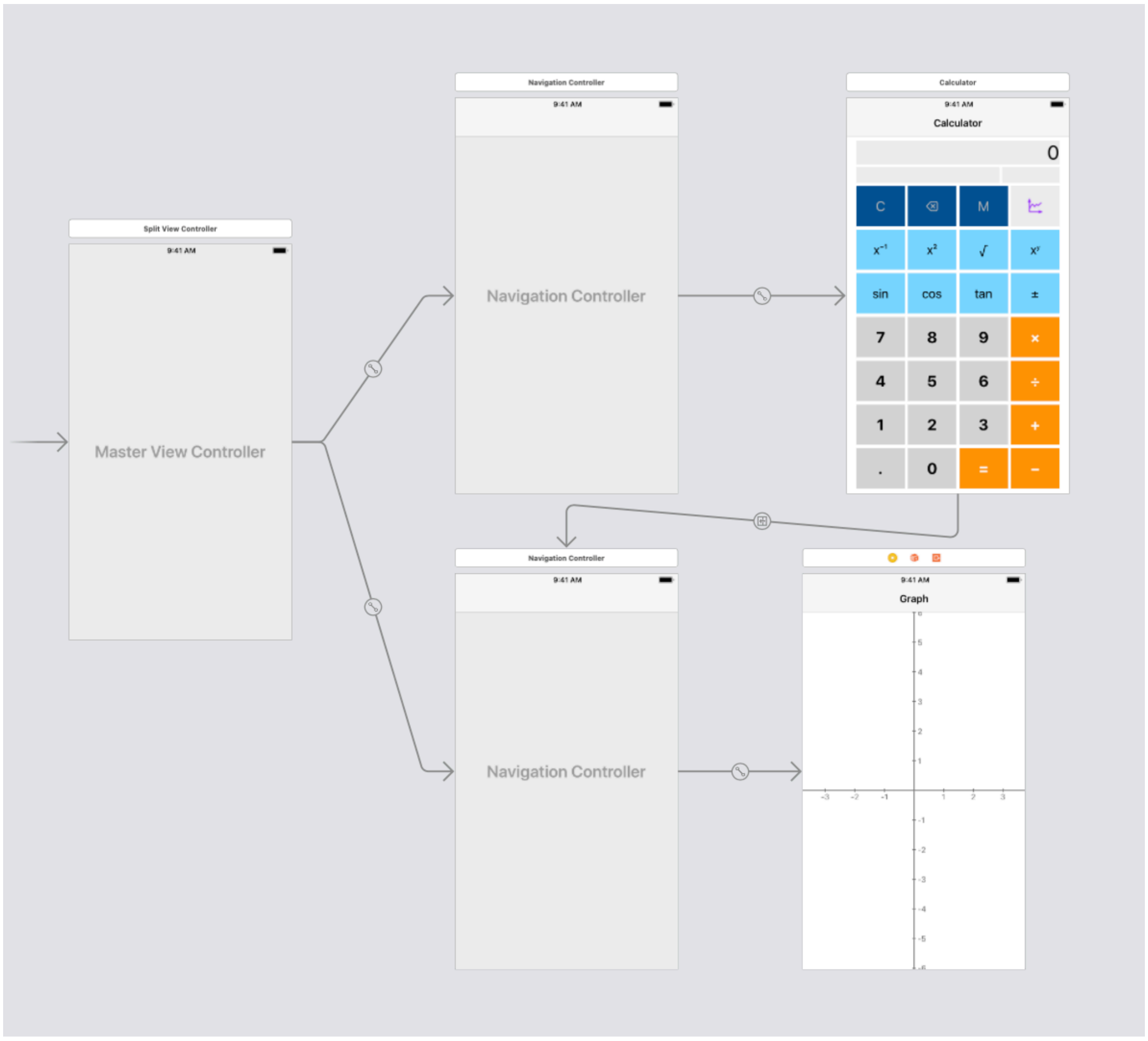
