DEVCON

RWDevCon 2018 Vault

By the raywenderlich.com Tutorial Team

Notice of Rights

Notice of Liability

Trademarks

Table of Contents: Overview

Table of Contents: Extended

Prerequisites

Some tutorials and workshops at RWDevCon 2018 have prerequisites.

If you plan on attending any of these tutorials or workshops, **be sure to complete the steps below before attending the tutorial**.

If you don't, you might not be able to follow along with those tutorials.

Note: All talks require **Xcode 9.2** installed, unless specified otherwise (such as in the ARKit tutorial and workshop).

5: Auto Layout Best Practices

You'll need to have CocoaPods installed - please see https://cocoapods.org for more details.

5: Auto Layout Best Practices

Auto Layout takes effort to learn, and can be notoriously painful to do so. But once you have the basics how can you become efficient at applying, editing and debugging constraints ? In this session we will examine some best practices for Auto Layout, looking at examples via Interface Builder and in code. The session will focus primarily on Auto Layout for iOS.

You will need CocoaPods installed on your machine as a prerequisite for this tutorial. You can find instructions on installing this and getting setup at https://cocoapods.org/.

Auto Layout Best Practices: Demo 1

By Gemma Barlow

In this demo, you will setup the board for an app you will build throughout the remainder of the tutorial. Ladders and Snakes is an adaptation of a classic board-game that originated in ancient India (see https://en.wikipedia.org/wiki/Snakes_and_Ladders for more details).

The steps here will be explained in the demo, but here are the raw steps in case you miss a step or get stuck.

> **Note**: Begin work with the starter project in **Demo1\starter**. Double click on **Ladders and Snakes.xcodeproj** to get started in Xcode.

1) Ensure a consistent board

After opening the project, press **Command-R** to build and run. When the project launches you should see a square representing the game's board.

With the Simulator selected, change its orientation using **Command-Right Arrow**. Repeat this action multiple times. Notice that on rotation the board expands to fill the screen, and parts of it are no longer visible.

Instead of this behavior, the board should remain consistently square, **in all orientations**.

Press **Command-Shift-O** and search for *Main.storyboard*. Press **Return** to open the file in Interface Builder.

Select the blue **Board Background** view on the canvas, and view all constraints associated with it.

Under the **Sibling and Ancestor Constraints** section of this inspector, select *Align Trailing to: Safe Area* and *Align Leading to: Safe Area*. Hit **Delete** to remove these constraints, as they cause some of the undesirable behavior described above.

Errors will appear; don't worry - we'll fix those in a moment.

Equal Widths and Equal Heights

Next, with the same view selected, **Ctrl-Drag** between the Board Background and the top-level view. Add an *Equal Widths* constraint between the Board Background view and its superview.

Repeat the **Ctrl-Drag** action to add an *Equal Heights* constraint.

Notice the view expand to fill the screen. This is not yet what we want.

Repeat the **Ctrl-Drag** action one more time. This time, hold **Shift** while you select constraints, so as to set multiple at once. Add an additional constraint for the width and an additional contstraint for the height of the view.

You should now have the following constraints for the **Board Background** view:

- *Equal Width to: Superview* (constant = 0, priority = 1000)
- *Equal Width to: Superview* (constant = 0, priority = 1000)
- *Equal Height to: Superview* (constant = 0, priority = 1000)
- *Equal Height to: Superview* (constant = 0, priority = 1000)

You will still see errors - again, this is fine.

Next, click **Edit** and modify each of the constraints. Modifications have been bolded.

- *Equal Width to: Superview* (constant = 0, **priority = 750**)
- *Equal Width to: Superview* (**constant <= 0**, priority = 1000)
- *Equal Height to: Superview* (constant = 0, **priority = 750**)
- *Equal Height to: Superview*(**constant <=0**, priority = 1000)

In the section above we have created two sets of constraints in each dimension - you can think of them as a *primary* constraint and a *fallback* constraint that will be used when the primary constraint cannot be satisfied.

Let's consider width. In the case above, two constraints are mandatory - indicated by them having a priority of 1000. They are:

- *1:1 Ratio to: Board Background* (i.e. the board must be square)
- *Equal width to: Superview* (constant <= 0) (i.e. the board must be less than or equal to the width of the superview).

One constraint is optional - indicated by it having a priority of less than 1000:

- *Equal Width to: Superview* (constant = 0) (i.e. if all the mandatory constraints can be satisfied, the board should take up the width of the superview).

In aggregate, this asks Auto Layout to try and find the maximum *width* the board can be, such that it remains a square.

The height constraints are simultaneously asking Auto Layout to find the maximum *height* the board can be, such that it remains a square.

The net effect of these constraints is that the board will be generated to be as big *as the smaller of the height or width of the view it is contained within*, whilst remaining a square. These constraints work in any orientation.

Missing Constraints

One error should still remain. Click the red arrow in the top right-hand corner of the left pane of Interface Builder to inspect it. It should read *Missing constraints. Board Background View - Need constraints for: X Position or Width*.

To correct this error, again Ctrl-Drag from the Board Background View to its parent, selecting *Center Horizontally in Safe Area* as the constraint to add. You have now adjusted all the required constraints. Build and run your code (**Command-R**).

