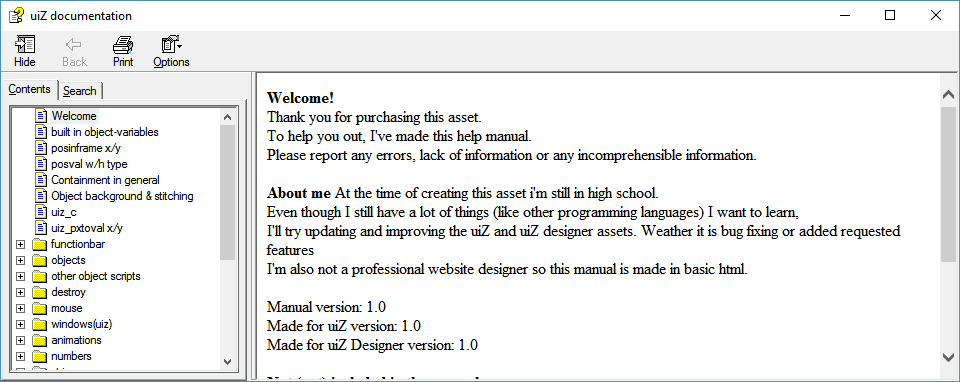
Hi, so far we’ve covered a lot about positioning and a lot of other stuff using a simple white square and a gradient. Today we’re going to learn how to customize objects, then we are going to learn how to use animations!

So a little bit about customization: It is impossible to make a different tutorial about every single customizable thing for every single little object. It would take hundreds of tutorials to do that and it would just be boring. Luckily, we’ve got a manual for this. The uiz documentation is called “uiZ.chm” and it should have come with uiz. Try to find it inside included files.

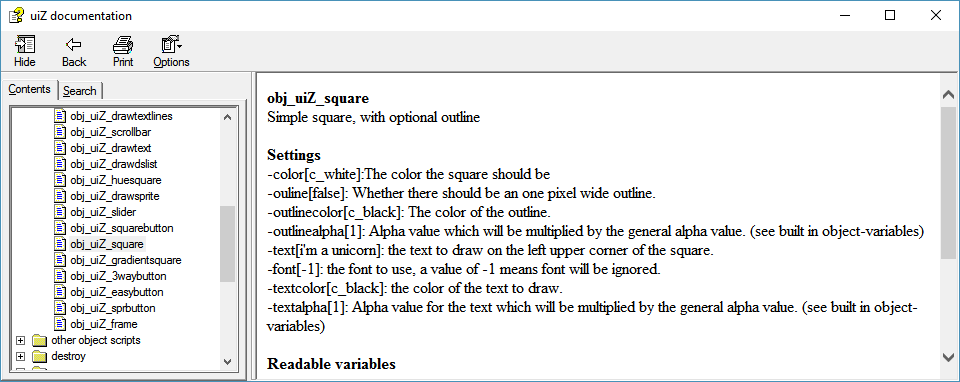
When you found it, it should look like:

IMAGE 28



Now, open up the objects folder in the tree and open “obj\_uiZ\_square”.

IMAGE 29



Here, you can see a simple description of our object. We can see that it had an optional outline. If we want to know how to enable that outline, we’d have to look into the “Settings” list. The settings list shows every single variable that you can change. It shows that in this format:

-variable\_name[default\_value]: Description.

If you want to know what kind of data you have to fill in, whether it be Booleans(true of false values), colors (like c\_white or c\_red), reals (numbers), or strings(text), then just look at the default value (the value between the [ ]). If the default value is a number, you can set the variable to another number. If the default variable is false, then you can set the variable to either true of false.

Now, before you look at the example, I want you to try this yourself. Make a new object in a new room and make a green square with an orange outline in the center of the screen of 3 dp in width and 3 dp height. Just don’t forget to init uiz, and don’t forget our “4 points”: Creation, parenting, variables/settings and fixing. (You can ignore parenting for now though)

The answer to the code:

EXAMPLE 23:

[CODE]

//initialize uiz

uiz\_init()

//create our square object

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

//setup some variables

square.posinframex=uiz\_center;

square.posinframey=uiz\_center;

square.posvalwtype=dp;

square.posvalhtype=dp;

square.posvalw=3;

square.posvalh=3;

//make our colors look how we want them to look:

square.color=c\_green;

square.outline=true;

square.outlinecolor=c\_orange;

