STL CocoaHeads
Circular Progress Views

Brian Coyner

Principal Software Engineer
NISC

Session Topics

- Techniques for building circular progress views
- Strategy design pattern
- Demo
- Fun with emitter layers
- Other awesome uses

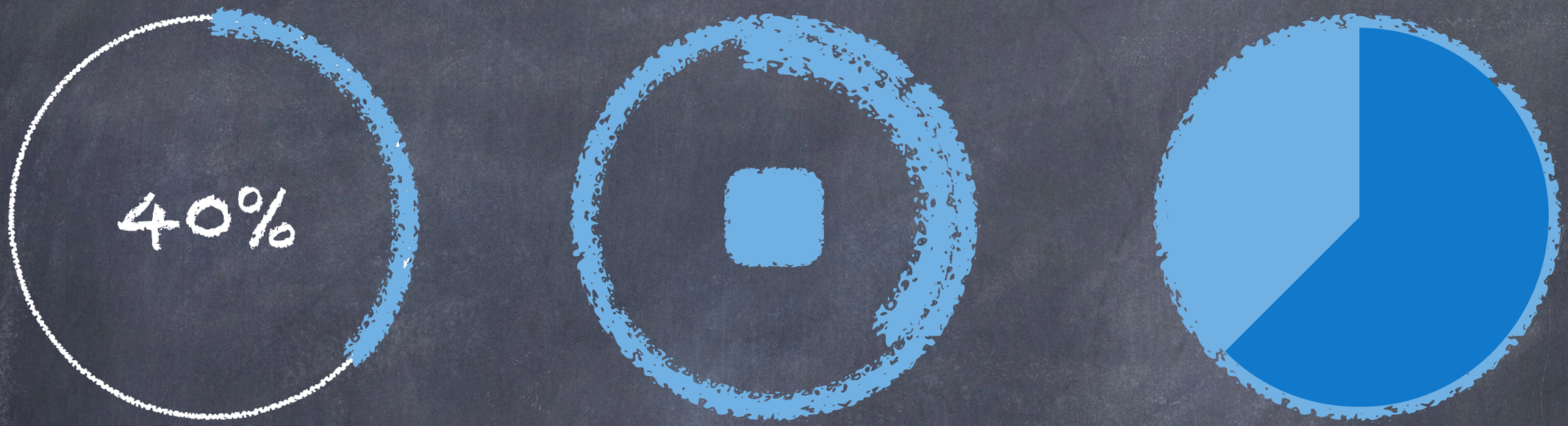
Some code is in the slides

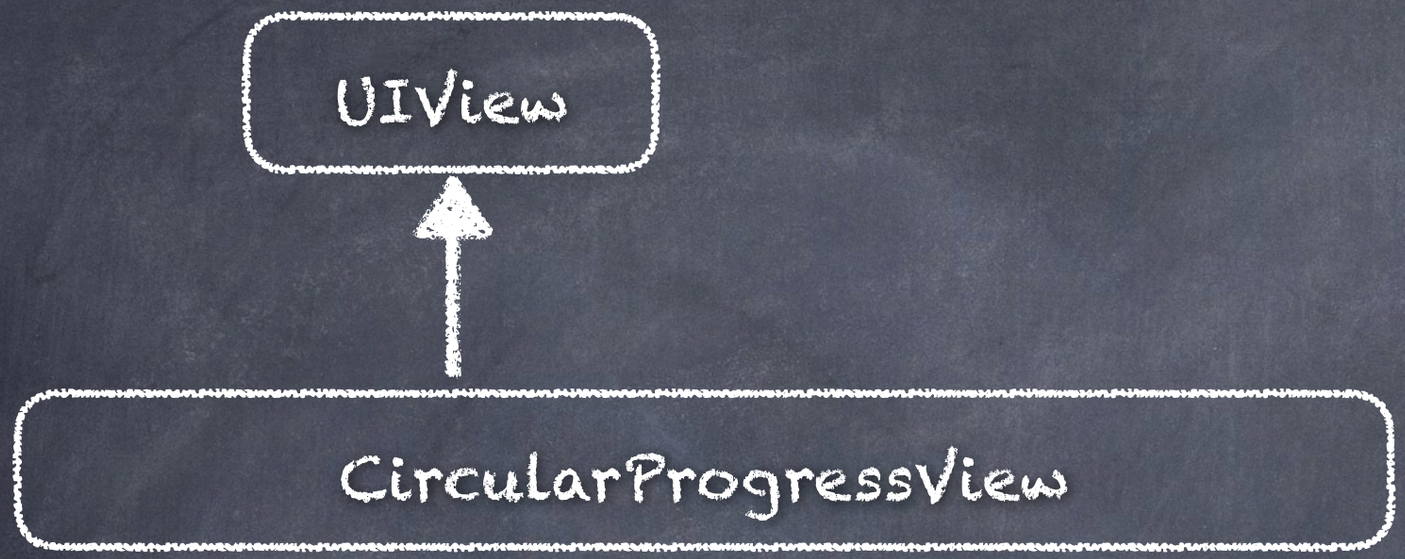
We'll see other code during the demo

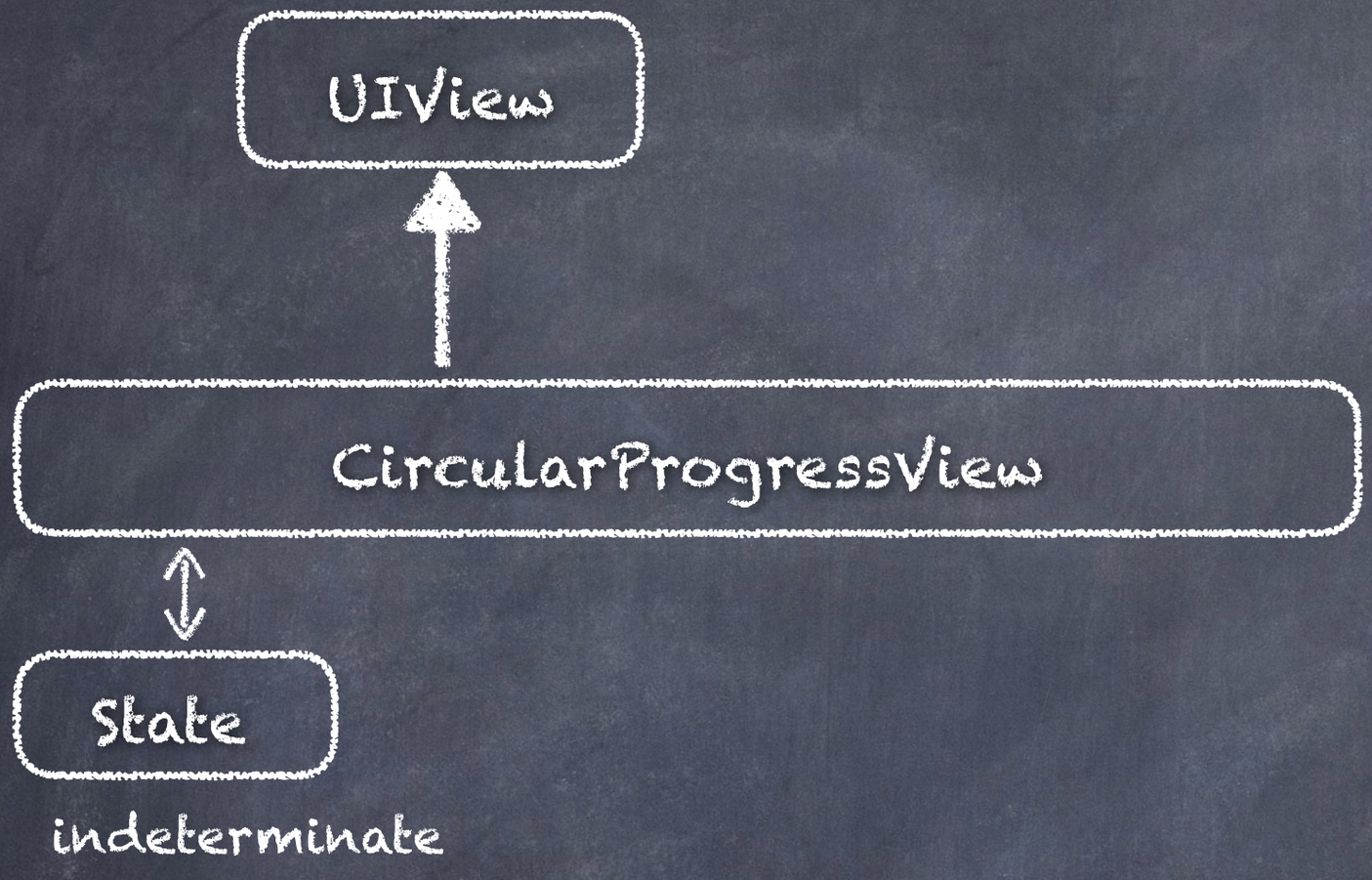
About Me

About You

what is your experience
with Core Animation?


40%


UIView
CircularProgressView



indeterminate

progress(FractionCompleted)

UIView

CircularProgressView

State

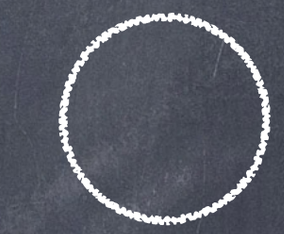
indeterminate
progress(FractionCompleted)

Layers

backgroundLayer: CAShapeLayer

progressLayer: ProgressLayer

additionalContentLayer: CALayer



56%

ProgressLayer

```swift
public final class ProgressLayer: CAShapeLayer {

    /// Developers call this method to change the progress of the layer.
    /// The progress is clamped to a value between 0.0 and 1.0.
    ///
    /// Upon changing, the `angle` property is updated to reflect the current
    /// angle (in radians).
    public var progress: FractionCompleted = 0.0 {
        didSet {
            angle = CGFloat(progress.rawValue * (.pi * 2))
        }
    }

    /// - Note: The `NSManaged` is a Core Data annotation that tells the compiler
    /// the implementation is provided at runtime. In this case, Core Animation
    /// provides the implementation. Without this we can't animate the `angle`
    /// property.
    ///
    /// - SeeAlso: `ProgressLayerDelegate`
    @NSManaged internal private(set) var angle: CGFloat
}
```

ProgressLayer

```swift
public final class ProgressLayer: CAShapeLayer {

    /// Developers call this method to change the progress of the layer.
    /// The progress is clamped to a value between 0.0 and 1.0.
    ///
    /// Upon changing, the `angle` property is updated to reflect the current
    /// angle (in radians).
    public var progress: FractionCompleted = 0.0 {
        didSet {
            angle = CGFloat(progress.rawValue * (.pi * 2))
        }
    }

    /// - Note: The `NSManaged` is a Core Data annotation that tells the compiler
    /// the implementation is provided at runtime. In this case, Core Animation
    /// provides the implementation. Without this we can't animate the `angle`
    /// property.
    ///
    /// - SeeAlso: `ProgressLayerDelegate`
    @NSManaged internal private(set) var angle: CGFloat
}
```

ProgressLayer

```swift
public final class ProgressLayer: CAShapeLayer {

    /// Developers call this method to change the progress of the layer.
    /// The progress is clamped to a value between 0.0 and 1.0.
    ///
    /// Upon changing, the `angle` property is updated to reflect the current
    /// angle (in radians).
    public var progress: FractionCompleted = 0.0 {
        didSet {
            angle = CGFloat(progress.rawValue * (.pi * 2))
        }
    }

    /// - Note: The `NSManaged` is a Core Data annotation that tells the compiler
    /// the implementation is provided at runtime. In this case, Core Animation
    /// provides the implementation. Without this we can't animate the `angle`
    /// property.
    ///
    /// - SeeAlso: `ProgressLayerDelegate`
    @NSManaged internal private(set) var angle: CGFloat
}
```

ProgressLayerDelegate

```
final class ProgressLayerDelegate: NSObject, CALayerDelegate {

    private weak var delegate: CAAnimationDelegate?

    init(delegate: CAAnimationDelegate) {
        self.delegate = delegate
    }
}

extension ProgressLayerDelegate {

    func action(for layer: CALayer, forKey event: String) -> CAAction? {
