Your Turn! Part 5

If you've been following the Bullfrog's programming tutorial over the last few issues, your game will be ready for the finishing touches. Read on as the boys (pictured above) explain what to do...

How to program your own games in assembler

Welcome to the fifth and final part of our series teaching you how to program games using the professional techniques of the Bullfrog programming team. This month, **Scott Johnston** explains how to add some finishing touches and tweaks to the game, and how you could take it further when it comes to starting out on your own games.

This is the last issue in our assembly programming series, and since the demo could almost be considered a 'proper' game, this month we will concentrate on enhancements – some simple, and others not quite so simple.

SOME SIMPLE CHANGES

The following changes are really tweaks to the equates we set up. The main problem with the game is the massive momentum we have on our guy. Way too much. What you can do is alter the variables:

MAX_SPEED, used to increase or decrease the speed at which you move.
ACCEL, the amount your man changes speed by when you move the joystick.
SLOW_DOWN, the speed at which you slow down when not moving the joystick.
JUMP, how far your man can jump, works in relation to GRAVITY.
GRAVITY, how quickly you will fall back to earth.
MAX_FALL, how far you can fall before suffering from a case of terminal death.
NO_LIVES, the actual number of lives that you have.

If you think that the collision is a bit dodgy, change the man's width, or the bad guy's width. To make it harder to die, change MAN_WIDTH to a smaller number: the smaller, the harder for you to touch something.

Continued overleaf

If you find it a bit difficult to stand on the platforms you can change the size of your feet: reduce the left foot, and increase the right foot. This will make it easier for you to stand on the platforms, but might have the strange visual effect of standing in mid-air.

If you run the version given away on this month's disk, you will see that a simple title screen is present. This was created with the code inside the routine `intro.s`, and was called at the start of `display.s`. You can do what you will with this code.

There is also a very simple 'Game over' display. At present only the highest score is coded. It will be quite a good exercise for you to change the routine that is present to that of a high-score table with names for the places.

Another thing for you to do is to try a code a delay. If you press the fire button for too long when you start, the first thing you will do is jump into the air. Similarly, when you have finished playing if you press fire at the end screen there is a chance of you going right past the introduction screen!

Collection Bonus

You may have noticed that the ankhs have a state defined for them: change the code so that if you pick up an ankh which has a different state, then you get double score or something similar. The graphics for a flashing ankh are also included in the file for this month's program listing, so give that a go.

Here it is - the final Bullfrog demo game, running and jumping with collision detection, a sound track and a scoring system.

Screen display

This is the slowest part of the program at present. It is drawn forwards at present: the data is stored consecutively, and is copied from memory to screen. Try storing the bytes as blocks of 48 (the amount we pick up inside the loop) but store them backwards. The bottom 48 bytes of the screen are stored as the 1st forty eight bytes of memory, so we can change the (a1) to a –(a1) and hence we would not need the lea command to adjust the address in a1. The quickest way would be to use the blitter chip, but it's too complex to explain here.

Speed

At present the whole screen is redrawn every turn. This is a large waste of time, in that there are only four or so sprites being updated at a time. If you change the system to a redraw, you will need to store a copy of the background that the sprites are about to be drawn onto. Then at the start of the turn, redraw the background, and then draw the new sprite positions. You should be able to get the system running at a 50th of a second very easily with this system.

Preshifted graphics

If this seems like too much work for you, then try using preshifted sprites. This is basically 16 copies of all the graphics, one at each possible shift position. These pre-shifted sprites are very quick to draw but take up over sixteen times the memory. Even with the limited graphics it will still involve a lot of work.

Organised palette

Having the palette in a specific order has some advantages. For instance, the font is an example of this, as is the bouncing logo, both are only drawn into a single plane.

There are of course loads of other little tricks that you could do to speed up this program, for example place all the drawing routines into the blitter. I am not going to tell you how to do all this for various reasons.

For starters, we do have to have *some* trade secrets! On top of that, they're mostly just little fixes and shortcuts we discovered when trying to solve a particular problem.

All the same, we think you've got enough to be going on with. This series should at any rate get you started in 68000 programming – the rest is up to you!

