASP

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
0



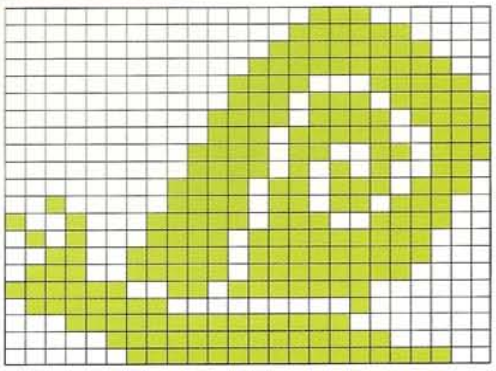
SPIDER

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
0



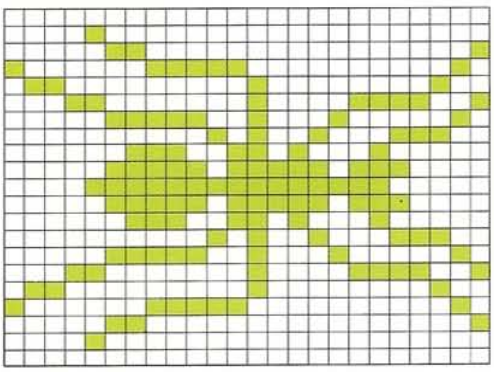
SNAIL

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
0



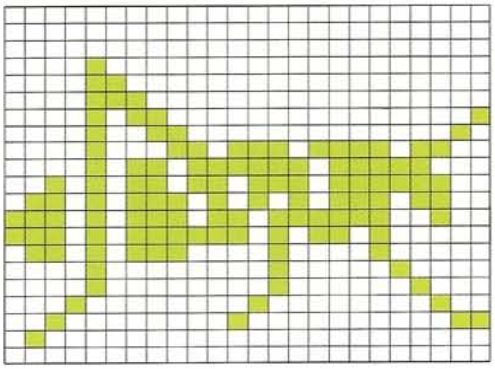
ANT

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
0



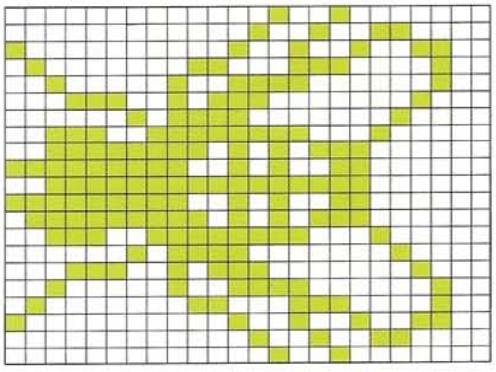
CRICKET

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
236,235,183,228,235,245,160
107,132,32,8,4,16,8
4,8,16,8,4,32,16
3,64,0,0,0,0,0
0



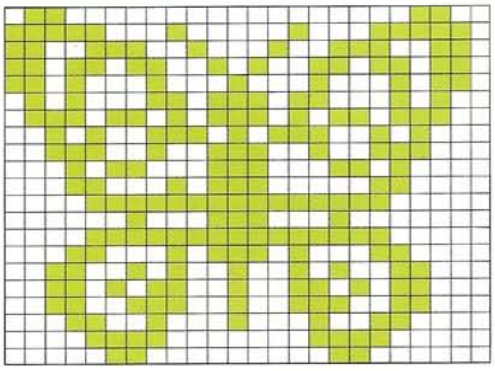
FLY

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
0



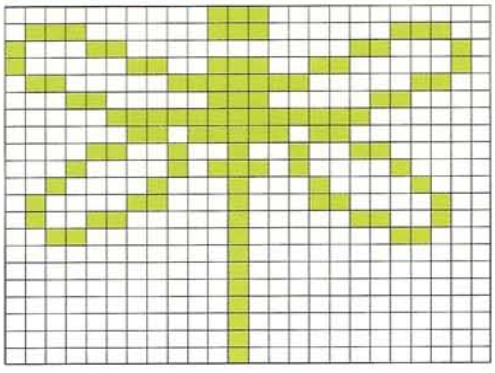
BUTTERFLY

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
0



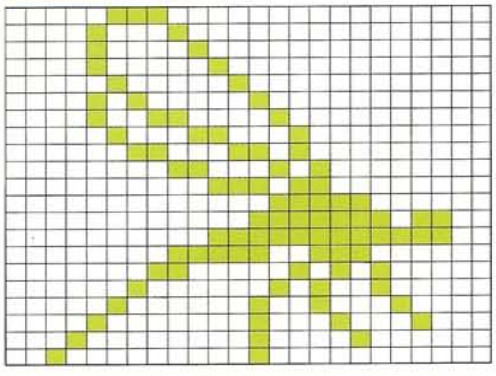
DRAGONFLY

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
0



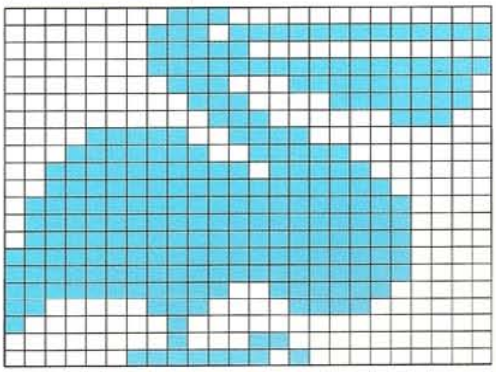
DRAGONFLY

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
0



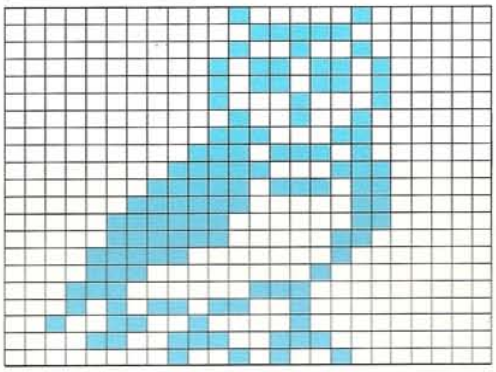
PELICAN

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
0



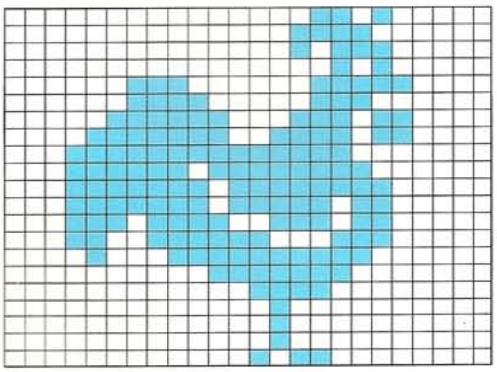
OWL

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
0



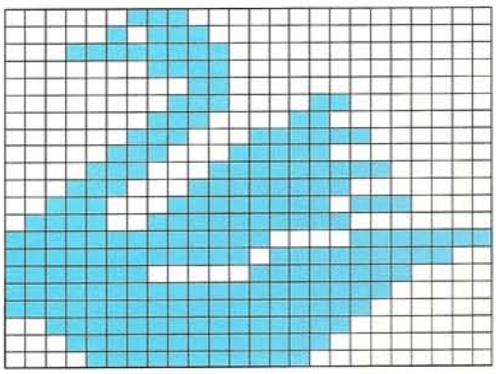
ROOSTER

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
0



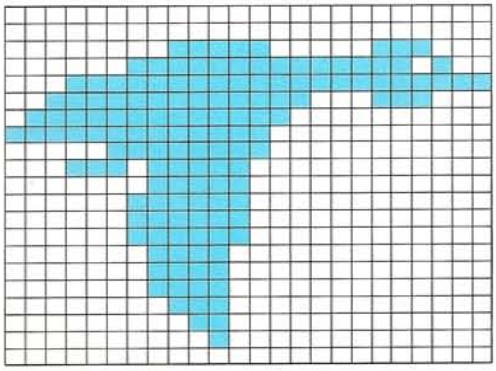
SWAN

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



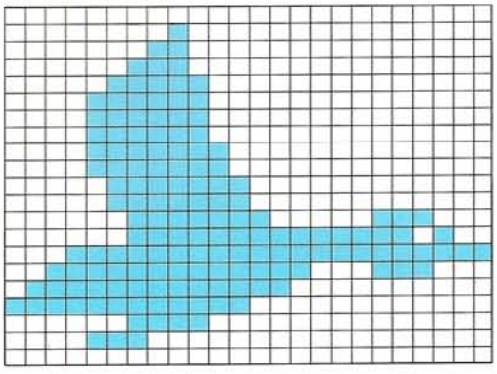
DUCK

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
0



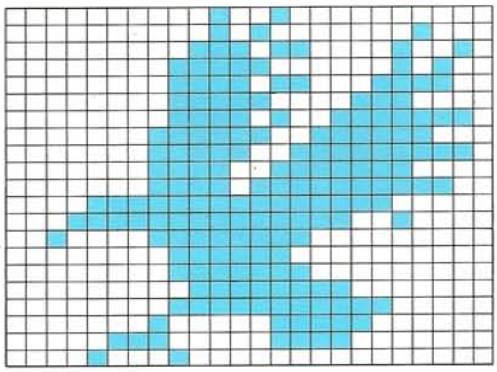
DUCK

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
0



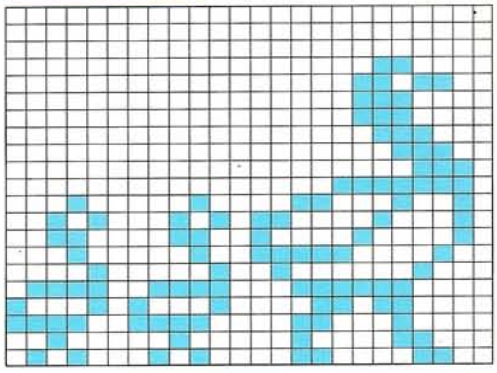
EAGLE

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
0



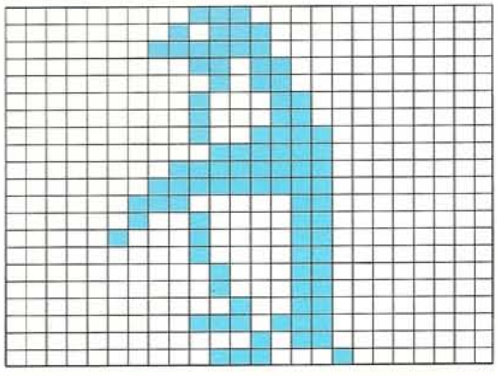
DUCK AND DUCKLINGS

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
0



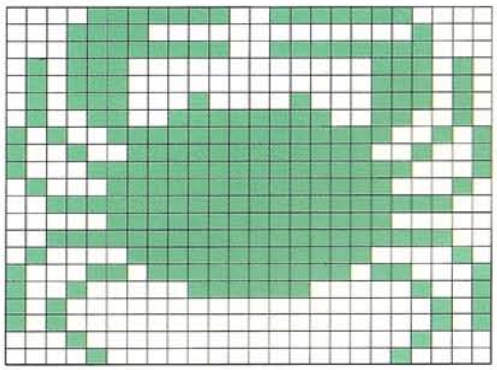
PENGUIN

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
0



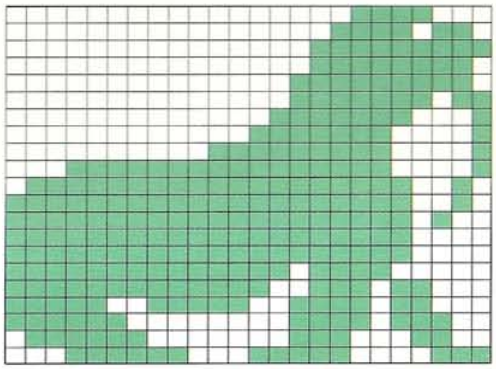
CRAB

```
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
0
```



