

<https://github.com/GroundSpeed/SwiftlyBreakingGround>

# SWIFTLY BREAKING GROUND WITH IOS

Don Miller  
Founder of GroundSpeed™  
Co-founder of Seed Coworking  
@donmiller



Don Miller

GroundSpeed™  
rapid web + mobile software

# SCHEDULE

Introductions

Discuss Terminology, Xcode Overview

Create Your First App: Hello World!

MVC Overview

Color Me Demo

Swift Basics and the Playground

Storyboards

UITableViews (if we have time)



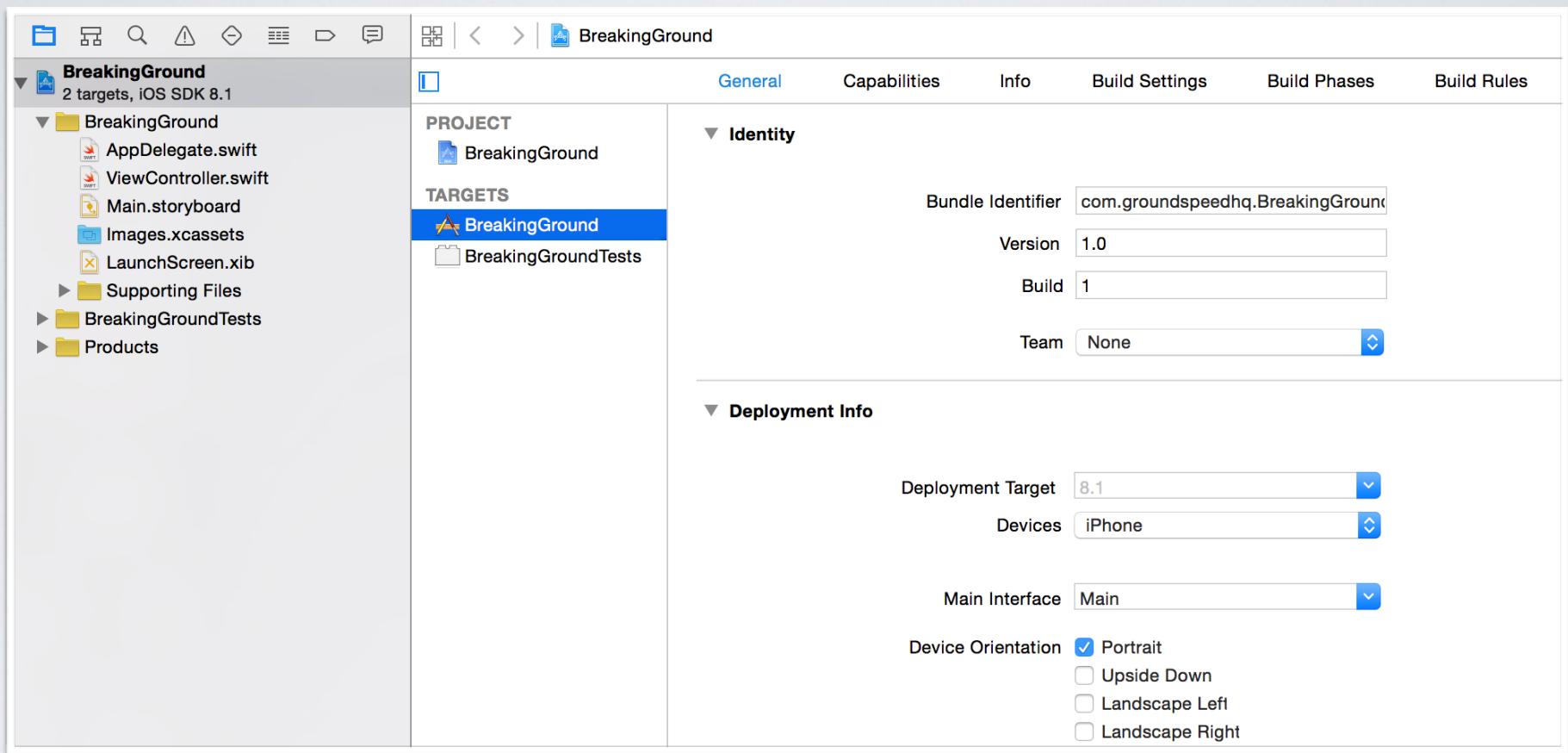
GroundSpeed™  
rapid web + mobile software

# TERMINOLOGY

- ▶ iOS
- ▶ iPhone
- ▶ iPad
- ▶ iPod
- ▶ MacBook
- ▶ iMac
- ▶ SSD
- ▶ SDK
- ▶ IDE
- ▶ Xcode
- ▶ OSX
- ▶ Mavericks
- ▶ Mountain Lion
- ▶ Lion
- ▶ Snow Leopard, etc.
- ▶ Retina



# XCODE 6.1 & IOS 8.1.2



GroundSpeed™  
rapid web + mobile software

# HELLO WORLD!



GroundSpeed™  
rapid web + mobile software

# CREATE A NEW SINGLE VIEW APPLICATION

Choose a template for your new project:

iOS				
Application	Master-Detail Application	Page-Based Application	Single View Application	Tabbed Application
Framework & Library				
Other				
OS X				
Application	Game			
Framework & Library				
System Plug-in				
Other				

**Single View Application**

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a storyboard or nib file that contains the view.

**Cancel** **Previous** **Next**



GroundSpeed™  
rapid web + mobile software

# ENTER THE PRODUCT NAME

Choose options for your new project:

Product Name:	HelloWorld
Organization Name:	GroundSpeed
Organization Identifier:	com.groundspeedhq
Bundle Identifier:	com.groundspeedhq.HelloWorld
Language:	Swift
Devices:	iPhone
<input type="checkbox"/> Use Core Data	

**Cancel** **Previous** **Next**



# UPDATE THE GENERAL PROJECT ATTRIBUTES

The screenshot shows the Xcode interface with the project 'HelloWorld' selected. The left sidebar shows the project structure with files like AppDelegate.swift, ViewController.swift, Main.storyboard, and LaunchScreen.xib. The main area displays the 'General' tab of the project settings. The 'Identity' section contains the Bundle Identifier (com.groundspeedhq.HelloWorld), Version (1.0), Build (1), and Team (None). The 'Deployment Info' section includes the Deployment Target (8.1), Devices (iPhone), Main Interface (Main), and Device Orientation (Portrait checked, Upside Down, Landscape Left, Landscape Right unchecked). The 'Status Bar Style' is set to Default, and the 'Hide status bar' option is unchecked. The right side of the screen shows the 'Identity and Type' panel with the project's name, location, and other document settings.

**Identity and Type**

- Name **HelloWorld**
- Location **Absolute**  
HelloWorld.xcodeproj
- Full Path /Users/donmiller/Documents/Projects/iOS/Playground/HelloWorld/HelloWorld.xcodeproj

**Project Document**

- Project Format **Xcode 3.2-compatible**
- Organization **GroundSpeed**
- Class Prefix

**Text Settings**

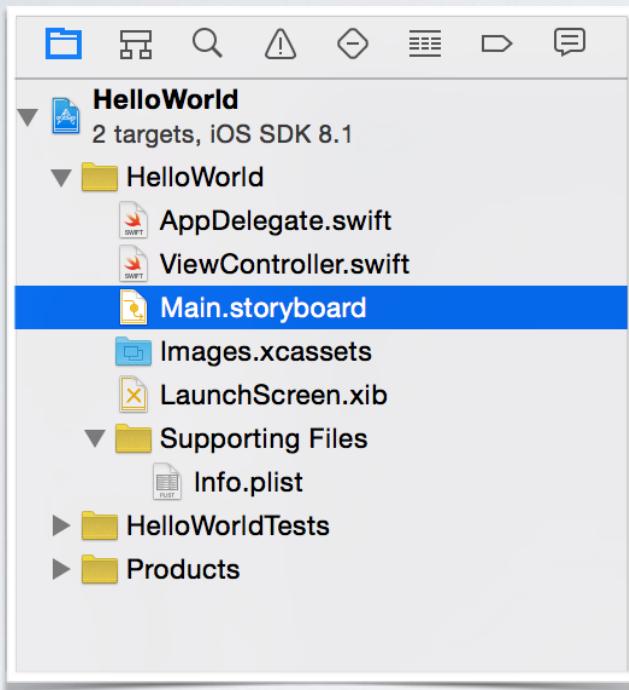
- Indent Using **Spaces**
- Widths **4** Tab **4** Indent
- Wrap lines

**Source Control**

- Repository --
- Type --
- Current Branch --
- Version --
- Status **No changes**
- Location



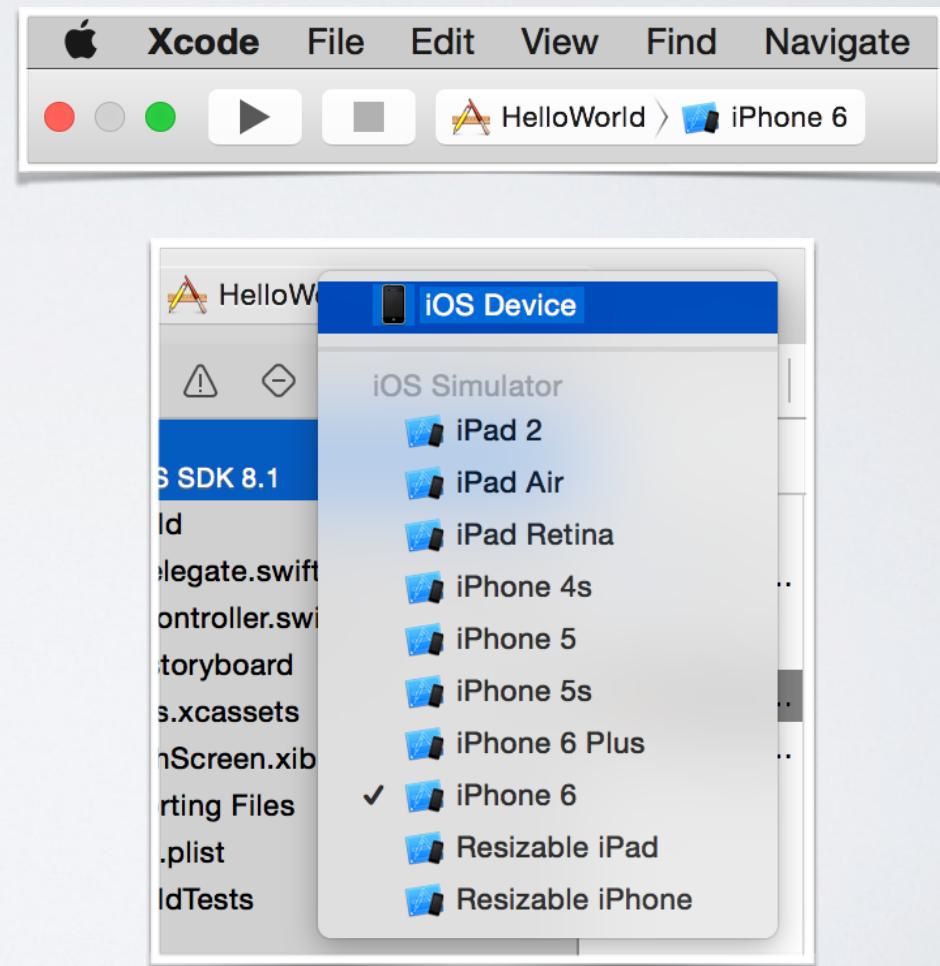
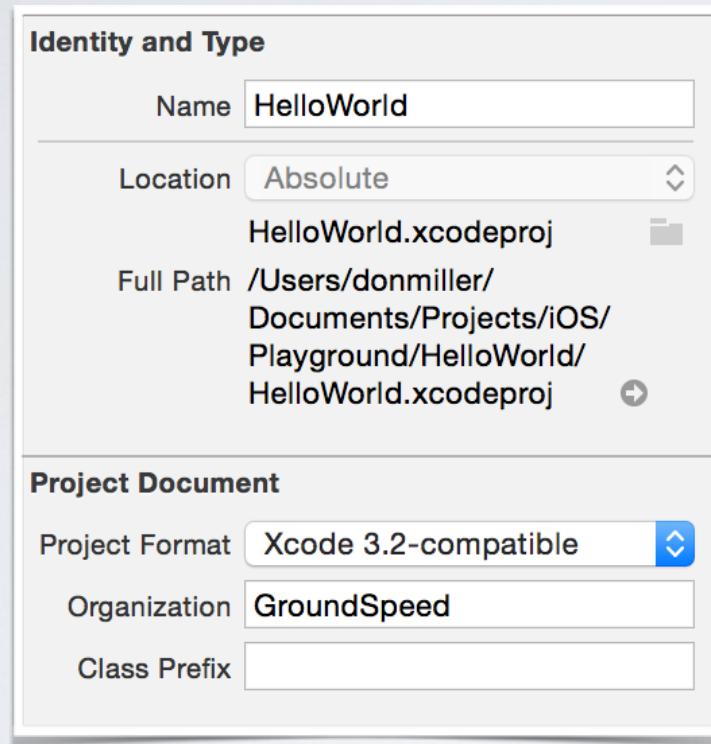
# PROJECT STRUCTURE



- Folders do not match physical layout on file system.
- Actually you will not find any sub folder named like Supporting Files in your physical project folder.
- The folder hierarchy in the XCode's navigator is simply a logical grouping of the resources.
- At this stage, we would be interested only on the Main.storyboard file.



# FILE INSPECTOR & SCHEMA SETTINGS



# DEVICES & TARGETS

General Capabilities Info Build Settings Build Phases Build Rules

**PROJECT**  
HelloWorld

**TARGETS**  
HelloWorld (selected)  
HelloWorldTests

**Identity**

Bundle Identifier: com.groundspeedhq.HelloWorld

Version: 1.0

Build: 1

Team: None

**Deployment Info**

Deployment Target: 8.1

Devices: iPhone

Main Interface: Main

Device Orientation:

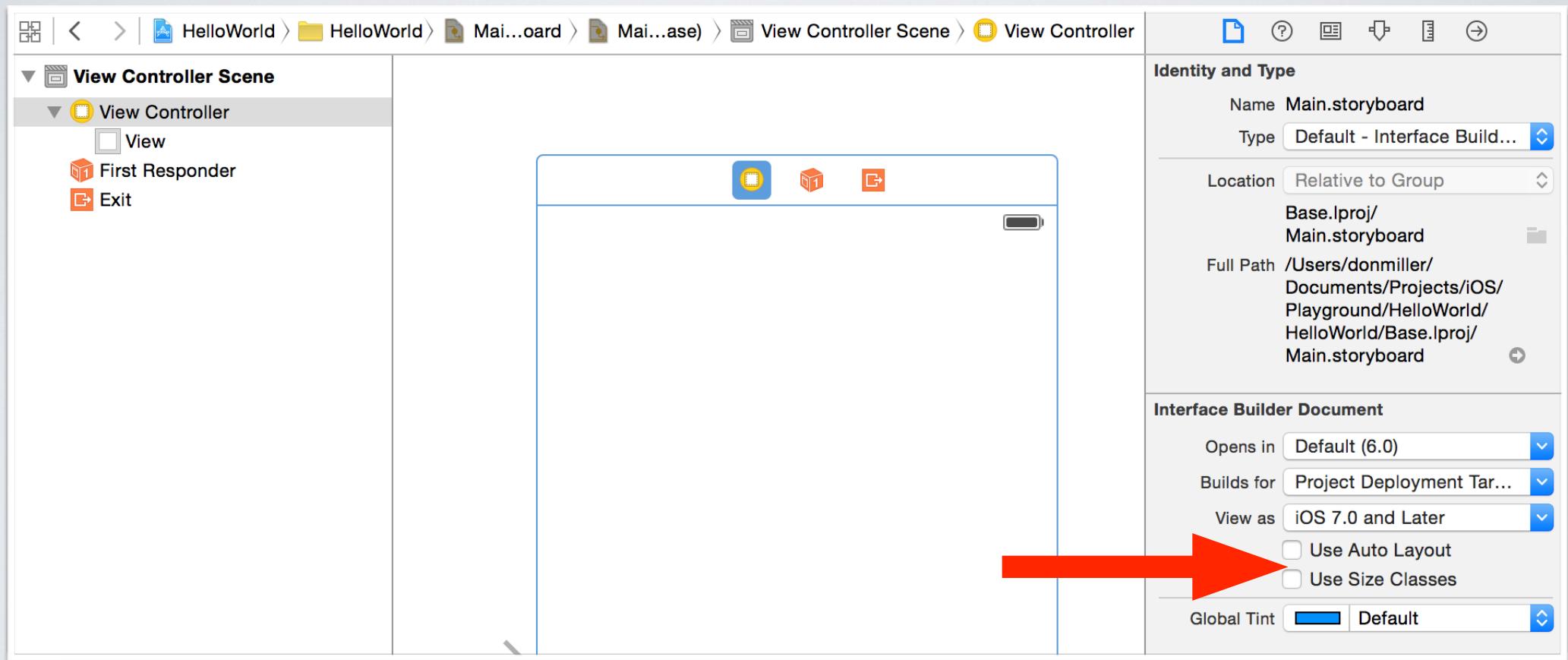
- Portrait
- Upside Down
- Landscape Left
- Landscape Right