        if event == #keyPath(ProgressLayer.angle) {
            let animation = CABasicAnimation(keyPath: event)

            animation.duration = 0.4

            animation.fromValue = layer.presentation()?.value(forKey: event)
            animation.delegate = delegate

            return animation
        } else {
            return nil
        }
    }
}
```

ProgressLayerDelegate

```
final class ProgressLayerDelegate: NSObject, CALayerDelegate {

    private weak var delegate: CAAnimationDelegate?

    init(delegate: CAAnimationDelegate) {
        self.delegate = delegate
    }
}

extension ProgressLayerDelegate {

    func action(for layer: CALayer, forKey event: String) -> CAAction? {
        if event == #keyPath(ProgressLayer.angle) {
            let animation = CABasicAnimation(keyPath: event)

            animation.duration = 0.4

            animation.fromValue = layer.presentation()?.value(forKey: event)
            animation.delegate = delegate

            return animation
        } else {
            return nil
        }
    }
}
```

ProgressLayerDelegate

```
final class ProgressLayerDelegate: NSObject, CALayerDelegate {

    private weak var delegate: CAAnimationDelegate?

    init(delegate: CAAnimationDelegate) {
        self.delegate = delegate
    }
}

extension ProgressLayerDelegate {

    func action(for layer: CALayer, forKey event: String) -> CAAction? {
        if event == #keyPath(ProgressLayer.angle) {
            let animation = CABasicAnimation(keyPath: event)

            animation.duration = 0.4

            animation.fromValue = layer.presentation()?.value(forKey: event)
            animation.delegate = delegate

            return animation
        } else {
            return nil
        }
    }
}
```

ProgressLayerDelegate

```
final class ProgressLayerDelegate: NSObject, CALayerDelegate {

    private weak var delegate: CAAnimationDelegate?

    init(delegate: CAAnimationDelegate) {
        self.delegate = delegate
    }
}

extension ProgressLayerDelegate {

    func action(for layer: CALayer, forKey event: String) -> CAAction? {
        if event == #keyPath(ProgressLayer.angle) {
            let animation = CABasicAnimation(keyPath: event)

            animation.duration = 0.4

            animation.fromValue = layer.presentation()?.value(forKey: event)
            animation.delegate = delegate

            return animation
        } else {
            return nil
        }
    }
}
```

CADisplayLink

```
extension CircularProgressView {

    /// A private implementation state used to help track the state of the
    /// animating `ProgressLayer`.
    fileprivate enum AnimationState {
        case idle
        case animating(displayLink: CADisplayLink, animations: Set<CAAnimation>)
    }
}
```

Begin Animating Progress Change

```
extension CircularProgressView: CAAnimationDelegate {

    //
    // The animation delegate is notified when the `ProgressLayer`'s `angle` changes.
    //

    public func animationDidStart(_ animation: CAAnimation) {
        guard case .progress(_) = state else {
            return
        }

        switch internalState {
        case .idle:
            let displayLink = CADisplayLink(target: self, selector: #selector(updateTimerFired(_:)))
            displayLink.add(to: .main, forMode: .common)
            internalState = .animating(displayLink: displayLink, animations: Set(arrayLiteral: animation))
        case .animating(let displayLink, let animations):
            var newAnimations = animations
            newAnimations.insert(animation)
            internalState = .animating(displayLink: displayLink, animations: newAnimations)
        }
    }
```