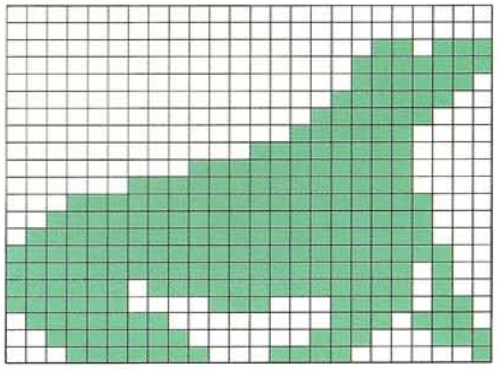
WALRUS

```
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
0
```



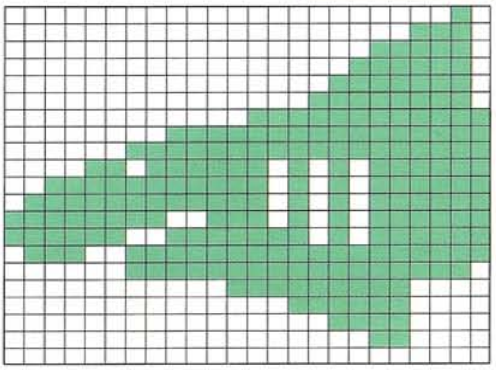
SEAL

```
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
0
```



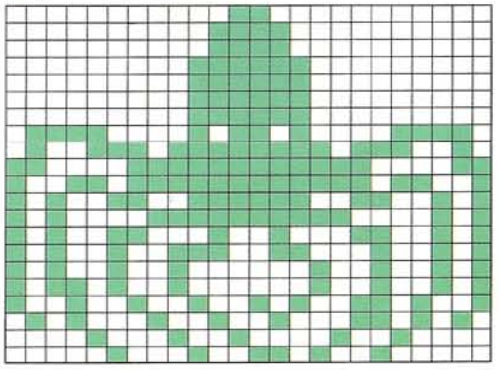
SHARK

```
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
0
```

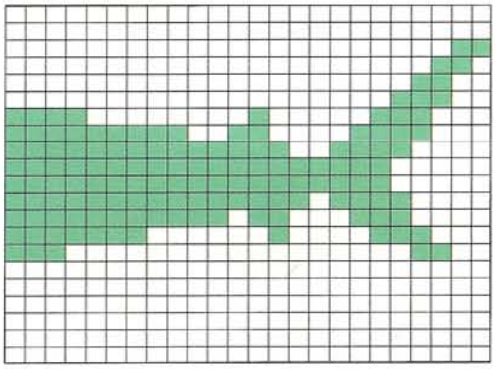


OCTOPUS

```
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
0
```

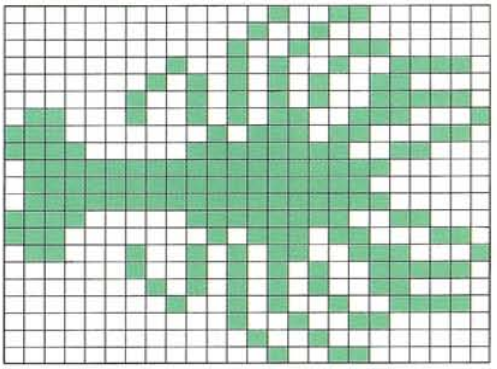


```
0,0,0,0,0,0,0
0,3,0,0,6,0,0
12,0,0,28,240,8,56
255,152,120,255,252,240,255
255,224,255,255,224,255,254
240,255,236,112,254,4,56
224,0,12,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0
```



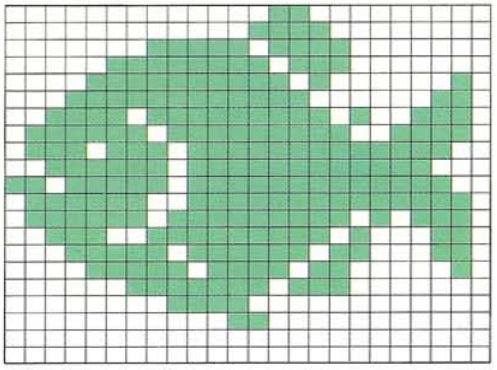
LOBSTER

```
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
0
```



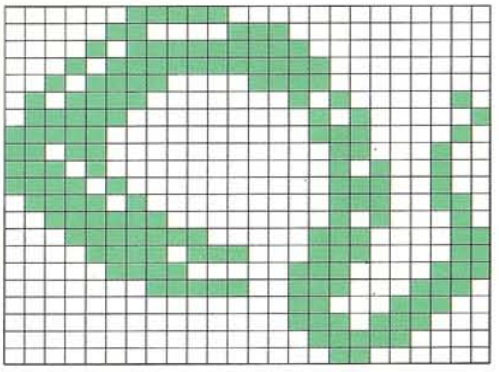
PIRANHA

```
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
0
```



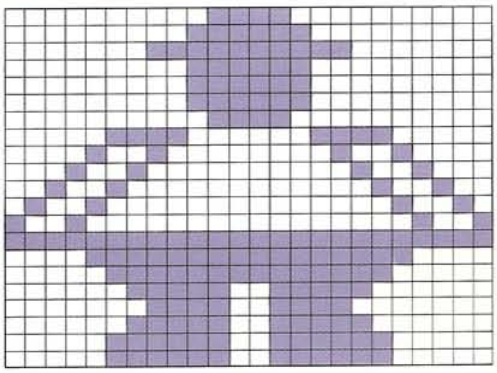
MORAY EEL

```
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
0
```



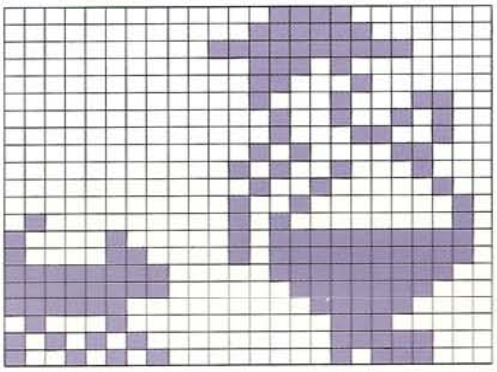
SHERRIFF

```
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
0
```



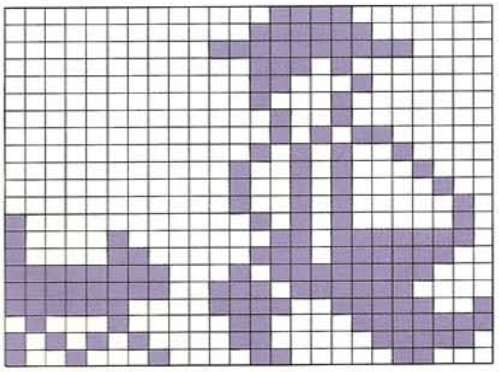
SHERRIFF

```
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
0
```



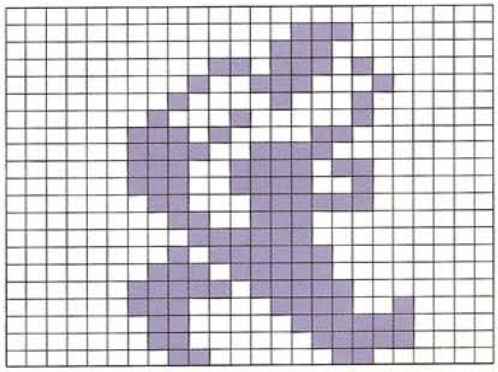
SHERRIFF

```
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
18,136,0,18,132,0,18
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
0
```



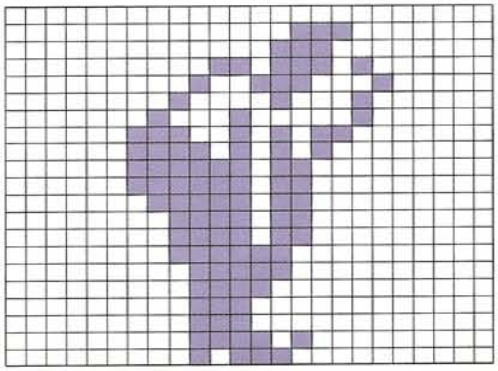
HUNCHBACK

```
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
0
```



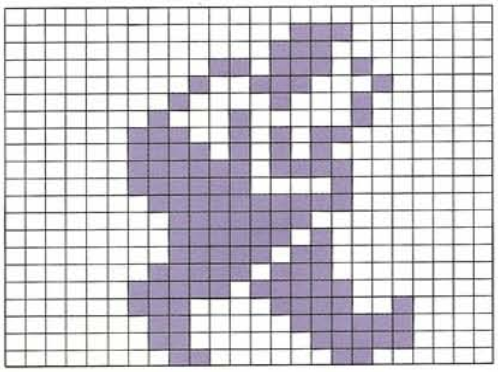
HUNCHBACK

```
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
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
0
```



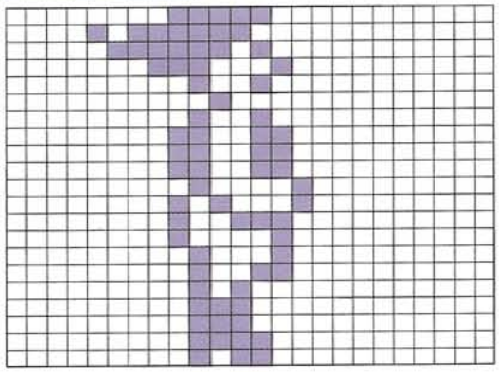
HUNCHBACK

```
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
0
```



