Lecture 5 Demo Code: FaceIt MVC and Gestures

Objective

Included below is the source code for the demo in lecture. It is provided under the same Creative Commons licensing as the rest of CS193p's course materials. Code that has not changed since the previous lecture is included, but grayed out. And here is the complete project.

```
//
//  ViewController.swift
//  FaceIt
//
//  Created by CS193p Instructor.
//  Copyright © 2017 Stanford University. All rights reserved.
//

import UIKit

class ViewController: UIViewController
{
   var expression = FacialExpression(eyes: .closed, mouth: .frown) {
        didSet {
            updateUI()
        }
    }

    private func updateUI()
    {
        switch expression.eyes {
        case .open:
            faceView?.eyesOpen = true
        case .closed:
            faceView?.eyesOpen = false
        case .squinting:
            faceView?.eyesOpen = false
        }
        faceView?.mouthCurvature = mouthCurvatures[expression.mouth] ?? 0.0
    }

    private let mouthCurvatures =
        [FacialExpression.Mouth.grin:0.5,.frown:-1.0,.smile:1.0,.neutral:0.0,.smirk:-0.5]
    @IBOutlet weak var faceView: FaceView! {
        didSet {
            let handler = #selector(FaceView.changeScale(byReactingTo:))
            let pinchRecognizer = UIPinchGestureRecognizer(target: faceView, action: handler)
            faceView.addGestureRecognizer(pinchRecognizer)
            let tapRecognizer = UITapGestureRecognizer(target: self, action: #selector(toggleEyes(byReactingTo:)))
            tapRecognizer.numberOfTapsRequired = 1
            faceView.addGestureRecognizer(tapRecognizer)
            let swipeUpRecognizer = UISwipeGestureRecognizer(target: self, action: #selector(increaseHappiness))
            swipeUpRecognizer.direction = .up
            faceView.addGestureRecognizer(swipeUpRecognizer)
            let swipeDownRecognizer = UISwipeGestureRecognizer(target: self, action: #selector(decreaseHappiness))
            swipeDownRecognizer.direction = .down
            faceView.addGestureRecognizer(swipeDownRecognizer)
            updateUI()
        }
    }

    func increaseHappiness()
    {
        expression = expression.happier
    }
    func decreaseHappiness()
    {
        expression = expression.sadder
    }

    func toggleEyes(byReactingTo tapRecognizer: UITapGestureRecognizer) {
        if tapRecognizer.state == .ended {
            let eyes: FacialExpression.Eyes = (expression.eyes == .closed) ? .open : .closed
            expression = FacialExpression(eyes: eyes, mouth: expression.mouth)
        }
    }
}
```

```swift
//
//  FaceView.swift
//  FaceIt
//
//  Created by CS193p Instructor.
//  Copyright © 2017 Stanford University. All rights reserved.
//

import UIKit

@IBDesignable
class FaceView: UIView
{
    // Public API

    // 1.0 is full smile and -1.0 is full frown
    @IBInspectable
    var mouthCurvature: Double = 0.5 { didSet { setNeedsDisplay() } }

    @IBInspectable
    var eyesOpen: Bool = true { didSet { setNeedsDisplay() } }

    @IBInspectable
    var scale: CGFloat = 0.9 { didSet { setNeedsDisplay() } }

    @IBInspectable
    var lineWidth: CGFloat = 5.0 { didSet { setNeedsDisplay() } }

    @IBInspectable
    var color: UIColor = UIColor.blue { didSet { setNeedsDisplay() } }

    func changeScale(byReactingTo pinchRecognizer: UIPinchGestureRecognizer)
    {
        switch pinchRecognizer.state {
        case .changed,.ended:
            scale *= pinchRecognizer.scale
            pinchRecognizer.scale = 1
        default:
            break
        }
    }

    // Private Implementation

    private struct Ratios {
        static let skullRadiusToEyeOffset: CGFloat = 3
        static let skullRadiusToEyeRadius: CGFloat = 10
        static let skullRadiusToMouthWidth: CGFloat = 1
        static let skullRadiusToMouthHeight: CGFloat = 3
