Hi, this tutorial we are going to look at how to implement backgrounds into your object. You should know what backgrounds are by following the basic tutorials, and now we are going to look how to implement them. This luckily, requires only a little bit of copy paste work.

Let’s start with our create event:

[CODE]

texturescript=uiz\_back\_circlestitch//i//texturescript//

texturescript\_over=-1//i//texturescript over//

texturescript\_in=-1//i//texturescript in//

texturescript\_out=-1//i//texturescript out//

sprite\_normal=spr\_uiZ\_border1//p//sprite normal//

color\_normal=c\_white//c//color normal//

margin\_normal=5//v//margin normal//

sprite\_texturemode=uiz\_texturemode\_fill//t//texturemode normal

border\_texturemode=-1//t//texturemode border//

sprite\_over=spr\_uiZ\_border1\_over//p//sprite over//

color\_over=-1//c//color over//

margin\_over=-1//v//margin over//

sprite\_in=spr\_uiZ\_border1\_in//p//sprite in//

color\_in=-1//c//color in//

margin\_in=-1//v//margin in//

sprite\_out=spr\_uiZ\_border1\_in//p//sprite out//

color\_out=-1//c//color out//

margin\_out=-1//v//margin out//

[/CODE]

Just copy and paste that code and put it inside the create event, in the same script as the rest of our uiz variable definitions. Now you’ve done that all we need to do is very simple:

Somewhere at the beginning of our draw event, but inside the uiz\_cntn statement, we can put this simple script:

[CODE]

uiz\_back()

[/CODE]

Yes, that was all, no arguments needed it just uses the local variables defined inside the create event.

Now that you now how to implement a background into a background, it might also be useful to know how to create one yourself. Turns out it’s quite similar to creating your own object.

The variables “rx”, “ry”, “rlx” and “rly” along with the rest of all local object values like alpha values, are available inside your script. Besides that the one you will also need to use are the local background variables, which we pasted into our create event.

To create a background, just create a normal game maker script. You don’t need to do more. You can then use it by setting the local texturescript variable of your objects to the name of this script without putting “()” behind it. This way, we are getting a reference to your script which uiz can use.

Just look at this simple background, which draws a square:

[CODE]

///uiz\_back([n])

draw\_square(rx,ry,rlx,rly,bkcol,alpha)

[/CODE]

This example was taken from uiz\_back\_square, and it only uses the background color values, not the sprite or margin values. But the right version out of all the other “color”, “margin” and “sprite” values can be get using:

-bkcol

-bkmar

-bkspr

Out of all the values for “sprite\_normal”, “sprite\_over”, “sprite\_in” and “sprite\_over”, bkspr would automatically be the right version of all those other variables.

Also a little bit about margins: They are always pixel values (unless you make a script which interprets them differently), and are supposed to define the “edges” of your background. For example, back\_circlestitch has a big sprite in the middle, and then draws it’s edges around that. The margin values would define how big the sprite of the borders are. When making objects, it’s always a good practice to keep things tight within those margins. (For Example, a stringbox would make sure no text would be drawn on the bordersprites.) The bariable bkmar is available after calling “uiz\_back()”.

**Other background scripts:**

Some objects might need multiple backgrounds. The problem is that there is no slick way to easily do this. Therefore, you might need to duplicate an already existing script and change up all the variables in there. You also need to copy a set inside the create event. If you want an example you can look into the folder “uiz>background>back scripts with extra prefixes” (in game maker) and open the script uiz\_back\_topbar. Compare it with the default “uiz\_back”. Then in create event we could use some code like:

[CODE]

topbar\_texturescript=uiz\_back\_square//i//texturescript//

topbar\_texturescript\_over=-1//i//texturescript over//

topbar\_texturescript\_in=-1//i//texturescript in//

topbar\_texturescript\_out=-1//i//texturescript out//

topbar\_sprite\_normal=spr\_uiZ\_border1//p//sprite normal//

topbar\_color\_normal=c\_gray//c//color normal//

topbar\_margin\_normal=5//v//margin normal//

topbar\_sprite\_texturemode=uiz\_texturemode\_fill//t//texturemode normal

topbar\_border\_texturemode=-1//t//texturemode border//

topbar\_sprite\_over=spr\_uiZ\_border1\_over//p//sprite over//

topbar\_color\_over=-1//c//color over//

topbar\_margin\_over=-1//v//margin over//

topbar\_sprite\_in=spr\_uiZ\_border1\_in//p//sprite in//

topbar\_color\_in=-1//c//color in//

topbar\_margin\_in=-1//v//margin in//

topbar\_sprite\_out=spr\_uiZ\_border1\_in//p//sprite out//

topbar\_color\_out=-1//c//color out//

topbar\_margin\_out=-1//v//margin out//

[/CODE]

There can also be back\_at scripts like uiz\_back\_at(rx,ry,rlx,rly,[n backmode]) or uiz\_back\_at\_topbar(rx,ry,rlx,rly,[n backmode]) These allow you to specify a special position from where to where your background should be. Always try to keep this within the (rx,ry) to (rlx,rly) range. For example you might use:

[CODE]

uiz\_back\_at (rx+width\*0.25,ry,rlx+width\*0.75,rly)

[/CODE]

This example will scale your background to be only half the size in it’s width.

Well, that was everything you need to know about backgrounds. Next time we’ll take a look how to use surfaces and shaders, while uiz is doing it’s thing making sure that you aren’t drawing in area’s in which you aren’t supposed to draw.