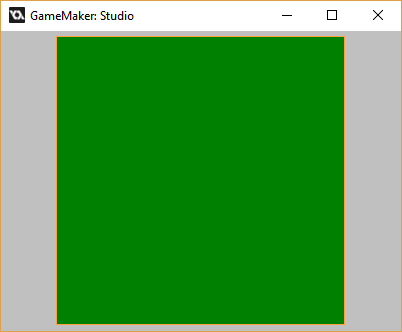
//fix our square object.

uiz\_fixgeneralpos(square)

[/CODE]

Giving:

IMAGE 30:



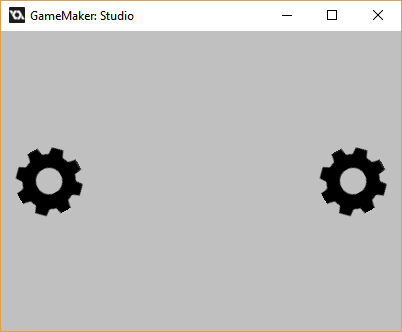
Beatifull isn’t it? Well… not really. But once you get into customizing other objects, then I’m sure you can come up with some good looking ui.

**Animations:** For this, we’ll use the object: obj\_uiZ\_rotator. What you should do now is look at the manual (open the obj\_uiZ\_rotator page in the objects folder) and read everything it says.

As the documentation says, this is just an object that draws a sprite and rotates it using an animation. But what about the animation? If we look at the default value of “animation” we can see that it’s set to “uiz\_striaght”. Well, I’m going to tell you, it’s quite easy.

Just like the immense options of settings for objects, there are also a lot of animations. We cannot mention all of them in the tutorial, but here using the documentation as a reference is the best way to look and learn to use all animations. Go to the manual and open the animation folder, and open “Animations in general” Here, there is an entire list showing you all possible animations, and a graph of what they look like. Now just for reference, create a new object in a new room and create 2 objects called “obj\_uiZ\_rotator”. Put them in positions you’d like it to be in, making them not overlap. And just create something like this:

IMAGE 31:



I created this using:

EXAMPLE 24:

[CODE]

//initialize uiz

uiz\_init()

//create our rot1 object

rot1=uiz\_c(obj\_uiZ\_rotator)

//setup some variables

rot1.posinframex=uiz\_snapleft;

rot1.posinframey=uiz\_center;

rot1.posvalwtype=dp;

rot1.posvalhtype=dp;

rot1.posvalw=1;

rot1.posvalh=1;

//fix our rot1 object.

uiz\_fixgeneralpos(rot1)

//create our rot2 object

rot2=uiz\_c(obj\_uiZ\_rotator)

//setup some variables

rot2.posinframex=uiz\_snapright;

rot2.posinframey=uiz\_center;

rot2.posvalwtype=dp;

rot2.posvalhtype=dp;

rot2.posvalw=1;

rot2.posvalh=1;

//fix our rot2 object.

uiz\_fixgeneralpos(rot2)

[/CODE]

*\*For this tutorial, I recommend trying the examples for yourself. Look into the object obj\_uiz\_basictutorial for a list.*

Now you can see 2 rotating gears. But they move a little bit fast. If we want to change the animation on them, it’d be nice to easily see them rotating. Right now it takes 1 second for the sprite to rotate 360 degrees. How do I know that? The manual! If you look at the variable “rotatingtime” you can see the description of it: “how many second one rotation takes”, and also that the default value is one second.

*Technical details:* **Why seconds?** Well, your game might run on a room\_speed of 30 (30 ticks per second), or on a room speed of 60. Using values in ticks instead of seconds would mean that the animation would take different times on different room speeds. 30 ticks would take one second if the room\_speed was 30. But if the room\_speed was 60, then it would take only half of that time. So uiz automatically calculates how many ticks fit in all those seconds, and then uses those values. It even supports mid-animation room\_speed changes which will not give you any problems.