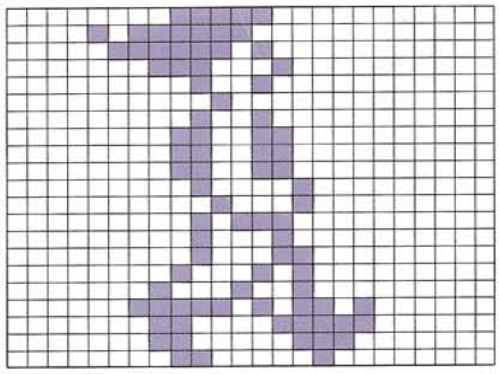
DWARF

```
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
204,0,0,66,0,0,162
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
0
```



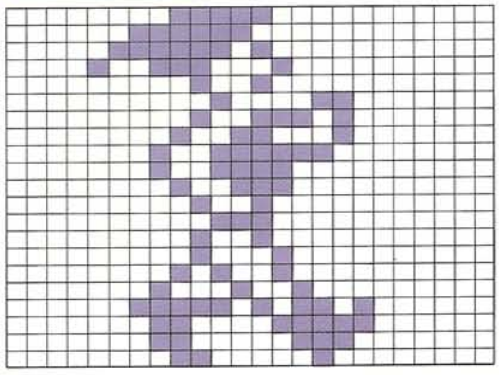
DWARF

```
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
0
```



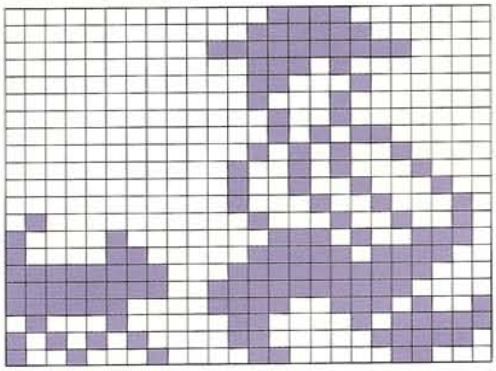
DWARF

```
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
0
```



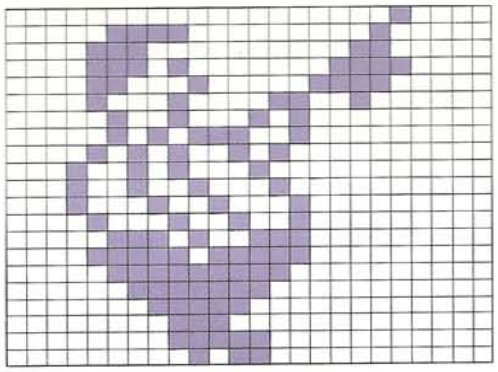
SHERRIFF

```
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
0
```



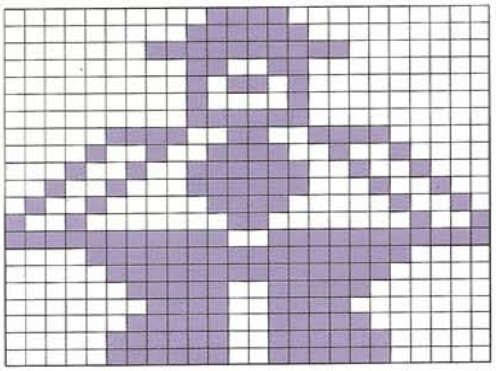
SHERRIFF

```
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
0
```



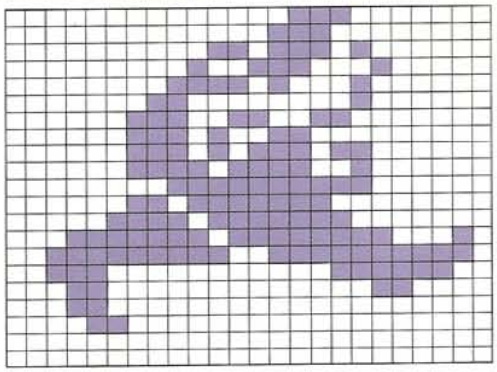
SHERRIFF

```
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
0
```



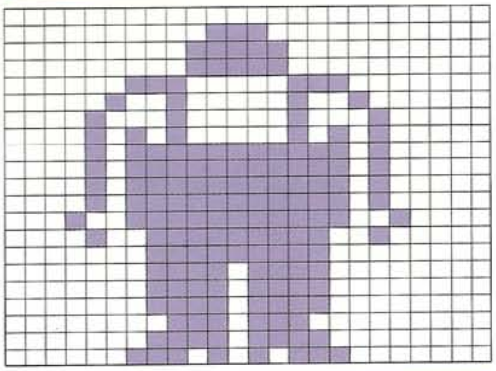
HUNCHBACK

```
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
0
```



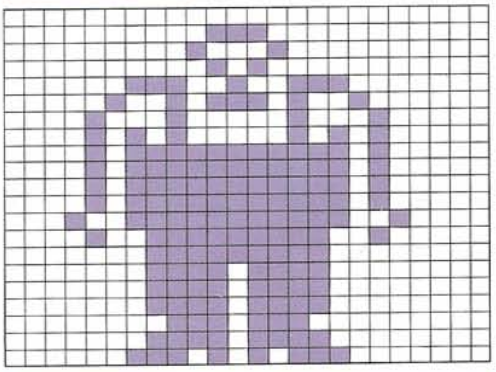
HUNCHBACK

```
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
0
```



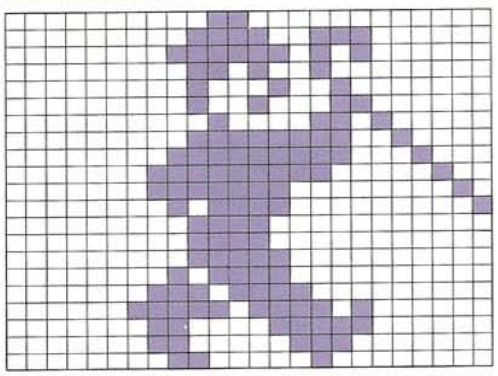
HUNCHBACK

```
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
0
```



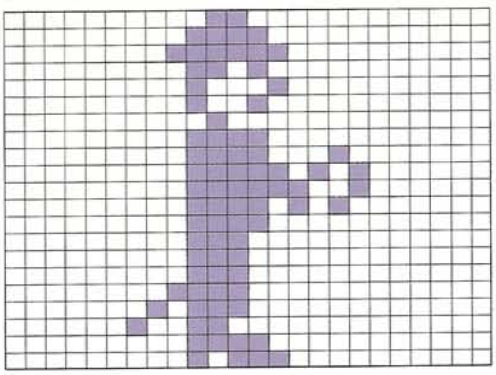
CHARLIE CHAPLIN

```
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
0
```



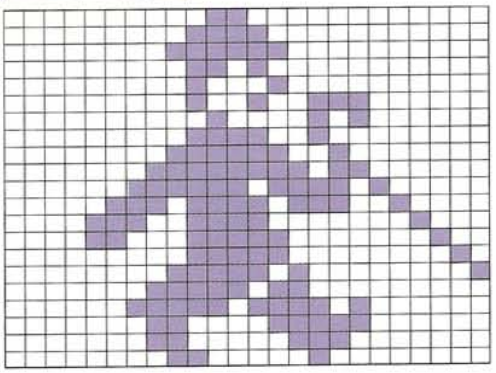
CHARLIE CHAPLIN

```
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
0
```



CHARLIE CHAPLIN

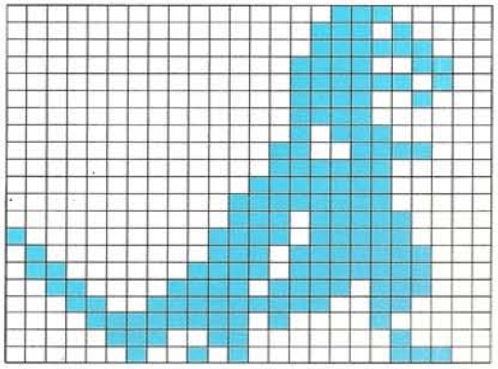
```
0,48,0,0,120,0,0
252,0,0,104,0,0,68
0,0,73,192,0,33,64
0,121,0,0,252,128,1
254,192,3,247,160,7,123
144,6,120,8,0,124,4
0,124,2,0,250,1,1
246,0,3,239,64,1,135
192,1,131,128,0,193,0
0
```



