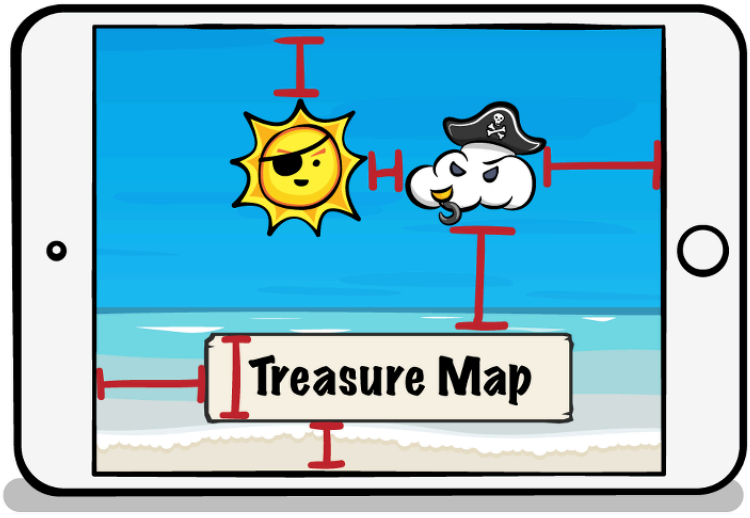
MASTERING

AUTO LAYOUT



HANDS-ON CHALLENGES

Mastering Auto Layout

Jerry Beers

Notice of Rights

Notice of Liability

Trademarks

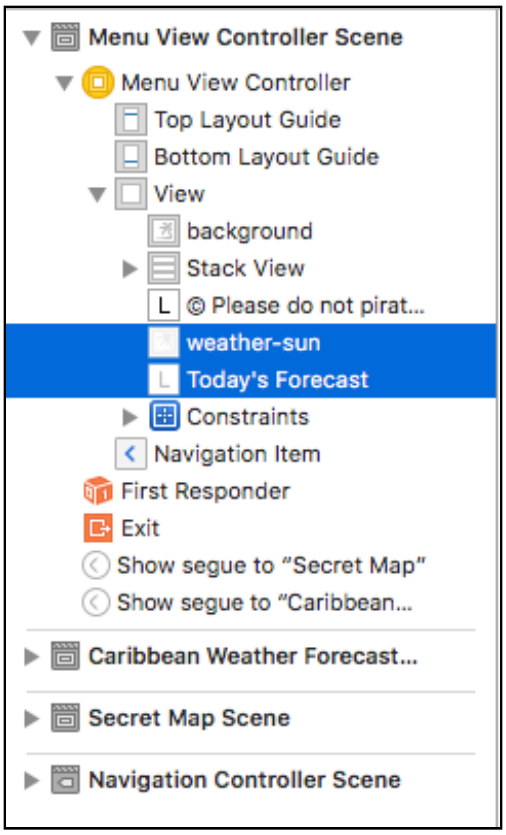
Challenge #9: The Adaptive Layout Environment

By Jerry Beers

Until now, we've had some (fake) weather functionality in two different places: on the menu - to the right in a horizontally regular environment, and on the weather forecast view. It wouldn't be good in a real app to have that functionality duplicated in both places. You'd want one view controller that could handle displaying weather and just change how it looks depending on where it was presented. That's what you're going to do in this challenge.

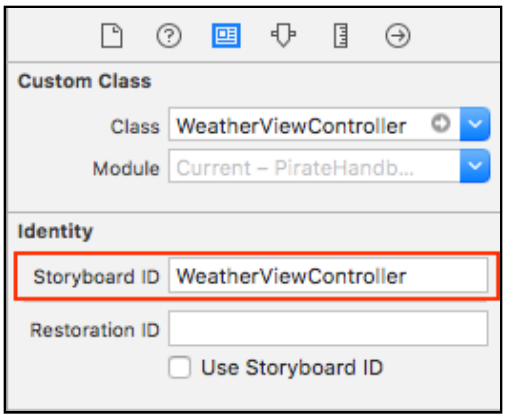
Preparing

To get started, you'll remove the current image view and label for the weather from the menu scene.