Status Bar Style: Default

Hide status bar



# FILE OWNER, VIEW, & FIRST RESPONDER

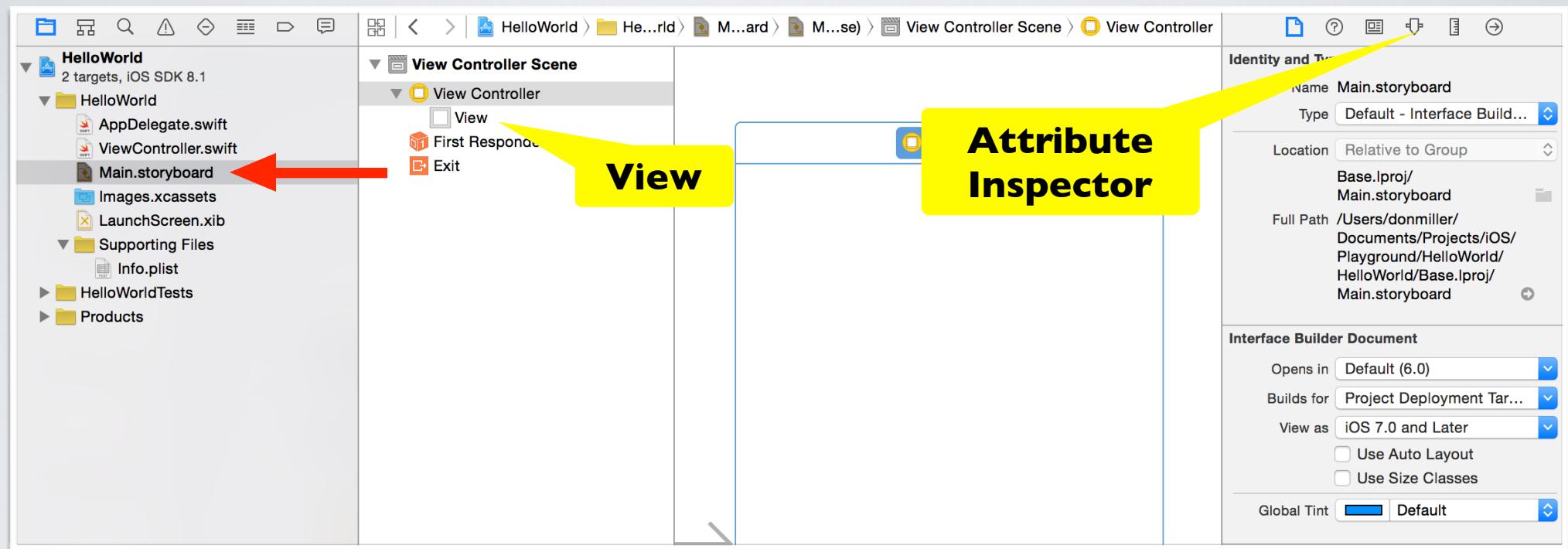


# RUN THE APPLICATION BY PRESSING PLAY

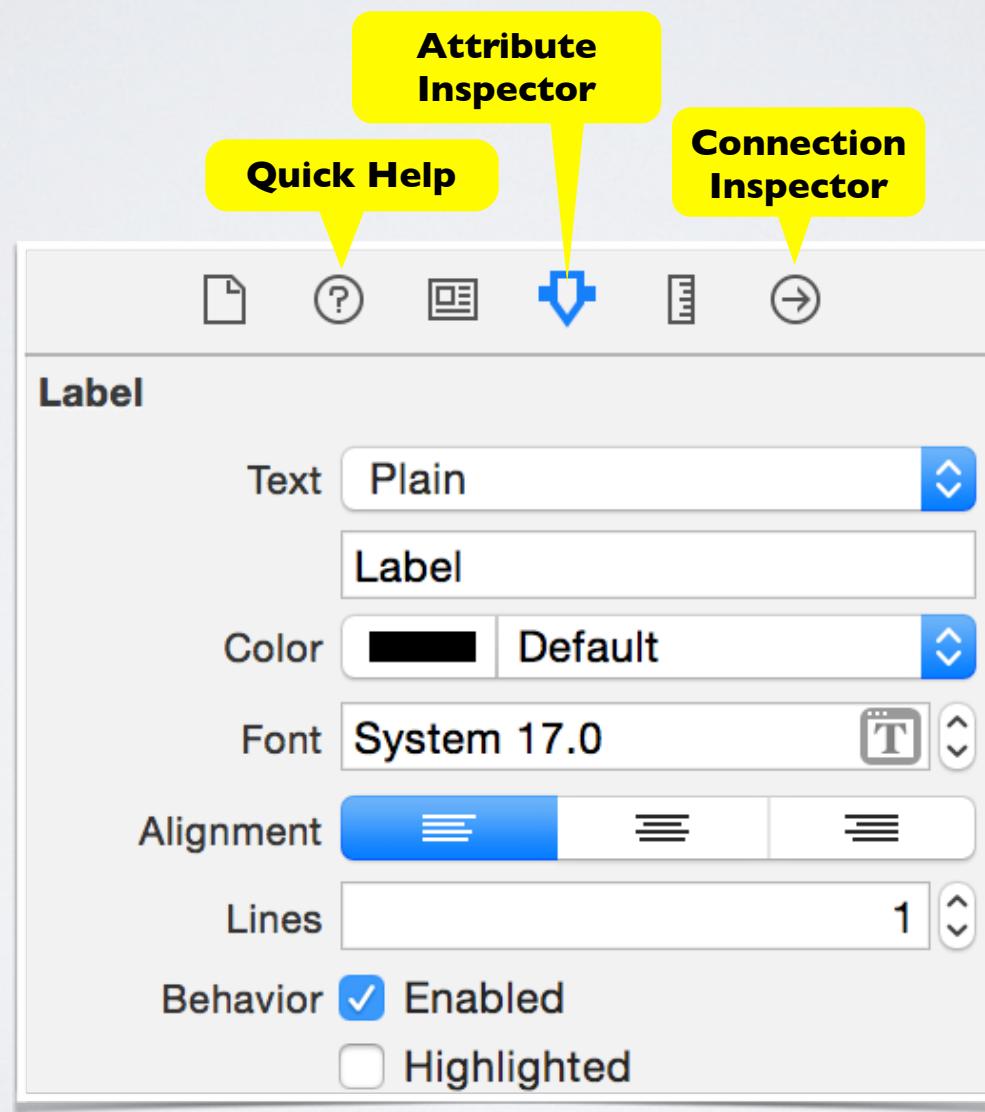


GroundSpeed™  
rapid web + mobile software

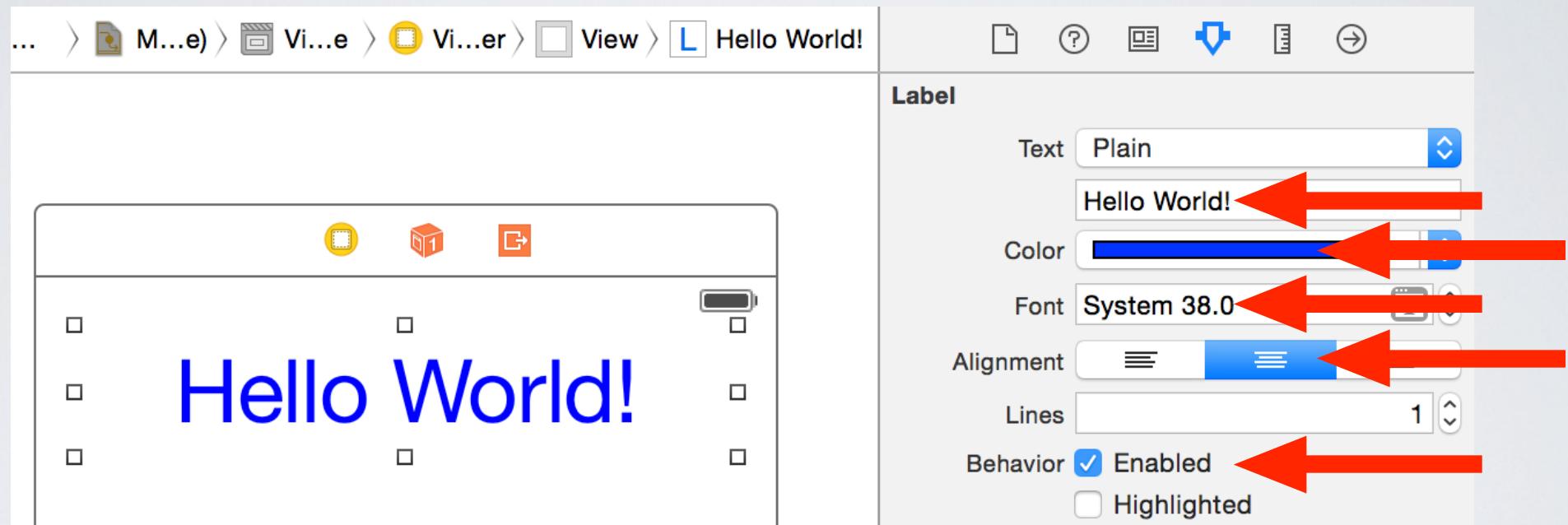
# ATTRIBUTE INSPECTOR



# INSPECTORS



# CREATE “HELLO WORLD!” LABEL



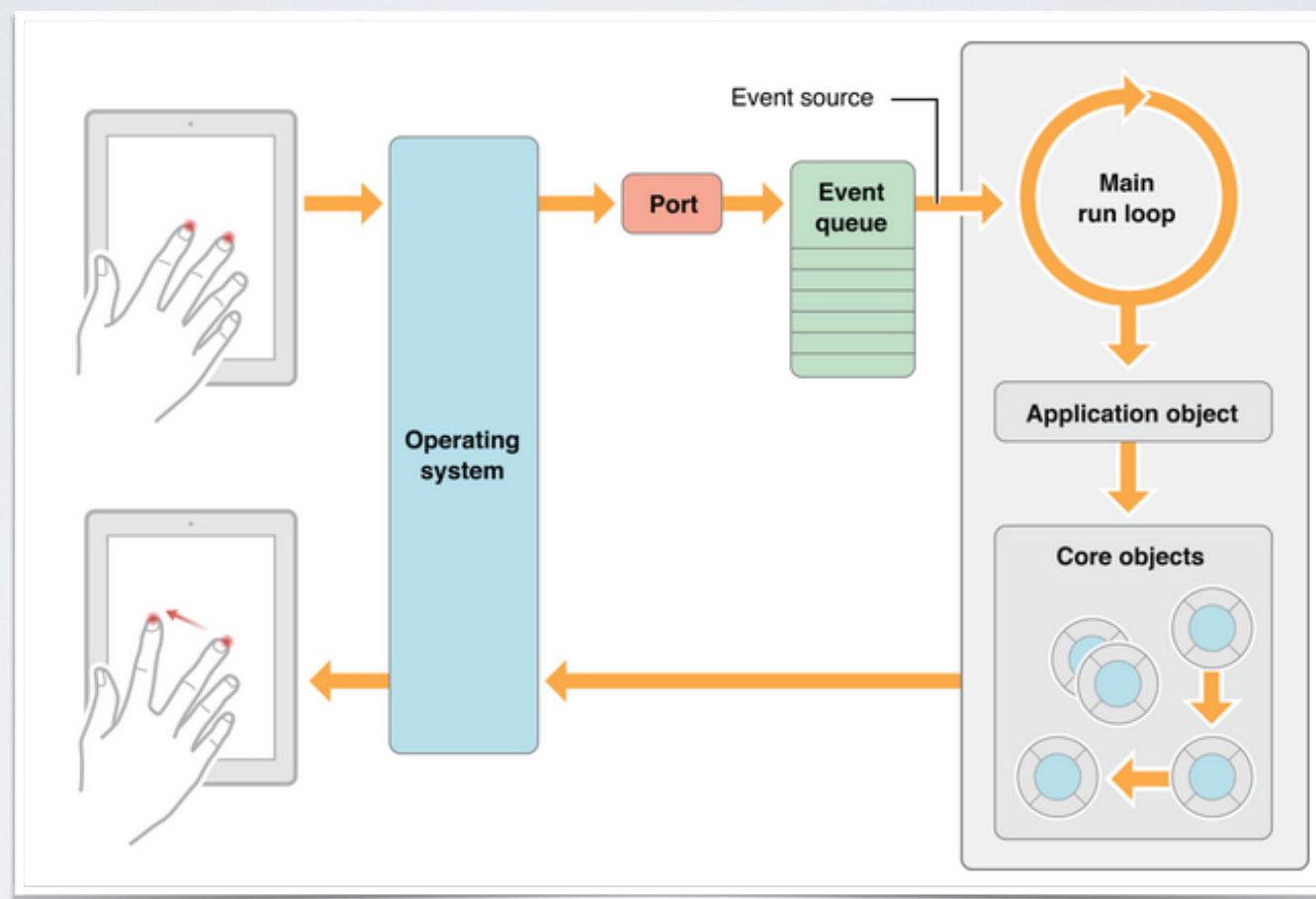
# Hello World!

# Demo



GroundSpeed™  
rapid web + mobile software

# IOS APPLICATION RUNTIME CYCLE

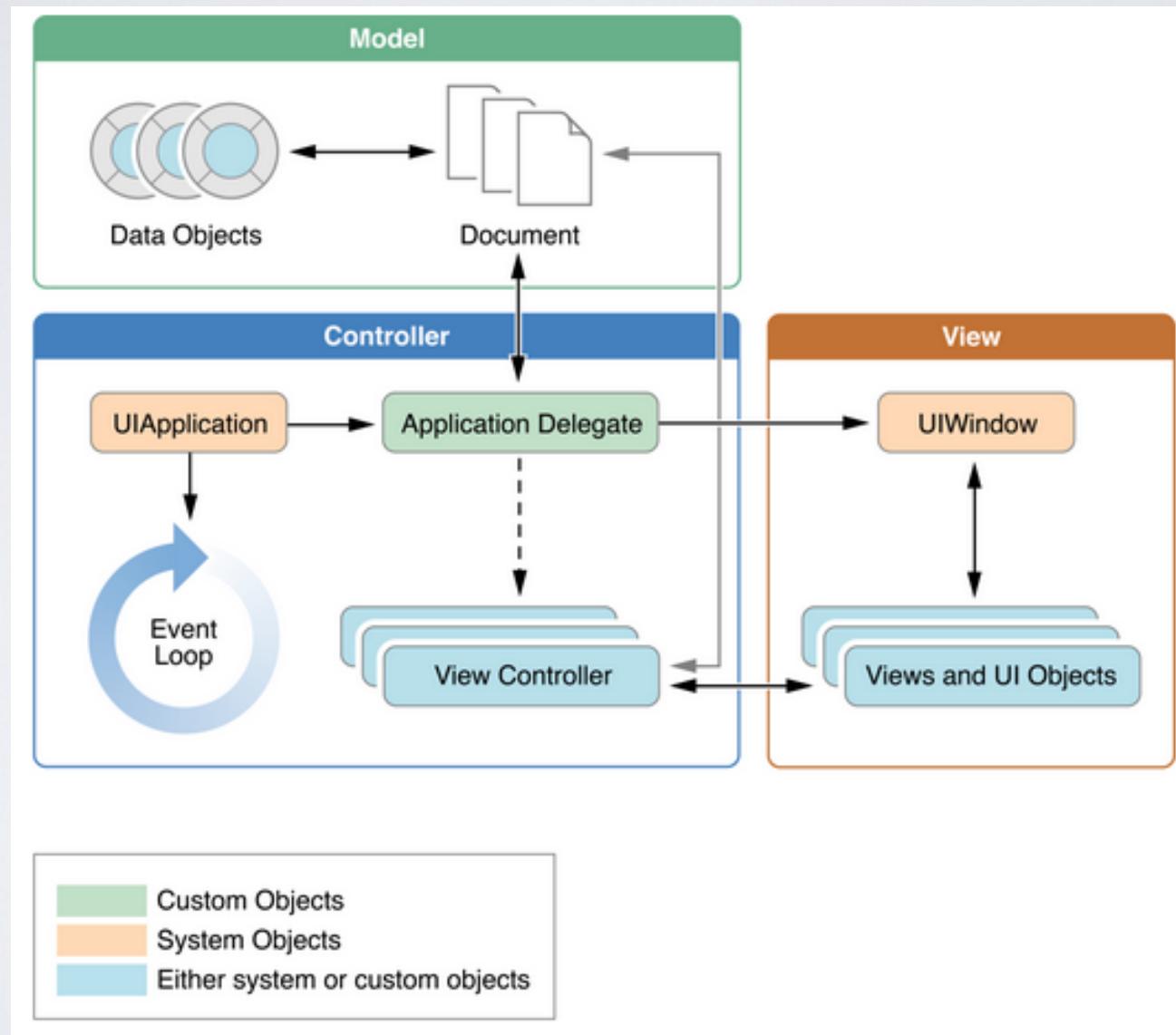


# COMMON TYPES OF EVENTS

Event type	Delivered to...	Notes
Touch	The view object in which the event occurred	Views are responder objects. Any touch events not handled by the view are forwarded down the responder chain for processing.
Remote control Shake motion events	First <a href="#">responder object</a>	Remote control events are for controlling media playback and are generated by headphones and other accessories.
Accelerometer Magnetometer Gyroscope	The object you designate	Events related to the accelerometer, magnetometer, and gyroscope hardware are delivered to the object you designate.
Location	The object you designate	You register to receive location events using the Core Location framework. For more information about using Core Location, see <a href="#">Location and Maps Programming Guide</a> .
Redraw	The view that needs the update	Redraw events do not involve an event object but are simply calls to the view to draw itself. The drawing architecture for iOS is described in <a href="#">Drawing and Printing Guide for iOS</a> .



# MODEL - VIEW - CONTROLLER



# MODEL - VIEW - CONTROLLER

**The Model:** The M in MVC stands for the Model: For the Model, we will need to develop an independent class (or a set of classes) that will serve the data requirements and standard business procedures.

**The View:** In this context the View is the UI screen's layout (the canvas). Here, we configure many User Interface items (also known as UI controls on the screen).

**The Controller:** This is a class that connects the view with the model. In its simplest form, it contains the name of the UIControls (that would be used in the app), and it also includes the signatures and the detail implementations of event handlers.

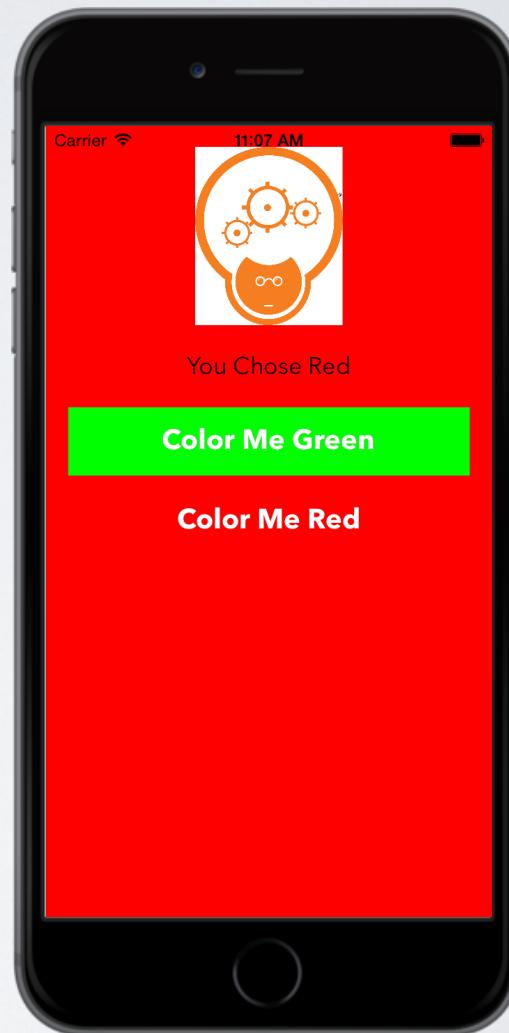
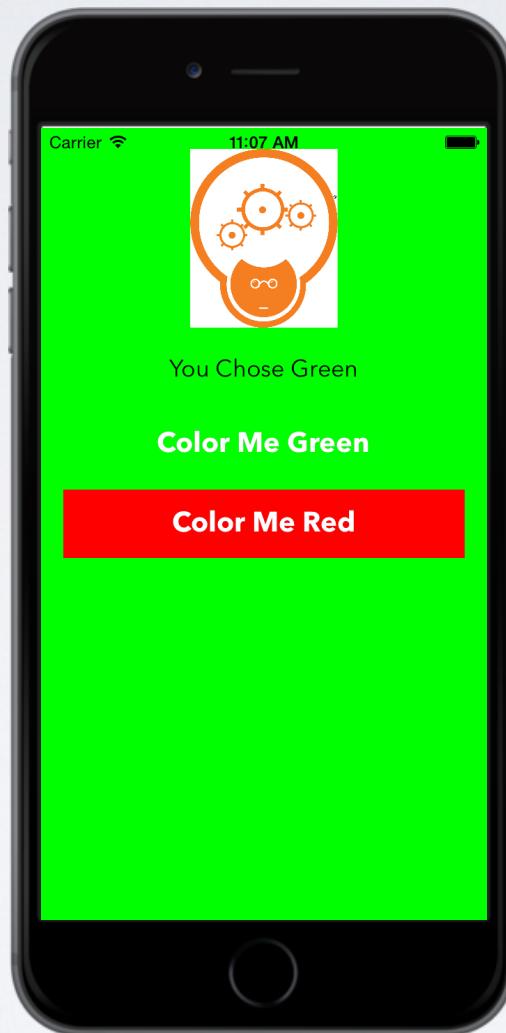
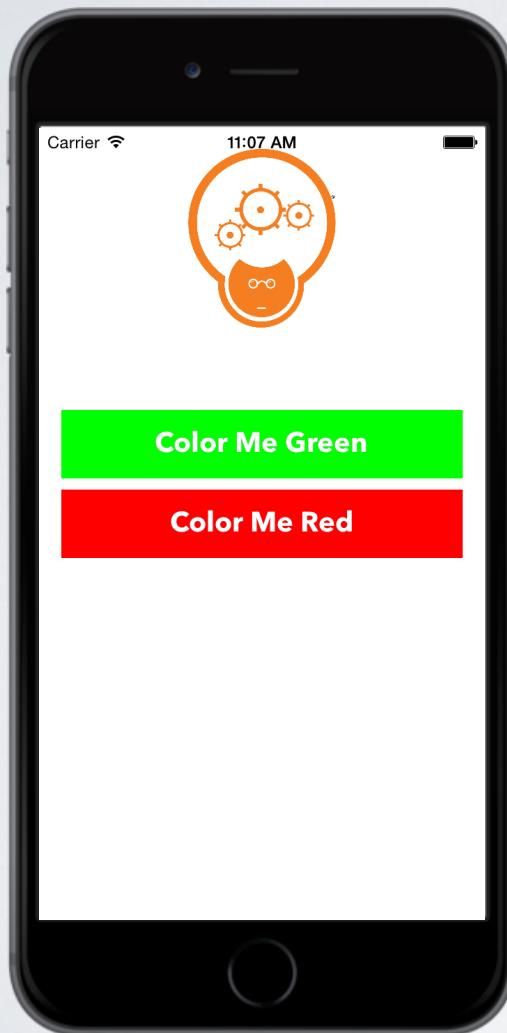


# Need a Break?



GroundSpeed™  
rapid web + mobile software

# ANOTHER QUICK DEMO - [COLOR ME]



GroundSpeed™  
rapid web + mobile software

# CREATE A SINGLE VIEW APPLICATION

Choose a template for your new project:

iOS

- Application
- Framework & Library
- Other

OS X

- Application
- Framework & Library
- System Plug-in
- Other

Master-Detail Application    Page-Based Application    Single View Application    Tabbed Application

Game

**Single View Application**

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a storyboard or nib file that contains the view.

