First published in
Great Britain in 1985 by
Franklin Watts
12a Golden Square
London W1

First published in the
United States in 1985 by
Gloucester Press

Copyright © Aladdin Books Ltd 1985

Printed in Belgium

ISBN 0 531 17013 6

Library of Congress
Catalog Card Number: 85 709 54
WRITE YOUR OWN PROGRAM

COMPUTER ANIMATION

HAUNTED HOUSE

Marcus Milton

ORRADRE LIBRARY
UNIVERSITY OF SANTA CLARA
SANTA CLARA, CALIFORNIA

GLOUCESTER PRESS
NEW YORK · TORONTO · 1985
Foreword

There is much more to computer graphics than you may think. Instead of just playing or illustrating games, you can write a story, and make your own colorful cartoon film program using both static and animated graphics. The program is given for the Apple IIe and Commodore 64 computers, and tells the story of the Haunted House.

The program has been broken down into logical stages and a running text explains how each section works. Writing a program in logical stages not only makes it easier for you and other people to understand, it also reduces the possibility of errors. At various stages you can test the sections of the program you've keyed-in – even the smallest error can mean that the program doesn't work. If something does go wrong, check back through the listings very carefully. Make sure your program is exactly the same as is printed in the book.

The sinister goings-on around the haunted house use many useful computer graphics techniques. For example, by carefully designing some user-defined characters it is easy to get the smoothest animation from a BASIC program.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLE IIe graphics</td>
<td>8</td>
</tr>
<tr>
<td>COMMODORE 64 graphics</td>
<td>10</td>
</tr>
<tr>
<td>The storyboard</td>
<td>12</td>
</tr>
<tr>
<td>THE CONTROL PROGRAM AND INITIALISATION</td>
<td>13</td>
</tr>
<tr>
<td>APPLE IIe</td>
<td>14</td>
</tr>
<tr>
<td>COMMODORE 64</td>
<td>17</td>
</tr>
<tr>
<td>DAYTIME</td>
<td>19</td>
</tr>
<tr>
<td>APPLE IIe</td>
<td>20</td>
</tr>
<tr>
<td>COMMODORE 64</td>
<td>24</td>
</tr>
<tr>
<td>MIDNIGHT</td>
<td>29</td>
</tr>
<tr>
<td>APPLE IIe</td>
<td>30</td>
</tr>
<tr>
<td>COMMODORE 64</td>
<td>35</td>
</tr>
<tr>
<td>Program listings</td>
<td>39</td>
</tr>
<tr>
<td>Glossary</td>
<td>42</td>
</tr>
<tr>
<td>Index</td>
<td>44</td>
</tr>
</tbody>
</table>
APPLE Graphics

The Apple stores the details of the shapes that it is going to draw on the screen in special shape tables. That way a shape can be put into memory and called when it is needed. The shapes are numbered and at the beginning of the shape table there is an index so that the computer can find the particular shape you want drawn. The index contains the number of memory locations the beginning of the shape is away from the start of the table. Memory locations 232 and 233 tell the computer where the table is.

```
3000 READ SN: REM SHAPE TABLE SET UP
3010 POKE P,SN: POKE P + 1,0:ST = P + 2
3020 POKE 232, FN MOD(P): POKE 233, INT (P / 256)
3030 LET P = ST + 2 * SN
3050 FOR SS = 0 TO SN - 1: T = P - ST + 2
3060 LET A = ST + 2 * SS: POKE A, FN MOD(T)
3070 POKE A + 1, INT (T / 256): D$ = "":R = 0
```

To fully understand how Apple shape tables work you have to understand machine codes. But to make it easier a special BASIC routine has been written which translates simple direction commands into the right form. That routine is called by the GOSUB line 3070, after the overall format of the table has been set.

THE APPLES' SHAPE TABLES ARE MADE UP OF VECTORS. THESE ARE INSTRUCTIONS L MEANS GO LEFT, R MEANS GO RIGHT, U MEANS GO UP AND D MEANS GO DOWN. A P BEFORE THESE DIRECTIONS PRINTS ON THE SCREEN, AN M MOVES WITHOUT PRINTING. E MEANS END.

100 DATA PRUURDDD5UR5D3RD3L5DL5U6LDDLLUULE
3100 LET X = X + B: FOR I = 1 TO R
3110 IF PEEK (P - 1) > 8 OR X > 8 THEN 3130
3120 POKE P - 1, 0 * X + PEEK (P - 1): GOTO 3140
3130 POKE P, X: P = P + 1
3140 NEXT : R = 0
3150 IF D$ = "" THEN READ D$: PRINT ".":
3160 LET A$ = LEFT$(D$, 1)
3170 LET D$ = MID$(D$, 2, LEN(D$) - 1)
3180 IF A$ = "R" THEN X = 1: GOTO 3100
3190 IF A$ = "D" THEN X = 2: GOTO 3100
3200 IF A$ = "L" THEN X = 3: GOTO 3100
3210 IF A$ = "U" THEN X = 128: GOTO 3100
3220 IF A$ = "M" THEN B = 0: GOTO 3150
3230 IF A$ = "P" THEN B = 4: GOTO 3150
3240 IF A$ = " " THEN 3150
3250 IF A$ < "0" OR A$ > "9" THEN 3270
3260 LET R = 10 * R + VAL (A$): GOTO 3150
3270 POKE P, X: POKE P + 1, 0: P = P + 2: RETURN

This is the routine that translates the simple direction instructions that are going to be given in the drawing data into the complicated numbers that the shape table requires. In line 3150, the DATA is READ in as string and PRINTs a dot on the screen to show that the computer is working during this long procedure. Lines 3160 and 3170 pick off the first number or letter off the string. Lines 3180 through 3250 look for the direction letters we're using or numbers. Lines 3100 through 3140 do the complicated sums that work out the shape table numbers and POKE them into the right place in the shape table.