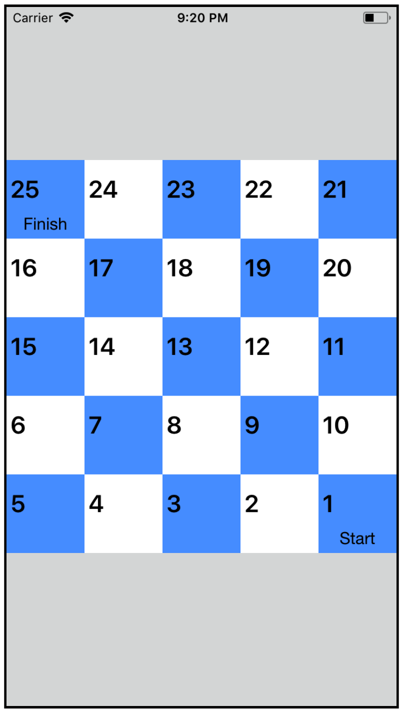
You should see the board background remain a square regardless of the rotation applied to the device.

Time to move on to the next stage of our game setup!

2) Add Progress Squares

In this version of Ladders and Snakes, 25 numbered squares should be visible on the board.

When you've completed this section, the board should look similar to the following:



You're welcome to choose your own color theme, of course!

Select the dice image from *Main.storyboard* and press delete to remove it. It's pretty, but may prove distracting for the remainder of this tutorial.

Next, search in Interface Builder for a **Vertical Stack View** and add it to the **Board Background** that exists on the canvas. Constrain its leading, trailing, top and bottom edges to the superview via the **Add New Constraints** button.

Click the stack view in the left-most pane in Interface Builder, then click it again and rename this newly-added stack view to 'Rows'. You'll see why a little later in the tutorial.

Next, search in Interface Builder for a **Horizontal Stack View** and add it to **Rows** Stack View you just created. Rename this view to be called **Items in Row 1**.

Square Setup

Add a new view to the **Items in Row 1** `UIStackView`. This will shortly become an instance of *BoardSquareViewLoader*.

To make this item, do the following:

- Drag a new *UIView* to be contained within **Items in Row 1**
- Apply a 1:1 *Aspect Ratio* constraint to ensure it will always be a square.
- Change the class of this view to be *BoardSquareViewLoader*

- Under the **Nib Name** attribute, add 'BoardSquareView'
- Under the **Number Text** attribute, add the value '25'
- Under the **Description Text** attribute, add 'Finish' as the text

Press **Command-R** to build and run. You should now see a giant square on the screen. Success!

> **Note**: Interface Builder should update the views to reflect the information you've entered above. If you don't see any adjustments in the canvas, shut down Xcode and reopen it, then take another look at **Main.storyboard**.

Squares, Squares Everywhere

Now that you have setup one square, let's go ahead and create the remaining twenty four (!).

First, setup five squares in Row 1:

- Copy the **BoardSquareView** item and paste it four times into the **Items in Row 1** `UIStackView`
- Select alternating squares, and set their background color to be something complementary. In the screenshot above, white and blue is used - but you can try out any color scheme you like.
- Remove the text in the **Number** field - only squares 25 and 1 should contain this text (initially).
- Remove the text in the **Description Text** field - only squares 25 and 1 should contain this text.

Next, replicate these rows:

- Copy the **Items in Row 1** `UIStackView`, and paste it four times into the **Rows** `UIStackView`
- Rename each row to read **Items in Row x** - where *x* is a value from 1 to 5
- Select alternating squares and alter their background colors to be complementary (refer to screenshot above for desired pattern)
- Edit the text in the **Description Text** field to read 'Start' for square 1.

All Auto Layout constraint issues should now be resolved. Press **Command-R** (build and run) to check the result of your hard work.

When the application launches, press **Command-Right-Arrow** to rotate the Simulator and admire your handiwork in all orientations.

Note: Due to time constraints, we will not update all of the squares on the board to be numbered during the live tutorial. This is left as an exercise for the attendee.

3) Dynamic Type

Finally, let's try our game with large fonts enabled.

Launch and run the application in the Simulator and notice that when Larger Text is used via the device (**Settings > General > Accessibility > Larger Text**) the text of your application doesn't adjust appropriately. You will see no change from running the app without this setting enabled.

We'd like the number on the board to increase when Larger Text is applied via accessibility settings. The 'Start' and 'Finish' labels, however are secondary information, and so we're happy leaving them at the existing size.

To adjust the layout of the board's squares in this scenario, press **Command-Shift-O** and search for **BoardSquareView.xib**. Select *numberLabel* and inspect its properties in Interface Builder.

To enable Dynamic Type for this label:

- Select *Headline* as the font style applied to the label's text
- Click the *Automatically Adjusts Font* checkbox for the label

Next, select both the *numberLabel* and *descriptionLabel* UILabel instances on the canvas, and pop them into a Vertical Stack view, using the **Embed in Stack** button in Interface Builder.

This will cause an error, as after the embed occurs, the `UIStackView` doesn't have a width and height set. Constrain the top, bottom, leading and trailing edges of the `UIStackView` to its superview.

Build and run the project, varying the size of the text used on the device. Note that this time around the numerical information on the square will grow and shrink to match accessibility settings. Success!

4) That's it!

Congrats, at this time you should have a good understanding of Auto Layout best practices with Interface Builder! It's time to move on to best practices with Auto Layout and code.

Auto Layout Best Practices: Demo 2

By Gemma Barlow

In this demo, you will add some ladders and snakes to your newly-created board.

The steps here will be explained in the demo, but here are the raw steps in case you miss a step or get stuck.

Note: Begin work with the starter project in **Demo2\starter.** As per the prerequisites for this tutorial, *CocoaPods should be installed on your machine as it will be needed to build and run this project*.