DINOSAURS

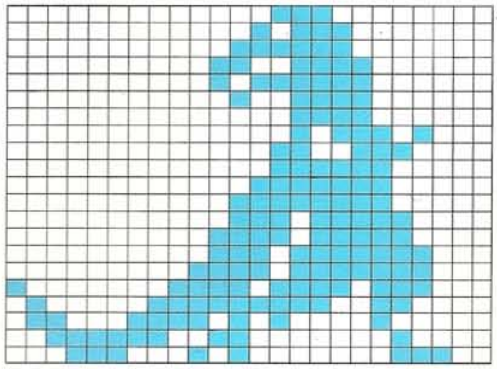
TYRANNOSAURUS

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
0



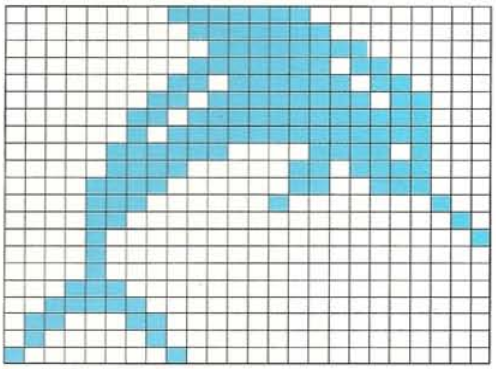
TYRANNOSAURUS

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
0



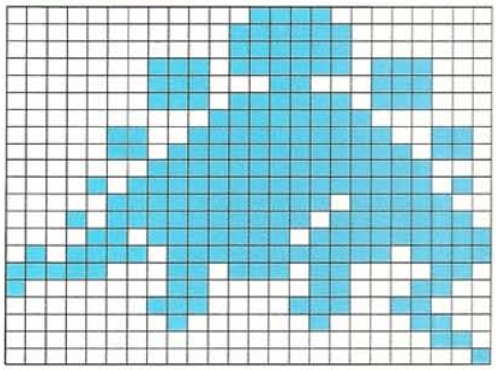
ICHTHYOSAURUS

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
0



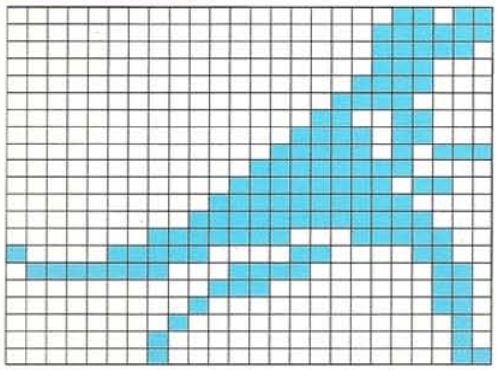
STEGOSAURUS

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
0



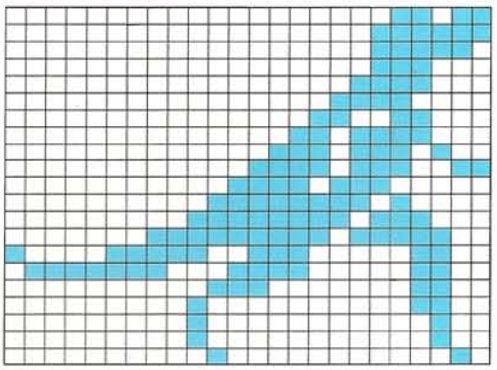
ALLOSAURUS

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
0



ALLOSAURUS

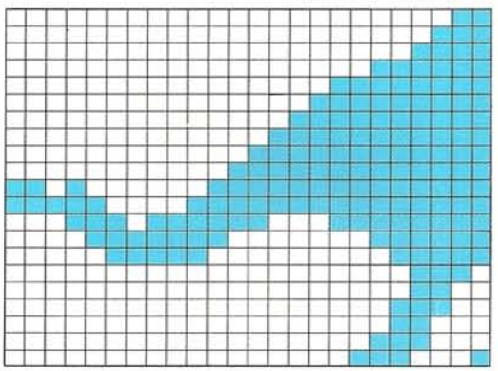
0,0,27,0,0,63,0
0,62,0,0,56,0,0
120,0,0,240,0,1,152
0,3,120,0,7,116,0
15,115,0,30,224,0,63
208,0,127,184,1,255,124
135,206,12,127,30,12,0
56,12,0,64,12,0,64
4,0,64,4,0,32,2
0

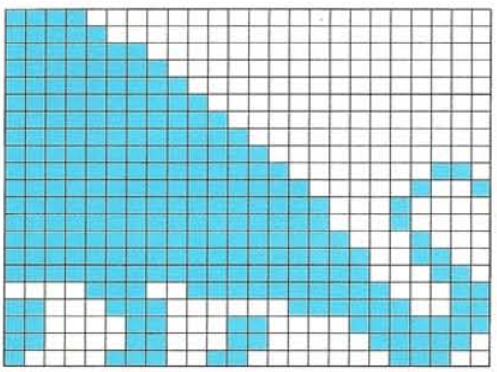


BRONTOSAURUS

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
0

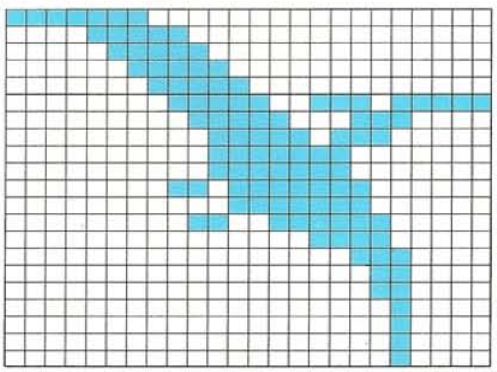
240,0,0,248,0,0,254
0,0,255,0,0,255,128
0,255,192,0,255,224,0
255,240,0,255,248,0,255
252,6,255,254,9,255,255
16,255,255,16,255,255,136
255,255,204,255,255,198,14
127,227,199,3,243,195,24
127,195,24,62,135,48,28
0





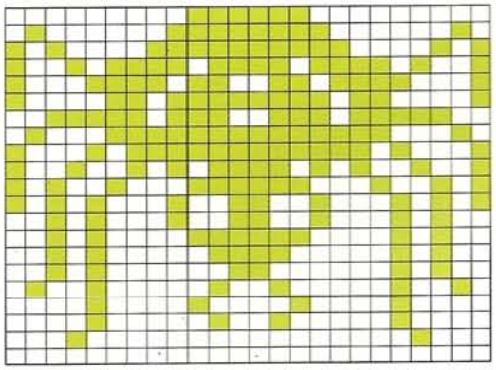
PTERANODON

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
0



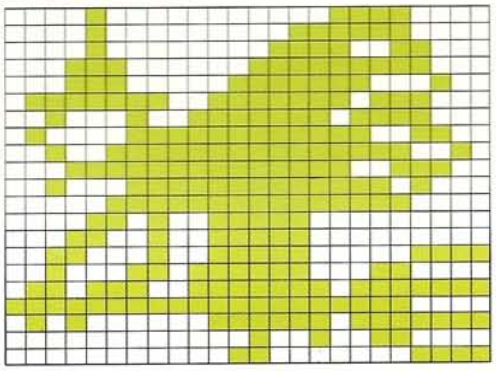
SPIDER

```
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
0
```



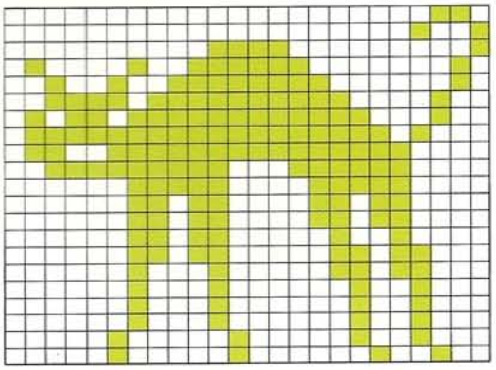
WITCH

```
8,0,240,0,3,240,0
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
0
```



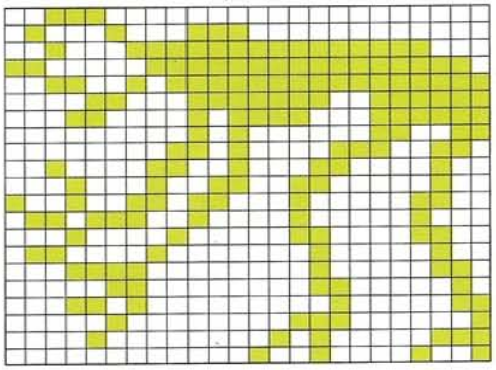
BLACK CAT

```
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
0
```



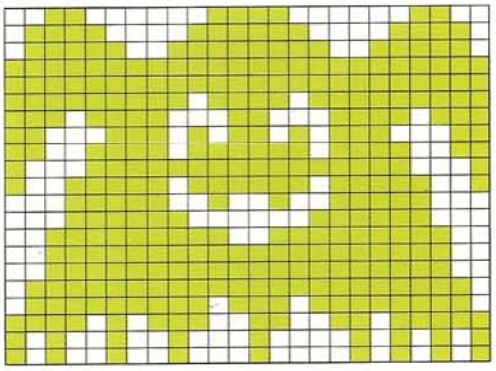
SPECTER

```
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
0
```



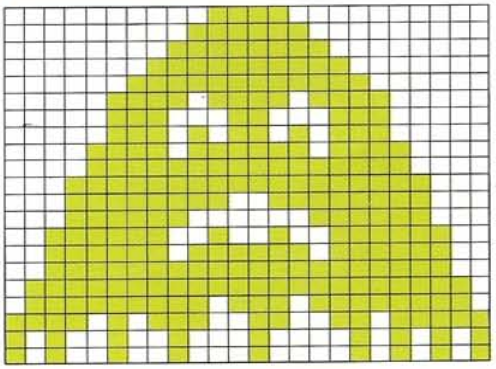
SPOOK

```
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
0
```



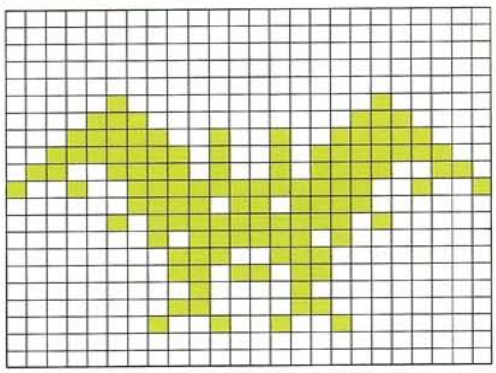
SPOOK

```
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
0
```



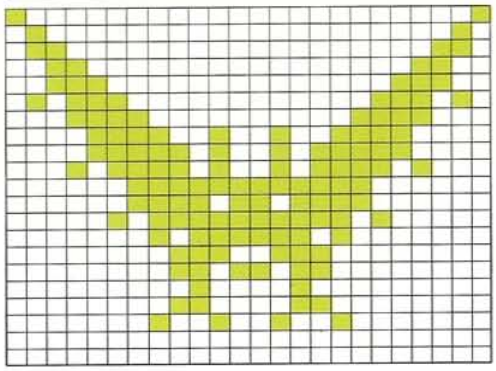
BAT

```
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
0
```



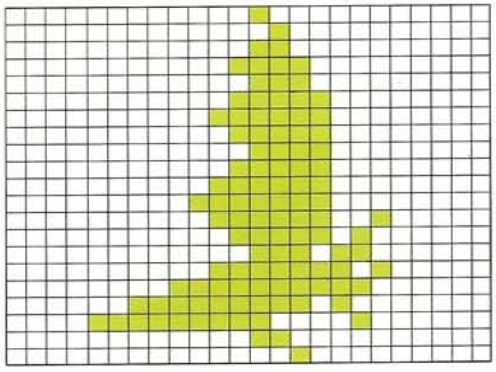
BAT

```
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
0
```



BAT

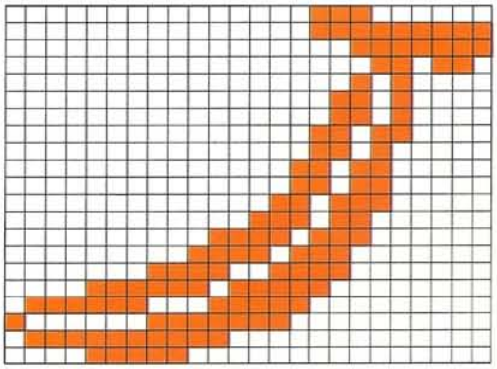
```
0,8,0,0,4,0,0
12,0,0,30,0,0,14
0,0,31,0,0;63,0
0,31,0,0,15,0,0
31,0,0,63,0,0,127
0,0,63,32,0,31,64
0,14,192,0,63,160,0
255,192,3,254,128,15,248
64,0,12,0,0,2,0
0
```