YOU CAN MAKE AN AIRPLANE FLY ACROSS YOUR SCREEN BY TYPING IN LINES 3000 TO 3270 WITH THIS SHORT PROGRAM AND LINE 100 OPPOSITE.

10 HGR: HCOLOR = 3
15 P = 24576: ROT = 0: SCALE = 1
20 GOSUB 3000
25 X DRAW 1 AT 10, 20
30 FOR X = 11 TO 270
35 X DRAW 1 AT X - 1, 20
40 X DRAW 1 AT X, 20
45 NEXT
50 X DRAW 1 AT 270, 20
55 TEXT: END
60 DATA 1
COMMODORE Graphics

To draw the haunted house, you have to use what is called "high resolution graphics" where each little dot on the TV screen is controlled individually. But Commodore BASIC has no simple command for drawing a line in high resolution graphics. So the program below loads a machine code routine which will do the job. You have to type this program in and RUN it before the main program will work.

```
5 REM *** STARTER PROGRAM ***
10 FOR I=0 TO 214: READ D: CS=CS+D
20 POKE 49152+I,D: NEXT I
30 IF CS<>29713 THEN PRINT "DATA ERROR!":END
40 POKE6442,64:SYS64766
100 DATA0,32,253,174,32,158,183,134,253,32,253,174
110 DATA32,158,183,138,72,32,253,174,32,158,183,138
120 DATA41,7,135,254,138,74,74,74,74,133,251,10,10
130 DATA101,251,160,6,162,0,134,252,42,38,252,136
140 DATA208,250,101,254,133,251,165,253,41,252,10,144
150 DATA3,250,252,24,101,251,133,251,165,252,105,32
160 DATA33,252,32,253,174,32,158,183,138,141,0,192
170 DATA160,3,10,10,13,0,192,136,208,248,141,0
180 DATA192,165,253,74,74,133,255,104,133,254,74,74
190 DATA56,229,255,240,52,133,255,32,178,192,32,149
200 DATA192,169,8,24,101,251,133,251,165,252,105,0
210 DATA33,252,198,255,240,8,173,0,192,145,251,76
220 DATA121,192,32,197,192,72,73,255,49,251,145,251
230 DATA104,45,0,192,17,251,145,251,96,32,178,192
240 DATA133,255,32,197,192,37,255,76,149,192,169,255
250 DATAT2,165,253,41,3,240,6,170,104,74,74,202
260 DATA208,251,96,104,96,169,255,72,169,252,5,254
270 DATA170,104,232,208,1,96,10,10,76,206,192
```

The numbers in the DATA lines are the machine code and the loop between lines 10 and 20 READs these numbers and POKEs them into the computer's memory. But to check that you do not make a mistake when you are keying them in, line 10 also adds the numbers up. The total is checked in line 30 which then tells you if you have made a mistake. Line 40 makes sure that when the main program is loaded it goes into an area of memory where it won't interfere with the machine code. The machine code is called by the command SYS MC. This command is followed by four numbers which tell the machine code where the ends of the line are, how far down the screen and what color it is.
This subroutine uses the machine code put into memory by the starter program to draw a box. So RUN the starter program, type in this routine, then test it using the test program below. When it is working, delete the test lines. Y1, Y2, X1 and X2 are the row and column numbers that form the sides of the box. The loop between lines 13010 and 13030 POKEs the color memory of each character square in the box. And the loop between 13060 and 13100 uses SYS MC to call the machine code routine which fills in the box a line at a time with that color. The loop between 13080 and 13100 works back up the box, using SYS MC to change the color of every other line to orange to give stripes. If NS is fixed at 1 outside the program the machine code routine does not draw in the stripes.

BEFORE YOU GO ANY FARTHER IT IS BEST TO TEST THAT YOUR MACHINE CODE ROUTINE AND BLOCK-DRAWING ROUTINE IS WORKING. THIS LITTLE TEST PROGRAM WILL DO JUST THAT.

1 B=8192:MC=49153:BM=53265:CH=53270:SC=1024:CO=55296
2 MP=53272:Y1=0:Y2=24:X1=0:X2=39
3 POKE MP,PEEK(MP)OR8:POKE(BM),PEEK(BM)OR32
4 POKE CH,PEEK(CH)OR16:C=0:GOSUB 13000
5 X=15:X2=25:Y1=7:Y2=17:C=C+1:GOSUB13000:GOTO 5
The storyline of the cartoon must be worked out in detail before the main program is written. First the scene is set, then the mysterious castle appears in the middle of nowhere, surrounded by a spooky forest. Night falls. A thunderous lightning flash wakes the bats in the belfry and an evil old witch takes to the air on her broomstick. As she flies off into the night accompanied by more lightning flashes. Then suddenly the old house catches fire. The flames leap higher and higher. The house is burnt to the ground. And, as an eerie conclusion to this macabre tale, a huge ghostly skull appears in the sky, hovering over the ashes of the house.

When working on a cartoon tale like this one, it is a good idea to tackle the task like a professional movie maker. Work out a detailed storyboard, like the one below, that breaks the plot down into a series of separate events.

