The Final App | iOS Apprentice Part 14

Ray Wenderlich para mim 17 de ago

José, welcome back to the raywenderlich.com **iOS Apprentice Email Course**!

You might be thinking, "OK, **Bull's Eye** is now done, and I can move on to the next app!" If you were, I'm afraid you are in for disappointment - there's just a teensy bit more to do in the game.

"What? What's left to do? We finished the task list!" you say? You are right. The game is indeed complete. However, all this time, you've been developing and testing for a 4" iPhone screen found on devices such as the iPhone 5, 5c, and SE. But what about other iPhones such as the 4.7-inch iPhone, the 5.5-inch iPhone Plus, or the 5.8-inch iPhone X which have bigger screens? Or the iPad with its multiple screen sizes? Will the game work correctly on all these different screen sizes?

And if not, shouldn't we fix it?

Today, we'll cover the following:

- **Support different screen sizes:** Ensure that the app will run correctly on all the different iPhone and iPad screen sizes.
- **Crossfade:** Add some animation to make the transition to the start of a new game a bit more dynamic.

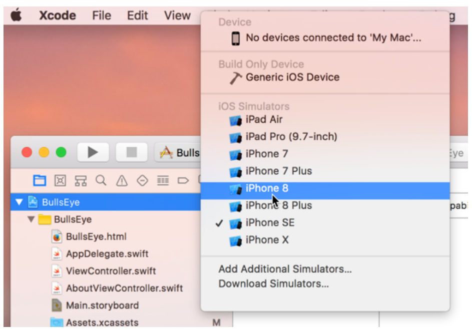
> **Note:** Prefer learning via video? You might like to check out videos #43-44 of the free video version of this course, which correspond to this email.

Support different screen sizes

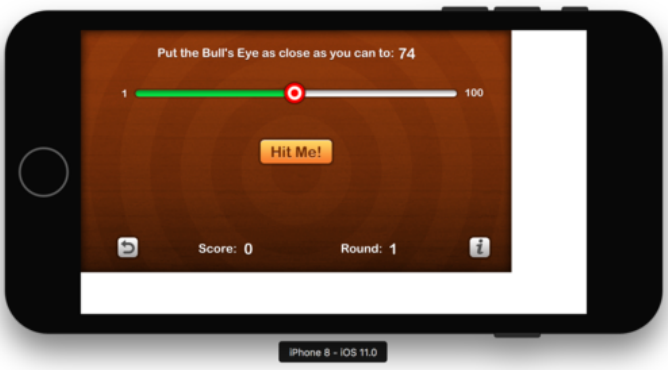
To get started, open up the Bull's Eye project where you left it off last time (or download the starter project from the corresponding forum discussion thread).

First, let's check if there is indeed an issue running Bull's Eye on a device with a larger screen. It's always good to verify that there's inded an issue before we do extra work, right? Why fix it, if it isn't broken? :]

➤ To see how the app looks on a larger screen, run the app on an iPhone simulator like the **iPhone 8**. You can switch between Simulators using the selector at the top of the Xcode window:



The result might not be what you expected:



Obviously, this won't do. Not everybody is going to be using a 4" iOS device. And you don't want the game to display on only part of the screen for the rest of the people!

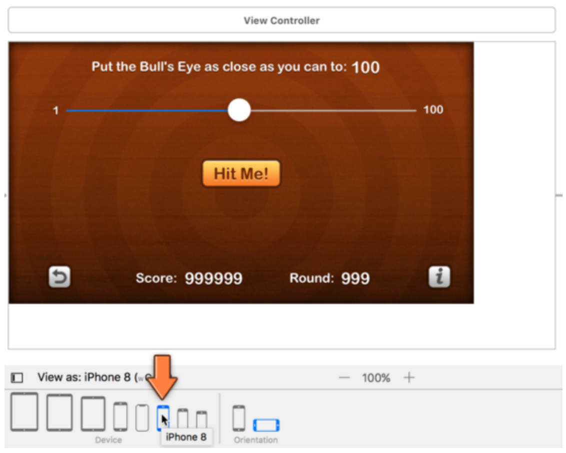
This is a good opportunity to learn about **Auto Layout**, a core UIKit technology that makes it easy to support many different screen sizes in your apps, including the larger screens of the 4.7-inch, 5.5-inch, and 5.8-inch iPhones, and the iPad.

