default the animation ⇒ 0.25 s

//BallLayer.swift

```
import UIKit

class BallLayer: CAShapeLayer {
    override func action(forKey event: String) -> CAAction? {
        return nil
    }

    override func awakeFromNib() {
        super.awakeFromNib()
        fillColor = UIColor.black.cgColor
        contentsScale = UIScreen.main.scale
        drawsAsynchronously = true
        needsDisplayOnBoundsChange = true
        setNeedsDisplay()
    }
}
```

//CustomControllView.swift

```
import UIKit

                    ↗Can also be UIView
class CustomControllView: UIControl {
    let backdropLayer: CAShapeLayer = CAShapeLayer()
    let temperatureLabel = UILabel()
```

```
let ringWidth: CGFloat = CGFloat(40.0)
var textFont: UIFont = UIFont.boldSystemFont(ofSize: 40)
let maxTemperature: Double = 30
var ballAngle: Int = 0 {
    didSet {
        var angle = ballAngle - 270
        if angle < 0 {
            angle += 360
        }
        let newTemp = Int(floor(Double(angle) / 360 * maxTemperature))
        temperatureLabel.text = "\(newTemp)"



    }
}

private var textFontSize: CGFloat {
    return textFont.pointSize
}

var halfRingWidth: CGFloat {
    return ringWidth / 2
}

private var centerPoint: CGPoint {
    return CGPoint(x: self.frame.size.width/2, y: self.frame.size.height/2)
}

private var radius: CGFloat {
    return bounds.height/2 - (ringWidth/2.0)
}

private lazy var ballIndicator: CAShapeLayer = {
    return BallLayer()
}()

override func awakeFromNib() {
    super.awakeFromNib()
    temperatureLabel.translatesAutoresizingMaskIntoConstraints = false
    addSubview(temperatureLabel())
    temperatureLabel.centerXAnchor.constraint(equalTo: centerXAnchor).isActive = true
    temperatureLabel.centerYAnchor.constraint(equalTo: centerYAnchor).isActive = true
    temperatureLabel.textAlignment = NSTextAlignment.center
    temperatureLabel.font = textFont
    temperatureLabel.textColor = UIColor.black
    temperatureLabel.text = "0°"
}
```

ctr + command + space ⇒ add symbols

```
override func didMoveToSuperView(){
    super.didMoveToSuperView()
    let circlePath: CGPath = UIBezierPath(ovalIn: CGRect(x: halfRingWidth, y: halfRingWidth,
        width: bounds.width-ringWidth, height: bounds.height-ringWidth)).cgPath
    backdropLayer.path = circlePath
    backdropLayer.lineWidth = ringWidth
    backdropLayer.strokeEnd = 1.0
    backdropLayer.fillColor = nil
    backdropLayer.strokeColor = UIColor(red: 112/255, green: 25/255, blue: 18/255, alpha: 1.0).cgColor
    layer.addSublayer(backdropLayer)
    let ballShape = UIBezierPath(ovalIn: CGRect(x:0, y:0, width: ringWidth, height: ringWidth))
    ballIndicator.path = ballShape.cgPath
    ballIndicator.frame = CGRect(x:0, y:0, width: ringWidth, height: ringWidth)
    ballIndicator.position = CGPoint(x: frame.width/2, y: ringWidth/2)
    layer.addSublayer(ballIndicator)

}

override func beginTracking(_ touch: UITouch, with event: UIEvent?) -> Boo{
    super.beginTracking(touch, with: event)
    let touchPosition = touch.location(in: self)
    let ballFrame = ballIndicator.frame
    if ballFrame.contains(touchPosition){
        return true
    }
    return false
}

override func continueTracking(_ touch: UITouch, with event: UIEvent?) -> Boo{
    super.continueTracking(touch, with: event)
    let lastPoint = touch.location(in: self)
    moveBall(to: lastPoint)
    self.sendActions(for: UIControl.Event.valueChanged)
    return true
}

func moveBall(to point: CGPoint){
    let currentAngle: Double = angleFromNorth(p1: centerPoint, p2: point)
    let angle = Int(floor(currentAngle))
    ballAngle = angle
    let actualAngle = Int(360 - angle)
    let handleCenter = pointFrom(angle: actualAngle)
    ballIndicator.position = handleCenter
}
```

```
override func endTracking(_ touch: UITouch, with event: UIEvent?) -> Bool {
    super.endTracking(touch, with: event)

    layoutSubviews()
}

func pointFrom(angle: Int) -> CGPoint {
    var result: CGPoint = CGPoint.zero
    let y = round(Double(radius) * sin(Double(-angle).degreesToRadians())) + Double(centerPoint.y)
    let x = round(Double(radius) * cos(Double(-angle).degreesToRadians())) + Double(centerPoint.x)
    result.y = CGFloat(y)
    result.x = CGFloat(x)
    return result
}

func angleFromNorth(p1: CGPoint, p2: CGPoint) -> Double {
    var v: CGPoint = CGPoint(x: p2.x - p1.x, y: p2.y - p1.y)
    let vmag: CGFloat = sqrt(v.x * v.x) + (v.y * v.y))
    var result: Double = 0.0
    v.x = v.x / vmag        // normalize the result
    v.y /= vmag
    let radians = Double(atan2(v.y, v.x))
    result = radians.radiansToDegrees()
    let finalResult = result >= 0 ? result : result + 360.0
    return finalResult
}
}

private extension Double {
    func degreesToRadius() -> Double {
        return self * Double.pi / 180.0
    }

    func radiansToDegrees() -> Double {
        return self * 180.0 / Double.pi
    }
}
```