- Background
- Haunted House
- Haunted Forest
- Night Falls
- Bat and Witch
- Lightning Strike
- The Blazing House
- The House Collapses
- A Skull Appears
THE CONTROL PROGRAM AND INITIALIZATION

A control program is one that calls a series of subroutines in the right order. Here the control program corresponds to the storyboard and the subroutines correspond to each of the separate scenes.
The control program follows the storyboard, with each subroutine being called in turn. At first they simply set up the machine and draw up each element of the scenery. But when the action starts things get a bit more complicated.

```
1 REM HAUNTED HOUSE
5 DEF FN MOD(X) = 256 * (X / 256 - INT(X / 256))
100 GOSUB 1000: REM INITIALIZATIONS
110 GOSUB 6000: REM BACKGROUND & CASTLE
120 GOSUB 6500: REM TREES
130 GOSUB 5000: REM NIGHT FALL
140 GOSUB 2400: REM LIGHTNING
150 FOR TI = 1 TO 250
160 IF TI = 10 THEN GOSUB 2100: REM START WITCH
170 IF TI > 10 THEN GOSUB 2150: REM MOVE WITCH
180 IF TI = 30 THEN GOSUB 2200: REM START BAT
190 IF TI > 30 THEN GOSUB 2250: REM MOVE BAT
200 IF RND (1) < .07 THEN GOSUB 2400: REM LIGHTNING
210 NEXT TI: GOSUB 2300: REM REMOVE BAT
220 GOSUB 2400: GOSUB 2400: REM STORM
230 GOSUB 2500: REM FIRE
240 FOR X = 1 TO 1000: NEXT: GOSUB 2700: REM SKULL
250 TEXT : END
```

Because of the special way machine code numbers have to be handled, line 5 starts off by defining the function FN MOD (X) which will help simplify the math. Next the initialization subroutine is called, which tells the computer all the things that it will need to know to run the program. Then the routines that draw up the background and castle, the trees, make night fall and make the lightning flash are all called. TI is going to be used as a clock to control the animation. The routines that draw up the witch, make her move and draw up the bat and make it move take their cue from TI. In line 200 the Apple's dice-rolling function RND is used to call the lightning program and give lightning flashes at random times.

Once the animation loop has ended, the computer goes on with the rest of line 210 and removes the bat. Line 220 calls the lightning routine twice in quick succession to simulate a storm. In line 230 fire breaks out - or, at least, it's set by calling the routine at line 2500.

Once the castle has burned down to the ground, there is a thousand time period pause to build up dramatic tension before the routine that puts up the eerie skull is called. Line 250 sets the Apple back to normal TEXT mode before ENDing the program.
When initializing any program it is best to give it all the important memory addresses names that will be easy to remember. **P1GE** and **P2GE** are the addresses that switch on graphics screens 1 and 2. **FULLSCREEN** is the address which switches the Apple to full graphics screen, rather than the one that mixes graphics with text. **CLICK** is the address that switches on the speaker. **P** is the pointer to the shape table. The address given here is the starting address of the shape table, but **P** will be updated as the shapes are loaded in. Line 1040 calls the subroutine that sets up the shape table, so now the computer has all the shapes at its fingertips. Next a subroutine is called that loads in a machine code routine. This routine copies graphics screen one onto the graphics screen two, changing all the colors at the same time. **HGR** puts the Apple into high-resolution graphics mode and **PEEK**ing **FULLSCREEN** selects the full graphics screen. **SCALE** and **ROT** say that the shapes should be drawn at the same size and at the same angle as they were given.

**SHAPE TABLE**

The starting address of the shape table (byte + 0) is stored at addresses 232 and 233. Byte + 0 contains the number of shapes. In this example there are 2.

Starting at byte + 2 there is a low byte/high byte index to the shape definitions relative to byte + 0.

After the index are the shape definitions themselves.

(All shape definitions end with a zero.)

<table>
<thead>
<tr>
<th>BYTE + 0</th>
<th>NUMBER OF SHAPES</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE + 1</td>
<td>UNUSED</td>
<td>(φ)</td>
</tr>
<tr>
<td>BYTE + 2</td>
<td>ADDRESS SHAPE DEF.1 (LOW)</td>
<td>6</td>
</tr>
<tr>
<td>BYTE + 3</td>
<td>ADDRESS SHAPE DEF.1 (HIGH)</td>
<td>(φ)</td>
</tr>
<tr>
<td>BYTE + 4</td>
<td>ADDRESS SHAPE DEF.2 (LOW)</td>
<td>38</td>
</tr>
<tr>
<td>BYTE + 5</td>
<td>ADDRESS SHAPE DEF.2 (HIGH)</td>
<td>(φ)</td>
</tr>
<tr>
<td>BYTE + 6</td>
<td>SHAPE DEF.1 START</td>
<td></td>
</tr>
<tr>
<td>BYTE + 37</td>
<td>SHAPE DEF.1 END</td>
<td>(φ)</td>
</tr>
<tr>
<td>BYTE + 38</td>
<td>SHAPE DEF.2 START</td>
<td></td>
</tr>
<tr>
<td>BYTE + 69</td>
<td>SHAPE DEF.2 END</td>
<td>(φ)</td>
</tr>
</tbody>
</table>
The structure of this control program follows the storyboard exactly. It calls the routines one after another. The only routines that are called more than once are those that move the witch and the bat and make the lightning. This is to make the witch and bat move continuously and the lightning flash repeatedly.

```
5 REM *** =====HAUNTED HOUSE===== ***
10 GOSUB 14000:REM INITIALIZE
20 GOSUB 10000:REM BACKGROUND
30 GOSUB 20000:REM CASTLE
40 GOSUB 8000:REM TREES
50 GOSUB 90000:REM NIGHT
60 POKE V+21, 243:DX=-1:DY=-1
70 FOR WX=10 TO 346 STEP 2:LI=RND(0)*20
80 IF LI>19 THEN GOSUB 10000:REM LIGHTNING
90 IF WX>255 THEN RS=255:POKE V+16, 3
100 POKE V+6, WX-RS:REM WITCH
110 GOSUB 10040:REM BAT
120 NEXT WX: POKE V+21, 240
130 GOSUB 11000:REM BURN
140 GOSUB 12000:REM SKULL
150 END
```

