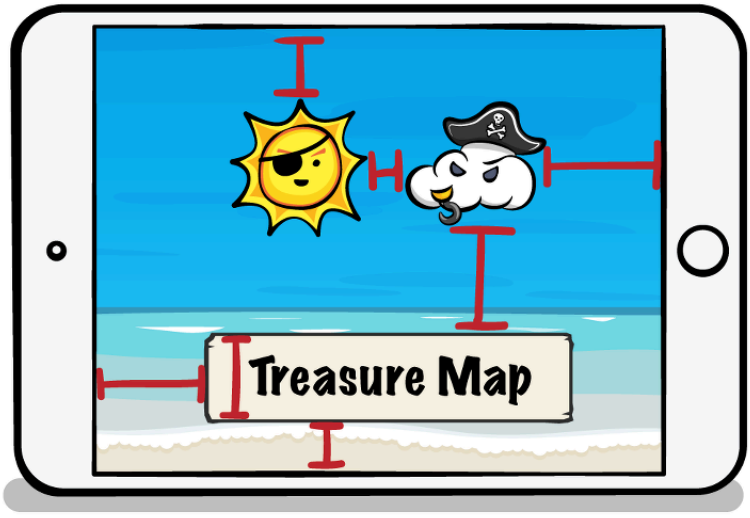
MASTERING

AUTO LAYOUT



HANDS-ON CHALLENGES

Mastering Auto Layout

Jerry Beers

Notice of Rights

Notice of Liability

Trademarks

Challenge #3: Constraint Priorities

By Jerry Beers

This challenge will use several concepts you've learned so far, and add one more - using constraints in scroll views.

In this challenge, we're going on a treasure hunt... to find the lost secrets of scroll view constraints. Let's get started!

Constraints in Scroll Views

The key to understanding Auto Layout in `UIScrollView` is to know how constraints between a scroll view and other views will behave. Constraints between a scroll view and any view "outside" act on the scroll view's frame, like normal. But constraints between a scroll view and anything "inside" act differently depending on the attribute involved in the constraint. Constraints to the scroll view height, width, or centers act on the frame, but constraints to the edges (including margins) act on the scroll view's content area.

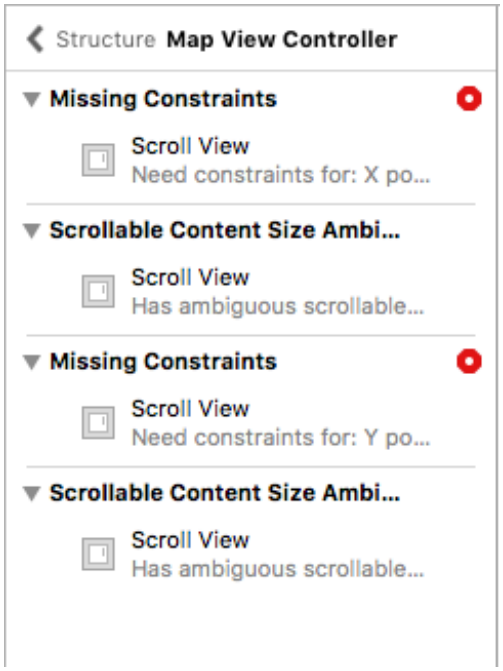
That means, if you have a subview constrained to the leading and trailing edge of a scroll view, you might think those constraints will define the width of the subview. But since those constraints are to the content area, not the scroll view frame, unless something else defines the width (like intrinsic size or another constraint), the layout is still ambiguous.

Finally, in order for a scroll view to be completely constrained, you need to have constraints that completely define both its frame and content size.

In this challenge, you'll create constraints on the map background image to define the content area. Then you'll create some constraints on the Marker and use priorities and inequalities to keep the marker on the screen, but pointing to the direction of the treasure.

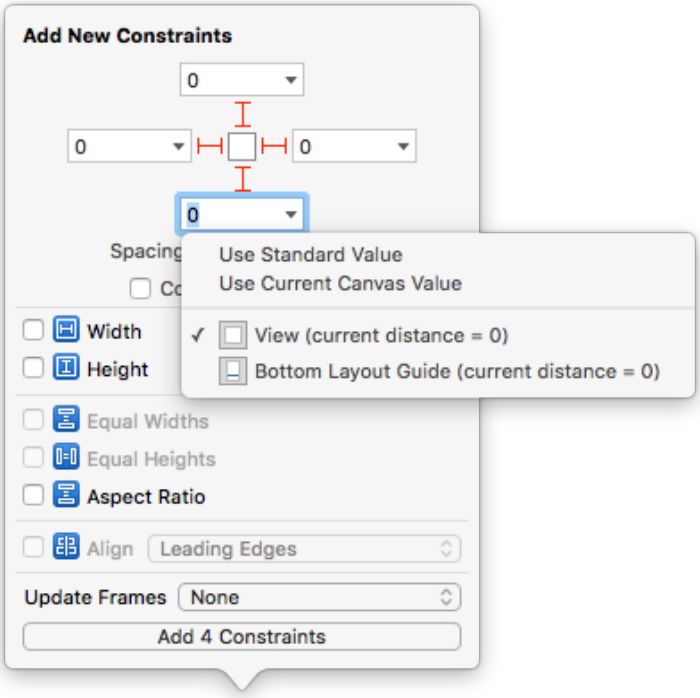
Defining the Scroll View Frame and Content Area

The starter project for the challenge has the views created, but without any constraints. You'll notice that Auto Layout is giving a bunch of errors about missing constraints in the storyboard, so let's fix that.