We’ll now set the speed of the rotation to 5 seconds:

EXAMPLE 25:

[CODE]

//initialize uiz

uiz\_init()

//create our rot1 object

rot1=uiz\_c(obj\_uiZ\_rotator)

//setup some variables

rot1.posinframex=uiz\_snapleft;

rot1.posinframey=uiz\_center;

rot1.posvalwtype=dp;

rot1.posvalhtype=dp;

rot1.posvalw=1;

rot1.posvalh=1;

rot1.rotationtime=5;

//fix our rot1 object.

uiz\_fixgeneralpos(rot1)

//create our rot2 object

rot2=uiz\_c(obj\_uiZ\_rotator)

//setup some variables

rot2.posinframex=uiz\_snapright;

rot2.posinframey=uiz\_center;

rot2.posvalwtype=dp;

rot2.posvalhtype=dp;

rot2.posvalw=1;

rot2.posvalh=1;

rot2.rotationtime=5;

//fix our rot2 object.

uiz\_fixgeneralpos(rot2)

[/CODE]

Great, now our sprites rotate way slower, so we can accually see what we are doing. If you look at the manual, we can see that “animation” defines what animation we use. Right now it’s set to uiz\_straight. If we look at “Animations in general” and look for the graph of uiz\_straight, we can see that it’s a straight line. Now image the horizontal axis on the graph to be time, starting from 0 to our given “rotationtime”. Then image the vertical axis being the rotation, the bottom being 0 degrees and the top being 360 degrees. The line is just straight meaning our rotation will be linear, and it will rotate with one speed for the entire time. If we now take uiz\_quadratic\_in for example, we can see that the graph will rotate way slower in the beginning and faster in the end.

IMAGE 33:

*\*graph of uiz\_straight*

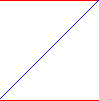
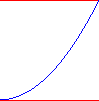


IMAGE 32:

*\*graph of uiz\_quadratic\_in*



So how do we change the animation? Simple, just change the variable animation of obj\_uiZ\_rotator to the desired animation. This can be done like this: (we are setting the right rotator’s animation to uiz\_quadratic\_in)

EXAMPLE 26:

[CODE]

//initialize uiz

uiz\_init()

//create our rot1 object

rot1=uiz\_c(obj\_uiZ\_rotator)

//setup some variables

rot1.posinframex=uiz\_snapleft;

rot1.posinframey=uiz\_center;

rot1.posvalwtype=dp;

rot1.posvalhtype=dp;

rot1.posvalw=1;

rot1.posvalh=1;

rot1.rotationtime=5;

//fix our rot1 object.

uiz\_fixgeneralpos(rot1)

//create our rot2 object

rot2=uiz\_c(obj\_uiZ\_rotator)

//setup some variables

rot2.posinframex=uiz\_snapright;

rot2.posinframey=uiz\_center;

rot2.posvalwtype=dp;

rot2.posvalhtype=dp;

rot2.posvalw=1;

rot2.posvalh=1;

rot2.rotationtime=5;

rot2.animation=uiz\_quadratic\_in//here we are setting our animation

//fix our rot2 object.

uiz\_fixgeneralpos(rot2)

[/CODE]

Now we can see that the left gear still rotates at the same speed. The right one starts of very slow, but goes faster, as it keeps rotating. You might also notice that I like “stops” when it has done a full rotation. This is because of how the graph works. The slope of the graph at the beginning and end don’t line up properly, which gives the stopping effect. Uiz\_quadratic\_inout on the other hand does have the same starting speed as ending speed. We’ll try using that and see how it looks:

EXAMPLE 27:

[CODE]

//initialize uiz

uiz\_init()

//create our rot1 object

rot1=uiz\_c(obj\_uiZ\_rotator)

//setup some variables

rot1.posinframex=uiz\_snapleft;

rot1.posinframey=uiz\_center;

rot1.posvalwtype=dp;

rot1.posvalhtype=dp;

rot1.posvalw=1;

rot1.posvalh=1;

rot1.rotationtime=5;

//fix our rot1 object.

uiz\_fixgeneralpos(rot1)

//create our rot2 object

rot2=uiz\_c(obj\_uiZ\_rotator)

//setup some variables

rot2.posinframex=uiz\_snapright;

rot2.posinframey=uiz\_center;

rot2.posvalwtype=dp;

rot2.posvalhtype=dp;

rot2.posvalw=1;

rot2.posvalh=1;

rot2.rotationtime=5;

rot2.animation=uiz\_quadratic\_inout//here we are setting our animation

//fix our rot2 object.

