SCREEN PLAY

In the last two instalments of this programming project, we designed screen displays for two special locations in the Digitaya adventure, for the Spectrum and BBC Micro. Here, we look at designing and programming these displays on the Commodore 64.

The designs of the adventure screens for the BBC Micro and the Spectrum were similar: both computers have high-resolution graphics and easy PRINT formatting in BASIC. The differences lay mainly in the screen dimensions and BASIC dialect words that handle high-resolution plotting. Designing similar screens for the Commodore 64, however, requires a radically different approach. The 64 does have high-resolution facilities, but effectively these are available only to the machine code programmer, as no BASIC commands are provided to handle high-resolution, and performing the relevant PEEKs and POKEs in BASIC to produce high-resolution displays is so slow as to make it unusable in this application. Instead, we must adapt the relatively easy-to-use facilities offered by the Commodore 64.

Graphics characters can be used to build up large letters or other displays by combining different characters. Sprites are a convenient method of introducing high-resolution shapes to the normal Commodore screen. PET characters can be positioned on the screen using either a series of PRINT statements or by POKEing the relevant character code to the screen. We shall demonstrate both methods.

JOYSTICK PORT SCREEN

Lines 8020 to 8170 of the Joystick Screen listing are concerned with reading in the data for the two sprites used in this routine. The first, sprite 0, is defined by the first 63 numbers in the group of DATA statements between lines 8450 and 8497, and represents the joystick port (shown in the diagram). Sprite data is usually positioned high in the BASIC program area, but with a large BASIC program this data stands a good chance of being overwritten. An alternative location is the cassette buffer area between locations 832 and 1022, where the data for up to three sprites can be held. This routine stores its sprite data in this safe area.

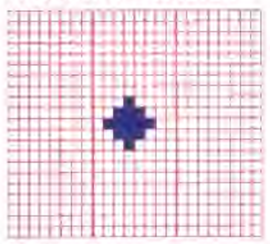
Sprite 0 is stretched to twice its original width to produce the final displayed shape by setting bit 0 of the horizontal expansion register in line 8170. Notice that all the registers controlling the attributes of sprites, such as colour, position and expansion, are related to the start address for the video control chip (VIC). Remembering that the horizontal expansion register has the address VIC+29 is much easier than memorising its actual memory location (53277). Some sprite attributes require a whole register for each sprite — for example, the X and Y co-ordinate registers — but where the eight bits independently control a function for the eight available sprites, attributes can be controlled by setting and unsetting the appropriate bit in a single register. Sprite 1 is defined by the remaining 13 numbers in the DATA statements and represents an object to be fired out of the joystick port.

Since the 'solid' part of sprite 1 is small (it represents a projectile), it is quicker and easier to enter the 63 bytes of data that define it in two stages. Firstly, POKE 63 zeros into the defining area, and then READ and POKE in the few numbers that define the shape. In this way, we can dispense with the large number of zeros that would otherwise be required as data.

Lines 8190 to 8220 are concerned with the construction of strings constituted from a series of PET graphics characters. LES forms a horizontal line the width of the screen by combining 40 of the special PET characters on the front right of the C key. DW$ is a series of cursor-down characters. LS$ and RS$ are groups of left and right diagonals (on the front right of the N and M keys) that are used to form a herringbone pattern in the foreground. This pattern introduces depth and perspective to the scene.

Shoot-Em-Up!
The joystick port's 'shooting' action is accomplished on the 64 through the use of a sprite defined as a projectile. The outlines are set by POKEing zeros into the defining area, then the solid areas are READ from DATA statements and poked into the correct sprite locations

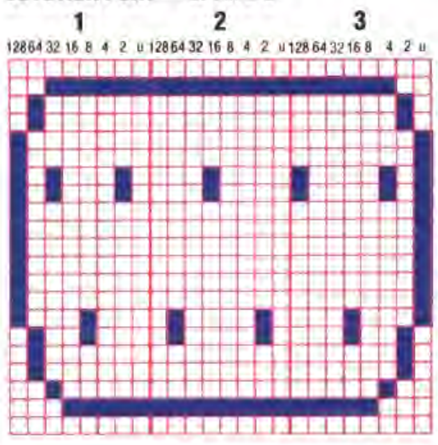
PROJECTILE — SPRITE 1



0	16	0
0	56	0
0	124	0
0	56	0
0	16	0

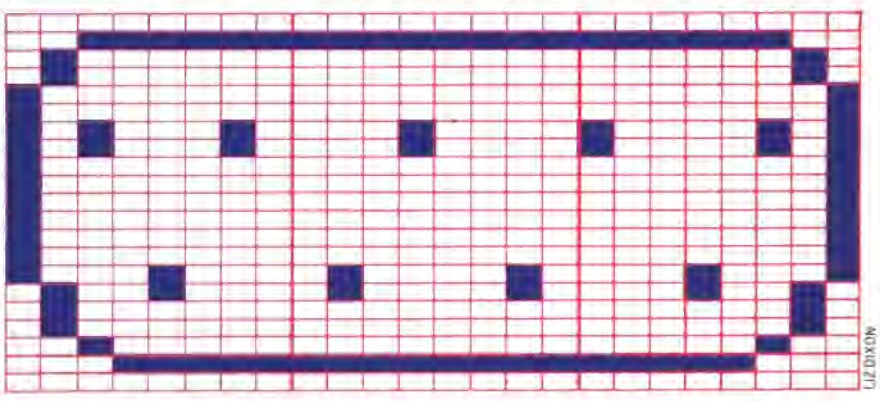
Sprite Stretching
The values listed for the joystick port sprite are READ from DATA statements and poked into the appropriate sprite locations. As designed, the joystick port is too compressed, but it can be stretched horizontally (as shown) by changing the value of the horizontal expansion register

JOYSTICK PORT — SPRITE 0



1
2
3
128 64 32 16 8 4 2 u 128 64 32 16 8 4 2 u 128 64 32 16 8 4 2 u

1	2	3
0	0	0
63	255	252
64	0	2
64	0	2
128	0	1
128	0	1
162	16	133
162	16	133
128	0	1
128	0	1
128	0	1
128	0	1
128	0	1
128	0	1
136	66	17
72	66	18
64	0	2
64	0	2
32	0	4
31	255	248
0	0	0



LIZ DIXON