CUSTOM CONTROLS IN iOS

HANDS-ON CHALLENGES

Custom Controls in iOS

Catie & Jessy Catterwaul

Challenge #3: Interaction & UIControl

By Catie & Jessy Catterwaul

In this action-packed challenge, you will add visual feedback to the user when the button is tapped by animating the background color and label appearance!

Go for animating the label appearance first. When the Deluxe Button is tapped, you want the label to squash down and the image to expand to take up the entire control. Because we used a stack view, we can achieve this merely by changing the `isHidden` property on the label when the control is tapped!

At the bottom of DeluxeButton.swift, create a public extension and override `touchesBegan`, `touchesCancelled`, and `touchesEnded`.

```
public extension DeluxeButton {
  override func touchesBegan(
    _ touches: Set<UITouch>,
    with event: UIEvent?
  ) {
    super.touchesBegan(touches, with: event)
  }

  override func touchesCancelled(
    _ touches: Set<UITouch>,
    with event: UIEvent?
  ) {
    super.touchesCancelled(touches, with: event)
  }

  override func touchesEnded(
    _ touches: Set<UITouch>,
    with event: UIEvent?
  ) {
    super.touchesEnded(touches, with: event)
  }
}
```

In the same extension, below these methods, create a function to call from the "touches" methods. The function will take a `Bool` to indicate whether the label should be hidden or not, based on if the Deluxe Button is currently "pressed".

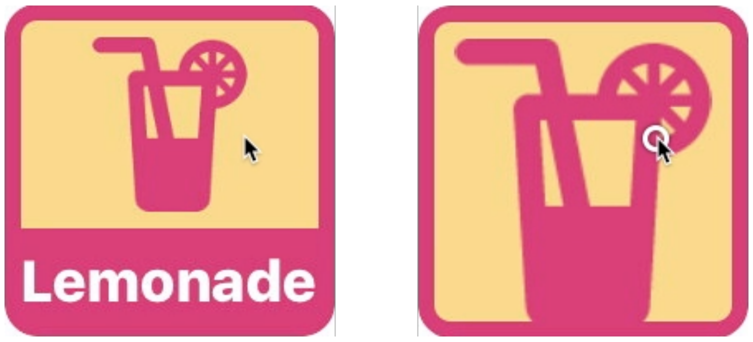
```
private func animate(isPressed: Bool) {
  UIView.animate(withDuration: 0.5){
    self.label.isHidden = isPressed
  }
}
```

Now call this function from each of the `touches...` methods.

```
// in touchesBegan
animate(isPressed: true)

// in touchesCancelled and touchesEnded
animate(isPressed: false)
```

Test it out in the live view!



That's looking pretty spiffy! It should go a bit faster, though. You'll fix that up in a minute as you animate the background color!

At the top of `DeluxeButton.swift`, add two new color variables and set them to whatever defaults you'd like:

```
//MARK: public
  public var pressedBackgroundColor = #colorLiteral(
    red: 0.8, green: 0.8, blue: 0.8, alpha: 1
  )

  public var unpressedBackgroundColor = #colorLiteral(
    red: 0.47, green: 0.84, blue: 0.98, alpha: 1
  ) {
    didSet {
      backgroundColor = unpressedBackgroundColor
    }
  }
```

Back in `animate(isPressed:)`, start by deciding the desired animation duration, background color, and the `isHidden` state for the label, based on the isPressed parameter:

```
    let (
      duration,
      backgroundColor,
      labelIsHidden
    ) = {
      isPressed
      ? (
        duration: 0.05,
        backgroundColor: pressedBackgroundColor,
        labelIsHidden: true
      )
      : (
        duration: 0.1,
        backgroundColor: unpressedBackgroundColor,
        labelIsHidden: false
      )
    }()
```

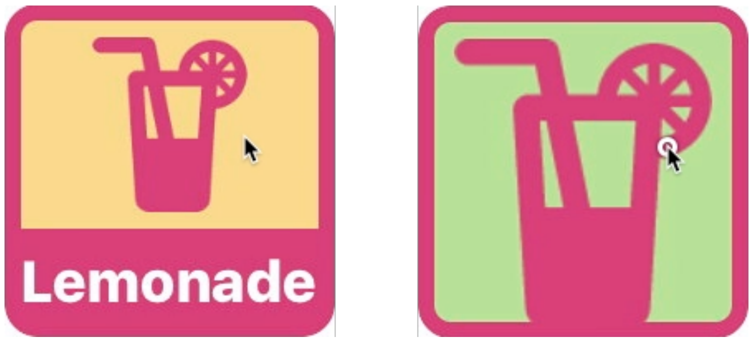
Update `UIView.animate(withDuration:)` to use these constants.

```
      UIView.animate(withDuration: duration){
        self.backgroundColor = backgroundColor
        self.label.isHidden = labelIsHidden
      }
```

In the Playground, replace the assignment of `deluxeButton.backgroundColor` with some fancy colors for the two new color variables.

```
  deluxeButton.unpressedBackgroundColor = #colorLiteral(
    red: 0.98, green: 0.85, blue: 0.55, alpha: 1
  )
  deluxeButton.pressedBackgroundColor = #colorLiteral(
    red: 0.72, green: 0.89, blue: 0.6, alpha: 1
  )
```

Finally, test out your finished animation in the live view!



Excellent! Now your Deluxe Button can both do something and appear as if it is doing something.