Hi, welcome back to this tutorial in which we’ll talk about kmouseover and scroll wheels!

**Kmouseover** is a variable that only some object have. You can check if the object you want to use has this variable by opening up it’s step event and looking for a line like:

[CODE]

kmouseover=uiz\_mouse\_isonobject\_leftcheck(id)

[/CODE]

What kmouseover will give you the following values:

0-mouse not over object.  
1-mouse over object  
2-mouse over object and being pressed  
3-mouse over object and just pressed(only works for one tick)  
4-mouse over object and just released(only works for one tick).

So, when your mouse clicks an object, you can expect kmouseover for that object to be a value of 2, while if the mouse wouldn’t be over the object at all, kmouseover would be 0. The pressed and released values (3 and 4) work like game maker.

Scrollwheels:

There are a three ways to implement scroll wheels. Within the code of an object, as a script, which we’ll cover inside a different tutorial in which I show you how to make your own objects. Also the “obj\_uiZ\_framescrollbar” object. But here we are going to look at the “obj\_uiZ\_framescrollbar” object and how to use it. Before we are actually going to make the object, we need to have some knowledge about the addx and addy variables.

**Addx/y** are variables that every objects has, but you’ll most likely only need to change these on frames. These variables are always pixels values, and they can add a or remove a little bit from your frame. To showcase these variables properly, create a new object in a new room, and in it’s create event, init uiz, and create a frame somewhere in the middle of the screen, just an empty frame.

You code should look somewhat like this:  
EXAMPLE 44:

[CODE]

//init uiz

uiz\_init()

//create a frame

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

//set values

frame.posinframex=uiz\_center

frame.posvalwtype=dp

frame.posvalw=1;

frame.posinframey=uiz\_center

frame.posvalhtype=dp

frame.posvalh=0.5;

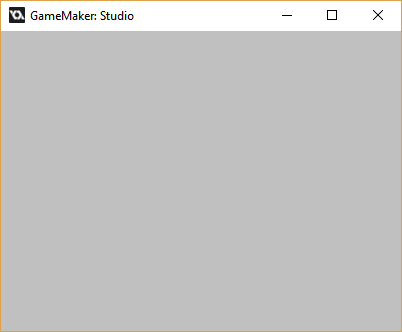
//fix our frame

uiz\_fixgeneralpos(frame)

[/CODE]

Which looks like:

IMAGE 57:



Now, we can’t see anything because the frame is invisible, but let’s add an object. In these cases I prefer using a gradient square, because it’s edges are easy to see and thus makes for a great debugging object. The default values for it might not be that pretty, but they are fine to showcase it to you now. We’ll make our gradientsquare higher than our frame though, make It twice or even trice as high.

EXAMPLE 45:

[CODE]

//init uiz

uiz\_init()

//create a frame

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

//set values

frame.posinframex=uiz\_center

frame.posvalwtype=dp

frame.posvalw=1;

frame.posinframey=uiz\_center

frame.posvalhtype=dp

frame.posvalh=0.5;

//fix our frame

uiz\_fixgeneralpos(frame)

//create gradientsquare

grad=uiz\_c(obj\_uiZ\_gradientsquare)

//put it in our frame

uiz\_setparent(grad,frame)

//make it fill our frame

grad.posinframex=uiz\_fill;

grad.posinframey=uiz\_top;

grad.posvalhtype=dp;

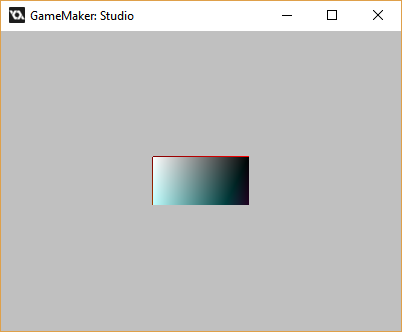
grad.posvalh=2;

//fix our grad

uiz\_fixgeneralpos(grad) [/CODE]

Now, this should look like:

IMAGE 58:



Now you can clearly see that the gradientsquare is not being fully drawn. This is because of the containment system uiz has. The left-bottom of the gradientsquare is c\_aqua and that of the right-bottom is c\_purple, which isn’t what we are seeing here.

