8 Chapter 8: The Final App

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?

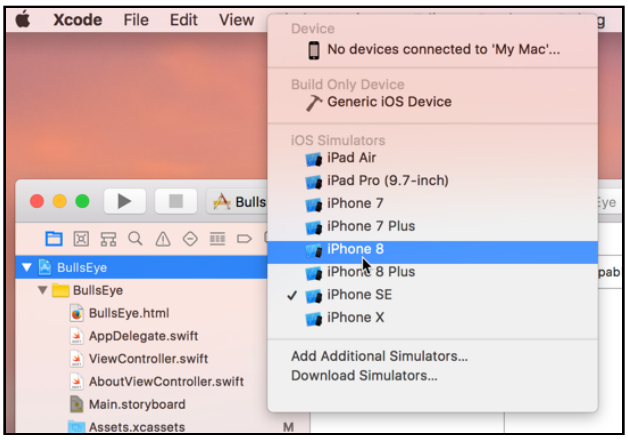
This chapter covers the following:

- **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.
- **The icon:** Add the app icon.
- **Display name:** Set the display name for the app.
- **Run on device:** How to configure everything to run your app on an actual device.

Support different screen sizes

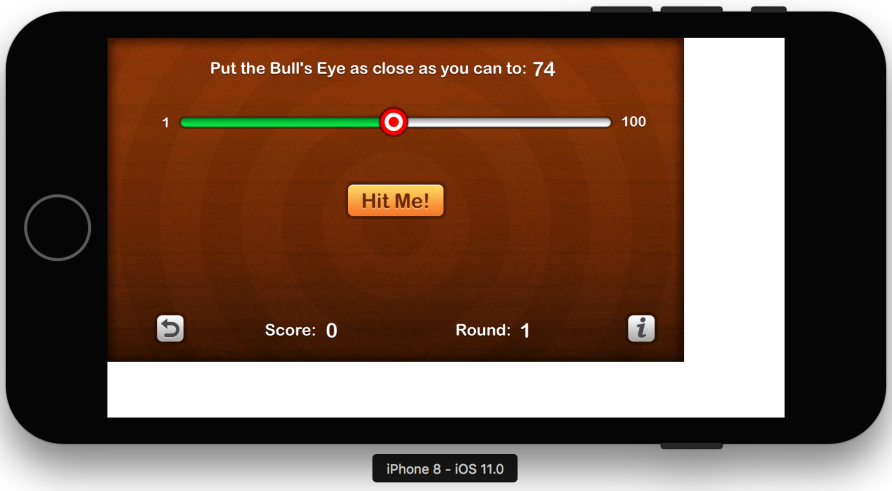
First, let's check if there is indeed an issue running Bull's Eye on a device with a larger screen. It's always good to verify that there's inded an issue before we do extra work, right? Why fix it, if it isn't broken? :]

➤ 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:



Using the scheme selector to switch to the iPhone 8 Simulator

The result might not be what you expected:



On the iPhone 8 Simulator, the app doesn't fill up the entire screen

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!

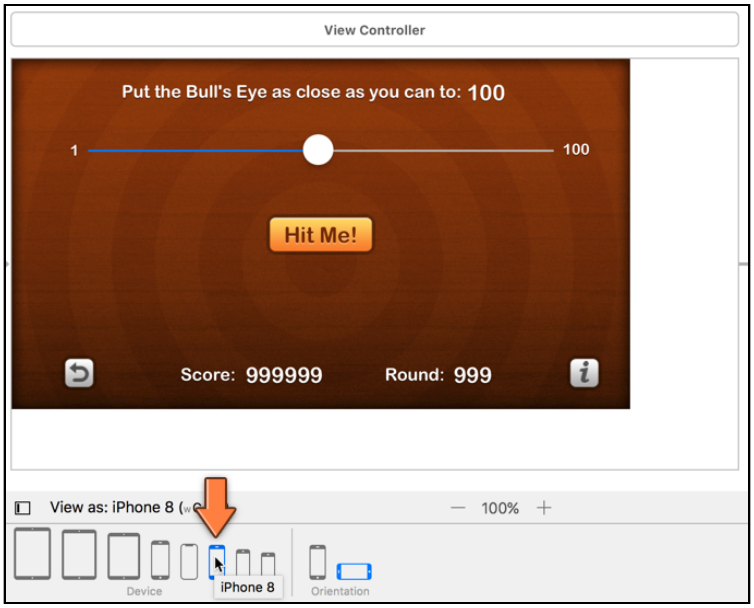
This is a good opportunity to learn about *Auto Layout*, a core UIKit technology that makes it easy to support many different screen sizes in your apps, including the larger screens of the 4.7-inch, 5.5-inch, and 5.8-inch iPhones, and the iPad.

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.)



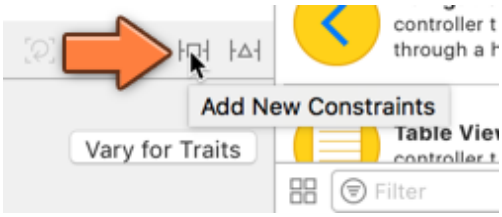
Viewing the storyboard on iPhone 8

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:

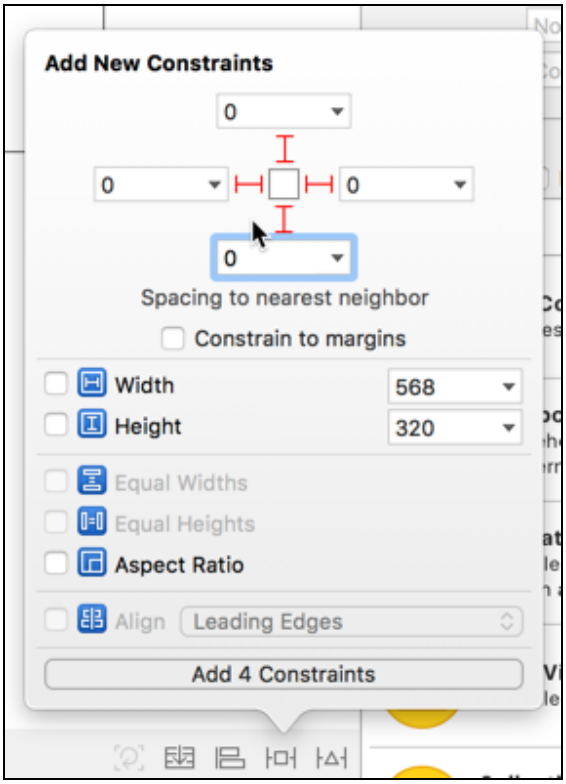


The Add New Constraints button

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.