Begin Animating Progress Change

```
extension CircularProgressView: CAAnimationDelegate {

    //
    // The animation delegate is notified when the `ProgressLayer`'s `angle` changes.
    //

    public func animationDidStart(_ animation: CAAnimation) {
        guard case .progress(_) = state else {
            return
        }

        switch internalState {
        case .idle:
            let displayLink = CADisplayLink(target: self, selector: #selector(updateTimerFired(_:)))
            displayLink.add(to: .main, forMode: .common)
            internalState = .animating(displayLink: displayLink, animations: Set(arrayLiteral: animation))
        case .animating(let displayLink, let animations):
            var newAnimations = animations
            newAnimations.insert(animation)
            internalState = .animating(displayLink: displayLink, animations: newAnimations)
        }
    }
```

Begin Animating Progress Change

```
extension CircularProgressView: CAAnimationDelegate {

    //
    // The animation delegate is notified when the `ProgressLayer`'s `angle` changes.
    //

    public func animationDidStart(_ animation: CAAnimation) {
        guard case .progress(_) = state else {
            return
        }

        switch internalState {
        case .idle:
            let displayLink = CADisplayLink(target: self, selector: #selector(updateTimerFired(_:)))
            displayLink.add(to: .main, forMode: .common)
            internalState = .animating(displayLink: displayLink, animations: Set(arrayLiteral: animation))
        case .animating(let displayLink, let animations):
            var newAnimations = animations
            newAnimations.insert(animation)
            internalState = .animating(displayLink: displayLink, animations: newAnimations)
        }
    }
```

CADisplayLink Callback

```
extension CircularProgressView {

    @objc
    fileprivate func updateTimerFired(_ displayLink: CADisplayLink) {
        guard let endAngle = progressLayer.presentation()?.angle else {
            return
        }
        update(progressLayer, endAngle: endAngle)
    }

    fileprivate func update(_ progressLayer: ProgressLayer, endAngle: CGFloat) {
        CATransaction.setDisableActions(true)
        progressLayer.path = strategy.progressPath(for: layerCenter, radius: radius, angle: endAngle)
        CATransaction.setDisableActions(false)
    }
}
```

CADisplayLink Callback

```
extension CircularProgressView {

    @objc
    fileprivate func updateTimerFired(_ displayLink: CADisplayLink) {
        guard let endAngle = progressLayer.presentation()?.angle else {
            return
        }
        update(progressLayer, endAngle: endAngle)
    }

    fileprivate func update(_ progressLayer: ProgressLayer, endAngle: CGFloat) {
        CATransaction.setDisableActions(true)
        progressLayer.path = strategy.progressPath(for: layerCenter, radius: radius, angle: endAngle)
        CATransaction.setDisableActions(false)
    }
}
```

CADisplayLink Callback

```
extension CircularProgressView {

    @objc
    fileprivate func updateTimerFired(_ displayLink: CADisplayLink) {
        guard let endAngle = progressLayer.presentation()?.angle else {
            return
        }
        update(progressLayer, endAngle: endAngle)
    }

    fileprivate func update(_ progressLayer: ProgressLayer, endAngle: CGFloat) {
        CATransaction.setDisableActions(true)
        progressLayer.path = strategy.progressPath(for: layerCenter, radius: radius, angle: endAngle)
        CATransaction.setDisableActions(false)
    }
}
```

Progress Animation Completes

```
public func animationDidStop(_ animation: CAAnimation, finished flag: Bool) {
    switch internalState {
    case .idle:
        // The internal state should not be idle when there are still animations.
        break
    case .animating(let displayLink, let animations):
        var updatedAnimations = animations
        updatedAnimations.remove(animation)

        if updatedAnimations.count == 0 {
            update(progressLayer, endAngle: progressLayer.angle)
            displayLink.invalidate()
            internalState = .idle
        } else {
            internalState = .animating(displayLink: displayLink, animations: updatedAnimations)
        }
    }
}
```

Progress Animation Completes

```
public func animationDidStop(_ animation: CAAnimation, finished flag: Bool) {
    switch internalState {
    case .idle:
        // The internal state should not be idle when there are still animations.
        break
    case .animating(let displayLink, let animations):
        var updatedAnimations = animations
        updatedAnimations.remove(animation)

        if updatedAnimations.count == 0 {
            update(progressLayer, endAngle: progressLayer.angle)
            displayLink.invalidate()
            internalState = .idle
        } else {
            internalState = .animating(displayLink: displayLink, animations: updatedAnimations)
        }
    }
  }
}
```

Progress Animation Completes

```
public func animationDidStop(_ animation: CAAnimation, finished flag: Bool) {
    switch internalState {
    case .idle:
        // The internal state should not be idle when there are still animations.
        break
    case .animating(let displayLink, let animations):
        var updatedAnimations = animations
        updatedAnimations.remove(animation)

        if updatedAnimations.count == 0 {
            update(progressLayer, endAngle: progressLayer.angle)
            displayLink.invalidate()
            internalState = .idle
        } else {
            internalState = .animating(displayLink: displayLink, animations: updatedAnimations)
        }
    }
}
```

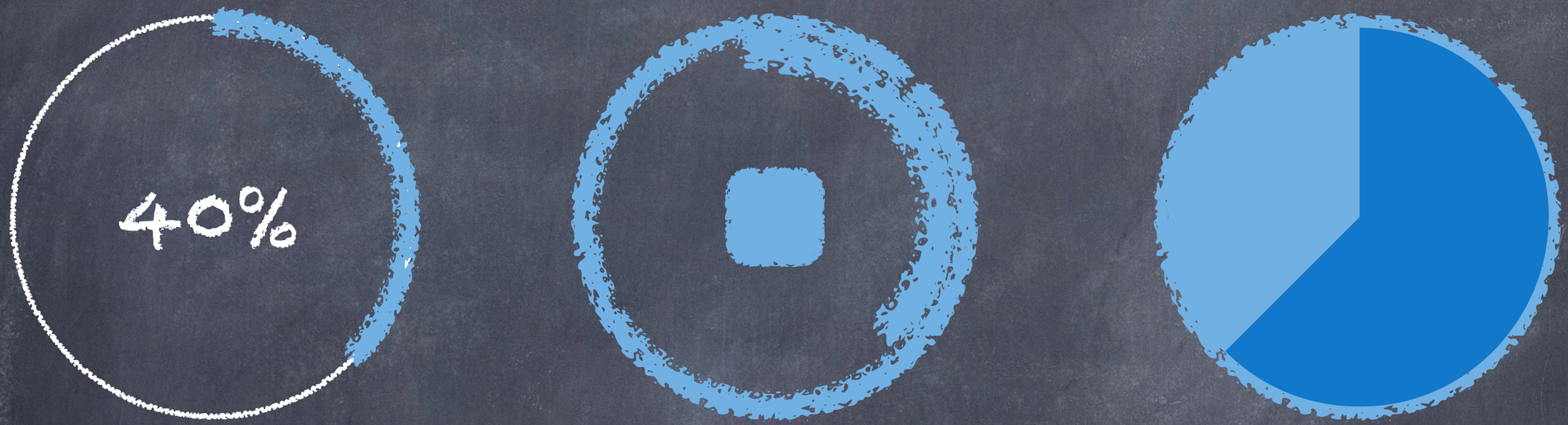
Progress Animation Completes

```
public func animationDidStop(_ animation: CAAnimation, finished flag: Bool) {
    switch internalState {
    case .idle:
        // The internal state should not be idle when there are still animations.
        break
    case .animating(let displayLink, let animations):
        var updatedAnimations = animations
        updatedAnimations.remove(animation)

        if updatedAnimations.count == 0 {
            update(progressLayer, endAngle: progressLayer.angle)
            displayLink.invalidate()
            internalState = .idle
        } else {
            internalState = .animating(displayLink: displayLink, animations: updatedAnimations)
        }
    }
  }
}
```