USE YOUR OWN GRAPHICS

Well, we finally got the *DPaint*-to-Sprite *Convert* program from the Bullfrogs, and just to make it as available to people as possible, it has been included on both Coverdisks – so if one has a read/write error, it can still be used from the other Coverdisk. Incidentally, if any of your Coverdisks don't work, you should send them back to the address at the bottom of the "Backing Up Your Coverdisk" bit on Page 10.

The *Convert* program is a handy utility for taking pictures done with a paint package like *Deluxe Paint* and transferring them into raw binary that can be included from within a *Devpac* listing using the Incbin (Include Binary) command. There are many examples of this function embedded in the demo code, so have a good ponder to see just how it's done. *Convert* can handle whole screens or those split up into a grid so you can use it for your background screens as well as your sprites.

Getting at it is not too much of a chore, but is not immedietely obvious. What you have to do is boot either Coverdisk (insert it into a freshly reset Amiga) and hold down the Ctrl and D keys. The startup-sequence will abort, and you will be left with a little CLI or Shell prompt to type at. Just type this and press the Return key (always press Return after typing Shell or CLI commands):

```
Convert
```

The program will run, and ask you for four answers. The first thing it needs to know is the path and filename of the picture that you want to change into binary code. Just typing in the name will not be enough — for instance, if you had a picture called Mypic.IFF on a disk called Empty, you would type in:

```
Empty:Mypic.IFF
```

Next you have to type the path and name of the data file that the *Convert* program produces. So if you want to put the binary file onto a disk called Devpac2, in a directory called Data, with a name of Mydata.BIN, you would type something like

```
Devpac2:Data/Mydata.BIN
```

The penultimate thing you have to type is the height of the graphics in pixels. If your sprites are, say, 32 pixels high, then type 32. If you want to do a whole screen (200 or 256 pixels high) just type the number. Don't forget to press Return.

Then type the width of each pixel block – again, the smaller the width, the bigger the number of blocks that the graphics will end up in. Check out the illustration for a couple more insights.

Please do not think that *Convert* is the only program that can get your graphics into raw code. There are many similar programs available, many of which are used by Public Domain coding groups like Kefrens, Rebels, LSD and many more. Some of them are better than this convert program (Kefren's *Metallion Utilities* are especially good, if incompatible with Workbench 2) and some aren't, but they all do the same job.

GETTING AT THE CODE

As usual, the code is not in a usable form straight from the Coverdisk – but it's really easy to get it onto your *Devpac* disk. Incidentally, many readers get worried that they need every single tutorial. This isn't true. So long as you have a copy of the *Devpac* disk from *Amiga Format* issue 39, you can use any of the demo code on its own.

Anyway, to get the code on to your disk all you have to do is swtich on your Amiga and load Workbench. Once the drive light has gone out and you are left at the Workbench screen, pop in the *Prodata* disk and double-click on its icon. You will see an icon labelled Install_Bullfrog_Files. Double-click on this, and insert the *Devpac2* disk when the machine asks for it. You shouldn't have to swap disks more than once or twice at the very most (though this depends on your Workbench version).

Some people have asked how they install the files onto a blank disk, rather than the actual *Devpac2* disk. The easiest way to do this is to format a blank disk, and call it *Devpac2*. Go through the above procedure, and instead of the real *Devpac2* disk, put your blank in. Then rename the disk to whatever you want. Simple eh!

OVER TO YOU

There has been a tremendous response to the Bullfrog programming series, yet this is the end of it. However, everybody wants more code! More examples, more tutorials, more techniques, more tricks, more things to avoid. So we are going to continue printing material on machine code and putting source code on the Coverdisk – *provided* we keep receiving suitable material.

There are two ways in which you can help. The first is by writing in to us and asking for whatever kind of stuff you want to see – we have a good idea of the sorts of routines and demos you want, but the more requests we get for a particular subject, the more chance we'll cover it.

The other way you can help is to send in suitable material – source, examples, questions and answers. Again, we already have some material to keep the series going – but we need more all the time. Programming was the subject which most people wanted to see more of in the magazine when we asked six months ago, so if you still want to see more, please help us. Send in your requests and submissions to; *Devpac Continued*, *Amiga Format*, 30 Monmouth St, Bath BA1 2BW, UK.