        static let skullRadiusToMouthOffset: CGFloat = 3
    }

    private var skullRadius: CGFloat {
        return min(bounds.size.width, bounds.size.height) / 2 * scale
    }

    private var skullCenter: CGPoint {
        return CGPoint(x: bounds.midX, y: bounds.midY)
    }
    private enum Eye {
        case left
        case right
    }

    private func pathForEye(_ eye: Eye) -> UIBezierPath
    {
        func centerOfEye(_ eye: Eye) -> CGPoint {
            let eyeOffset = skullRadius / Ratios.skullRadiusToEyeOffset
            var eyeCenter = skullCenter
            eyeCenter.y -= eyeOffset
            eyeCenter.x += ((eye == .left) ? -1 : 1) * eyeOffset
            return eyeCenter
        }

        let eyeRadius = skullRadius / Ratios.skullRadiusToEyeRadius
        let eyeCenter = centerOfEye(eye)

        let path: UIBezierPath
        if eyesOpen {
            path = UIBezierPath(
                arcCenter: eyeCenter, radius: eyeRadius,
                startAngle: 0, endAngle: CGFloat.pi * 2,
                clockwise: true
            )
        } else {
            path = UIBezierPath()
            path.move(to: CGPoint(x: eyeCenter.x - eyeRadius, y: eyeCenter.y))
            path.addLine(to: CGPoint(x: eyeCenter.x + eyeRadius, y: eyeCenter.y))
        }
        path.lineWidth = lineWidth

        return path
    }


    private func pathForMouth() -> UIBezierPath
    {
        let mouthWidth = skullRadius / Ratios.skullRadiusToMouthWidth
        let mouthHeight = skullRadius / Ratios.skullRadiusToMouthHeight
        let mouthOffset = skullRadius / Ratios.skullRadiusToMouthOffset

        let mouthRect = CGRect(
            x: skullCenter.x - mouthWidth / 2,
            y: skullCenter.y + mouthOffset,
            width: mouthWidth,
            height: mouthHeight
        )

        let smileOffset = CGFloat(max(-1, min(mouthCurvature, 1))) * mouthRect.height

        let start = CGPoint(x: mouthRect.minX, y: mouthRect.midY)
        let end = CGPoint(x: mouthRect.maxX, y: mouthRect.midY)
        let cp1 = CGPoint(x: start.x + mouthRect.width / 3, y: start.y + smileOffset)
        let cp2 = CGPoint(x: end.x - mouthRect.width / 3, y: start.y + smileOffset)

        let path = UIBezierPath()
        path.move(to: start)
        path.addCurve(to: end, controlPoint1: cp1, controlPoint2: cp2)
        path.lineWidth = lineWidth
        return path
    }

    private func pathForSkull() -> UIBezierPath {
        let path = UIBezierPath(
            arcCenter: skullCenter, radius: skullRadius,
            startAngle: 0, endAngle: 2 * CGFloat.pi,
            clockwise: false
        )
        path.lineWidth = lineWidth
        return path
    }

    override func draw(_ rect: CGRect) {
        color.set()
        pathForSkull().stroke()
        pathForEye(.left).stroke()
        pathForEye(.right).stroke()
        pathForMouth().stroke()
    }
}
```

```swift
//
//  FacialExpression.swift
//  FaceIt
//
//  Created by CS193p Instructor.
//  Copyright © 2015-17 Stanford University. All rights reserved.
//

import Foundation

// UI-independent representation of a facial expression

struct FacialExpression
{
    enum Eyes: Int {
        case open
        case closed
        case squinting
    }

    enum Mouth: Int {
        case frown
        case smirk
        case neutral
        case grin
        case smile

        var sadder: Mouth {
            return Mouth(rawValue: rawValue - 1) ?? .frown
        }
        var happier: Mouth {
            return Mouth(rawValue: rawValue + 1) ?? .smile
        }
    }

    var sadder: FacialExpression {
        return FacialExpression(eyes: self.eyes, mouth: self.mouth.sadder)
    }
    var happier: FacialExpression {
        return FacialExpression(eyes: self.eyes, mouth: self.mouth.happier)
    }

    let eyes: Eyes
    let mouth: Mouth
}
```