Progress Animation Completes

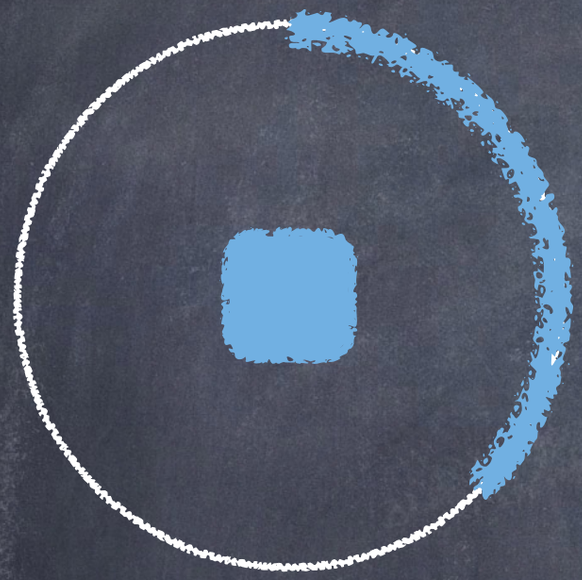
```
public func animationDidStop(_ animation: CAAnimation, finished flag: Bool) {
    switch internalState {
    case .idle:
        // The internal state should not be idle when there are still animations.
        break
    case .animating(let displayLink, let animations):
        var updatedAnimations = animations
        updatedAnimations.remove(animation)

        if updatedAnimations.count == 0 {
            update(progressLayer, endAngle: progressLayer.angle)
            displayLink.invalidate()
            internalState = .idle
        } else {
            internalState = .animating(displayLink: displayLink, animations: updatedAnimations)
        }
    }
}
```

Different Layout Strategies



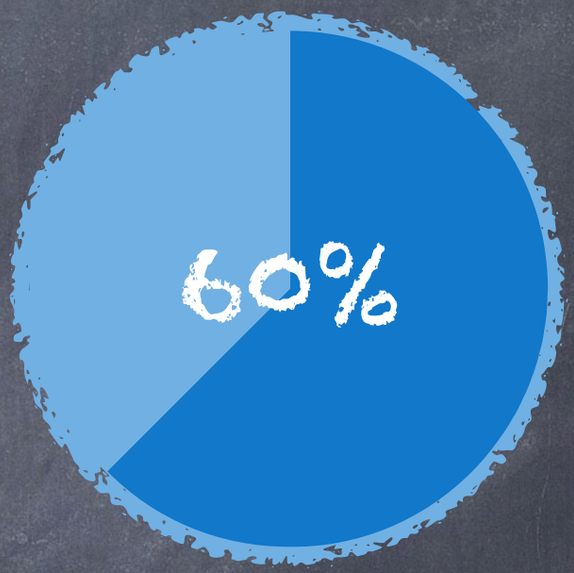
"Basic" "Inset" "Filled"

Different Layout Strategies



"Basic"

"Inset"



"Filled"

UIView

CircularProgressView

State

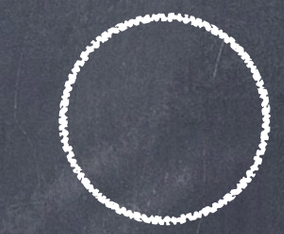
indeterminate
progress(FractionCompleted)

Layers

backgroundLayer: CAShapeLayer

progressLayer: ProgressLayer

additionalContentLayer: CALayer



56%


UIView
CircularProgressView
<protocol>
CircularProgressViewStrategy
BasicCircularProgressViewStrategy
InsetCircularProgressViewStrategy
FilledCircularProgressViewStrategy
Your Great Strategy

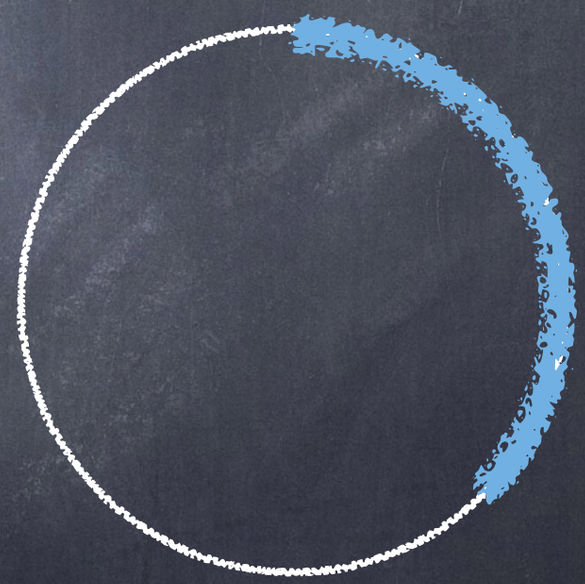
UIView

CircularProgressView

<protocol>

CircularProgressViewStrategy

- BasicCircularProgressViewStrategy
- InsetCircularProgressViewStrategy
- FilledCircularProgressViewStrategy
- Your Great Strategy