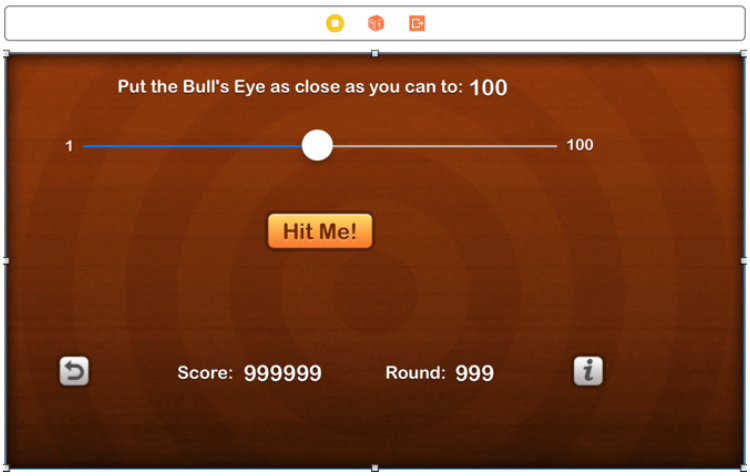
In order for the background image to stretch from edge-to-edge on the screen, the left, top, right, and bottom edges of the image should be flush against the screen edges. The way to do this with Auto Layout is to create two alignment constraints, one horizontal and one vertical.

➤ 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.):



Using the Add New Constraints menu to position the background image

➤ 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.)



The background image now covers the whole view

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

The new Auto Layout constraints appear in the Document Outline

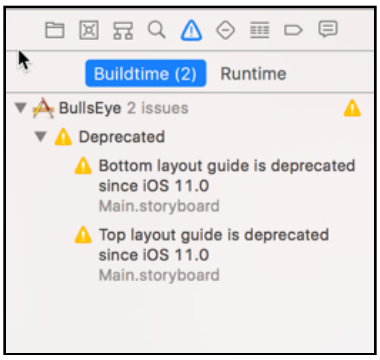
There should be four constraints listed there, one for each edge of the image.

➤ 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:



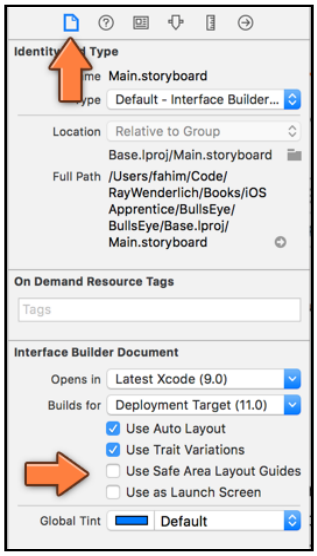
Auto Layout deprecation warnings

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**.



Enable Safe Area Layout Guides

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

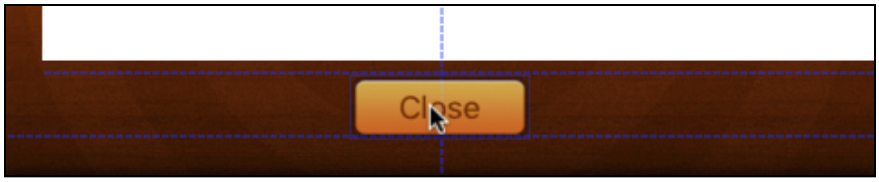
Let's repeat the background image fix for the About screen, too.

➤ 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.

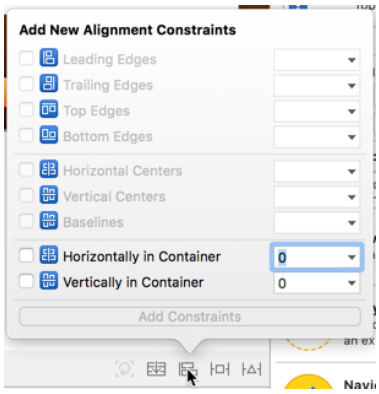
Interface Builder shows a handy guide, the dotted blue line, near the edges of the screen, which is useful for aligning objects by hand.



The dotted blue lines are guides that help position your UI elements

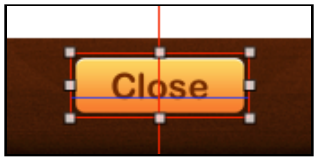
You want to create a centering constraint that keeps the Close button in the middle of the screen, regardless of how wide the screen is.

➤ 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**.



The Align menu

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



The Close button has red constraints

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.

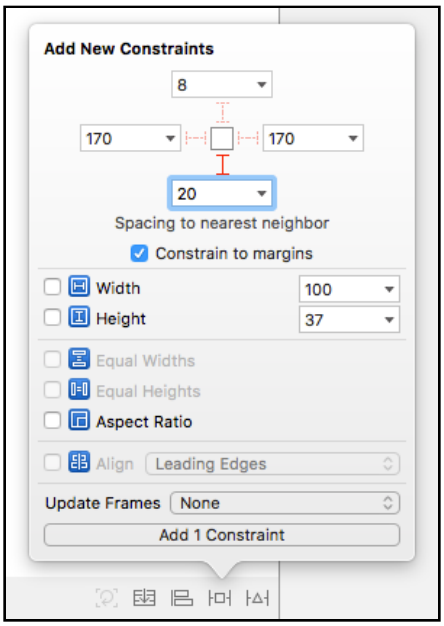
The thing to remember is this: for each view, there must always be enough constraints to define both its position and its size. The Close button already knows its size – you typed this into the Size inspector earlier – but for its position there is only a constraint for the X-coordinate (the alignment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

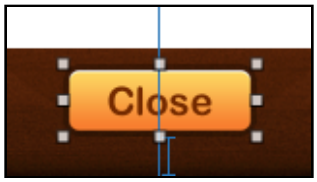
➤ In the **Add New Constraints** menu, in the **Spacing to nearest neighbor** section, set the bottom spacing to **20** and make sure that the I-beam above the text box is enabled.



