Tutorial 3:

Welcome back!, this time we’ll cover position types.

Remember the weird “posinframex” and “posinframey” variables from the first tutorials we skipped back then? Well, we’re going to talk about those in this tutorial.

[CODE]

square.posinframex=px;

square.posinframey=px;

[/CODE]

Let’s get some terms and macros down. Uiz has quite a few of its own macros. (macro is about the same as a constant). The one we’ve seen so far is “px”. What px stands for is pixels. Just “pixels”. When we defined a position by setting using this code:

[CODE]

square.posvalx=25;

square.posvaly=25;

[/CODE]

We’ve setting the position of square at a position of 25 pixels to the right, and 25 pixels down.

What “posinframex/y” does is it changes the way “posvalx/y” behave and work. There are a few different basic modes:

-dp: stands for density pixel.

-fc: stands for factor.

-uiz\_snapleft

-uiz\_snapright

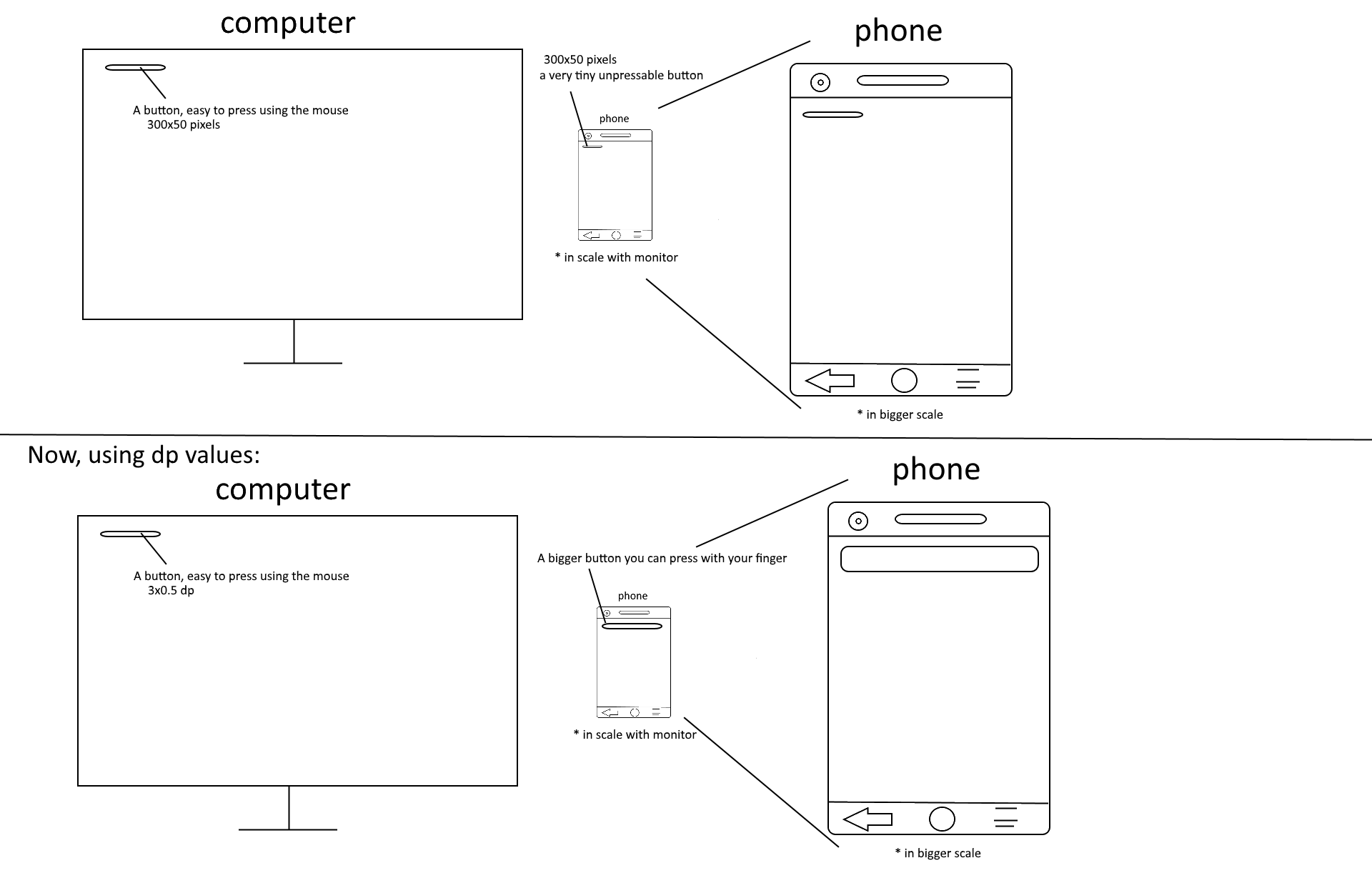
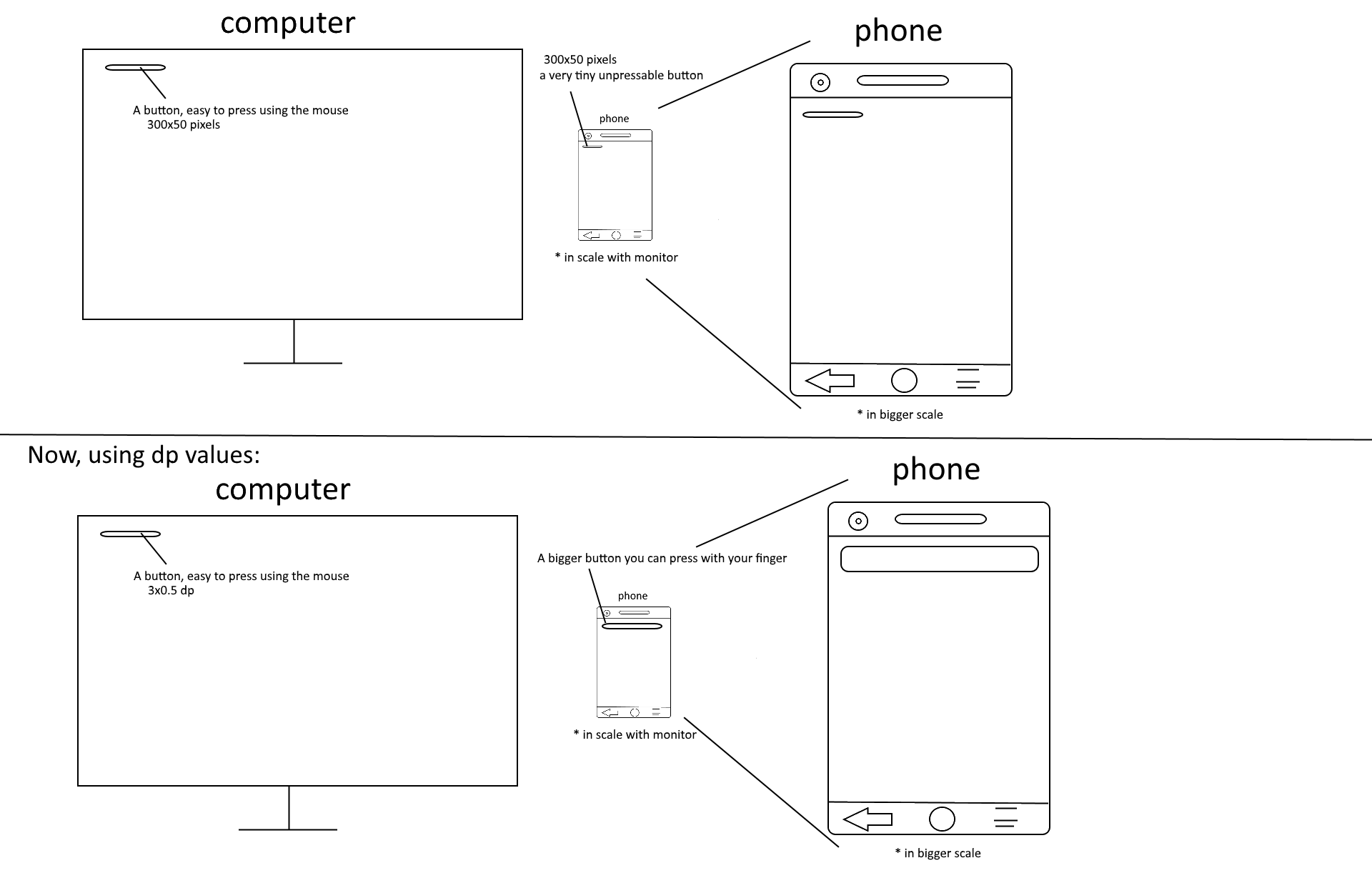
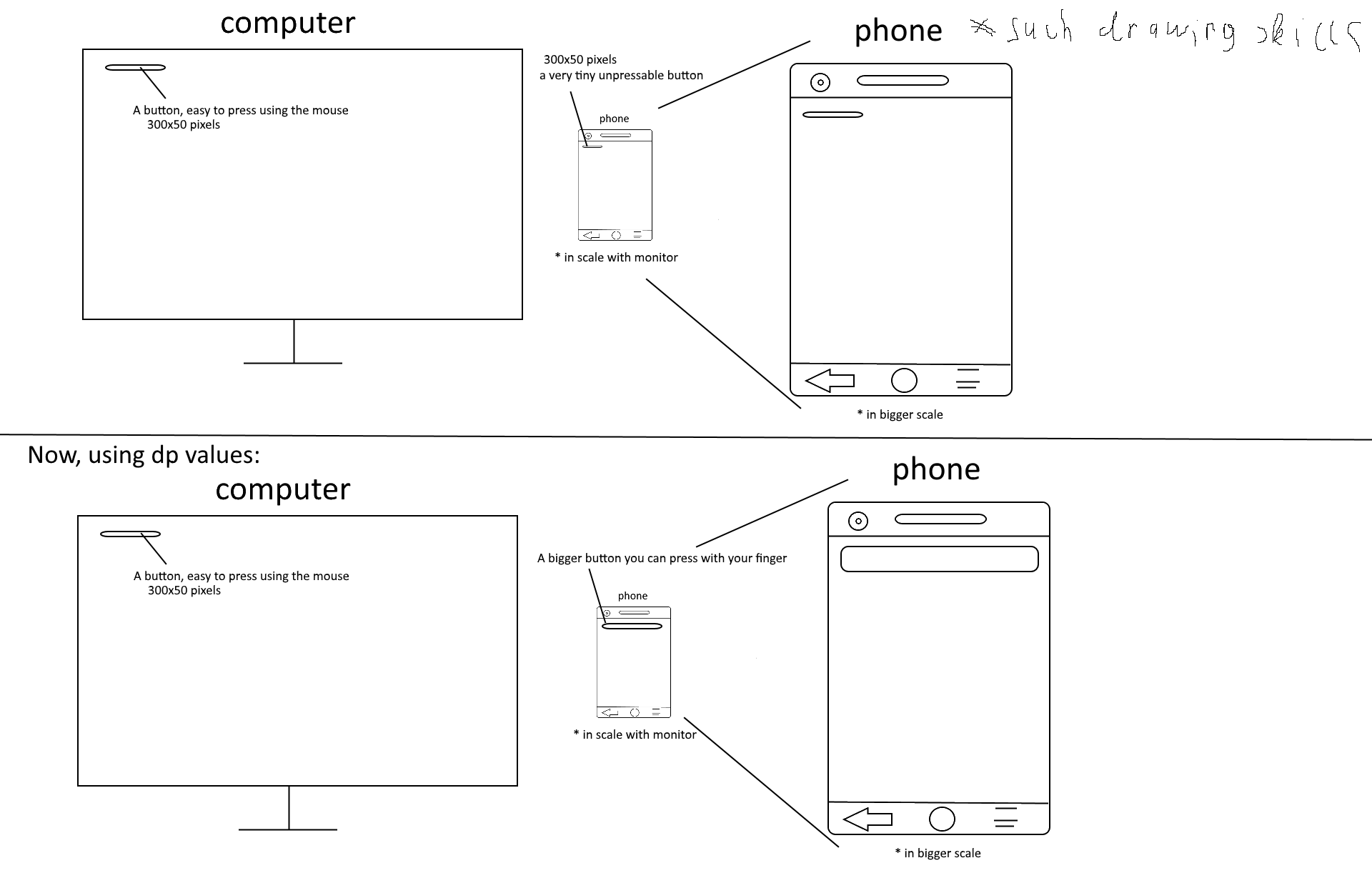
-uiz\_top