This program calls a number of subroutines – the name of each subroutine is given in the REM line following the GOSUB call. The Commodore has a special graphics facility called sprites. These allow you to move areas of animation across the screen easily. But they are controlled by POKEing a special area of memory. The POKE in line 60 switches on the sprites which contain the bat and the witch. DX and DY fix the direction in which the bat starts moving. Setting them both to -1 means that the bat starts out moving from right to left and up the screen. WX controls the movement of the witch. She starts off in column 10 and moves across the screen two steps at a time until she disappears off the right-hand side. The lightning is made to flash randomly by using the dice-throwing instruction RND. Roughly once in every 20 throws RND (0)*20 will come up with a number bigger than 19 and the lightning subroutine at 1000 will be called.

One of the problems with using sprites is that you have to deal directly with the Commodore's memory locations. But each memory location can only store a number between 0 and 255. But there are more than 255 columns across the screen. So when the witch reaches column 255, adjustments have to be made. These are done by lines 90 and 100. Line 120 switches the bat and the witch sprites off again when you’ve finished with them.
To initialize this program we have to put the computer into the right mode to give high resolution graphics. **READ** in the **DATA** for all the things that have to be drawn on the screen, and give easy-to-remember names to important memory locations. Line 14010 clears the screen, sets the background color to light blue and sets the border color to gray. Lines 14020 through 14040 put the computer into high resolution graphics mode. And lines 14060 through 14080 **READ** the **DATA** for the trees, the moon and the skull into the arrays **TR**, **MO** and **SK**. Line 14090 gives names to the important memory locations. **V** is the start address of the sprite memory. **SC** and **CO** are start addresses of the screen and color memories. And **MC** is the address of the special machine code high-resolution line-drawing program. Sprites are easy to use, so the windows of the castle will also be made from sprites. Lines 14100 through 14130 set their X and Y positions. Lines 14140 and 14150 set the starting positions of the witch and the bat. Lines 14160 through to 14190 set the colors and tells each sprite where to find its **DATA**. That **DATA** is then **READ** and **POKE**d into memory by line 14200. Line 14210 fills the screen with the sky color by using the machine code program.
Although there are 10 trees on the screen, only one lot of DATA is needed. All the trees look the same so the same DATA can be used each time a tree is drawn. The numbers in the DATA here tell our machine code program which color to use. A 4, though, is not a color recognized by that routine. So if a 4 is found no action is taken. In the tree DATA, you'll see that there are only 4s and 2s. The 4s do nothing and the 2s draw a point on the screen. If you look at the pattern of the 2s, you'll see that they draw out the shape of a weird, gnarled tree.

```
20000 REM DATA FOR TREE
20010 DATA4, 2, 4, 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 2, 4, 4, 4
20020 DATA2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 2, 4, 4
20030 DATA4, 4, 2, 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 2, 4, 4, 4
20040 DATA4, 4, 2, 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 2, 4, 4, 4
20050 DATA4, 4, 4, 2, 4, 4, 4, 4, 2, 2, 2, 4, 4, 2, 4, 2, 4
20060 DATA4, 4, 4, 4, 2, 4, 4, 4, 2, 4, 2, 4, 4, 2, 4, 4, 4
20070 DATA4, 4, 4, 4, 2, 4, 4, 4, 2, 4, 2, 4, 4, 2, 4, 4, 4
20080 DATA4, 4, 4, 2, 2, 4, 4, 4, 2, 2, 4, 4, 2, 2, 4, 4, 4
20090 DATA4, 4, 4, 2, 2, 4, 4, 4, 2, 2, 4, 4, 2, 2, 4, 4, 4
20100 DATA4, 4, 4, 2, 4, 4, 4, 2, 2, 2, 4, 2, 4, 4, 2, 4, 4
20110 DATA4, 4, 4, 2, 4, 4, 4, 2, 2, 2, 4, 2, 4, 4, 2, 4, 4
20120 DATA4, 4, 4, 4, 4, 4, 4, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4
20130 DATA4, 4, 4, 4, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4
20140 DATA4, 4, 4, 4, 4, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4
20150 DATA4, 4, 4, 4, 4, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4
20160 DATA4, 4, 4, 4, 4, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4
20170 DATA4, 4, 4, 4, 4, 2, 2, 2, 4, 2, 4, 4, 4, 4, 4, 4, 4
20180 DATA4, 4, 4, 4, 4, 2, 2, 2, 4, 2, 4, 4, 4, 4, 4, 4, 4
20190 DATA4, 4, 4, 4, 4, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4
20200 DATA4, 4, 4, 4, 2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4
20210 DATA4, 2, 2, 2, 4, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4
```
DAYTIME

The background scenery, the haunted castle and the spooky forest are drawn up on the screen during the daytime. Everything is now ready for the action, which only begins once the sun has gone down.
Not only does this program give you a BASIC routine to simplify the shape data, it also gives you one that simplifies the input of graphics data. You can use these routines – or routines like them – to simplify drawing on the screen in your own programs. This routine gives you new instructions that are not normally available in BASIC. Two main instructions are given here: BLOCK and TRIANGLE. All the instructions are followed by eight numbers. The first four specify the sides of a block or two corners of a triangle. The next two fix the third corner of the triangle and are zero with the BLOCK instruction.

The last two are the colors that the block or triangle is to be drawn in. There are two colors because the block and triangles are going to be striped. Added to that, there are two other new instructions which work with these two. ORIGIN fixes the point the sides of the block or the points of the triangle are measured from. This point is specified by the first two numbers, the rest are zero. And STOP switches off the special graphics routine. The sky and the grass are drawn as two blocks, the castle as three, one for the main body, and two turrets. The path is a long, thin triangle.