The red I-beams decide the sides that are pinned down

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

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



The constraints on the Close button are valid

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

The views are not positioned according to the 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.

The Update Frames button

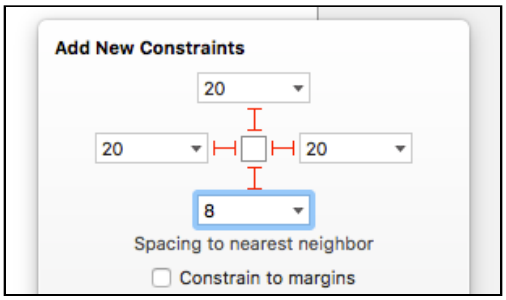
You can also use the **Editor → Resolve Auto Layout Issues → Update Frames** item from the menu bar.

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.

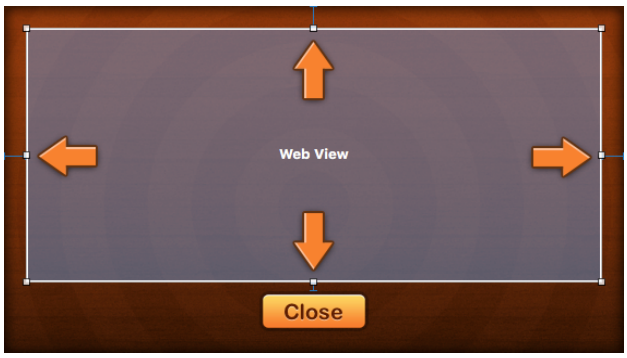
➤ Select the **Web View** and open the **Add New Constraints** menu. First, make sure **Constrain to margins** is unchecked. Then click all four I-beam icons so they become solid red and set their spacing to 20 points, except the bottom one which should be 8 points:



Creating the constraints for the web view

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

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



The four constraints on the web view

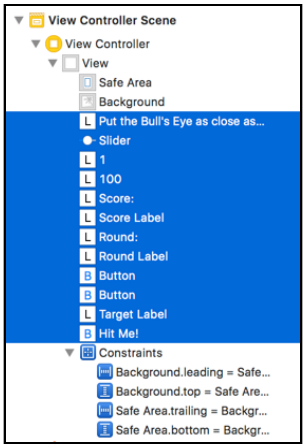
Three of these pin the web view to the main view, so that it always resizes along with it, and one connects it to the Close button. This is enough to determine the size and position of the web view in any scenario.

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.

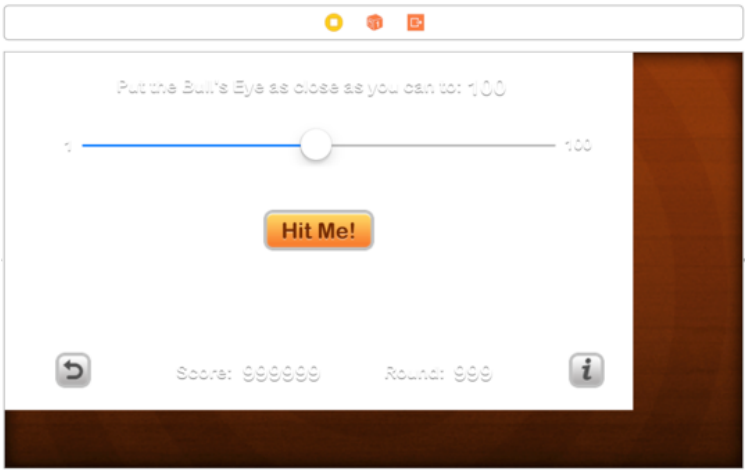
➤ Select all the labels, buttons, and the slider. You can hold down ⌘ and click them individually, but an easier method is to go to the **Document Outline**, click on the first view (for me that is the "Put the Bull's Eye as close as you can to:" label), then hold down Shift and click on the last view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



Selecting the views from the Document Outline

You should have selected everything but the background image view.

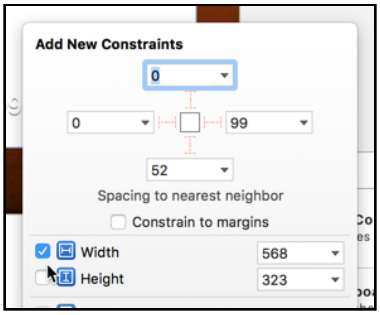
➤ From Xcode's menu bar, choose **Editor → Embed In → View**. This places the selected views inside a new container view:



The views are embedded in 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.



Pinning the width and height of the container view

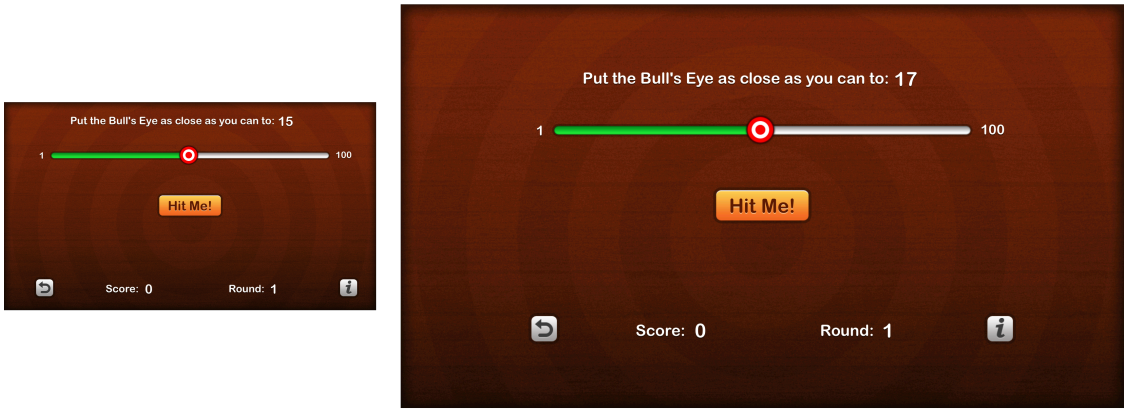
Interface Builder now draws several bars around the view that represent the Width and Height constraints that you just made, but they are red. Don't panic! It only means there are not enough constraints yet. No problem, you'll add the missing constraints next.

