

Enhancing DarkBASIC Professional with FPS Creator

When I started planning this tutorial, I felt like a fraudster. Firstly, the content is based on a presentation given by Lee Bamber at the DarkBASIC Professional Convention last November. Secondly, when you see just how little work is involved in using an FPSC level in your DB Pro game, you'll realise why I felt like I was about to short-change you this month. However, I've dug a little deeper, rooted out some more of the finer details of this process, highlighted the areas that you may struggle with and added more features to what was a fascinating insight into the world of FPSC levels.

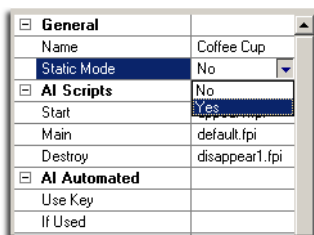
Firstly, I will be working with the code from last month's Unity tutorial. Did you notice the corridors that the butterfly was navigating around looked very like an FPSC design? If you had your suspicions, you were right! The entire warren of beautifully lightmapped corridors was "knocked together" in 15 minutes using FPSC as my level editor. I even made my own Unity wall-sign to prove you can customise the décor.



Do you remember the lightmapped corridors?

Building the level

Let's start by preparing the level. It is recommended that you start at the top-left point of the FPSC map editor, which is 0,0 on the XZ plain.



Coffee Cups are Dynamic, make them static if you want to include them.

Remember also that the editor defaults to layer 5, and you might want to go down to layer 0 as your starting point, which will take you to point 0 on the Y axis. We can import all static items into DB Pro with very little effort, so this is what will be working with. Walls, floors, ceilings and even static entities such as chairs, cupboards and crates can be used. You need to bear in mind that some entities will default to dynamic mode, and you will need to change this in the item properties. When imported into DB Pro, they will be part of the static scenery. Remember also that doors are dynamic, so they will not be available to us.

Let there be light

You can also add static lights, which will be lightmapped according to your specifications when the game is built. Remember that lights can be dynamic, but these won't be available to us in the imported map. It is worth noting at this point that the highest quality settings can be used at the time the game is built, and can be adjusted later, when the level is imported.

Although the built map is included with the tutorial download, the FPSC design file isn't. This is simply because it uses the Kitchen pack which you may not have acquired, and also external texture files such as the Unity wall-sign. However, it is very easy to create a similar map in just a small amount of time. For those people without FPSC, the exported level is included in the download so that you can experiment with the code.

Design Considerations

Before we go any further, there is another important design aspect to consider. We will see the optimisation in action later on in the tutorial. The level model, without any additional involvement on our part, has the ability to "remove" hidden faces, thus maintaining the frame rate. This is one of the core aspects of FPSC that allows complex and huge levels to be built without too much consideration given to performance.

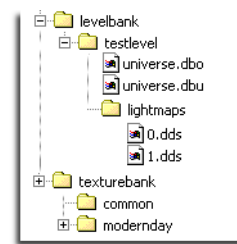
To take advantage of this feature, it is important to enclose areas when building large maps. This is achieved by ensuring rooms and corridors have ceilings first and foremost. It is also improved by obstructing visibility from one area to another, by putting corners in corridors, or other obstructions (such as walls) between different zones to divide the map into distinctive areas that the optimisation can benefit from. Later on, we will see the optimisation in action, and be able to assess our maps for suitability.

Moving On...

You can now "Build Game" in FPSC, and put your level-editing tool to one side. It's almost time to move on to the DB Pro code. Before we do, we need to unzip a couple of files, and there's a lot of files that are of no concern to us in the build folder, which should be deleted. Simply follow these step by step directions:

- Delete all of the subfolders with the exception of *levelbank* and *texturebank*.
- Create a new subfolder within *levelbank* called *testlevel*.
- Within *levelbank*, there is a zip folder named *level1.zip*. Open this, and extract the 2 files named *universe.dbo* and *universe.dbu* into your new *testlevel* subfolder. The password for this zip file is "mypassword".
- Also to be extracted from the zip file are all files in the *lightmaps* subfolder. This is a subfolder of *testlevel*

Your preparation is now complete. Your directory structure will look like the diagram opposite, but with different lightmap and texture files:



Writing the code

The hard work is now done. Using this level in your DB Pro application is very easy. Here is the one line of code that loads the level:

```
load static objects map$, 0
```

where map\$ is the name of the dbo file. Let's put it into context, with a few more lines of code:

```
texturesize=0
map$="levelbank\testlevel\universe.dbo"
```

```
load static objects map$,texturesize
```

The *texturesize* parameter is what was referred to earlier when we built the game with the highest quality settings. Using the example of 512 x 512 texture sizes, a *texturesize* parameter of 2 will load the textures at half the resolution. Thus, you can set the detail level at loading time rather than design time.