Now, let’s just fiddle a bit with the “addy” variable. How about we set the addy variable of the frame to -100? (the minus part is important). That will give:  
EXAMPLE 46:

[CODE]

//init uiz

uiz\_init()

//create a frame

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

//set values

frame.posinframex=uiz\_center

frame.posvalwtype=dp

frame.posvalw=1;

frame.posinframey=uiz\_center

frame.posvalhtype=dp

frame.posvalh=0.5;

//add to our frame;

frame.addy=-100

//fix our frame

uiz\_fixgeneralpos(frame)

//create gradientsquare

grad=uiz\_c(obj\_uiZ\_gradientsquare)

//put it in our frame

uiz\_setparent(grad,frame)

//make it fill our frame

grad.posinframex=uiz\_fill;

grad.posinframey=uiz\_top;

grad.posvalhtype=dp;

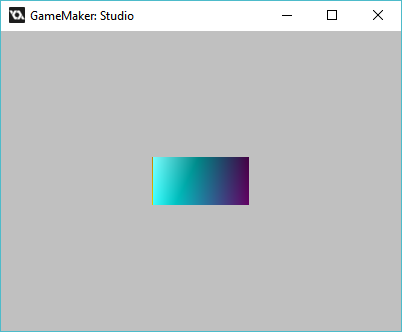
grad.posvalh=2;

//fix our grad

uiz\_fixgeneralpos(grad) [/CODE]

Which will look like:

IMAGE 59:



And voila, you can see that it the gradientsquare has moved!

Now we are going to do something slightly tricky. We are going to be moving over to a step event (we’ve only worked with create event before) and we are now going to change the value of addy using our arrow keys. To do this, we need to check for keyboard pressed, then change the addy value accordingly, and last but not least: don’t forget to fix your object, after you change it. But there is something weird about fixing our object. We are changing addy on our frame, which would logically mean that we would need to “fix” our frame, but addy doesn’t have any effect on the frame itself. It only has effect on the objects in the frame. For this reason, we need to fix all the objects inside the frame, like our gradientsquare. However, we now what objects are inside our frame in this example. It’s the gradientsquare. But sometimes and with more flexible ui, you might not know. That’s why there is a script called “uiz\_fixchildren” which will “fix” everything that is inside the frame. Because addy doesn’t directly influence the frame itself, we don’t actually need to fix the frame itself. All we need to do is call fixchildren, give it our frame’s instance id. And set “grandchildren” to true. What this grandchildren=true stuff means is that we our fix won’t only be applied to our gradientsquare, but also to possible objects within that gradientsquare. Those also need to be fixed.

[CODE]

if keyboard\_check(vk\_down) then{frame.addy++ uiz\_fixchildren(frame,true)}

if keyboard\_check(vk\_up) then{frame.addy-- uiz\_fixchildren(frame,true)}

[/CODE]

This code should work well on example 46.