1) Get Set Up

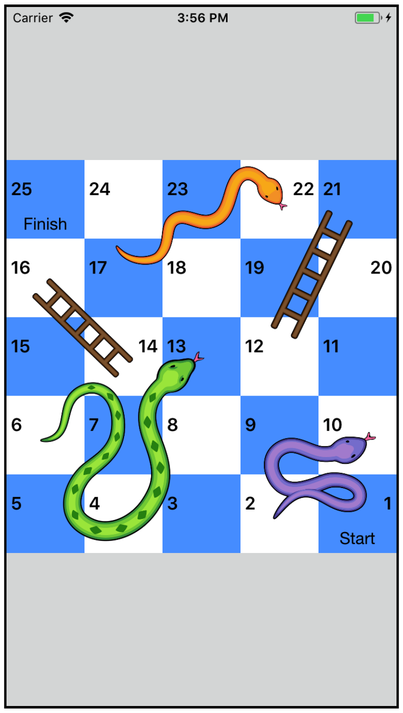
With CocoaPods installed, navigate to the **Demo2\starter** folder and issue the `pod install` command via Terminal to setup the project.

Once this completes, open the .xcworkspace file that CocoaPods generates - **not the original .xcodeproj file** - in Xcode.

Before you get started on making changes, build and run the Ladders and Snakes app (**Command-R**), to confirm everything compiles. You should see the Ladders and Snakes board, with numbered squares, and an indication of where the game starts and finishes.

You'll notice that the numbers in some of the squares have been adjusted to align with the other side of the board. You'll also notice that we've added CocoaPods to the project, so that we can try out SnapKit, a third-party library, in later stages of this demo.

When you've completed Demo 2, your board should look something like the following:



2) Add some Ladders

In the game **Ladders and Snakes**, when a player lands on a square containing the base of a ladder, they are automatically 'climbed' to the top of it. This is one way a player makes progress in the game.

Let's add some ladders to the board.

Navigate to **Other > Assets.xcassets** and look under the **Ladders** folder for the image we'll use to represent a ladder. We will place it on the board in two separate orientations.

Right-Leaning Ladder

Next, press **Command-Shift-O** and perform a search for `LadderView.swift`.

Inspect the file, noticing that protocols `Flippable` and `Rotatable` have already been defined and implemented via extensions on `UIImageView`. A third protocol, `Sizeable` is defined, but is missing its implementation. We'll return to this later.

Locate the following code in `LadderView.swift` which defines a `LadderView`:

```
class LadderView: UIImageView {

}
```

Add a class function, `create(size:rotation:direction:alpha:)`. This function takes some inputs and returns a new `LadderView`, ready for use.

```
class LadderView: UIImageView {

  class func create(size: CGFloat,
                    rotation: CGFloat = 0,
                    direction: Direction = .facesRight,
                    alpha: Alpha = .hidden) -> LadderView {
    let ladder = LadderView(image: #imageLiteral(resourceName: "ladder"))
    ladder.alpha = alpha.rawValue
    ladder.contentMode = .scaleAspectFit
    ladder.flip(to: direction)
    ladder.rotate(degrees: rotation)
    ladder.addSizeConstraints(width: size, height: size)
    return ladder
  }
}
```

Note that the `ladder.addSizeConstraints(width:height:)` method is currently empty. You will return to complete this later in the tutorial.

Next, open `BoardViewController` and add a `addRightLeaningLadder(to:)` method:

```
@discardableResult
private func addRightLeaningLadder(to view: UIView) -> LadderView {
  let span = squareWidth * 2
  let ladder = LadderView.create(size: span, direction: .facesRight)
  view.addSubview(ladder)

  let guide = view.layoutMarginsGuide
  ladder.centerYAnchor.constraint(equalTo: guide.centerYAnchor, constant:
-squareWidth).isActive = true
  ladder.rightAnchor.constraint(equalTo: guide.rightAnchor).isActive =
true
  return ladder
}
```

As its name suggests, this method creates a `LadderView`, and ensures it leans to the right.

`NSLayoutAnchor` APIs are used here to position the LadderView relative to the `view` parameter. Aligning simple aspects of the `LadderView` (edges, centers) to its superview works well with these APIs.

The right anchors of the `LadderView` and `BoardView` are aligned (ensuring the ladder view is flush with the right side of the board), and the `LadderView` center anchor is offset 'half a square's width' from the top of the board.

Next, call this from `addObstaclesToBoard`:

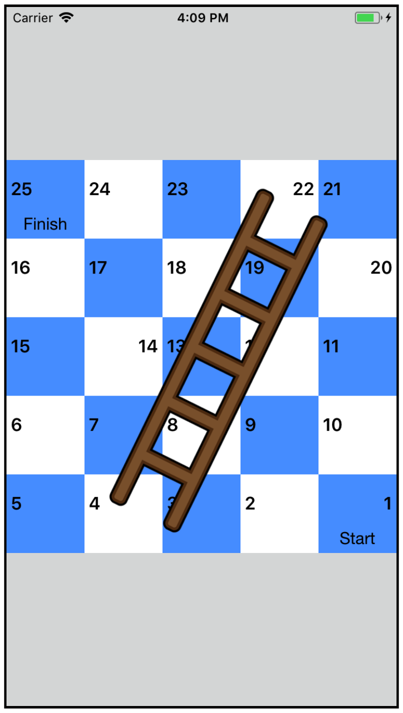
```
let right = addRightLeaningLadder(to: board)
```

Edit the `obstacles` array to include this new item so that it is faded in when the screen loads:

```
let obstacles = [right, top, bottomLeft, bottomRight]
```

You've now successfully added a ladder to your board. Build and run the project (**Command-R**). How does it look?

Don't be alarmed if it doesn't look quite right. Below is a representation of what you should see when the application launches.



The ladder has been placed on the board, and is leaning right, but has no size constraints applied to it currently. Let's return to `LadderView.swift` and use the Visual Format Language (VFL) to add constraints to size the view before placing it on the board.