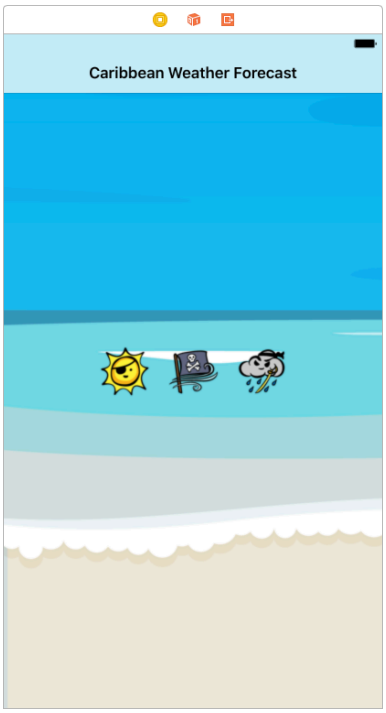


Just select those two and delete them.

While you're in the storyboard, add an identifer for the WeatherViewController:



And let's put the weather icons in a stack view so we can remove that code.



Add constraints to the stack view to center it and see the solution project if you get stuck.

WeatherViewController

Now, let's prepare the WeatherViewController to be able to handle both states - the current one and being displayed as a child on the menu.

First, remove all the properties at the top of the class. Delete these lines:

```
let daysToForecast = 3
let gap: CGFloat = 8
var imageViews: [UIImageView] = []
var imageConstraints: [NSLayoutConstraint] = []
let leftGapGuide = UILayoutGuide()
let rightGapGuide = UILayoutGuide()
```

Add an outlet for the background image, called `backgroundImageView` and an outlet to the stack view called `stackView`.

Next, remove the `setupImageConstraints` method and replace it with an empty one called `setupImages`:

```
private func setupImages(forTraitCollection
  traitCollection: UITraitCollection) {
}
```

Now you can remove all the code in `viewDidLoad` with this:

```
override func viewDidLoad() {
  super.viewDidLoad()

  setupImages(forTraitCollection: traitCollection)
}
```

And replace all the code after the call to `super` in `willTransition` with this:

```
if newCollection.horizontalSizeClass !=
    traitCollection.horizontalSizeClass ||
    newCollection.verticalSizeClass !=
  traitCollection.verticalSizeClass {
    setupImages(forTraitCollection: newCollection)
}
```

Now you're all setup to call the `setupImages` method when the view loads or when the size class changes. All you have left is to implement the method. Add this code to the empty method you added:

```
guard stackView.arrangedSubviews.count > 1 else { return }

let compact = traitCollection.horizontalSizeClass == .compact &&
  traitCollection.verticalSizeClass == .compact
for subview in stackView.arrangedSubviews.suffix(from: 1) {
  subview.isHidden = compact
}
```

This will simply hide all but the first weather image in a compact/compact environment.

MenuViewController

Now that the weather view is capable of handling both states, let's add it as a child view controller to our menu.

First, add some empty methods to handle adding and removing the child controller. Add these to the bottom of `MenuViewController`:

```
private func setupWeatherView() {
}

private func removeWeatherView() {
}
```

And add an override for `viewDidLoad`:

```
override func viewDidLoad() {
  super.viewDidLoad()
  if traitCollection.horizontalSizeClass == .regular {
    setupWeatherView()
    weatherViewController?.view.isHidden = false
  }
```

```
  }
```

This will make sure that if our menu is created in a horizontally regular environment, the weather child view will get setup too.