// Speed Gauge

```
import UIKit

class ViewController: UIViewController{
    @IBOutlet weak var gaugeView: OPLGaugeView!

    override func viewDidLoad(){
        super.viewDidLoad()

    }

    @IBAction func sliderDidChange(_ sender: UISlider){
        let sliderValue = sender.value
        let speed = gaugeView.maxSpeed * CGFloat(sliderValue)
        gaugeView.rotateGauge(newSpeed: speed)
    }
```

// OPLGageLayer.swift

```
import UIKit

open class OPLGaugeLayer: CAShapeLayer{

        internal var disableSpringAnimation: Bool = true

        open override func action(forKey event: String) -> CAAction?{
            if event == "transform" && !disableSpringAnimation{
                let springAnimation = CASpringAnimation(keyPath: event)
                springAnimation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingTimingFunctionEaseOut)
                springAnimation.initialVelocity = 0.8
                springAnimation.damping = 1
                return springAnimation
            }
            return super.action(forKey: event)
        }
```

```swift
// OPLGaugeView.swift

import UIKit
@IBDesignable
open class OPLGaugeView: UIView {

    private lazy var gauge: OPLGaugeLayer = {
        let _gauge = OPLGaugeLayer()
        _gauge.drawAsynchronously = true
        _gauge.disableSpringAnimation = true
        return _gauge
    }()

    @IBInspectable open var gaugeColor: UIColor = UIColor.orange {
        didSet {
            gauge.fillColor = gaugeColor.cgColor
            setNeedsDisplay()                    // only call when drawing need to redraw
        }
    }

    @IBInspectable open var markers: Int = 5 {
        didSet {
            setNeedsDisplay()
        }
    }

    @IBInspectable open var minSpeed: CGFloat = 0 {
        didSet {
            setNeedsDisplay()
        }
    }


    @IBInspectable open var maxSpeed: CGFloat = 25 {
        didSet {
            setNeedsDisplay()
        }
    }

    @IBInspectable open var textFont: UIFont = UIFont.boldSystemFont(ofSize: 24) {
        didSet {
            setNeedsDisplay()
        }
    }

    @IBInspectable open var textColor: UIColor = UIColor.black {
        didSet {
            setNeedsDisplay()
        }
    }
```

```
private let lineWidth: CGFloat = 6

open override func draw(_ rect: CGRect) {
    let center = CGPoint(x: bounds.width/2, y: bounds.height/2)
    let radius = min(bounds.width, bounds.height)/2
    let startAngle: CGFloat = 0.75 * CGFloat(Double.pi)
    let endAngle: CGFloat = 0.25 * CGFloat(Double.pi)
    let outlinePath = UIBezierPath(arcCenter: center, radius: radius - CGFloat(lineWidth/2), startAngle: startAngle,
        endAngle: endAngle, clockwise: true)
    outlinePath.lineWidth = CGFloat(lineWidth)
    gaugeColor.setStroke()
    outlinePath.stroke()

    let context = UIGraphicsGetCurrentContext()!
    context.saveGState()
    gaugeColor.setFill()
    let angleDifference: CGFloat = 2*.pi - startAngle + endAngle
    let arcLengthPerMark: CGFloat = angleDifference / CGFloat(markers)
    let multiplePerMark: CGFloat = (maxSpeed - minSpeed) / CGFloat(markers)
    let markerWidth: CGFloat = lineWidth-1
    let markerSize: CGFloat = markerWidth * 2
    let markerPath = UIBezierPath(rect: CGRect(x: -markerWidth/2, y: 0, width: markerWidth,
        height: markerSize).Integral)
    context.translateBy(x: rect.width/2, y: rect.height/2)

    for i in 0...markers {
        context.saveGState()
        let angle = arcLengthPerMark * CGFloat(i) + startAngle - .pi/2
        context.rotate(by: angle)
        context.translateBy(x: 0, y: rect.height/2 - markerSize)
        markerPath.fill()
        context.rotate(by: CGFloat(Double.pi))
        let label = round(value: multiplePerMark * CGFloat(i) + minSpeed)
        let labelWidth = label.width(font: textFont)
        let textPos = CGPoint(x: -labelWidth/2, y: 0)
        label.draw(at: textPos, withAttributes: [NSAttributedStringKey.font: textFont,
            NSAttributedStringKey.foregroundColor: textColor])

        context.restoreGState()
    }
    context.restoreGState()
}
```

