Lab 8: UIGestureRecognizer and UIMenuController

In this assignment, you will enhance your TouchTracker app. You will use three gesture recognizers to allow TouchTracker users to select, move, and delete lines. You will also see how to use another interesting iOS class, UIMenuController.

1. Create a new repository on GitHub and add your instructors as collaborators.

2. Open your TouchTracker project, then open the source control navigator. Remove your previous labs remote repository and add your new repository as a remote.

3. In the previous lab, you handled raw touches by implementing methods from UIResponder. Sometimes you want to detect a specific pattern of touches that make a gesture, like a pinch or a swipe. Instead of writing code to detect common gestures yourself, you can use instances of UIGestureRecognizer. Open DrawView.swift. Add an init?(coder:) method and instantiate a UITapGestureRecognizer that requires two taps to fire and calls the action method on its target.

```
required init?(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)

    let doubleTapRecognizer = UITapGestureRecognizer(target: self,
        action: #selector(DrawView.doubleTap(_:)))
    doubleTapRecognizer.numberOfTapsRequired = 2
    addGestureRecognizer(doubleTapRecognizer)
}
```

4. Now when a double-tap occurs on an instance of DrawView, the method doubleTap(_:) will be called on that instance. Implement this method in DrawView.swift.

```
@objc func doubleTap(_ gestureRecognizer: UIGestureRecognizer) {
    print("Recognized a double tap")

    currentLines.removeAll()
    finishedLines.removeAll()
    setNeedsDisplay()
}
```

5. Build and run the application, draw a few lines, and double-tap the screen to clear them. Notice that the first taps shows a red dot.

6. To prevent the red dot from appearing temporarily, you must prevent touchesBegan(_:with:) from being called on the view. You can tell a UIGestureRecognizer to delay calling touchesBegan(_:with:) on its view if it is still possible that its gesture might be recognized for that touch. In DrawView.swift, modify init?(coder:) to do just this.

```
required init?(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)

    let doubleTapRecognizer = UITapGestureRecognizer(target: self,
        action: #selector(DrawView.doubleTap(_:)))
    doubleTapRecognizer.numberOfTapsRequired = 2
    doubleTapRecognizer.delaysTouchesBegan = true
    addGestureRecognizer(doubleTapRecognizer)
}
```

7. Build and run the application, draw some lines, and then double-tap to clear them. You will no longer see the red dot while double-tapping.

8. The next step is to add another gesture recognizer that allows the user to select a line. (Later, a user will be able to delete the selected line.) You will install another UITapGestureRecognizer on the DrawView that only requires one tap. In DrawView.swift, modify init?(coder:) to add this gesture recognizer.

```
required init?(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)

    let doubleTapRecognizer = UITapGestureRecognizer(target: self,
        action: #selector(DrawView.doubleTap(_:)))
    doubleTapRecognizer.numberOfTapsRequired = 2
    doubleTapRecognizer.delaysTouchesBegan = true
    addGestureRecognizer(doubleTapRecognizer)

    let tapRecognizer =
        UITapGestureRecognizer(target: self, action: #selector(DrawView.tap(_:)))
    tapRecognizer.delaysTouchesBegan = true
    addGestureRecognizer(tapRecognizer)
}
```

9. Now, implement tap(_:) in DrawView.swift to log the tap to the console.

```
@objc func tap(_ gestureRecognizer: UIGestureRecognizer) {
    print("Recognized a tap")
}
```

10. Build and run the application. Try tapping and double-tapping. Tapping once logs the appropriate message to the console. Double-tapping, however, triggers both tap(_:) and doubleTap(_:). In situations where you have multiple gesture recognizers, it is not uncommon to have one gesture recognizer fire and claim a touch when you really wanted another gesture recognizer to handle it. In these cases, you set up dependencies between recognizers that say, "Wait a moment before you fire, because this touch might be mine!"