Add the following to `addSizeConstraints(width:height:)`:

```
translatesAutoresizingMaskIntoConstraints = false
```

This line tells the compiler that you are planning to set Auto Layout constraints in code for this view, and it should *not* attempt to generate any constraints for the view based on its current auto-resizing mask.

> **Note**: For more details on `translatesAutoresizingMaskIntoConstraints` see the relevant Apple Documentation.

Next, define the views that will be involved:

```
  let views = ["sizeable" : self]
  let widthFormat = "[sizeable(\(width))]"
  let heightFormat = "V:[sizeable(\(height))]"
```

Here we create a `views` dictionary that indicates that the `self` parameter (in our current case, an instance of a `LadderView`) should be referred to as `sizeable`. We then create two constraint formats in VFL referencing it - one for width and one for height of the view.

Turn these format strings into constraints by adding the following lines:

```
  let widthConstraints = NSLayoutConstraint.constraints(withVisualFormat:
  widthFormat, metrics: nil, views: views)
  let heightConstraints = NSLayoutConstraint.constraints(withVisualFormat:
  heightFormat, metrics: nil, views: views)
```

Finally, activate them:

```
  let all = widthConstraints + heightConstraints
  NSLayoutConstraint.activate(all)
```

Build and run your project again. The ladder should now be in a much more appropriate location and be a more workable size.

Left-Leaning Ladder

Let's place one more ladder on the board. Define the following method in `BoardViewController.swift`:

```
@discardableResult
private func addLeftLeaningLadder(to view: UIView) -> LadderView {
  let ladder = LadderView.create(size: squareWidth * 1.8, rotation: 20,
direction: .facesLeft)
  view.addSubview(ladder)

  // NSLayoutAnchor abstracted - see UIView+NSLayoutAnchor.swift
  ladder.addCenterYAnchor(to: view, constant:  -squareWidth / 3)
  ladder.addLeftAnchor(to: view)

  return ladder
}
```

This method will generate a ladder that is a similar size to the one we just created, but faces left and is rotated 20 degrees. It is anchored to the left side of our board, and is offset from the top by a third of a square.

This method makes use of extension methods defined in `UIView+NSLayoutAnchor.swift`; an effort to make Auto Layout in code easier to read.

Call `addLeftLeaningLadder(to:)` from `addObstaclesToBoard`, making sure it is added *below* existing code:

```
let left = addLeftLeaningLadder(to: board)
```

Edit the `obstacles` array to include this new item so that it is faded in when the screen loads:

```
let obstacles = [left, right, top, bottomLeft, bottomRight]
```

Build and run the project again (**Command-R**). Your board should now have two usable ladders on it -

- One leaning left, reaching from square 14 to 16
- One leaning right, reaching from square 12 to 21

You're now ready to add some snakes to your board !

3) Add some Snakes

When a player lands on a square that contains the head of a snake in Ladders and Snakes, they are automatically 'slithered' to the square containing the tail. Let's add some of these obstacles to the board.

Navigate to **Other > Assets.xcassets** and look under the **Snakes** folder for the three different snakes we can add to the board - orange, purple or green.

Press **Command-Shift-O** and search for `SnakeView.swift`. Open the file and note that a `Color` enum captures each of the snakes we can add to the board.

Similar to the approach we took for `LadderView`, define a `SnakeView` class with a class method for construction:

```
class SnakeView: UIImageView {
  class func create(size: CGFloat, color: Color = .orange, direction: 
Direction = .facesRight, alpha: Alpha = .hidden) -> SnakeView {
    let snake = UIImage(named: color.rawValue)
    let snakeView = SnakeView(image: snake)
    snakeView.alpha = alpha.rawValue
    snakeView.contentMode = .scaleAspectFit
    snakeView.flip(to: direction)
    snakeView.addSizeConstraints(width: size, height: size)
    return snakeView
  }
}
```

Note that by default, the above code will create a hidden orange snake that faces right. A `size` value must be supplied.

Next, press **Command-Shift-O** and open the `BoardViewController.swift` file again.

Notice that the following three methods are called under `addObstaclesToBoard`, but they haven't yet been implemented:

```
let top = addTopCenteredSnake(to: board)
let bottomLeft = addBottomLeftSnake(to: board)
let bottomRight = addBottomRightSnake(to: board)
```

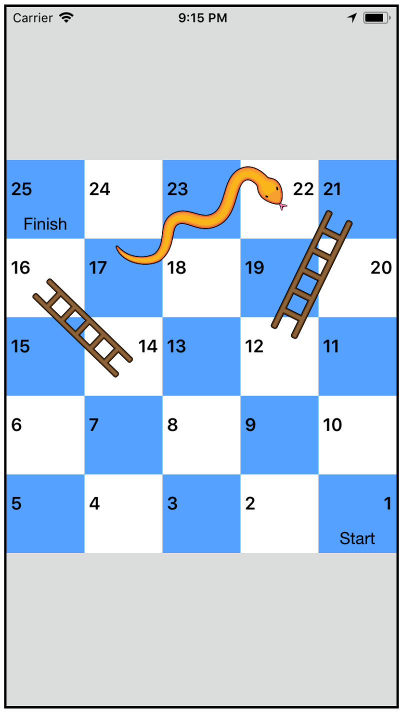
Command-click on `addTopCenteredSnake(to:)` and jump to the definition for that method. Add the following:

```
@discardableResult
private func addTopCenteredSnake(to view: UIView) -> SnakeView {
  let snake = SnakeView.create(size: squareWidth * 2.2)
  view.addSubview(snake)

  snake.addTopAnchor(to: view, constant: -squareWidth / 2)
  snake.addCenterXAnchor(to: view)

  return snake
}
```

Press **Command-R** to build and run the application. You should now see an orange snake appear at the top of the board:



Place two other snakes on the board, by completing the implementation for `addBottomLeftSnake(to:)` and `addBottomRightSnake(to:)`:

```
@discardableResult
private func addBottomLeftSnake(to view: UIView) -> SnakeView {
  let snake = SnakeView.create(size: squareWidth * 2.5, color: .green)
  view.addSubview(snake)
```

```
    snake.addBottomAnchor(to: view)
    snake.addLeftAnchor(to: view, constant: squareWidth / 8)

    return snake
  }

  @discardableResult
  private func addBottomRightSnake(to view: UIView) -> SnakeView {
    let snake = SnakeView.create(size: squareWidth * 1.5, color: .purple, 
  direction: .facesLeft)
    view.addSubview(snake)

    let inset = -squareWidth / 8
    snake.addBottomAnchor(to: view, constant: inset)
    snake.addRightAnchor(to: view, constant: inset)
    return snake
  }
```

Press **Command-R** to build and run the app again.

Congratulations - you should now have a nicely balanced **Ladders and Snakes** board containing three snakes and two ladders.

4) SnapKit

To improve our code even further we will now re-write some of the code we'd previously used with SnapKit. SnapKit is a third-party library for creating Auto Layout constraints in code.

Still in `BoardViewController.swift`, add the following at the top of the file:

```
import SnapKit
```

Find the `addRightLeaningLadder(to:)` method within `BoardViewController.swift`. Replace the following lines

```
let guide = view.layoutMarginsGuide
ladder.centerYAnchor.constraint(equalTo: guide.centerYAnchor, constant: -
squareWidth).isActive = true
ladder.rightAnchor.constraint(equalTo: guide.rightAnchor).isActive = true
```

with

```
ladder.snp.makeConstraints { make in
  make.centerY.equalTo(view).offset(-squareWidth)
  make.right.equalTo(view)
}
```

Build and run the project again (**Command-R**). Notice that nothing has changed - success! We've replaced some difficult-to-read code with some more legible code to create constraints.

If time permits, familiarize yourself with the documentation and see what other constraints you can replace using SnapKit.

5) That's it!

Congratulations, at this time you should have a good understanding of Auto Layout in code. Take a break, take another sip of coffee and then it's time to move on to debugging some tricky Auto Layout issues.

Auto Layout Best Practices: Demo 3

By Gemma Barlow

In this demo, you will debug some new Auto Layout issues with the Ladders and Snakes game.

The steps here will be explained in the demo, the raw steps are provided in case you miss something or get stuck.

> **Note**: Begin work with the starter project in **Demo3\starter**. As per the prerequisites for this tutorial, *CocoaPods should be installed on your machine as it will be needed to build and run this project*.

1) Get Setup

With CocoaPods installed, navigate to the **Demo3\starter** folder and issue the `pod install` command via Terminal to setup the project.

Once this completes, open the .xcworkspace file that CocoaPods generates - **not the original .xcodeproj file** - in Xcode.

Build and run the application by pressing **Command-R**. Note that the game has changed a little since Demo 2 and a rogue colleague seems to have introduced some additional functionality (but also some additional Auto Layout issues!) into the project.

2) Oh no! The Game Crashes on Launch

The first thing you will notice is that the game crashes on launch. This seems as good a place to start debugging as any.

Scroll through the Xcode console looking for the following -

```
*** Terminating app due to uncaught exception 'NSGenericException',
reason: 'Unable to activate constraint with anchors <NSLayoutYAxisAnchor:
0x60c000468e80 "Ladders_and_Snakes.PieceView:0x7f8954724460.centerY"> and
<NSLayoutYAxisAnchor:0x60c000468f00 "Ladders_and_Snakes.BoardSquareView:
0x7f8954512870.centerY"> because they have no common ancestor.  Does the
constraint or its anchors reference items in different view hierarchies?
That's illegal.'
```

This error is helpful - it suggests we are trying to create some constraints between `PieceView` and `BoardSquareView` - a reasonable thing to do when trying to place a game piece on the board.

Open `BoardViewController.swift` in the project and locate `setupPieces`. Note that we are creating a `PieceView` for each player here, and later adding constraints in `move(piece:to:animated)` but do not ever add these pieces to the `BoardView`. The order of these operations is very important and getting it wrong is a common Auto Layout mistake - **you must always add the view to the hierarchy and *then* add the constraints to the view**.

Add the following lines to `setupPieces` -

```
board.addSubview(playerOne)
board.addSubview(playerTwo)
```

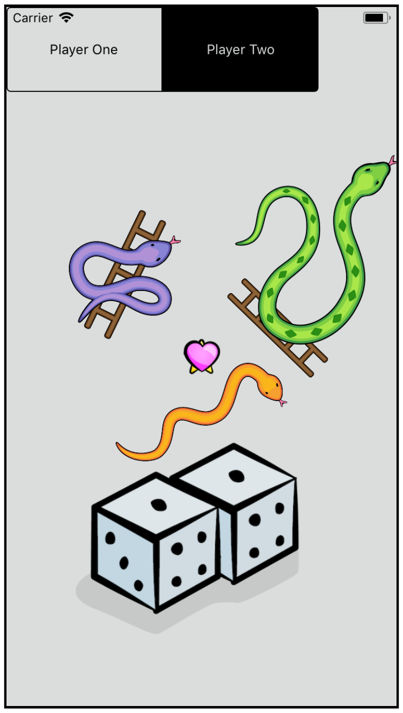
Build and run the project again. It doesn't crash! First debugging achievement unlocked.

3) New Features

With the Simulator launched, rotate the board by pressing **Command-Right Arrow**. Note that the board maintains its aspect ratio when it rotates. Rotate back to Portrait Orientation by pressing **Command-Left Arrow**.

