In which we animate a ball moving on the screen.

LAB 06: Bouncing Ball, Part 1

Objective

Now that we have played around with the basics of what Storyboarding in Xcode has to offer, it's time to start writing code.

I'd like to start demonstrating the basics of writing code straight away. But, I can't.

There's a lot of stuff to put into place first.

So, what we're going to do is build a simple physics framework. Meaning a small-ish iPad app that shows a ball bouncing around.

We'll use this project as the foundation from which to explore other coding concepts.

Don't worry if you get confused. We're going to start off difficult, then pull back after this Lab and make things simpler.

It's kind of like being thrown in the deep end, in order to learn how to swim.

Giving up, or thinking programming "isn't for you" at this point (or after this lab), would be like drowning.

Please don't drown. We'll get through this together. I promise.

With that… onward!

Create a new iPad project

This project is going to look a lot like the first one (the one from LAB 1). That's the one where you created all the colored rectangular views.

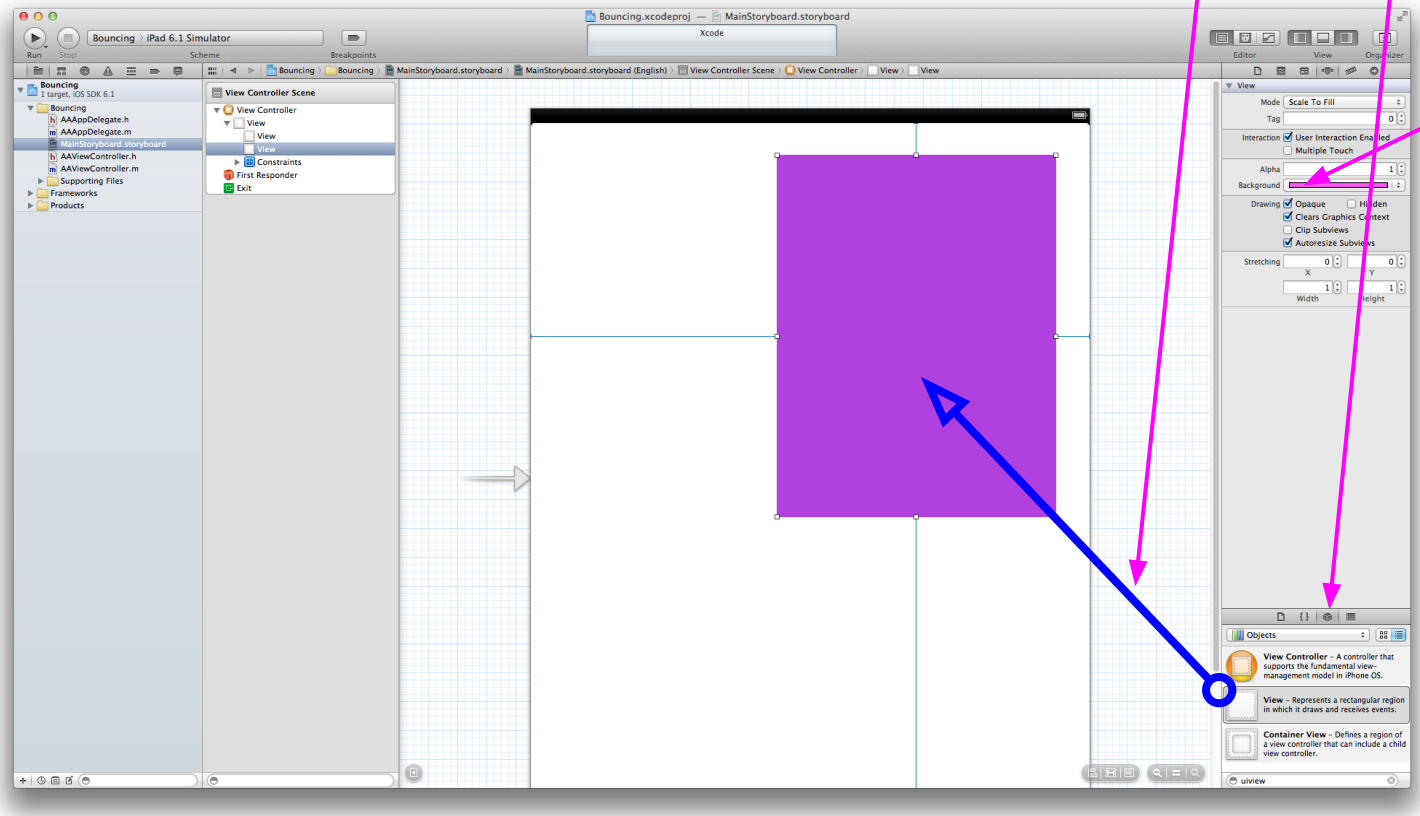
Now, instead of using plain ol' UIViews, we're going to use UIButtons.

Create another iPad project, using the same settings as before:.

- Single View Application
- Devices: iPad
- Use Storyboards
- Use Automatic Reference Counting

Add a view

Drag and drop a **View** from the **Object library**.



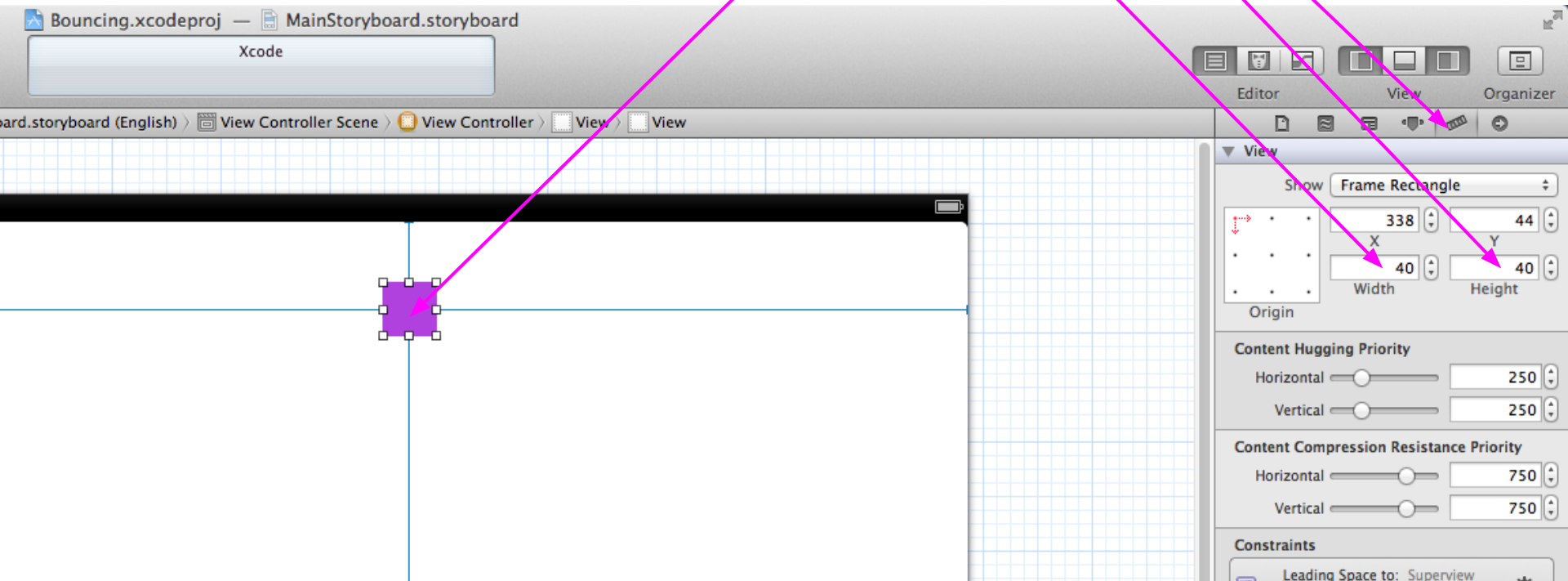
Change the view's **Background** color.

Resize the view... make it a ball.

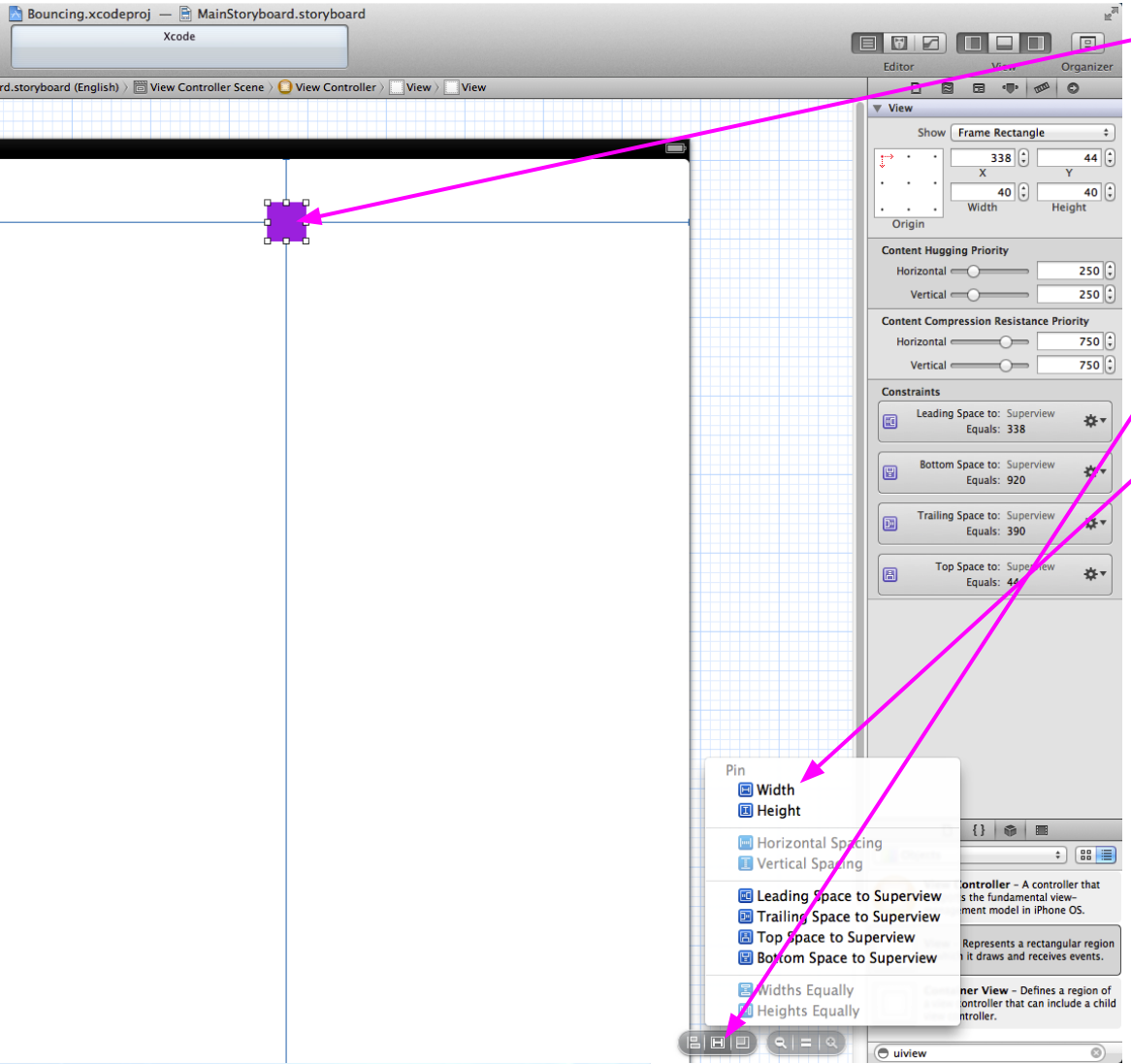
With your new **View** selected.

Select the **Size inspector** tab.

Then adjust the **Width** and **Height** values of the view, making it 40 points by 40 points.



Add a width constraint



With your ball selected, add a width constraint and a height constraint. To do this…

Tap the **Pin** layout constraint button.

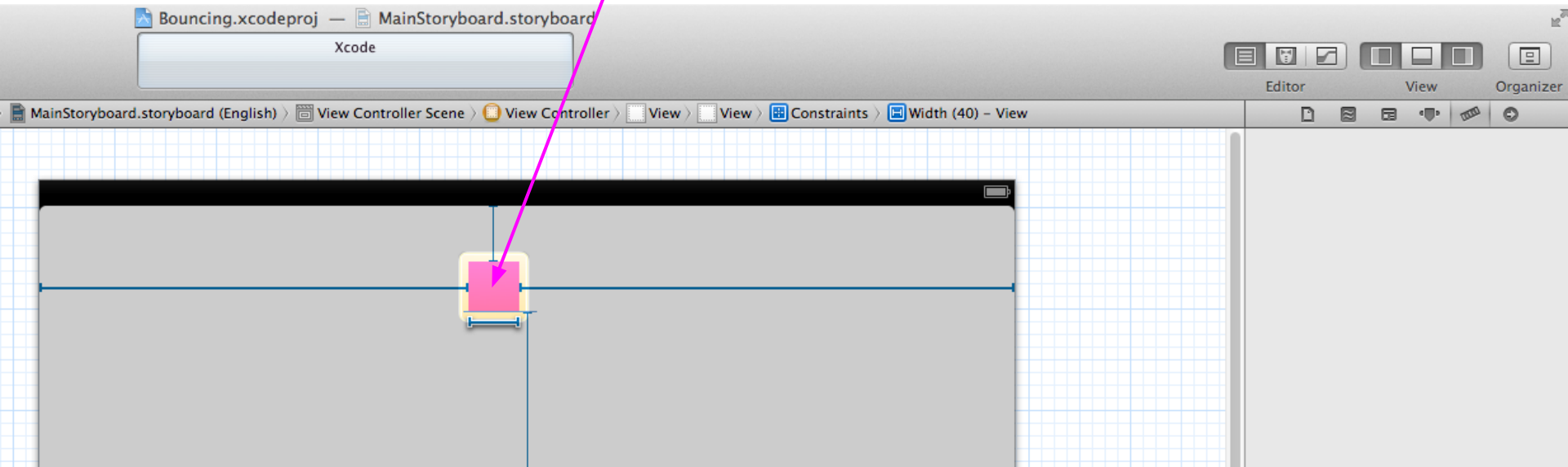
Then select **Width** to add a width constraint.