Tip: You can use the **Window** → **Scale** menu to resize a simulator if it doesn't fit on your screen. Some of those simulators, like the iPad one, can be monsters! Also, with Xcode 9 onwards, you can resize a simulator window by simply dragging on one corner of the window - just like you do to resize any other window on macOS.

Interface Builder has a few handy tools to help you make the game fit on any screen.

The background image

➤ Go to **Main.storyboard**. Open the **View as:** panel at the bottom and choose the **iPhone 8** device. (You may need to change the orientation back to landscape.)

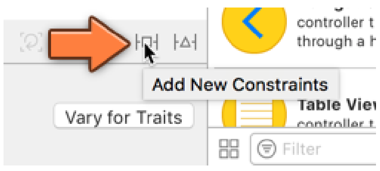


The storyboard should look like your screen from when you ran on the iPhone 8 Simulator. This shows you how changes on the storyboard affect the bigger iPhone screens.

First, let's fix the background image. At its normal size, the image is too small to fit on the larger screens.

This is where Auto Layout comes to the rescue.

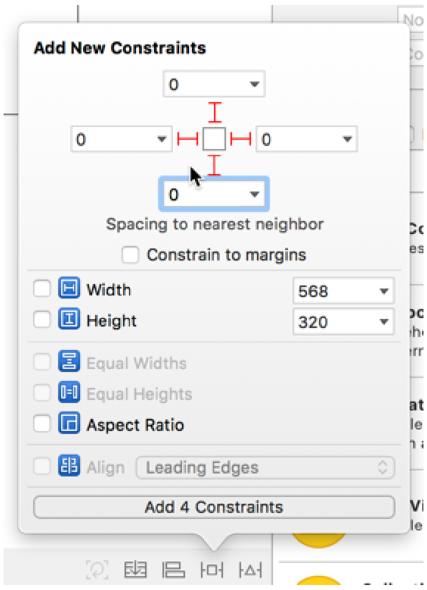
➤ In the storyboard, select the **Background image view** on the main **View Controller** and click the small **Add New Constraints** button at the bottom of the Xcode window:



This button lets you define relationships, called **constraints**, between the currently selected view and other views in the scene. When you run the app, UIKit evaluates these constraints and calculates the final layout of the views. This probably sounds a bit abstract, but you'll see soon enough how it works in practice.

In order for the background image to stretch from edge-to-edge on the screen, the left, top, right, and bottom edges of the image should be flush against the screen edges. The way to do this with Auto Layout is to create two alignment constraints, one horizontal and one vertical.

➤ In the **Add New Constraints** menu, set the **left**, **top**, **right**, and **bottom** spacing to zero and make sure that the red I-beam markers next to (or below) each item is enabled. (The red I-beams are used to specify which constraints are enabled when adding new constraints.):


Add New Constraints

0

0 0

0

Spacing to nearest neighbor

☐ Constrain to margins

☐ Width 568

☐ Height 320

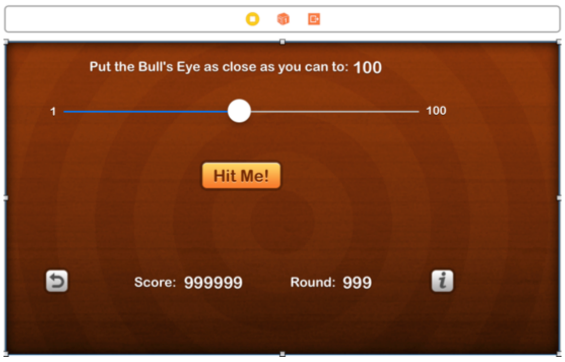
☐ Equal Widths

☐ Equal Heights

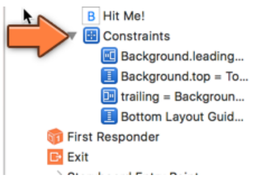
☐ Aspect Ratio

☐ Align Leading Edges

➤ Press **Add 4 Constraints** to finish. The background image will now cover the view fully. (Press Undo and Redo a few times to see the difference.)