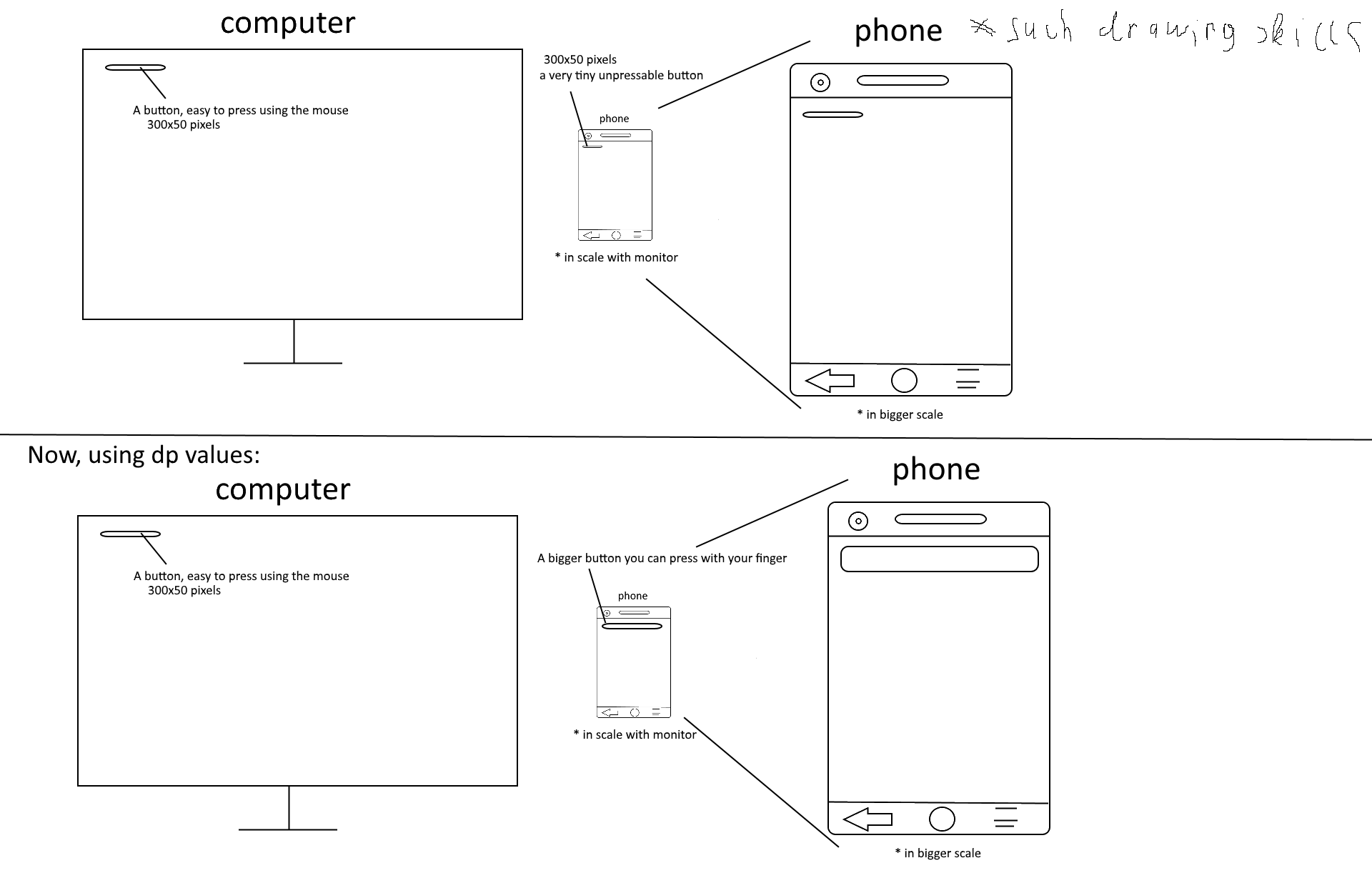
-uiz\_bottom

-uiz\_center

-uiz\_fill

**DP:** This means “density pixel” and those who have developed before on mobile platforms might be familiar with this term (also know as dpi). Practically this looks at the density of your pixels (as the name suggests). What is does is it takes a the amount of pixels that fit into one real life inch, which will differ per screen. On a computer your density might be very low, and a simple button can be only a few 100 pixels wide and even less than 100 pixels high. But if you would take a button that was say, 200x75, on a phone with a small screen, you would get a very tiny button and the user would have a hard time pressing it. That’s where dp comes in. if the density of your computer screen is 100 pixels per inch, and that of your phone is 500 pixels per inch, then the button would need to be 5 times as big to get the desired result. That’s where the DP data type comes in. saying a distance should be “3 dp” would mean that it would be 300 pixels wide on your computer screen , and 1500 pixels wide on you phone’s screen. IMAGE 7:





Now, how do I put an object at a position of say ,(2dp ,1dp)? , easy! Just use following code:

[CODE]

square.posinframex=dp;

square.posinframey=dp;

square.posvalx=2;

square.posvaly=1;

[/CODE]

This should give you a total code of:

EXAMPLE 7:

[CODE]

//initialize uiz

uiz\_init()

//create our square object

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

//setup some variables

square.posinframex=dp;//set the type of posvalx to a pixel density value.

square.posinframey=dp; //set the type of posvaly to a pixel density value.

square.posvalx=2;//this in combination with posinframex specifies “2 dp”

square.posvaly=1;//this in combination with posinframey specifies “1 dp”

square.posvalwtype=px;

square.posvalhtype=px;

square.posvalw=50;

square.posvalh=50;

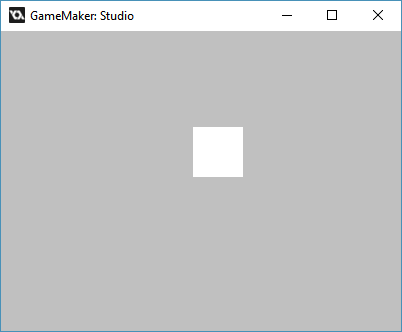
//fix our square object.

uiz\_fixgeneralpos(square)

[/CODE]

This should give this result:

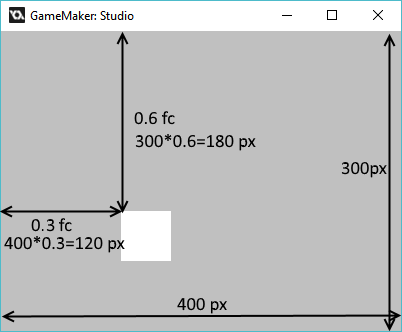
IMAGE 8



You can see that dp is quite a different scale from px.