UIView

CircularProgressView

<protocol>

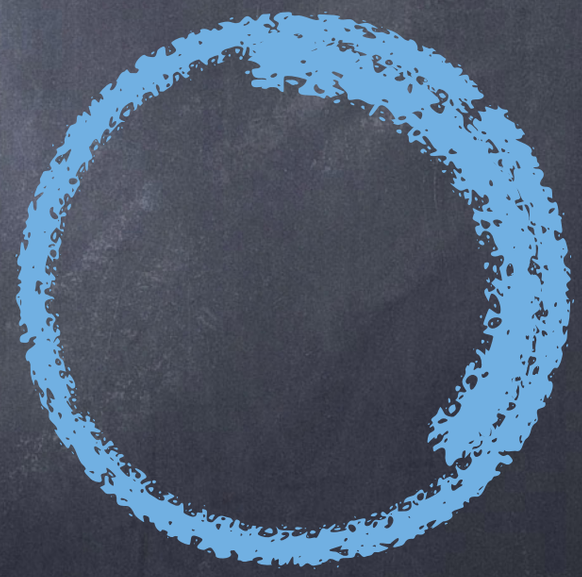
CircularProgressViewStrategy

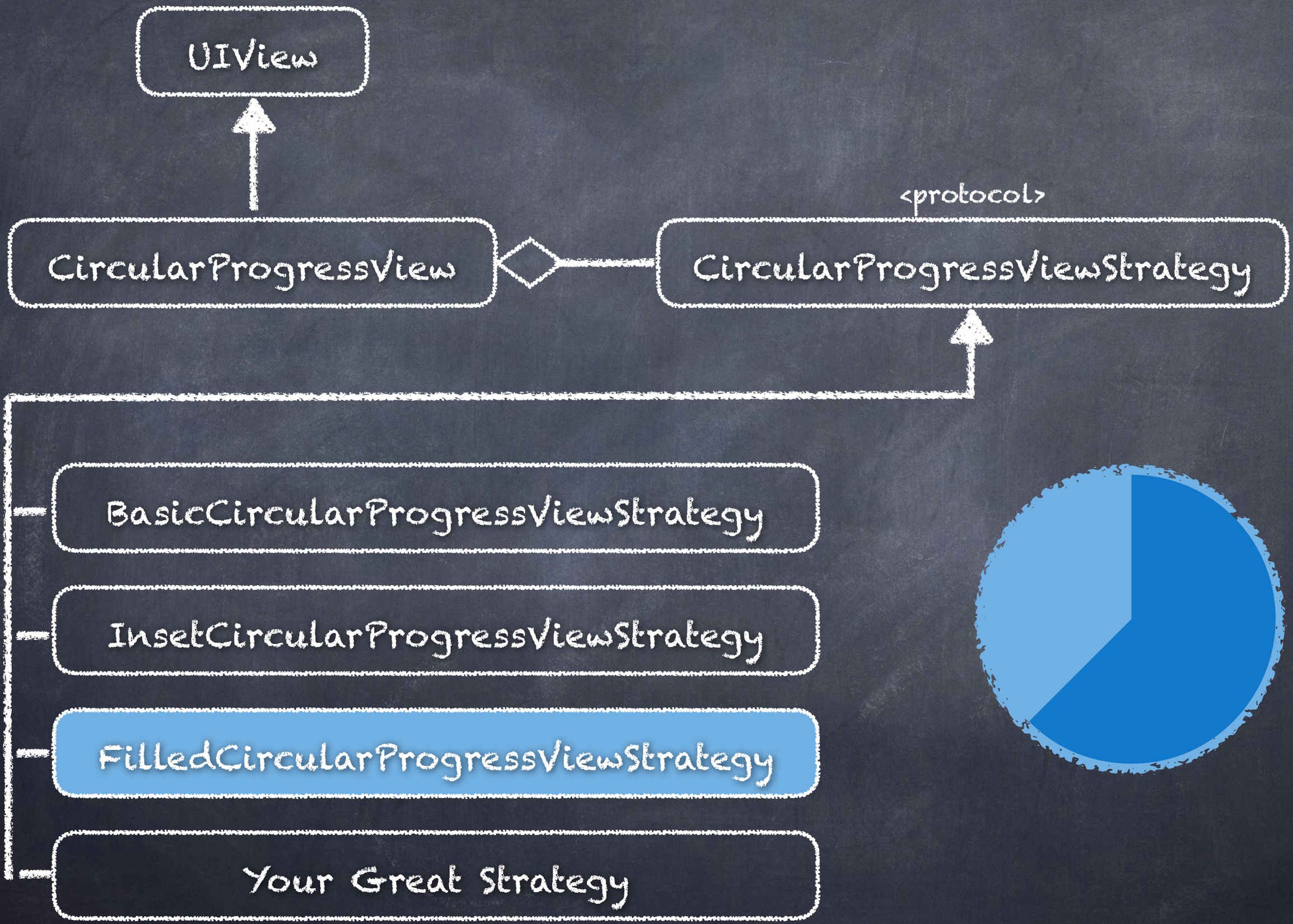
BasicCircularProgressViewStrategy

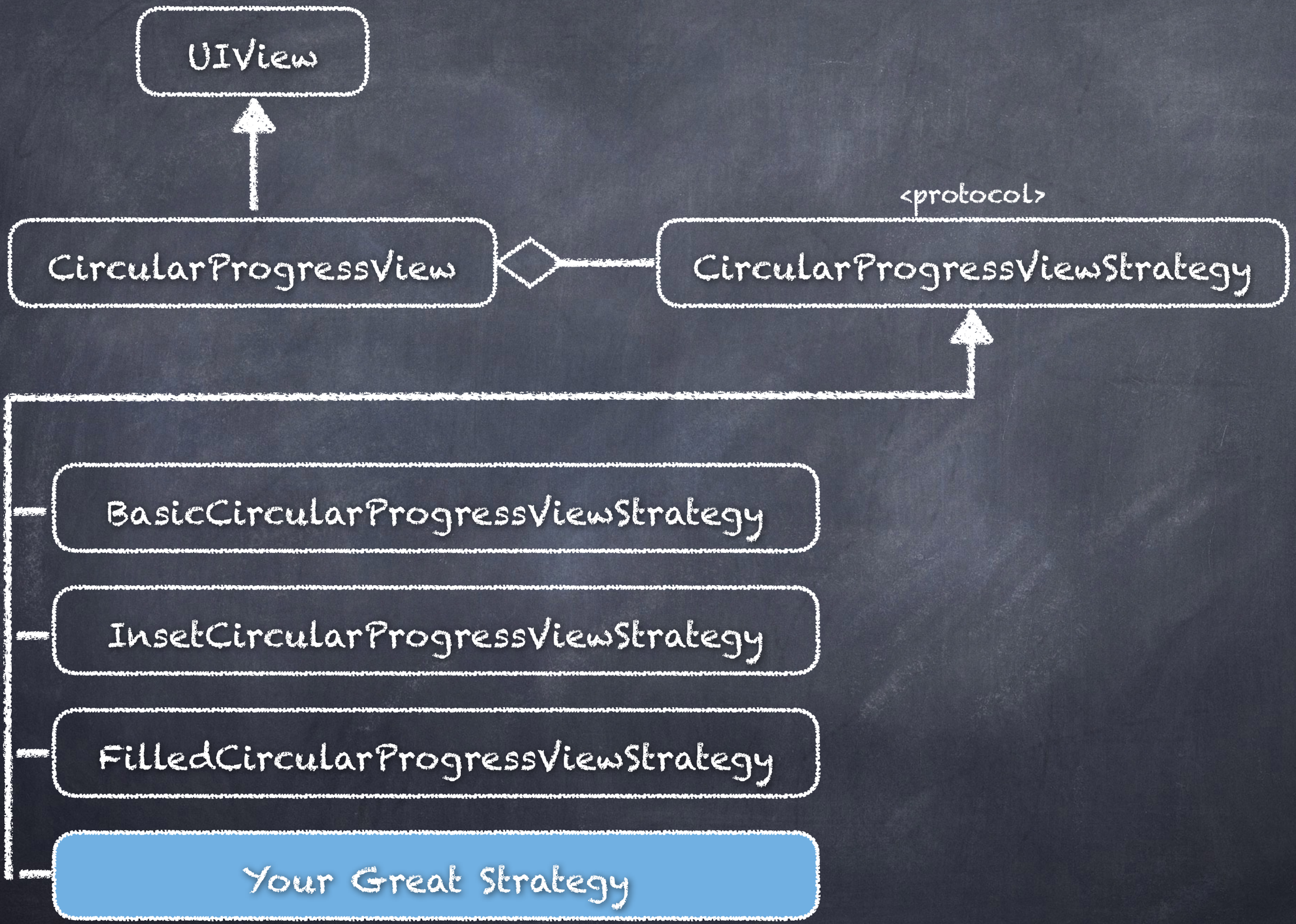
InsetCircularProgressViewStrategy

FilledCircularProgressViewStrategy

Your Great Strategy




UIView
CircularProgressView
<protocol>
CircularProgressViewStrategy
BasicCircularProgressViewStrategy
InsetCircularProgressViewStrategy
FilledCircularProgressViewStrategy
Your Great Strategy


UIView
CircularProgressView
<protocol>
CircularProgressViewStrategy
BasicCircularProgressViewStrategy
InsetCircularProgressViewStrategy
FilledCircularProgressViewStrategy
Your Great Strategy

InsetCircularProgressViewStrategy

FilledCircularProgressViewStrategy

Your Great Strategy

<protocol>

CircularProgressViewAdditionalContentLayoutStrategy

RoundedRectAdditionalContentLayoutStrategy

LocalizedProgressAdditionalContentLayoutStrategy

40%

Strategy

```
public protocol CircularProgressViewStrategy {

    func layoutLayers(
        _ layers: CircularProgressView.Layers,
        center: CGPoint,
        radius: CGFloat
    )

    func transitionLayers(
        _ layers: CircularProgressView.Layers,
        to state: CircularProgressView.State,
        tintColor: UIColor
     )


    func updateTintColor(
        _ tintColor: UIColor,
        on layers: CircularProgressView.Layers,
        state: CircularProgressView.State
    )

    func progressPath(
        for center: CGPoint,
        radius: CGFloat,
        angle: CGFloat
    ) -> CGPath
}
```

