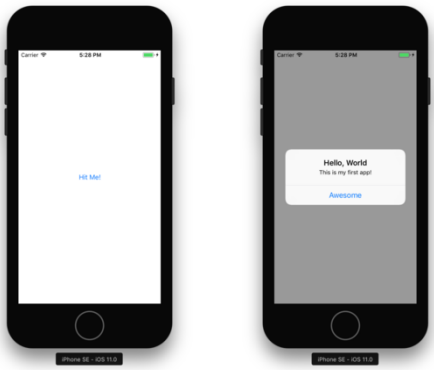
Ray Wenderlich para mim 26 de jul

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

In the last part of the course, you created a "programming to-do" list that outlined the steps to create the Bull's Eye game. Likely, one of the items on your list was to create a button the player can tap to submit their guess.

So let's start by making an extremely simple first version of the game that just displays a single button. When you press the button, the app pops up an alert message. That's all you are going to do for now. Once you have this working, you can build the rest of the game on this foundation.

The app will look like this:

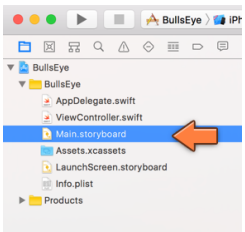


Open up the Bull's Eye project where you left it off last time (or download the starter project from the corresponding forum discussion thread), and take a look at Xcode.

The left pane of the Xcode window is named the **Navigator area**. The row of icons along the top lets you select a specific navigator. The default navigator is the **Project navigator**, which shows the files in your project.

The organization of these files roughly corresponds to the project folder on your hard disk, but that isn't necessarily always so. You can move files around and put them into new groups and organize away to your heart's content. We'll talk more about the different files in your project later.

➤ In the **Project navigator**, find the item named **Main.storyboard** and click it once to select it:

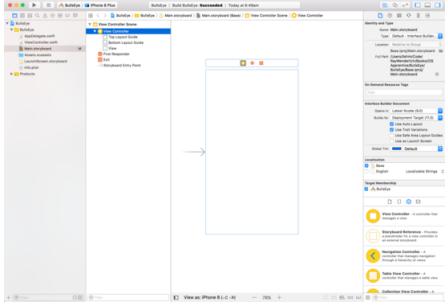


Like a superhero changing his/her clothes in a phone booth, the main editing pane now transforms into the **Interface Builder**. This tool lets you drag-and-drop user interface components such as buttons to create the UI of your app. (OK, bad analogy, but Interface Builder is a super tool in my opinion.)

➤ If it's not already blue, click the **Hide or show utilities** button in Xcode's toolbar:

These toolbar buttons change the appearance of Xcode. This one in particular opens a new pane on the right side of the Xcode window.

Your Xcode should now look something like this:



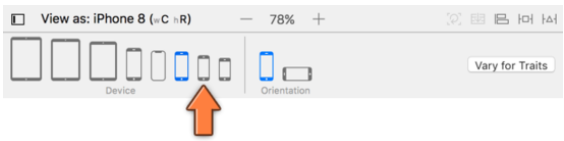
This is the **storyboard** for your app. The storyboard contains the designs for all of your app's screens, and shows the navigation flow in your app from one screen to another.

Currently, the storyboard contains just a single screen or **scene**, represented by a rectangle in the middle of the Interface Builder canvas.

> **Note:** If you don't see the rectangle labeled "View Controller" but only an empty white canvas, then use your mouse or trackpad to scroll the storyboard around a bit. Trust me, it's in there somewhere! Also make sure your Xcode window is large enough. Interface Builder takes up a lot of space…

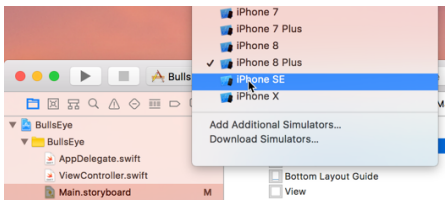
The scene currently has the size of an iPhone 8. To keep things simple, you will first design the app for the iPhone SE, which has a slightly smaller screen. Later you'll also make the app fit on the larger iPhone models.

