•Beginners Mac Assembly•


Chapter 6
Sprites+Games

In chapter five we got as far as directly addressing the screen for Macs with Ram Based Video (RBV). In chapter 6 we continue with the graphics theme and introduce sprites. We also have to assume a basic knowledge of assembler by now - although we've included a sprite package with Fantasm, the concepts and the driving code are not so trivial as a simple program to scroll the screen. You will need Fantasm V2 to use the sprite routines and assemble the example programs, oh and at least an 020 processor.

In chapter 6 we adopt an informal, chatty approach. In this chapter we have to assume you can at least "read" assembly programs, and would also suggest you re-read over the other chapters on the Macintosh screen layout for ram based video machines.

All the routines discussed in this chapter expect 256 colour mode - they don't check which mode the Mac is in!

A word about copyright.

Lightsoft sprites are distributed as part of the Fantasm V2.XX package.

Sprites - eh?

Games programmers think of a sprite as the mainstay of they're games.

A sprite is a graphic object, that can be easily controlled by the programmer. Sprites are something the programmer doesn't have to know how it works to use them - they just tell the sprites where to go and forget them.

Sprites come in two varieties - hardware (Commodore 64, Amiga, SNES, Megadrive etc), and software sprites.

The Mac does not have hardware sprites, they are simulated with software. The best way of getting an idea of sprites is to run the demo game "lander.app" in the lander folder of Fantasm V2.00 - NOTE it expects your Mac to be in 256 colour mode - it doesn't check.

This is a simple copy of the old lunar lander games whereby you have to land the spaceship safely. In this partially completed demo you cannot safely land the craft, as the code to do that isn't there - however it does get across all that we want to cover in this chapter. You control the ship with the arrow keys to fire the thrusters - "Q" will end the demo.

When the ship crashes the game simply restarts.

Sprite priorities and transparency.

An important aspect of sprites is that they have priorities - they are numbered from 1 upwards, where 1 is the lowest priority. Why? Well what happens if two sprites cross on the screen - do they just merge into a multicolour mess? Well, no, the sprites have priorities so that the highest numbered sprite moves in front of the lower numbered one. You can see this in the demo when the vehicle passes the table that is moving up and down.

This highlights the other nicety of sprites - transparency. Lightsoft sprites define colour zero as being transparent - that is wherever a pixel is colour zero (the first colour in the palette), whatever is behind shows through the sprite.

How sprites work.

Assume we have a "sprite" we want to move across the screen. The simplest way of doing it is to print the sprite on the screen, wait a while, then remove the old sprite and print the new one in the next position. In reality it is a little more complex than that because of the background. If the background is all black, then to remove a sprite, you just clear the VRAM. However, if we have a background, then we have to print the old background in place of the sprite, so as the sprite moves, the background is not destroyed.. To be able to do this the routines keep a copy of the background in memory, so it can be printed over the old sprite to delete it. Then the sprite is printed at its new location, after the routine has copied the background (so the sprite can be subsequently deleted on the next frame).

Take the spaceship as an example: it starts at the top of the screen, and assuming no thrusters are fired, it drifts slowly down under the influence of "gravity".

Initially, the program has to put the sprite at the top of the screen. The program assigns a shape to a sprite - in this case a ship shape is

assigned to sprite 1.
Next the program defines the sprites' x and y coordinates on the screen - for example 50 x and 40 y, where the top left of the screen is 0,0.

That's all the information the system needs to be able to draw this sprite. Now the program can call the sprite routine to draw the ship at these coordinates, then increment the y coordinate, and call the sprite routine again, and again and again - the result is the ship moves down the screen.

The sprite routines carry out the following actions:

1. Wait for the Vertical BLanking interrupt.

2. If this is the first time this sprite has been used, goto step 4.

3. Put the background for the old position of this sprite back onto the screen.

4. Copy the background at 50x, 40y as a 32 by 32 pixel rectangle into a buffer.

5. Print the shape assigned to sprite 1 at 50x, 40y

6. Any more sprites to do? If yes, goto to step 1, if no, end.

Step 1 - wait for the VBL - the VBL is an interrupt generated by the Mac every time the electron beam in the monitor gets to the bottom of the screen. Its important to wait for the VBL because we hope to be able to print the old backgrounds, copy the new backgrounds and print the sprites in their new places before the monitor starts drawing the next frame - if we don’t do this then some or all sprites will flicker as the Mac prints the old backgrounds and updates the sprite positions. Hence sprite code has to be extremely fast!