AUNTIE JOHN

How to be a smoothie

AJ puts on his best suit and tries to chat up the firmware. Will he succeed?

Hi there! Well, I made it back from my motorbiking excursion through Wales without falling off, so it's only fair that I keep my side of the bargain we made last month and exlore how to be a smoothie when it comes to graphics.

Cast your mind back to last month, and you will probably remember that we managed to achieve a small bouncing star on the screen. It might not have seemed much, but it was our first piece of moving computer animation. It was just a bit jerky, that's all. The reason for the jerky-ness was simple: we were moving it in steps that were just too large.

Here's a nice mental image to help you get a feel for what's happening. Imagine you have a video camera that can take a series of frames, one at a time. The camera is trained on your kitchen table, on which there is a box of matches. (Remember kids, always make sure you have permission before you play with matches. Borrowing the video camera is probably ok, but accidentally setting fire to the house definitely isn't).

Now let's start making a film. The first frame is taken, then the box of matches is moved 10cm to the left. The next frame is taken and so forth, until the matches have fallen off the table. Now rewind and watch the film. Jerky, isn't it?

If we were to remake the film, but moved the box of matches 1cm each time, the film would be a lot smoother, woudn't it? And this is how we solve the problem of the jerky graphics.

In last month's example, the star was positioned using the normal text cursor positioning firmware. This meant that we were working in a grid of about 40 across by 25 down. Chunky!

By using the graphics firmware instead, we should be able to position the ball to within a pixel's width. By doing so, we'll have created the smoothest movement possible.

Try typing the listing into your assembler and running it. The program is a direct conversion from last month's so it should look familiar. The biggest change is the use of register pairs to hold the X and Y co-ordinates and directions instead of single registers. We have to do this because the firmware calls GraWrChar and MoveAbs require us to do so. This makes sense, because they may need numbers greater than 255 – the largest number a single register can hold.

The use of 16 bit numbers makes some of our sums that little bit more tricky. Take the code in the routine 'boing' as an example. Now, I was in such a hurry to take my hols last month that I didn't get time to properly explain how or what the routine does. So I'll do it now if you don't mind.

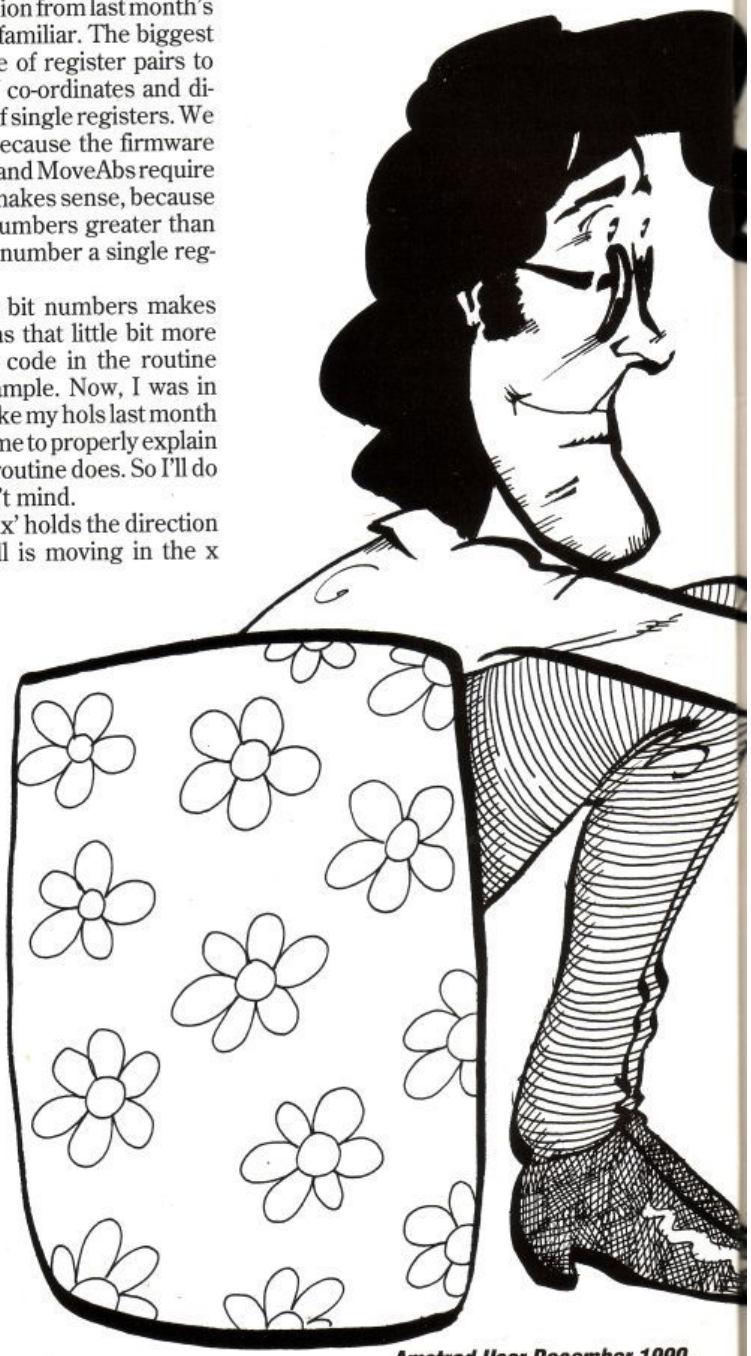
The variable 'dx' holds the direction in which the ball is moving in the x direction. If it is one, the ball is moving to the right. If it is minus one, it is moving to the left. Somehow our program needs to change the -1 to a 1 and vice versa at the right moments.