➤ At the bottom of the Interface Builder window, click **View as: iPhone 8** to open up the following panel:



Select the **iPhone SE**, the second smallest iPhone, thus resizing the preview UI you see in Interface Builder to be set to that of an iPhone SE. You'll notice that the scene's rectangle now becomes a bit smaller, corresponding to the screen size of the iPhone 5, iPhone 5s, and iPhone SE models.

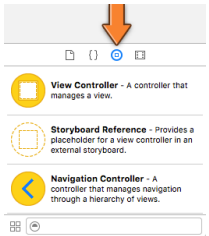
➤ In the Xcode toolbar, make sure it says **BullsEye > iPhone SE** (next to the Stop button). If it doesn't then click it and pick iPhone SE from the list:



Now when you run the app, it will run on the iPhone SE Simulator (try it out!).

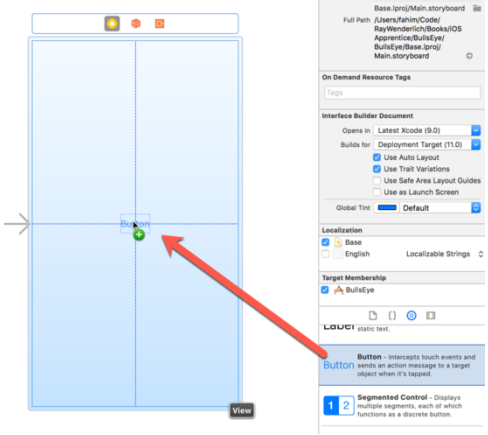
Back to the storyboard:

➤ At the bottom of the Utilities pane you will find the **Object Library** (make sure the third button, the one that looks like a circle, is selected):



Scroll through the items in the Object Library's list until you see **Button**. (Alternatively, you can type the word "button" in to the search/filter box at the bottom of the Object Library.)

➤ Click on **Button** and drag it into the working area, on top of the scene's rectangle.



That's how easy it is to add new buttons, just drag & drop. That goes for all other user interface elements too. You'll be doing a lot of this, so take some time to get familiar with the process.

➤ Drag-and-drop a few other controls, such as labels, sliders, and switches, just to get the hang of it.

This should give you some idea of the UI controls that are available in iOS. Notice that the Interface Builder helps you to layout your controls by snapping them to the edges of the view and to other objects. It's a very handy tool!

➤ Double-click the button to edit its title. Call it Hit Me!

Hit Me!

It's possible that your button has a border around it:

Hit Me!

This border is not part of the button, it's just there to show you how large the button is. You can turn these rectangles on or off using the **Editor → Canvas → Show Bounds Rectangles** menu option.

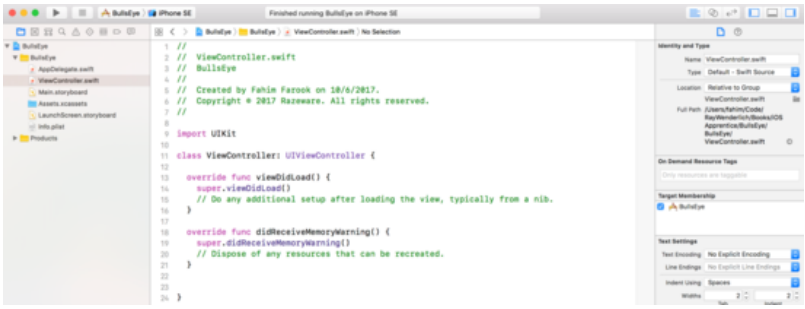
When you're done playing with Interface Builder, press the Run button from Xcode's toolbar. The app should now appear in the Simulator, complete with your "Hit Me!" button. However, when you tap the button it doesn't do anything yet. For that you'll have to write some Swift code!

The source code editor

A button that doesn't do anything when tapped is of no use to anyone. So, let's make it show an alert popup. In the finished game the alert will display the player's score, but for now we shall limit ourselves to a simple text message (the traditional "Hello, World!").

➤ In the **Project navigator**, click on **ViewController.swift**.