Next, attempt to play the game - tap the dice to get started. You'll notice immediately that the board disappears and the snakes, ladders, and pieces animate to new positions. This is definitely not the intended behavior!

Don't be alarmed if your app suddenly looks like the following:



We'll examine this behavior and the errors it produces shortly, but first let's recap the desired game functionality.

The Ladders and Snakes game should alternate turns between Player One (star) and Player Two (heart). The first to the 'Finish' square wins - but obstacles make things more exciting.

Encountering a snake should see a player *slither* back a few places. Finding the base of a ladder should help them *climb* forward.

Upon rolling the dice, a player's piece should be moved across the board automatically.

4) So many errors in the Xcode Console!

In this section you will do the 'Error Message Interpretative Dance' to fix issues with Auto Layout. Error messages are being printed to the console due to 'Unsatisfiable Constraints'.

After starting to play the game, you'll notice quickly that:

- as the pieces move across the board Auto Layout errors start being logged to the console
- the board disappears altogether

- the snakes, ladders, and pieces animate to new positions

You are not left with many debugging options in situations like this.

One tool that is useful for long error messages logged to the console is **wtfautolayout.com**.

Search for the following log from the last constraints issue outputted:

```
(
"<NSAutoresizingMaskLayoutConstraint:0x6000000901d0 h=-&- v=-&-
Ladders_and_Snakes.BoardSquareView:0x7fa38d420ad0.midX ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50ff90.midX
(active)>",
"<NSAutoresizingMaskLayoutConstraint:0x600000090c70 h=-&- v=-&-
Ladders_and_Snakes.BoardSquareView:0x7fa38d421fd0.midX ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630.midX
(active)>",
"<NSAutoresizingMaskLayoutConstraint:0x600000090d60 h=-&- v=-&-
Ladders_and_Snakes.BoardSquareView:0x7fa38d421fd0.height ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630.height
(active)>",
"<NSLayoutConstraint:0x604000097a70
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50ff90.width ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50ff90.height
(active)>",
"<NSLayoutConstraint:0x604000097ac0
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0.width ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0.height
(active)>",
"<NSLayoutConstraint:0x608000286bd0 UILayoutGuide:
0x6080001b7d80'UIViewSafeAreaLayoutGuide'.bottom == UIStackView:
0x7fa38d4221c0.bottom + 4   (active)>",
"<NSLayoutConstraint:0x608000286c70 UIStackView:0x7fa38d4221c0.top ==
UILayoutGuide:0x6080001b7d80'UIViewSafeAreaLayoutGuide'.top + 4
(active)>",
"<NSLayoutConstraint:0x604000097b10
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630.width ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630.height
(active)>",
"<SnapKit.LayoutConstraint:0x6040000bc6e0@BoardView.swift#97
Ladders_and_Snakes.PieceView:0x7fa3900016e0.centerX ==
Ladders_and_Snakes.BoardSquareView:0x7fa38d421fd0.centerX>",
"<SnapKit.LayoutConstraint:0x6000000bfe60@BoardView.swift#97
Ladders_and_Snakes.PieceView:0x7fa3900016e0.centerX ==
Ladders_and_Snakes.BoardSquareView:0x7fa38d420ad0.centerX>",
"<NSLayoutConstraint:0x600000091080 'UISV-alignment'
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50f8f0.bottom ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50ff90.bottom
(active)>",
"<NSLayoutConstraint:0x6000000910d0 'UISV-alignment'
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50f8f0.bottom ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0.bottom
(active)>",
"<NSLayoutConstraint:0x600000091120 'UISV-alignment'
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50f8f0.bottom ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630.bottom
(active)>",
"<NSLayoutConstraint:0x6000000911c0 'UISV-alignment'
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50f8f0.top ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50ff90.top   (active)>",
"<NSLayoutConstraint:0x600000091210 'UISV-alignment'
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50f8f0.top ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0.top   (active)>",
"<NSLayoutConstraint:0x600000091260 'UISV-alignment'
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50f8f0.top ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630.top   (active)>",
"<NSLayoutConstraint:0x600000090f40 'UISV-spacing' H:
[Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50ff90]-(0)-
[Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0]   (active)>",
"<NSLayoutConstraint:0x600000090f90 'UISV-spacing' H:
[Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0]-(0)-
[Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630]   (active)>",
"<NSLayoutConstraint:0x608000286b30 'UIViewSafeAreaLayoutGuide-bottom' V:
[UILayoutGuide:0x6080001b7d80'UIViewSafeAreaLayoutGuide']-(0)-|
(active, names: '|':Ladders_and_Snakes.BoardSquareView:
0x7fa38d421fd0 )>",
"<NSLayoutConstraint:0x608000286a90 'UIViewSafeAreaLayoutGuide-top' V:|-
(0)-[UILayoutGuide:0x6080001b7d80'UIViewSafeAreaLayoutGuide']   (active,
names: '|':Ladders_and_Snakes.BoardSquareView:0x7fa38d421fd0 )>"
)
```

To help visualize the issue, copy and paste it into **wtfautolayout.com**.

Note that `PieceView` is mentioned in the list: the associated constraint suggests that its horizontal center should be aligned with `BoardSquareView1`.

Scroll to the top of the console and review another set of constraints.

