Exploring Gestures

Project Overview

Working with Gesture Recognizers

UIView supports a number of gesture recognizers that simplify the implementation of complex gesture recognition. UIGestureRecognizer is the superclass that supports a number of pre-built subclasses, including:

- UIPinchGestureRecognizer – using two or more fingers in a finger pinch motion to squeeze or expand.
- UIPanGestureRecognizer – using one or more fingers in a dragging motion to move around a parent view.
- UIRotationGestureRecognizer – using two or more fingers in a rotating motion to twist a view's orientation around a 360º access.
- UITapGestureRecognizer – using one or more fingers touching the screen one or more times to indicate a number of taps on the view.
- UILongPressGestureRecognizer – using one or more fingers touching down for an extended period on a view.
- UISwipeGestureRecognizer – using one or more fingers in a dragging motion left, right, up, or down to indicate a vertical or horizontal swipe motion.

UIGestureRecognizers are created by their class using the built-in initializers for target and action. Below is an example of a UIGestureRecognizer instantiation:

```
var gr: UIGestureRecognizer =
   UIPinchGestureRecognizer(target:self, action: #selector(someFunction(_:)))
```

Once you create your gesture recognizer, all you have to do is add it to your target view. As a 'for instance' using the example above:

```
someView.addGestureRecognizer(gr)
```

Certain UIGestureRecognizer pre-built subclasses support specific properties, such as numberOfTouchesRequired, numberOfTapsRequired, etc, that allow you to customize the gesture recognizer's specific recognition characteristics.

Your handler functions are passed the gesture recognizer object that triggered the call. Each gesture recognizer contains properties such as:

- .view – the view the gesture occurred on.
- .state – what state the gesture recognizer is in:
    - .began
    - .changed
    - .ended
    - .failed
- .scale – for UIPinchGestureRecognizers only.
- .translation(in:) – the x,y coordinate of the gesture as translated for the specified view.

Exploring Gestures

Some gesture recognizers may interfere with the gesture recognition of other gesture recognizers, such as a UIPanGestureRecognizer and a UISwipeGestureRecognizer. Both look for one or more fingers dragging across the view. A UIGestureRecognizerDelegate group of functions exist to help resolve these conflicts. Lookup UIGestureRecognizerDelegate for a detailed list and description of these delegate functions.

A UIGestureRecognizer can only be added to a single view, not multiple views. You must create separate gesture recognizers for each view you want to add one to.

Create a new project

1. Download the “TestPattern.png” image from eCampus for this project.
2. Create a new Xcode project for a “Single View” application using Swift and Universal, and save it as “Exploring Gestures” to your Desktop. Then drag your downloaded image to this project by first dragging it into your “Exploring Gestures” folder, then into your project.
3. Create a new UIImageView in your UIViewController’s main view in Storyboard, and initialize it with the “TestPattern.png” image. Enable ‘User Interaction Enabled’ and ‘Multiple Touch’, as these are off by default in a UIImageView. ‘User Interaction Enabled’ enables touch detection on the view, while ‘Multiple Touch’ allows for detection of multiple fingers. Set its size to 150 x 150, and Content Mode to ‘Scale to Fill’.
4. Connect the “TestPattern.png” image view to your “ViewController.swift” source as “testPattern”.

Add the main UIView UIGestureRecognizers

5. In viewController.swift, create two new class vars for ‘viewFrame’ and ‘viewCenter’. Type viewFrame as an auto-dereference optional of type CGRect, and viewCenter as an auto-dereference optional of type CGPoint. Also add an optional auto-dereference CGFloat var named rads.
6. Create a swipe gesture handler function to handle all four possible swipe directions. You can read the swipe direction right off of the gestureRecognizer parameter passed by the gesture recognizer. You will want to trigger your handling to occur when the gestureRecognizer state is .ended, so you only trigger once during a long swipe operation. You will call this handler from all four gesture recogniers. Here’s an example to get you started:

```
func swipeGesture(_ gestureRecognizer: UISwipeGestureRecognizer) {
    if gestureRecognizer.state == .ended {
        let viewSize = self.view.bounds.size
        let testSize = self.testPattern.bounds.size
        UIView.animate(withDuration: 0.75, animations: {
            switch gestureRecognizer.direction {
            case UISwipeGestureRecognizerDirection.left:
                self.testPattern.frame.origin.x = 0
            case UISwipeGestureRecognizerDirection.right:
                let x = viewSize.width - testSize.width
                self.testPattern.frame.origin.x = x
            case UISwipeGestureRecognizerDirection.up:
                self.testPattern.frame.origin.y = 0
            case UISwipeGestureRecognizerDirection.down:
                let y = viewSize.height - testSize.height
                self.testPattern.frame.origin.y = y
            default:
                break
            }
        })
    }
}
```

7. In the 'viewDidLoad' override, begin adding UISwipeGestureRecognizers for '.left', '.right', '.up', and '.down' to your main view (self.view). Have each one use the same action selector to the handler function created above.
8. Run your app and check for bugs.