You should see this.

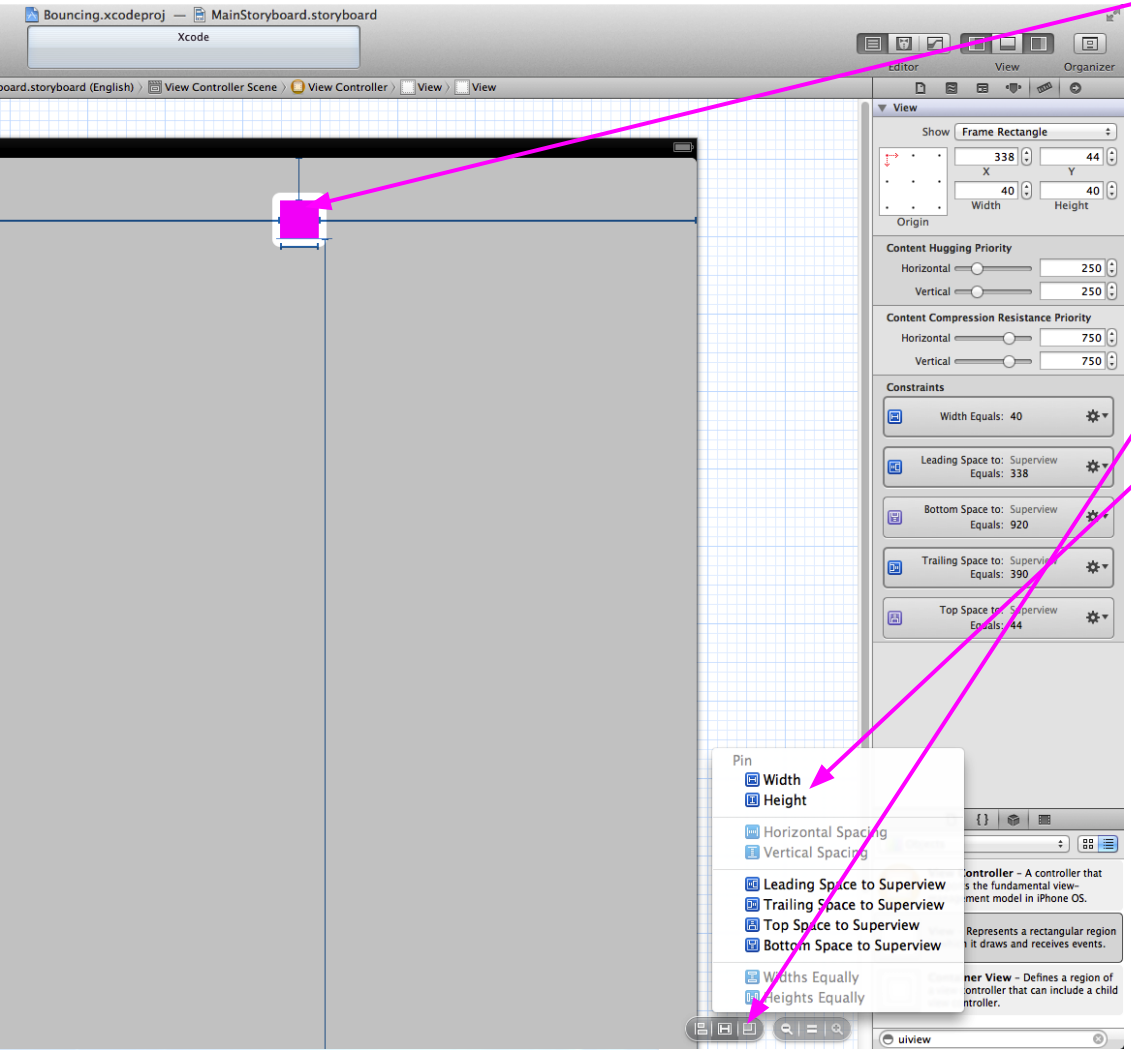
Your new width constraint should be visible now.

It's also the selected item.

You need to select your ball view to continue. Tap on the colored part of the ball view to select it.



Add a height constraint

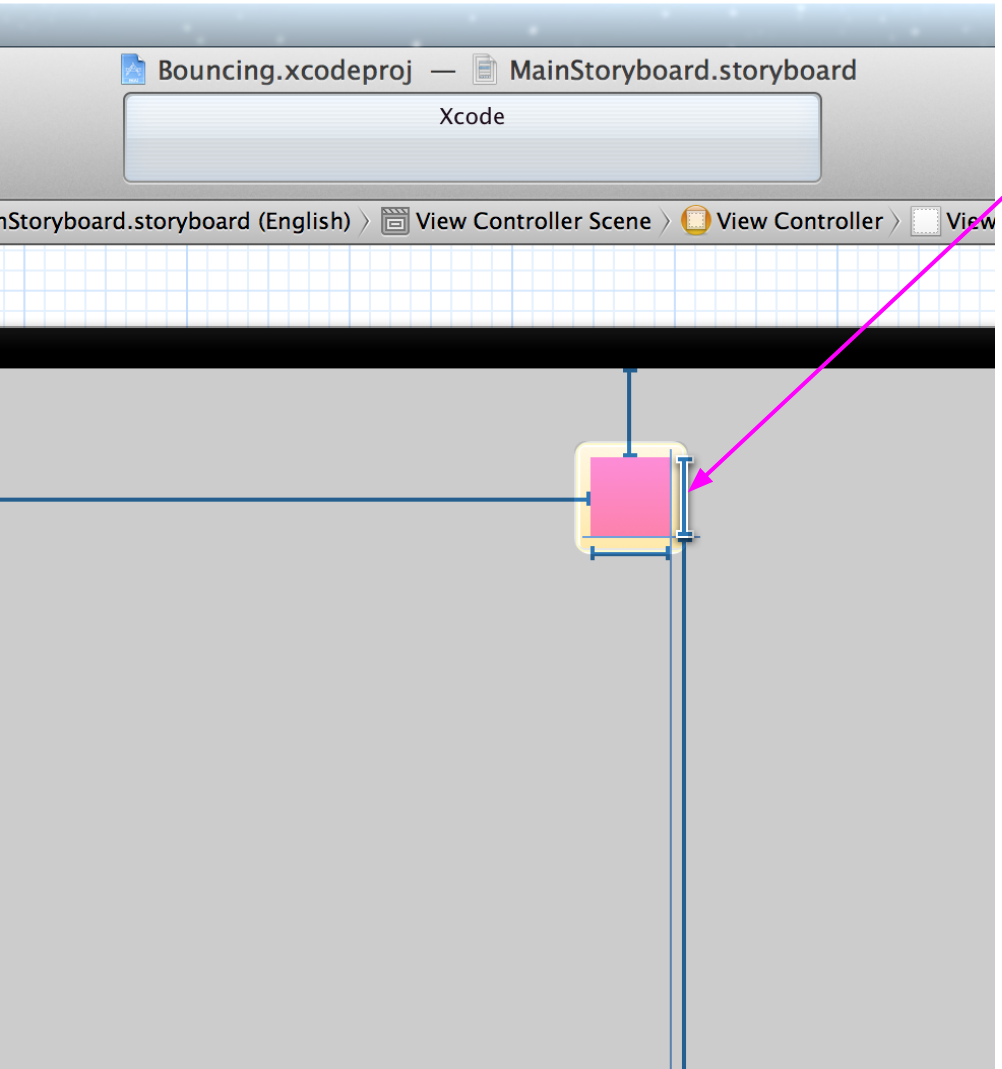


Verify your ball is selected. Now, we'll add a height constraint.

Tap the **Pin** layout constraint button.

Then select **Height** to add a height constraint.

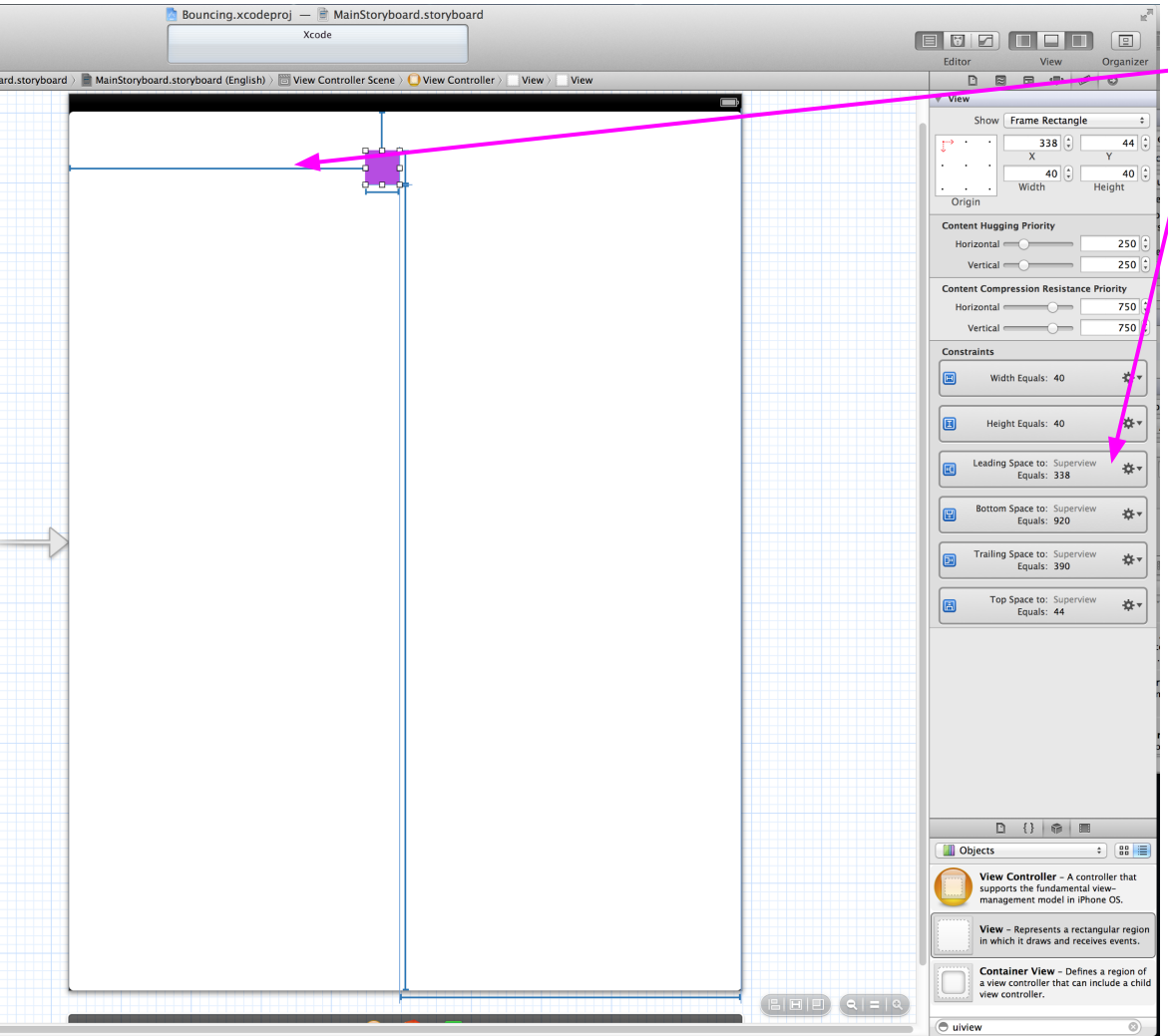
Et Voilà ! Your Height constraint.



See the new **Height** constraint?

That's what you want.

The other constraints.

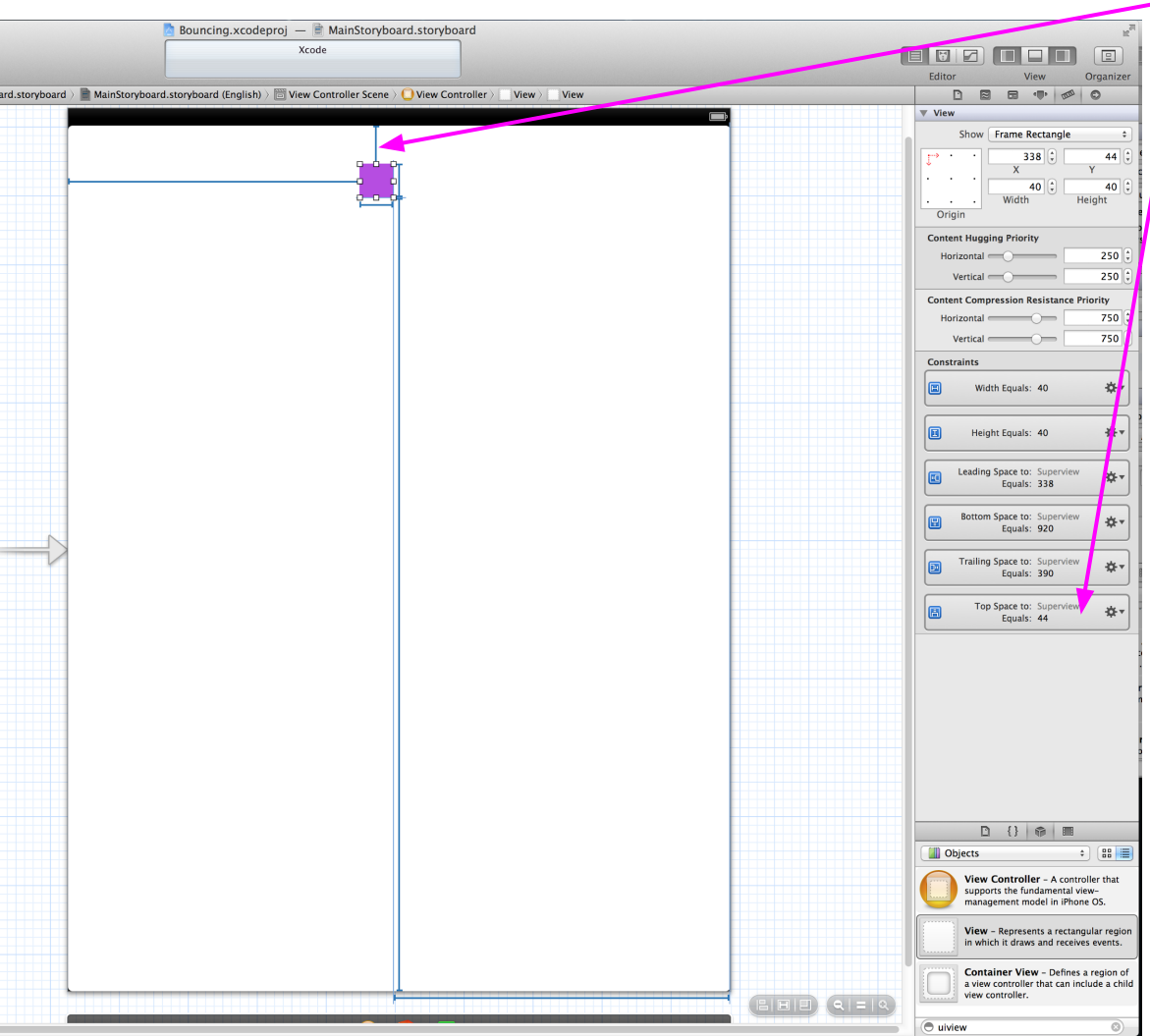


Besides the Height and Width constraints, there are a four others.