Again, copy and paste the following log into **wtfautolayout.com**, for visualization:

```
(
"<NSAutoresizingMaskLayoutConstraint:0x6000000901d0 h=-&- v=-&-
Ladders_and_Snakes.BoardSquareView:0x7fa38d420ad0.midX ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50ff90.midX
(active)>",
"<NSAutoresizingMaskLayoutConstraint:0x6000000907c0 h=-&- v=-&-
Ladders_and_Snakes.BoardSquareView:0x7fa38d418fb0.height ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0.height
(active)>",
"<NSAutoresizingMaskLayoutConstraint:0x600000090c70 h=-&- v=-&-
Ladders_and_Snakes.BoardSquareView:0x7fa38d421fd0.midX ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630.midX
(active)>",
"<NSLayoutConstraint:0x604000097a70
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50ff90.width ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50ff90.height
(active)>",
"<NSLayoutConstraint:0x608000286310 UILabel:0x7fa38d421cf0.height == 20
(active)>",
"<NSLayoutConstraint:0x608000286630 UILayoutGuide:
0x6080001b7bc0'UIViewSafeAreaLayoutGuide'.bottom == UIStackView:
0x7fa38d421650.bottom + 4   (active)>",
"<NSLayoutConstraint:0x6080002866d0 UIStackView:0x7fa38d421650.top ==
UILayoutGuide:0x6080001b7bc0'UIViewSafeAreaLayoutGuide'.top + 4
(active)>",
"<NSLayoutConstraint:0x604000097ac0
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0.width ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0.height
(active)>",
"<NSLayoutConstraint:0x604000097b10
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630.width ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630.height
(active)>",
"<SnapKit.LayoutConstraint:0x6040000bc6e0@BoardView.swift#97
Ladders_and_Snakes.PieceView:0x7fa3900016e0.centerX ==
Ladders_and_Snakes.BoardSquareView:0x7fa38d421fd0.centerX>",
"<SnapKit.LayoutConstraint:0x6000000bfe60@BoardView.swift#97
Ladders_and_Snakes.PieceView:0x7fa3900016e0.centerX ==
Ladders_and_Snakes.BoardSquareView:0x7fa38d420ad0.centerX>",
"<NSLayoutConstraint:0x600000091080 'UISV-alignment'
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50f8f0.bottom ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50ff90.bottom
(active)>",
"<NSLayoutConstraint:0x6000000910d0 'UISV-alignment'
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50f8f0.bottom ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0.bottom
(active)>",
"<NSLayoutConstraint:0x600000091120 'UISV-alignment'
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50f8f0.bottom ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630.bottom
(active)>",
"<NSLayoutConstraint:0x6000000911c0 'UISV-alignment'
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50f8f0.top ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50ff90.top   (active)>",
"<NSLayoutConstraint:0x600000091210 'UISV-alignment'
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50f8f0.top ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0.top   (active)>",
"<NSLayoutConstraint:0x600000091260 'UISV-alignment'
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50f8f0.top ==
Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630.top   (active)>",
"<NSLayoutConstraint:0x600000090450 'UISV-canvas-connection' UIStackView:
0x7fa38d421650.top == UILabel:0x7fa38d421870'2'.top   (active)>",
"<NSLayoutConstraint:0x6000000904a0 'UISV-canvas-connection' V:[UILabel:
0x7fa38d421cf0]-(0)-|   (active, names: '|':UIStackView:
0x7fa38d421650 )>",
"<NSLayoutConstraint:0x6000000904f0 'UISV-spacing' V:[UILabel:
0x7fa38d421870'2']-(0)-[UILabel:0x7fa38d421cf0]   (active)>",
"<NSLayoutConstraint:0x600000090f40 'UISV-spacing' H:
[Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d50ff90]-(0)-
[Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0]   (active)>",
"<NSLayoutConstraint:0x600000090f90 'UISV-spacing' H:
[Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d5103e0]-(0)-
[Ladders_and_Snakes.BoardSquareViewLoader:0x7fa38d510630]   (active)>",
"<NSLayoutConstraint:0x608000286590 'UIViewSafeAreaLayoutGuide-bottom' V:
[UILayoutGuide:0x6080001b7bc0'UIViewSafeAreaLayoutGuide']-(0)-|
(active, names: '|':Ladders_and_Snakes.BoardSquareView:
0x7fa38d418fb0 )>",
"<NSLayoutConstraint:0x6080002864f0 'UIViewSafeAreaLayoutGuide-top' V:|-
(0)-[UILayoutGuide:0x6080001b7bc0'UIViewSafeAreaLayoutGuide']   (active,
names: '|':Ladders_and_Snakes.BoardSquareView:0x7fa38d418fb0 )>"
```

```
)
```

Note that `PieceView` is again mentioned - this time `BoardSquareView3` and `BoardSquareView2`- related constraints.

Given:

- the timing of the error (it occurs when board pieces are animating); and
- the fact that three constraints for this class type are mentioned, but we only have two pieces on the board

this class seems like a good place to start exploring the code.

Inspect the application of constraints to a piece view when the dice is rolled - press **Command-Shift-O** to open `BoardView.swift` and navigate to `adjustConstraintsToMovePiece(piece:to:)`:

```
private func adjustConstraintsToMovePiece(piece: PieceView, to square:
BoardSquareView) {
  piece.snp.makeConstraints { make in
    make.center.equalTo(square)
  }
}
```

Each time this code is called, we are placing additional constraints on `piece` but never removing any of the old ones. We can do this simply via SnapKit using `remakeConstraints(completion:)`:

```
private func adjustConstraintsToMovePiece(piece: PieceView, to square:
BoardSquareView) {
  piece.snp.remakeConstraints { make in
    make.center.equalTo(square)
  }
}
```

Build and run the project again. Success - most of the errors will have disappeared.

Now that the functionality appears to be working, try playing a full game - start by tapping to roll the dice. Note that you will still see some visual errors along the way. Once you've completed the game, tap 'Play Again' to reset the board.

5) Why does the Segmented Control look strange?

In this section you will practice the 'Just Re-Do It' technique to fix issues with Auto Layout via Interface Builder.

Having run the application once, you will have noticed that the `UISegmentedControl` at the top of the screen looks strange. You may also have noticed errors in Interface Builder. Press **Command-Shift-O** to open `Main.storyboard` and investigate.