You might have also noticed that the Document Outline now has a new item called **Constraints**:



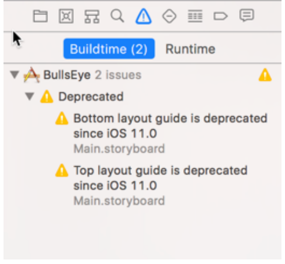
There should be four constraints listed there, one for each edge of the image.

➤ Run the app again on the iPhone 8 Simulator and also on the iPhone SE Simulator. In both cases, the background should display correctly now. (Of course, the other controls are still off-center, but we'll fix that soon.)

If you use the **View as:** panel to switch the storyboard back to the iPhone SE, the background should display correctly there too.

Compiler warnings

When you run the app after adding your first autolayout constraints, sometimes you might see some compiler warnings similar to this:

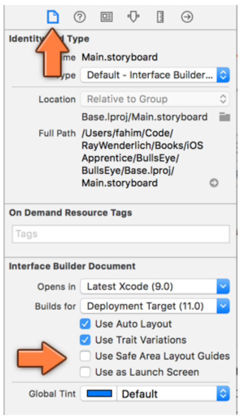

Buildtime (2) Runtime
BullsEye 2 issues
Deprecated
Bottom layout guide is deprecated since iOS 11.0
Main.storyboard
Top layout guide is deprecated since iOS 11.0
Main.storyboard

If this happens to you, that is because in iOS 11 there were some changes to how autolayout works and how constraints are set up. In previous versions of iOS, you had markers called **top layout guide** and **bottom layout guide** which defined the usable area of a screen. These guides were useful in setting your own views to stretch to the top edge (or the bottom of edge) of the screen without covering any on-screen elements provided by the OS such as navigation bars or tab bars.

However, in iOS 11, they introduced a new layout mechanism which was more flexible than the previously used top and bottom layout guides. These new layout guides are known as the **safe area layout guides**.

So how do you use these new safe area layout guides, you ask? Simple enough, you just have to enable them for your storyboard :]

Switch to your **storyboard**, select your view controller, and then on the right-hand pane, go to the **File Inspector**.


Identity and Type
Name Main.storyboard
Type Default - Interface Builder...
Location Relative to Group
Base.lproj/Main.storyboard
Full Path /Users/fahim/Code/RayWenderlich/Books/iOS Apprentice/BullsEye/BullsEye/Base.lproj/Main.storyboard
On Demand Resource Tags
Tags
Interface Builder Document
Opens in Latest Xcode (9.0)
Builds for Deployment Target (11.0)
[✓] Use Auto Layout
[✓] Use Trait Variations
[ ] Use Safe Area Layout Guides
[ ] Use as Launch Screen
Global Tint Default

Under the **Interface Builder Document** section, there should be a checkbox for **Use Safe Area Layout Guides** - check it. That's it, you are now using safe area layout guides in your storyboard and the compiler warnings should go away!

If you do not see the **Use Safe Area Layout Guides** checkbox, make sure that you have the view controller selected - that particular option appears only when you have a view controller selected.

The About screen

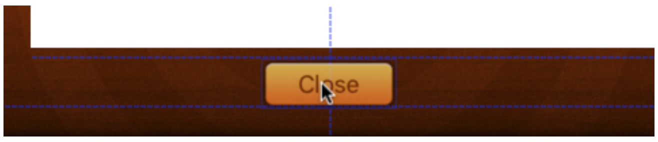
Let's repeat the background image fix for the About screen, too.

➤ Use the **Add New Constraints** button to pin the About screen's background image view to the parent view.

The background image is now fine. Of course, the Close button and web view are still completely off.