➤ 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:



The game running on 4-inch and 5.5-inch iPhones

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*.

> **Exercise:** 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.

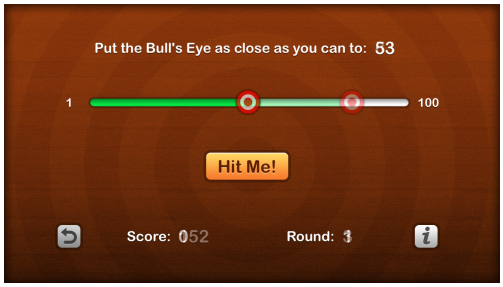
➤ Change `startNewGame()` to:

```
@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.

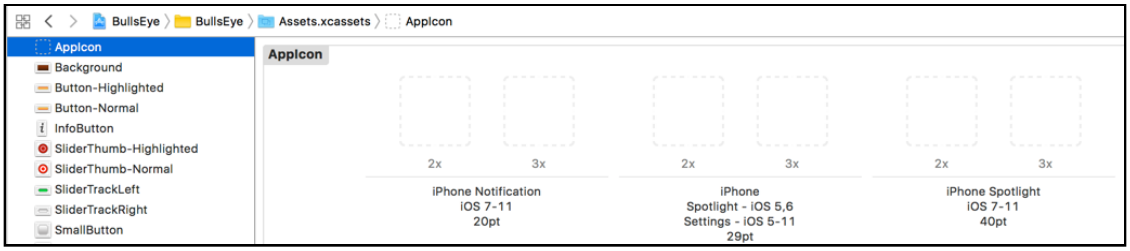


The screen crossfades between the old and new states

The icon

You're almost done with the app, but there are still a few loose ends to tie up. You may have noticed that the app has a really boring white icon. That won't do!

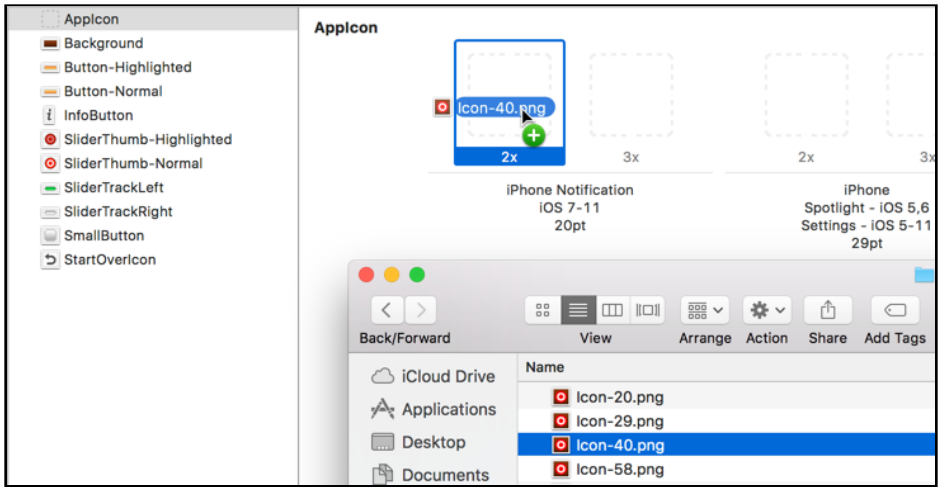
➤ Open the asset catalog (**Assets.xcassets**) and select **AppIcon**:



The AppIcon group in the asset catalog

This has ten groups for the different types of icons the app needs.

➤ In Finder, open the **Icon** folder from the resources. Drag the **Icon-40.png** file into the first slot, **iPhone Notification 20pt**:



Dragging the icon into the asset catalog

You may be wondering why you're dragging the Icon-40.png file and not the Icon-20.png into the slot for 20pt. Notice that this slot says **2x**, which means it's for Retina devices and on Retina screens one point counts as two pixels. So, 20pt = 40px. And the 40 in the icon name is for the size of the icon in pixels. Makes sense?

➤ Drag the **Icon-60.png** file into the **3x** slot next to it. This is for the iPhone Plus devices with their 3x resolution.

➤ For **iPhone Spotlight & Settings 29pt**, drag the **Icon-58.png** file into the 2x slot and **Icon-87.png** into the 3x slot. (What, you don't know your times table for 29?)

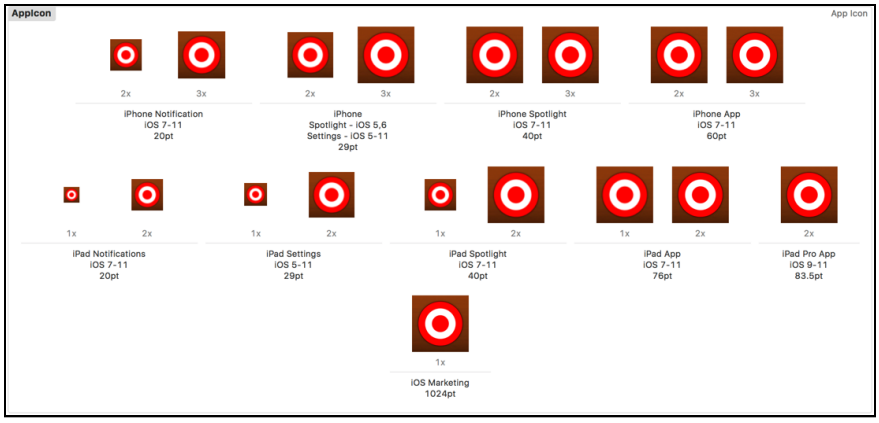
➤ For **iPhone Spotlight 40pt**, drag the **Icon-80.png** file into the 2x slot and **Icon-120.png** into the 3x slot.

➤ For **iPhone App 60pt**, drag the **Icon-120.png** file into the 2x slot and **Icon-180.png** into the 3x slot.

That's four icons in two different sizes. Phew!

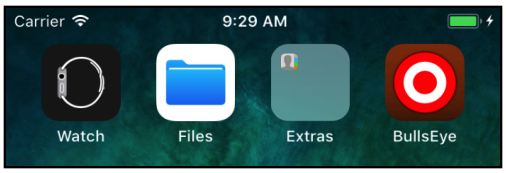
The other AppIcon groups are mostly for the iPad.

➤ Drag the specific icons (based on size) into the proper slots for iPad. Notice that the iPad icons need to be supplied in 1x as well as 2x sizes (but not 3x). You may need to do some mental arithmetic here to figure out which icon goes into which slot!



The full set of icons for the app

➤ Run the app and close it. You'll see that the icon has changed on the Simulator's springboard. If not, remove the app from the Simulator and try again (sometimes the Simulator keeps using the old icon and re-installing the app will fix this).



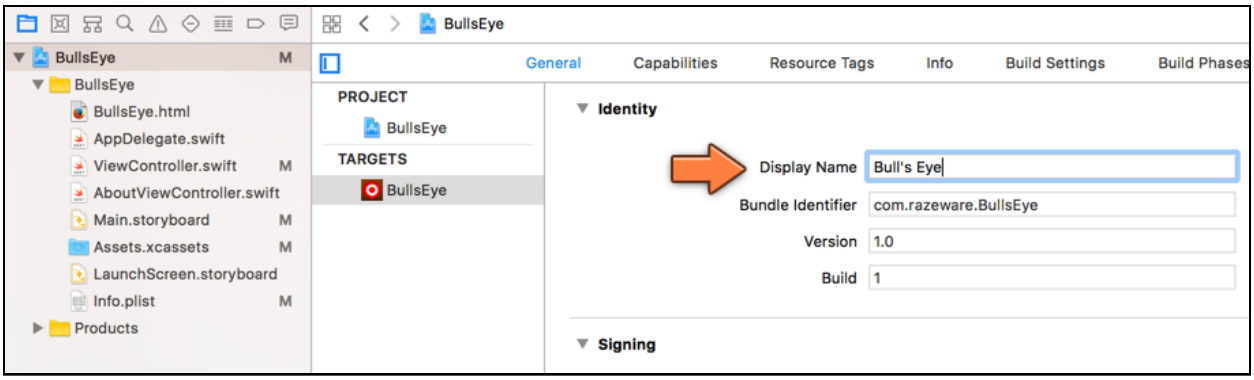
The icon on the Simulator's springboard

Display name

One last thing. You named the project **BullsEye** and that is the name that shows up under the icon. However, I'd prefer to spell it "**Bull's Eye**".

There is only limited space under the icon and for apps with longer names you have to get creative to make the name fit. For this game, however, there is enough room to add the space and the apostrophe.

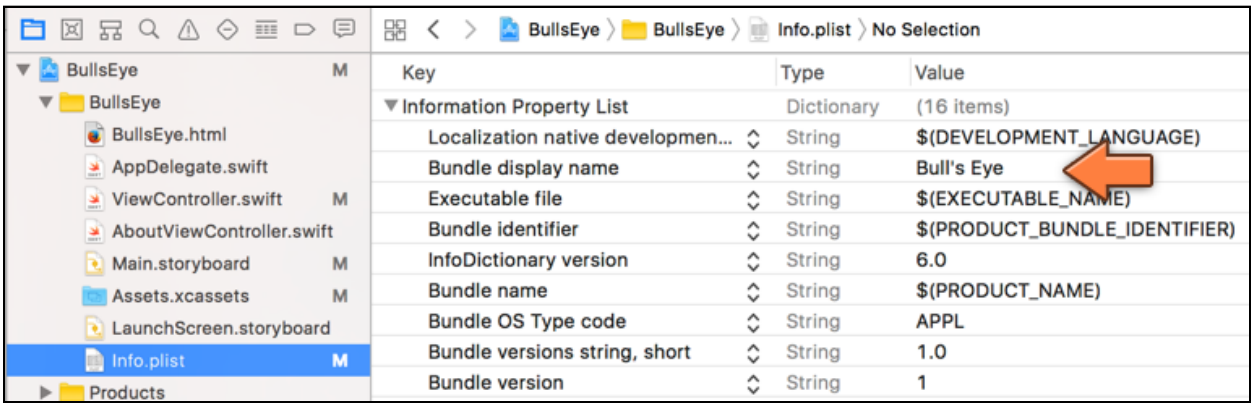
➤ Go to the **Project Settings** screen. The very first option is **Display Name**. Change this to **Bull's Eye**.



Changing the display name of the app