Cancel    Previous    Next

Choose options for your new project:

Product Name: ColorMe

Organization Name: GroundSpeed

Organization Identifier: com.goundspeedhq.ColorMe

Bundle Identifier: com.goundspeedhq.ColorMe

Language: Swift

Devices: iPhone

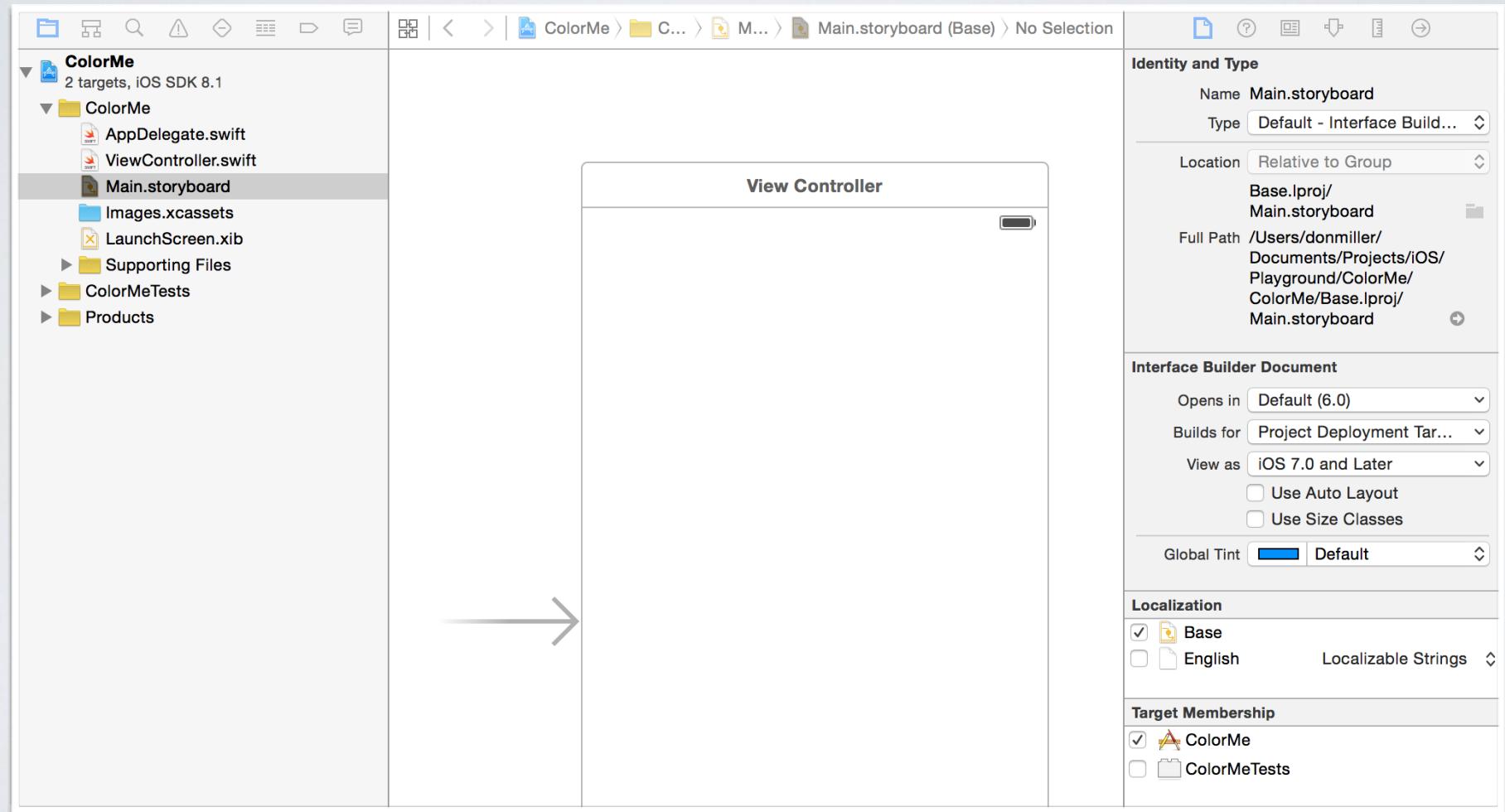
Use Core Data

Cancel    Previous    Next

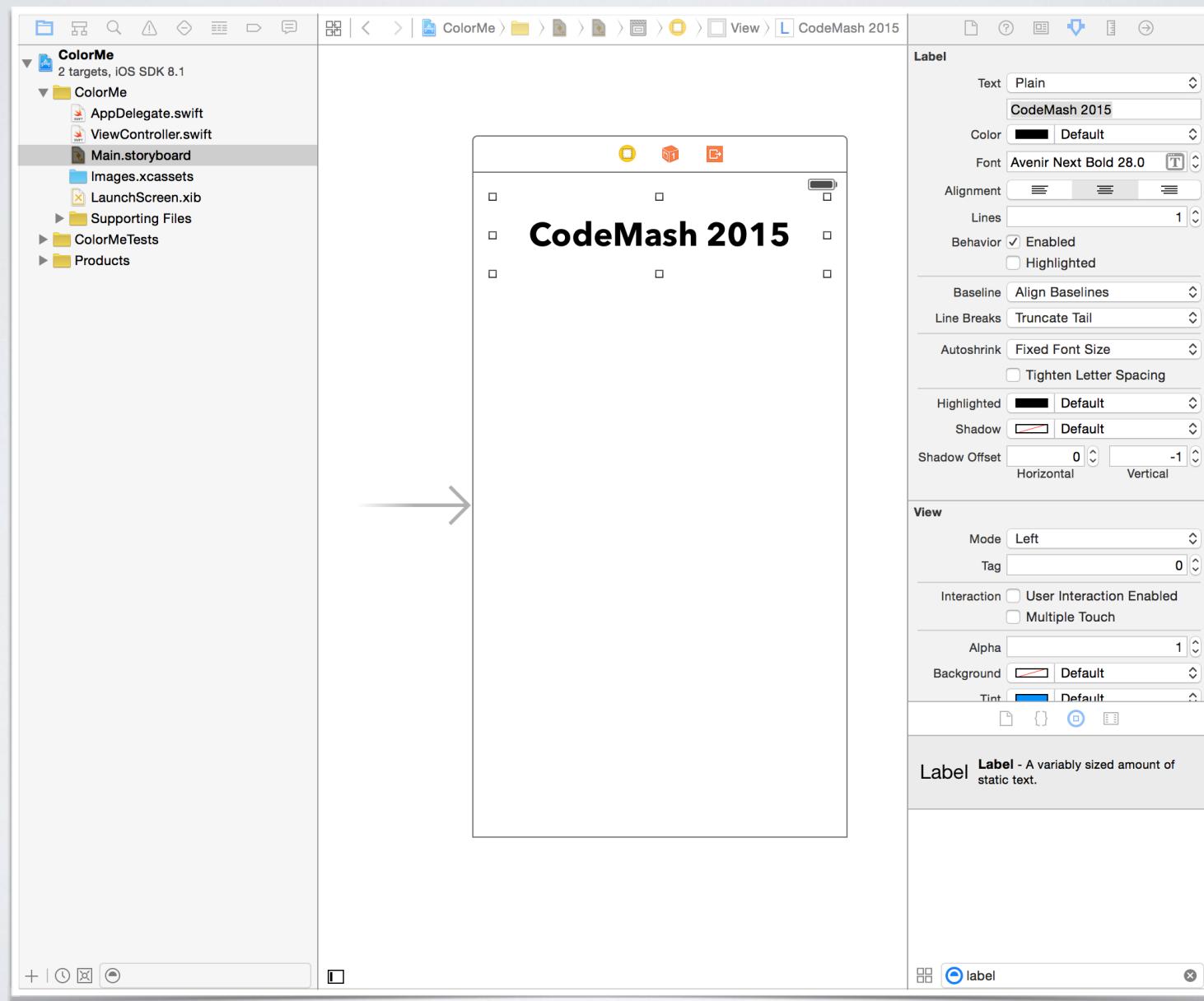


GroundSpeed™  
rapid web + mobile software

# CONFIGURE STORYBOARD BY DISABLING AUTO LAYOUT AND SIZE CLASSES

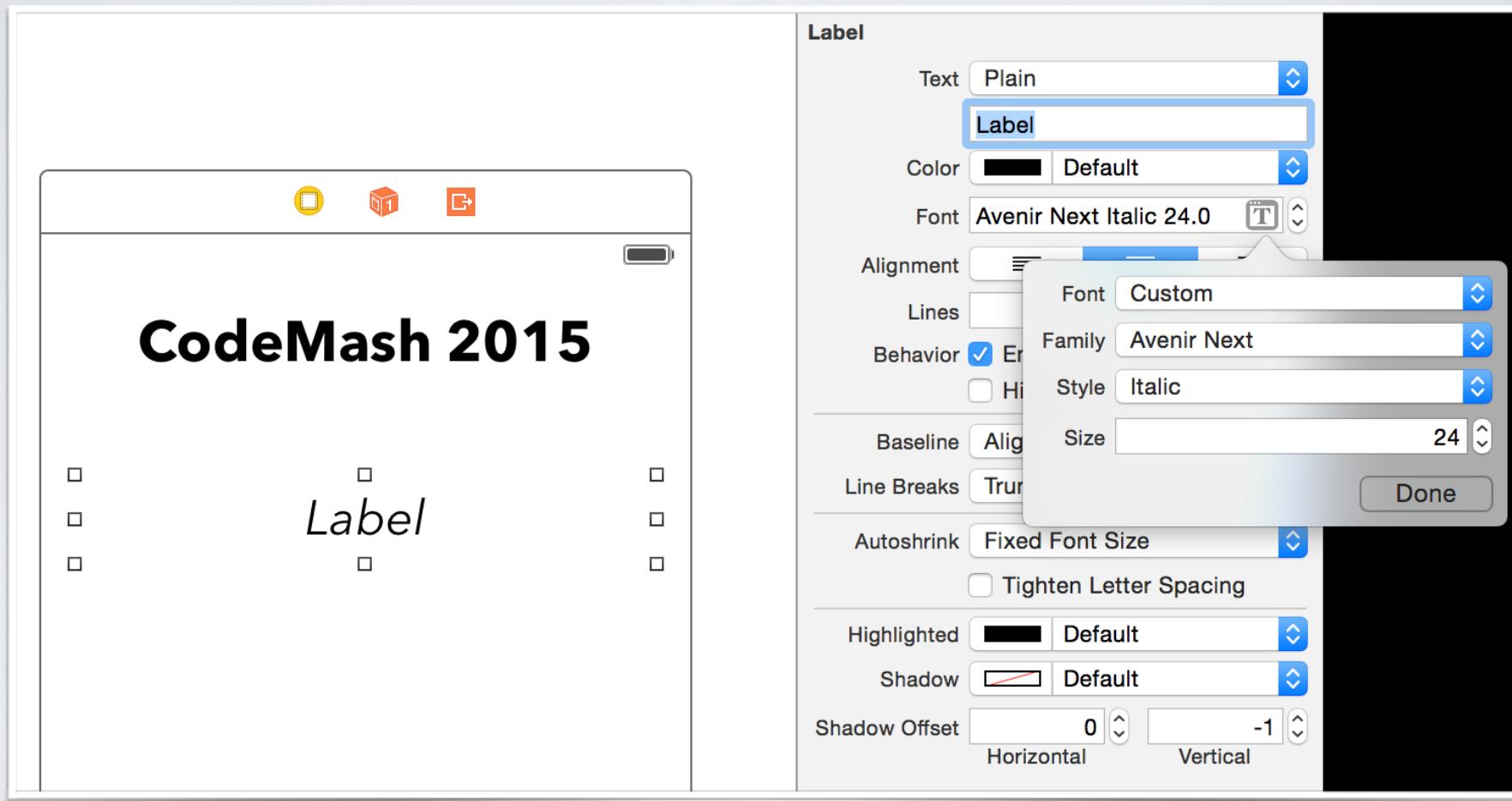


# DRAG A LABEL TO THE STORYBOARD

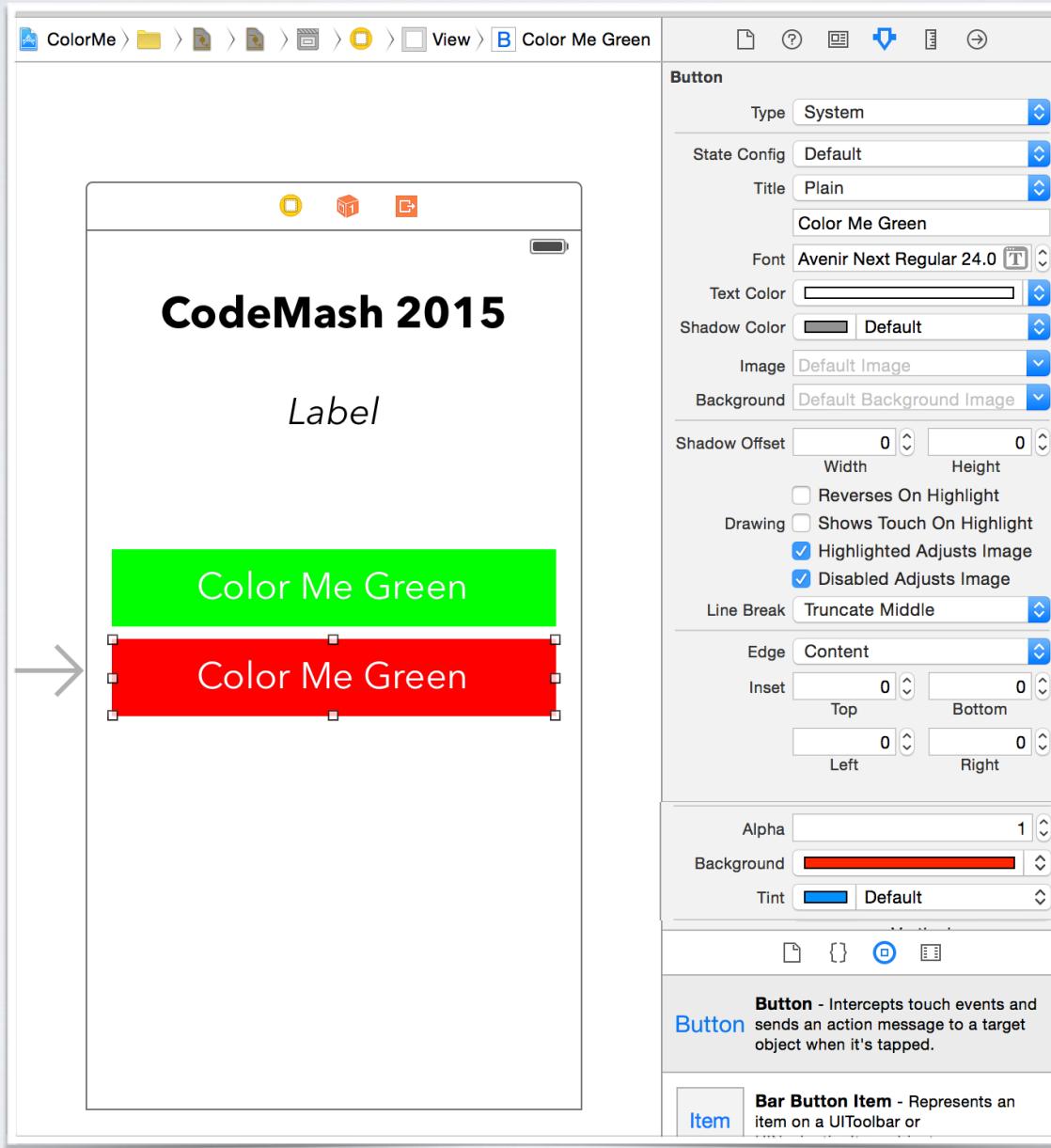


GroundSpeed™  
rapid web + mobile software

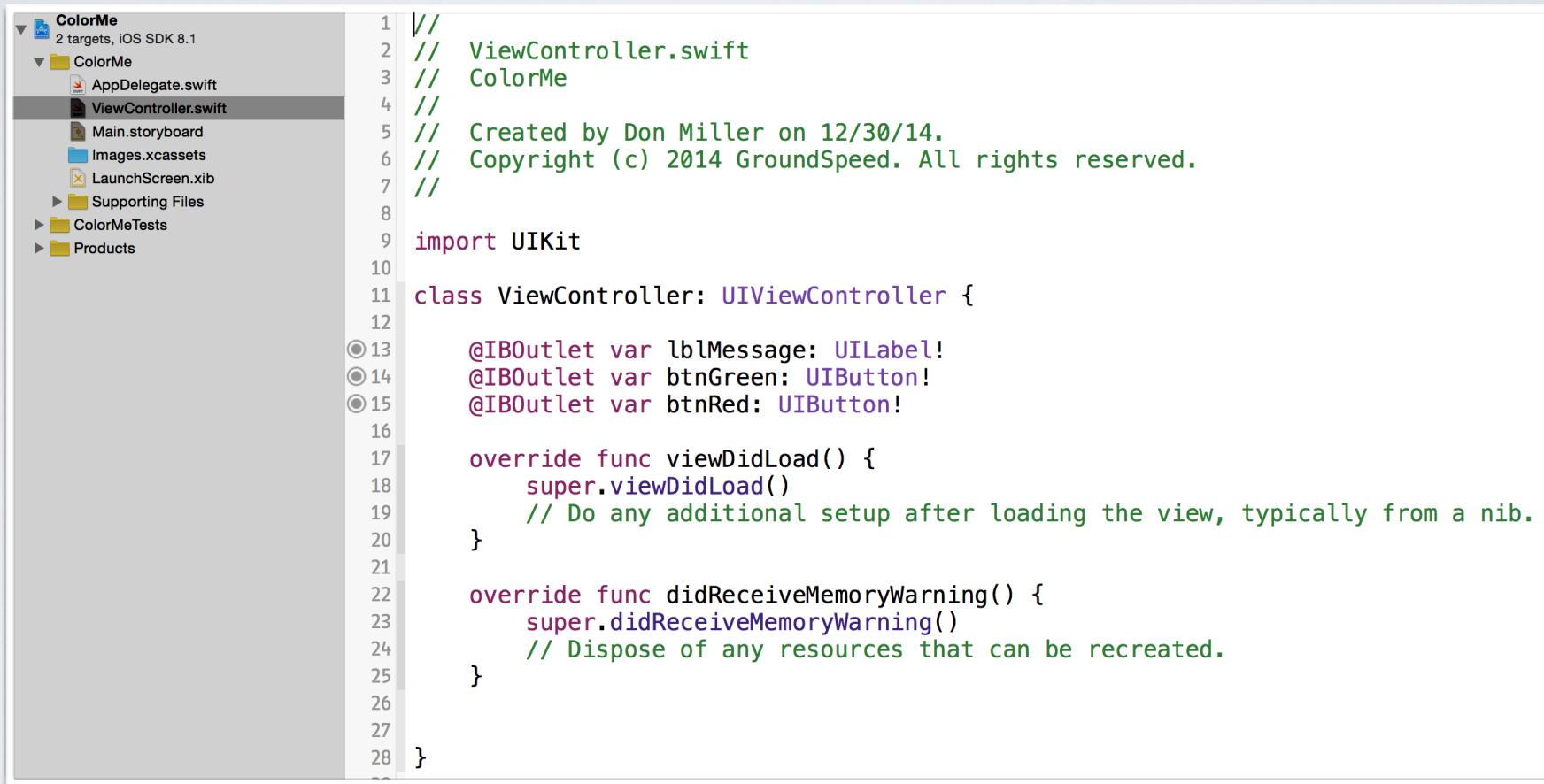
# CREATE A BLANK LABEL FOR OUTPUT



# CREATE TWO BUTTONS



# ADD OUTLETS TO CONTROLLER



The screenshot shows the Xcode project structure for 'ColorMe' with 'ViewController.swift' selected. The code editor displays the following Swift code:

```
1 // ViewController.swift
2 // ColorMe
3 // Created by Don Miller on 12/30/14.
4 // Copyright (c) 2014 GroundSpeed. All rights reserved.
5 //
6
7 import UIKit
8
9 class ViewController: UIViewController {
10
11     @IBOutlet var lblMessage: UILabel!
12
13     @IBOutlet var btnGreen: UIButton!
14
15     @IBOutlet var btnRed: UIButton!
16
17     override func viewDidLoad() {
18         super.viewDidLoad()
19         // Do any additional setup after loading the view, typically from a nib.
20     }
21
22     override func didReceiveMemoryWarning() {
23         super.didReceiveMemoryWarning()
24         // Dispose of any resources that can be recreated.
25     }
26
27
28 }
```

The code editor highlights three lines of code at the top of the class definition, which correspond to the three outlets added in the storyboard.



# ADD ACTIONS TO CONTROLLER AND UPDATE VIEWDIDLLOAD



The screenshot shows the Xcode interface with the project 'ColorMe' selected. The 'ViewController.swift' file is open in the editor. The code displays the implementation of the `viewDidLoad()` method and two custom actions: `displayGreen()` and `displayRed()`. The `viewDidLoad()` method sets the initial message text and background color. The custom actions update the message text and change the view's background color.