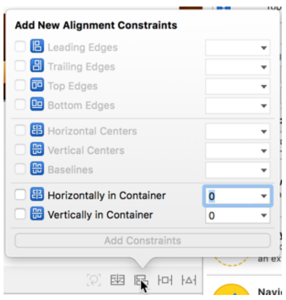
➤ In the storyboard, drag the **Close** button so that it snaps to the center of the view as well as the bottom guide.

Interface Builder shows a handy guide, the dotted blue line, near the edges of the screen, which is useful for aligning objects by hand.

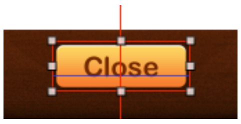


You want to create a centering constraint that keeps the Close button in the middle of the screen, regardless of how wide the screen is.

➤ Click the **Close** button to select it. From the **Align** menu (which is to the left of the Add New Constraints button), choose **Horizontally in Container** and click **Add 1 Constraint**.



Interface Builder now draws a red bar to represent the constraint, and a red box around the button as well.



That's a problem: the bars are all supposed to be blue, not red. Red indicates that something is wrong with the constraints, usually that there aren't enough of them.

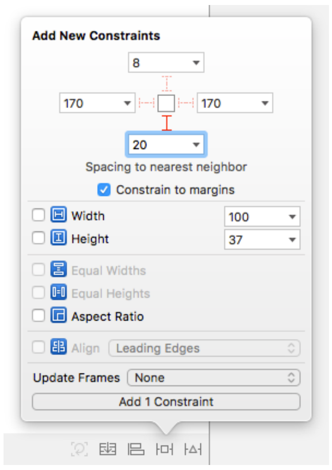
The thing to remember is this: for each view, there must always be enough constraints to define both its position and its size. The Close button already knows its size – you typed this into the Size inspector earlier – but for its position there is only a constraint for the X-coordinate (the alignment in the horizontal direction). You also need to add a constraint for the Y-coordinate.

As you've noticed, there are different types of constraints - there are alignment constraints and spacing constraints, like the ones you added via the Add New Constraints button.

➤ With the **Close** button still selected, click on the **Add New Constraints** button.

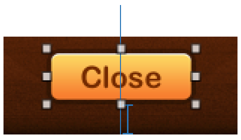
You want the Close button to always sit at a distance of 20 points from the bottom of the screen.

➤ In the **Add New Constraints** menu, in the **Spacing to nearest neighbor** section, set the bottom spacing to **20** and make sure that the I-beam above the text box is enabled.

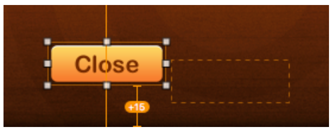


➤ Click **Add 1 Constraint** to finish.

The red constraints will now turn blue, meaning that everything is OK:

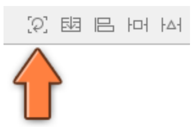


If at this point you don't see blue bars but orange ones, then something's still wrong with your Auto Layout constraints:



This happens when the constraints are valid (otherwise the bars would be red) but the view is not in the right place in the scene. The dashed orange box off to the side is where Auto Layout has calculated the view should be, based on the constraints you have given it.

To fix this issue, select the **Close** button again and click the **Update Frames** button at the bottom of the Interface Builder canvas.



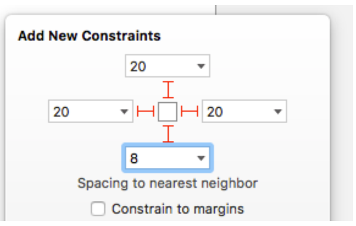
You can also use the **Editor** → **Resolve Auto Layout Issues** → **Update Frames** item from the menu bar.

The Close button should now always be perfectly centered, regardless of the device screen size.

Note: What happens if you don't add any constraints to your views? In that case, Xcode will automatically add constraints when it builds the app. That is why you didn't need to bother with any of this before.
However, these default constraints may not always do what you want. For example, they will not automatically resize your views to accommodate larger (or smaller) screens. If you want that to happen, then it's up to you to add your own constraints. (Afterall, Auto Layout can't read your mind!)
As soon as you add just one constraint to a view, Xcode will no longer add any other automatic constraints to that view. From then on you're responsible for adding enough constraints so that UIKit always knows what the position and size of the view will be.

