Touch Events and UIResponder-7

In this assignment, you will build a new app called TouchTracker. TouchTracker is an app that lets the user draw by touching the screen. You will create a view that draws lines in response to the user dragging across it. Using multitouch, the user will be able to draw more than one line at a time.

1. Create a new repository on GitHub and add your instructor as a collaborator.

2. You will be starting a new app. Open Xcode (if your old project is loaded close it) and create a new single view app project, name it TouchTracker.

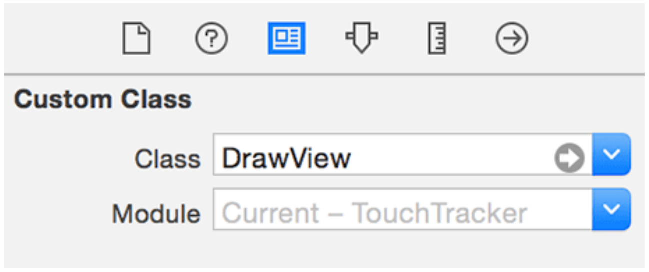
3. Create a new Swift file named Line. In Line.swift, import CoreGraphics and declare the Line struct. Declare two CGPoint properties that will determine the beginning and ending point for the line.

```
import Foundation
import CoreGraphics

struct Line {
   var begin = CGPoint.zero
   var end = CGPoint.zero
}
```

4. In addition to a custom struct, TouchTracker needs a custom view. Create a new Swift file named DrawView. In DrawView.swift, define the DrawView class. Add two properties: an optional Line to keep track of a line that is possibly being drawn and an array of Lines to keep track of lines that have been drawn.

```
~~import Foundation~~
import UIKit

class DrawView: UIView {
   var currentLine: Line?
   var finishedLines = [Line]()
}
```

5. An instance of DrawView will be the view of the application's rootViewController, the default ViewController included in the project. The view controller needs to know that its view will be an instance of DrawView. Open Main.storyboard. Select the View using the document outline of the default ViewController, and open the identity inspector (Command-Option-3). Under Custom Class, change the Class to DrawView.



Custom Class

Class: DrawView

Module: Current – TouchTracker

6. An instance of DrawView needs to be able draw lines. You are going to write a method that uses UIBezierPath to create and stroke a path based on the properties of a given Line. Then you will override draw(_:) to draw the lines in the array of finished lines as well as the current line, if any. In DrawView.swift, implement the method for stroking lines and override draw(_:).

```
var currentLine: Line?
var finishedLines = [Line]()

func stroke(_ line: Line) {
    let path = UIBezierPath()
    path.lineWidth = 10
    path.lineCapStyle = .round

    path.move(to: line.begin)
    path.addLine(to: line.end)
    path.stroke()
}

override func draw(_ rect: CGRect) {
    // Draw finished lines in black
    UIColor.black.setStroke()
    for line in finishedLines {
        stroke(line)
    }

    if let line = currentLine {
        // If there is a line currently being drawn, do it in red
        UIColor.red.setStroke()
        stroke(line)
    }
}
```

7. A line is defined by two points. Your Line stores these points as properties named begin and end. When a touch begins, you will create a Line and set both of its properties to the point where the touch began. When the touch moves, you will update the Line's end. When the touch ends, you will have your complete Line. In DrawView.swift, implement touchesBegan(_:with:) to create a new line.

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
  let touch = touches.first!

  // Get location of the touch in view's coordinate system
  let location = touch.location(in: self)

  currentLine = Line(begin: location, end: location)