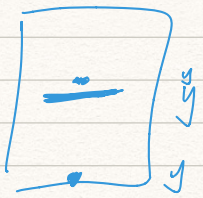
0.75π

2π -

```
open override func didMoveToSuperview(){
    super.didMoveToSuperview()
    gauge.fillColor = gaugeColor.cgColor
    layer.addSublayer(gauge)
    let anchorPoint = CGPoint(x: 0.5, y: 1.0)
    let newPoint = CGPoint(x: gauge.bounds.size.width * anchorPoint.x, y: gauge.bounds.size.height * anchorPoint.y)
    let oldPoint = CGPoint(x: gauge.bounds.size.width * gauge.anchorPoint.x, y: gauge.bounds.size.height * gauge.anchorPoint.y)
    var position = gauge.position
    position.x -= oldPoint.x      ⇒ position.x = position.x − oldPoint.x
    position.x += newPoint.x
    position.y -= oldPoint.x
    position.y += newPoint.y
    gauge.position = position
    gauge.anchorPoint = anchorPoint
}
```

CALayers ⇒ anchor points in the center

views ⇒ anchor points in the top left




```
open override func layoutSubviews(){
    super.layoutSubviews()
    let viewWidth = frame.width
    let halfViewWidth = viewWidth/2
    let viewHeight = frame.height
    let gaugeYPos: CGFloat = viewHeight * 0.05
    let gaugeHeight: CGFloat = viewHeight * 0.45
    let gaugeWidth: CGFloat = gaugeHeight * 0.16
    let gaugeFrame = CGRect(x: halfViewWidth - (gaugeWidth/2), y: gaugeYPos, width: gaugeWidth,
        height: gaugeHeight).integral
    gauge.bounds.size = gaugeFrame.size
    gauge.position.x = gaugeFrame.origin.x + (gaugeFrame.width/2)
    gauge.position.y = gaugeFrame.origin.y + gaugeFrame.height

    let gaugePath = UIBezierPath()
    gaugePath.move(to: CGPoint(x: gaugeWidth/2, y: 0))   ← relative to the bounds of the gauge
    gaugePath.addLine(to: CGPoint(x: gaugeWidth, y: gaugeHeight))
    gaugePath.addLine(to: CGPoint(x: 0, y: gaugeHeight))
    gaugePath.close()
    pague.path = gaugePath.cgPath
}

func rotateGauge(newSpeed: CGFloat){
    var speed = newSpeed
    if speed > maxSpeed
        speed = maxSpeed
    }
    if speed < minSpeed{
        speed = minSpeed
    }
    let fractalSpeed = (speed - minSpeed)/(maxSpeed - minSpeed)
    let newAngle = 0.75 * CGFloat(Double.pi) * (2 * fractalSpeed - 1)
    gauge.transform = CATransform3DMakeRotation(newAngle, 0, 0, 1)
```

```
    }

    private func round(value: CGFloat) -> String {
        let divisor = pow(10, Double(1))
        let roundedNumber = (Double(value) * divisor).rounded() / divisor
        if roundedNumber.isZero {
            return "0"
        }
        let intRoundedNumber = Int(roundedNumber)
        if Double(intRoundedNumber) == roundedNumber {
            return "\(intRoundedNumber)"
        }
        return "\(roundedNumber)"
    }
}

fileprivate extension String {
    func width(font: UIFont) -> CGFloat {
        let constraintRect = CGSize(width: .greatestFiniteMagnitude, height: CGFloat(220))
        let boundingBox = self.boundingRect(with: constraintRect,
            options: .usesLineFragmentOrigin, attributes: [.font: font], context: nil)
        return ceil(boundingBox.width)
    }
}
```