The level files are very particular about the file structure; you must ensure you are working in the directory where the *levelbank* folder is situated. You can use **set dir** to change the folder if necessary:

```
texturesize=0
map$="levelbank\testlevel\universe.dbo"
directory$ = "C:\MyLevel\"

set dir directory$
load static objects map$, texturesize
```

That is all there is to loading your FPSC level! You can open the source code in the tutorial download and see how it was implemented in the Unity tutorial. The Lua code has been integrated into the DB pro code to allow non-Unity users to work through the tutorial. You can also build your own hive of corridors and use it with the code to see it working. There are several other new commands in use, which are highlighted below, along with a quick and easy technique for watching the optimisation in action.

More fun with FPSC maps

Optimisation of your FPSC levels has been mentioned earlier in this tutorial, and the good news is that it all happens automatically. It is possible to see it in action, and be able to check your maps are well designed. By viewing your map in wireframe mode, you can see the different sections appearing and disappearing. The simplest way to implement this is to attach a change in viewing mode to the mouse button. This has been done in the source code in the downloadable example.

```
if mouseclick()=1 then set static objects wireframe on
if mouseclick()=0 then set static objects wireframe off
```

To make the most of your FPSC maps, there are a number of new commands:

static raycast(ox#, oy#, oz#, nx# , ny#, nz#)

This command allows you to return the distance to the static world along the specified ray. It works in exactly the same way as intersect object, but an object number is not specified.

This is an extremely useful command, and returns a lot of information, which is stored in the DarkBASIC Professional checklist. It allows for many actions, such as easy placement of decals, and vectors for bouncing objects off walls. You will note that a new command, *checklist fvalue()* has been introduced to return floats. The data returned is as follows:

Checklist string\$(1)

a string with the first part the mesh number and the second part the frame number

Checklist string\$(2)

a string with 3 parts storing the vertex indices for the triangle that was intersected

checklist fvalue a(3), checklist fvalue b(3), checklist fvalue c(3)

object space coordinates for vertex 0 e.g. 1.0, 2.0, 3.0

checklist fvalue a(4), checklist fvalue b(4), checklist fvalue c(4)

object space coordinates for vertex 1 e.g. 1.0, 2.0, 3.0

checklist fvalue a(5), checklist fvalue b(5), checklist fvalue c(5)

object space coordinates for vertex 2 e.g. 1.0, 2.0, 3.0

checklist fvalue a(6), checklist fvalue b(6), checklist fvalue c(6)

world space coordinate where the collision took place e.g. 1.0, 2.0, 3.0

checklist fvalue a(7), checklist fvalue b(7), checklist fvalue c(7)

normal of polygon that was hit e.g. 0.0, 1.0, 0.0

checklist fvalue a(8), checklist fvalue b(8), checklist fvalue c(8)

reflection normal based on angle of impact e.g. 0.0, 1.0, 0.0

get static collision x()

Returns the x coordinate of the static raycast collision point. There are two more corresponding commands; *get static collision y()* and *get static collision z()*

Delete static objects

This command will delete the map loaded with *load static objects*

static volume(ox#,oy#,oz#,ox#,oy#,oz#,1.0)

This is specific to FPS style games. It tests for collision of a volume against the static world. The two sets of coordinates are the old and new positions. The final parameter is the scale of your volume. A scale of 1.0 is an ellipsoid of X/Y/Z dimensions 10,30,10. This volume represents your character, and the command will return a value of 1 if a collision has occurred.

This tutorial has opened up a new world of possibilities by using FPS Creator as a level editor for DarkBASIC professional. It will also help those interested in modifying the FPSC game engine by unravelling some of the new commands.

Until next time,
Happy Coding!

Steve Vink