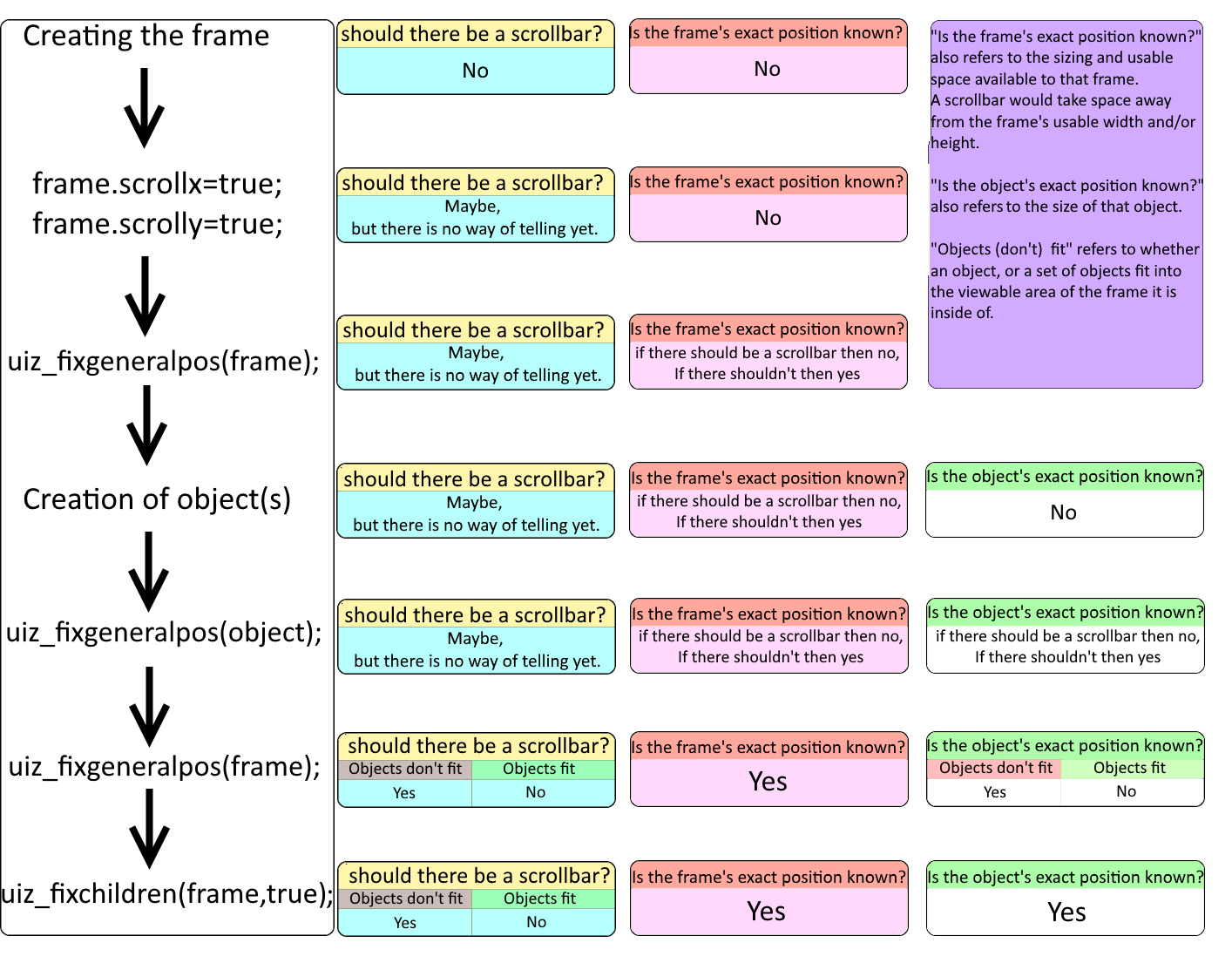
**Getting the max scrollability**. What I mean by this is how far we should be able to scroll. For example, our object is 2dp high, so we should be able to scroll for 2dp. However this would form a problem because if we would scroll all the way down now up to 2dp, the bottom of our object would be at the top of our frame, practically outside our frame. For this we need to subtract the size of our frame itself from it

Luckily, this is a simple task because there are scripts in uiz who get this for you. Considering you start placing object from 0, and no objects are placed negative coordinates, scrolling should go from 0 up till the returnvalue of uiz\_getmaxyscrollinframe(instance id). That’s some name for a script. Practically you should make sure that your addy value is always between these 2 values. Use the clamp() function for this.