11. In init?(coder:), make it so the tapRecognizer waits until the doubleTapRecognizer fails to recognize a double-tap before claiming the single tap for itself.

```
required init?(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)

    let doubleTapRecognizer = UITapGestureRecognizer(target: self,
        action: #selector(DrawView.doubleTap(_:)))
    doubleTapRecognizer.numberOfTapsRequired = 2
    doubleTapRecognizer.delaysTouchesBegan = true
    addGestureRecognizer(doubleTapRecognizer)

    let tapRecognizer =
        UITapGestureRecognizer(target: self, action: #selector(DrawView.tap(_:)))
    tapRecognizer.delaysTouchesBegan = true
    tapRecognizer.require(toFail: doubleTapRecognizer)
    addGestureRecognizer(tapRecognizer)
}
```

12. Build and run the application again and try out some taps. A single tap now takes a small amount of time to fire after the tap occurs, but double-tapping no longer triggers the tap(_:) message.

13. Next, let's build on the DrawView so that the user can select a line when it is tapped. First, add a property at the top of DrawView.swift to hold on to the index of a selected line.

```
class DrawView: UIView {

    var currentLines = [NSValue:Line]()
    var finishedLines = [Line]()
    var selectedLineIndex: Int?
    …
```

14. Now modify draw(_:) to draw the selected line in green.

```
override func draw(_ rect: CGRect) {
    finishedLineColor.setStroke()
    for line in finishedLines {
        stroke(line)
    }

    currentLineColor.setStroke()
    for (_,line) in currentLines {
        stroke(line)
    }

    if let index = selectedLineIndex {
        UIColor.green.setStroke()
        let selectedLine = finishedLines[index]
        stroke(selectedLine)
    }
}
```

15. Still in DrawView.swift, add an indexOfLine(at:) method that returns the index of the Line closest to a given point.

```
func indexOfLine(at point: CGPoint) -> Int? {
    // Find a line close to point
    for (index, line) in finishedLines.enumerated() {
        let begin = line.begin
        let end = line.end

        // Check a few points on the line
        for t in stride(from: CGFloat(0), to: 1.0, by: 0.05) {
            let x = begin.x + ((end.x - begin.x) * t)
            let y = begin.y + ((end.y - begin.y) * t)

            // If the tapped point is within 20 points, let's return this line
            if hypot(x - point.x, y - point.y) < 20.0 {
                return index
            }
        }
    }

    // If nothing is close enough to the tapped point, then we did not select a line
    return nil
}
```

16. In DrawView.swift, update tap(_:) to call location(in:) on the gesture recognizer, pass the result to indexOfLine(at:), and make the returned index the selectedLineIndex.

```
@objc func tap(_ gestureRecognizer: UIGestureRecognizer) {
    print("Recognized a tap")

    let point = gestureRecognizer.location(in: self)
    selectedLineIndex = indexOfLine(at: point)

    setNeedsDisplay()
}
```

17. If the user double-taps to clear all lines while a line is selected, the application will trap. To address this, update doubleTap(_:) to set the selectedLineIndex to nil.

```
@objc func doubleTap(_ gestureRecognizer: UIGestureRecognizer) {
    print("Recognized a double tap")

    selectedLineIndex = nil
    currentLines.removeAll()
    finishedLines.removeAll()
    setNeedsDisplay()
}
```

18. Build and run the application. Draw a few lines and then tap one. The tapped line should appear in green (remember that it takes a moment before the tap is known not to be a double-tap).

