Play time with auntie

Get a grip with auntie John's step by step guide to game making.

How are you doing, machine code fans? Have you been having fun with the Z80? Good, glad to hear it! Me too!

One of the reasons you may have for learning machine code is the deep desire to write your own action-packed arcade games on the CPC. There is absolutely nothing wrong with this – it is a very commendable way of using your new found skills. Nothing tests a programmer's skill more than writing a challenging, enjoyable and successful game. Unless you count making lots of money out of writing such a game.

If you are thinking of writing the next chart buster, you must remember to spend a great deal of time with the graphics. The graphics are the most important part of a game, because if it doesn't look good no one will want to go near it, never mind have a go. The gameplay may be superb, the sound effects may be sensational, but if the player's spaceship is a capital letter A and the aliens lower case letter p's, then the game is a loser.

(Unless of course you happen to be programming a ZX81, where the lack of graphics means you do indeed have to make creative use of the character set).

The new CPC range just announced (don't they look lovely!) pay even more attention to graphics by redesigning the screen display hardware to provide more colours and "sprites". Sprites are graphical shapes that the hardware of the computer handles itself – wonderful stuff that makes games so much easier.

But how does this all fit into out vast machine code plan? Does the firmware have a routine like:
Address: & BEDO
Input: Star System, Nastiness quotient
Output: Horribly beweaponed alien fight craft
Output: Causes a fleet of flying saucers to swoop down the screen, dropping bombs and generally making a nuisance of themselves (corrupts the BC registers)

Unfortunately, the answer is no. There is no such firmware call, and so we will have to construct our own code from what actually exists.

As you probably know by now, the number of characters you can fit on one screen line depends on the screen Mode you have chosen. for example, in Mode 1 you can have 40 characters across the screen, whereas in Mode 2 you can have up to 80.

Now imagine that we have a Mode 1 screen set-up, all ready to go. In our hypothetical game, a small ball is going to bounce around the screen. If the ball was a standard character, like a letter A for example, then we can move the ball anywhere in a grid of 40 by 25 places.

This kind of movement can lead to the graphics appearing a bit juddery, because the amount the ball moves is equivalent to its own size.

If you don't believe me, type in listing number one into your assembler and run it. Hardly state of the art, is it?

It works by drawing a ball on the screen (well, ok, it's a star) and then erasing it by drawing a space over the top of it. By performing a call to Frame (see last month's issue to see what this does) we can get the illusion of movement.

The program needs four co-ordinates: two to hold the ball's X, Y position on the screen, and two to hold the direction values. For example, when the program starts, the X co-ordinate is 1, and the X direction is also 1. The program adds the direction onto the co-ordinate, and then checks to see if the ball has reached the edge of the screen.

If it has, the direction is made the negative of whatever it was: in other words, if it was 1 it becomes -1, and if it was -1 it becomes 1. This is achieved by some cunning binary maths – don't worry if you can't understand it, we'll come back to it soon.

You'll notice that I am only changing the X co-ordinates in this program. It will be a good exercise for you to update the program to include the Y co-

ordinates. If you do, the star will bounce around the screen in a very Pong-like way. Remember that the height of the screen is not the same as the width, and so you'll need to change the checking value.

The solution is to move the ball in much finer steps: if possible we would like to move it pixel by pixel. Why a pixel's width? Well this is the smallest possible bit that can be displayed on-screen. We simply couldn't move the ball any more smoothly.

And guess what? Yup, we'll look at a program to do it next month. That is assuming I make it back from touring around Wales on my motorbike safe and sound. Bearing in mind that I fell off it whilst I was in the driveway, there doesn't seem much hope I'm afraid...

Listing

```
org &8000
Mode equ &BC0E
PrintChar equ &BB5A
Locate equ &BB75
TestKey equ &BB1E
Frame equ &BD19

ld a,1
call Mode
loop
            call PrintBall
            call Frame:call Frame
            call Frame:call Frame
            call EraseBall
            call MoveBall

                  ld a.,47
                  call TestKey
                  jr z, loop
                  ret

MoveBall
                  ld a, (ballx)
                  ld b,a
                  ld a, (dx)
                  add b
                  ld 9ballx), a
                  cp 1
                  jrz, boing
                  cp 40
                  ret nz

                  boing
                  ld a,(dx)
                  xor 255
                  ld (dx), a
                  ret

PrintBall
                  ld a, (ballx)
                  ld h,a
                  ld a,(bally)
                  ld l,a
                  call Locate
                  ld a,42
                  call PrintChar
                  ret

EraseBall
                  ld a,(ballx)
                  ld h,a
                  ld a,(bally)
                  ld l,a
                  call Locate
                  ld a,32
                  call PrintChar
                  ballx db 1
                  bally db 2
                  dx db 1
                  dy db 1
```