**FC:** Fc can be very useful when working with the scaling, animating, etc. of objects. What it does is it takes the size of your parent (which is the size of you window/screen when no parent has been set), and when it has that size, it multiplies that size by a given value and uses that value. For example, if we want to put an object in the middle of the screen, we’d have to give it an fc value of 0.5 fc. This gives us the middle of the screen. An fc of 0 would give you the left side of the screen, and a value of 1 fc the right of the screen. You can say the same about an object when it has a parent, just replace “screen” with “parent”.

IMAGE 9:



We can apply this to our code by setting posinframex and posinframey both to the macro “fc”:

EXAMPLE 8:

[CODE]

//initialize uiz

uiz\_init()

//create our square object

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

//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=50;

square.posvalh=50;

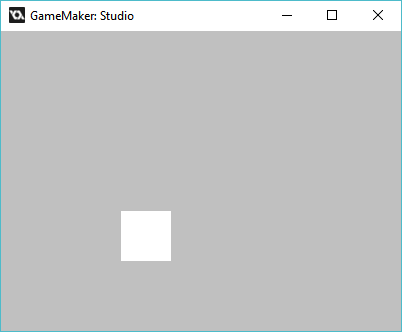
//fix our square object.

uiz\_fixgeneralpos(square)

[/CODE]

This will give:

IMAGE 10:



If we get back to example 6 from tutorial 2, in which we used parents, we can apply the same values (of 0.3x0.6 fc) to the square inside the gradient. We’ll get something like this:

EXAMPLE 9:

[CODE]

//initialize uiz

uiz\_init()

//create our gradientsquare object

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

//our parent is the uiz controller object.

//setup some variables

gradient.posinframex=px;

gradient.posinframey=px;

gradient.posvalx=50;

gradient.posvaly=50;

gradient.posvalwtype=px;

gradient.posvalhtype=px;

gradient.posvalw=200;

gradient.posvalh=200;

//fix our square object.

uiz\_fixgeneralpos(gradient)

//create our square object

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

//set the parent

uiz\_setparent(square,gradient)

//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;

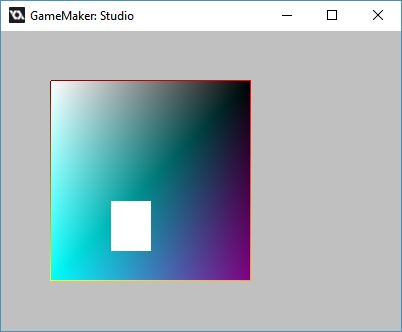
square.posvalh=50;

//fix our square object.

uiz\_fixgeneralpos(square)

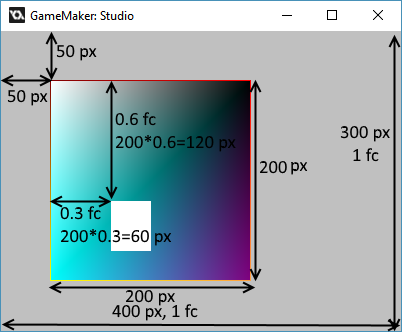
[/CODE]

Which looks like:



We can add lines and to make thing a little more clear again:

IMAGE 12:



**Modes that disable the use of posvalx and/or posvaly**

Some values for posinframex and posinframey disable the use of posvalx and posvaly, as they aren’t needed for the purpose of what posinframex/y is set to. Some examples of these values are:

-uiz\_snapleft

-uiz\_snapright

-uiz\_top

-uiz\_bottom

-uiz\_center

-uiz\_fill

From the name you can kind of guess what they do. If we say “posinframex=uiz\_snapleft” than that will mean that the object will be placed on the left of it’s parent, or on the left of the screen if no parent is set. The variable “posvalx” takes no part in this.