GAMES SYMBOLS

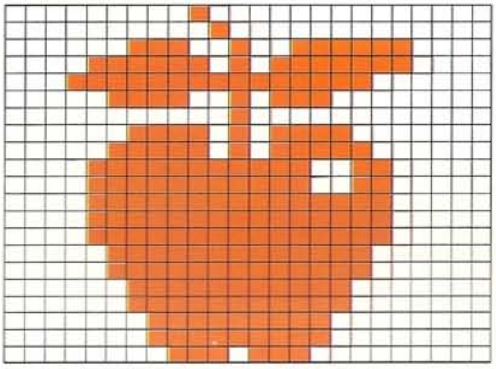
BANANA

```
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
0
```



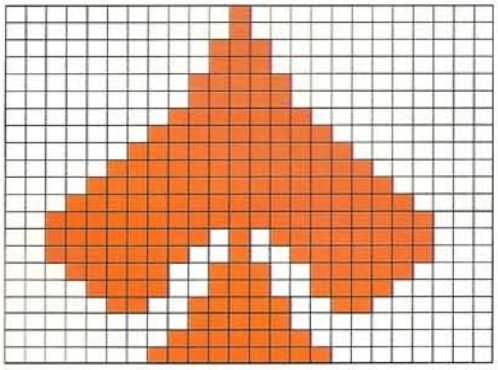
APPLE

```
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
0
```



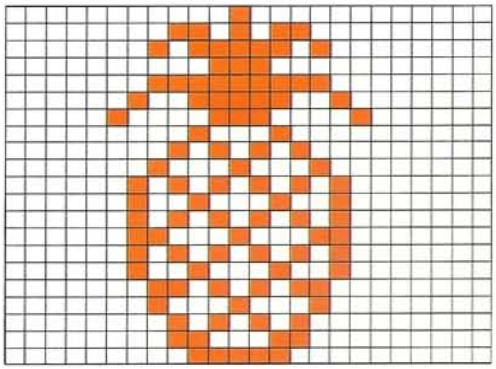
SPADE

```
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
0
```



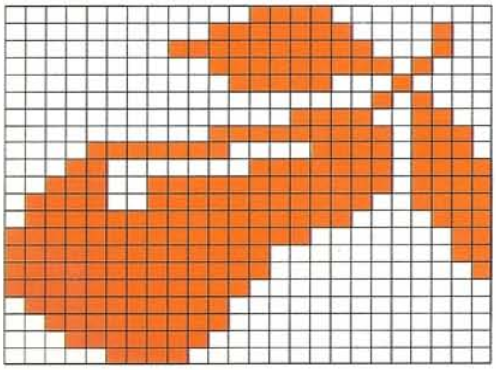
PINEAPPLE

```
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
0
```



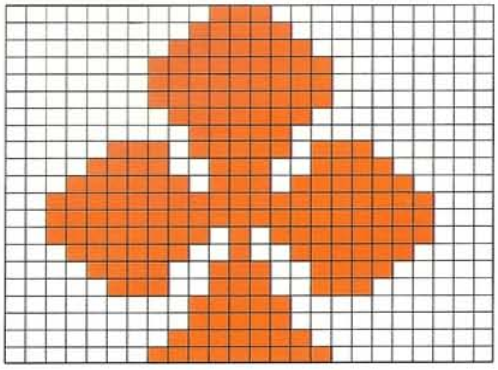
PEAR

```
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
0
```



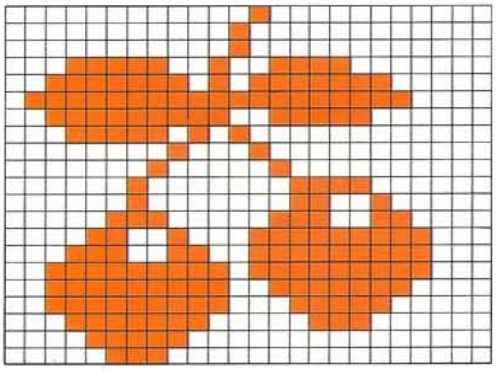
CLUB

```
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
0
```



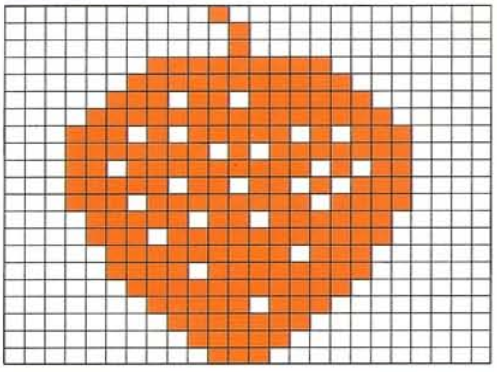
CHERRIES

```
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
0
```



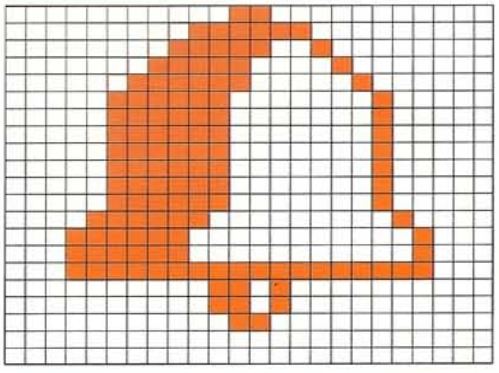
STRAWBERRY

```
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
0
```



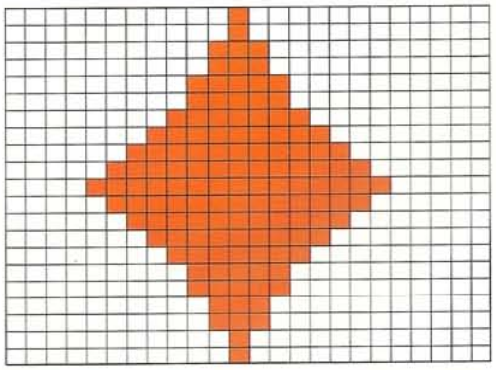
BELL

```
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
0
```



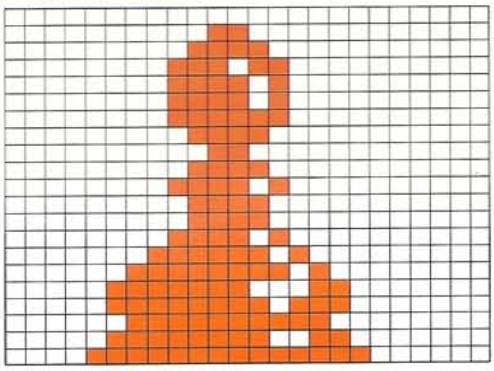
DIAMOND

```
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
0
```



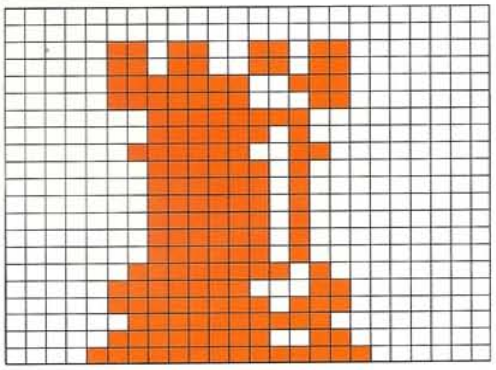
PAWN

```
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
0
```



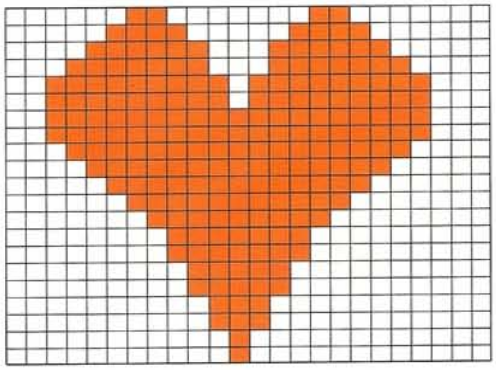
ROOK

```
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
0
```



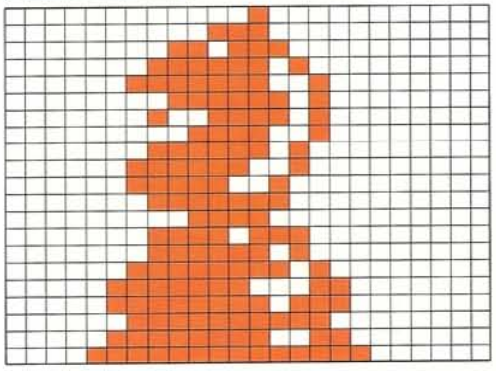
HEART

```
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
0
```



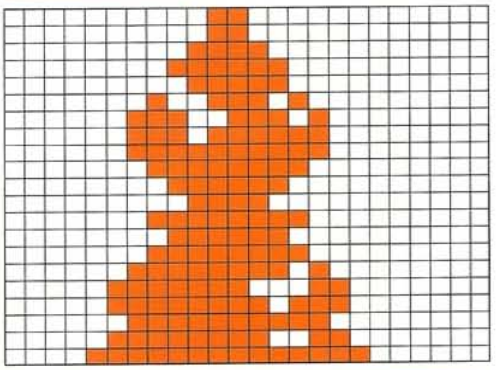
KNIGHT

```
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
0
```



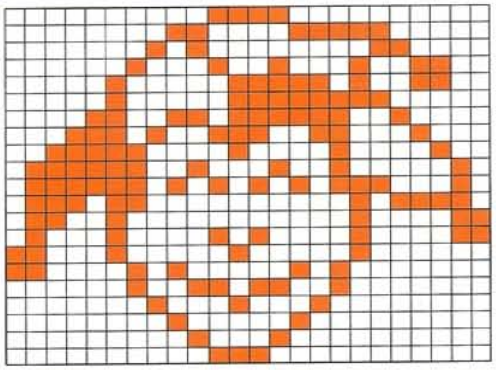
BISHOP

```
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
0
```



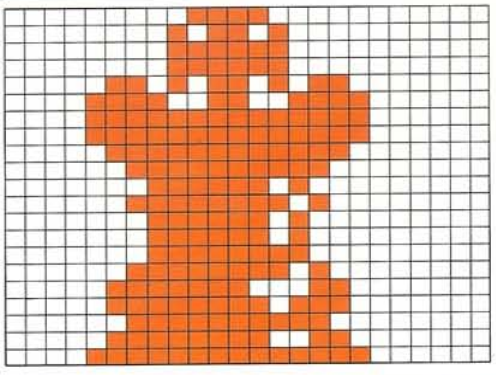
JOKER

```
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
0
```