1) There an x-position constraint (a.k.a. **Leading Space to Superview** constraint).

This is the amount of space between the left edge of the screen and the left edge of the ball view.

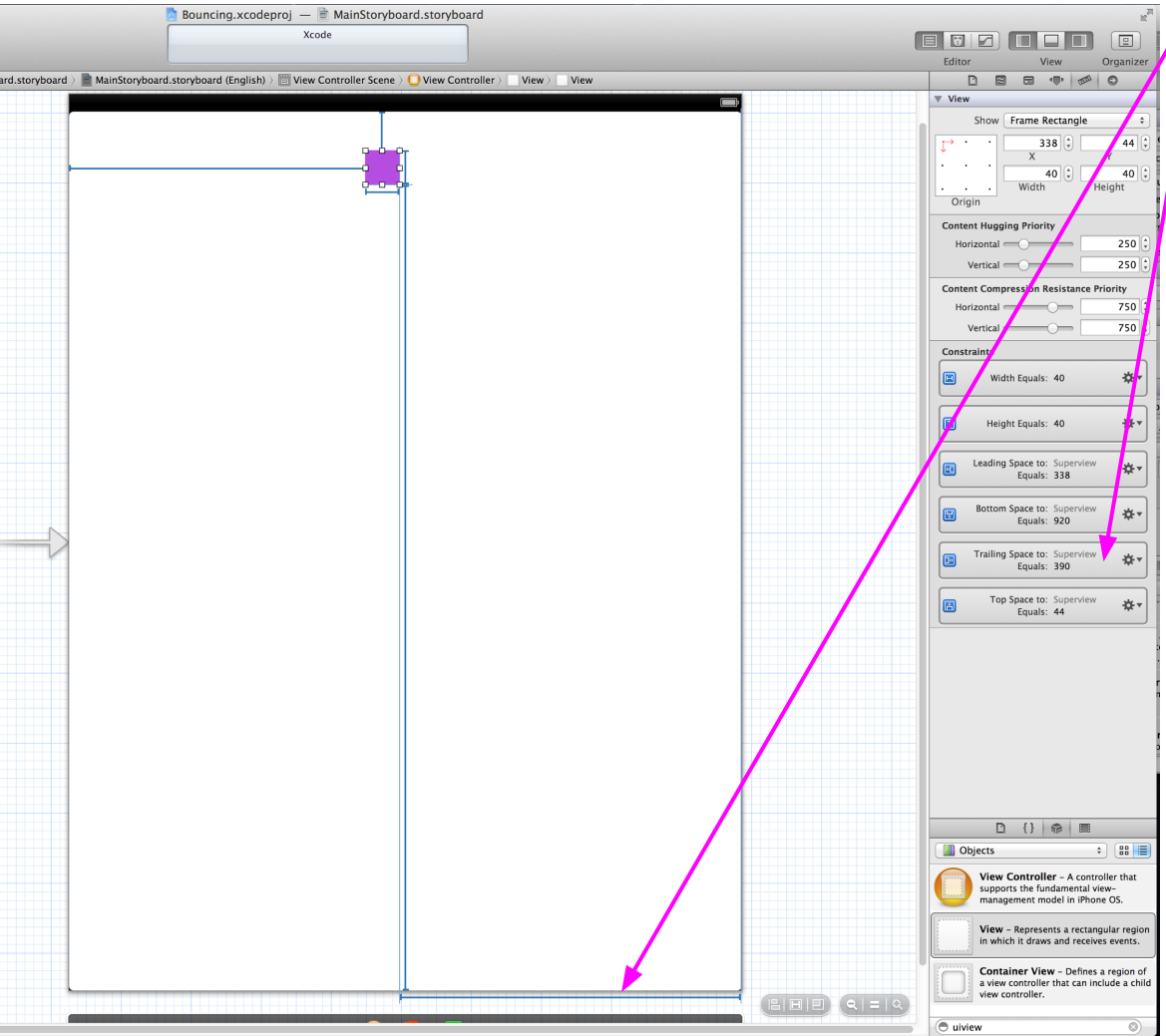
More constraints...



2) There a y-position constraint (a.k.a. **Top Space to Superview** constraint).

This is the amount of space between the top edge of the screen and the top edge of the ball view.

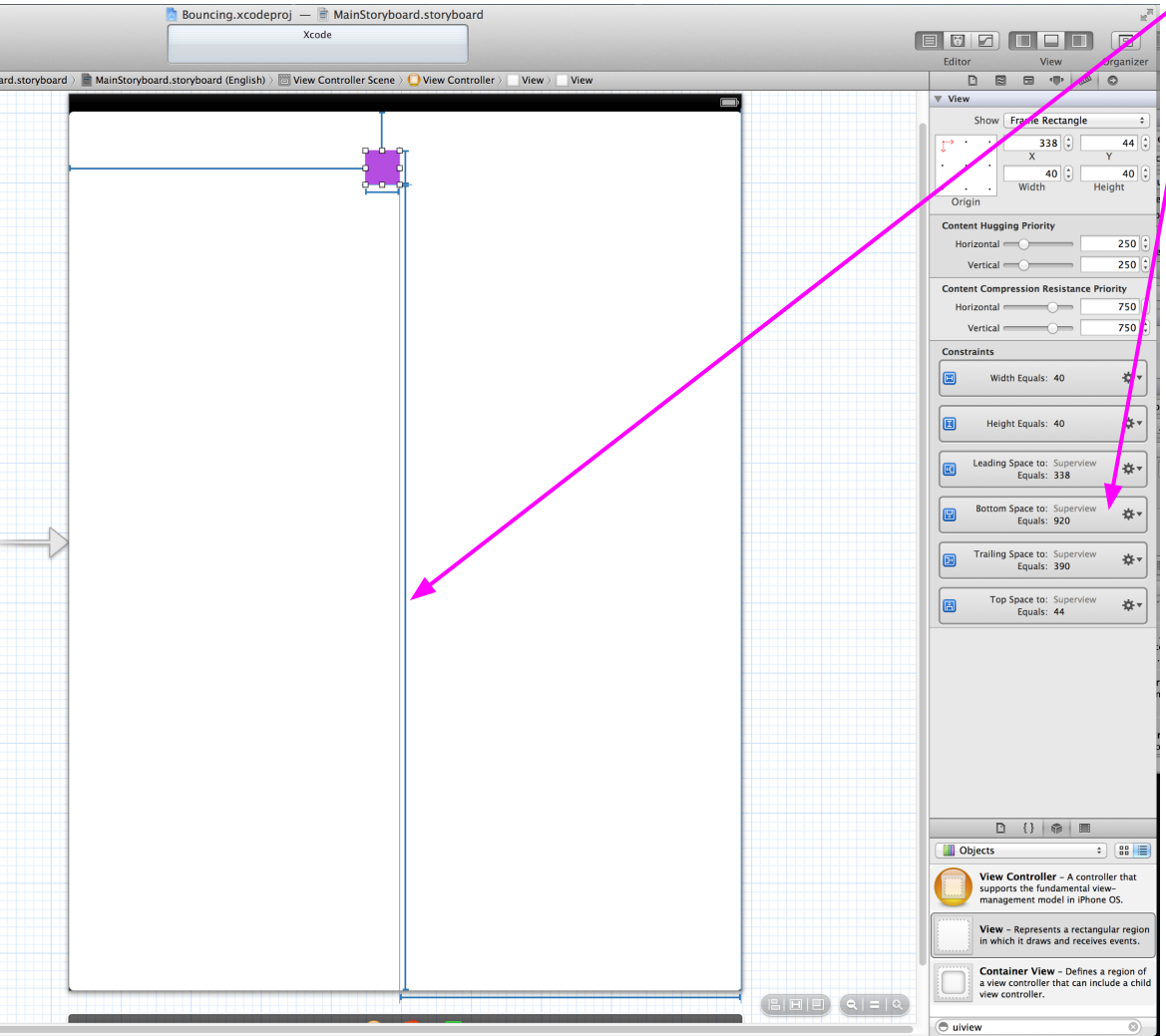
More constraints...



3) There's another x-position constraint, but for the opposite edge of the view (a.k.a. **Trailing Space to Superview** constraint).

This is the amount of space between the right edge of the screen and the right edge of the ball view.

More constraints...



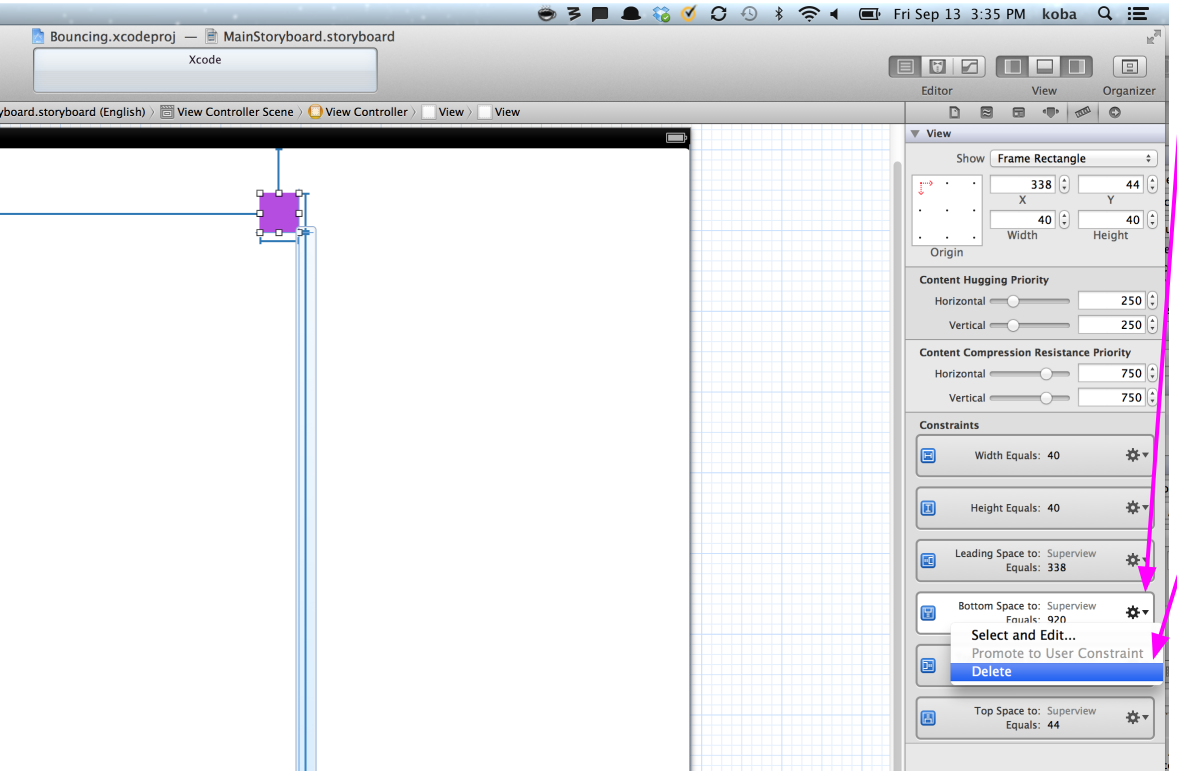
4) There's another y-position constraint, but for the opposite edge of the view (a.k.a. **Bottom Space to Superview** constraint).

This is the amount of space between the bottom edge of the screen and the bottom edge of the ball view.

That's all four.

But, we only need two of those, in addition to the **Height** and **Width** constraints.

Delete the Bottom Space constraint.