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
    lblMessage.text = ""
}

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

@IBAction func displayGreen(sender: AnyObject) {
    lblMessage.text = "You wanted green"
    self.view.backgroundColor = UIColor.greenColor()
}

@IBAction func displayRed(sender: AnyObject) {
    lblMessage.text = "You wanted red"
    self.view.backgroundColor = UIColor.redColor()
}
```



# CONNECTING THE OUTLETS AND ACTIONS

The screenshot shows a storyboard scene with the following elements:

- A label with the text "Label".
- A green button with the text "Color Me Green".
- A red button with the text "Color Me Green".

A connection line is being drawn from the "Touch Up Inside" action of the green button to the "displayGreen:" outlet of the view controller. The connection is currently highlighted.

The right side of the interface shows the "Connections Inspector" with the following sections and items:

- Triggered Segues**: action (radio button)
- Outlet Collections**: gestureRecognizers (radio button)
- Sent Events**:
  - Did End On Exit (radio button)
  - Editing Changed (radio button)
  - Editing Did Begin (radio button)
  - Editing Did End (radio button)
  - Touch Cancel (radio button)
  - Touch Down (radio button)
  - Touch Down Repeat (radio button)
  - Touch Drag Enter (radio button)
  - Touch Drag Exit (radio button)
  - Touch Drag Inside (radio button)
  - Touch Drag Outside (radio button)
  - Touch Up Inside** (radio button) — connected to **\* View Controller displayGreen:**
  - Touch Up Outside (radio button)
  - Value Changed (radio button)
- Referencing Outlets**:
  - btnGreen** (radio button) — connected to **\* View Controller**
  - New Referencing Outlet (radio button)
- Referencing Outlet Collections**: New Referencing Outlet Collection (radio button)



GroundSpeed™  
rapid web + mobile software

# Color Me Demo



GroundSpeed™  
rapid web + mobile software

# SWIFT BASICS

Xcode Playgrounds

Variables & Constants (Mutable and Immutable)

Types

- Strings, Integers, Floats, Booleans

Operators

- Binary and Unary Operators

Collection Types

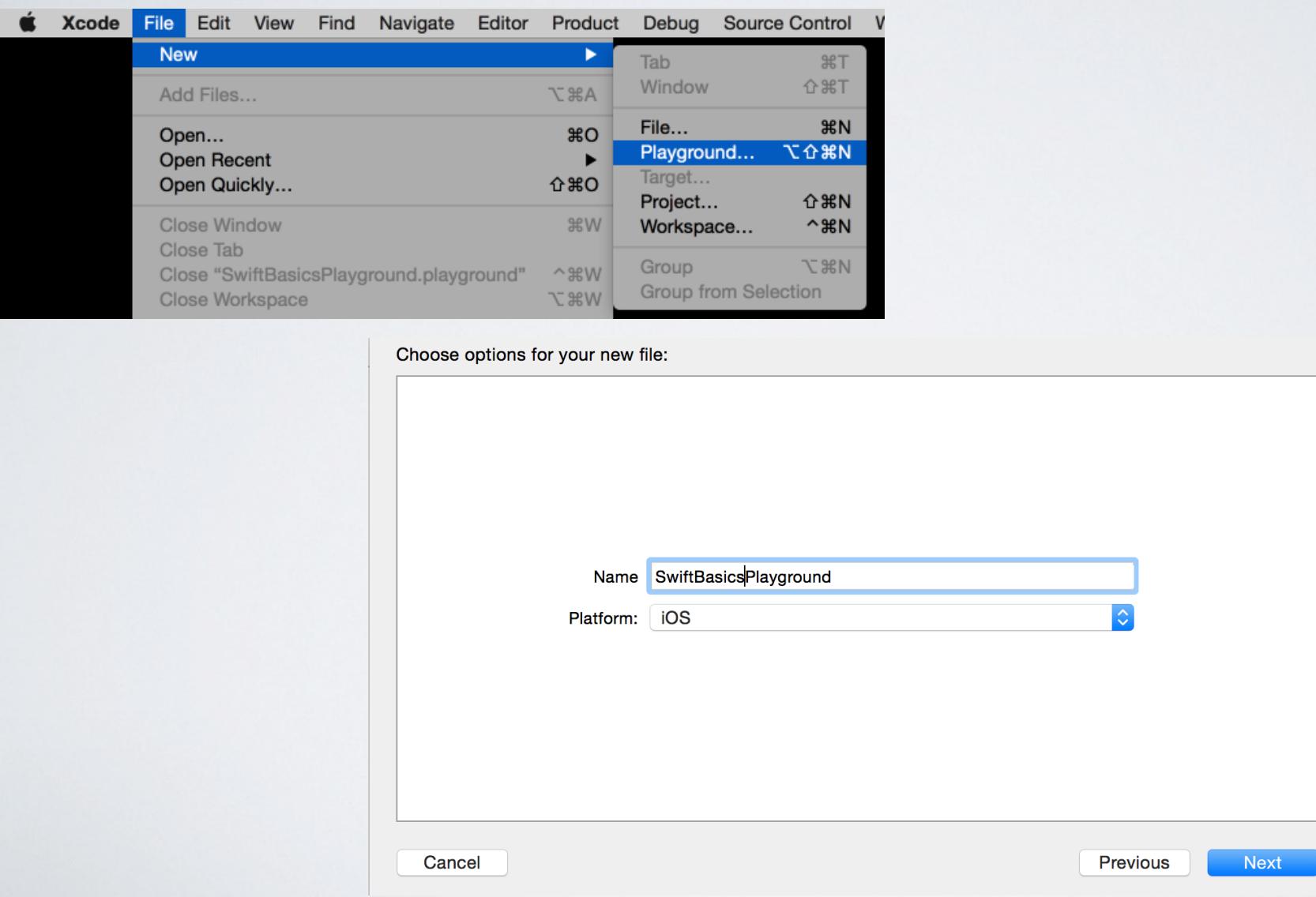
- Arrays and Dictionaries

Control Flow

- For-In, While, For-Condition, If, Switch, Comparison and Logical Operators



# EXAMPLE: HAVE FUN IN THE PLAYGROUND



# ABOUT MUTABLE AND IMMUTABLE OBJECTS OR VARIABLES AND CONSTANTS

```
3 import UIKit
4
5 let immutable = "This cannot be changed"
❶ 6 immutable = "This will give an error"
7
8 var mutable = "This cannot be changed"
9 mutable = "This will NOT give an error"
10
```

"This cannot be changed"

"This cannot be changed"  
"This will NOT give an error"



# DECLARING TYPES: STRINGS, INTEGERS, FLOATS, AND BOOLEANS

5	let string1 = "This is a string"	"This is a string"
6	let string2: String = "This is string typed"	"This is string typed"
7		
8	var intNumber1 = 1	1
9	intNumber1 = 2.5	
10	! Type 'Int' does not conform to protocol 'FloatLiteralConvertible'	
11	let intNumber2: Int = 3	3
12		
13	var floatNumber1 = 1.0	1.0
14	floatNumber1 = 2	2.0
15	floatNumber1 = 2.5	2.5
16		
17	let floatNumber2: Float = 1	1.0
18		
19	let bool1 = true	true
20	let bool2: Bool = false	false
21		



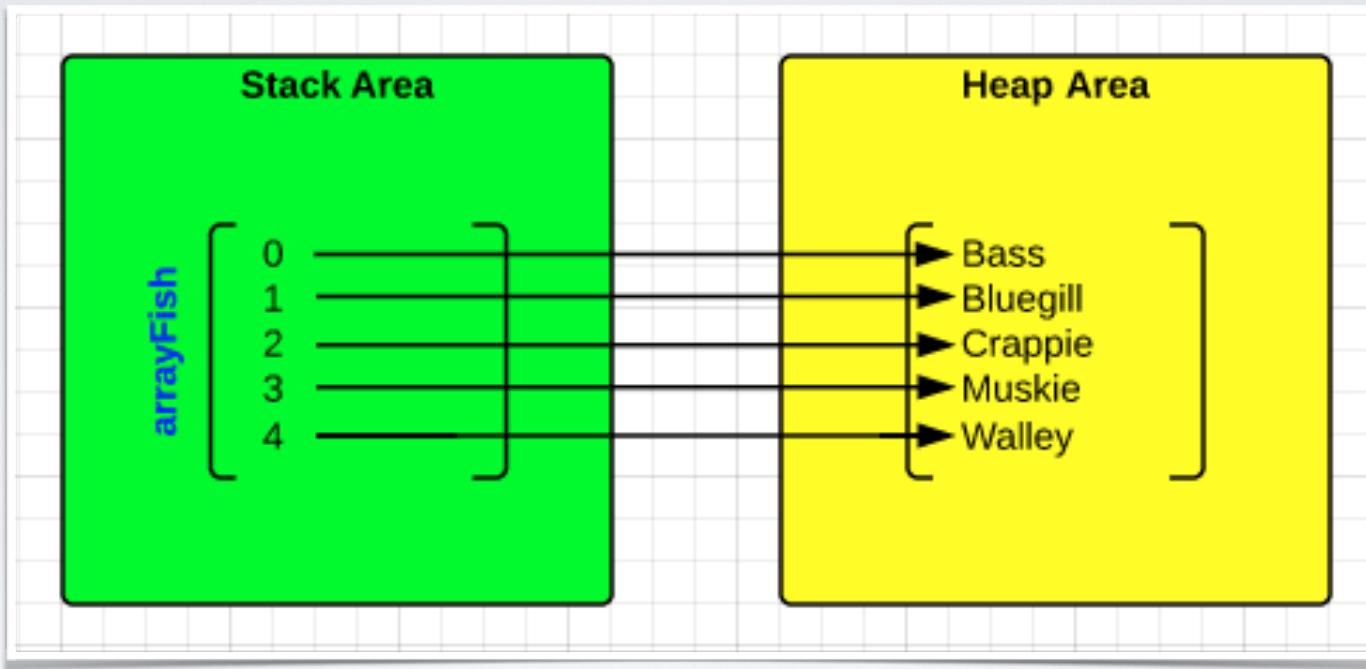
# OPERATORS: BINARY AND UNARY ...AND MAYBE A TERNARY TOO

```
3 import UIKit
4
5 // binary
6 var a = 1
7 a = a + 1
8 a = a - 1
9 a = a / 1
10 a = a * 4
11 a = a % 3
12
13 // unary
14 var b = 1
15 ++b
16 b++
17 b
18 b--
19 b
20
21 // ternary
22 var bool = true
23 var c = (bool ? 1 : 3)
24 bool = false
25 c
26 c = (bool ? 1 : 3)|
```

1	
2	
1	
1	
4	
1	
1	
1	
2	
2	
3	
3	
2	
1	
true	
1	
false	
1	
3	



# ARRAYS



```
5 var arrayFish = ["Bass", "Bluegill",
  "Crappie", "Muskie", "Walley"]
6
7 arrayFish.count
8 arrayFish[3]
! 9 arrayFish[5] ! Execution was interrupted, reason: EXC_...      ["Bass", "Bluegill", "Crappie", "Muskie", "Walley"]
5
"Muskie"
! error
```



# CREATING ARRAYS AND DICTIONARIES WITH SWIFT

```
6 // Arrays
7 var arrayFish = ["Bass", "Bluegill", "Crappie", "Muskie", "Walley"]
8
9 arrayFish.count
10 arrayFish[3]
11
12
13 // Dictionaries
14 var dictStates = ["OH": "Ohio", "MI": "Michigan", "KY": "Kentucky", "IN": "Indiana", "IL": "Illinois", "PA": "Pennsylvania", "NY": "New York"]
15
16 dictStates.count
17 dictStates["OH"]!
18
19 // For Loop through dictionary
20 for (stateCode, stateName) in dictStates {
21     println("\(stateCode): \(stateName)")
22 }
23
24 for stateCode in dictStates.keys {
25     println("\(stateCode)")
26 }
27
28 for stateName in dictStates.values {
29     println("\(stateName)")
30 }
31
32 // Array of stateCodes
33 let stateCodes = [String](dictStates.keys)
```



# CONTROL FLOW - LOOPING

```
5 // For-In
6 for index in 1...5 {
7     println("\(index) times 5 is \(index * 5)")
8 }
9
10 let arrayFish = ["Bass", "Bluegill", "Crappie", "Muskie", "Walley"]
11 for fish in arrayFish {
12     println("Fish array: \(fish)")
13 }
14
15 // For
16 for var index = 0; index < 5; ++index {
17     println("For Index is \(index)")
18 }
19
20 // While
21 var index = 0
22 var max = 5
23 while index < max {
24     println("While Index is \(index++)")
25 }
26
27 // Do-While
28 index = 0
29 max = 5
30 do {
31     println("Do-While Index is \(index++)")
32 } while index < max
```

x Console Output

```
1 times 5 is 5
2 times 5 is 10
3 times 5 is 15
4 times 5 is 20
5 times 5 is 25
Fish array: Bass
Fish array: Bluegill
Fish array: Crappie
Fish array: Muskie
Fish array: Walley
For Index is 0
For Index is 1
For Index is 2
For Index is 3
For Index is 4
While Index is 0
While Index is 1
While Index is 2
While Index is 3
While Index is 4
Do-While Index is 0
Do-While Index is 1
Do-While Index is 2
Do-While Index is 3
Do-While Index is 4
```



# CONTROL FLOW - DECISION

5 // IF	90	
6 var temperatureInFahrenheit = 90		
7 if temperatureInFahrenheit <= 32 {		
8     println("It's very cold. Consider wearing a scarf.")		
9 } else if temperatureInFahrenheit >= 86 {		"It's really warm. Don't forget to wear sunscreen."
10    println("It's really warm. Don't forget to wear sunscreen.")		
11 } else {		
12     println("It's not that cold. Wear a t-shirt.")		
13 }		
14		
15 // SWITCH		
16 let someCharacter: Character = "e"	"e"	
17 switch someCharacter {		
18 case "a", "e", "i", "o", "u":		
19     println("\(someCharacter) is a vowel")		"e is a vowel"
20 case "b", "c", "d", "f", "g", "h", "j", "k", "l", "m",		
21 "n", "p", "q", "r", "s", "t", "v", "w", "x", "y", "z":		
22     println("\(someCharacter) is a consonant")		
23 default:		
24     println("\(someCharacter) is not a vowel or a consonant")		
25 }		



# CLASS ILLUSTRATIONS

```
5 class Person {  
6     var ssn: String!  
7     var name: String!  
8     var city: String!  
9     var state: String!  
10 }  
11  
12 class Employee: Person {  
13     var empId: String!  
14     var department: String!  
15     var healthInsurance: String!  
16  
17     func printChecks() {  
18         println("Printing Checks")  
19     }  
20 }  
21  
22 class SalariedEmployee: Employee {  
23     var salary: String!  
24 }  
25  
26 class HourlyEmployee: Employee {  
27     var hourlyRate: String!  
28 }  
29
```

```
30 let hourlyEmployee = HourlyEmployee()  
31 hourlyEmployee.name = "Don Miller"  
32 hourlyEmployee.empId = "1234"  
33 hourlyEmployee.hourlyRate = "9.99"  
34  
35 var output = "Name: \(hourlyEmployee.name) \r"  
36 output += "Employee ID is \(hourlyEmployee.empId) \r"  
37 output += "Hourly wage is \(hourlyEmployee.hourlyRate) \r"  
38  
39 println("\(output)")  
40 hourlyEmployee.printChecks()  
41
```

## Console Output