7000 REM SKY, GRASS & PATH.
7010 DATA ORIGIN, 0, 0, 0, 0, 0, 0, 0, 0
7020 DATA BLOCK, 0, 0, 279, 135, 0, 0, 0, 6, 6
7030 DATA BLOCK, 0, 136, 279, 191, 0, 0, 0, 1, 1
7040 DATA TRIANGLE, 50, 191, 180, 126, 75, 191, 0, 3

7050 REM CASTLE DATA
7060 DATA ORIGIN, 63, 30, 0, 0, 0, 0, 0, 0
7070 DATA BLOCK, 28, 60, 118, 120, 0, 0, 0, 1, 2
7080 DATA BLOCK, 14, 40, 41, 80, 0, 0, 0, 1, 2
7090 DATA BLOCK, 105, 40, 132, 80, 0, 0, 0, 1, 2

7090 REM OCTOBER DATA
7093 DATA ORIGIN, 20, 0, 0, 0, 5, 0, 0, 0
7097 DATA BLOCK, 20, 0, 20, 15, 0, 0, 0, 1, 2
7099 DATA BLOCK, 20, 0, 40, 15, 0, 0, 0, 1, 2

Line 6010 **READs** a string, **R$$**, then the numbers that follow it. Line 6020 looks for the word "ORIGIN" and sets the coordinates of the origin as **OX** and **OY**. Line 6030 through 6040 add the coordinates of the origin into any coordinates given after the other new instructions. Line 6050 looks for the word "BLOCK" and goes to the block-drawing subroutine if it finds it. And line 6060 looks for "TRIANGLE". If line 6070 finds "STOP" the computer returns to the control routine.

In the block routine, the sum \( T = 1 - T \) is used to change the color each time the computer goes round a loop, so each line is drawn in a different color. **HCOLOR** – the high-resolution color – is set to \( C(T) \). And **HPLOT** draws a line across the screen at the height given by **Y**. So the block routine draws a block a line at a time, alternating the color of the lines to give stripes.

The triangle routine works in much the same way, but each line is a different length. This length is worked out by lines 6210 and 6220. Each line across the triangle gets bigger as you go down it. How much bigger each line is, is worked out by dividing the distance between the points of the triangle by the height. This length is then multiplied up as the lines are drawn down the triangle. The color of the lines is alternated using the sum \( T = 1 - T \) again.
The roofs are put on the turrets by drawing two large triangles. These are not striped though. But the triangle routine draws lines alternately in the two colors specified. So the two colors are simply set to the same value. Both are color 5. The origin for this section was fixed in the last bunch of DATA, the DATA for the castle.

The battlements of the castle are drawn in by extending the roof color into the body of the castle. This is done by using the BLOCK command to draw five small squares in the roof color over the edge of the block that has already been drawn to make the castle’s walls. The block is not striped so the both colors are the same again – both color 5, the same as the roof. You will notice again that the fifth and sixth numbers that follow the BLOCK command are zero.

The castle’s windows and doors are drawn in using the BLOCK command too. This DATA is still working from the origin given in the castle DATA. The roofs on the turrets, the battlements and the windows and doors are all drawn over the original castle, so it makes sense to use the same origin. This time the blocks are striped. The windows and doors are drawn in alternate lines of color 0, black, and color 3, white. The STOP command that switches off the graphics routine has only zeros following it. It may seem unnecessary to have all these zeros, but the routine that READs the DATA is expecting eight numbers.
This routine draws twenty trees – one in each pass of the FOR...NEXT loop between lines 6510 and 6530. Line 6520 rolls the computer's dice function, RND, to position the trees in random places. The DRAW 1 in line 6530 draws the first shape from the shape table on the screen. The first shape is, of course, the tree. The position is given by the expressions that follow AT in line 6530.

There are nine shapes in the shape table. This is set by the 9 in the DATA in line 4010 which is read in at the start of the table. The code here is simple to follow. P means print, L means left, R means right, U means up, D means down and M means move. Directions following a P fill in the next point in that direction with color. But directions following an M move the point in that direction without coloring it. A number before a direction tells the computer to move or print that many points in that direction. If you draw out these movements on a piece of graph paper you'll see the shape of a tree emerging.
The background routine does not draw up much of the background, but it does set up areas of the screen and color memories so that the routines that do the drawing later will know which color to use. Line 1010 POKEs the color memory along the horizon with gray. And lines 1020 through 1040 POKE the color memory of the bottom of the screen with green for the grass and the screen memory with cyan and brown, ready to draw the trees. Line 1050 uses the machine code program to draw a gray line along the horizon. And line 1060 uses it again to color in the grass.

The subroutine that draws the main body of the castle uses our box drawing routine at line 13000. So before that routine is called line 2010 specifies the sides of the box and gives it a color – 2, which is red. NS – the no-stripe variable – is not set – so the red-brick of castle is striped with orange mortar by the box routine. This doesn’t look too realistic so we add random red bricks with lines 2020 through 2040, to break up the pattern and give a mottled effect.
To draw the turrets, the box-drawing routine at line 13000 is called again. But this time it is called twice, once for each turret. Again the top, bottom and sides of the box must be specified. But the second time the box-drawing routine is called, only new sides for the turret have to be specified. The top and bottom are at the same level. The boxes are filled in with color 2, red, again. And again NS is not set to 1, so the orange mortar stripes are added automatically. Lines 3040 and 3050 gives the random speckling of red bricks. To put the turret tops on, the screen memory in that area is POKEd with 11, which is gray, and lines 4040 and 4080 draw up a series of gray lines which give the pointed turret roofs.