19. Next you are going to make it so that when the user selects a line, a menu with the option to delete that line appears where the user tapped. Do this in DrawView.swift's tap(_:) method if the user has tapped on a line. If the user tapped somewhere that is not near a line, the currently selected line will be deselected and the menu controller will hide.

```
@objc func tap(_ gestureRecognizer: UIGestureRecognizer) {
    print("Recognized a tap")

    let point = gestureRecognizer.location(in: self)
    selectedLineIndex = indexOfLine(at: point)

    // Grab the menu controller
    let menu = UIMenuController.shared

    if selectedLineIndex != nil {

        // Make DrawView the target of menu item action messages
        becomeFirstResponder()

        // Create a new "Delete" UIMenuItem
        let deleteItem = UIMenuItem(title: "Delete",
                                    action: #selector(DrawView.deleteLine(_:)))
        menu.menuItems = [deleteItem]

        // Tell the menu where it should come from and show it
        let targetRect = CGRect(x: point.x, y: point.y, width: 2, height: 2)
        menu.setTargetRect(targetRect, in: self)
        menu.setMenuVisible(true, animated: true)
    } else {
        // Hide the menu if no line is selected
        menu.setMenuVisible(false, animated: true)
    }

    setNeedsDisplay()
}
```

20. If you have a custom view class that needs to become the first responder, you must also override canBecomeFirstResponder. In DrawView.swift, override this property to return true.

```
override var canBecomeFirstResponder: Bool {
    return true
}
```

21. Finally, implement deleteLine(_:) in DrawView.swift.

```
@objc func deleteLine(_ sender: UIMenuController) {
    // Remove the selected line from the list of finishedLines
    if let index = selectedLineIndex {
        finishedLines.remove(at: index)
        selectedLineIndex = nil

        // Redraw everything
        setNeedsDisplay()
    }
}
```

22. Build and run the application. Draw a line, tap on it, and then select Delete from the menu item.

23. Add a property observer to selectedLineIndex in DrawView.swift that sets the menu controller to be not visible if the index is set to nil.

```
var selectedLineIndex: Int? {
    didSet {
        if selectedLineIndex == nil {
            let menu = UIMenuController.shared
            menu.setMenuVisible(false, animated: true)
        }
    }
}
```

24. Build and run the application. Draw a line, select it, and then double-tap the background. The line and the menu controller will no longer be visible.

25. In DrawView.swift, instantiate a UILongPressGestureRecognizer in init?(coder:) and add it to the DrawView. The long press will be used to indicate the user wants to move a line. In a later step we will implement another gesture recognizer that will take care of moving the line as the user moves their finger across the screen.

```
required init?(coder aDecoder: NSCoder) {
    ...
    addGestureRecognizer(tapRecognizer)

    let longPressRecognizer = UILongPressGestureRecognizer(target: self,
            action: #selector(DrawView.longPress(_:)))
    addGestureRecognizer(longPressRecognizer)
}
```

26. In DrawView.swift, implement longPress(_:).

```
@objc func longPress(_ gestureRecognizer: UIGestureRecognizer) {
    print("Recognized a long press")

    if gestureRecognizer.state == .began {
        let point = gestureRecognizer.location(in: self)
        selectedLineIndex = indexOfLine(at: point)

        if selectedLineIndex != nil {
            currentLines.removeAll()
        }
    } else if gestureRecognizer.state == .ended {
        selectedLineIndex = nil
    }

    setNeedsDisplay()
}
```

27. Build and run the application. Draw a line and then press and hold it; the line will turn green and become the selected line. When you let go, the line will revert to its former color and will no longer be the selected line.

28. In DrawView.swift, declare a UIPanGestureRecognizer as a property so that you have access to it in all of your methods.

```
class DrawView: UIView {

    var currentLines = [NSValue:Line]()
    var finishedLines = [Line]()
    var selectedLineIndex: Int? {
        ...
    }
    var moveRecognizer: UIPanGestureRecognizer!
```

29. Next, in DrawView.swift, add code to init?(coder:) to instantiate a UIPanGestureRecognizer, set one of its properties, and add it to the DrawView.

```
…
  let longPressRecognizer = UILongPressGestureRecognizer(target: self,
        action: #selector(DrawView.longPress(_:)))
  addGestureRecognizer(longPressRecognizer)

  moveRecognizer = UIPanGestureRecognizer(target: self,
                              action: #selector(DrawView.moveLine(_:)))
  moveRecognizer.cancelsTouchesInView = false
  addGestureRecognizer(moveRecognizer)
}
```