Now, because the menu can transition from a horizontally regular to horizontally compact environment, we want to handle that transition for the weather view too. Add this override of `willTransition`:

```
  override func willTransition(to newCollection:
      UITraitCollection, with coordinator:
      UIViewControllerTransitionCoordinator) {
    super.willTransition(to: newCollection, with: coordinator)

    // 1
    if newCollection.horizontalSizeClass == .regular,
        newCollection.horizontalSizeClass !=
        traitCollection.horizontalSizeClass {
      setupWeatherView()
    } else if newCollection.horizontalSizeClass == .compact {
      removeWeatherView()
    }
    // 2
    coordinator.animate(alongsideTransition: nil,
        completion: { context in
      self.weatherViewController?.view.isHidden = false
    })
  }
```

Here's what this code is doing:

1. If the horizontal size class is changing to regular, setup the weather view. If it's compact, remove the weather view.
2. The view controller was already in the process of a transition - with an animation. Use the `UIViewControllerTransitionCoordinator` that was passed in to add our animation to it. In this case, showing the weather view.

Now, we need an outlet for the stack view (for creating constraints to it) and a reference to the child controller we're about to add. Add these lines to the top of the view controller:

```
  @IBOutlet var buttonStackView: UIStackView!
  var weatherViewController: WeatherViewController?
```

And connect the outlet to the top level stack view in the storyboard.

All that's left is to add and remove the child view controller. Add this code to `setupWeatherView`:

```
  // 1
  guard let storyboard = storyboard else { return }
  guard let weatherViewController =
```

```
    storyboard.instantiateViewController(withIdentifier:
    String(describing: WeatherViewController.self)) as?
    WeatherViewController else { return }

  // 2
  weatherViewController.view.
    translatesAutoresizingMaskIntoConstraints = false
  weatherViewController.willMove(toParentViewController: self)
  weatherViewController.view.isHidden = true
  // 3
  weatherViewController.backgroundImageView.isHidden = true
  view.addSubview(weatherViewController.view)
  // 4
  weatherViewController.view.centerYAnchor.constraint(equalTo:
    buttonStackView.centerYAnchor).isActive = true
  weatherViewController.view.leadingAnchor.constraint(equalTo:
    buttonStackView.trailingAnchor, constant: 100).isActive = true
  addChildViewController(weatherViewController)

  // 5
  let horizontallyCompact =
    UITraitCollection(horizontalSizeClass: .compact)
  let verticallyCompact =
    UITraitCollection(verticalSizeClass: .compact)
  let compact = UITraitCollection(traitsFrom:
    [horizontallyCompact, verticallyCompact])
  // 6
  setOverrideTraitCollection(compact,
    forChildViewController: weatherViewController)
  // 7
  self.weatherViewController = weatherViewController
```

Taking this line-by-line:

1. First, we create the weather view controller from the storyboard. This should always succeed, but we guard against the case where it does not.

2. This starts a block of the normal code for creating a child view controller, with a couple of interesting additions. First, the storyboard will create the top level view using autoresizing masks, but we want to use our own constraints for the view, so we make sure `translatesAutoresizingMaskIntoConstraints` is false.

3. The weather view has a background, but we don't need it here because we have our own background, so we hide it.

4. Here we add our constraints to the view with an x and y position. The constraints in the weather view controller itself will determine its size.

5. Now for the interesting part, we want to override the trait collection for the child view controller, and this is how to do it. You can only init `UITraitCollection` objects with one trait at a time. To create a collection, you create an instance with each trait and then use the `init(traitsFrom:)` method to combine them.

6. Now we set the override, passing our trait collection and the child controller to

override for.

7. Finally, we hold on to a reference to the child controller so we can remove it later.

That takes care of the "add" case, now for the "remove" part. Add this code to the `removeWeatherView` method:

```
guard let weatherViewController = weatherViewController
  else { return }

weatherViewController.willMove(toParentViewController: nil)
weatherViewController.view.removeFromSuperview()
weatherViewController.removeFromParentViewController()
self.weatherViewController = nil
```

That's it! Now when the weather view controller is added as a child, it will have the trait collection that we specified and use those traits for its layout.