```
Name: Don Miller  
Employee ID is 1234  
Hourly wage is 9.99  
Printing Checks
```



# STORYBOARD OVERVIEW

Storyboards are graphic organizers displayed in sequence for the purpose of pre-visualizing a motion picture or animation. The storyboarding process, in the form it is known today, is used by developers to give the designer, developer, and end user an opportunity to see what the application or web site will look like at a high level.

In iOS, a Storyboard is a container of screens that may have embedded view controllers, navigation controllers, and tab bar controllers. It allows the user to graphically connect screens to one another through embedded objects on the screen. These connections are called segues (pronounced seg-wey). The storyboard eliminates the need for xib files and has one file that contains all of your screens and transitions. It is possible to connect two storyboard files with one another; however, unlikely with smaller applications.



# PROS OF STORYBOARDS

## Pros of using Storyboards:

- With a storyboard you have a better conceptual overview of all the screens in your app and the connections between them. It's easier to keep track of everything because the entire design is in a single file, rather than spread out over many separate nibs.
- The storyboard describes the transitions between the various screens. These transitions are called “segues” and you create them by simply ctrl-dragging from one view controller to the next. Thanks to segues you need less code to take care of your UI.
- Storyboards make working with table views a lot easier with the new prototype cells and static cells features. You can design your table views almost completely in the storyboard editor, something else that cuts down on the amount of code you have to write.



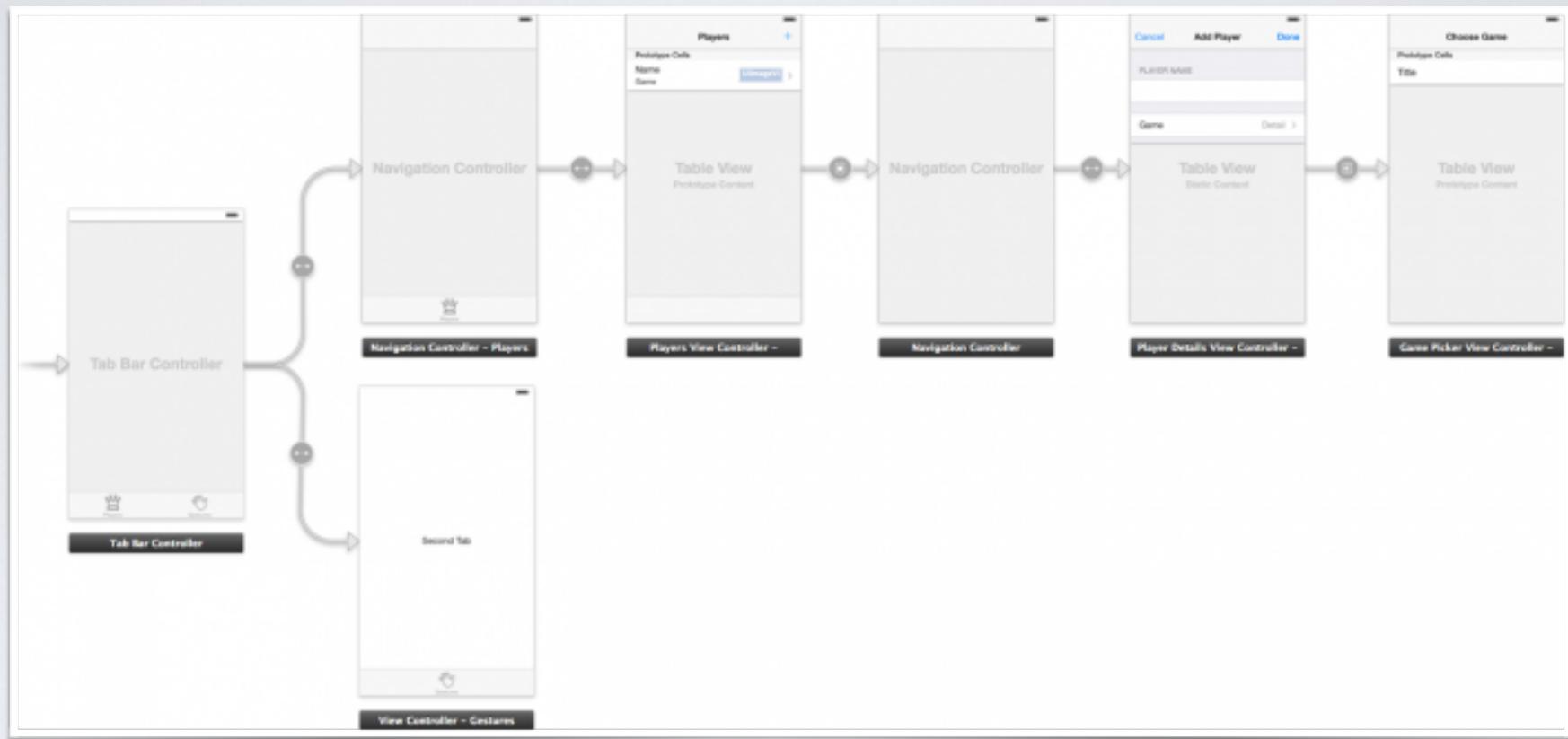
# CONS OF STORYBOARDS

## Cons of using Storyboards:

- It's not easy to work with storyboards in a team, since only one participant can work on the storyboard at once (because it's one binary file).
- If you need to do things storyboard doesn't offer, it's not quite easy to get storyboard mixed with programmatically created views)
- Since all objects are in one file, a larger project could cause the storyboard to be very slow and unresponsive. It loads the entire thing into memory. It loads the entire item into memory.

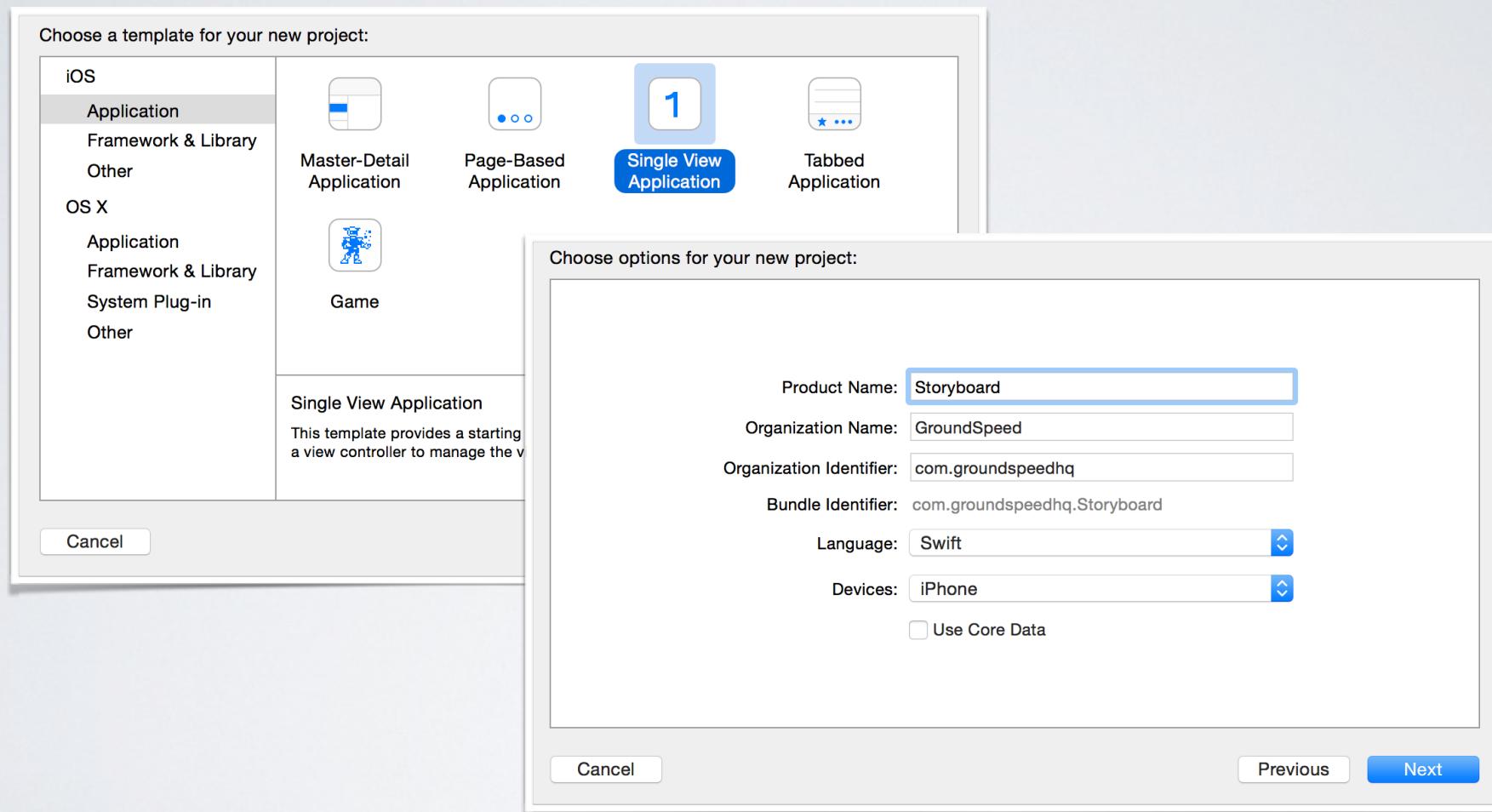


# STORYBOARD OVERVIEW



# STORYBOARD DEMO

## STEP 1: CREATE A NEW SINGLE VIEW APPLICATION



GroundSpeed™  
rapid web + mobile software

# STEP 2: OBSERVE THE SUMMARY PAGE AND TURN OFF SIZE CLASSES

The screenshot shows the Xcode storyboard summary page with two main sections: **Identity** and **Deployment Info**.

**Identity Section:**

- Bundle Identifier: com.goundspeedhq.Storyboard
- Version: 1.0
- Build: 1
- Team: None

**Deployment Info Section:**

- Deployment Target: 8.1
- Devices: iPhone
- Main Interface: Main
- Device Orientation:
  - Portrait
  - Upside Down
  - Landscape Left
  - Landscape Right
- Status Bar Style: Default
- Hide status bar

A red arrow points to the **Interface Builder Document** section of the right-hand panel, specifically highlighting the **Use Size Classes** checkbox.

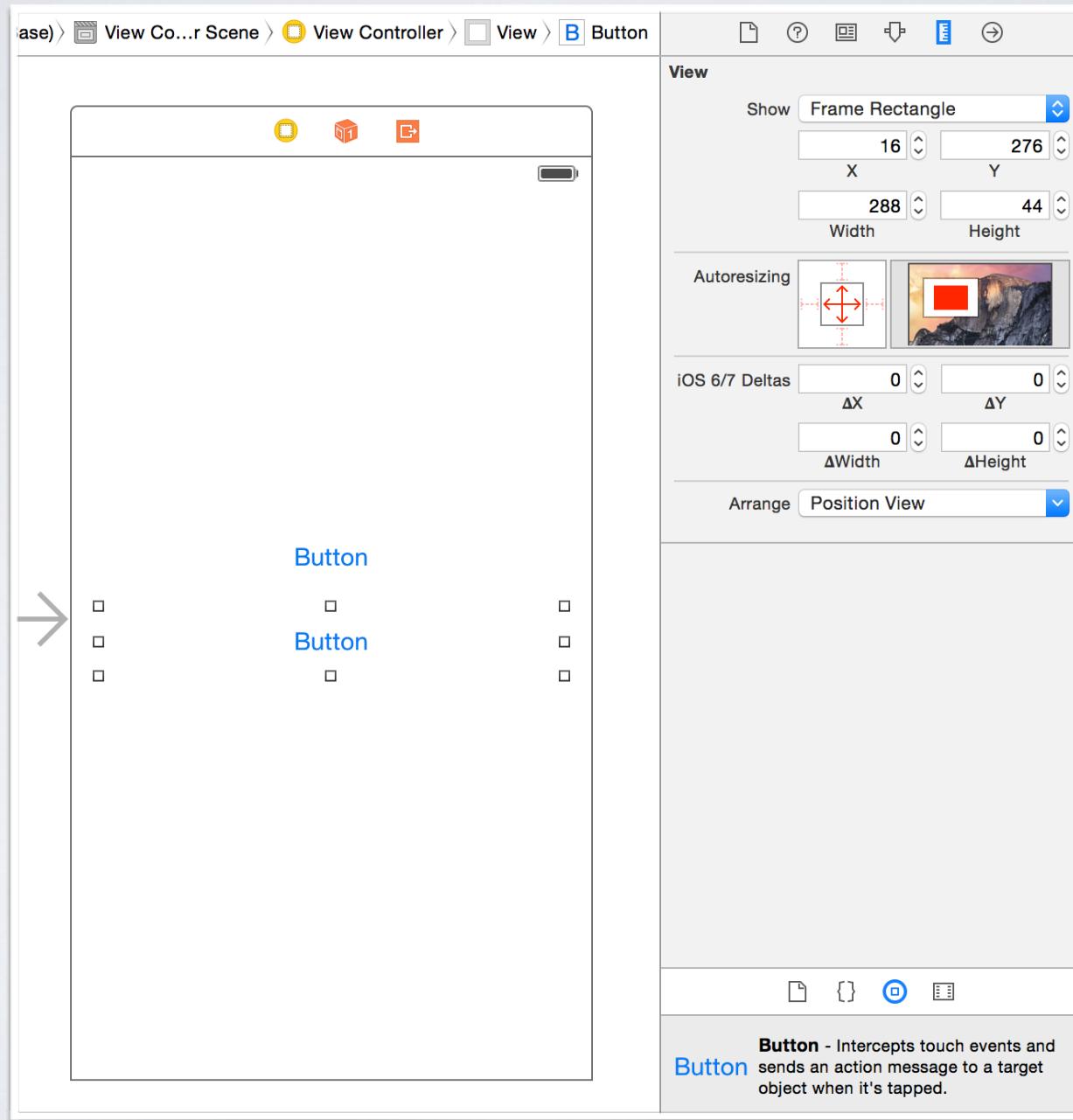
**Interface Builder Document Section:**

- Name: Main.storyboard
- Type: Default - Interface Builder Document
- Location: Relative to Group
- Base.Iproj/  
Main.storyboard
- Full Path: /Users/donmiller/  
Documents/Projects/iOS/  
Playground/HellWorld/  
HelloWorld/Base.Iproj/  
Main.storyboard
- Opens in: Default (6.0)
- Builds for: Project Deployment Tar...
- View as: iOS 7.0 and Later
  - Use Auto Layout
  - Use Size Classes
- Global Tint: Default

**GroundSpeed™** logo and text: rapid web + mobile software

# STEP 3: ADD 2 BUTTONS TO THE VIEW CONTROLLER

# STEP 4: LET'S MAKE THEM THE SAME WIDTH IN SIZE INSPECTOR



# STEP 5: CHANGE THE BUTTON LABELS TEXT TO RED AND GREEN

The image shows a user interface editor with a modal window and its corresponding properties panel.

**Modal Window Content:**

- A large red button labeled "Red Push".
- A green button labeled "Green Modal".

**Properties Panel (Button settings):**

- Type: System
- State Config: Default
- Title: Plain
- Font: System 15.0
- Text Color: Green
- Shadow Color: Default
- Image: Default Image
- Background: Default Background Image
- Shadow Offset: 0 (Width), 0 (Height)
- Checkboxes:
  - Reverses On Highlight
  - Shows Touch On Highlight
  - Highlighted Adjusts Image (checked)
  - Disabled Adjusts Image (checked)
- Line Break: Truncate Middle
- Edge: Content
- Inset:
  - Top: 0
  - Bottom: 0
  - Left: 0
  - Right: 0

**Control:**

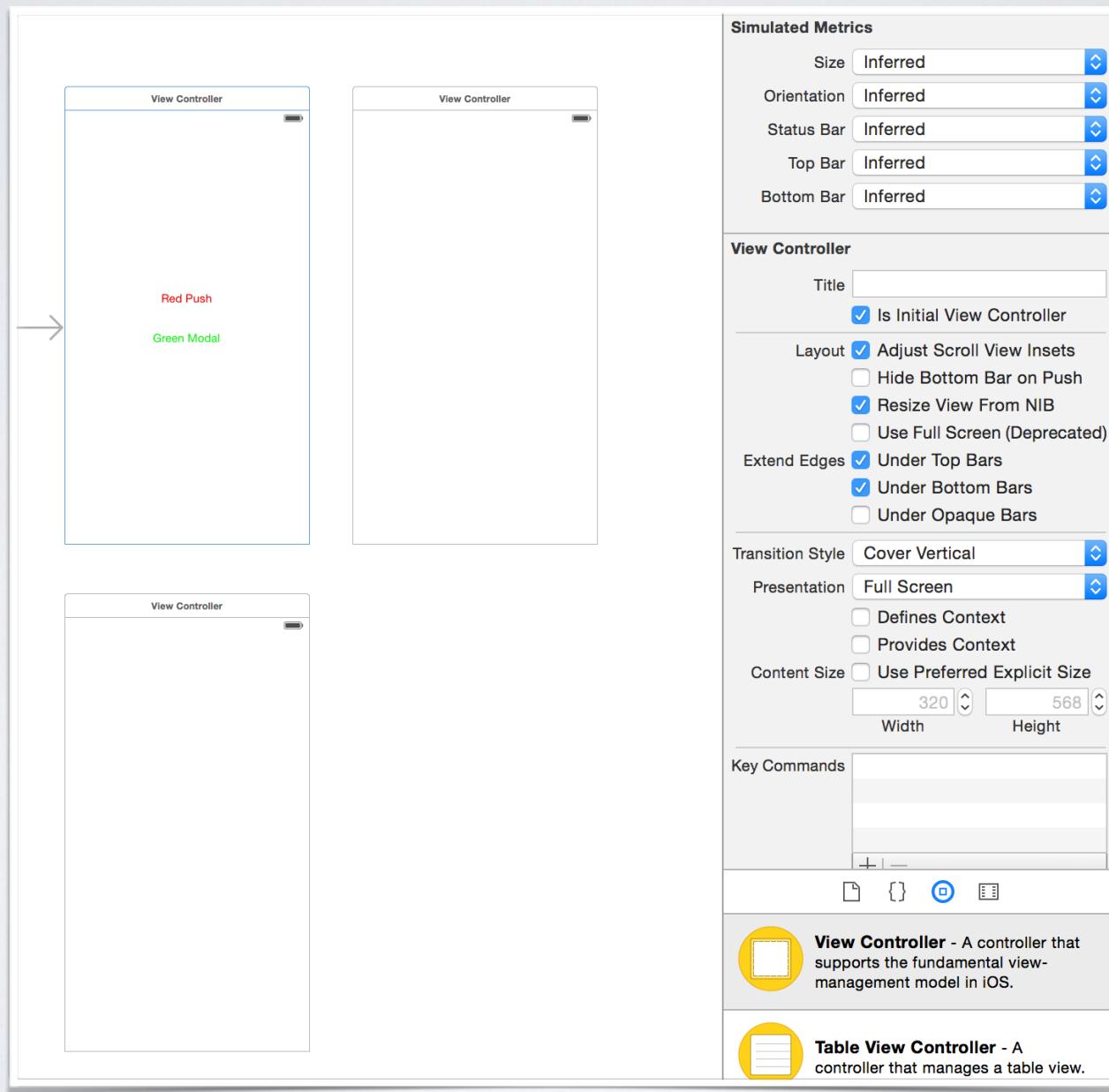
- Alignment:
  - Horizontal
  - Vertical

**Description:**

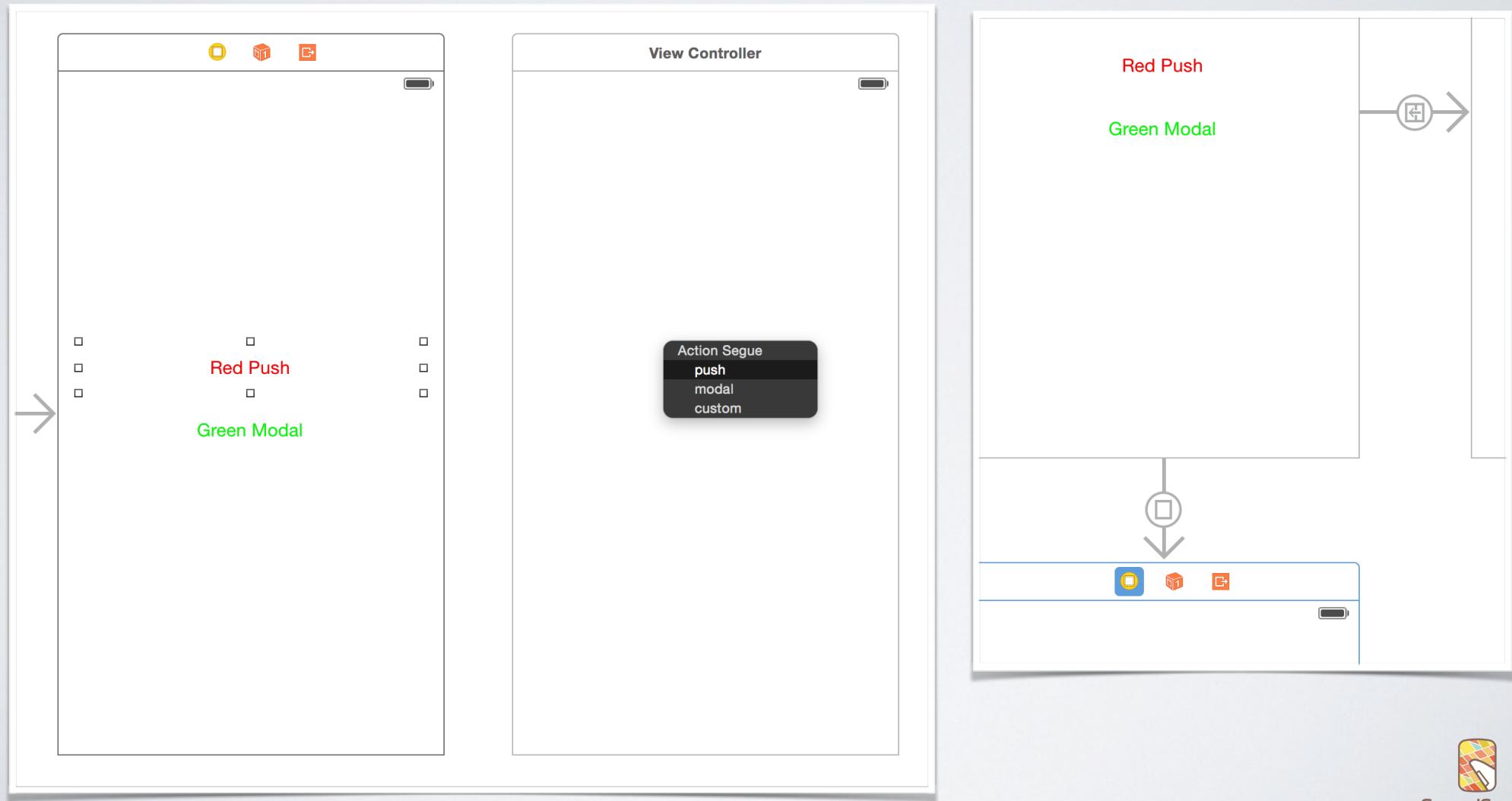
**Button** - Intercepts touch events and sends an action message to a target object when it's tapped.



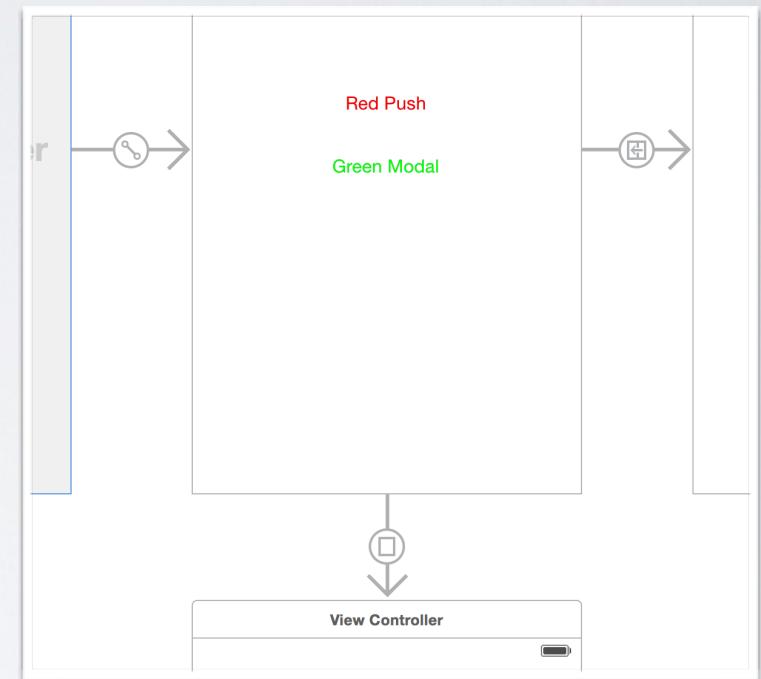
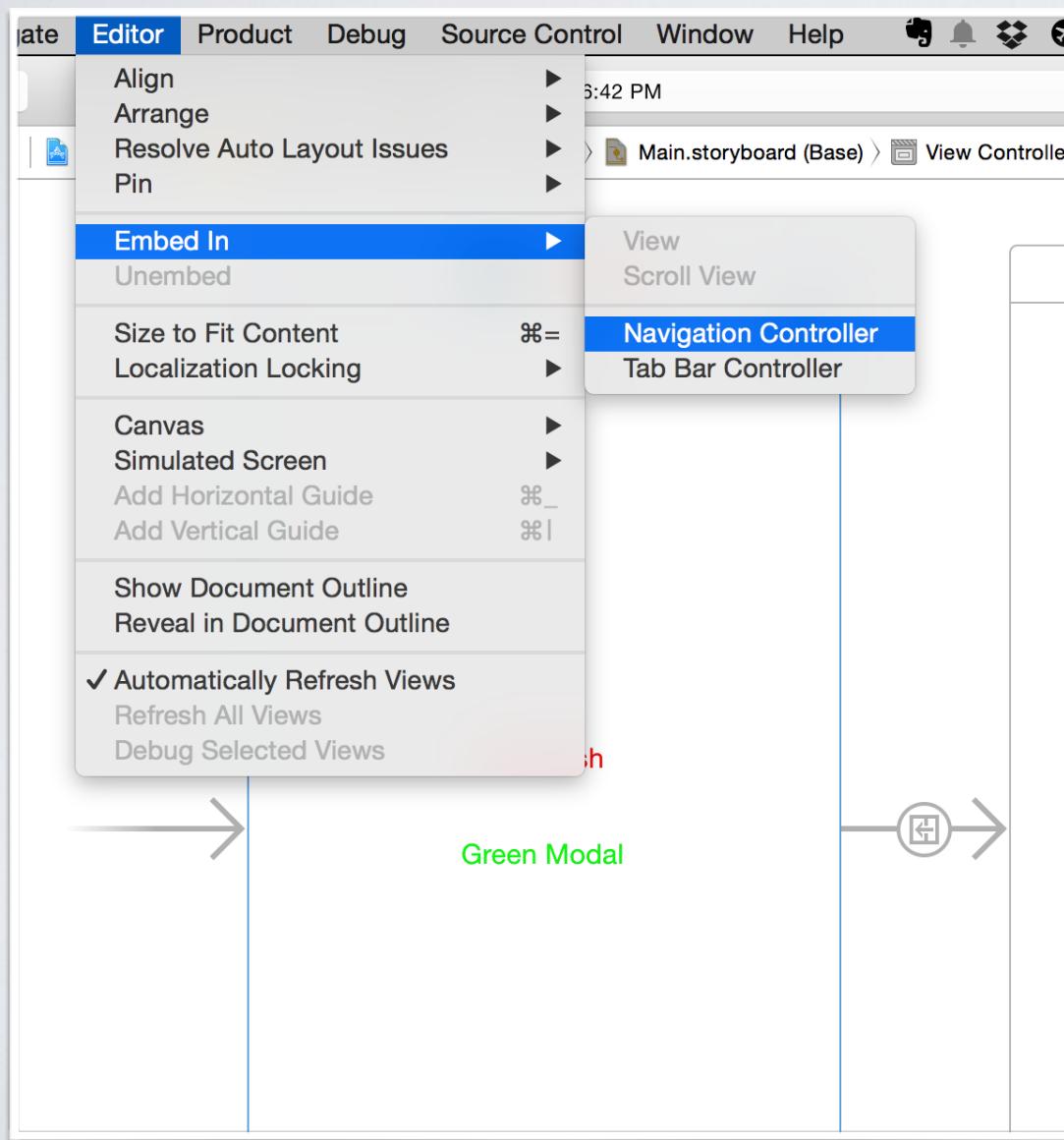
# STEP 6: ADD TWO VIEW CONTROLLERS TO THE STORYBOARD BY DRAGGING THEM OVER



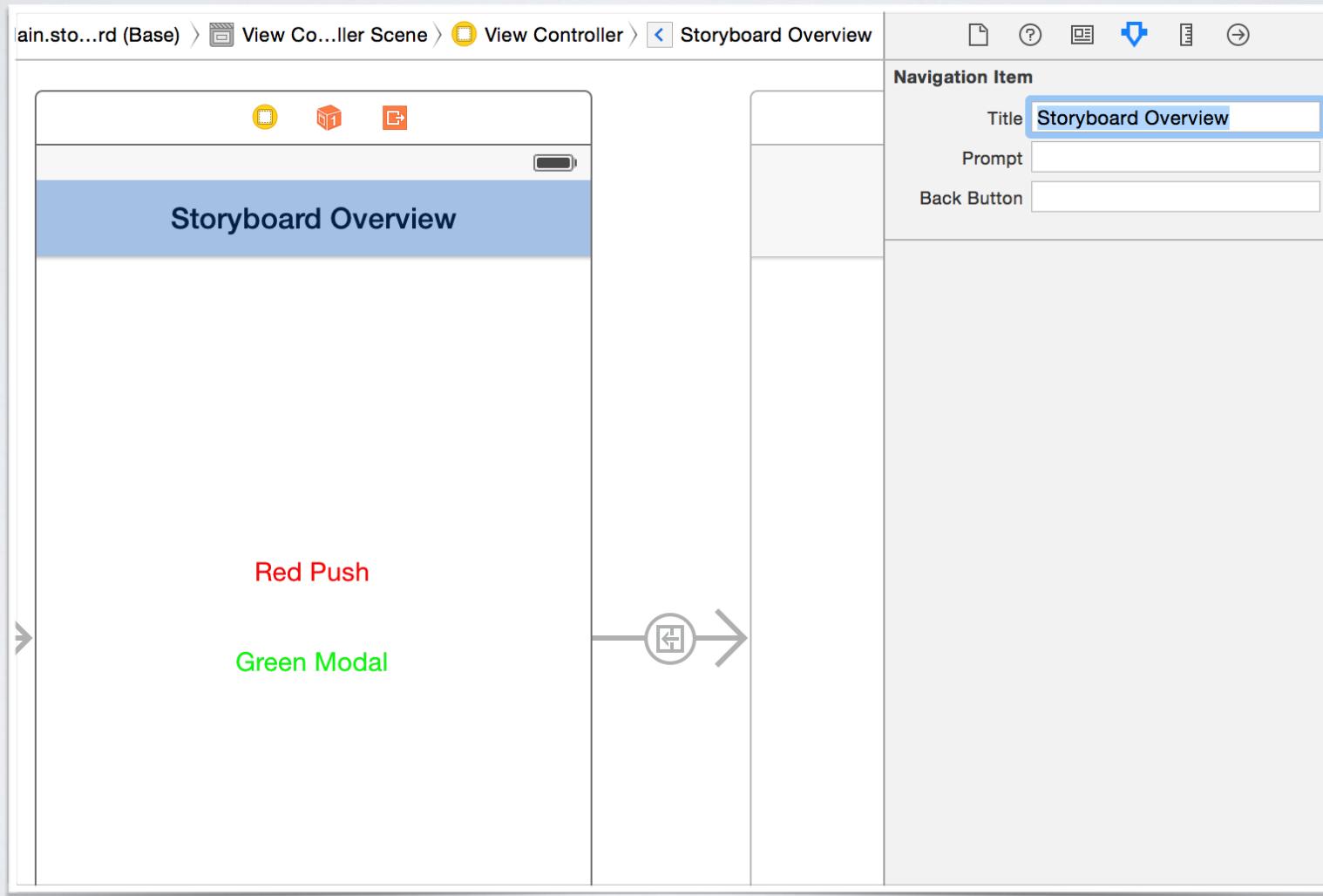
# STEP 7: CONNECT THE BUTTONS TO THE NEW VIEW CONTROLLERS BY HOLDING DOWN THE [CONTROL] KEY AND DRAGGING FROM THE BUTTON TO THE NEW VIEW CONTROLLER



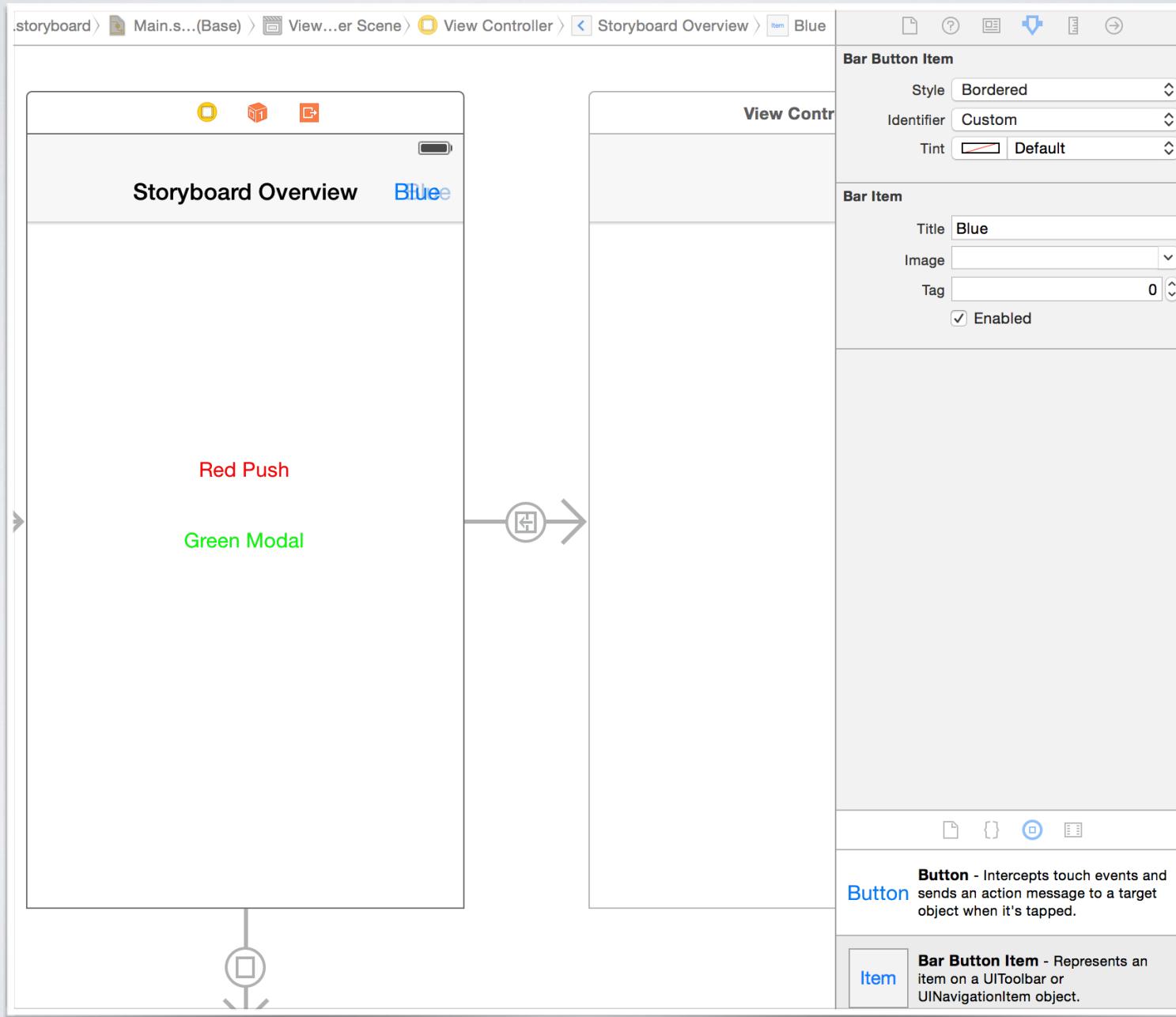
# STEP 8: EMBED A NAVIGATION CONTROLLER ON YOUR PRIMARY VIEW



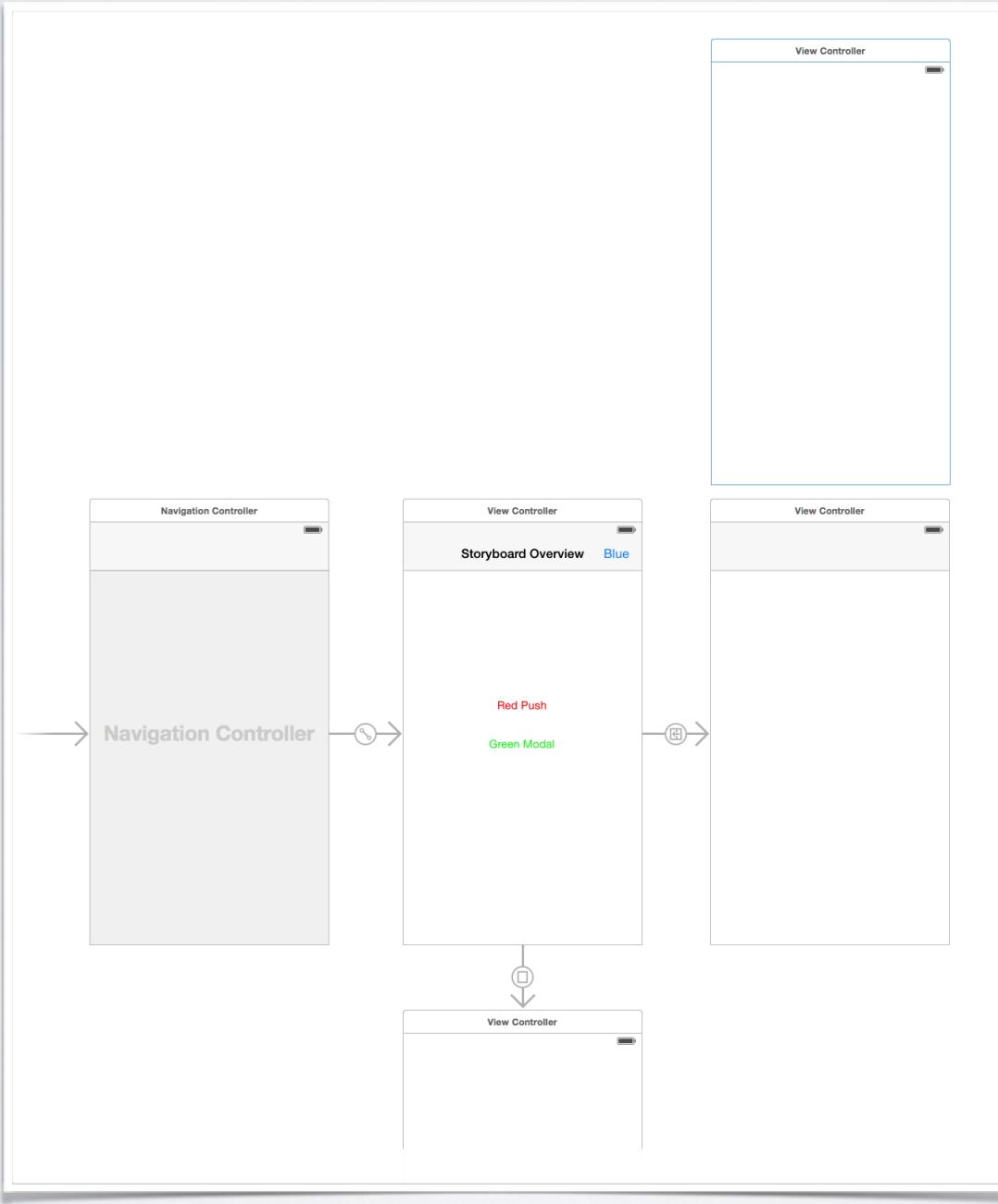
# STEP 9: CHANGE THE TITLE OF THE NAVIGATION BAR



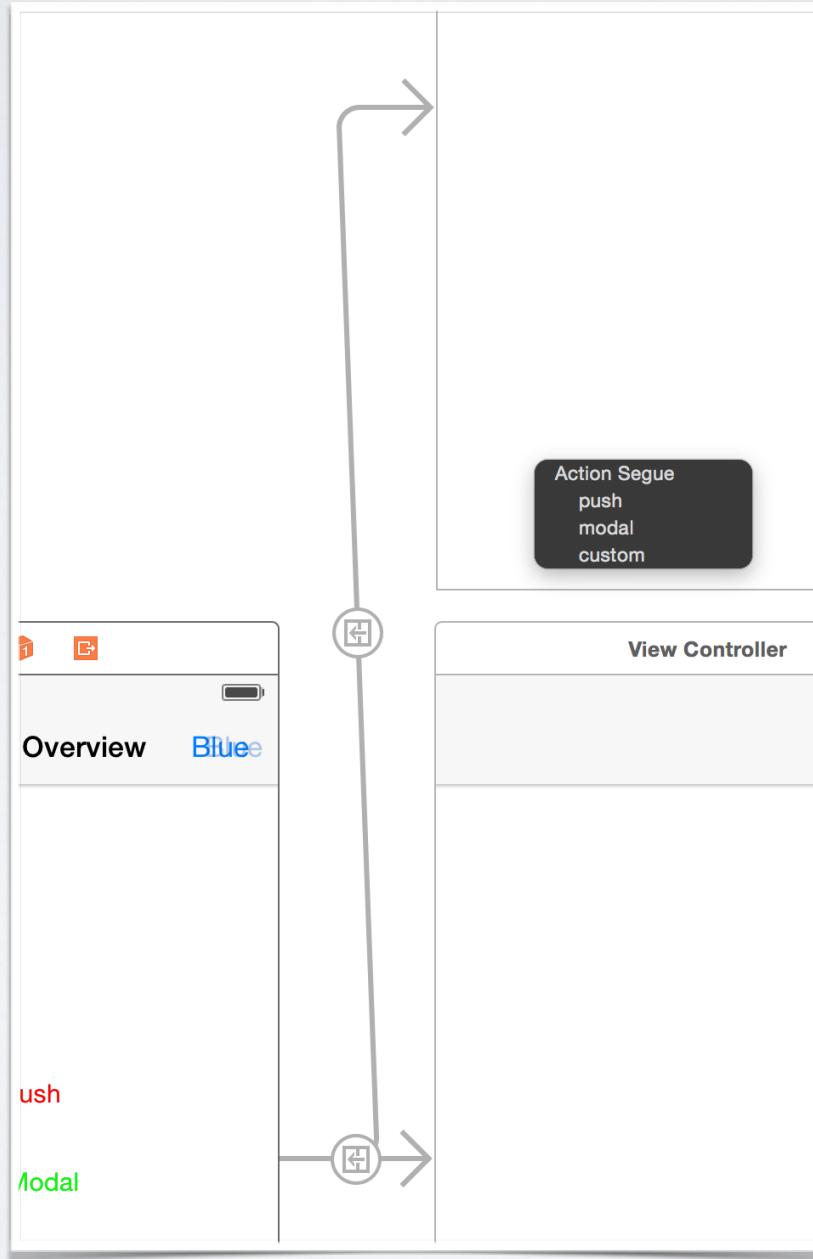
# STEP 10: ADD A BAR BUTTON ITEM TO THE NAVIGATION BAR AND CHANGETHE TITLE TO BLUE



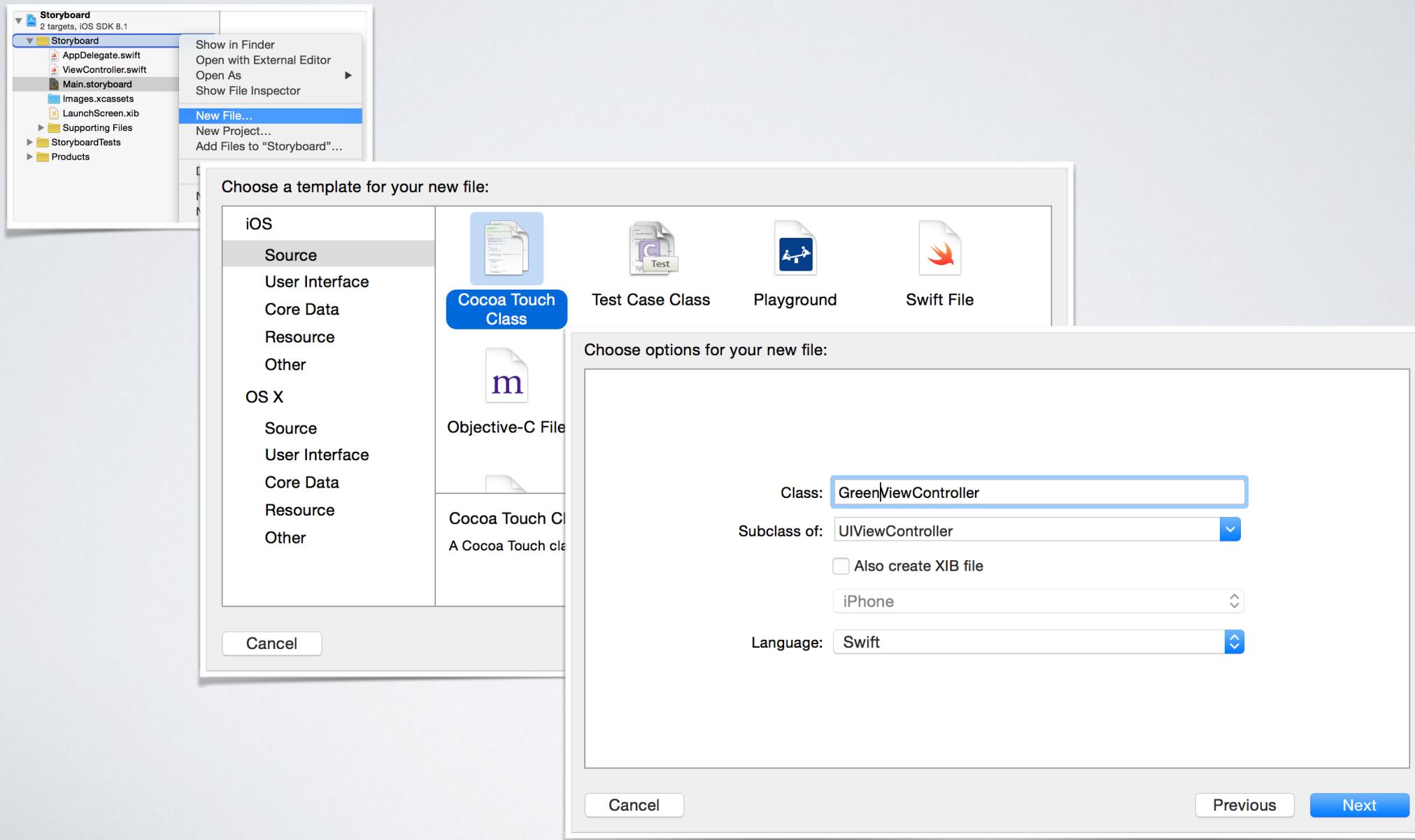
# STEP III: ADD OUR THIRD AND FINAL VIEW CONTROLLER TO THE STORYBOARD



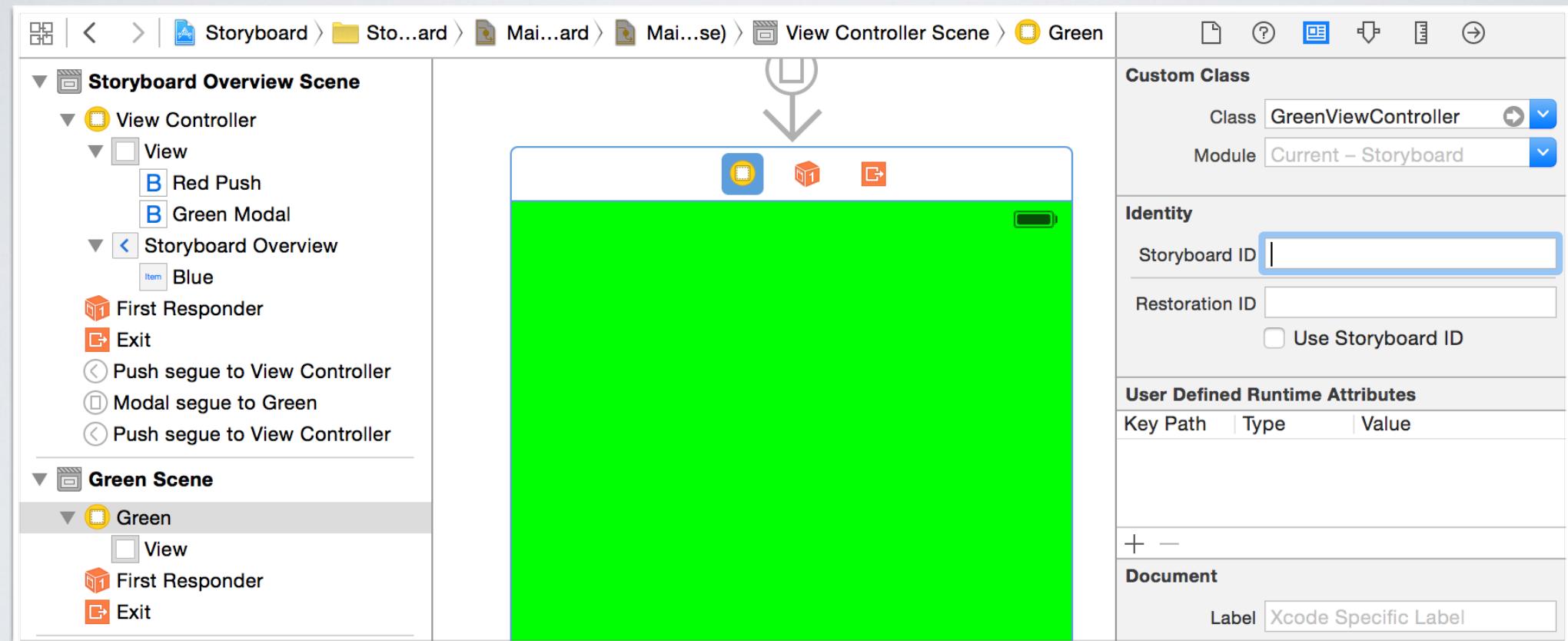
STEP 12: CONNECT OUR BAR BUTTON ITEM TO THE NEW VIEW CONTROLLER AND SELECT PUSH. REMEMBER TO HOLD DOWN THE [CONTROL] BUTTON WHILE CLICKING AND DRAGGING



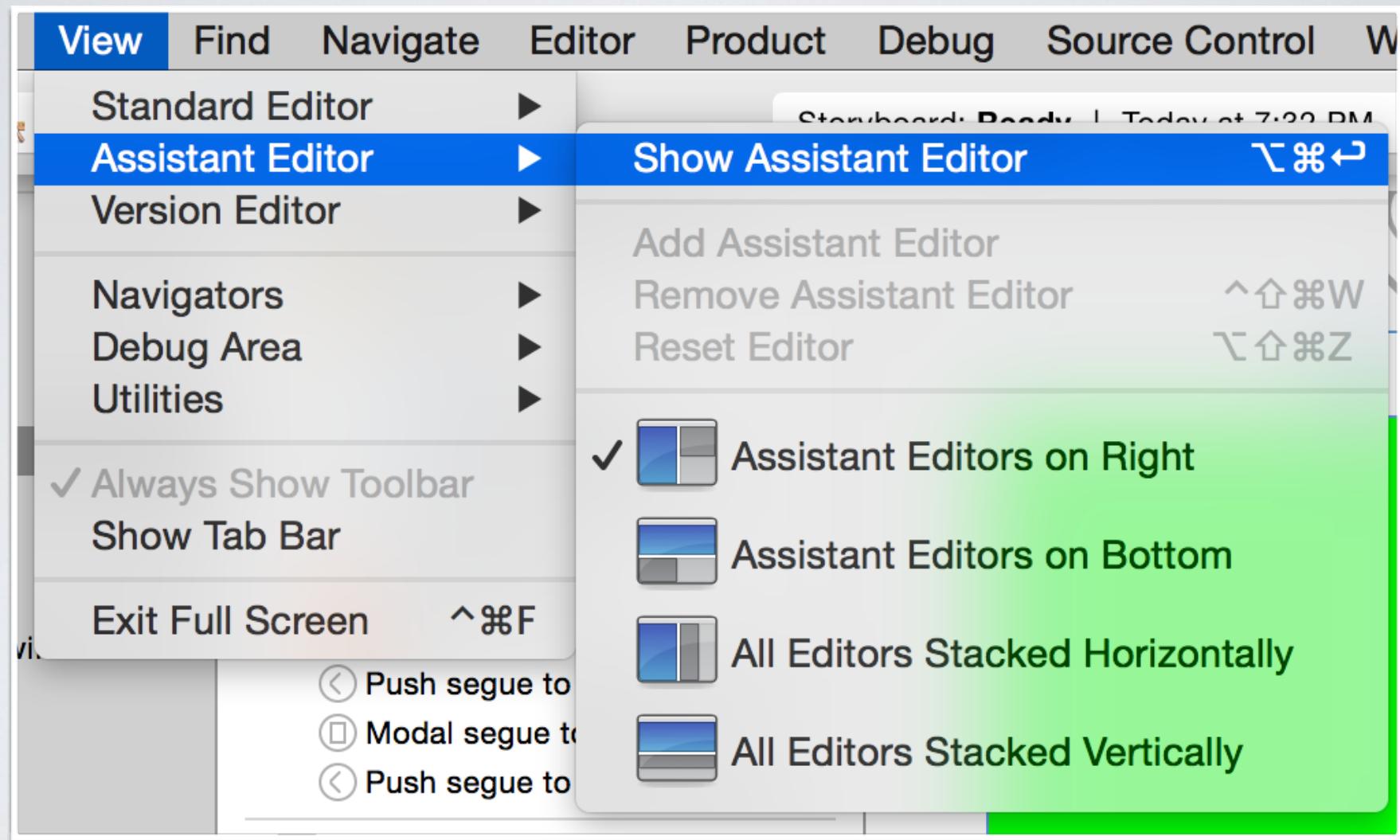
# STEP 13: NOW WE NEED TO DO A LITTLE PROGRAMMING. ADD A CONTROLLER OBJECT AND NAME IT “GREENVIEWCONTROLLER” AS SHOWN BELOW



# STEP 14: ASSIGN YOUR VIEW IN THE STORYBOARD TO THE NEW OBJECT VIEW CONTROLLER YOU JUST CREATED



# STEP 15: SHOW THE ASSISTANT EDITOR TO HAVE OUR CODE GENERATED FOR US BY CLICKING



# STEP 16: ADD A BUTTON TO THE GREEN VIEW

The screenshot shows the Xcode interface with the storyboard and code editor side-by-side.

**Storyboard View:** The storyboard contains a single green view controller. Inside the view, there is a button labeled "Exit Modal". The "View" object in the object library is selected. The "Green Scene" section of the object library also lists "First Responder" and "Exit".

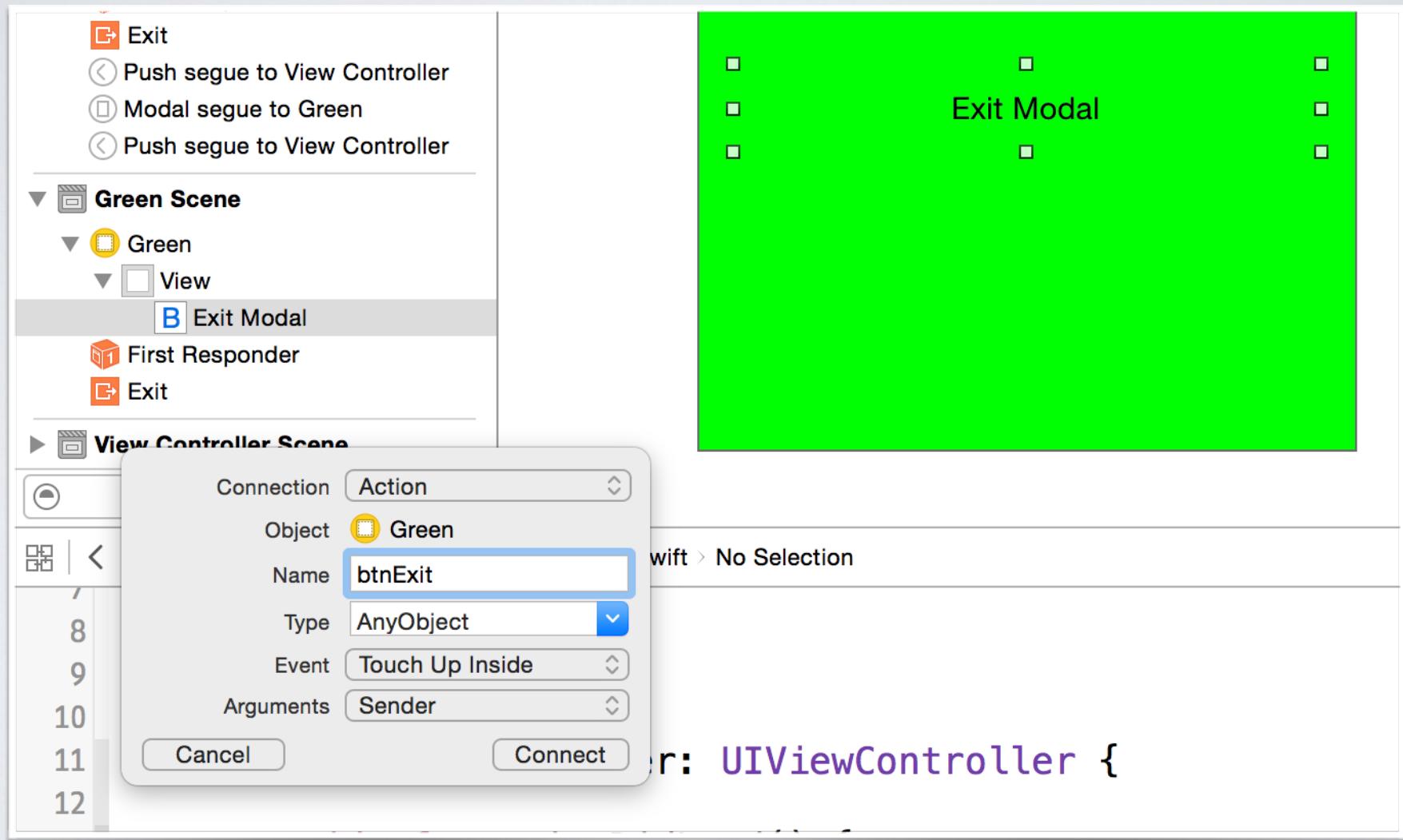
**Code Editor View:** The code editor displays the `GreenViewController.swift` file. The code is as follows:

```
1 //  
2 //  GreenViewController.swift  
3 //  Storyboard  
4 //  
5 //  Created by Don Miller on 12/30/14.  
6 //  Copyright (c) 2014 GroundSpeed. All rights reserved.  
7 //  
8  
9 import UIKit  
10  
11 class GreenViewController: UIViewController {  
12 }
```



GroundSpeed™  
rapid web + mobile software

STEP 17: HOLD DOWN THE CONTROL WHEN CLICKING FROM THE BUTTON TO YOU HEADER FILE AND DRAG IT UNDER THE INTERFACE LINE. CHANGE THE CONNECTION TO ACTION AND THE NAME TO BTNEXIT

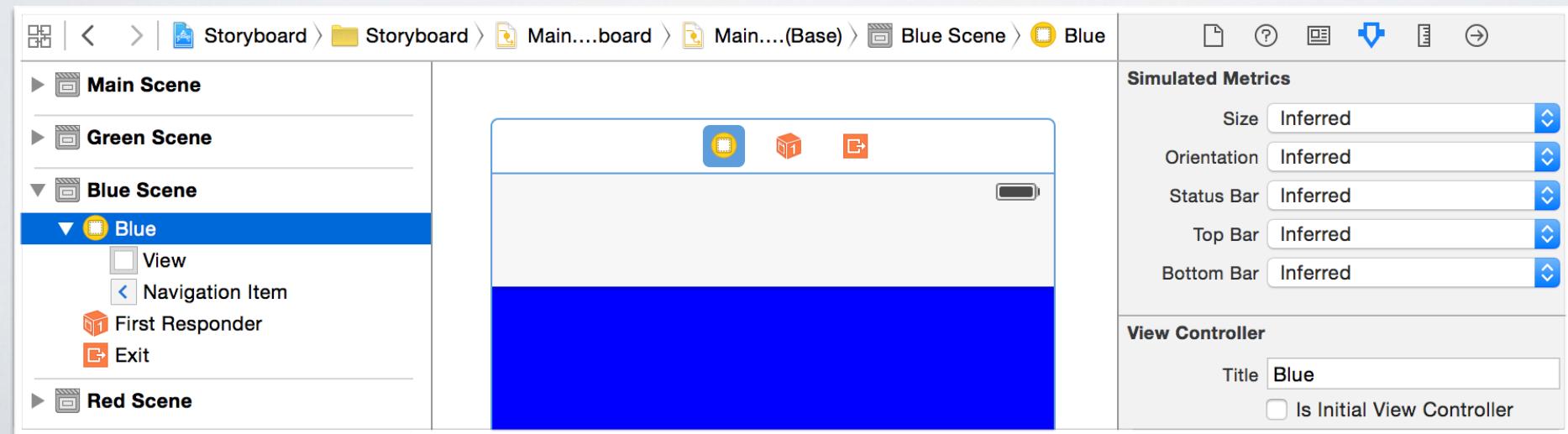
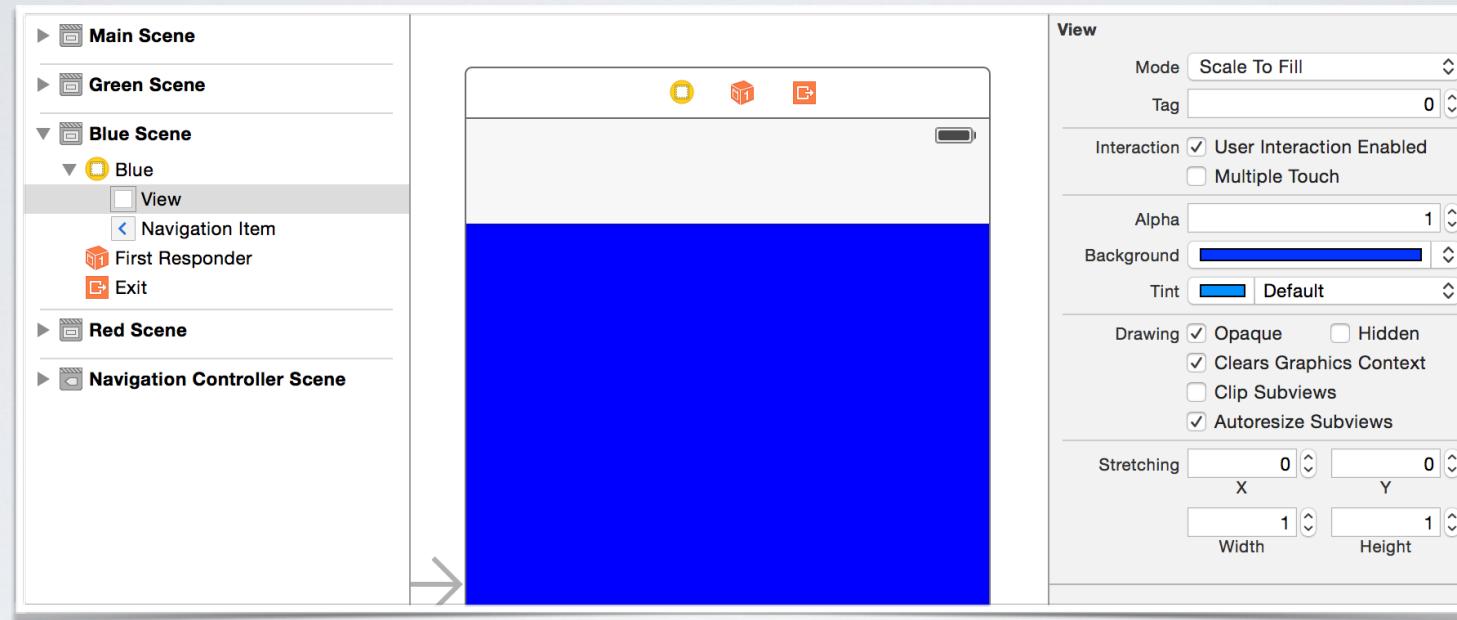


# STEP 18: ADD THE FOLLOWING LINE OF CODE IN THE BTNEXTIT METHOD TO DISMISS THE MODAL WINDOW