In Main.storyboard, on the **Map View Controller Scene**, start by adding these constraints on the scroll view:

- Top, left, bottom, and right constraints of 0 to the view. Deselect the Constraint to Margins checkbox and make sure the bottom constraint is to the view, not the bottom layout guide.



Since these constraints are between the scroll view and a view outside, the top-level view, they will act on the scroll view's frame and make it take up the whole view. But you haven't created any constraints on the content yet.

Add these constraints to the **map** image view:

- Top, bottom, leading, and trailing constraints of 0 to the scroll view.

Of course, an image view has an intrinsic size equal to the size of its image, so those constraints are defining the width and the height of the map image view. Those constraints together with the ones you just added are enough to tell the scroll view what its content size should be.

Build and run and you can scroll around the whole map.

Constraining the Marker Image

You haven't created any constraints on the Marker view or image view yet, so Xcode creates some constraints for you. But, as usual, they're not what you want.

First, you always want the marker to stay on the screen, even when the scroll view is scrolling around. So follow these steps to add constraints to the Marker view:

1. In the document outline, control-drag from the Marker view to the top-level view
2. Hold down **Option** and **Shift** and click on **Leading Space to View** and

Trailing Space to View

3. Let go of Option and Shift and click on **Vertical Spacing to Top Layout Guide** and **Vertical Spacing to Bottom Layout Guide**
4. Click **Add Constraints**
5. Edit each constraint to have a constant of 0 and a relation of >=

These constraints will make sure that the marker always stays within the visible area of the screen. Now, if the "treasure" is also on the screen, you want the marker over that point. Add these constraints to the Marker view:

- Leading space constraint with a constant of 1610 to the super view
- Top space constraint with a constant of 1610 to the super view

You don't want these constraints to be required, because then they conflict with the first set of constraints. So set the priority on both to be 999.

In just a minute, you'll see why you have both the Marker view and the image view as a subview, but for now, you need to create constraints on the image view. Add these constraints:

- Top, bottom, leading, and trailing constraints of 0 to the super view.

Fixing Priorities

You made the priority of the positioning constraints to be 999, but you may notice that the size of the marker view is now 0 x 0. This is because the size of the image view is determined by its intrinsic size constraints, which have a compression resistance priority of 750 by default. To fix this, you'll change the compression resistance of the marker image view, horizontal and vertical, to 999. Apple recommends that you **not** make content hugging or compression resistance priorities required, so you'll use 999. But to avoid a tie with the positioning constraints, you'll change those to 998.

- Select the marker image view and in the Size inspector, change the **Content Compression Resistance Priority** to 999 for Horizontal and Vertical.
- Select the marker parent view and find the leading and top space constraints to its superview. Change the priorities of both to 998.

Build and run and you'll see the marker move to indicate where the treasure is, but there are two problems. You want the arrow to rotate to point in the direction of the treasure, and you want the image to change to an "X" when it's over the correct spot. To fix these, you'll have to write a little code.

Changing the Image

First, you need properties for the two positioning constraints so you can deal with them in code. Open the assistant editor to **MapViewController.swift** and control-drag from the constraints between the Marker view and it's super view:

```
@IBOutlet weak var desiredXConstraint: NSLayoutConstraint!
@IBOutlet weak var desiredYConstraint: NSLayoutConstraint!
```

Now, add this code to `decideArrowOrX()`:

```
// 1
let currentPoint = marker.frame.origin
let desiredPoint = CGPoint(x: desiredXConstraint.constant,
                           y: desiredYConstraint.constant)
// 2
if currentPoint == desiredPoint {
  markerImageView.image = UIImage(named: "x")
  markerImageView.transform = CGAffineTransform.identity
} else {
  // 3
  markerImageView.image = UIImage(named: "arrow")
  // 4
  let angle = angleBetween(currentPoint, desiredPoint)
  markerImageView.transform =
    CGAffineTransform(rotationAngle: angle)
}
```

Taking this line by line:

1. First, we make two points. The first point is just the current origin of the marker. The second point is created from the constant for the top and leading constraints. That's where the treasure is on the map.

2. If the origin of the marker is currently at the treasure, you change the image to show the "X" and reset the rotation of the image.

3. If the origin of the marker is *not* currently at the treasure, you change the image to the arrow and calculate the angle between the two points.

4. You use a transform to rotate the arrow by the angle you calculated to get it to point to where the treasure is on the map. Transforms change the frame of the view they're applied to. This is why you use an image view inside the Marker view. It makes comparing the constraints to the current frame cleaner.

Now that you've written the implementation for this method, it's time to call it. You want to initialize the Marker when the app first runs and whenever the user scrolls through the map. Add the following override of `viewDidLoad()`:

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

```
    decideArrowOrX()
  }
```

First, you make sure that Auto Layout has updated the marker, then you call the method to set the image and rotation. That takes care of the startup case. Add the following code at the very bottom, after the end of the class definition:

```
extension MapViewController: UIScrollViewDelegate {
  func scrollViewDidScroll(_ scrollView: UIScrollView) {
    decideArrowOrX()
  }
}
```

Now, when you scroll around the map, the marker will show the correct image and point to the treasure, if it's off the screen. And most of the work was done with a few constraints!

Congratulations, you've created a layout in a scroll view and used Auto Layout to define the content size. And you created some constraints between content of the scroll view and a view outside the scroll view for some advanced behavior!