Add the 'testPattern' UIImageView UIGestureRecognizers

9. Add the following gesture recognizer handler functions to ViewController.swift:

```
func pinchZoomGesture(_ gestureRecognizer: UIPinchGestureRecognizer) {
    if gestureRecognizer.state == .began {
        viewFrame = gestureRecognizer.view!.bounds
    }
    let scale = gestureRecognizer.scale
    let width = viewFrame.size.width * scale
    let height = viewFrame.size.height * scale
    gestureRecognizer.view!.bounds.size.width = width
    gestureRecognizer.view!.bounds.size.height = height
}
func panGesture(_ gestureRecognizer: UIPanGestureRecognizer) {
    if gestureRecognizer.state == .began {
        viewCenter = gestureRecognizer.view!.center
    }
    var movePoint = gestureRecognizer.translation(in: self.view)
    movePoint.x += viewCenter.x
    movePoint.y += viewCenter.y
    gestureRecognizer.view!.center = movePoint
}
func rotateGesture(_ gestureRecognizer: UIRotationGestureRecognizer) {
    if gestureRecognizer.state == .began {
        rads = acos(gestureRecognizer.view!.transform.a)
        let b = acos(gestureRecognizer.view!.transform.b)
        let c = acos(gestureRecognizer.view!.transform.c)
        if b > c {
            rads = CGFloat(2.0 * M_PI - Double(rads))
        }
    }
    let rotation = gestureRecognizer.rotation + rads
    let transform = CGAffineTransform(rotationAngle: rotation)
```

```
        gestureRecognizer.view!.transform = transform
    }
    func singleTapGesture(_ gestureRecognizer: UITapGestureRecognizer) {
        if gestureRecognizer.state == .ended {
            UIView.animate(withDuration: 0.75, animations: {
                gestureRecognizer.view!.center = self.view.center
            })
        }
    }
    func doubleTapGesture(_ gestureRecognizer: UITapGestureRecognizer) {
        if gestureRecognizer.state == .ended {
            UIView.animate(withDuration: 0.75, animations: {
                gestureRecognizer.view!.transform =
                               CGAffineTransform(rotationAngle: 0)
            })
        }
    }
    func longPressGesture(_ gestureRecognizer: UITapGestureRecognizer) {
        if gestureRecognizer.state == .began {
            UIView.animate(withDuration: 0.75, animations: {
                gestureRecognizer.view!.center = self.view.center
                gestureRecognizer.view!.transform =
                               CGAffineTransform(rotationAngle: 0)
                gestureRecognizer.view!.bounds.size.width = 150
                gestureRecognizer.view!.bounds.size.height = 150
            })
        }
    }
```

10. In the 'viewDidLoad' override, add the following gesture recognizers to the 'testPattern' UIImageView:
    a) UIPinchGestureRecognizer
    b) UIPanGestureRecognizer
    c) UIRotateGestureRecognizer
    d) UILongPressGestureRecognizer
    e) UITapGestureRecognizer for Single-Tap recognition
    f) UITapGestureRecognizer for Double-Tap recognition
11. Run your app and check for bugs.

Exploring Gestures

Optional Code Enhancements / Optimizations

1. Consider the following code fragment:

```
func addSwipeGesture(direction: UISwipeGestureRecognizerDirection) {
    let sgr = UISwipeGestureRecognizer(target: self,
                                       action: #selector(swipeGesture(_:)))
    sgr.direction = direction
    self.view.addGestureRecognizer(sgr)
}
```

   How could you use this code to optimize and simplify your implementation? Try modifying your code to take advantage of this code fragment.

2. Consider the following code fragments:

```
// Code Fragment #1
enum Gestures {
    case pinch
    case pan
    case rotate
    case longpress
    case tap
}

// Code Fragment #2
func addGeneralGesture(gesture: Gestures) -> UIGestureRecognizer {
    var gr: UIGestureRecognizer!
    switch gesture {
    case .pinch:
        gr = UIPinchGestureRecognizer(target: self,
                                  action: #selector(pinchZoomGesture(_:)))
    case .pan:
        gr = UIPanGestureRecognizer(target: self,
                                  action: #selector(panGesture(_:)))
    case .rotate:
        gr = UIRotationGestureRecognizer(target: self,
                                  action: #selector(rotateGesture(_:)))
    case .longpress:
        gr = UILongPressGestureRecognizer(target: self,
                                  action: #selector(longPressGesture(_:)))
    case .tap:
        gr = UITapGestureRecognizer(target: self,
                                  action: #selector(doubleTapGesture(_:)))
    }
    testPattern.addGestureRecognizer(gr)

    return gr
}

// Code Fragment #3
_ = addGeneralGesture(gesture: .pinch)
_ = addGeneralGesture(gesture: .pan)
```

```
_ = addGeneralGesture(gesture: .rotate)
_ = addGeneralGesture(gesture: .longpress)
(addGeneralGesture(gesture: .tap) as! UITapGestureRecognizer)
                            .numberOfTapsRequired = 1
(addGeneralGesture(gesture: .tap) as! UITapGestureRecognizer)
                            .numberOfTapsRequired = 2
```

How could you use this code to optimize and simplify your implementation? Try modifying your code to take advantage of this code fragment.

What is the purpose of the '_ =' in the 3rd code fragment? What happens if you just ignore the return value?

3. Are there any places on the screen where your various gesture recognizers do not work as expected? Can you explain why?