"Basic" Strategy

```swift
public final class BasicCircularProgressViewStrategy: CircularProgressViewStrategy {
}


extension BasicCircularProgressViewStrategy {

    public func layoutLayers(
        _ layers: CircularProgressView.Layers,
        center: CGPoint,
        radius: CGFloat
    ) {

        let path = UIBezierPath(arcCenter: center, radius: radius).cgPath

        layers.backgroundLayer.path = path
        layers.progressLayer.path = path
        layers.progressLayer.lineWidth = max(radius * 0.125, 4.0)
    }
}
```

"Basic" Strategy

```
extension BasicCircularProgressViewStrategy {

    public func transitionLayers(
        _ layers: CircularProgressView.Layers,
       to state: CircularProgressView.State,
       tintColor: UIColor
    ) {

        animateLayers(layers, to: state)
    }
}
```

See BasicCircularProgressViewStrategy.swift
for how different CAAnimations are added/ removed

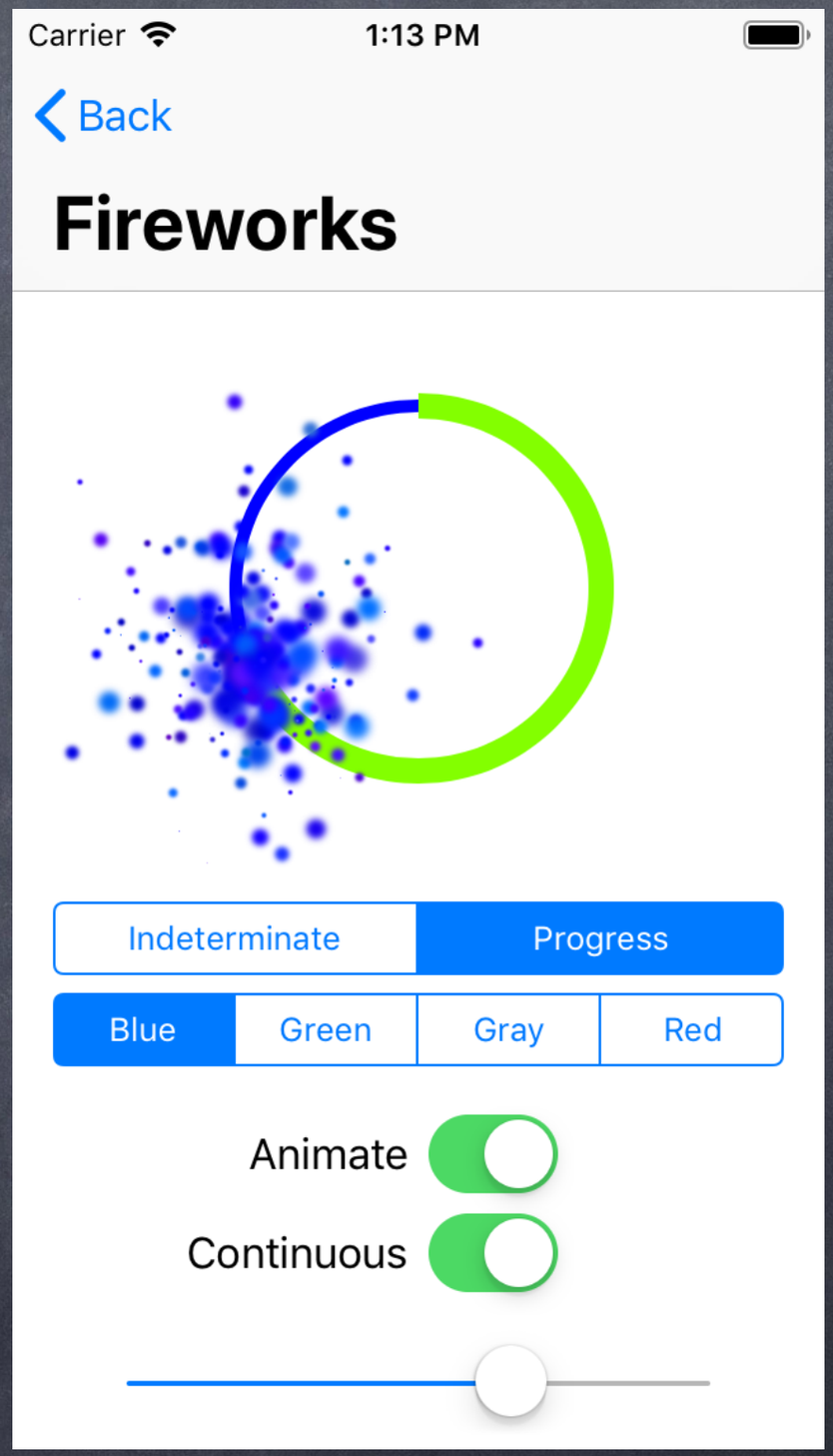
"Basic" Strategy

```
extension BasicCircularProgressViewStrategy {

    public func updateTintColor(
        _ tintColor: UIColor,
        on layers: CircularProgressView.Layers,
        state: CircularProgressView.State)
    {
        layers.backgroundLayer.fillColor = UIColor.clear.cgColor
        layers.backgroundLayer.strokeColor = UIColor.black.cgColor

        layers.progressLayer.fillColor = UIColor.clear.cgColor
        layers.progressLayer.strokeColor = tintColor.cgColor

        additionalContentLayoutStrategy.updateTintColor(tintColor, state: state)

        switch state {
        case .indeterminate:
            layers.progressLayer.opacity = 0.3
        case .progress(_):
            layers.progressLayer.opacity = 1.0
        }
    }
}
```

"Basic" Strategy

```
extension BasicCircularProgressViewStrategy {

    public func progressPath(
        for center: CGPoint,
        radius: CGFloat,
        angle: CGFloat
    ) -> CGPath {

        return UIBezierPath(
            arcCenter: center,
            radius: radius,
            startAngle: 0.0,
            endAngle: angle,
            clockwise: true
        ).cgPath
    }
}
```