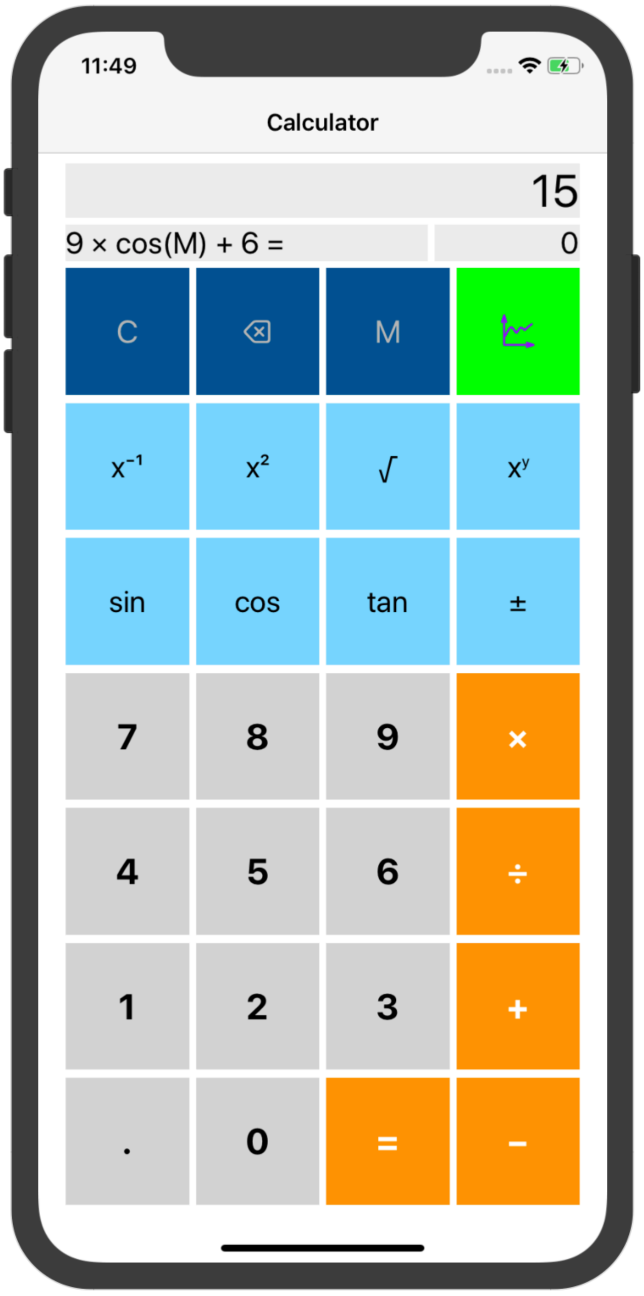
Assignment 2