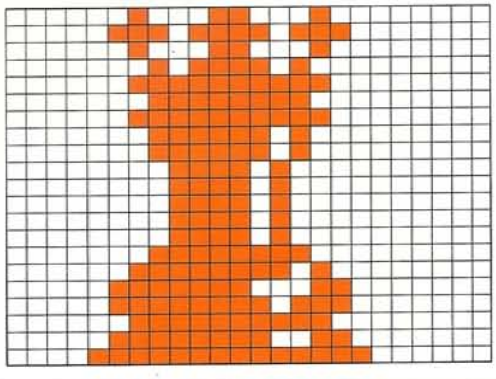
KING

```
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
0
```



QUEEN

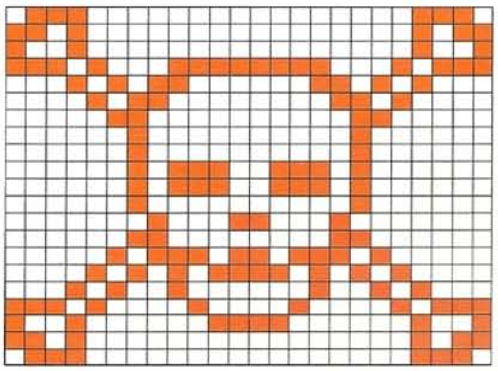
```
2,49,0,7,123,128,2
49,0,1,122,0,3,255
0,1,254,0,3,255,0
1,250,0,1,250,0,0
244,0,0,244,0,0,244
0,0,244,0,0,244,0
1,254,0,3,253,0,7
241,128,7,251,128,3,255
0,7,249,128,15,255,192
0
```



GAMES SYMBOLS

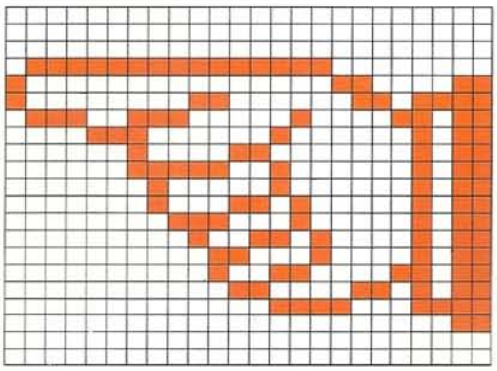
SKULL AND CROSSBONES

```
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
0
```



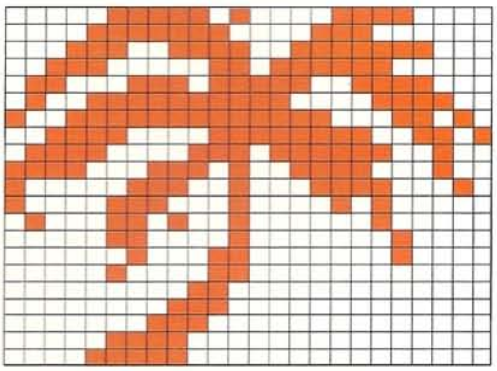
POINTING HAND

```
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
0
```



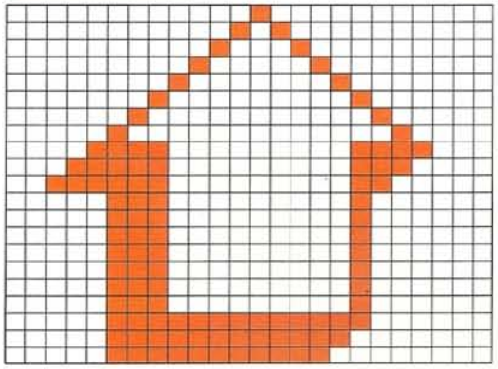
PALM TREE

```
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
0
```



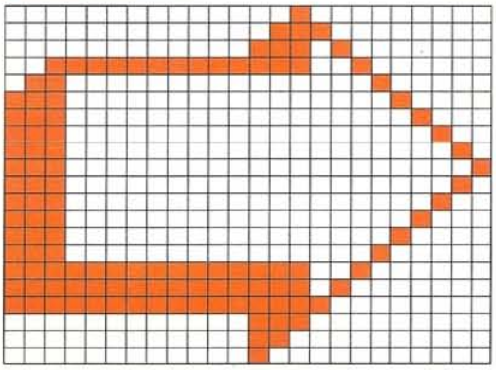
ARROW

```
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
0
```



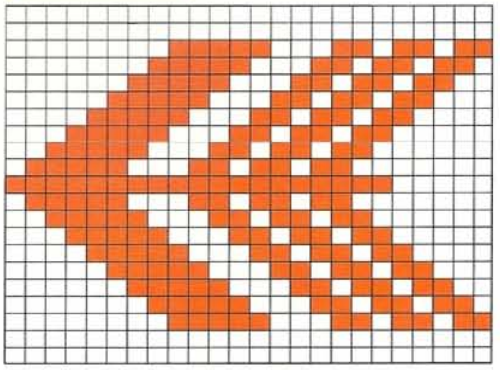
ARROW

```
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
0
```



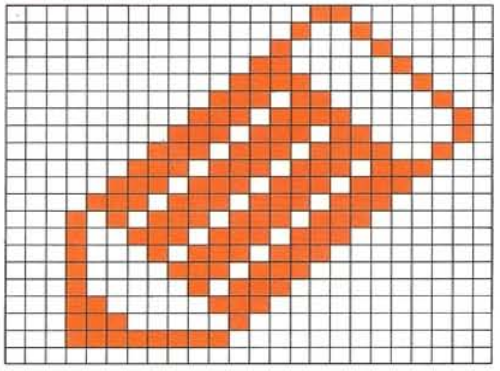
ARROW

```
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
0
```



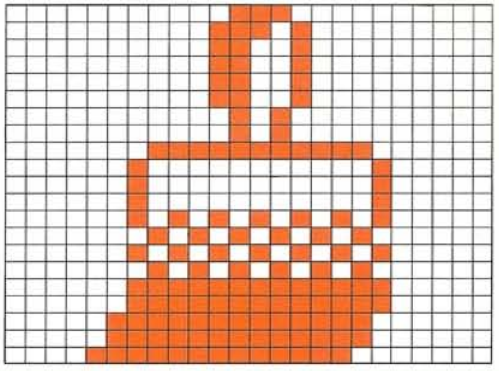
PENCIL

```
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
0
```



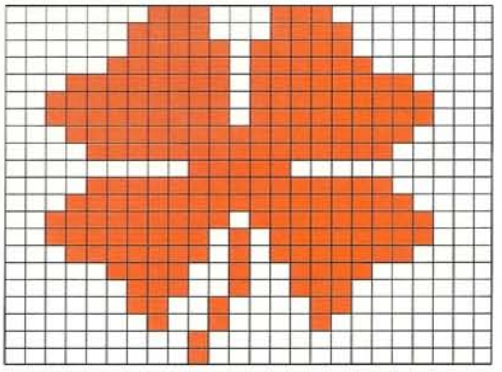
PAINTBRUSH

```
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
0
```



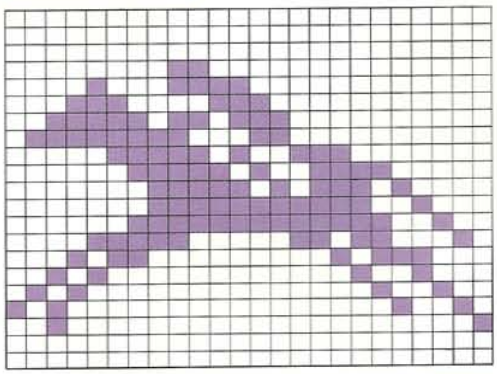
SHAMROCK

```
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
0
```



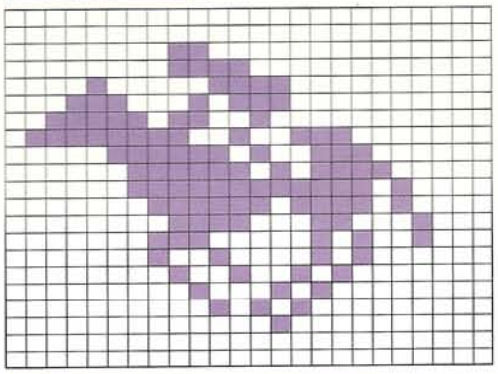
HORSE AND JOCKEY

```
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
0
```



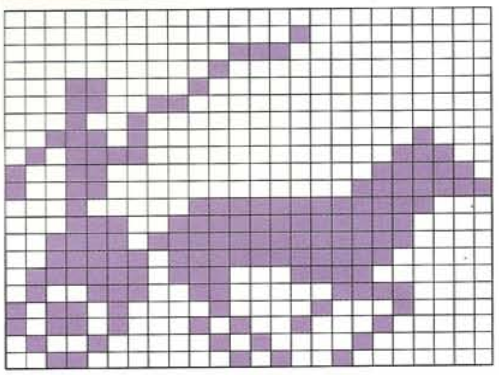
HORSE AND JOCKEY

```
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
0
```



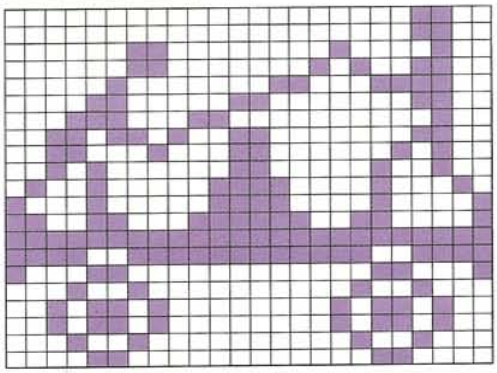
CHARIOT

```
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
0
```



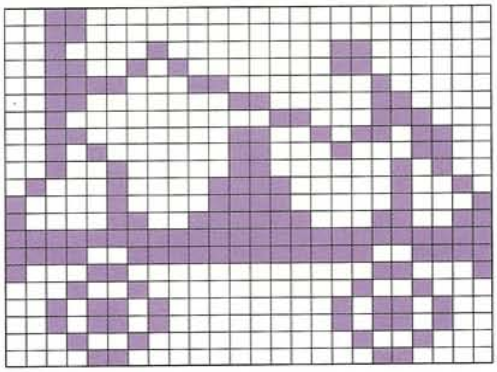
TROLLEY

```
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
0
```



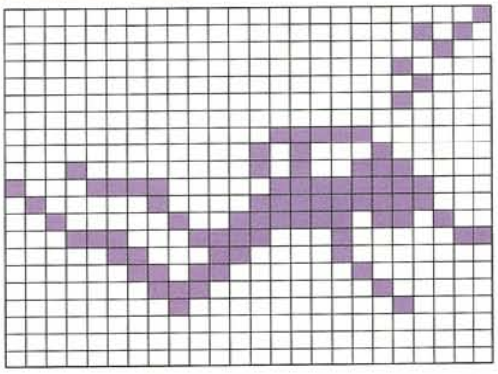
TROLLEY

```
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
0
```