Finding 'the right moments' is easy: They occur just at the edge of the screen. Using a text screen (like we did last month) the edges are a co-ordinates 1 and 4. In a graphics screen they are much higher, because of the improved resolution.

Changing the -1 to a 1 and so forth could have been done with a piece of code which in BASIC goes something like:

IF dx=-1 then dx=1 else dx=-1

In machine code this would mean



lots of checking and jumping around. Instead I used a cunning method that relies on the way numbers are stored internally. As we discussed many moons ago, the values can be treated as 8 bits of binary data: or a row of 8 ones or zeros. You may wonder how the computer can store a negative number with only space for 1's or 0's. This is a good question.

Basically it cheats, and decides on a set of rules so it knows whether or not the number is positive or negative.

The rule works like this:

'To turn a positive number into a negative number, turn all the 1's to 0's and all the 0's to 1's. Then add 1 to the new number.

The reason why this works is all to do with 2's. Complement arithmetic. I can't be bothered to explain it all here, so if you are interested ask your maths or computer science teacher what it's all about. He or she will only be too happy to explain, as they will think you are taking a real interest in your studies.

So that's what the code in 'boing' does. There is a slight complication in that we are dealing with 16 bit numbers in this month's listing, so we have to change ALL the bits from 1's to 0's and add 1 to the whole lot. In other words we have to change both the H register and the L register individually.

When watching the program move your little star around the screen, a thought may strike you: it's a bit on the slow side. Although we have solved the jerk problem, we've come across another one. How can we speed it up?

Here's a little trick that might make a difference. Change the program to print a full-stop instead of a star, by changing the number 42 in the PrintBall routine to 46. Now remove the line 'Call EraseBall' from the listing and re-assemble it.

Now the shape of the full-stop with all the blank spaces around it will automatically erase itself as it moves along. If you doubt, try changing the full-stop back into a star and watching the trails it leaves behind.

The listing

```
org & 800

Mode equ &BCOE
GraWrChar &BBFC
MoveAbs & BBCCO
TestKey equ &BB1E
Frame equ %BD19

      ld a,1
      call Mode

loop

      call PrintBall
      call Frame

      call EraseBall
      call MoveBall

      ld a,47
      call TestKey
      jr z, loop

MoveBall
      ld hl, (ballx)
      ld de,(dx)
      add hl, de
      ld (ballx), hl

      ld a,1
      cp 1
      jr z, boing
      cp 200
      ret nz

boing

      ld hl, (dx)
      ld, a,h
      xor 255
      ld h,a
      ld l,a
      xor 255
      add 1
      ld l,a
      ld (dx), hl
      ret

PrintBall
      ld de,(ballx)
      le hl, (bally)
      call MoveAbs
      ld a,42
      call GrWrChar
      ret

EraseBall
      ld de, (ballx)
      ld hl, (bally)
      call MoveAbs
      ld a,32
      call GrWrChar
      ret

ballx dw 1
bally dw 100
dx dw 1
dy dw 1
```