```
9 import UIKit
10
11 class GreenViewController: UIViewController {
12
13     override func viewDidLoad() {
14         super.viewDidLoad()
15
16         // Do any additional setup after loading the view.
17     }
18
19     override func didReceiveMemoryWarning() {
20         super.didReceiveMemoryWarning()
21         // Dispose of any resources that can be recreated.
22     }
23
24     @IBAction func btnExit(sender: AnyObject) {
25         self.dismissViewControllerAnimated(true, completion: nil)
26     }
27 }
```

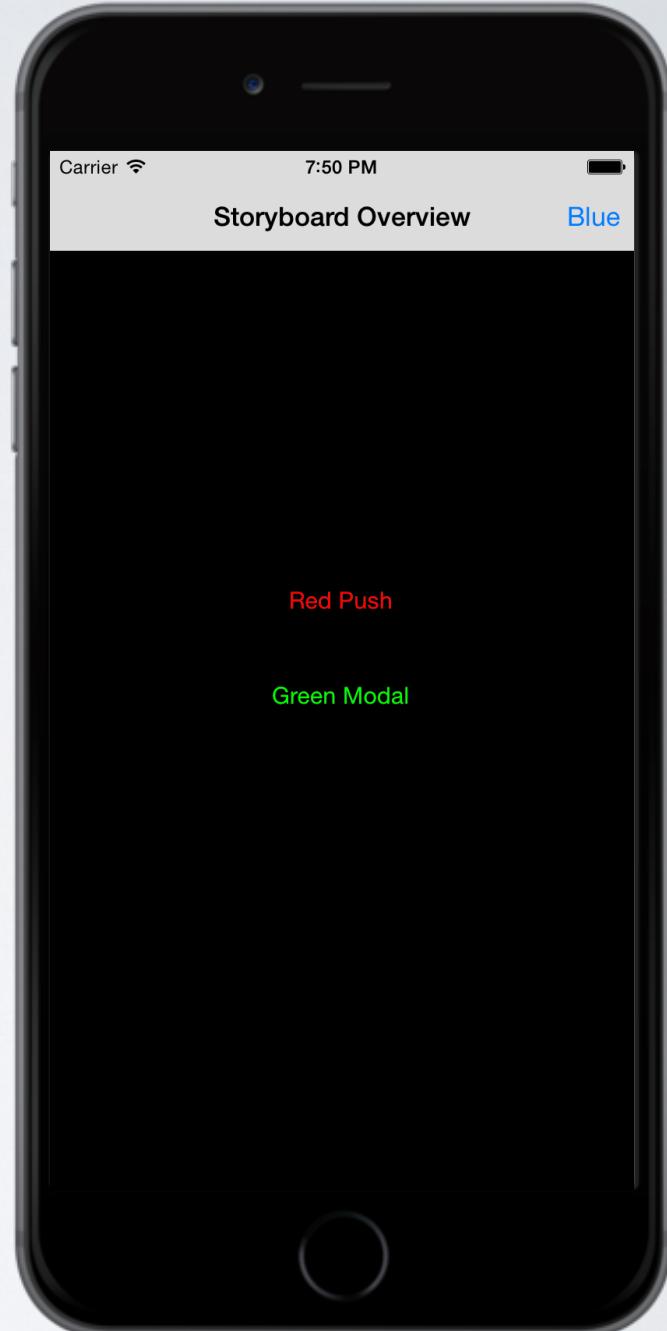


# STEP 19: FINAL TOUCHES. CHANGE THE VIEW CONTROLLER TITLES AND BACKGROUNDS TO REFLECT THE COLOR DESCRIBED IN THE CALLING BUTTON



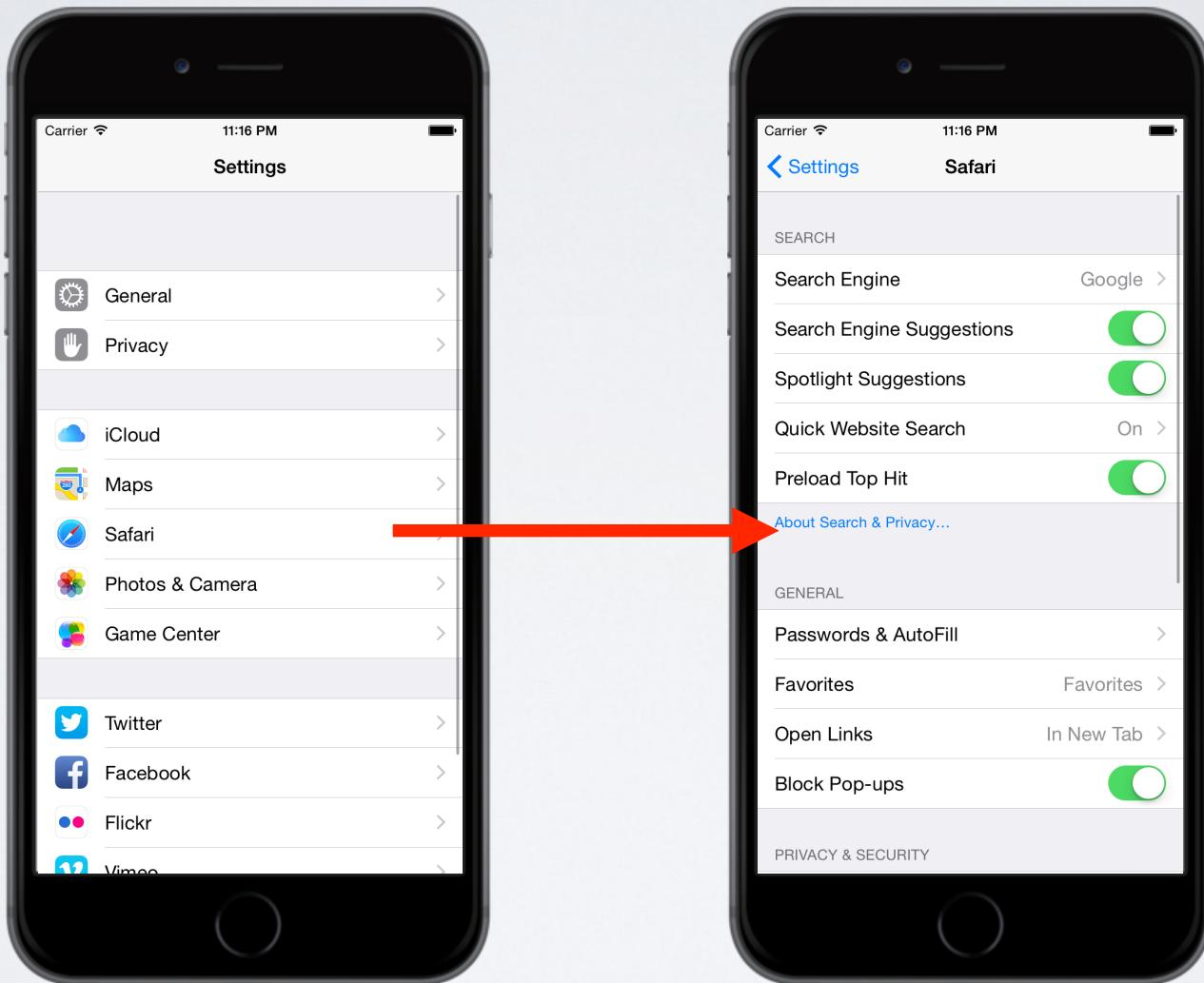
GroundSpeed™  
rapid web + mobile software

# STEP 20: RUN IT!



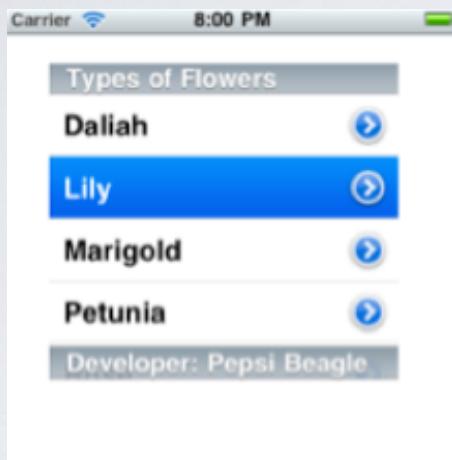
GroundSpeed™  
rapid web + mobile software

# UITABLEVIEW OVERVIEW



# UITABLEVIEW OVERVIEW

## A Simple UITableView with 4 Rows



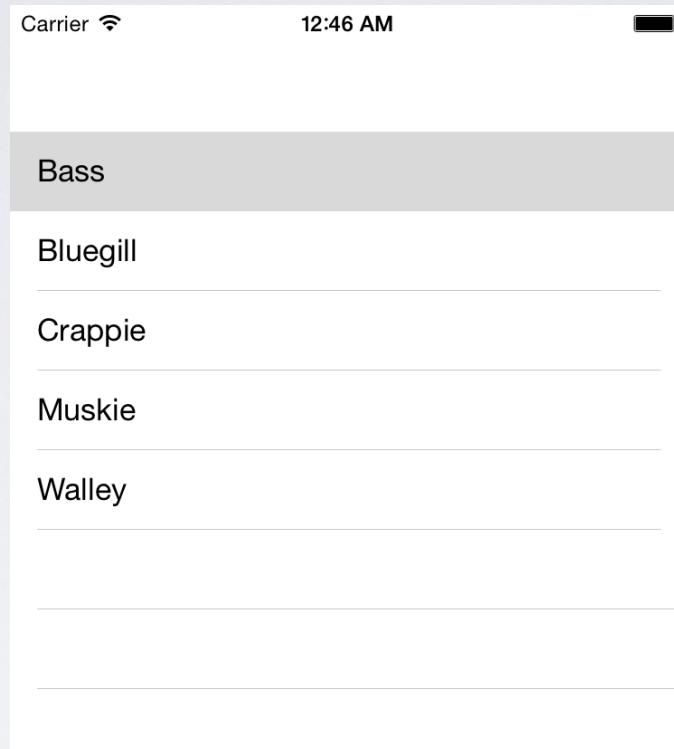
## A Sectioned UITableView

Kentucky	and
Ohio	
Columbus	
Cleveland	
Toledo	
Texas	
Austin	
Dallas	
Houston	
Lubbock	

- Programmatically, the UIKit identifies rows and sections through their index number.
- Sections are numbered 0 through n-1 from the top of a table
- Rows are numbered 0 through n-1 within a section
- As Mentioned earlier, the table view comes in one of two basic styles: plain or grouped.



# TABLE VIEW EXAMPLE



In this example, we will develop a simple table view to display the names of our fish from the example earlier. The completed project will run as shown.



# STEP 1: CREATE A SINGLE VIEW BASED APP.

Choose a template for your new project:

iOS

- Application
- Framework & Library
- Other

OS X

- Application
- Framework & Library
- System Plug-in
- Other

Master-Detail Application

Page-Based Application

Single View Application

Tabbed Application

Game

Choose options for your new project:

## Let's name it: TblVu1

Product Name:

Organization Name:

Organization Identifier:

Bundle Identifier:

Language:

Devices:

Use Core Data

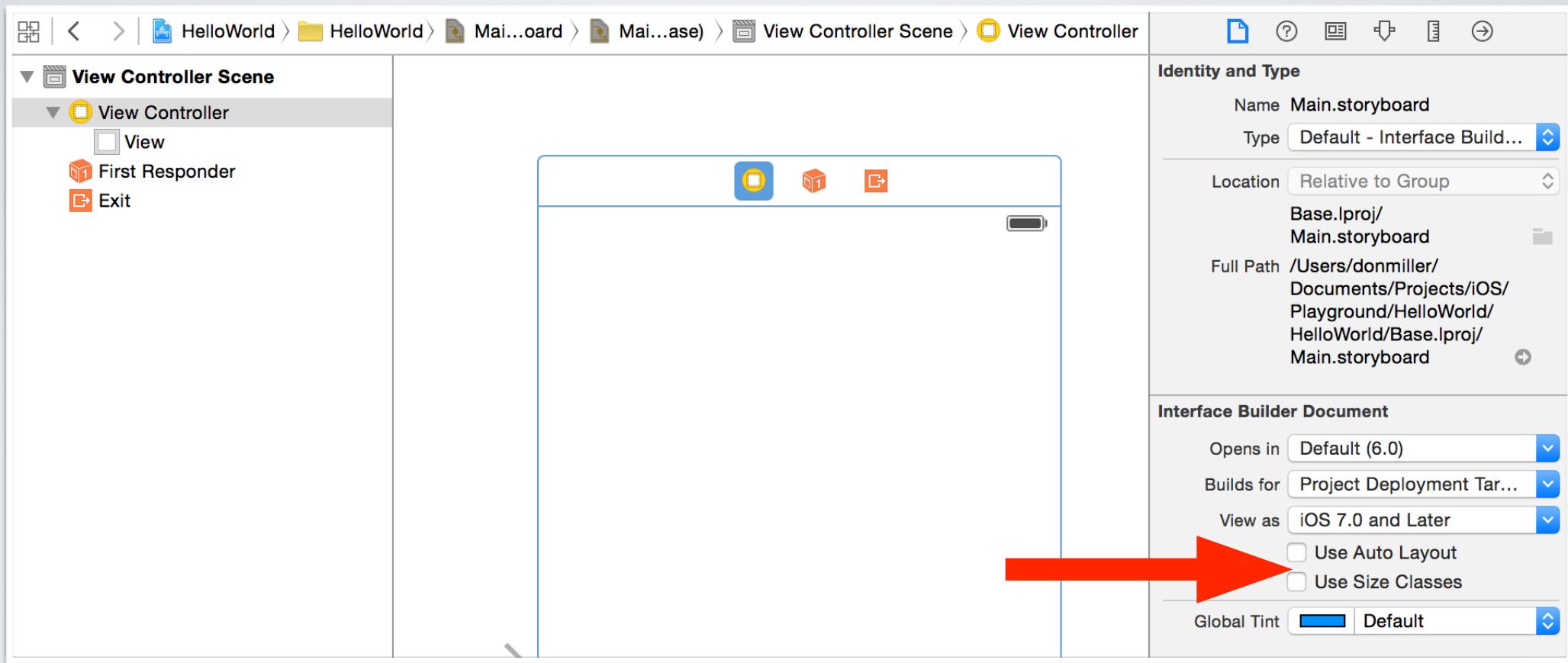
Cancel

Cancel

Previous

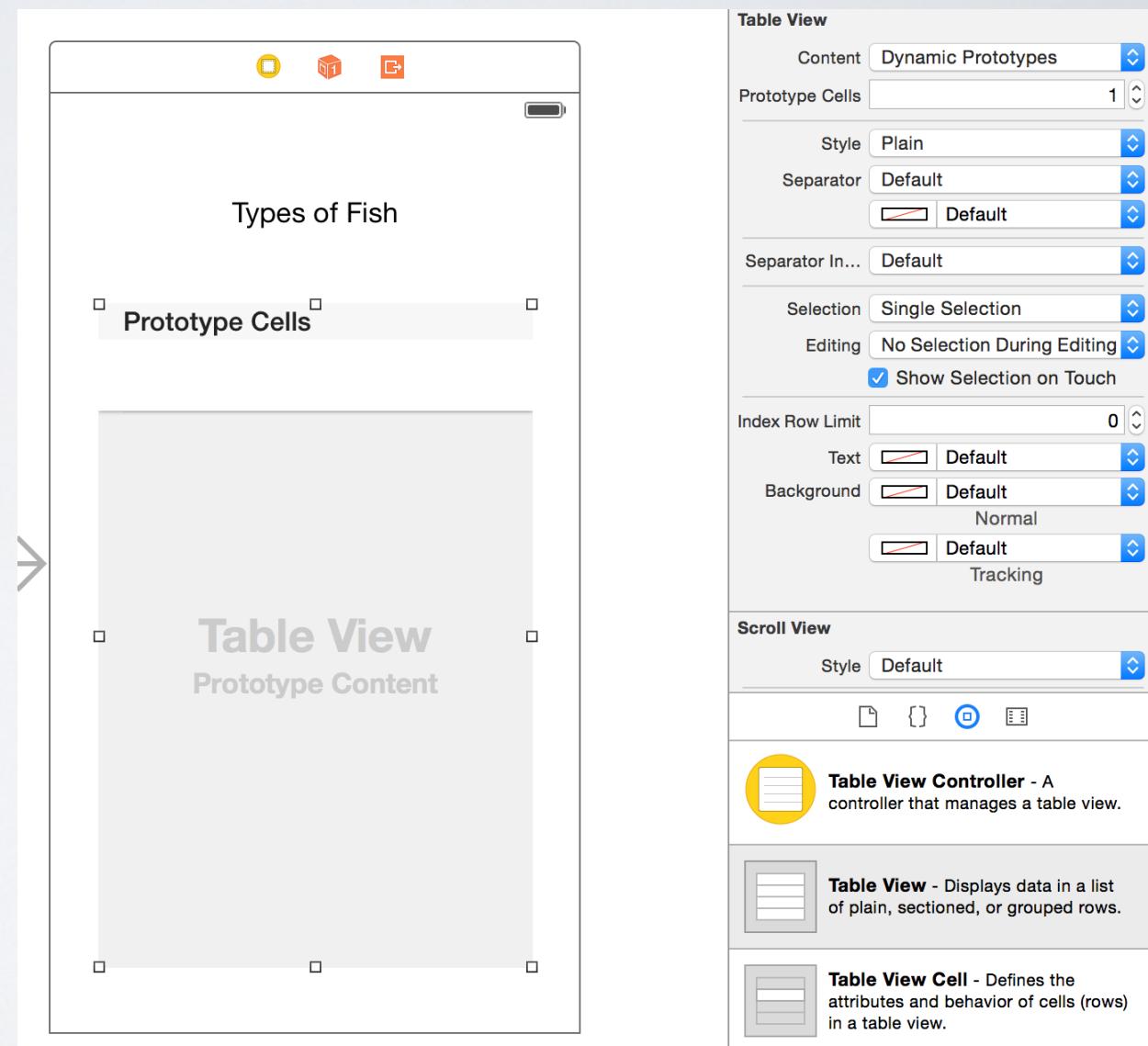
Next

# STEP 2: TURN OFF SIZE CLASSES

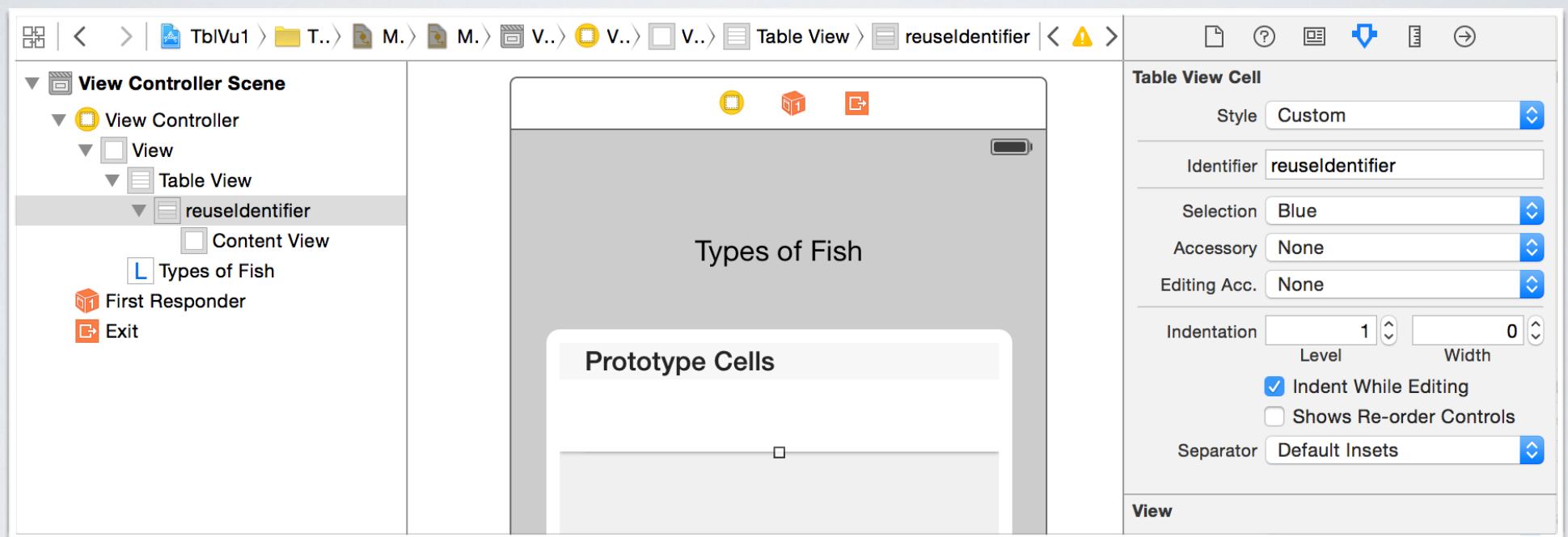


# STEP 3: DRAW TABLE VIEW

- Open the **Main.storyboard** file.
- Draw a label and change its text to "Types of Fish".
- Now draw a **TableView** as shown

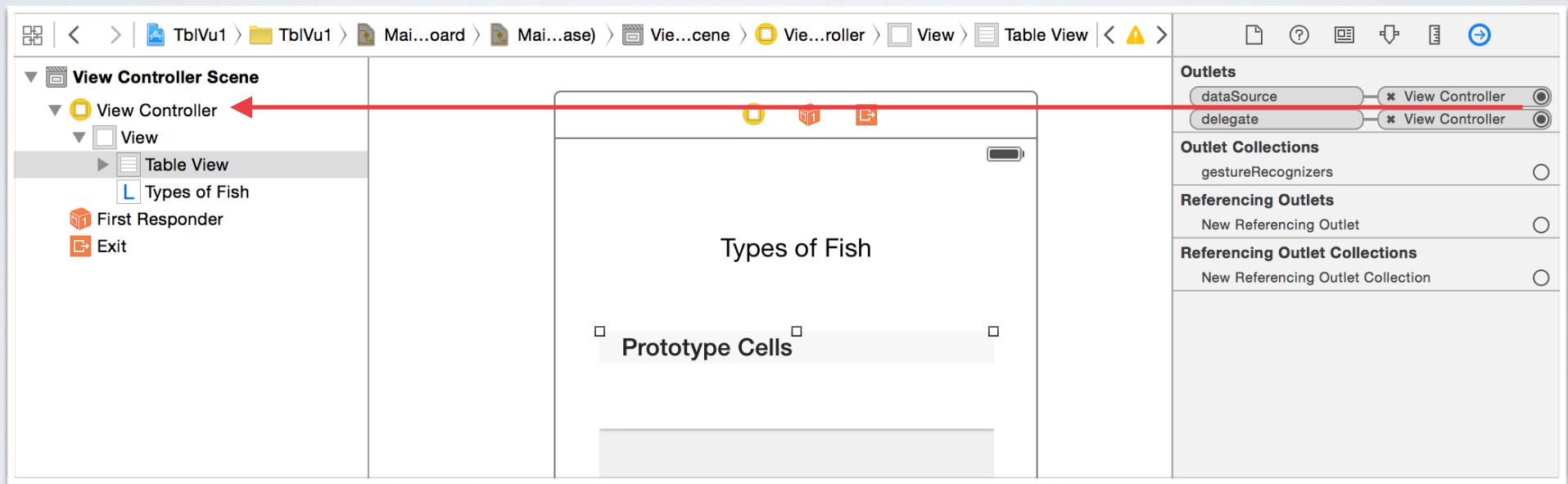


# STEP 4: UPDATE PROTOTYPE CELL WITH REUSE IDENTIFIER



# STEP 5: DATASOURCE & DELEGATE

**Open the Property Inspector of the Table View, then drag its dataSource and delegate outlets and drop those on the File's Owner.**



# STEP 6: OPEN THE VIEWCONTROLLER.SWIFT FILE AND INSERT THE FOLLOWING CODE