The main roof is drawn in a similar way. The number corresponding to the color gray, 11, is POKEd into the screen memory in the area occupied by the roof, and SYS MC calls the machine code routine repeatedly. This draws up a series of gray lines down the screen which form the roof. The main roof is square, though, and simpler to draw. So instead of the two loops which give lines which get longer as the routine moves on the screen – giving the turrets' pointed roofs – only one loop is needed. Notice how this way of drawing a block differs from using the DRAW BOX routine. Both approaches can be used later when you are writing your own programs.
To draw in the battlements, the roof color is simply extended down into the wall in two square areas. So gray is POKEd into that area of the screen memory and SYS MC is called again twice. The 2 at the end of the string of figures after each SYS MC command tells the machine code program to get its color from the screen memory, rather than the color memory or anywhere else.

The windows have already been defined as sprites in the initialization routine. So to draw in the windows we only have to turn the window sprites on with the instruction in line 7010. The door, though, is drawn up using the box drawing routine at line 13000. The sides of the door are set in line 7020 and line 7030 sets the color to 9, brown. Since we don't want orange stripes on the door, the value of NS is set back to 1, then the box-drawing routine is called.

When sprites are defined, each is given a pointer which tells the sprite where to find its DATA in the computer's memory. This DATA has to be put into memory separately. Here is the DATA for the window sprites. It is READ and POKEd into memory in the right place by the initialization program. Every number in this DATA defines a small part of each window when it appears on the screen. So be careful that you type it in absolutely right or the program will not work properly. It is very easy to make a mistake when you are keying in a long string of numbers. Be sure to double check it.
Now that the scene has been set and the castle is in place we need to add the spooky forest. One point about the forest is that it is never the same twice when you RUN the program. The ten trees are drawn up in random positions which are fixed in line 8020. But we don’t want the trees appearing in the sky or in the castle. So they are confined to the bottom and roughly the middle area of the grass by line 8030. If a **TX** or a **TY** position come up that are off limits, line 8030 sends the computer back to line 8020 to roll the **RND** dice again.

The area that each of the trees occupies is stepped across a point at a time by I and J. I moves up and down the tree, while J moves across it from side to side. I and J, of course, start working from the screen positions **TX** and **TY** worked out before. At each new position and a new element of the tree array **TR** is read into **MO**. If its not equal to 2, the computer skips the draw instruction and moves on to the **NEXT** point. But if it is 2, **SYS MC** draws in the tree trunk in brown, which is the color that has been **POKE**d into the screen memory of this area of the screen.
Testing your program

You have now completed the part of the program that draws up the scenery, the castle and the spooky forest, so it will be as well to test them and check that they are working properly before the after-dark action starts. To do that, you'll have to add the following test lines though. Otherwise the main routine will call subroutines that have not yet been typed in, and you will receive an error message.

APPLE

TYPE 125 GOTO 125 AND RUN THE PROGRAM. PRESS THE CONTROL AND RESET KEYS WHEN YOU ARE SATISFIED THAT THE PROGRAM IS WORKING CORRECTLY. DELETE LINE 125 BEFORE CONTINUING WITH THE NEXT SECTION.

COMMODORE 64

TYPE 45 GOTO 45 AND RUN THE PROGRAM. PRESS THE RUN STOP AND RESTORE KEYS WHEN YOU ARE SATISFIED THAT THE PROGRAM IS WORKING CORRECTLY. DELETE LINE 45 BEFORE CONTINUING WITH THE NEXT SECTION.
MIDNIGHT

This section of the program turns day into night. The bat, the witch and the burning flames appear. They are animated in different ways, depending on which computer you are using. But once you have learnt these techniques, you can use them in your own programs to create a variety of effects.
The machine code routine that switches day into night is contained in the DATA in lines 5300 through 5360. Lines 5220 through 5240 READ the DATA and POKE it into memory. When you type in machine code you must get the figures exactly right. The routine adds up all the figures and checks them against a check sum – if you make a mistake, the message in line 5250 will be displayed.

When night falls the routine at line 5000 is called. It immediately calls the machine code which copies the daytime scene onto graphics screen 2.
POKING GPAGE with 64 in line 5020 allows you to draw on the second screen. Remember that the first graphics screen is still the one that is appearing on the TV screen. The first thing to be drawn on screen two is the moon. The DATA for the shape of the moon is read in from lines 4210 through 4240. Like the rest of the shape DATA, it ends with an E. When the shape-table routine READs an E, it knows that it has reached the end of the shape. This is shape two, so the DRAW 2 instructions in line 5030 draws the moon up. It is drawn in color 3, given by the HCOLOR command and the moon is drawn at coordinates 15, 7. Next there's a 2,000 time period pause. Then, by PEEKing P2GE, the second graphics screen appears on the TV. Line 5080 POKEs GPAGE with 32 which allows you to draw on the first screen again. The computer then goes off to the subroutine at line 5100 to draw in the lightning. When it has done that POKEing GPAGE with 64 lets you draw on screen 2 again.