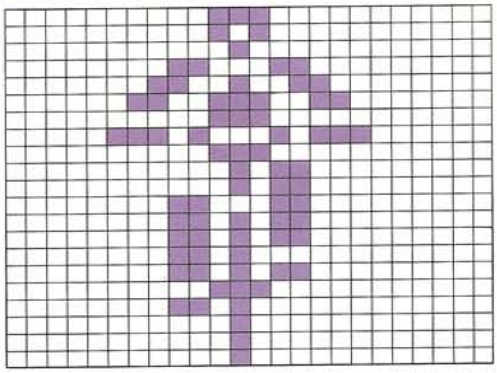
FROGMAN

```
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
0
```



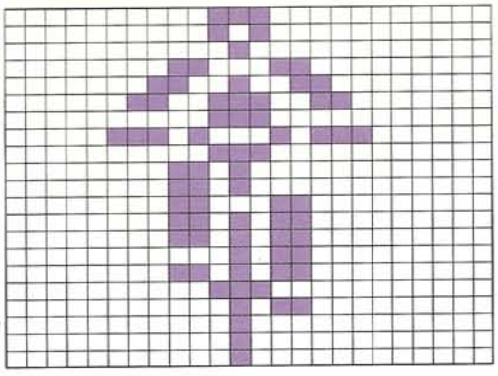
BMX RIDER

```
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
0
```



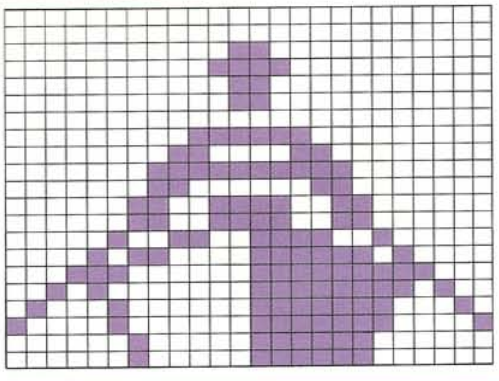
BMX RIDER

```
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
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
0
```



MACHINE-CODE ROUTINES

The listings featured on these two pages are used to put machine-code instructions into memory in order to speed up Commodore graphics programs. If you have read Book Three in this series, you will recognize them as a selected number of the routine blocks featured in that book. If you have not used the routines in these blocks before, don't worry. All you need to do is key them in when the text says they are required. Don't try to key them in and run them on their own, because they are designed only to be called by a main program. Apart from making the computer put numbers into its memory, they will not produce anything if you try to run them in isolation.

How to use the routine blocks

Each block contains one or more machine-code routines, and these are listed at the top of each block. To run a program using these routines, you must have keyed in the appropriate blocks before adding the main program.

There are two programs in this book which need machine-code routines — the Fruit Machine program on pages 20-23 and the Sprite Editor program on pages 24-32. The Fruit Machine program uses only some of the routines (you can see which in the panel on page 20) while the Sprite Editor program needs all of them.

To enable any machine-code routine to work, you must key in the block listing in its entirety. You should enter the blocks needed before you start on the rest of the program. Make sure that you enter them in the order of their lines — don't, for example, key in block D before block C.

When you run a program which uses these routines, there will first be a pause while the machine-code is put into memory. Subsequently, this process will be skipped, and the program will immediately start producing results on the screen.

BLOCK F

ERASE routine

Note
This routine is used only by the Sprite Editor program. Key it in directly after the draw routine (block D). Do not attempt to use the routine in your own programs without first reading the instructions in Book Three.

```
3200 IF PEEK(50560)=32 THEN 3230
3210 SYS A3,3240 : FOR C=50560 TO 50578
3220 READ B : POKE C,B : NEXT C
3230 F1=50560
3240 DATA 32,40,192,152,208,7,138
3250 DATA 208,4,141,29,192,96,169
3260 DATA 1,141,29,192,96
```

BLOCK A

HIGH-RESOLUTION, LOW-RESOLUTION, RESTORE, RESCUE and MERGE routines

```
100 IF PEEK(49192)=32 THEN 130
110 RESTORE : FOR C=49152 TO 49404
120 READ B : POKE C,B : NEXT C
130 A1=49237 : A2=49254
140 A3=49209 : A4=49271
150 A5=49297
160 DATA 0,0,0,0,0,0,0,0
170 DATA 0,0,0,0,0,0,0,0
180 DATA 0,0,0,0,0,0,0,0
190 DATA 0,0,0,0,0,0,0,0

200 DATA 0,0,0,0,0,0,0,0
210 DATA 0,0,0,0,0,32,121
220 DATA 0,32,253,174,32,138,173
230 DATA 32,191,177,166,100,164,101
240 DATA 96,32,40,192,132,63,132
250 DATA 20,134,64,134,21,32,19
260 DATA 166,165,95,56,233,1,133
270 DATA 65,165,96,233,0,133,66
280 DATA 96,169,8,13,24,208,141
290 DATA 24,208,169,32,13,17,208

300 DATA 141,17,208,96,169,247,45
310 DATA 24,208,141,24,208,169,223
320 DATA 45,17,208,141,17,208,96
330 DATA 169,8,160,1,145,43,32
340 DATA 51,165,165,34,133,45,133
350 DATA 47,133,49,165,35,133,46
360 DATA 133,48,133,50,96,32,121
370 DATA 0,32,253,174,169,0,133
380 DATA 10,32,212,225,165,43,72
390 DATA 165,44,72,56,165,45,233

400 DATA 2,133,43,165,46,233,0
410 DATA 133,44,169,0,133,185,166
420 DATA 43,164,44,32,213,255,176
430 DATA 14,134,45,132,46,32,51
440 DATA 165,104,133,44,104,133,43
450 DATA 96,170,201,4,208,5,164
460 DATA 186,136,240,206,104,133,44
470 DATA 104,133,43,24,108,0,3
480 DATA 224,2,176,7,224,1,176
490 DATA 1,96,192,64,96,56,138
500 DATA 208,3,152,201,200,96,105
510 DATA 0,133,46,133,48,133,50
520 DATA 96
```

BLOCK B

CLEAR-AND-COLOR and BLOCK-COLOR routines

```
600 IF PEEK(49408)=173 THEN 630
610 SYS A3,650 : FOR C=49408 TO 49682
620 READ B : POKE C,B : NEXT C
630 B1=49559
640 B2=49634
650 DATA 173,8,192,72,41,7,141
660 DATA 1,192,104,41,248,133,253
670 DATA 173,12,192,72,41,7,141
680 DATA 2,192,104,41,248,72,74
690 DATA 74,74,141,3,192,74,74

700 DATA 24,109,3,192,141,3,192
710 DATA 104,10,10,10,13,2,192
720 DATA 24,101,253,133,253,173,3
730 DATA 192,109,9,192,24,105,32
740 DATA 133,254,169,128,174,1,192
750 DATA 232,202,240,4,74,56,176
760 DATA 249,141,0,192,96,173,26
770 DATA 192,41,248,141,26,192,24
780 DATA 110,27,192,110,26,192,110
790 DATA 27,192,110,26,192,110,27
```

```
800 DATA 192,110,26,192,169,0,133
810 DATA 252,173,28,192,74,74,74
820 DATA 141,4,192,10,10,24,109
830 DATA 4,192,10,10,38,252,10
840 DATA 38,252,24,109,26,192,133
850 DATA 251,169,4,101,252,109,27
860 DATA 192,133,252,96,32,40,192
870 DATA 140,5,192,169,32,133,252
880 DATA 169,0,133,251,165,252,201
890 DATA 63,208,6,165,251,201,64

900 DATA 240,13,162,0,138,129,251
910 DATA 230,251,208,235,230,252,208
920 DATA 231,169,4,133,252,169,0
930 DATA 133,251,165,252,201,7,208
940 DATA 6,165,251,201,232,240,15
950 DATA 162,0,173,5,192,129,251
960 DATA 230,251,208,233,230,252,208
970 DATA 229,96,32,40,192,32,224
980 DATA 192,144,7,32,6,169,32
990 DATA 251,168,96,142,27,192,140

1000 DATA 26,192,32,40,192,32,236
1010 DATA 192,176,235,140,28,192,32
1020 DATA 40,192,140,5,192,32,82
1030 DATA 193,160,0,173,5,192,145
1040 DATA 251,96
```

BLOCK C

PLOT routine

```
1200 IF PEEK(49712)=32 THEN 1230
1210 SYS A3,1240 : FOR C=49712 TO 49787
1220 READ B : POKE C,B : NEXT C
1230 C1=49712
1240 DATA 32,40,192,32,224,192,169
1250 DATA 0,144,2,165,1,141,24
1260 DATA 192,142,9,192,140,8,192
1270 DATA 32,40,192,32,236,192,169
1280 DATA 0,144,2,169,1,13,24
1290 DATA 192,141,24,192,142,13,192

1300 DATA 140,12,192,240,1,96,32
1310 DATA 0,193,160,0,173,29,192
1320 DATA 201,0,240,8,177,253,77
1330 DATA 0,192,145,253,96,177,253
1340 DATA 13,0,192,145,253,96
```