The Interface Builder will disappear and the editor area now contains a bunch of brightly colored text. This is the Swift source code for your app:



➤ Add the following lines directly above the very last } bracket in the file:

```
@IBAction func showAlert() {
}
```

The source code for **ViewController.swift** should now look like this:

```
//
//  ViewController.swift
//  BullsEye
//
//  Created by <you> on <date>.
//  Copyright © <year> <your organization>. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

  override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a n
ib.
  }

  override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    // Dispose of any resources that can be recreated.
  }

  @IBAction func showAlert() {
  }
}
```

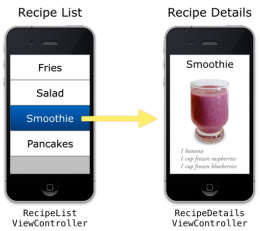
How do you like your first taste of Swift? Before we can tell you what this all means, we have to introduce the concept of a view controller.

View controllers

You've edited the **Main.storyboard** file to build the user interface of the app. It's only a button on a white background, but a user interface nonetheless. You also added source code to **ViewController.swift**.

These two files – the storyboard and the Swift file – together form the design and implementation of a **view controller.** A lot of the work in building iOS apps is making view controllers. The job of a view controller is to manage a single screen in your app.

Take a simple cookbook app, for example. When you launch the cookbook app, its main screen lists the available recipes. Tapping a recipe opens a new screen that shows the recipe in detail with an appetizing photo and cooking instructions. Each of these screens is managed by a view controller.



What these two screens do is very different. One is a list of several items; the other presents a detail view of a single item.

That's why you need two view controllers: one that knows how to deal with lists, and another that can handle images and cooking instructions. One of the design principles of iOS is that each screen in your app gets its own view controller.

Currently **Bull's Eye** has only one screen (the white one with the button) and thus only needs one view controller. That view controller is simply named "ViewController" and the storyboard and Swift file work together to implement it. (If you are curious, you can check the connection between the screen and the code for it by switching to the Identity inspector on the right sidebar of Xcode in the storyboard view. The class value shows the current class associated with the storyboard scene.)

Simply put, the Main.storyboard file contains the design of the view controller's user interface, while ViewController.swift contains its functionality – the logic that makes the user interface work, written in the Swift language.

Because you used the Single View Application template, Xcode automatically created the view controller for you. Later you will add a second screen to the game and you will create your own view controller for that.

Make connections

The line of source code you have just added to ViewController.swift lets Interface Builder know that the controller has a "showAlert" action, which presumably will show

an alert popup. You will now connect the button on the storyboard to that action in your source code.

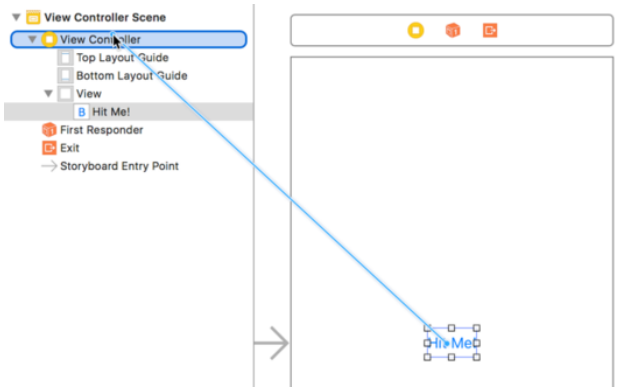
➤ Click **Main.storyboard** to go back into Interface Builder.

In Interface Builder, there should be a second pane on the left, next to the navigator area, the **Document Outline**, that lists all the items in your storyboard. If you do not see that pane, click the small toggle button in the bottom-left corner of the Interface Builder canvas to reveal it.

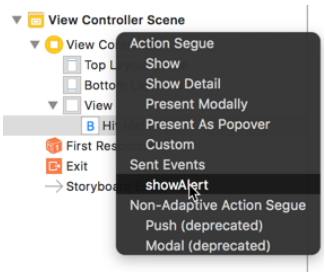
➤ Click the **Hit Me button** once to select it.