```
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {  
  
    let arrayFish = ["Bass", "Bluegill", "Crappie", "Muskie", "Walley"]  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
    }  
  
    func numberOfSectionsInTableView(tableView: UITableView) -> Int {  
        return 1  
    }  
  
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  
        return self.arrayFish.count  
    }  
  
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->  
    UITableViewCell {  
        let cell = tableView.dequeueReusableCellWithIdentifier("reuseIdentifier", forIndexPath:  
indexPath) as UITableViewCell  
  
        // Configure the cell...  
        cell.textLabel?.text = self.arrayFish[indexPath.row]  
  
        return cell  
    }  
}
```

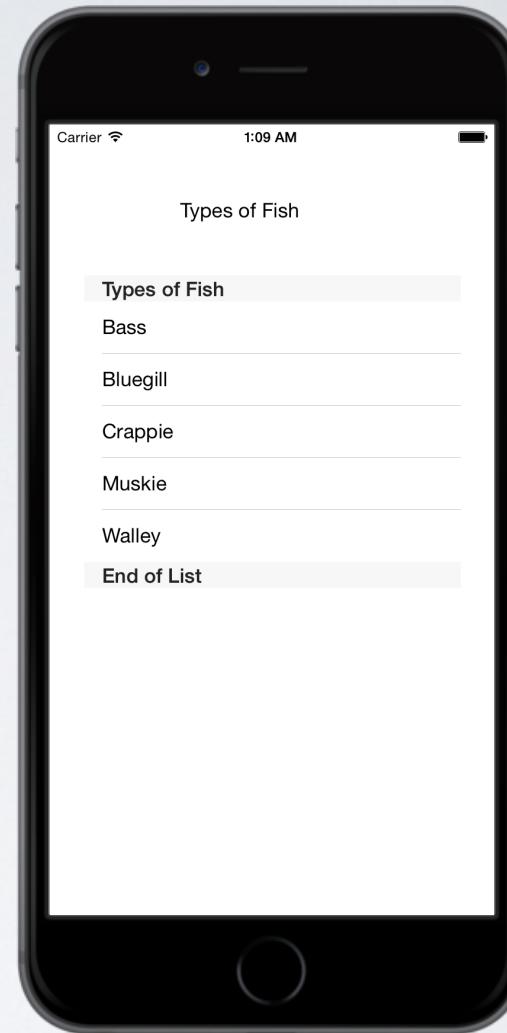


## STEP 7: OPTIONAL HEADER AND FOOTER TITLE OF SECTIONS: YOU MAY IMPLEMENT THE FOLLOWING METHODS FOR THE HEADER AND FOOTER TITLES

```
func tableView(tableView: UITableView, titleForHeaderInSection section: Int) -> String?  
{  
    return "Types of Fish"  
}  
  
func tableView(tableView: UITableView, titleForFooterInSection section: Int) -> String?  
{  
    return "End of List"  
}
```



# YOU ARE DONE!



**GroundSpeed™**  
rapid web + mobile software

**Don Miller**

**don@GroundSpeedHQ.com**

**@donmiller**

**http://github.com/donmiller**

**http://github.com/groundspeed**



**GroundSpeed™**  
rapid web + mobile software