BLOCK D

DRAW routine

```
1500 IF PEEK(49792)=173 THEN 1530
1510 SYS A3,1540 : FOR C=49792 TO 50180
1520 READ B : POKE C,B : NEXT C
1530 DI=49792
1540 DATA 173,24,192,141,25,192,32
1550 DATA 40,192,32,224,192,169,0
1560 DATA 144,2,169,1,141,24,192
1570 DATA 142,7,192,140,6,192,32
1580 DATA 40,192,32,236,192,169,0
1590 DATA 144,2,169,1,13,24,192

1600 DATA 141,24,192,13,25,192,141
1610 DATA 25,192,142,11,192,140,10
1620 DATA 192,56,173,6,192,237,8
1630 DATA 192,170,173,7,192,237,9
1640 DATA 192,168,16,17,56,138,73
1650 DATA 255,105,0,170,152,73,255
1660 DATA 105,0,168,169,255,208,17
1670 DATA 208,4,224,0,240,9,169
1680 DATA 0,141,15,192,169,1,208
1690 DATA 5,169,0,141,15,192,141

1700 DATA 14,192,142,18,192,140,19
1710 DATA 192,56,173,10,192,237,12
1720 DATA 192,170,173,11,192,237,13
1730 DATA 192,168,16,17,56,138,73
1740 DATA 255,105,0,170,152,73,255
1750 DATA 105,0,168,169,255,208,17
1760 DATA 208,4,224,0,240,9,169
1770 DATA 0,141,17,192,169,1,208
1780 DATA 5,169,0,141,17,192,141
1790 DATA 16,192,142,20,192,140,21
```

```
1800 DATA 192,142,22,192,140,23,192
1810 DATA 173,29,192,208,3,32,205
1820 DATA 195,173,23,192,48,45,208
1830 DATA 5,173,22,192,240,38,56
1840 DATA 173,22,192,237,18,192,141
1850 DATA 22,192,173,23,192,237,19
1860 DATA 192,141,23,192,24,173,12
1870 DATA 192,109,16,192,141,12,192
1880 DATA 173,13,192,109,17,192,141
1890 DATA 13,192,173,23,192,48,7

1900 DATA 208,43,173,22,192,208,38
1910 DATA 24,173,22,192,109,20,192
1920 DATA 141,22,192,173,23,192,109
1930 DATA 21,192,141,23,192,24,173
1940 DATA 8,192,109,14,192,141,8
1950 DATA 192,173,9,192,109,15,192
1960 DATA 141,9,192,32,205,195,173
1970 DATA 8,192,205,6,192,208,145
1980 DATA 173,9,192,205,7,192,208
1990 DATA 137,173,12,192,205,10,192

2000 DATA 208,129,173,13,192,205,11
2010 DATA 192,208,230,96,173,25,192
2020 DATA 208,28,32,0,193,160,0
2030 DATA 173,29,192,201,0,240,8
2040 DATA 177,253,77,0,192,145,253
2050 DATA 96,177,253,13,0,192,145
2060 DATA 253,96,174,9,192,172,8
2070 DATA 192,32,224,192,176,11,174
2080 DATA 13,192,172,12,192,32,236
2090 DATA 192,144,206,96
```

BLOCK H

ROM-COPY and TEXT routines

```
4200 IF PEEK(51104)=173 THEN 4230
4210 SYS A3,4240 : FOR C=51104 TO 51306
4220 READ B : POKE C,B : NEXT C
4230 H1=51104 : H2=51167
4240 DATA 173,14,220,41,254,141,14
4250 DATA 220,165,1,41,251,133,1
4260 DATA 160,0,132,253,132,251,169
4270 DATA 17,133,254,169,209,133,252
4280 DATA 32,215,199,230,254,169,208
4290 DATA 133,252,32,215,199,165,1

4300 DATA 9,4,133,1,173,14,220
4310 DATA 9,1,141,14,220,96,177
4320 DATA 251,145,253,200,208,249,96
4330 DATA 32,40,192,32,224,192,144
4340 DATA 3,76,234,193,152,41,248
4350 DATA 141,8,192,142,9,192,32
4360 DATA 40,192,32,236,192,176,236
4370 DATA 152,41,248,141,12,192,142
4380 DATA 13,192,32,121,0,32,253
4390 DATA 174,32,158,173,32,163,182

4400 DATA 141,2,198,169,0,141,1
4410 DATA 198,133,252,32,0,193,165
4420 DATA 254,201,64,144,7,165,253
4430 DATA 201,64,144,1,96,172,2
4440 DATA 198,208,1,96,136,140,2
4450 DATA 198,172,1,198,177,34,200
4460 DATA 140,1,198,10,38,252,10
4470 DATA 38,252,10,38,252,133,251
4480 DATA 165,252,105,16,133,252,160
4490 DATA 7,177,251,145,253,136,16
4500 DATA 249,24,165,253,105,8,133
4510 DATA 253,165,254,105,0,133,254
4520 DATA 169,0,133,252,76,29,200
```

BLOCK I

DEFINE-CHARACTER routine

```
4600 IF PEEK(51328)=169 THEN 4630
4610 SYS A3,4640 : FOR C=51328 TO 51377
4620 READ B : POKE C,B : NEXT C
4630 I1=51328
4640 DATA 169,0,133,252,32,40,192
4650 DATA 152,10,38,252,10,38,252
4660 DATA 10,38,252,133,251,24,165
4670 DATA 252,105,16,133,252,169,0
4680 DATA 141,4,192,32,40,192,152
4690 DATA 172,4,192,145,251,200,140
4700 DATA 4,192,152,201,8,208,238
4710 DATA 96
```

SPRITEMAKING CHECKLIST

The checklist below shows you which sprite feature each VIC register controls. Some registers control features that have just two possible states, like "turn sprite on" and "turn sprite off". In cases like this, a single register controls all eight sprites, with individual sprites being controlled by one bit within the register. The rest of the registers control features which have more than two states, like position. In these cases, each register controls a single sprite and allows any of 256 different states to be specified.

SPRITE PROGRAMMING REFERENCE CHART

VIC	Effect	Sprite(s) controlled by register	How to use the register
V+0–V+14 (even numbers)	Set horizontal positions	V+0 = sprite 0 V+14 = sprite 7	Key in register followed by a horizontal coordinate (0-255)
V+1–V+15 (odd numbers)	Set vertical positions	V+1 = sprite 0 V+15 = sprite 7	Key in register followed by a vertical coordinate (0-255)
V+16	Specifies horizontal position in either left or right half of screen	All sprites	Key in register followed by the total bit values for the sprites you want to appear in the right-hand half of the screen
V+21	Turns on sprites	All sprites	Key in register followed by the total bit values of the sprites you want to turn on
V+23	Expands sprites vertically	All sprites	Key in register followed by the total bit values of the sprites you want to be expanded
V+28	Turns on the multi-color mode for one or more sprites	All sprites	Key in register followed by the total bit values of the sprites you want to be multi-color
V+29	Expands sprites horizontally	All sprites	Key in register followed by the total bit values of the sprites you want to be expanded
V+30	Records sprite-sprite collisions	All sprites	PEEK the register. The bit values making this up indicate which sprite has collided with another
V+31	Records sprite-background collisions	All sprites	PEEK the register. The bit values making this up indicate which sprite has collided with the background
V+37	Sets first multi-color	All sprites	Key in register followed by code for the first multi-color
V+38	Sets second multi-color	All sprites	Key in register followed by code for the second multi-color
V+39–V+46	Sets sprite color	V+39 = sprite 0 V+46 = sprite 7	Key in register followed by a color code

Color and multi-color

Normally, sprites are shown in a single color which is set by a color code (see the table below). If you want to produce multi-color sprites, first switch on the multi-color mode with register V+28, and then select two additional colors by putting a color code into registers V+37 and V+38. In the multi-color mode, pixels are treated in pairs — you cannot specify them individually. To work out a DATA number for each row of a sprite, take each pixel pair in turn, note its position along its 8-pixel row. Select its color and use the table to find its contribution to the DATA number.

COMMODORE COLOR CODES

0 Black	4 Purple	8 Orange	12 Medium gray
1 White	5 Green	9 Brown	13 Light green
2 Red	6 Blue	10 Light red	14 Light blue
3 Cyan	7 Yellow	11 Dark gray	15 Light gray

MULTI-COLOR TABLE

Color	Pixel pair number			
	0	1	2	3
Screen background	0	0	0	0
Multi-color register 1 (V+37)	64	16	4	1
Color register (V+39 — V+46)	128	32	8	2
Multi-color register 2 (V+38)	192	48	12	3

SPRITEMAKING GRID

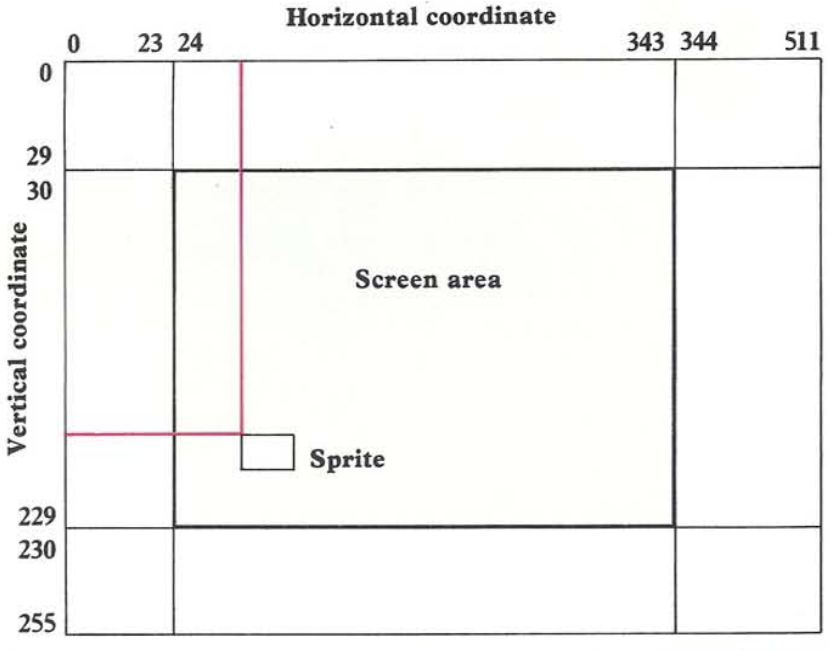
To work out DATA values for sprites, you can either use the Sprite Editor program, which will produce the values automatically, or you can use the grid below. Each number can then be POKEd into memory to specify a sprite row. To use the grid, first pencil in a design, and then add up the bit values for each group of eight pixels. You can then record the totals in the columns on the right, prior to putting them into memory. To code a multi-color sprite, you will need to compile the DATA in a different way, described on the opposite page.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
0																								
1																								
2																								
3																								
4																								
5																								
6																								
7																								
8																								
9																								
10																								
11																								
12																								
13																								
14																								
15																								
16																								
17																								
18																								
19																								
20																								

DATA values		

Positioning sprites

Sprite positioning coordinates can have any value from 0-255 vertically or 0-511 horizontally. However, only part of this range is visible on the screen. The diagram on the right shows how these coordinates relate to the screen. The central panel is the visible area of the screen. This extends from 24 to 343 horizontally and from 30 to 229 vertically. This means that you can move sprites on and off the screen smoothly. Because vertical position runs from 0 to 255, it can be controlled by a single byte of information. Horizontal position, on the other hand, needs two bytes if the whole of the range from 0 to 511 (=2x255) is to be used. If you want a sprite to continue moving past horizontal position 255, the V+16 location must be turned on for the sprite concerned.



INDEX

Acknowledgments
Dorling Kindersley would like to thank all those who helped in the preparation of this book, especially Steve Wilson (design), James Burnie and Rachel Cornes (program checking), Fred Gill (proofreading), and Richard Bird (indexing).

DK

Screen Shot

PROGRAMMING SERIES

The bestselling teach-yourself programming course now offers the first complete full-colour book on Commodore 64 sprites.

Illustrated with over 300 screen-shot photographs, it contains programs for single and multicolour sprites, animation, setting priorities and detecting collisions, and stretching and enlarging, 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 the Commodore 64.

"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

ISBN 0-86318-088-4

9 780863 180880