BEGINNING iOS ANIMATIONS

Hands-On Challenges

Beginning iOS Animations

Catie & Jessy Catterwaul

Notice of Rights

Notice of Liability

Trademarks

Table of Contents: Overview

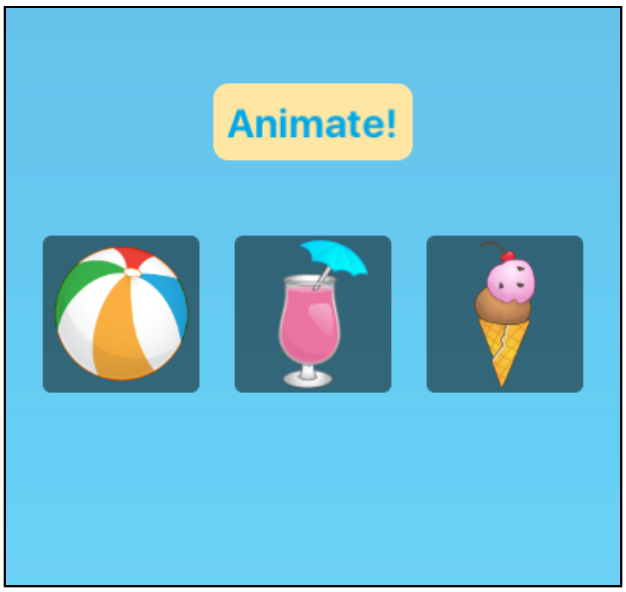
Table of Contents: Extended

4 Challenge 4: Spring Animations

By Catie & Jessy Catterwaul

Part 1 - Experiment with Springs!

The springs playground contains all of the instructions you need for this part of the challenge.



Part 2 - Back to the App

This challenge is not about adding new code, but about making decisions in spring animations.

As you saw in the video, creating spring animations is very easy - you just add two more parameters to your animation call and UIKit takes care of the rest. However, really learning how to model spring animations takes a bit of practice.

If you spent some time in the playground, hopefully you've gained some insight from trying different values for the spring damping and velocity parameters. Now, you'll apply this back in the project.

Spring damping

Let's play a bit more with the spring animation you created previously in `showItem()`. Right now you have a 0.4 and you get a nice bounce at the end of the animation.

What do you think the result will be if you increased the spring damping? Replace the value with `0.9` and observe the result.

```
//Animate in
UIView.animate(...
  usingSpringWithDamping: 0.9,
  initialSpringVelocity: 10.0,
  animations: {.....
)
```

This time the animation doesn't have any bouncing, but it does have a smooth easing towards the end.

Try few more values:

- `0.8` - produces a smooth animation with a very subtle bounce
- `0.5` - gives the animation a big bounce, followed by a smaller, subtler finish
- `0.25` - makes the preview jiggle a fair bit at the end. To use so little damping, you also need to increase the duration of the animation so the user can enjoy the effect better.

You see the pattern - the smaller the damping the more the view bounces at the end of the animation. The closer the parameter is to `1.0` the less bouncing you get at the end of the animation. If you try `1.0` you get straight movement from point A to B.

There is one more animation that you haven't tried to convert yet. The one that animates the preview off screen. Try converting that into a spring animation:

```
UIView.animate(
  withDuration: 0.8,
  delay: 1.0,
  usingSpringWithDamping: 0.4,
  initialSpringVelocity: 0.0,
  animations: {
    conBottom.constant = imageView.frame.size.height
    conWidth.constant = -50.0
    self.view.layoutIfNeeded()
  },
  completion: {_ in
```

```
      imageView.removeFromSuperview()
    }
  )
```

Does that change anything about the animation, visually? Why?

Initial velocity

Now let's try the `initialSpringVelocity` parameter - its value sets the initial velocity of the animated view.

If you set this parameter to `1.0`, and the animation moves the view 50 points across the screen - it will give the view 50 points/sec of initial velocity. If you set the parameter to `2.0` - the view will have 100 points/sec initial velocity.

Try setting `initialSpringVelocity` for the first animation in `showItem()` to `1.0` and run the app.

```
  //Animate in
  UIView.animate(...
    usingSpringWithDamping: 0.25,
    initialSpringVelocity: 1.0,
    animations: {.....
  )
```

Nothing much changes. Apparently this velocity isn't enough to alter the animation.

Try a much bigger value - set `initialSpringVelocity` to `100.0` and run the app again.

```
  //Animate in
  UIView.animate(...
    usingSpringWithDamping: 0.25,
    initialSpringVelocity: 100.0,
    animations: {.....
  )
```

This time the preview overshoots the end point by a lot thanks to its initial velocity. Playing with other values besides the two you've already tried will alter how much the view overshoots the end point, proportionally.

That's all for this challenge - take a break and head to the next video where you will learn about view transition animations.