Like many of the project's settings you can also find the display name in the app's Info.plist file. Let's have a look.

➤ From the **Project navigator**, select **Info.plist**.



The display name of the app in Info.plist

The row **Bundle display name** contains the new name you've just entered.

> **Note:** If **Bundle display name** is not present, the app will use the value from the field **Bundle name**. That has the special value "$(PRODUCT_NAME)", meaning Xcode will automatically put the project name, BullsEye, in this field when it adds the Info.plist to the application bundle. By providing a **Bundle display name** you can override this default name and give the app any name you want.

➤ Run the app and quit it to see the new name under the icon.

The bundle display name setting changes the name under the icon

Awesome, that completes your very first app!

You can find the project files for the finished app under **08 - The Final App** in the Source Code folder.

Run on device

So far, you've run the app on the Simulator. That's nice and all but probably not why you're learning iOS development. You want to make apps that run on real iPhones and iPads! There's hardly a thing more exciting than running an app that you made on your own phone. And, of course, to show off the fruits of your labor to other people!

Don't get me wrong: developing your apps on the Simulator works very well. When developing, I spend most of my time with the Simulator and only test the app on my iPhone every so often.

However, you do need to run your creations on a real device in order to test them properly. There are some things the Simulator simply cannot do. If your app needs the iPhone's accelerometer, for example, you have no choice but to test that functionality on an actual device. Don't sit there and shake your Mac!

Until a few years back, you needed a paid Developer Program account to run apps on your iPhone. Since Xcode 7, however, you can do it for free. All you need is an Apple ID. And the latest Xcode makes it easier than ever before.

Configure your device for development

➤ Connect your iPhone, iPod touch, or iPad to your Mac using a USB cable.

➤ From the Xcode menu bar select **Window → Devices and Simulators** to open the Devices and Simulators window.

Mine looks like this (I'm using an iPhone 6s):

The Devices and Simulators window

On the left is a list of devices that are currently connected to my Mac and which can be used for development.

➤ Click your device name to select it.

If this is the first time you're using the device with Xcode, the Devices window will say something like, "iPhone is not paired with your computer." To pair the device with Xcode, you need to unlock the device first (hold the home button). After unlocking, an alert will pop up on the device asking you to trust the computer you're trying to pair with. Tap on **Trust** to continue.

Xcode will now refresh the page and let you use the device for development. Give it a few minutes (see the progress bar in the main Xcode window). If it takes too long, you may need to unplug the device and plug it back in.

At this point it's possible you may get the error message, "An error was encountered while enabling development on this device." You'll need to unplug the device and reboot it. Make sure to restart Xcode before you reconnect the device.