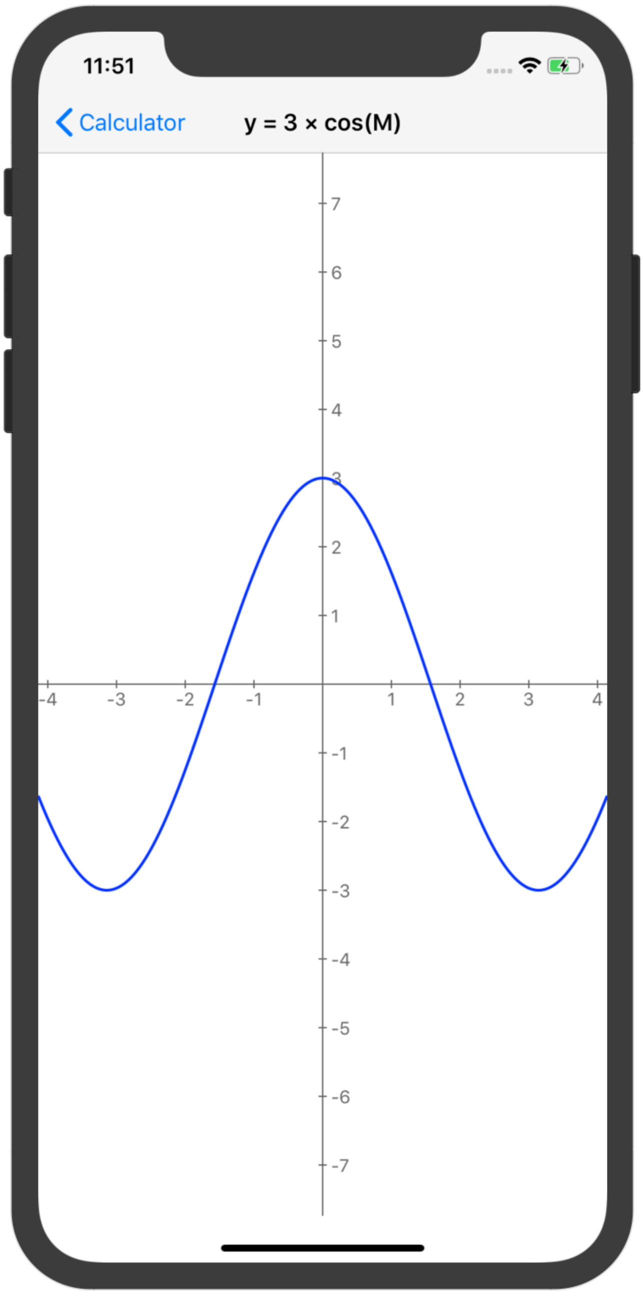
2016302580149 赵世晗

Using SplitViewController to achieve the graph calculator. As for the layout, we use the master view controller to show the calculator design and use the detail view controller to draw the function graph. We use the navigation controller to ensure we can back to the main view. The layout is displayed as follows:

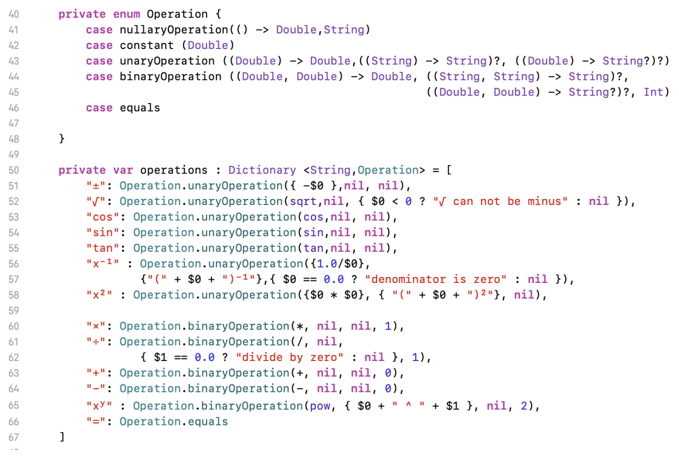


When it comes to the output result, here are the calculator interface and drawing graph interface. We use the stack view to standardize the buttons and labels and initialize the AxesDrawer to draw the plot.

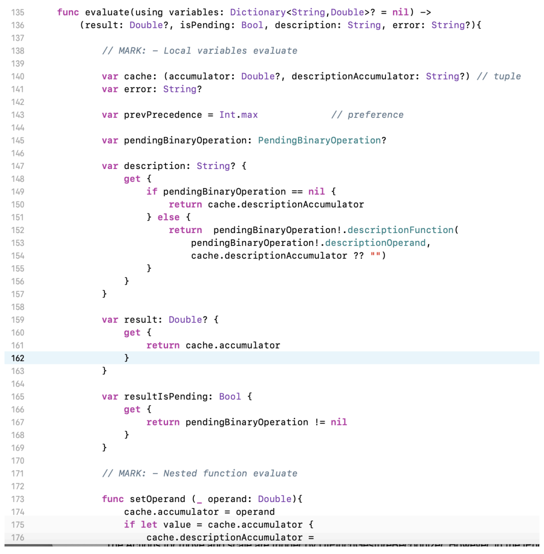




For the calculator controller, we use the stack structure to achieve the dual-variables operation. We store the operand in a dictionary to make the mapping and write the generic evaluate function to calculate the result. Here is the main code:



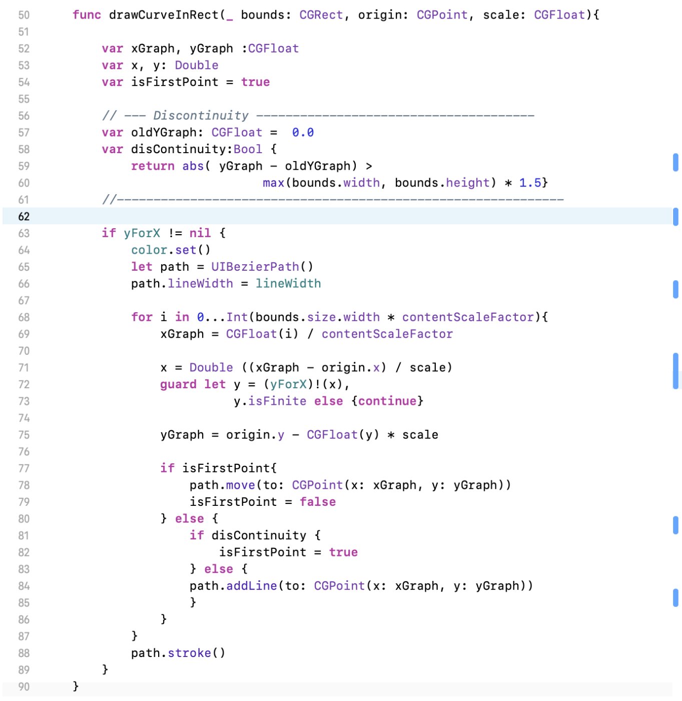
```swift
    private enum Operation {
        case nullaryOperation(() -> Double,String)
        case constant (Double)
        case unaryOperation ((Double) -> Double,((String) -> String)?, ((Double) -> String?)?)
        case binaryOperation ((Double, Double) -> Double, ((String, String) -> String)?,
                                                          ((Double, Double) -> String?)?, Int)
        case equals

    }

    private var operations : Dictionary <String,Operation> = [
        "±": Operation.unaryOperation({ -$0 },nil, nil),
        "√": Operation.unaryOperation(sqrt,nil, { $0 < 0 ? "√ can not be minus" : nil }),
        "cos": Operation.unaryOperation(cos,nil, nil),
        "sin": Operation.unaryOperation(sin,nil, nil),
        "tan": Operation.unaryOperation(tan,nil, nil),
        "x⁻¹" : Operation.unaryOperation({1.0/$0},
               {"(" + $0 + ")⁻¹"},{ $0 == 0.0 ? "denominator is zero" : nil }),
        "x²" : Operation.unaryOperation({$0 * $0}, { "(" + $0 + ")²"}, nil),

        "×": Operation.binaryOperation(*, nil, nil, 1),
        "÷": Operation.binaryOperation(/, nil,
               { $1 == 0.0 ? "divide by zero" : nil }, 1),
        "+": Operation.binaryOperation(+, nil, nil, 0),
        "−": Operation.binaryOperation(-, nil, nil, 0),
        "xʸ" : Operation.binaryOperation(pow, { $0 + " ^ " + $1 }, nil, 2),
        "=": Operation.equals
    ]
```