With the Hit Me button selected, hold down the **Control** key, click on the button and drag up to the **View Controller** item in the Document Outline. You should see a blue line going from the button up to View Controller.

(Instead of holding down Control, you can also right-click and drag, but don't let go of the mouse button before you start dragging.)



Once you're on View Controller, let go of the mouse button and a small menu will appear. It contains two sections, "Action Segue" and "Sent Events", with one or more options below each. You're interested in the **showAlert** option under Sent Events. The Sent Events section shows all possible actions in your source code that can be hooked up to your storyboad and **showAlert** is the name of the action that you added earlier in the source code of **ViewController.swift**.

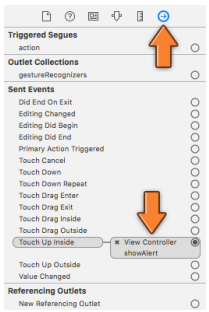


➤ Click on **showAlert** to select it. This instructs Interface Builder to make a connection between the button and the line `@IBAction func showAlert()`.

From now on, whenever the button is tapped the `showAlert` action will be performed. That is how you make buttons and other controls do things: you define an action in the view controller's Swift file and then you make the connection in Interface Builder.

You can see that the connection was made by going to the **Connections inspector** in the Utilities pane on the right side of the Xcode window.

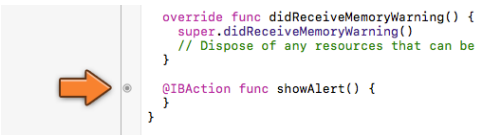
➤ Click the small arrow-shaped button at the top of the pane to switch to the Connections inspector:



In the Sent Events section, the “Touch Up Inside” event is now connected to the showAlert action. You can also see the connection in the Swift file.

➤ Select **ViewController.swift** to edit it.

Notice how to the left of the line with `@IBAction func showAlert()`, there is a solid circle? Click on that circle to reveal what this action is connected to.



Act on the button

You now have a screen with a button. The button is hooked up to an action named `showAlert` that will be performed when the user taps the button.

Currently, however, the action is empty and nothing will happen (try it out by running the app again, if you like). You need to give the app more instructions.

➤ In **ViewController.swift**, modify `showAlert` to look like the following:

```
@IBAction func showAlert() {
  let alert = UIAlertController(title: "Hello, World",
                                message: "This is my first app!",
                         preferredStyle: .alert)

  let action = UIAlertAction(title: "Awesome", style: .default,
                             handler: nil)

  alert.addAction(action)

  present(alert, animated: true, completion: nil)
}
```

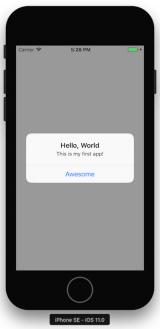
The new lines of code implement the actual alert display functionality.

The commands between the `{ }` brackets tell the iPhone what to do, and they are performed from top to bottom.

The code in `showAlert` creates an alert with a title “Hello, World”, a message “This is my first app!” and a single button labeled “Awesome”.

If you’re not sure about the distinction between the title and the message: both show text, but the title is slightly bigger and in a bold typeface.

➤ Click the **Run** button from Xcode’s toolbar. If you didn’t make any typos, your app should launch in the Simulator and you should see the alert box when you tap the button.



Congratulations, you've completed another major milestone in your iOS journey! What you just did may have been mostly gibberish to you, but that shouldn’t matter. We take it one small step at a time.

You can strike off the first two items from the to-do list already: putting a button on the screen and showing an alert when the user taps the button.

Take a little break, let it all sink in, and get ready for the next part of the course! You’re only just getting started…

> **Note:** Just in case you got stuck, we have provided the complete Xcode projects which are snapshots of the project in the corresponding forum discussion thread. That way you can compare your version of the app to ours, or – if you really make a mess of things – continue from a version that is known to work. Also, that's a great place to ask any questions about this part of the course!

In the next few days, I'll send you another email, where you'll continue going down your task list and tick off some other items by adding some more controls to your game. Stay tuned!

- Ray

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