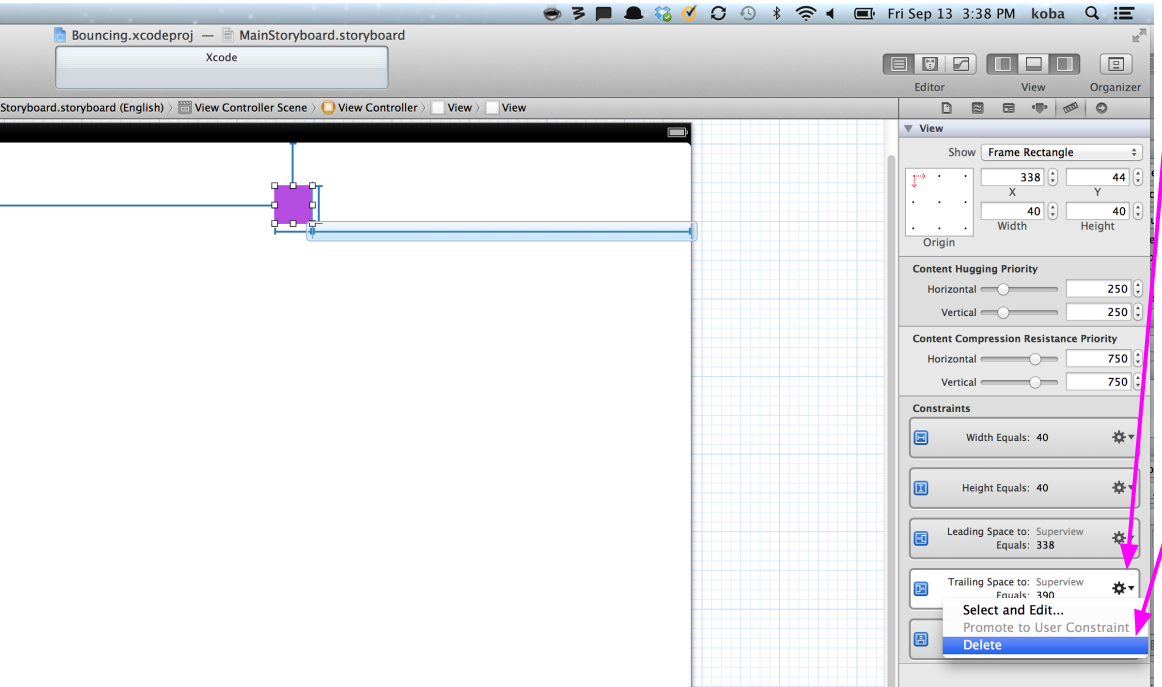


See that drop-down list button that's on the right-side of the **Bottom Space** constraint?

We don't want the **Bottom Space** constraint. So, select the **Delete** option.

We don't need it, because it is redundant. We already have a **Top Space** constraint and a **Height** constraint. That is enough to position the ball view vertically in the screen.

Delete the Trailing Space constraint.



See that drop-down list button that's on the right-side of the **Trailing Space** constraint?

We don't want the **Trailing Space** constraint. So, select the **Delete** option.

We don't need it, because it is redundant. We already have a **Leading Space** constraint and a **Width** constraint. That is enough to position the ball view horizontally in the screen.

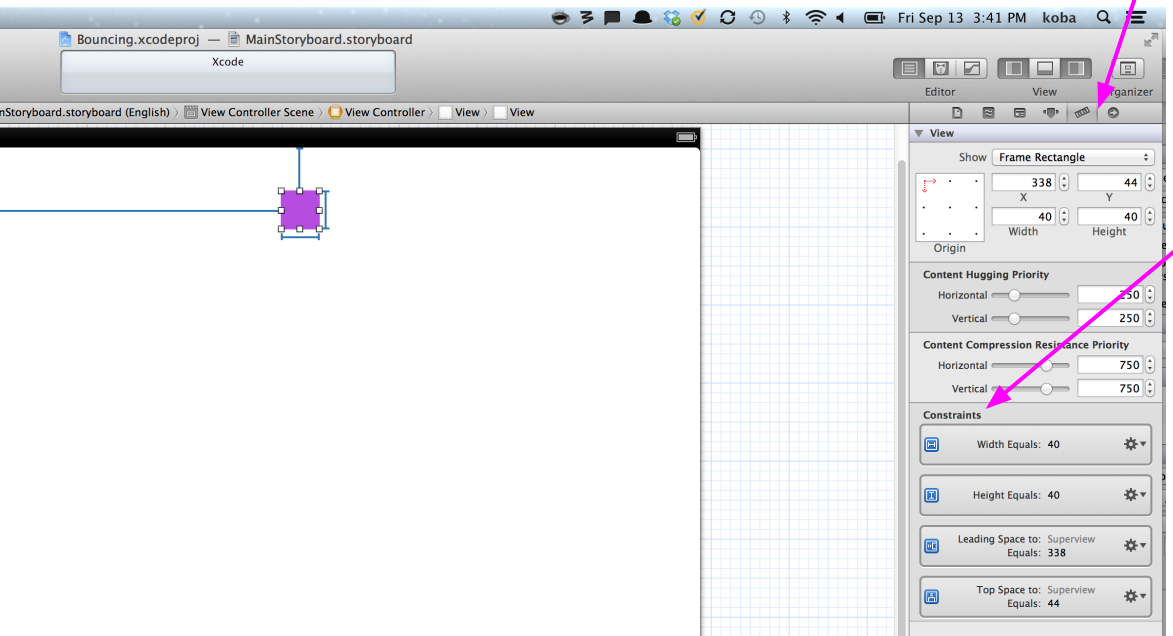
Only four constraints left.

Look at the blue constraints surrounding the ball view. There are four left.

You can also look at the **Size inspector**...

and see that there are four **Constraints** listed:

- Width
- Height
- Leading Space
- Top Space

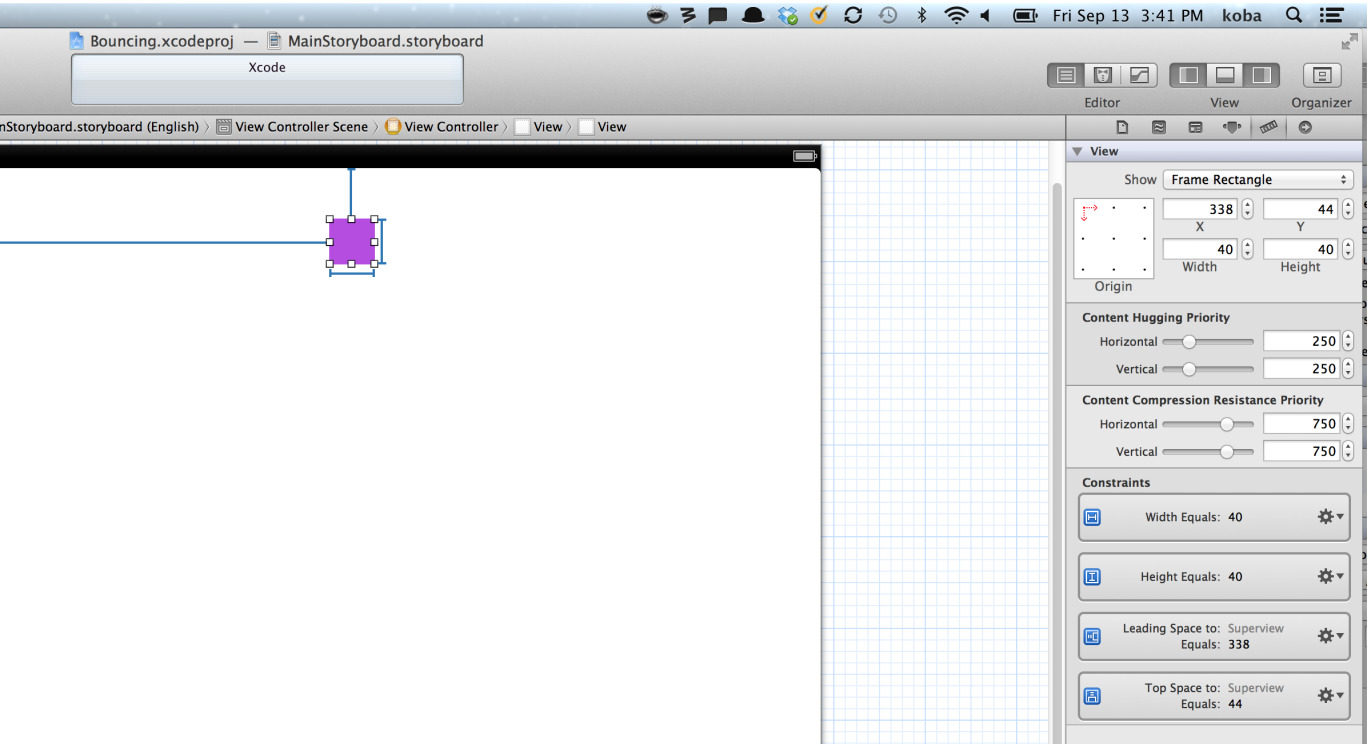


What for?

So, why do we care? Well, we're going to use the **Leading Space** (x-position) constraint and the **Top Space** (y-position) constraint to move the ball around on the screen.

If we increase the value of the **Leading Space** constraint, we move the ball to the right. If we decrease the value, it moves to the left.

If we increase the value of the **Top Space** constraint, we move the ball **DOWN**. If we decrease the value, it moves up.



It may be helpful to think of these constraints as the distance between the edge of the ball view and the screen edge.

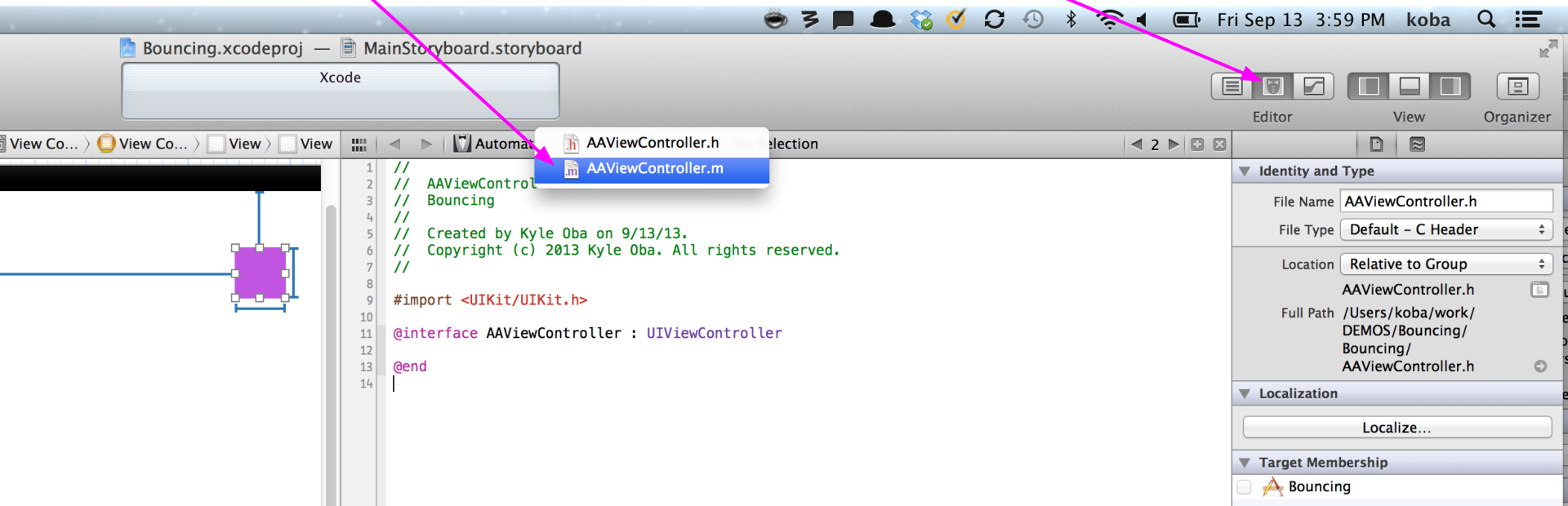
By changing the value of these constraints, we can change the position of the ball view.

Resurrect the Assistant Editor

We're going to connect that ball view now. So we need to see both the ball view and your implementation (.m) file at the same time.

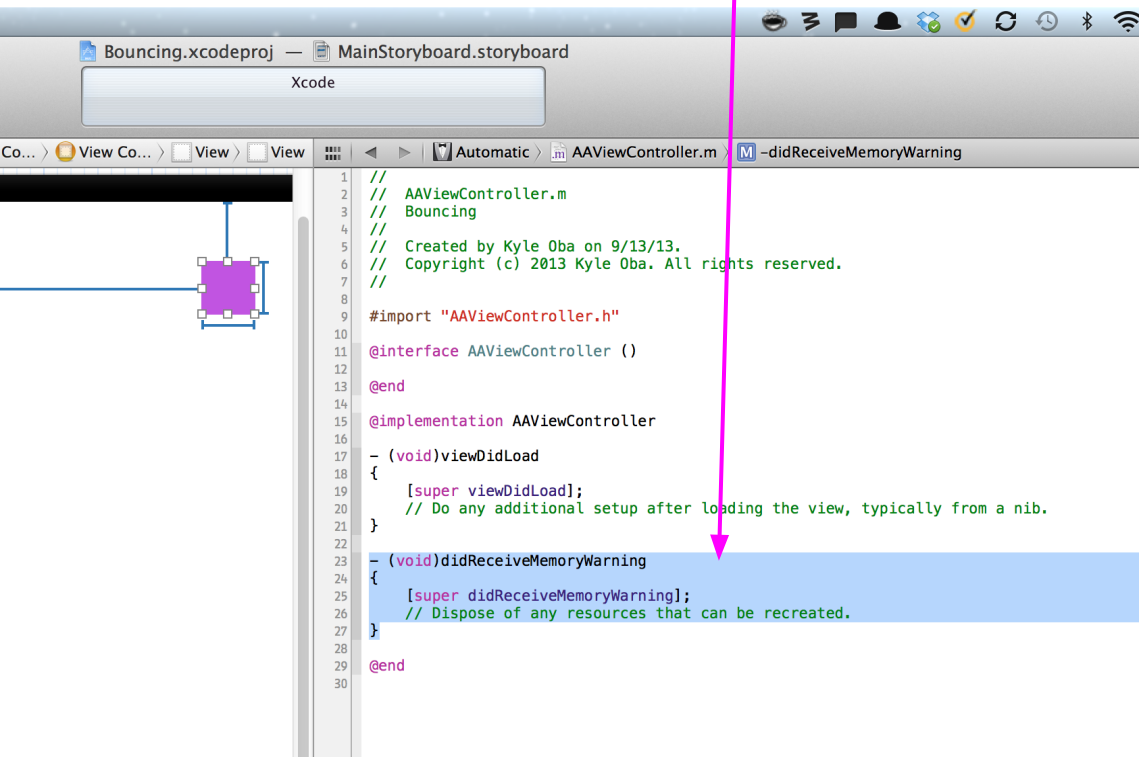
Open the **Assistant editor**.