**Implementing scrollbars with just a few lines.** There is functionality build into frames that allow them to easily have scrollbars. And there is one that handles everything automatically, also everything said above here. So why bother explaining all that stuff and then shoving it away? Well, it’s if you want to do some other custom stuff with scrolling, you do need knowledge of how the addx/y variables work and you should also know to use all the right scripts.

To get started with using scrollbars you simply need the following line “frameobjectid.scroll x or y = true;”. This line will say “This frame is now able to have a scrollbar.” It doesn’t have to mean that there is a scrollbar. If the content inside the frame is smaller than the frame itself, a scrollbar doesn’t have to be drawn at all, so uiZ disables it. However, there are a few problems with this. First, the frame must know all the positions of all the objects inside the frame. But that requires the objects inside the frame to know the position of the parent. And having or not having a scrollbar can influence the usable space of the frame. It almost looks like some sort of stupid loop. However, it is not. Just follow the diagram below:

IMAGE 61:



**Making a more custom scrollbar using objects.** Another way of making our frame completely scrollable, we’ll add the object “obj\_uiZ\_framescrollbar” However we need to position the object ourselves and when we’ve done that we just need to set the variable “frame” to the frame we want to scroll, and we’re automatically done.

EXAMPLE 46:

[CODE]

//init uiz

uiz\_init()

//create a big

bigframe=uiz\_c(obj\_uiZ\_frame);

//set values

bigframe.posinframex=uiz\_center

bigframe.posvalwtype=dp

bigframe.posvalw=1;

bigframe.posinframey=uiz\_center

bigframe.posvalhtype=dp

bigframe.posvalh=0.5;

//fix our frame

uiz\_fixgeneralpos(bigframe)

//create a frame

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

//put into big frame

uiz\_setparent(frame,bigframe)

//set values

frame.posinframex=uiz\_snapleft

frame.posvalwtype=fc

frame.posvalw=0.9;

frame.posinframey=uiz\_fill

//fix our frame

uiz\_fixgeneralpos(frame)

//create scrollbar

scrollbar=uiz\_c(obj\_uiZ\_framescrollbar)

//set to the right of our frame

uiz\_setparent(scrollbar,bigframe)

scrollbar.posinframex=uiz\_snapright;

scrollbar.posvalwtype=fc;

scrollbar.posvalw=0.1;

//scrollbar.posinframex=uiz\_fill;

scrollbar.posinframey=uiz\_fill;

//setup scrollbar

scrollbar.frame=frame;

scrollbar.horizontal=false;

//fix scrollbar

uiz\_fixgeneralpos(scrollbar)

//create gradientsquare

grad=uiz\_c(obj\_uiZ\_gradientsquare)

//put it in our frame

uiz\_setparent(grad,frame)

//make it fill our frame

grad.posinframex=uiz\_fill;

grad.posinframey=uiz\_top;

grad.posvalhtype=dp;

grad.posvalh=2;

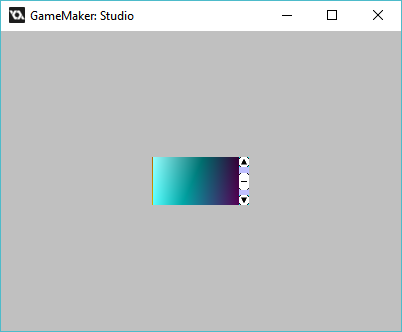
//fix our grad

uiz\_fixgeneralpos(grad)

[/CODE]

Which looks like:

IMAGE 60:



Congrats, you now know how to make a scrollbar.

That was actually the last part in the basic tutorial. In the future other tutorials will be added, maybe they will be part of the basic tutorials series, maybe they won’t. Anyways, I hope you are now able to create your own ui. This tutorial didn’t go too far into all the settings of objects, and you will have to look at the manual for that. But at least you know how to position them anywhere in a structured ui. From here I would recommend learning how to use the uiz designer using those tutorials, and after that the pro’s can follow the “make your own object” tutorials. Good luck!