  setNeedsDisplay()
}
```

8. This code first figures out the location of the touch within the view's coordinate system. Then it calls setNeedsDisplay(), which flags the view to be redrawn at the end of the run loop. Next, also in DrawView.swift, implement touchesMoved(_:with:) so that it updates the end of the currentLine.

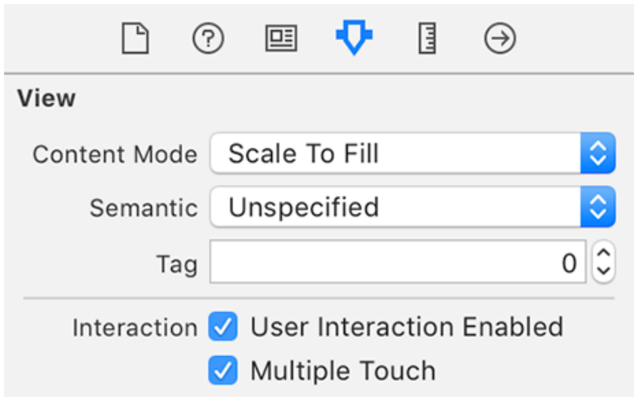
```
override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
  let touch = touches.first!
  let location = touch.location(in: self)

  currentLine?.end = location

  setNeedsDisplay()
}
```

9. Still in DrawView.swift, update the end location of the currentLine and add it to the finishedLines array when the touch ends.

```
override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
  if var line = currentLine {
    let touch = touches.first!
    let location = touch.location(in: self)
    line.end = location

    finishedLines.append(line)
  }
  currentLine = nil

  setNeedsDisplay()
}
```

10. Build and run the application and draw some lines on the screen. While you are drawing, the lines will appear in red. Once finished, they will appear in black.

11. In Main.storyboard, select the Draw View and open the attributes inspector. Check the box labeled Multiple Touch, which will set the DrawView instance's multipleTouchesEnabled property to true.



View
Content Mode: Scale To Fill
Semantic: Unspecified
Tag: 0
Interaction: [✓] User Interaction Enabled
[✓] Multiple Touch

12. Instead of adding more properties, you are going to replace the single Line with a dictionary containing instances of Line. In DrawView.swift, add a new property to replace currentLine.

```
class DrawView: UIView {
  ~~var currentLine: Line?~~
  **var currentLines = [NSValue:Line]()**
```

13. The key to store the line in the dictionary will be derived from the UITouch object that the line corresponds to. As more touch events occur, you can use the same algorithm to derive the key from the UITouch that triggered the event and use it to look up the appropriate Line in the dictionary. In DrawView.swift, update the code in touchesBegan(_:with:).

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
  ~~let touch = touches.first!~~
  ~~// Get location of the touch in view's coordinate system~~
  ~~let location = touch.location(in: self)~~
  ~~currentLine = Line(begin: location, end: location)~~
  // Log statement to see the order of events
  print(#function)
  for touch in touches {
    let location = touch.location(in: self)
    let newLine = Line(begin: location, end: location)

    let key = NSValue(nonretainedObject: touch)
    currentLines[key] = newLine
  }
  setNeedsDisplay()
}
```

14. Next, update touchesMoved(_:with:) in DrawView.swift so that it can look up the right Line.

```
override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
   ~~let touch = touches.first!~~
   ~~let location = touch.location(in: self)~~

   ~~currentLine?.end = location~~

   // Log statement to see the order of events
   print(#function)

   for touch in touches {
      let key = NSValue(nonretainedObject: touch)
      currentLines[key]?.end = touch.location(in: self)
   }

   setNeedsDisplay()
}
```

15. Now, update touchesEnded(_:with:) to move any finished lines into the finishedLines array.

```
override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
   ~~if var line = currentLine {~~
      ~~let touch = touches.first!~~
      ~~let location = touch.location(in: self)~~
      ~~line.end = location~~

      ~~finishedLines.append(line)~~
   ~~}~~
   ~~currentLine = nil~~

   // Log statement to see the order of events
   print(#function)

   for touch in touches {
      let key = NSValue(nonretainedObject: touch)
      if var line = currentLines[key] {
         line.end = touch.location(in: self)

         finishedLines.append(line)
         currentLines.removeValue(forKey: key)
      }
   }

   setNeedsDisplay()
}
```

16. Update draw(_:) to draw each line in currentLines.

```
override func draw(_ rect: CGRect) {
   // Draw finished lines in black
   UIColor.black.setStroke()
   for line in finishedLines {
      stroke(line)
   }

~~   if let line = currentLine {~~
~~      // If there is a line currently being drawn, do it in red~~
~~      UIColor.red.setStroke()~~
~~      stroke(line)~~
~~   }~~

   // Draw current lines in red
   UIColor.red.setStroke()
   for (_, line) in currentLines {
      stroke(line)
   }
}
```

17. Build and run the application and start drawing lines with multiple fingers. (On the simulator, hold down the Option key while you drag to simulate multiple fingers.) Notice the ordering of the log messages in the console.