Click on the **Assistant editor's** top menu to select your implementation (.m) file.



Remove junk.

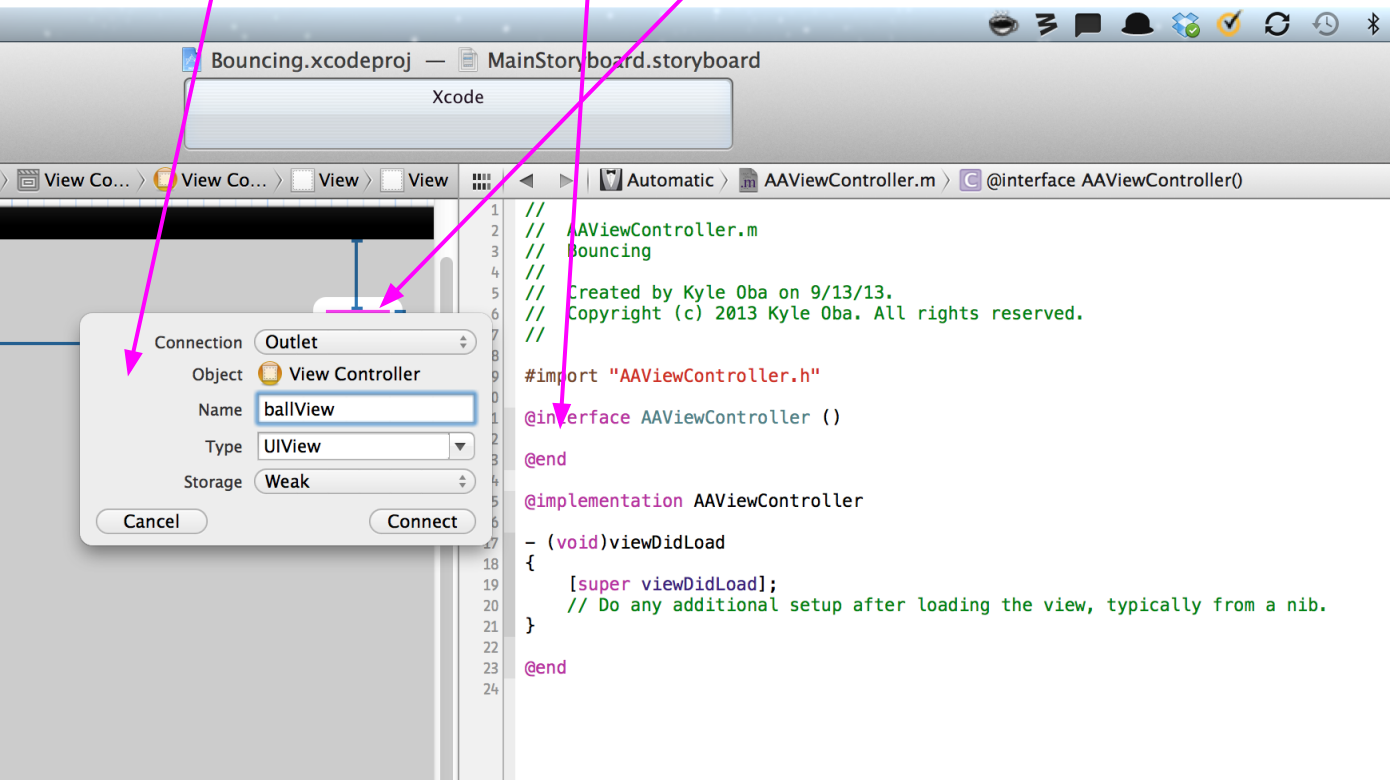
The default code in your view controller is nice. But, some of it is unnecessary. Delete the method here in blue.



Connect the ball view via Outlet.

Hold down the **control** key and click and drag from your ball view (just barely visible in this image) to the **@interface** section of your implementation (.m) file.

This menu should pop up.

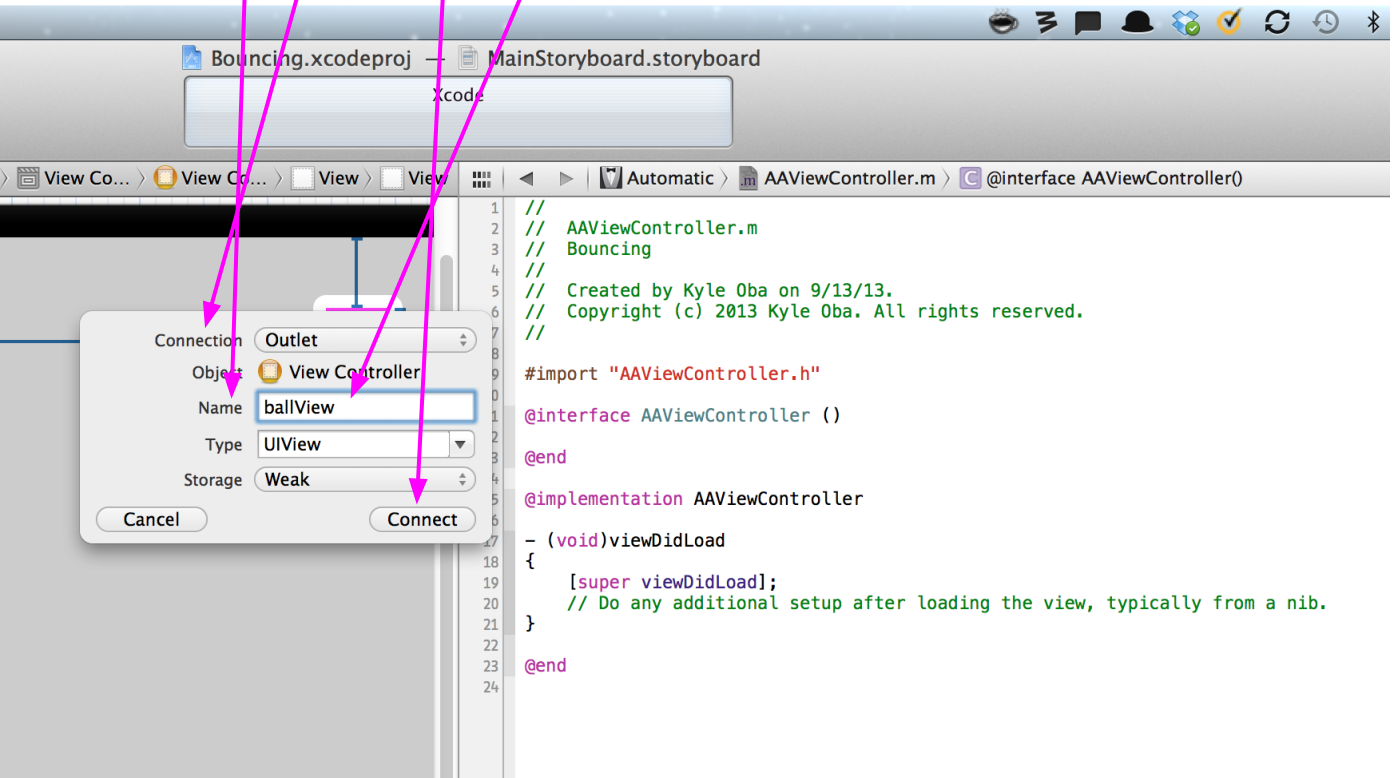


Configure the Outlet.

Make sure your **Connection** field is set to **Outlet**.

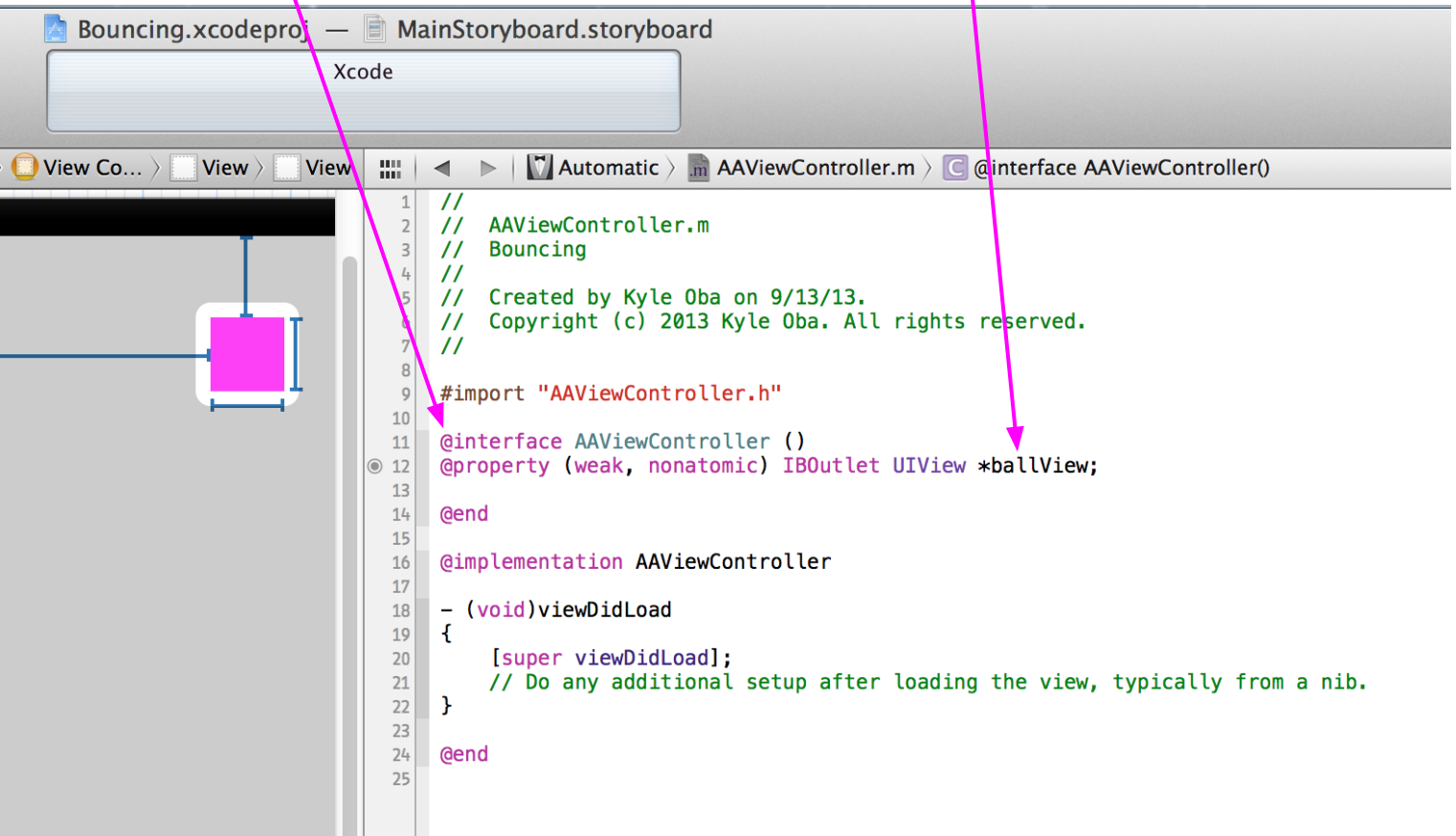
In the **Name** field, type in the name **ballView**.

When ready, click the **Connect** button to create the **Outlet**.



Your Outlet.

After you press the Connect button, Xcode should insert a new property in the **@interface** section of your implementation file.



Frames and Animation

Flipbooks. We're going to build something like a flipbook, except we're drawing our pages (frames) on an iPad screen, instead of on paper.

Also, the iPad handles flipping the pages at 30 frames per second.

Thanks iPad.

Refresh your memory about how flipbooks works by checking out the Vimeo "flipbook" video channel:

http://vimeo.com/channels/flipbooks

We need a frame timer.

There this thing called a **CADisplayLink**, that iOS provides for us. It is used to notify us when a new frame has been created.

We'll use this like a frame timer. Every time the **CADisplayLink** tells us that there's a new frame, we'll move the ball.

The iPad will draw the ball in the new location. It will appear to move. Animation!

But first, we need to create this **CADisplayLink** frame timer.

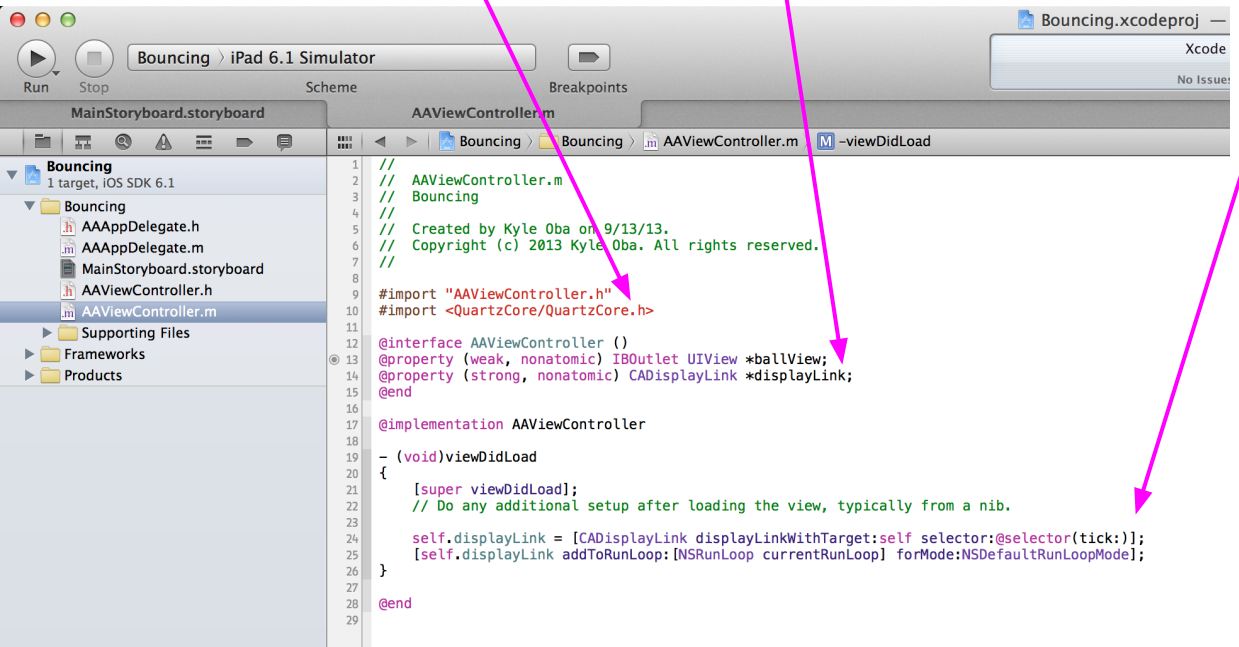
Let’s add a frame timer.