There is one thing left to fix in the About screen and that is the web view.

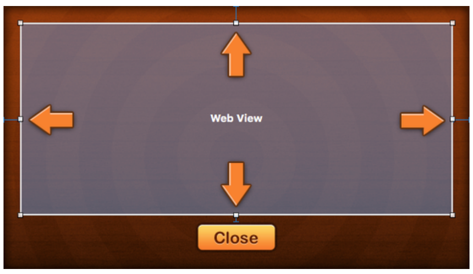
➤ Select the **Web View** and open the **Add New Constraints** menu. First, make sure **Constrain to margins** is unchecked. Then click all four I-beam icons so they become solid red and set their spacing to 20 points, except the bottom one which should be 8 points:



Add New Constraints
20
20
20
8
Spacing to nearest neighbor
☐ Constrain to margins

➤ Finish by clicking **Add 4 Constraints**.

There are now four constraints on the web view - indicated by the blue bars on each side:



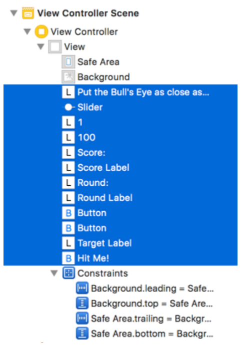
Three of these pin the web view to the main view, so that it always resizes along with it, and one connects it to the Close button. This is enough to determine the size and position of the web view in any scenario.

Fix the rest of the main scene

Back to the main game scene, which still needs some work.

The game looks a bit lopsided now on bigger screens. You will fix that by placing all the labels, buttons, and the slider into a new “container” view. Using Auto Layout, you’ll center that container view in the screen, regardless of how big the screen is.

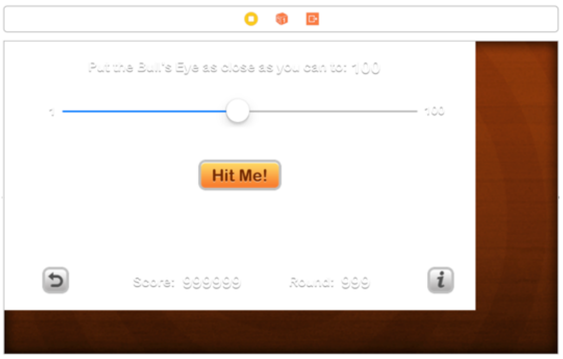
➤ Select all the labels, buttons, and the slider. You can hold down ⌘ and click them individually, but an easier method is to go to the **Document Outline**, click on the first view (for me that is the “Put the Bull’s Eye as close as you can to:” label), then hold down Shift and click on the last view (in my case the Hit Me! button):



```
▼ View Controller Scene
  ▼ View Controller
    ▼ View
        Safe Area
        Background
        Put the Bull's Eye as close as...
        Slider
        1
        100
        Score:
        Score Label
        Round:
        Round Label
        Button
        Button
        Target Label
        Hit Me!
      ▼ Constraints
          Background.leading = Safe...
          Background.top = Safe Are...
          Safe Area.trailing = Backgr...
          Safe Area.bottom = Backgr...
```

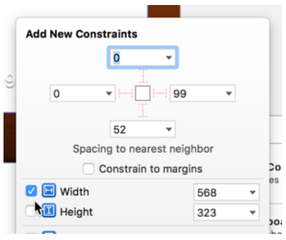
You should have selected everything but the background image view.

➤ From Xcode’s menu bar, choose **Editor** → **Embed In** → **View**. This places the selected views inside a new container view:



This new view is completely white, which is not what you want eventually, but it does make it easier to add the constraints.

➤ Select the newly added **container view** and open the **Add New Constraints** menu. Check the boxes for **Width** and **Height** in order to make constraints for them and leave the width and height at the values specified by Interface Builder. Click **Add 2 Constraints** to finish.