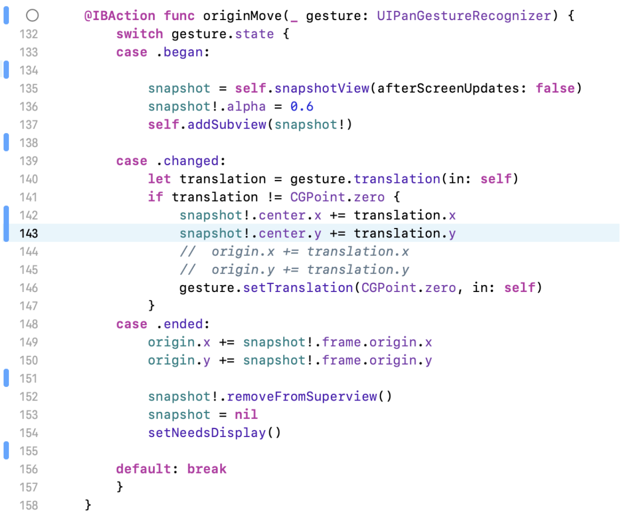
```swift
    func evaluate(using variables: Dictionary<String,Double>? = nil) ->
        (result: Double?, isPending: Bool, description: String, error: String?){

            // MARK: - Local variables evaluate

            var cache: (accumulator: Double?, descriptionAccumulator: String?) // tuple
            var error: String?

            var prevPrecedence = Int.max             // preference

            var pendingBinaryOperation: PendingBinaryOperation?

            var description: String? {
                get {
                    if pendingBinaryOperation == nil {
                        return cache.descriptionAccumulator
                    } else {
                        return  pendingBinaryOperation!.descriptionFunction(
                            pendingBinaryOperation!.descriptionOperand,
                            cache.descriptionAccumulator ?? "")
                    }
                }
            }

            var result: Double? {
                get {
                    return cache.accumulator
                }
            }

            var resultIsPending: Bool {
                get {
                    return pendingBinaryOperation != nil
                }
            }

            // MARK: - Nested function evaluate

            func setOperand (_ operand: Double){
                cache.accumulator = operand
                if let value = cache.accumulator {
                    cache.descriptionAccumulator =
```

As regard to variable M, we have to consider the expression as a whole. In this case, we use pending to decide whether or not it is a complete expression for the x-y function to draw. The draw button background color is decided by this boolean as well.

The Actions for move and scale are trigger by UIPinchGestureRecognizer. However, in the iPhone simulator I can not debug the scale manipulation. Here only the move action can work well.



```swift
    func drawCurveInRect(_ bounds: CGRect, origin: CGPoint, scale: CGFloat){

        var xGraph, yGraph :CGFloat
        var x, y: Double
        var isFirstPoint = true

        // --- Discontinuity --------------------------------------
        var oldYGraph: CGFloat =  0.0
        var disContinuity:Bool {
            return abs( yGraph - oldYGraph) >
                            max(bounds.width, bounds.height) * 1.5}
        //-----------------------------------------------------------

        if yForX != nil {
            color.set()
            let path = UIBezierPath()
            path.lineWidth = lineWidth

            for i in 0...Int(bounds.size.width * contentScaleFactor){
                xGraph = CGFloat(i) / contentScaleFactor

                x = Double ((xGraph - origin.x) / scale)
                guard let y = (yForX)!(x),
                          y.isFinite else {continue}

                yGraph = origin.y - CGFloat(y) * scale

                if isFirstPoint{
                    path.move(to: CGPoint(x: xGraph, y: yGraph))
                    isFirstPoint = false
                } else {
                    if disContinuity {
                        isFirstPoint = true
                    } else {
                    path.addLine(to: CGPoint(x: xGraph, y: yGraph))
                    }
                }
            }
            path.stroke()
        }
    }
```



```swift
    @IBAction func originMove(_ gesture: UIPanGestureRecognizer) {
        switch gesture.state {
        case .began:

            snapshot = self.snapshotView(afterScreenUpdates: false)
            snapshot!.alpha = 0.6
            self.addSubview(snapshot!)

        case .changed:
            let translation = gesture.translation(in: self)
            if translation != CGPoint.zero {
                snapshot!.center.x += translation.x
                snapshot!.center.y += translation.y
            //  origin.x += translation.x
            //  origin.y += translation.y
                gesture.setTranslation(CGPoint.zero, in: self)
            }
        case .ended:
            origin.x += snapshot!.frame.origin.x
            origin.y += snapshot!.frame.origin.y

            snapshot!.removeFromSuperview()
            snapshot = nil
            setNeedsDisplay()

        default: break
        }
    }
```