We need to do a few things before we can start working with the frame timer.

This frame timer is part of a framework that iOS provides called, “QuartzCore.” QuartzCore isn’t available to you by default.

We have to *import* it so that Xcode knows you want to use it. Copy the import statement into your code.

After you’ve imported QuartzCore, you can create a new property to hold your **CADisplayLink** frame timer. Copy this **displayLink** property declaration.



Now that you have a **displayLink** property. You have to create the displayLink and start it.

To create it and start it, copy these two lines of code here.

The first line creates it.

The second line starts it.

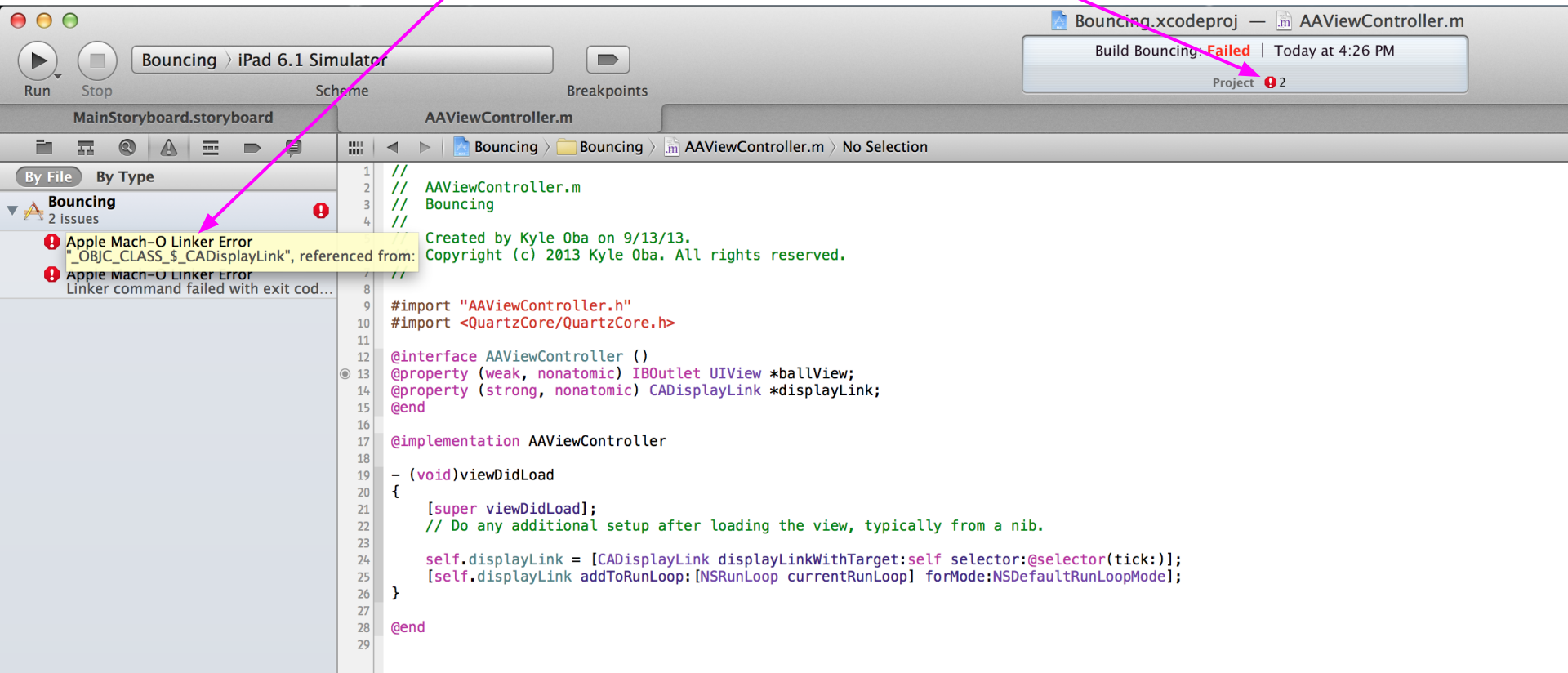
FAIL

Try to run the app now.

It should explode with errors.

Let's take a look at those errors now. See the red exclamation point? That says that you have two errors. Click on it to pull up the error list.

The error list shows a crazy error. It basically means that has no idea what the **CADisplayLink** is.

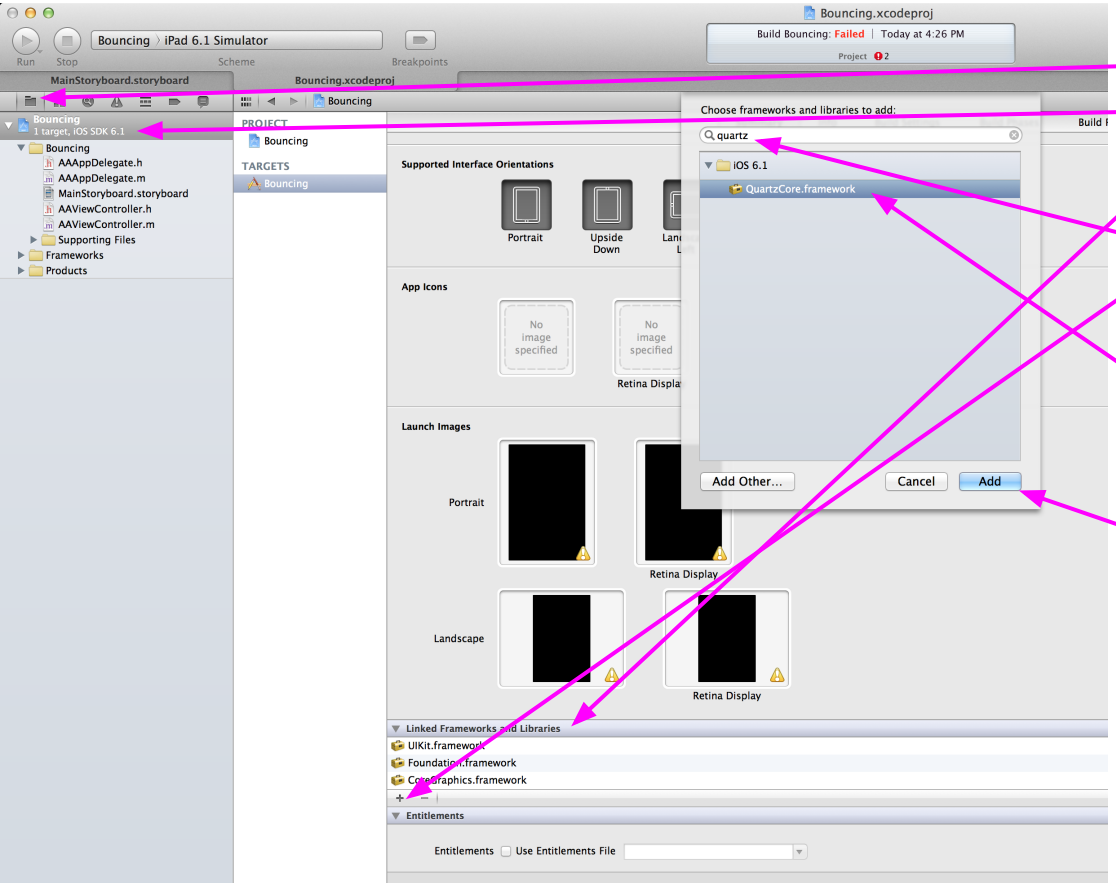


Add the QuartzCore framework.

You're probably wondering why Xcode doesn't know about something that iOS provides.

Well, since QuartzCore isn't available by default. You have to add it to your project. I know this seems like extra work. But, you really don't want to add it unless you're *really* going to use it. Something to do with efficiency.

We're *really* going to use it.



Select the **Project navigator**.

Select your project (click on it).

Scroll down a bit until you see the **Linked Frameworks and Libraries** section.

Click the **+** button to add a new framework.

A sheet will drop down from the top. In the search box, type in the search term: **quartz**

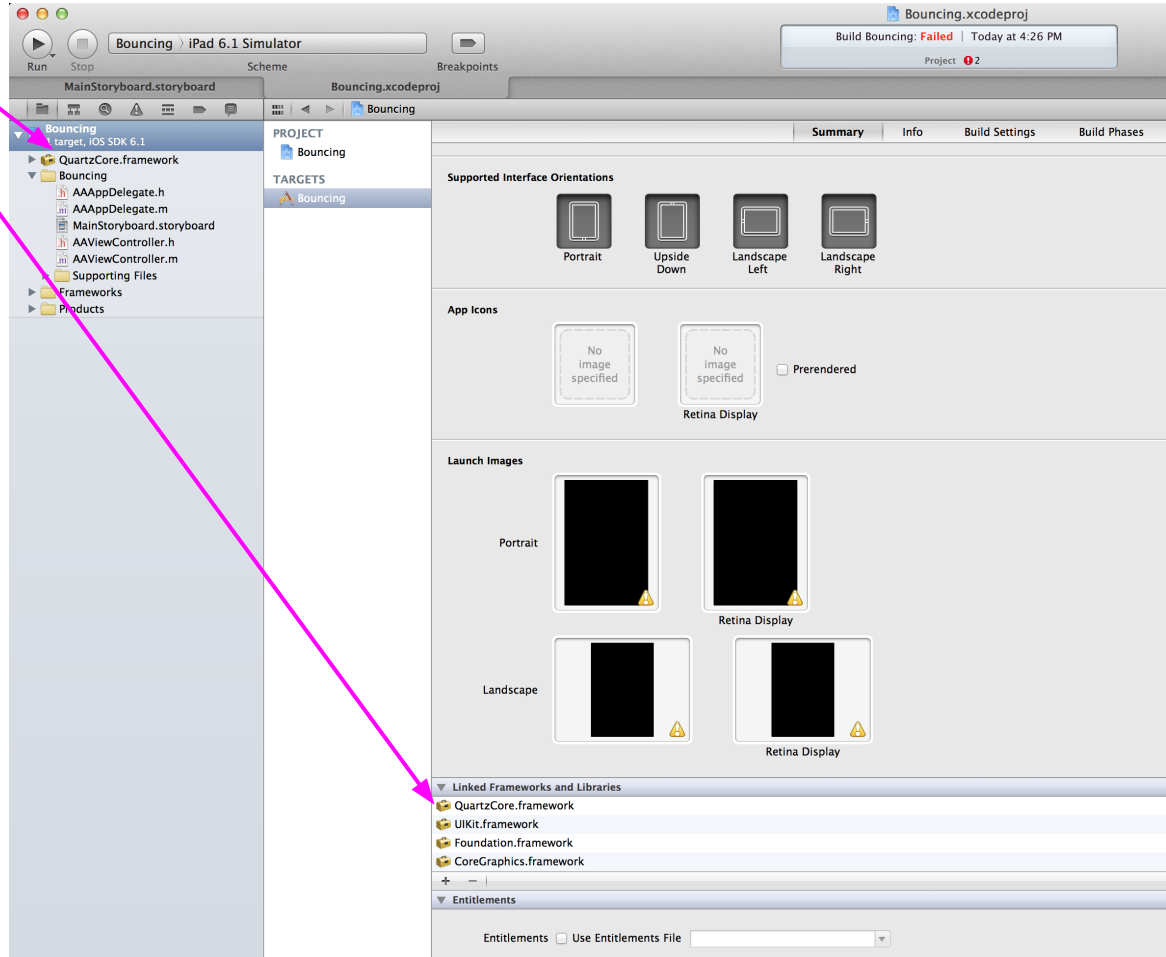
You should see the **QuartzCore.framework** item. Select it (click on it).

Finally, click the **Add** button to add the **QuartzCore** framework.

It should be there.

You should now see the QuartzCore framework in your project.

It should be listed in two locations.



DOUBLE FAIL

Now, **Run** the app again.