Add New Constraints

0

0 | 99

52

Spacing to nearest neighbor

☐ Constrain to margins

☑ Width 568

☐ Height 323

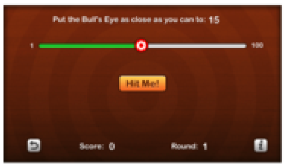
Interface Builder now draws several bars around the view that represent the Width and Height constraints that you just made, but they are red. Don’t panic! It only means there are not enough constraints yet. No problem, you’ll add the missing constraints next.

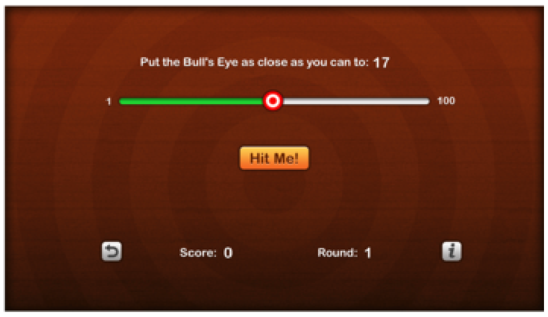
➤ With the container view still selected, open the **Align menu**. Check the **Horizontally in Container** and **Vertically in Container** options. Click **Add 2 Constraints**.

All the Auto Layout bars should be blue now and the view is perfectly centered.

➤ Finally, change the **Background** color of the container view to **Clear Color** (in other words, 100% transparent).

You now have a layout that works correctly on any iPhone display! Try it out:





Auto Layout may take a while to get used to. Adding constraints in order to position UI elements is a little less obvious than just dragging them into place.

But this also buys you a lot of power and flexibility, which you need when you're dealing with devices that have different screen sizes.

You'll learn more about Auto Layout in the other parts of **The iOS Apprentice**.

Challenge: As you try the game on different devices, you might notice something - the controls for the game are always centered on screen, but they do not take up the whole area of the screen on bigger devices! This is because you set the container view for the controls to be a specific size. If you want the controls to change position and size depending on how much screen space is available, then you have to remove the container view (or set it to resize depending on screen size) and then set up the necessary autolayout constraints for each control separately. Are you up to the challenge of doing this on your own?

Crossfade

There's one final bit of knowledge I want to impart before calling the game complete - Core Animation. This technology makes it very easy to create really sweet animations, with just a few lines of code, in your apps. Adding subtle animations (with emphasis on subtle!) can make your app a delight to use.

You will add a simple crossfade after the Start Over button is pressed, so the transition back to round one won't seem so abrupt.

➤ In **ViewController.swift**, add the following line at the top, right below the other import:

```
import QuartzCore
```

Core Animation lives in its own framework, QuartzCore. With the `import` statement you tell the compiler that you want to use the objects from this framework.

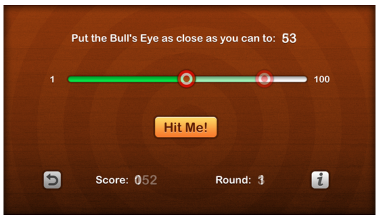
➤ Change `startNewGame()` to:

```
@IBAction func startNewGame() {
  ...
  startNewRound()
  // Add the following lines
  let transition = CATransition()
  transition.type = kCATransitionFade
  transition.duration = 1
  transition.timingFunction = CAMediaTimingFunction(name:
                                        kCAMediaTimingFunctionEaseOut)
  view.layer.add(transition, forKey: nil)
}
```

Everything after the comment telling you to add the following lines, all the `CATransition` stuff, is new.

I’m not going to go into too much detail here. Suffice it to say you’re setting up an animation that crossfades from what is currently on the screen to the changes you’re making in `startNewRound()` – reset the slider to center position and reset the values of the labels.

➤ Run the app and move the slider so that it is no longer in the center. Press the Start Over button and you should see a subtle crossfade animation.



Guess what... there's only one part of the course left! You're on fire.

Stay tuned for my next email, where you'll put the finishing touches on Bull's Eye, and get it running on your own device.

- Ray

If you'd like to stop receiving the iOS Apprentice Email Course but still stay subscribed to raywenderlich.com Weekly, click here.