Welcome back to the fifth tutorial already! You should have a decent understanding of how to position your objects, and from here on we’ll focus on some more complicated stuff, which application outside the tutorials might be somewhat unclear at first. But just try to understand all of it, and later on when working with uiz, it might become clearer.

Anyways, today we’ll talk about frames and a little about containment. Let’s get the frames out the way first.

**Frames:** Frames are like an empty canvas something to put objects in. They can provide a layer for you to use in your programming. Frames are important if you want to add structure to your ui, without actually placing objects. Think about the example from tutorial 2:

“Another example might be is that you have some sort of square, in which you want to put some health bars, stamina bars, etc…. That square with the different bars could be snapped to the right side of your screen. When we change the size of our window, the square should move with it, but we also want our bars to move with it.“ –Tutorial 2, a while ago.

We could put our health and stamina bar inside a square, but what if we don’t want it inside a **visible** square? We can use a frame instead, creating an **invisible** square. This way, we can move the bars around just by moving the frame around, without it being visible to the end user.

**How do I make frames?** Easy, frames are just objects. Just use:

[CODE]

Frame=uiz\_c(obj\_uiZ\_frame)

[/CODE]

A frame is just an object like a square, it just doesn’t really do or draw anything. This doesn’t mean it’s useless though, they are very important for structure in your ui.

An example:

EXAMPLE 16:

[CODE]

//initialize uiz

uiz\_init()

//create our frame

frame=uiz\_c(obj\_uiZ\_frame)

//our parent is the uiz controller object.

//setup some variables

frame.posinframex=px;

frame.posinframey=px;

frame.posvalx=50;

frame.posvaly=50;

frame.posvalwtype=px;

frame.posvalhtype=px;

frame.posvalw=200;

frame.posvalh=200;

//fix our square object.

uiz\_fixgeneralpos(frame)

//create our square object

square=uiz\_c(obj\_uiZ\_square)

//set the parent

uiz\_setparent(square,frame)

//setup some variables

square.posinframex=fc;

square.posinframey=fc;

square.posvalx=0.3;

square.posvaly=0.6;

square.posvalwtype=px;

square.posvalhtype=px;

square.posvalw=40;

square.posvalh=50;

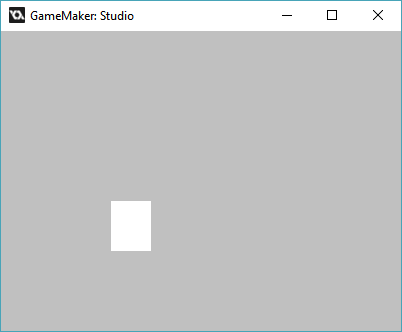
//fix our square object.

uiz\_fixgeneralpos(square)

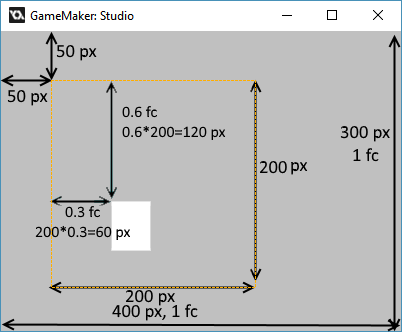
[/CODE]

Which looks like:

IMAGE 19:

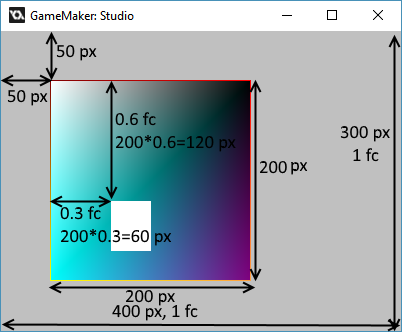


,Or with given lengths:



Using a gradientsquare:

IMAGE 12:



Not that the square object in the frame is in the exact same position as the one in the gradientsquare.

**Containment.**

What is this? This is some stuff can be your greatest friend and your worst enemy at the same time. It is a feature built into every uiz object and makes sure that that object is never drawn at places where it isn’t supposed to be drawn. Let’s say we have a window, and some buttons in it. Now the user resizes the window and makes it very small, due to a combination of space restrictions, the use of px or dp values and/or some bad programming practices, the buttons might get stuck outside the window for some reason. This obviousely cannot happen, and there would be a few ways to prevent this from happening like setting a minimum size for the window. But there is no way you can prevent this in every single case. So the “containment” mechanics in uiZ make sure that the button is never drawn outside the window. And example of this can be shown here:

*An image where the window is big enough for the button:*



*And, an image where the window is not big enough for the window:*



In the bottom picture you can see that button isn’t being drawn outside the window.

No example code will be given for this here, since there hasn’t been any explination on windows, but if you want to see it, look for example 17 in the obj\_uiz\_basictutorial object.

Technical details:

There are 2 ways in which uiz “contains” stuff, the default is using a shader, and the secondary option for when shaders aren’t compatible is using surfaces. Because of uiz’s implementation, you’ll need to make some workarounds when you are making your own uiZ objects. But you don’t need to worry about that now, you can read all about it in the tutorials showing you how to make your own objects.

Anyways this is it for this tutorial, until next time.