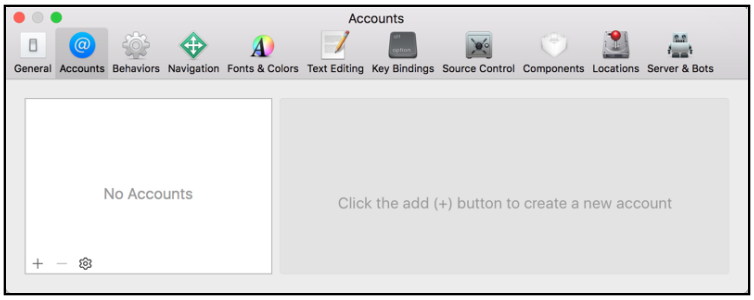
Also, note the checkbox which says **Connect via network**? That checkbox (gasp!) allows you to run and debug code on your iPhone over WiFi! Yes, that's new in Xcode 9. (I still prefer to do my debugging with my phone connected via USB cable since the last time I checked, the over network debugging was very slow. But your mileage may vary - so give it a try...)

Cool, that is the device sorted.

Add your developer account to Xcode

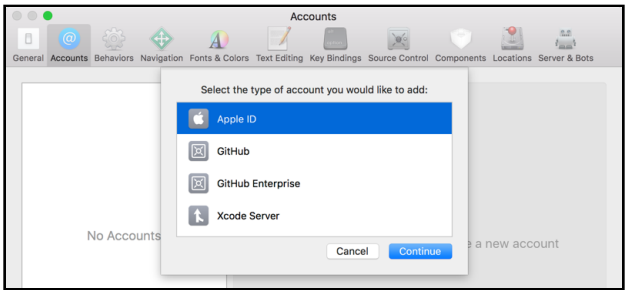
The next step is setting up your Apple ID with Xcode. It's OK to use the same Apple ID that you're already using with iTunes and your iPhone, but if you run a business, you might want to create a new Apple ID to keep things separate. Of course, if you've already registered for a paid Developer Program account, you should use that Apple ID.

➤ Open the **Accounts** pane in the Xcode Preferences window:



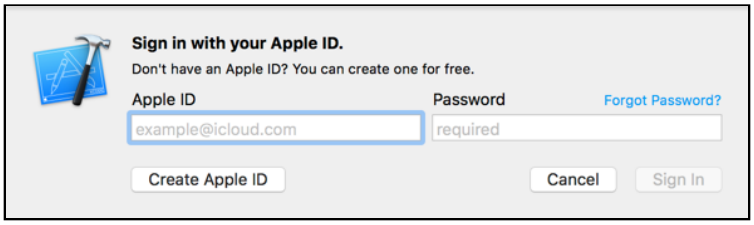
The Accounts preferences

➤ Click the + button at the bottom and select **Add Apple ID** from the list of options.



Xcode Account Type selection

Xcode will ask for your Apple ID:



Adding your Apple ID to Xcode

➤ Type your Apple ID username and password and click **Sign In**.

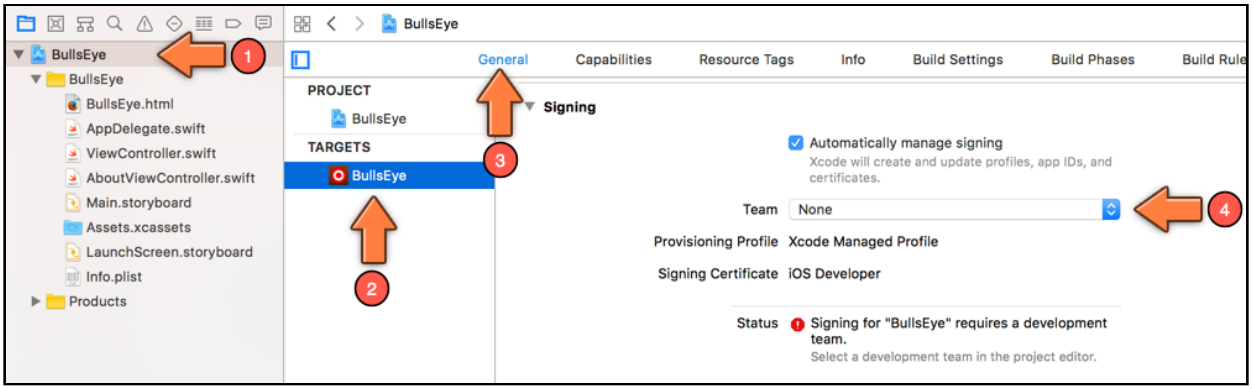
Xcode verifies your account details and adds it to the stored list of accounts.

> **Note:** It's possible that Xcode is unable to use the Apple ID your provided - for example, if it has been used with a Developer Program account in the past that is now expired. The simplest solution is to make a new Apple ID. It's free and only takes a few minutes. appleid.apple.com

You still need to tell Xcode to use this account when building your app.

Code signing

➤ Go to the **Project Settings** screen for your app target. In the **General** tab go to the **Signing** section.



The Signing options in the Project Settings screen

In order to allow Xcode to put an app on your iPhone, the app must be *digitally signed* with your **Development Certificate**. A *certificate* is an electronic document that identifies you as an iOS application developer and is valid only for a specific amount of time.

Apps that you want to submit to the App Store must be signed with another certificate, the **Distribution Certificate**. To use the distribution certificate you must be a member of the paid Developer Program, but using the development certificate is free.

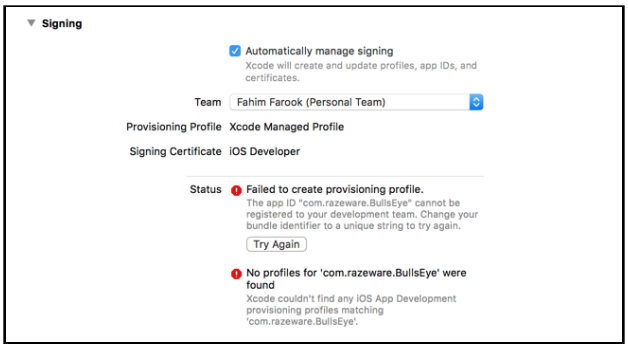
In addition to a valid certificate, you also need a **Provisioning Profile** for each app you make. Xcode uses this profile to sign the app for use on your particular device (or devices). The specifics don't really matter, just know that you need a provisioning profile or the app won't go on your device.

Making the certificates and provisioning profiles used to be a really frustrating and error-prone process. Fortunately, those days are over: Xcode now makes it really easy. When the **Automatically manage signing** option is enabled, Xcode will take care of all this business with certificates and provisioning profiles and you don't have to worry about a thing.