Errors should be visible upon opening the Storyboard. Click the red arrow at the top of the pane to inspect them.

Interface Builder is complaining due to Ambiguous Layout ('Missing Contraints') and Unsatisfiable Constraints ('Conflicting Constraints') in this scenario. An odd aspect ratio (23:4) seems to have been applied here - as well as some very specific width and height constraints.

Given that we'd just like to center the segmented control between the board and the top of the screen, this seems like too many contraints already - but let's see if Xcode can help us out. Next to 'Segmented Control: Need Constraints for Y Position' click the red Fix It tooltip and 'Add Missing Contraints'. Observe the change in the Visual Editor.

Oh dear, definitely worse. Let's try a different approach.

Press the 'Back' button to return to the 'Board View Controller Scene' and select the Segmented Control. With the control selected, navigate to the following menu item 'Editor > Resolve AutoLayout Issues > Clear Constraints' (the first one, under the 'Selected Views' heading).

Note: I use this regularly enough in my work that I've mapped it to a key command in Xcode. I've chosen Command-E, but as it conflicts with an existing default key-mapping - **Use Selection for Find** - you'll need to remove that mapping or choose another combination.

You can read more about key-mapping in Xcode by searching the 'Help' menu for 'Adjust keyboard shortcut mappings'.

After this action, things should start to look a little better. With the control still selected Control-Drag between the Segmented Control and the Top Container. Press and hold the **Shift** key to set multiple constraints at once - 'Center Horizontally in Container' and 'Center Vertically In Container' and then pick 'Add Constraints'.

Finally, use the 'Update Frames' option under the warnings menu in Interface Builder to address the 'Misplaced Views' warning. The control should now be appropriately centered in the view. Build and run the application (**Command-R**) to confirm things look better at runtime.

6) Why is the dice image stretched?

In this section you will practice 'Visualization' techniques to fix Auto Layout Issues in your application.

The dice image being stretched is an example of the - you guessed it! - 'Doesn’t Look Quite Right' issue type.

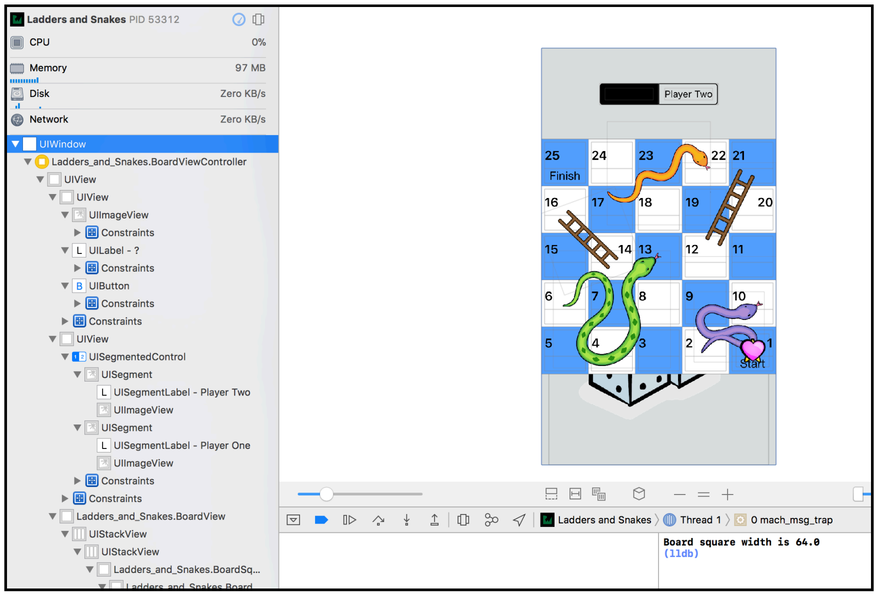
With the application still running, note that the `UISegmentedControl` is now fixed, but the dice image doesn't look good. Open `Main.storyboard` and select the dice view.

Note: The dice view is a UIImage hidden behind a button and may be difficult to select using the Visual Editor. Pressing Shift-Right Click in the general area will allow you to view a hierarchy of overlapping views to select it.

Inspect the constraints applied to the view. We see that a constraint indicating that the width of the image should be 150pt is present, as is an 'Aspect Ratio' contraint that should keep the image as a square. Note also the difference between the Visual Editor and what we're seeing in the Simulator.

To debug this further, we're going to need to inspect it at runtime. With the application still running, click the 'Debug View Hierarchy' button.

Your Xcode windows should suddenly adjust, and you should be presented with a debugging view similar to the image below.



```
Board square width is 64.0
(lldb)
```

Drag the view around to get a full sense of what it can do; there's a lot of excellent visualization functionality here and we'll only get time to dive into a small piece of it.

Next, locate the `UIImageView` representing the dice in the left-hand pane, and take a look in the Size Inspector at the constraints currently applied. Note that it has a width and height of `300 (content size)`. These `(content size)` constraints are applied by iOS at runtime - our specified constraint of `width = 150` is not working.

Stop the Simulator and return to `Main.storyboard`. Locate the `width = 150` constraint that is applied to the `UIImageView` and inspect its details more carefully. Note that a checkbox for **Remove at build time** has been (accidentally) checked here. Click to uncheck it, then build and run to confirm the dice now displays at runtime as expected.

7) That's it!

Congratulations on practicing three techniques that will aid in debugging your future Auto Layout headaches. Celebrate with a few more rounds of Ladders and Snakes.

If you find yourself with extra time at the end of the tutorial, consider adding animation to celebrate a player winning the game. You can find examples of animation using Auto Layout throughout the sample code, providing a starting point.