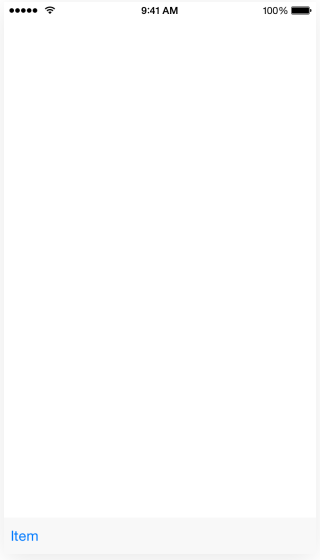
FingerPainter
Lesson 4

Description

Add an Image View for the drawing, and update the implementation of the event handlers to simulate the act of drawing from point to point.



Learning Outcomes

- Generalize the purposes of `UIView` and `UIImageView` interface elements.
- Practice establishing outlet connections between a view and controller, and declaring properties.
- Strategize a drawing method using points generated by touch input.
- Practice using swift optional binding to check for the presence of values.
- Observe how external and local parameter names can lend better semantics when calling and implementing functions.
- Practice creating custom breakpoint actions to print console messages and to inspect app behavior.

Vocabulary

Image View	`UIImageView`	outlet connection
`@IBOutlet`	property	optional
`CGPoint`	optional binding	`if let`
external parameter name		

Materials

- **FingerPainter Lesson 4** Xcode project

Opening

Now that we have the points of the touches, how might we draw a continuous line as a finger moves across the screen?

Agenda

- Discuss how one approach is to draw a line from point to point as the user drags a finger across the screen, updating the image displayed on the screen as the user "draws."
- Explain how `drawRect:` has a particular purpose, and can become inefficient for dynamic drawing.
- Discuss the use of a `UIImageView` object to accomplish the task of displaying a continuously updated image containing the line to be drawn.
- Using Interface Builder, delete the custom View object from the interface.
- Using the Project Navigator (⌘1), delete the **CanvasView.swift** file.
- Using Interface Builder and the Object Library (⌥⌘L), add a new Image View object that occupies the remainder of the interface. Use the menu item *Editor* > *Resolve Auto Layout Issues* > *Add Missing Constraints* to add Auto Layout constraints for the view.
- Using the Assistant Editor (⌥⌘↩), Control-drag from the Image View to the `ViewController` class to create an outlet property.

```
@IBOutlet weak var canvas: UIImageView!
```

- Discuss how the controller will need to keep track of the first point obtained when `touchesBegan:withEvent:` is called, to serve as the first starting point for the line to be drawn.
- Declare a `CGPoint` property in the `ViewController` class.

```
var start: CGPoint?
```

- Discuss how the `start` point will frequently change as the user touches the screen, and how the type is an optional, since the `ViewController` initializer will not assign a value to the property.
- Update the implementation of `touchesBegan:withEvent:` to obtain the coordinate from the `UIImageView`, and to store the point of the touch in the `start` property.

```
override func touchesBegan(touches: Set<NSObject>,
   withEvent event: UIEvent) {
   let touch = touches.first as! UITouch
   start = touch.locationInView(canvas)
}
```

- Discuss how, because the drawn line will follow the finger as it moves, the `touchesMoved:withEvent:` method should draw a line from the controller `start` property to the new point captured within the `touchesMoved:withEvent:` method.
- Discuss how, for each subsequent movement, a new `start` should be stored, so that the next time `touchesMoved:withEvent:` is called, a line can be drawn from the new `start` to the new point captured by the subsequent call to `touchesMoved:withEvent:`.
- Update the implementation of `touchesMoved:withEvent:`.

```
override func touchesMoved(touches: Set<NSObject>,
   withEvent event: UIEvent) {
   let touch = touches.first as! UITouch
   let end = touch.locationInView(canvas)
   if let start = self.start {
      drawFromPoint(start, toPoint: end)
   }
   start = end
}
```

- Discuss how Swift optional binding is used to check the value of the `start` property, ensuring that a `CGPoint` value has been assigned, before passing it to `drawFromPoint:toPoint:`.
- Add an empty implementation of `drawFromPoint:toPoint:`.

```
func drawFromPoint(start: CGPoint, toPoint end: CGPoint){
   // print coordinate with breakpoint here
}
```

- Explain the use of the external and local parameter names `toPoint` and `end`.
- Delete any existing breakpoints in the `ViewController` class, and add a new custom breakpoint to the body of `drawFromPoint:toPoint:` that uses a **Log Message** action to print the `x` and `y` components of the `start` and `end` `CGPoint` arguments, and automatically continues (e.g., **Draw from @start.x@,@start.y@ to @end.x@,@end.y@**).
- Run the app (⌘R), click and drag to simulate a moving touch, and observe the console (⇧⌘C) output.

Closing

Why do you think `drawFromPoint:toPoint:` uses a local and external parameter name? Discuss the semantic benefit of this approach when calling the function, and predict the benefits when writing the body of the function.

Modifications And Extensions

- Dig deeper into the Apple documentation to discover the difference between `UIView` and `UIImageView` rendering. Explain why, for dynamic views or animations, one object is better than the other.

Resources

UIKit User Interface Catalog: Image Views https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/UIKitUICatalog/UIImageView.html

UIImageView Class Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIImageView_Class/index.html

Interface Builder Help: Adding an Object to Your Interface https://developer.apple.com/library/ios/recipes/xcode_help-IB_objects_media/Chapters/AddingObject.html

Interface Builder Help: Creating an Outlet Connection https://developer.apple.com/library/ios/recipes/xcode_help-IB_connections/chapters/CreatingOutlet.html

The Swift Programming Language: Properties https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Properties.html

Event Handling Guide for iOS http://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/Introduction/Introduction.html

CGGeometry Reference https://developer.apple.com/library/ios/documentation/GraphicsImaging/Reference/CGGeometry/index.html

The Swift Programming Language: Function Parameter Names https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Functions.html#//apple_ref/doc/uid/TP40014097-CH10-ID166

The Swift Programming Language: Optional Binding https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html#//apple_ref/doc/uid/TP40014097-CH5-ID333

Setting Breakpoint Actions and Options http://developer.apple.com/library/ios/recipes/xcode_help-breakpoint_navigator/articles/setting_breakpoint_actions_and_options.html