➤ Click on **Team** to select your Apple ID.

Xcode will now automatically register your device with your account, create a new Development Certificate, and download and install the Provisioning Profile on your device. These are all steps you had to do by hand in the past, but now Xcode takes care of all that.

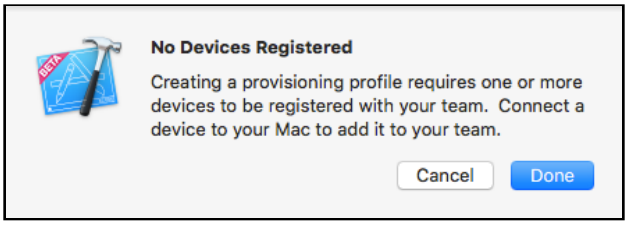
You could get some signing errors like these:



Signing/team set up errors

The app's Bundle Identifier – or App ID as it's called here – must be unique. If another app is already using that identifier, then you cannot use it anymore. That's why you're supposed to start the Bundle ID with your own domain name. The fix is easy: change the Bundle Identifier field to something else and try again.

It's also possible you get this error (or something similar):



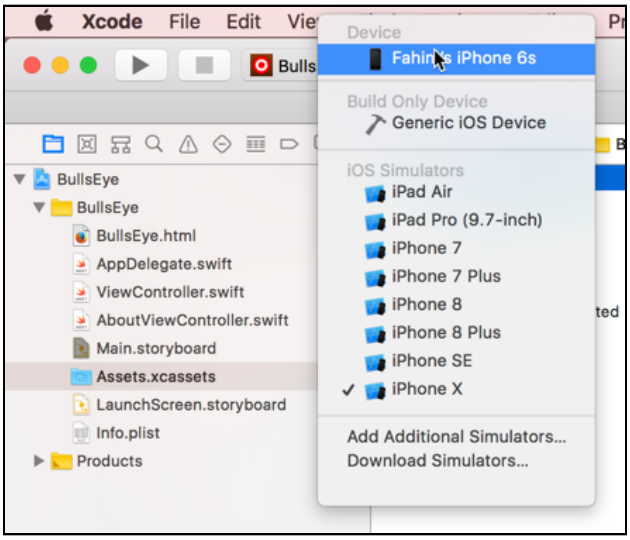
No devices registered

Xcode must know about the device that you're going to run the app on. That's why I asked you to connect your device first. Double-check that your iPhone or iPad is still connected to your Mac and that it is listed in the Devices window.

Run on device

If everything goes smoothly, go back to Xcode's main window and click on the box in the toolbar to change where you will run the app. The name of your device should be in that list somewhere.

On my system it looks like this:



Setting the active device

You're all set and ready to go!

➤ Press **Run** to launch the app.

At this point you may get a popup with the question "codesign wants to sign using key ... in your keychain". If so, answer with **Always Allow**. This is Xcode trying to use the new Development Certificate you just created - you just need to give it permission first.

Does the app work? Awesome! If not, read on...

When things go wrong...

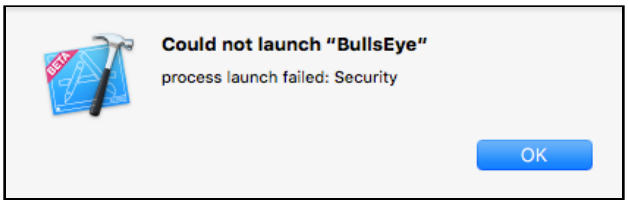
There are a few things that can go wrong when you try to put the app on your device, especially if you've never done this before, so don't panic if you run into problems.

The device is not connected

Make sure your iPhone, iPod touch, or iPad is connected to your Mac. The device must be listed in Xcode's Devices window and there should not be a yellow warning icon.

The device does not trust you

You might get this warning:



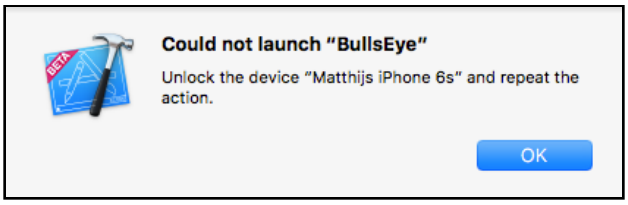
Quick, call security!

On the device itself there will be a popup with the text, "Untrusted Developer. Your device management settings do not allow using apps from developer ...".

If this happens, open the Settings app on the device and go to **General** → **Profile**. Your Apple ID should be listed in that screen. Tap it, followed by the Trust button. Then try running the app again.

The device is locked

If your phone locks itself with a passcode after a few minutes, you might get this warning:



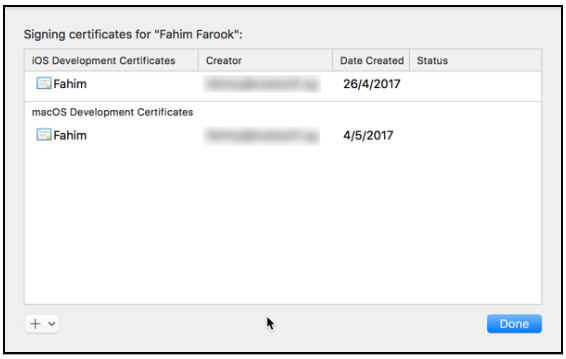
The app won't run if the device is locked

Simply unlock your device (hold the home button or type in the 4-digit passcode) and press Run again.

Signing certificates

If you're curious about these certificates, then open the **Preferences** window and go to the **Accounts** tab. Select your account and click the **Manage Certificates...** button in the bottom-right corner.

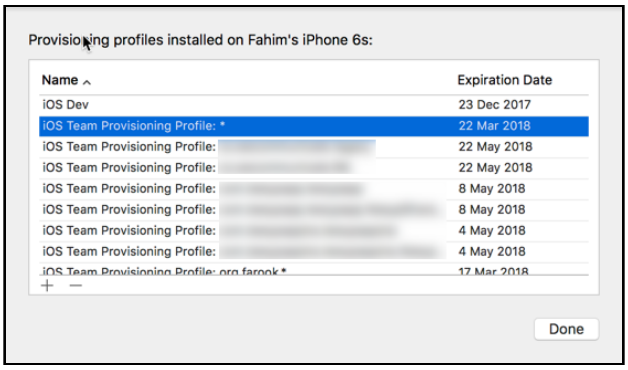