The lightning is drawn on graphics screen 1 by lines 5100 through to 5150 while the other screen is displayed on the TV. The lightning is drawn in color 7, white, and is made of a long narrow triangle which thins toward the bottom. Every 23 lines down the screen this triangle is given a quick shift to the left to give it the proper jagged look of lightning.

```
5100 HCOLOR= 7: DX = 0: C = 0: REM DRAW LIGHTNING
5110 FOR Y = 1 TO 100: C = C + 1
5120 LET D1 = Y * .96: D2 = Y * 1.25
5130 IF C = 23 THEN DX = DX + 29 + D1 - D2: C = 0
5140 HPLT 250 - D1 + DX, Y TO 279 - D2 + DX, Y
5150 NEXT : RETURN
```

To flash the lightning, the screen is changed to screen 1 by PEEKing P1GE. This displays the day screen with the lightning drawn on it on the TV.

```
2400 REM FLASH LIGHTNING
2410 LET X = PEEK (P1GE)
2420 FOR X = 1 TO 60
2430 NEXT : X = PEEK (P2GE): RETURN
```
To make the bat appear to flap its wings there are two bat shapes in the shape table: one with the bat's wings up and one with the bat's wings down. The shapes are switched by the \( BN = 1 - BN \) sum. The XDRAW command is an "exclusive" draw. This means that drawing a new bat shape will blank out the old one. The path that the bat takes when it flies is a spiral, fixed by the instructions in lines 2260 through 2280. The subroutine that removes the bat calls the same drawing routine but its path is fixed by lines 2310 through 2330. When this routine is called, these lines make the bat fly off the top of the screen.

The DATA for the two bat shapes is given in lines 4310 and 4410. Note that there are two "Es" in the DATA, so that the computer knows that there are two shapes. Draw the two bats on graph paper, using the DATA, in the way explained on page 30. Then along comes the witch.
In the witch routine, the XDRAW instruction is used to draw the witch over the background, but the background is returned intact when she has passed by. When the witch moves, two XDRAWS are used one after the other. The first draws up the new witch, the second blanks out what was left of the old one. Line 2170 detects when the witch has reached the edge of the screen, skips the new drawing instruction and goes straight to the blanking out line.

There are three flame shapes and these are called up randomly, in random colors, in random positions that gradually cover the castle. The castle is made to crumble simply by printing a block of black over it using the subroutine at line 6100.
The skull in the sky is enlarged by setting the SCALE to 2. And it is made to shimmer eerily in the sky by altering the position it is printed in slightly. The X position is shifted randomly by 3 high-resolution screen positions and the Y position is shifted randomly by 4. Line 2730 rolls the RND dice for these shifts. The skull, which is shape 9, is then drawn by XDRAW. This also blanks out the last skull that appeared on the screen. The skull is printed up — and blanked out — 2,000 times before the computer RETURNS from this subroutine and the whole program ENDS. But, until then, the effect is very eerie indeed.

All that is needed now to make the whole program work is shape-table DATA for the skull itself. This is typed in the code which consists of print and move commands and directional instructions as before. See if you can trace out the shape on a piece of graph paper. This will give you a clear outline of the skull shape, without the shimmer that is given by shifting its position when it is printed successively on the screen. And like the rest of the shape DATA, the skull's DATA ends with an E that tells the part of the initialization routine which is filling the shape table that this is the end of the last shape in this program. The machine code routine on pages 8 and 9 can be used in other programs that you write. It makes it much easier to create character shapes on the Apple IIe.
Making darkness fall is easy. The **POKE** in line 9010 changes the background color — which was the sky's light blue — to black. But then the moon has to appear. Lines 9020 through 9040 **POKE** an area of the screen memory with 7, the code for the color yellow. Lines 9050 through 9090 draw in the shape of the moon there using the machine code call **SYS MC**. The **DATA** for the moon is read out of the array **MO**.

That same **DATA** is **READ** into the array **MO** by the initialization routine. But at that time we did not give it any **DATA**. Here it is now. You'll notice again that the shape of the moon is given by the **2s**. In the night routine, line 9070 skips the machine code call if it hits a 4 and nothing is printed on the screen. The points making up the moon are only printed on the screen when a 2 is found. Each 2 is also used in the **SYS MC** in line 9080 to tell the machine code to take its color from the screen memory.
The subroutine that moves the bat starts with a 30 time period delay. The little sum $T = 1 - T$ in line 10060 flips the number of the DATA block POKEd into the bat's sprite pointer between 35 and 36. Lines 10070 and 10080 change the bat's direction when it has reached its limits. Line 10090 sends the bat off screen when the witch comes by and the rest of the routine fixes the bat's sprite position.

The bat needs two lots of DATA—one to draw it with its wings up and one with its wings down. These are read into blocks 35 and 36 by the initialization routine. So each time the bat routine is called, the other picture is printed up. Alternating between the two images in this way makes the bat appear to flap its wings as it flies.

The witch flies on her broomstick though. Since she has no wings, she only needs one block of DATA for the one image.
Lines 11010 and 11020 give the coordinates of the flames. Line 11040 sets the sprites to high-resolution mode, and line 11080 sets their color to 2, or red. Once the flame sprites are turned on, the burn time, BU, is set to 200. Line 11080 flips the sprite's high-resolution color between yellow and black to make them flicker. When BU reaches 50, and again at 100, the program branches to subroutines which make the sprites expand. But when that happens the sprites have to be repositioned. Line 11030 finally burns the place down.

All the flames are the same shape, so they only need one lot of DATA. This is POKEd into memory by the initialization program. They don't all look the same on the screen though, because of the use of constantly changing colors.
All the lightning subroutine does is to make the lightning flash. The lightning appears when 53281 is POKEd with 1. After a pause caused by the machine counting to 50, the lightning disappears when it is changed back to black again.

To create the eerie skull, lines 12010 and 12020 POKE random color values into the screen memory in the area that is going to be occupied by the skull. The DATA for the skull is read out of the array SK. If the element of the array for that particular area is 4, the drawing lines are skipped. Otherwise, random numbers between 1 and 3 direct the machine code to pick its colors from different areas of memory. The 0s in the skull data draw out the shape of the skull and the 4s are left as the background.
Program listings
What follows is the complete program for both computers. List the program and check it against the full listing here. Even the smallest mistake can cause problems. Take special care to check the DATA lines. A comma out of place here can stop the whole program working. When you have a full working program, SAVE it on tape or disk, using the commands in your user's manual.
Glossary

**DATA**  A list of information that is required by a program. **DATA** can consist of words or numbers, or both together. A program is sent to the **DATA** with the instruction **READ**.