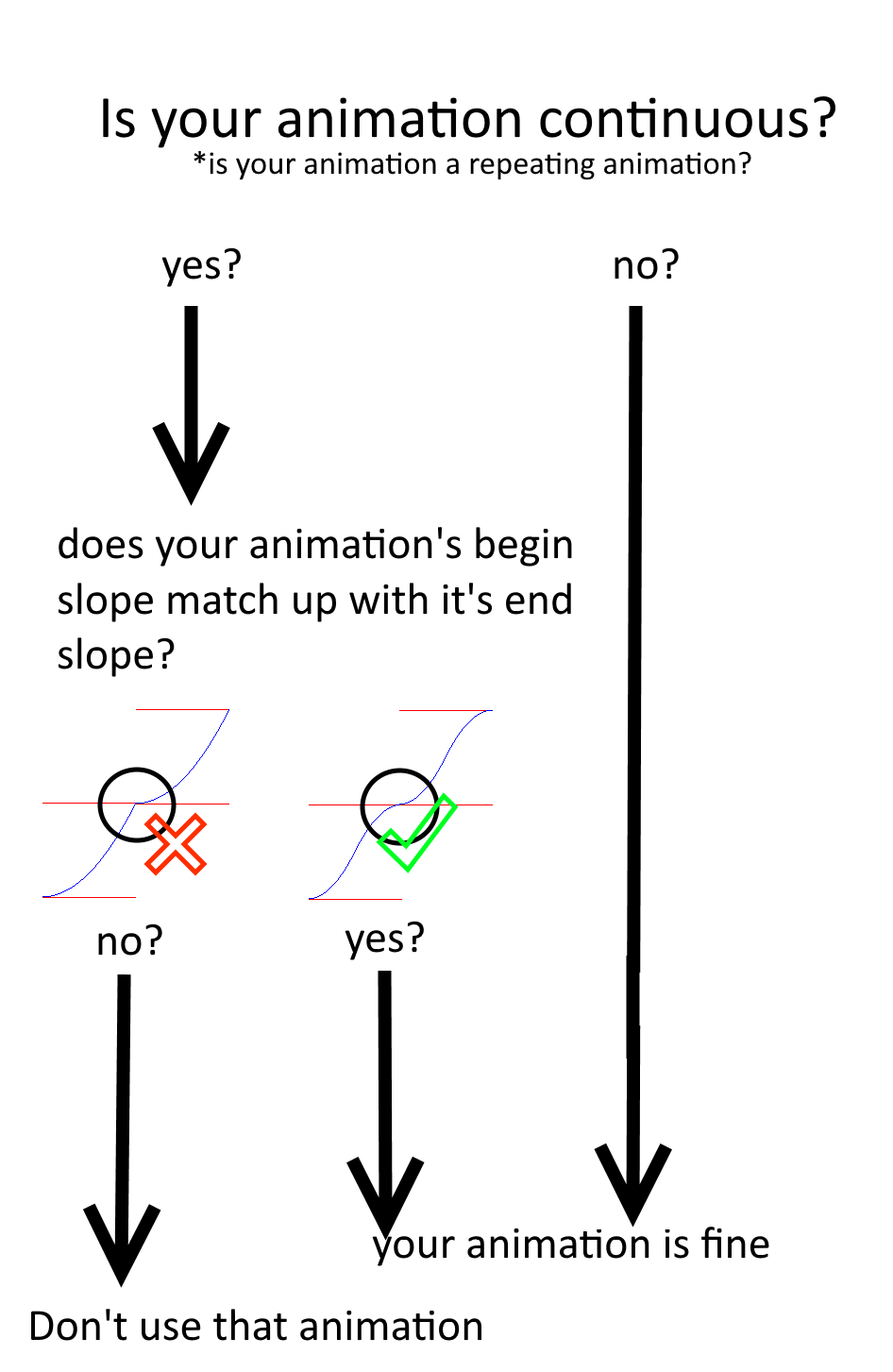
uiz\_fixgeneralpos(rot2)

[/CODE]

Way better isn’t it? We’ve now set an nice animation for the object, which first goes slow, then faster and faster, and then slows down again.

**How do I know which animations to use and which not to use?** Not all animations fit in every case. The quadratic\_in animation for example creating a nasty looking stop in our rotator object. Does this mean we should never use uiz\_quadratic\_in? No, the rotator object is an object with a repeating animation. When we press a button on a scrollbar, then that scrollbar should move down a little bit. This can be done using animations in uiz and since pressing that button and moving the scrollbar is not a continuously running animation, uiz\_quadratic\_in would be ok to use. So just use this plan:

IMAGE 34:



That concludes it for animations, Just know that you can apply your animations to a whole lot more things than rotating sprites. They are in sliders, scrollbars, and a lot of other moving things!