This brings up another panel, listing your signing certificates:



The Manage Certificates panel

When you're done, close the panel and go to the **Devices and Simulators** window.

You can see the provisioning profiles that are installed on your device by right-clicking the device name and choosing **Show Provisioning Profiles**:



The provisioning profiles on your device

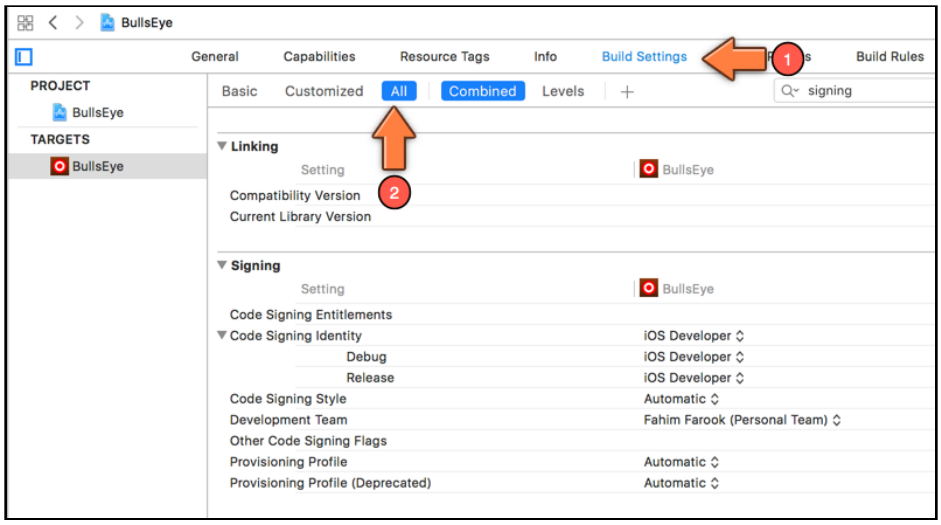
The "iOS Team Provisioning Profile: *" is the thing that allows you to run the app on your device. (By the way, they call it the "team" profile because often there is more than one developer working on an app and they can all share the same profile.)

You can have more than one certificate and provisioning profile installed. This is useful if you're on multiple development teams or if you prefer to manage the provisioning profiles for different apps by hand.

To see how Xcode chooses which profile and certificate to sign your app with, go to the **Project Settings** screen and switch to the **Build Settings** tab. There are a lot of settings

in this list, so filter them by typing **signing** in the search box. (Also make sure **All** is selected, not Basic.)

The screen will look something like this:



The Code Signing settings

Under **Code Signing Identity** it says **iOS Developer**. This is the certificate that Xcode uses to sign the app. If you click on that line, you can choose another certificate. Under **Provisioning Profile** you can change the active profile. Most of the time you won't need to change these settings, but at least you know where to find them now.

And that concludes everything you need to know about running your app on an actual device.

The end... or the beginning?

It has been a bit of a journey to get to this point – if you're new to programming, you've had to get a lot of new concepts into your head. I hope your brain didn't explode!

At least you should have gotten some insight into what it takes to develop an app.

I don't expect you to understand exactly everything that you did, especially not the parts that involved writing Swift code. It is perfectly fine if you didn't, as long as you're enjoying yourself and you sort of get the basic concepts of objects, methods and variables.

If you were able to follow along and do the exercises, you're in good shape!

I encourage you to play around with the code a bit more. The best way to learn programming is to do it, and that includes making mistakes and messing things up. I hereby grant you full permission to do so! Maybe you can add some cool new features to the game (and if you do, please let me know).

In the Source Code folder for this book you can find the complete source code for the *Bull's Eye* app. If you're still unclear about some of what you did, it might be a good idea to look at this cleaned up source code.

If you're interested in how I made the graphics, then take a peek at the Photoshop files in the Resources folder. The wood background texture was made by Atle Mo from subtlepatterns.com.

If you're feeling exhausted after all that coding, pour yourself a drink and put your feet up for a bit. You've earned it! On the other hand, if you just can't wait to get to grips with more code, let's move on to our next app!