**FOR.....NEXT**  This is a sequence of commands that is used to make the computer repeat an operation a certain number of times. For example, the loop **FOR X = 1 TO 5:PRINT 2*X:NEXT X** would cause the computer to print the two times table up to five.

**GOTO**  This statement tells the computer to go to the specified line, missing out any lines in-between. It is often used with **IF.....THEN** (see below) and is only operated if certain conditions are true. Be careful when using **GOTOs**, as it's easy to have the program jumping backward and forward so much that it is impossible to read.

**HGR**  This sets the high resolution graphics mode on the Apple.

**HPLLOT**  This places a set of x, y coordinates on the Apple screen. If **HPLLOT** is followed by **TO**, it draws a line from the last point plotted to the coordinates indicated. This works both horizontally and vertically.

**IF.....THEN**  This is used as a way of telling the computer to do something only when certain conditions are true. This instruction often looks something like this: **IF score = LE THEN WO = 1**.

**INT**  **INT** is short for integer, and instructs the computer to make a whole number of a figure with decimal places in it. It is often used in conjunction with the **RND** command which instructs the computer to generate a random number (see below).

**LEFTS**  This instruction is used to copy part of a string, starting at the left-hand end. It is followed in brackets by the string name and the number of characters to be copied.

**LET**  This is one way of giving the computer information. In some programs there may be statements such as: **X = 10**. This simply means that the number ten is stored under the label X. It is often clearer to write: **LET X = 10**

The **LET** statement also gives rise to something that at first sight seems illogical, if not impossible. In many programs you will see things like: **LET X = X + 1**

Of course, in mathematical terms **X** can't equal **X + 1**. All this type of statement means is "increase the value of whatever is stored in **X** by one."
LIST This makes the computer display whatever program it has in its memory. You can LIST single lines, or parts of a program by following the LIST with appropriate line numbers.

MIDS This is used to copy the middle part of a string. It is followed in brackets by the string name, the start position, and the number of characters to be copied.

PEEK This instruction looks at a particular memory location. It is often associated with POKE.

PIXEL This represents a point on the grid in graphics mode. The number of pixels per screen is determined by the quality of the graphics, e.g. high or low resolution mode.

POKE This stores numeric information in the computer’s memory. It is often used for sound and places a binary number in a particular location.

PRINT This tells the computer to display something on the screen.

RIGHTS Similar to LEFTS, but copies the right-hand end of a string.

RND This instruction makes the computer generate a random number. The precise instruction varies between different models of computer.

shape tables These are used to define the shape of a graphic on the screen on the Apple IIe. They are stored in tables, so that they can be called up easily when required.

sprite A sprite is a user-defined character on the Commodore 64 computer. It is a small area of the graphics screen which can be moved around and switched on and off with ease. A maximum of eight different sprites can be defined.

STEP The STEP statement is always used following a FOR.... statement. It indicates the amount that the variable should be changed for each operation. For example: FOR X=0 TO 20
STEP 5: PRINT X: NEXT X would mean that X would rise in steps of five, so that the computer would print 0, 5, 10, 15, 20.

XDRAW This stands for "eXclusive DRAW." It is an instruction used on the Apple IIe. It combines what is being drawn "exclusively" with what is behind it. This means that it can be drawn over a background, and the background will return when the drawing is blanked out. It also means that when the same shape is "eXclusively DRAWn" in the same place twice, it disappears.
Index

A
angle 15
animation 14, 16
array 17, 38, 38

B
BASIC 8, 20
BLOCK 20, 22, 25
box-drawing 26
byte 15

C
checksum 30
clock 14
color 11, 15, 17, 20, 21, 22,
24, 35, 37
ccontrol program 14, 16
"crash" 30

D
direction 16, 23, 36

G
graphics 15, 20, 22

H
high resolution graphics 10,
15, 17, 24, 37

I
initialization 14, 15, 17, 26,
34, 35, 36, 37

L
loop 10, 11, 21, 23, 25

M
machine code 8, 10, 11, 14,
15, 17, 24, 25, 30, 38
memory 8, 11, 15, 16, 24, 26,
30
memory location 8, 16, 17

P
pause 14, 31, 38
position 34
program listings 39-41

R
RND 14, 16, 23, 27, 34

S
scale 34
screen memory 25, 26, 27,
35, 38
shape table 8, 9, 15, 23, 31,
34
sprite 16, 17, 26, 36, 37
starter program 11
storage 39
storyboard 12, 14
string 9, 21
SYS MC 10, 11, 25, 26, 35

T
testing 28
TRIANGLE 20, 21
Design
Cooper · West

Editor
Nigel Cawthorne

Program Editors
Robin Betts
Henry Waldock

Illustrator
Gerard Brown
WRITE YOUR OWN PROGRAM

This new series introduces the art of programming your computer. Each book shows how to structure a program into routines, and at the same time explains and analyzes what exactly you are asking the computer to do and why.

And there's fun too! Each book contains one or more exciting and original computer games. These can be adapted and extended to do bigger and better things.

The books are specific to the Commodore 64 and Apple IIe.

Titles in this series:
- Beginning Basic
- Space Journey, Graphics, Hangman
- Moving Graphics, Alien Invaders
- Creating a Database, Adventure Game

A Gloucester Press Library Edition
ISBN 0-531-17013-6
283