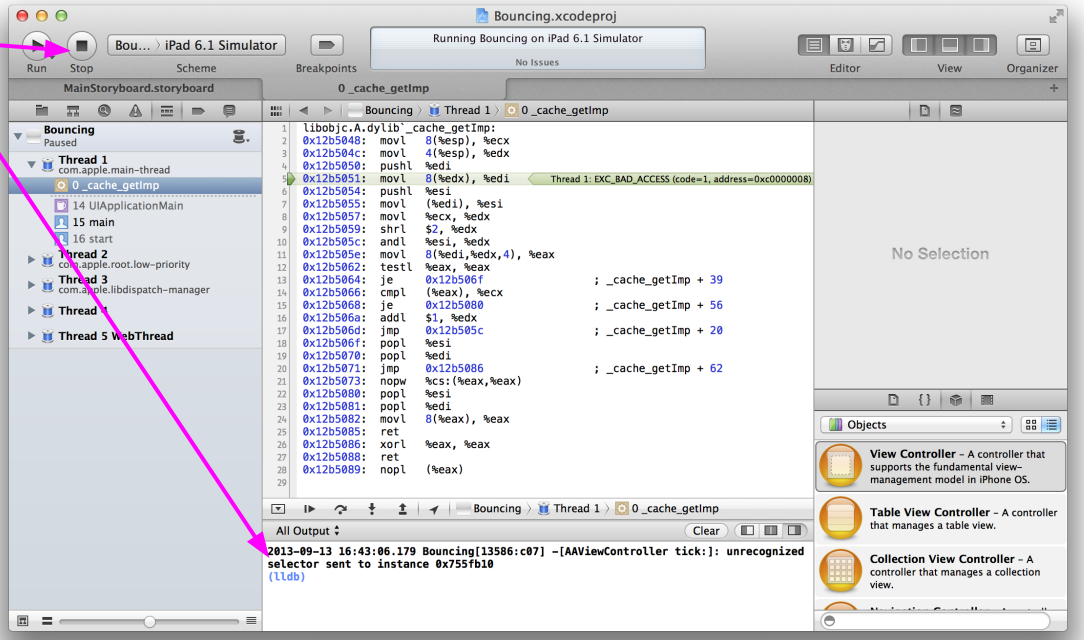
It fails even more spectacularly now.

Look at all the craziness below. Don't try to make sense of it. Also, your spectacular failure may look different than mine.

The point of me showing you is this: Things go wrong. You'll see failure messages like this from time to time. Don't be afraid.

Check out this tiny little message here. It's telling us that the **tick** method is missing.

Press the **Stop** button, to return to normal.

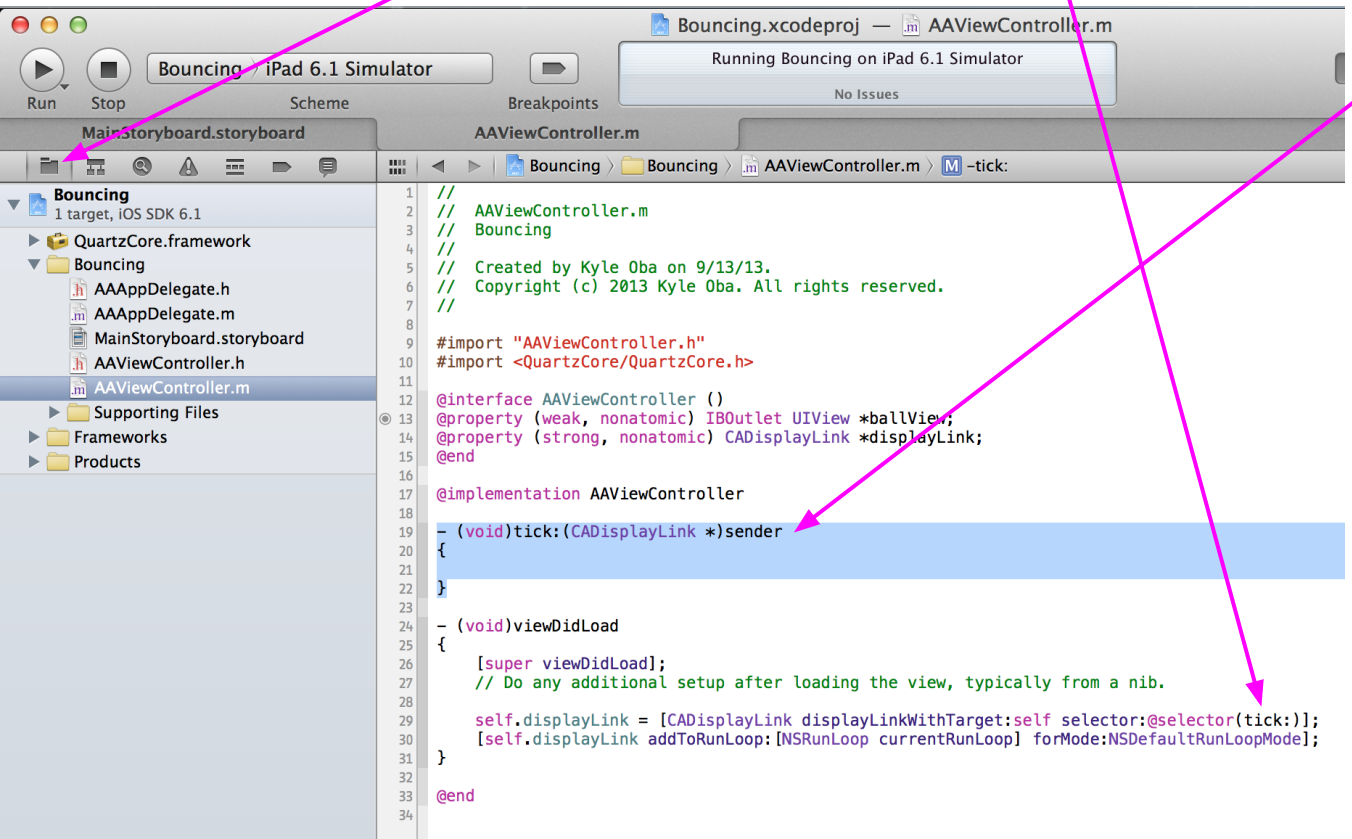


Add a tick method.

What is the **tick** method? It's a bit of code (a function) that is executed whenever a new frame is created.

You need one, so that the **displayLink** frame timer can find it. This is where we told the **displayLink** what method to call (**tick**) on each frame.

Select the **Project navigator**.



In your implementation (.m) file, add the **tick** method.

Leave it empty, with nothing between the curly braces, for now… just as you see it here.

Run your app now. The **Simulator** should show your ball. It should just sit there and do nothing.

B O R I N G

What now?

Okay. So, we have this **displayLink** frame timer thing. It's calling our **tick** method. But, our **tick** method is empty.

So, what do we do now?

Well, we want our ball view to move around on the screen. We need a way to change the position of the ball view. Then, we can make a small change to the position of the ball view every time we have a new frame.

Using this technique, we can animate the ball view. We can make it appear to move around on the screen.

We know from our experience with flipbooks that we're just drawing the ball in a new position and showing *a bunch* of frames in rapid succession.

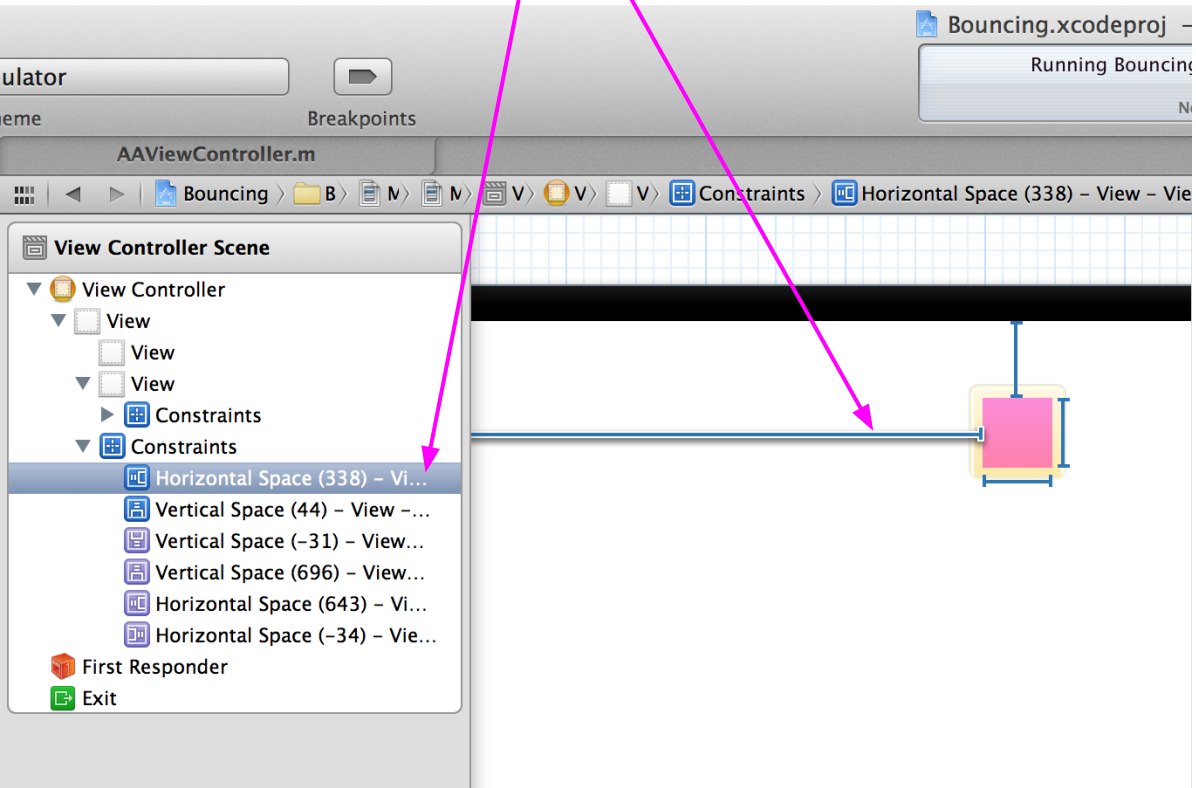
So, how do we change the position of the ball view.

We'll use the X & Y constraints.

Remember the X & Y constraints? We need to create **Outlets** to each of those, so we can change them from the code.

It's a bit tricky. But, click on the x constraint (a.k.a. **Leading Space** constraint) to select it.

Not how this **Horizontal Space** constraint is highlighted in the left pane.



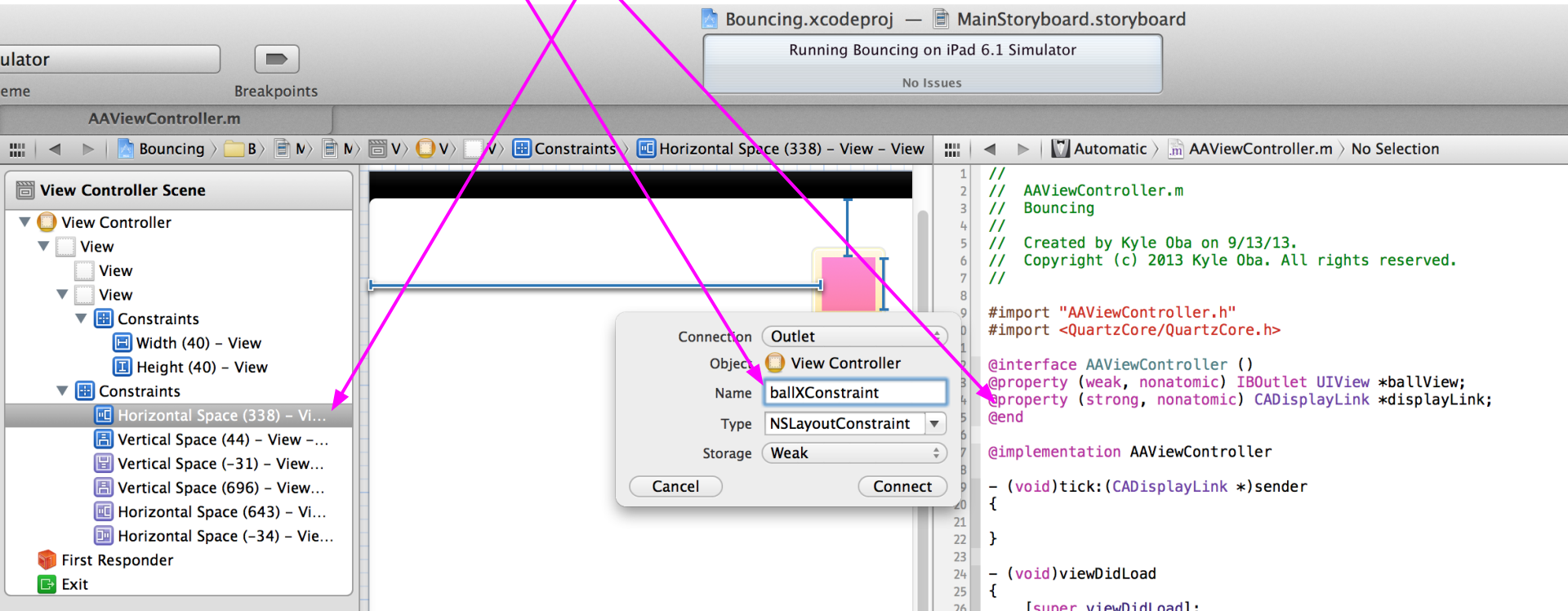
X constraint first.

Holding down the **control** key, click and drag from the **Horizontal Space** constraint item and release in the **@interface** section of your implementation (.m) file.

You can place it under your last property.

You'll see the **Outlet** creation pop-up window again.

In the **Name** field, enter the **ballXConstraint**. Then, press the **Connect** button.