18. The last thing left for the basics of TouchTracker is to handle what happens when a touch is canceled. A touch can be canceled when the application is interrupted by the OS (for example, when a phone call comes in) while a touch is currently on the screen. When a touch is canceled, any state it set up should be reverted. In this case, you should remove any lines in progress. In DrawView.swift, implement touchesCancelled(_:with:).

```
override func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?) {
   // Log statement to see the order of events
   print(#function)

   currentLines.removeAll()

   setNeedsDisplay()
}
```

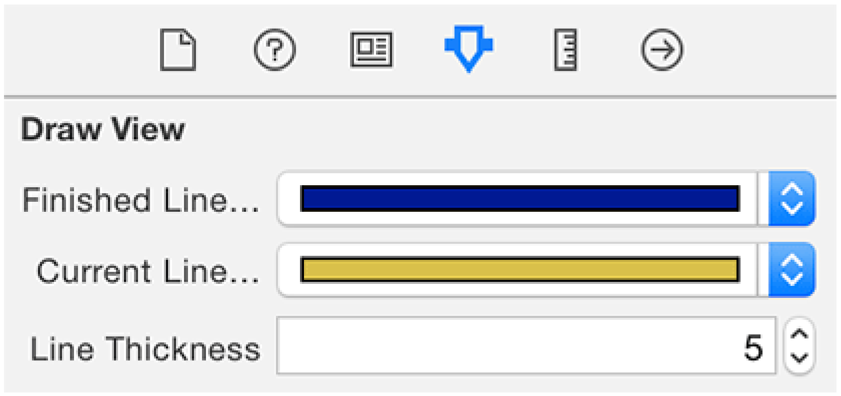
19. In DrawView.swift, declare three properties to reference these values. Give them default values and have the view flag itself for redrawing whenever these properties change.

```
var currentLines = [NSValue:Line]()
var finishedLines = [Line]()

@IBInspectable var finishedLineColor: UIColor = UIColor.black {
  didSet {
    setNeedsDisplay()
  }
}

@IBInspectable var currentLineColor: UIColor = UIColor.red {
  didSet {
    setNeedsDisplay()
  }
}

@IBInspectable var lineThickness: CGFloat = 10 {
  didSet {
    setNeedsDisplay()
  }
}
```

20. The @IBInspectable keyword lets Interface Builder know that this is a property that you want to customize through the attributes inspector. Many of the common types are supported by @IBInspectable: Booleans, strings, numbers, CGPoint, CGSize, CGRect, UIColor, UIImage, and a few more are all candidates. Now update stroke(_:) and drawView(_:) to use these new properties.

```
func stroke(_ line: Line) {
  let path = UIBezierPath()
  ~~path.lineWidth = 10~~
  **path.lineWidth = lineThickness**
  path.lineCapStyle = .round

  path.move(to: line.begin)
  path.addLine(to: line.end)
  path.stroke()
}

override func draw(_ rect: CGRect) {
  ~~// Draw finished lines in black~~
  ~~UIColor.black.setStroke()~~
  **finishedLineColor.setStroke()**
  for line in finishedLines {
    stroke(line)
  }

  ~~// Draw current lines in red~~
  ~~UIColor.red.setStroke()~~
  **currentLineColor.setStroke()**
  for (_, line) in currentLines {
    stroke(line)
  }
}
```

21. Now, when you add a DrawView to the canvas in Interface Builder, you can customize these three properties in the attributes inspector to be different for different instances.


Draw View

Finished Line... 🟦

Current Line... 🟨

Line Thickness 5

22. Change the three properties from the storyboard, then build and run the application. See how the lines draw based on the properties you changed.