Now what does uiz\_snapleft look like? Well, setting “posinframex=uiz\_snapleft” would look like:

[CODE]

//initialize uiz

uiz\_init()

//create our square object

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

//setup some variables

square.posinframex=uiz\_snapleft;

square.posinframey=fc;

square.posvaly=0.6;

square.posvalwtype=px;x

square.posvalhtype=px;

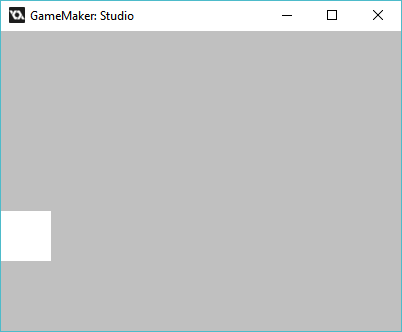
square.posvalw=50;

square.posvalh=50;

//fix our square object.

uiz\_fixgeneralpos(square)

[/CODE]



If we want our object to be in the right bottom of the screen we could set “posinframey=uiz\_snapbottom” and “posinframe=uiz\_snapright”like this:

EXAMPLE 11:

[CODE]

//initialize uiz

uiz\_init()

//create our square object

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

//setup some variables

square.posinframex=uiz\_snapright;

square.posinframey=uiz\_snapbottom;

square.posvalwtype=px;

square.posvalhtype=px;

square.posvalw=50;

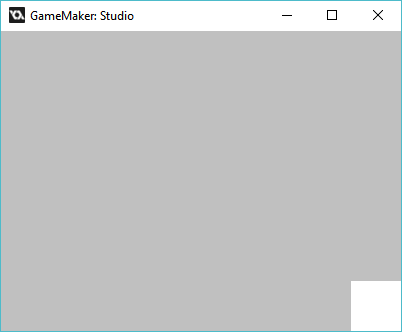
square.posvalh=50;

//fix our square object.

uiz\_fixgeneralpos(square)

[/CODE]

Which looks like:



**Uiz\_center**

There is also an macro called uiz\_center, which as the name suggests centers the object. A code like this will give following results:

EXAMPLE 12:

[CODE]

///Example 12:

//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=px;

square.posvalhtype=px;

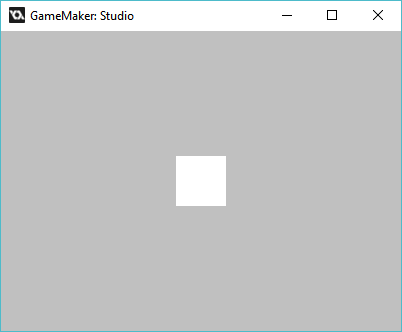
square.posvalw=50;

square.posvalh=50;

//fix our square object.

uiz\_fixgeneralpos(square)

[/CODE]



**Uiz\_fill**

This is a special type, since it not only ignores the input from posvalx and posvaly, it also completely ignores the variables posvalwtype, posvalhtype, posvalw and posvalh. After setting “posinframex=uiz\_fill”, posvalx, posvalwtype and posvalh will be rendered useless. Setting “posinframex=uiz\_fill” does not change the use of the variables posvaly, posvalhtype and posvalh, these can disabled if posinframey was also uiz\_fill.

If we set the y to fill, we get this result:

[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\_fill;

square.posvalwtype=px;

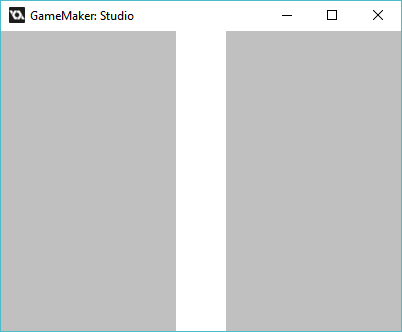
square.posvalw=50;

//fix our square object.

uiz\_fixgeneralpos(square)

[/CODE]

IMAGE 14:



Well, that was it for this relatively long tutorial, Next time we’ll look at how to use the types (you we used on posinframex/y) on posvalw/htype.