30. In DrawView.swift, add a simple implementation for the action method:

```
@objc func moveLine(_ gestureRecognizer: UIPanGestureRecognizer) {
  print("Recognized a pan")
}
```

31. Build and run the app and draw some lines. Because cancelsTouchesInView is false, the pan gesture is recognized, but lines can still be drawn. You can comment out the line that sets cancelsTouchesInView and run again to see the difference. **Be sure to uncomment the cancelsTouchesInView line before moving forward**.

32. We want our gesture recognizers to work together simultaneously and therefore we will have to incorporate a delegate method form the UIGestureRecognizerDelegate protocol in order to make this happen. In DrawView.swift, declare that DrawView conforms to the UIGestureRecognizerDelegate protocol.

```
class DrawView: UIView, UIGestureRecognizerDelegate {

  var currentLines = [NSValue:Line]()
  var finishedLines = [Line]()
  var selectedLineIndex: Int? {
    ...
  }
  var moveRecognizer: UIPanGestureRecognizer!
```

33. In init?(coder:), set the DrawView to be the delegate of the UIPanGestureRecognizer.

```
…
    let longPressRecognizer = UILongPressGestureRecognizer(target: self,
          action: #selector(DrawView.longPress(_:)))
    addGestureRecognizer(longPressRecognizer)

    moveRecognizer = UIPanGestureRecognizer(target: self,
                                action: #selector(DrawView.moveLine(_:)))
    moveRecognizer.delegate = self
    moveRecognizer.cancelsTouchesInView = false
    addGestureRecognizer(moveRecognizer)
}
```

34. In DrawView.swift, implement the delegate method to return true.

```
func gestureRecognizer(_ gestureRecognizer: UIGestureRecognizer,
      shouldRecognizeSimultaneouslyWith
      otherGestureRecognizer: UIGestureRecognizer) -> Bool {
   return true
}
```

35. In DrawView.swift, implement moveLine(_:). Notice that because you will send the gesture recognizer a method from the UIPanGestureRecognizer class, the parameter of this method must be a reference to an instance of UIPanGestureRecognizer rather than UIGestureRecognizer.

```
@objc func moveLine(_ gestureRecognizer: UIPanGestureRecognizer) {
  print("Recognized a pan")

  // If a line is selected...
  if let index = selectedLineIndex {
    // When the pan recognizer changes its position...
    if gestureRecognizer.state == .changed {
      // How far has the pan moved?
      let translation = gestureRecognizer.translation(in: self)

      // Add the translation to the current beginning and end points of the line
      // Make sure there are no copy and paste typos!
      finishedLines[index].begin.x += translation.x
      finishedLines[index].begin.y += translation.y
      finishedLines[index].end.x += translation.x
      finishedLines[index].end.y += translation.y

      // Redraw the screen
      setNeedsDisplay()
    }
  } else {
    // If no line is selected, do not do anything
    return
  }
}
```

36. Build and run the application. Touch and hold on a line and begin dragging – and you will immediately notice that the line and your finger are way out of sync. What is going on? You are adding the current translation over and over again to the line's original end points. You really need the gesture recognizer to report the change in translation since the last time this method was called instead. Fortunately, you can do this. You can set the translation of a pan gesture recognizer back to the zero point every time it reports a change. Then, the next time it reports a change, it will have the translation since the last event.

37. Near the bottom of moveLine(_:) in DrawView.swift, add the following line of code.

```
finishedLines[index].end.x += translation.x
finishedLines[index].end.y += translation.y

gestureRecognizer.setTranslation(CGPoint.zero, in: self)

// Redraw the screen
setNeedsDisplay()
```

38. Build and run the application and move a line around. Works great!

39. When you have completed this assignment commit and push your code to GitHub, then submit your commit ID and repository URL using the LMS.