CS275
Fall 2020

iOS: Controlling Animations

BNRG CHAPTER 8

Topics

- Animations
- Completion functions
- Programmatic control of constraints

Animations

Animations are UI aids

- they can bring an interface element into focus
- they can draw the user's attention to a particular item that needs action
- they can provide active feedback
- etc.

It's important to be aware of the *First Rule of UI Design*:

- "Just because you *can* do something doesn't mean you *should* do it."

note: this is my First Rule of UI Design, but that fact doesn't diminish its value

However, animations, used properly, can increase the quality and effectiveness of a UI

Basic Animation

Basic animation: changes the visual appearance of a UI element

- by changing gradually from a start value to an end value

Alpha value: the degree of transparency

- by animating the alpha value of a label, the label can fade in or fade out

Basic Animation

Basic animation comes through a class method on `UIView`

- remember: a class method is a static function defined on a class itself

```
class func animate(withDuration duration: TimeInterval, animations: () -> void)
```

The function has two parameters

- `TimeInterval` (an alias for Double)
- `animations`: a closure

Animating a Label to Fade In

We can define a closure that specifies what should happen to the property in question

- initially, we'll set the alpha value to 0
- the closure will set it to 1

```
let animationClosure = { () -> Void in
    self.questionLabel.alpha = 1
}
```

Animating a Label to Fade In

And then use the closure in `UIView.animate(withDuration:animations:)`

```
func animateLabelTransitions() {
    let animationClosure = { () -> Void in
        self.questionLabel.alpha = 1
    }
    // animate the alpha
    UIView.animate(withDuration: 0.5, animations: animationClosure)
}
```

Then, if we set `questionLabel.alpha` to zero initially, this function will cause the label to fade in

- set up the animication this in `viewWillAppear(_:)`

Animating a Label to Fade In

Or, even more succinctly:

```
func animateLabelTransitions() {
    // animate the alpha
    UIView.animate(withDuration: 0.5, animations: {
        self.questionLabel.alpha = 1
    })
}
```

here, the closure is inline

along with:

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    self.questionLabel.alpha = 0
}
```

Fade-Out / Fade-In

In order to transition smoothly from one label to another (so that the second label replaces the first)

- need to have two `UILabel` objects
- fade the first one out and the second one in, simultaneously

```
func animateLabelTransitions() {
    UIView.animate(withDuration: 1.5, animations: {
        self.currentQuestionLabel.alpha = 0
        self.nextQuestionLabel.alpha = 1
    })
}
```

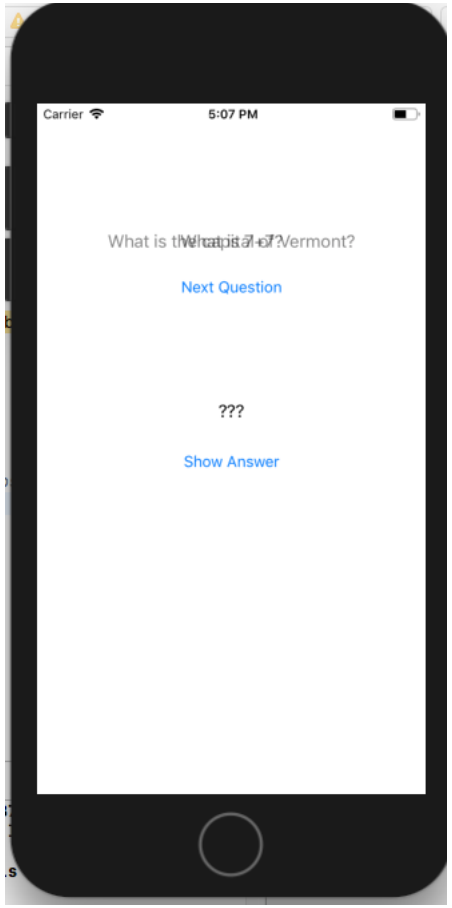
- and the alpha value for the first starts at 1, whereas the alpha value for the second starts at 0

Fade-Out / Fade-In

And the way to fade out one label and fade in another is actually to have two labels

- set the text in label #2 to the desired value
- animate the alpha of label #1 from 1 to 0
- animate the alpha of label #2 from 0 to 1

This will make it appear that the second label is replacing the first label



Animation Completions

In order to do the fade-in/fade-out correctly, we have to know when an animation has completed

- the animation will animate the alpha of label #1 from 1 to 0 and the alpha of label #2 from 0 to 1
- then, we have to swap the labels, so that the text for label #1 is the next question

But we can only swap the text when we know that the animation is complete

- we can give the `UIView.animate()` method a completion handler: a closure that will be executed with the animation is complete

Completion Handler

Here's the full label-animation method:

```
func animateLabelTransitions() {
    UIView.animate(withDuration: 1.5,
                   delay: 0,
                   animations: {
                       self.currentQuestionLabel.alpha = 0
                       self.nextQuestionLabel.alpha = 1},
                   completion: { (_ flag) -> Void in
                       swap(&self.currentQuestionLabel, &self.nextQuestionLabel)}
    )
}
```

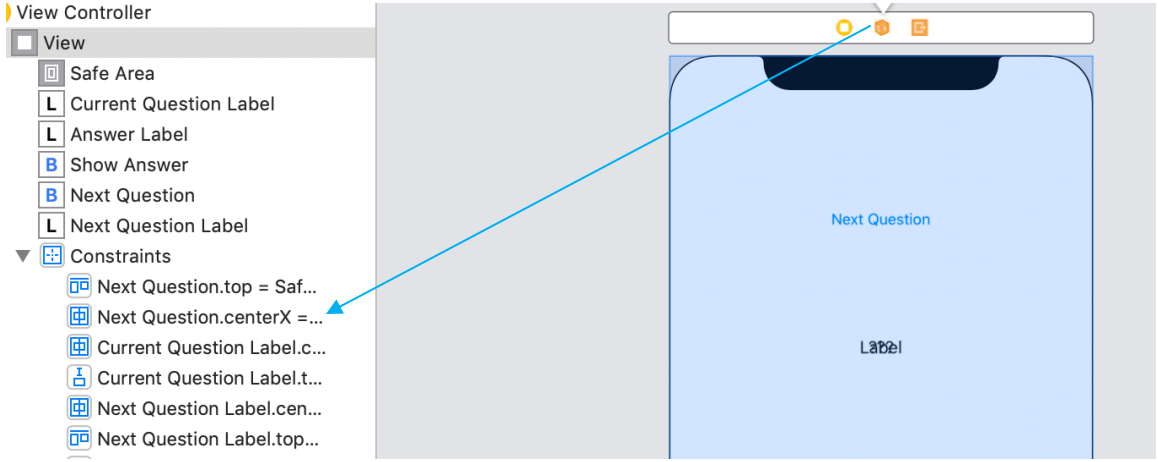
More concisely:

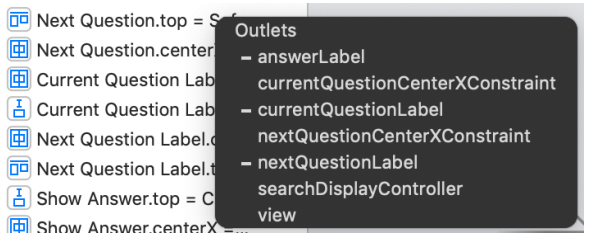
```
completion: { _ in swap(&self.currentQuestionLabel, &self.nextQuestionLabel)}
```

Animating Constraints

- We can refer in code to any object that is defined in a storyboard
  - not just UI elements—constraints as well
- An `@IBOutlet` is reference to a storyboard object
- In code, constraints are instances of `NSLayoutConstraint`

control drag, then select the `@IBOutlet` you want





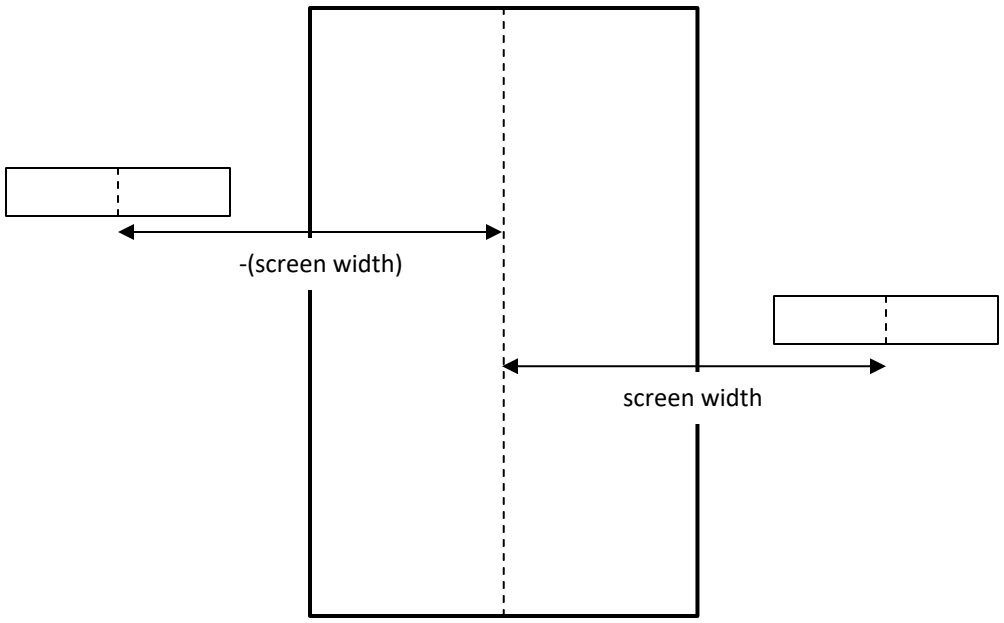
Animating Constraints

To make a label fly in from the left:

- animate its "center x in container" constraint
- set the constant for this constraint to be -(screen width) initially
- then set it to be zero

To make it fly off to the right

- set the constant for the "center x in container" constraint to be the screen width



layoutIfNeeded()

- One slight little gotcha: the effect of changing constraints by default won't be animated
- To see the animation of the constraints, must call `layoutIfNeeded()` on the parent view

```
UIView.animate(withDuration: 1.5,
                        delay: 0,
                        animations: {
                            self.currentQuestionLabel.alpha = 0
                            self.nextQuestionLabel.alpha = 1
                            self.view.layoutIfNeeded()},
                        completion: { (_ flag) -> Void in
                            swap(&self.currentQuestionLabel, &self.nextQuestionLabel)
                            swap(&self.currentQuestionCenterXConstraint,
                                 &self.nextQuestionCenterXConstraint)
                         self.updateOffScreenLabel()}
```

Timing Functions

Timing functions control the interpolation of start value to end value in an animation

For example: ease-in / ease-out

- start out slow, gain speed, then reduce speed gradually

Other functions available include ease-in / abrupt-stop, etc.

Conclusion

A small GUI embellishment, which is easily accessible from the UIKit, can make a boring interface look more professional

Challenges: Try These for Fun!

- The Bronze Challenge: using "spring animation"
  - see https://developer.apple.com/documentation/uikit/uiview/1622594-animatewithduration
  - and I would say also: disable the "Next Question" button during the animation; see UIButton.isUserInteractionEnabled
- Silver Challenge
  - not necessary to change the code
  - view.frame.width takes into account the rotation!