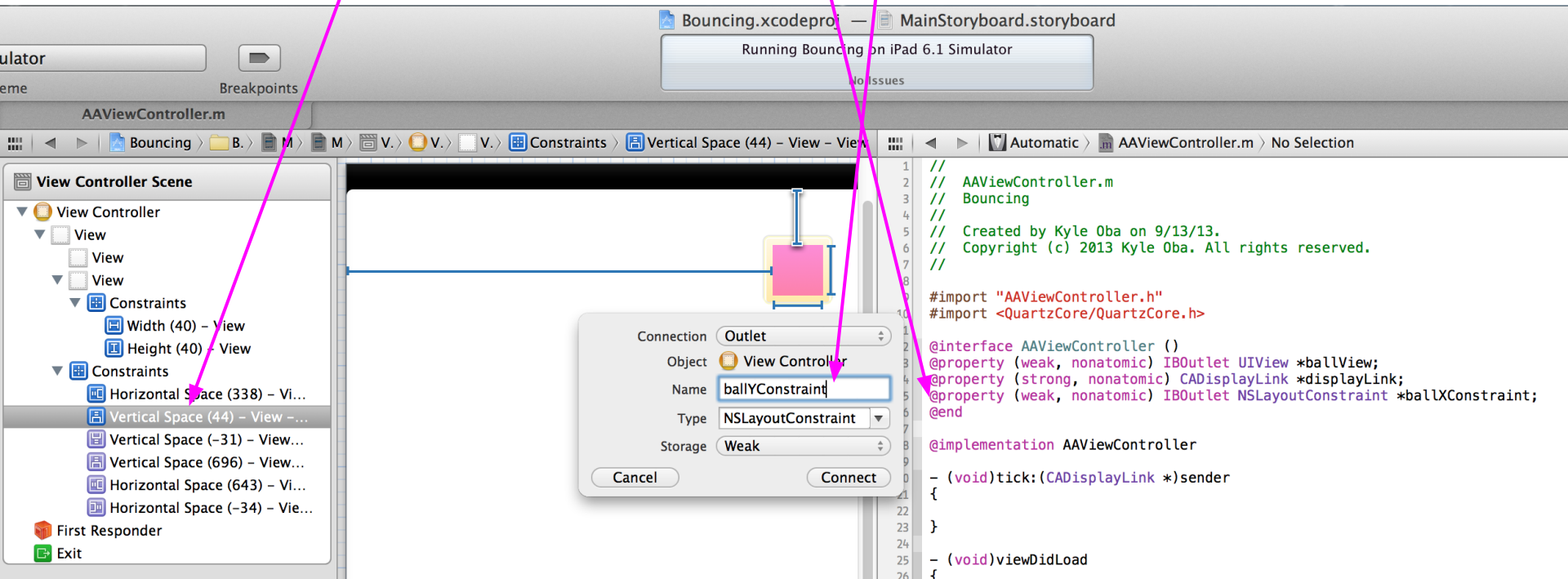
Now the Y constraint.

Do the same thing for the **Vertical Space** constraint.

Hold down the control key, then click and drag into the **@interface** section.

But, in the **Outlet** creation pop-up window, name this constraint **ballYConstraint**.

Press the **Connect** button



Outlets Complete.

You have all your **Outlets** set up now.

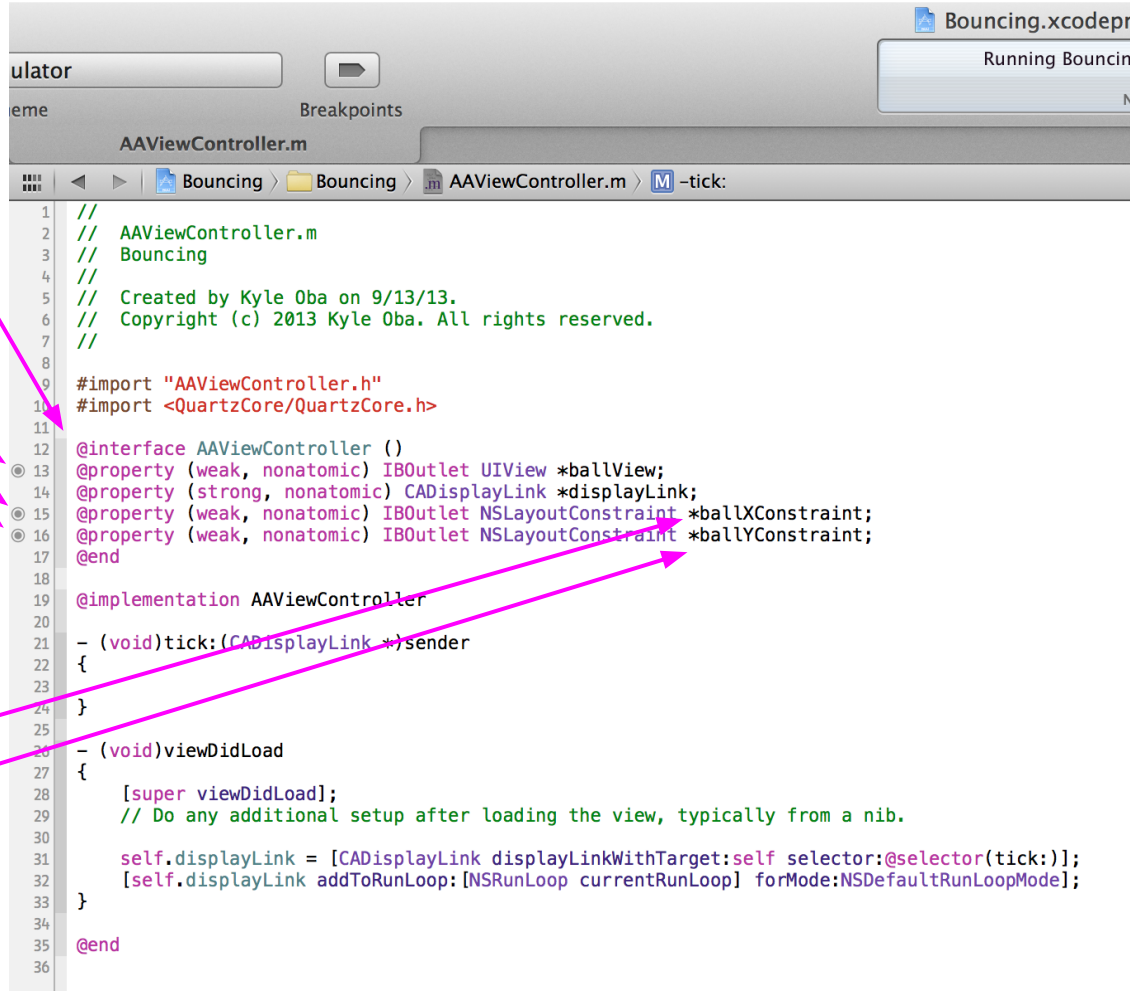
Take a look at the properties here.

The **Outlets** have a dot next to them.

There are two layout constraints:

one for the X position (horizontal), and

one for the Y position (vertical).



Enough chit-chat. Move ball now.

Add this code here to the **tick** method. Make it look *exactly* the same.

The first part creates a **CGPoint** variable, named **pos**. A **CGPoint** is just a special type of variable that is used to store a point. A point stores an X value and a Y value. Think of it like a special bucket for carrying a single X and a single Y value around.

We create our **pos** variable here by creating a **CGPoint** with the X value equal to the X constraint's value (a.k.a. **constant**).

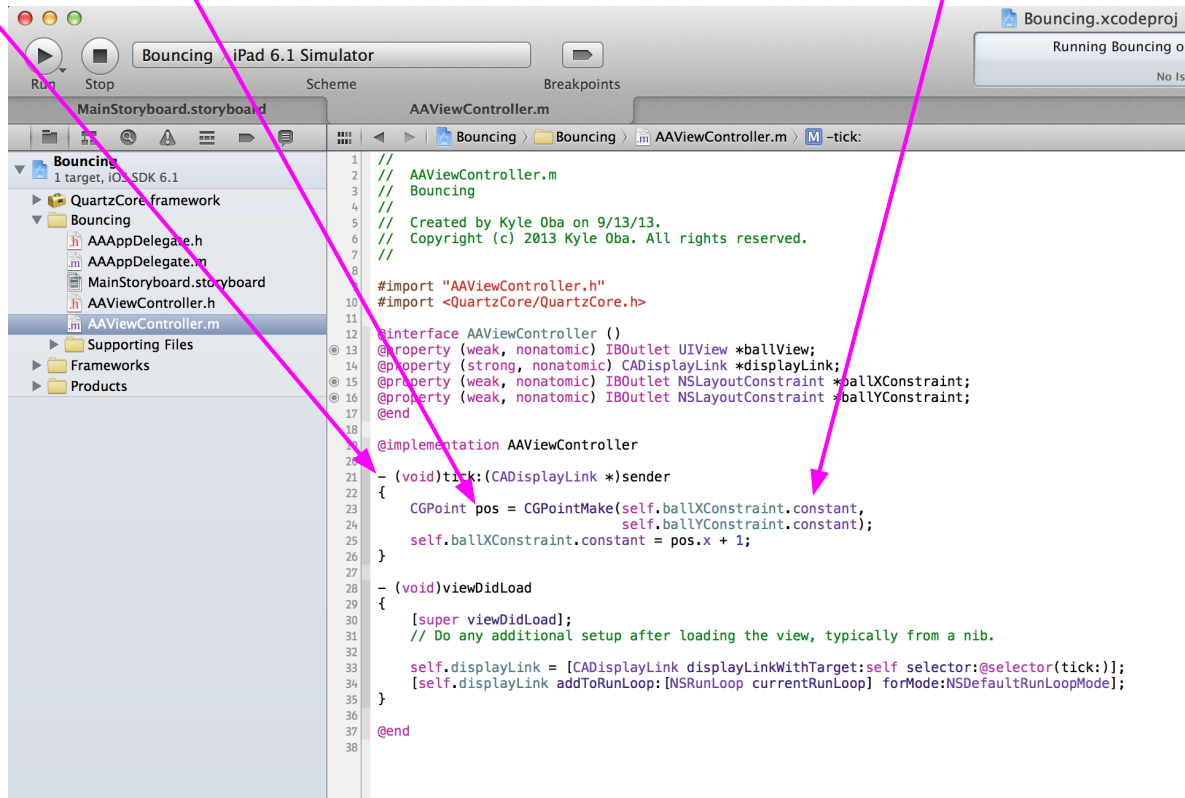
Remember that the X constraint stores the distance between the left edge of the iPad's screen and the left edge of the ball view.

The Y value of **pos** is set to the value of our Y constraint.

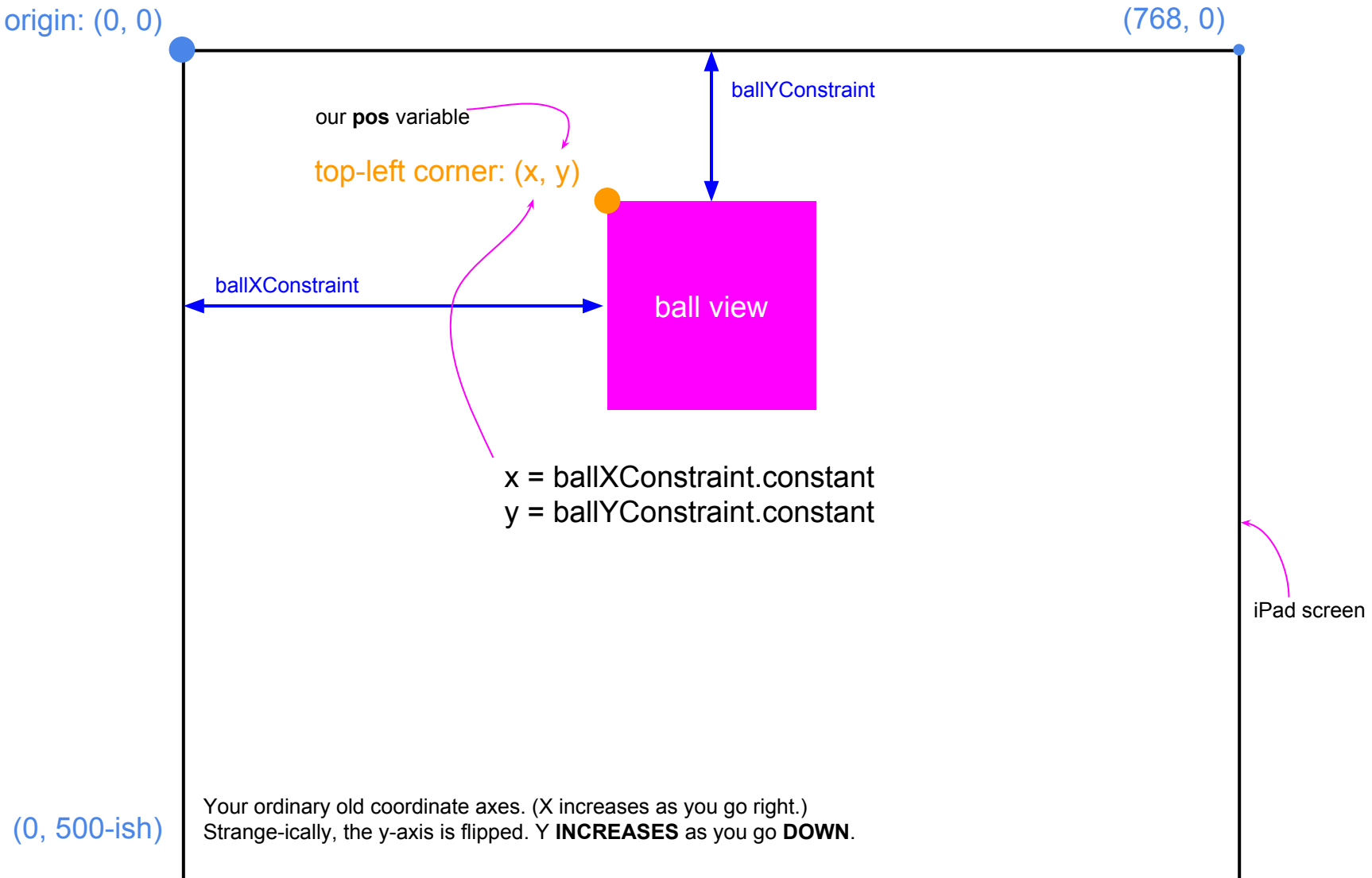
Remember that the Y constraint holds the distance between the top edge of the screen and the top edge of the ball view.

If you hold both index fingers to opposite sides of your head, you can probably figure out that this **pos** variable holds the location of the top-left point of the ball view.

Here's a diagram to help you make sense of that last point...



X & Y constraint values



Sorry

Too much graphing. Too much math.

I promise there’s a point to all this.

Back to bouncing the ball...

Take another look at the tick method.

Now that you know what the **pos** variable holds...

Let's move the ball to the right by one point per frame.

We do this by adding **1** to the current X position.

We then make this updated X position the new X constraint.

That should do it.

Run your app to see a moving all.