Demo



https://github.com/briancoyner/Circular-Progress

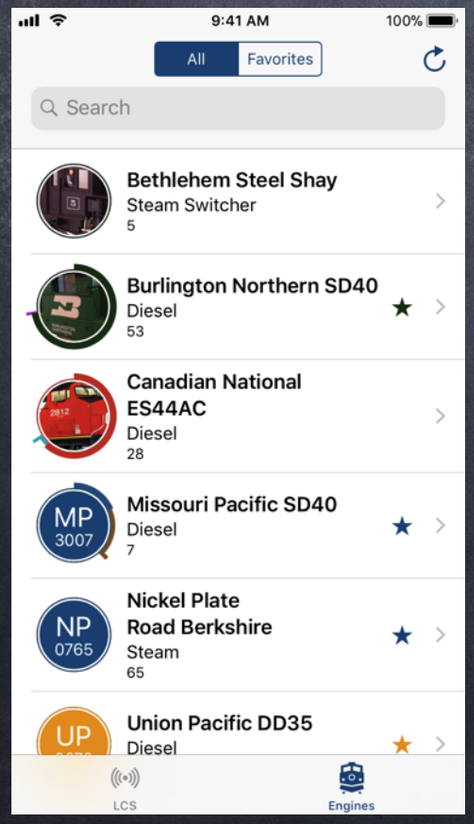
Other Uses

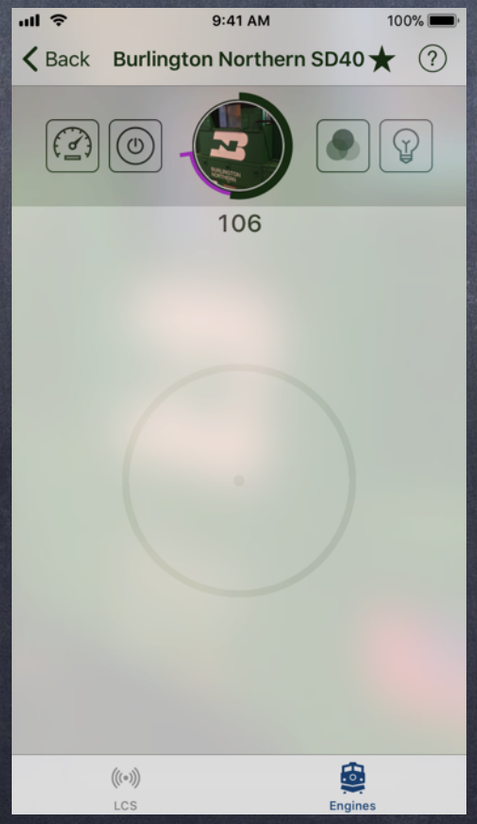
It's good to have side projects

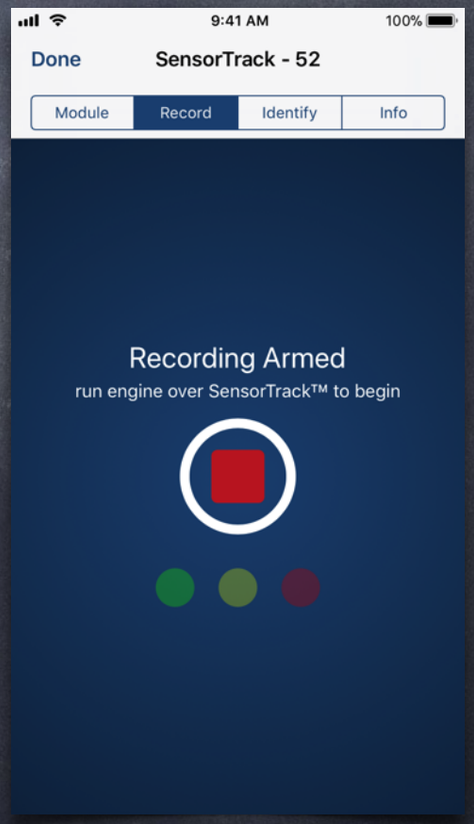
High Rail

highrailcompany.com

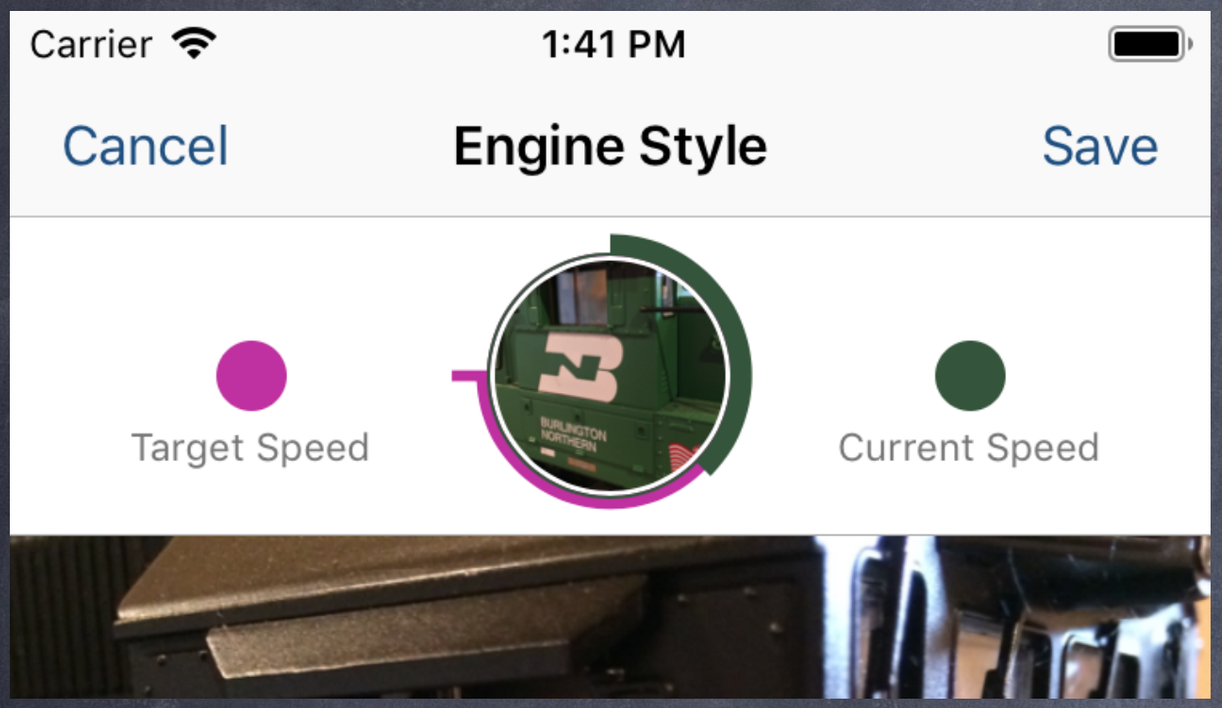
- my "playground" app
- originally developed in ObjC
- re-wrote from the ground up in Swift
- now Swift 4.1
- 10 unique gestures
- 3D Touch engine speed control







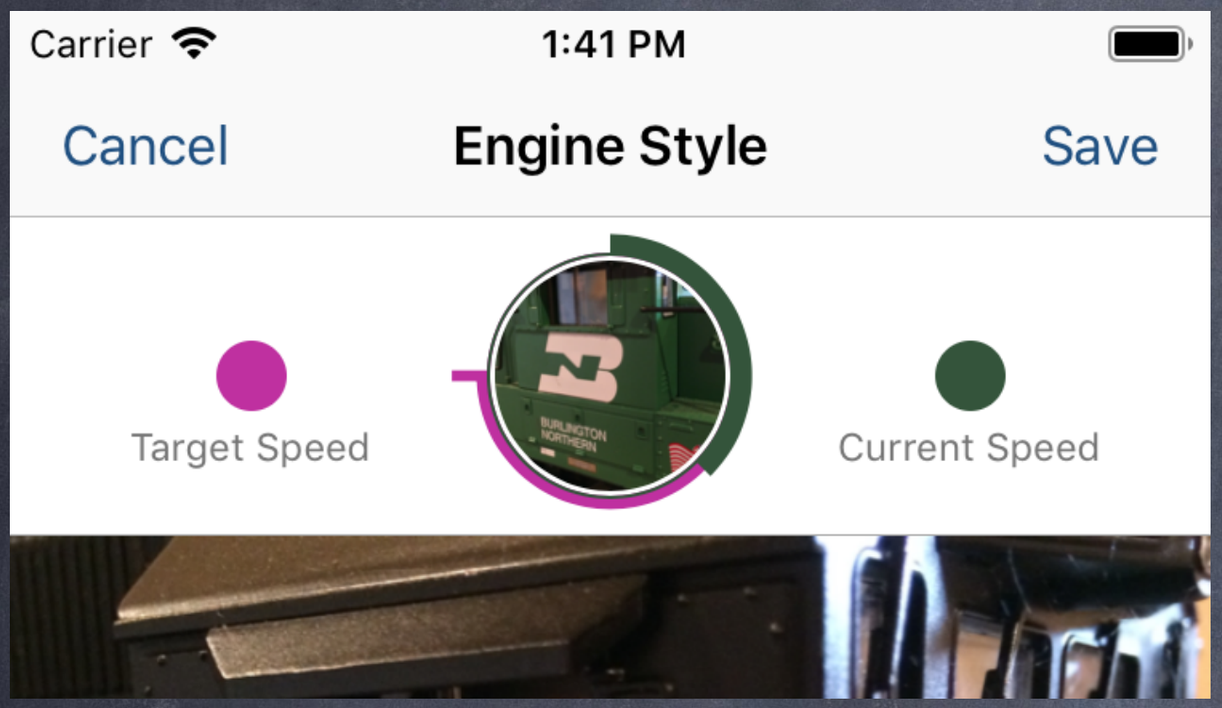
Speed Graphs



There are 200 speed steps

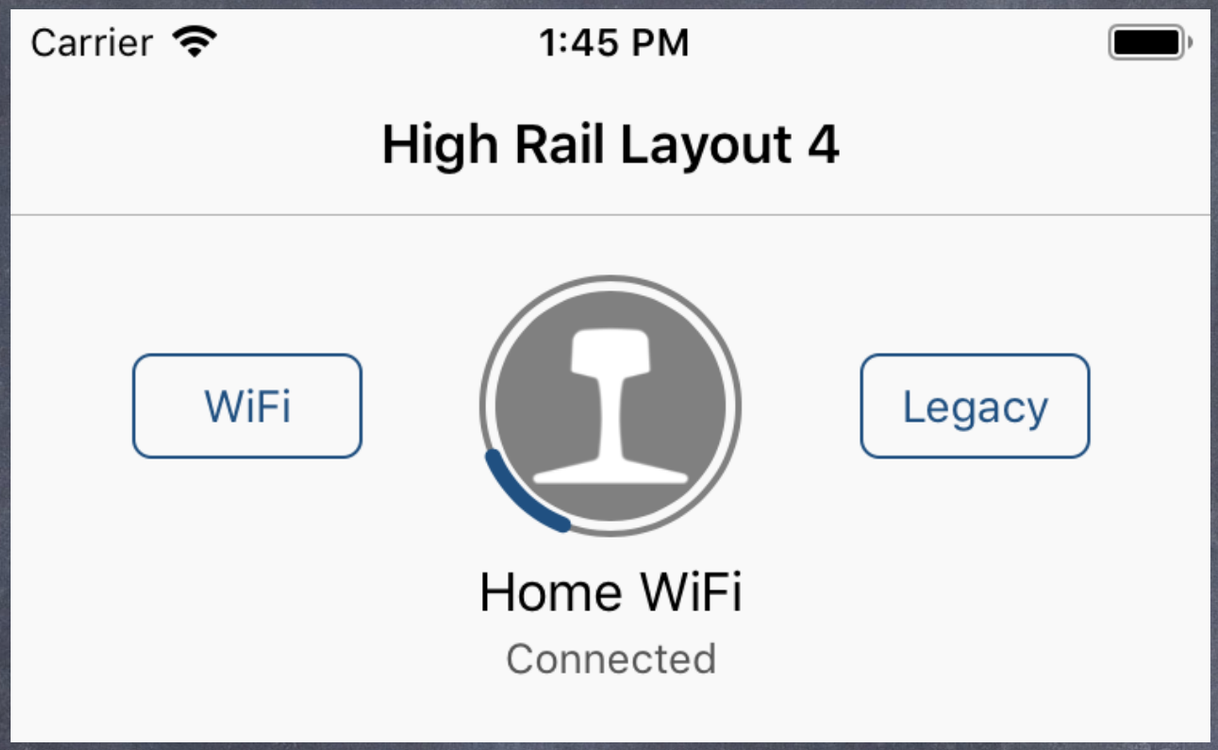
progress = current speed / 200

Speed Graphs



Two "Circular" progress views stack on top of each other

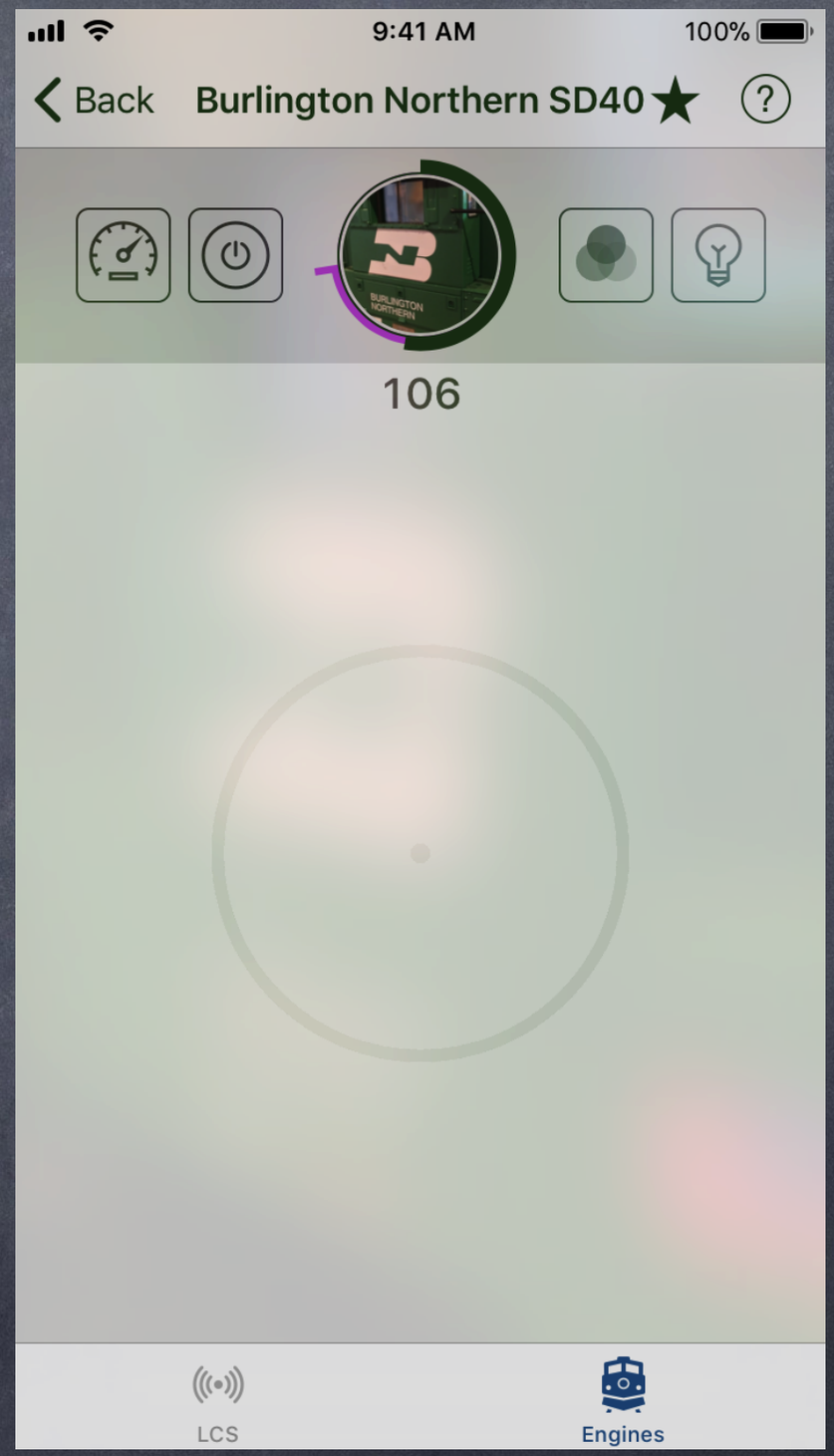
Connection Views



"Indeterminate" while connecting

Spin a "blue" arc

High Rail Demo



https://highrailcompany.com

Thanks