Sample Code

Supporting gesture interaction in your apps

Enrich your app's user experience by supporting standard and custom gesture interaction.

Download

iOS 13.0+ | iPadOS 13.0+ | Xcode 12.0+

Overview

Gesture interaction is one of the most intuitive user experiences on iOS, and gesture recognizers provide an easy way for apps to implement it. iOS defines some standard gestures, and apps can use system-provided recognizers to support these standard gestures, or create custom ones for non-standard gestures. This sample demonstrates how to use standard gesture recognizers by adding pan, pinch, and rotation support for three colored views, which this sample calls *pieces*, and how to implement a custom recognizer to reset the pieces to their initial state.

Set up a gesture recognizer

To support gesture interaction on a view, create an appropriate gesture recognizer and associate it with the view by calling the `addGestureRecognizer(_:)` method, as shown below.

```
let resetGestureRecognizer = ResetGestureRecognizer(target: self, action: #selector(
view.addGestureRecognizer(resetGestureRecognizer)
```

Xcode includes UIKit's standard gesture recognizers in its Objects library so developers can use them in storyboards. To associate a recognizer with a view in a storyboard:

1. Select the storyboard in Xcode to open the storyboard view.

2. Find the gesture recognizer in Xcode's Objects library.

3. Control-drag the recognizer onto the view.

With these steps, Xcode automatically adds the recognizer into the storyboard and associates it with the view. Developers can then add an action for the recognizer the same way they do for other UI elements.

Handle pan, pinch, and rotation gestures

This sample uses pan, pinch, and rotation gestures to move, scale, and rotate the views. When users pan a view, UIKit triggers the associated action method, which is `panPiece(_:)` in this sample, passing in a `UIPanGestureRecognizer` instance that carries a translation. The method then calculates and moves the view to the new position, and clears the translation by setting it to `.zero` so next time, its value is still the delta relative to the current position.

```
let translation = panGestureRecognizer.translation(in: piece.superview)
piece.center = CGPoint(x: piece.center.x + translation.x, y: piece.center.y + transl
panGestureRecognizer.setTranslation(.zero, in: piece.superview)
```

Similarly, when users pinch and rotate a view, this sample uses the `scale` and `rotation` properties of `UIPinchGestureRecognizer` and `UIRotationGestureRecognizer` to calculate and apply a new `transform`, and then clears the properties.

```
let scale = pinchGestureRecognizer.scale
piece.transform = piece.transform.scaledBy(x: scale, y: scale)
pinchGestureRecognizer.scale = 1 // Clear scale so that it is the right delta next t
```

```
piece.transform = piece.transform.rotated(by: rotationGestureRecognizer.rotation)
rotationGestureRecognizer.rotation = 0 // Clear rotation so that it is the right del
```

The `scale` and `rotation` properties are relative to the `anchorPoint` property of the view's `layer`. To be more intuitive when scaling or rotating a view, this sample moves the anchor point to the location of the gestures, which is usually the centroid of the touches involved in the gestures.

```
let locationInPiece = gestureRecognizer.location(in: piece)
let locationInSuperview = gestureRecognizer.location(in: piece.superview)
let anchorX = locationInPiece.x / piece.bounds.size.width
let anchorY = locationInPiece.y / piece.bounds.size.height
```

```
piece.layer.anchorPoint = CGPoint(x: anchorX, y: anchorY)
piece.center = locationInSuperview
```

Allow recognizing multiple gestures simultaneously

Users sometimes intuitively expect multiple gestures to work simultaneously, like pinching and rotating a view at the same time. This sample allows that by implementing the following `UIGestureRecognizerDelegate` method, which returns `true` to allow all gesture recognizers in this sample to work together.

```
func gestureRecognizer(_ gestureRecognizer: UIGestureRecognizer,
                       shouldRecognizeSimultaneouslyWith otherGestureRecognizer: UIG
    return true
}
```

For the delegate method to take effect, this sample sets the gesture recognizer's `delegate` property to `self`, which is the view controller that implements the method.

```
resetGestureRecognizer.delegate = self
```

For gesture recognizers that are in the storyboard, this sample sets up their `delegate` property by Control-dragging the gesture recognizers onto the view controller and selecting `delegate`.

`UIGestureRecognizerDelegate` provides methods for apps to control the order in which gestures are recognized as well. For more information on gesture recognition order, see Preferring one gesture over another.

Create a custom gesture recognizer

This sample creates a custom gesture recognizer, `ResetGestureRecognizer`, which recognizes a gesture that contains at least three horizontal turnings. Users can use the gesture to easily reset the views to their initial state. To do that, move the views away from their initial positions, then put one finger on the area the colored views don't cover, and move right, left, right, and left (or vice versa), like shaking the finger on the screen.

To recognize the gesture, this sample picks up a valid touch in `touchesBegan(_:with:)`, and gathers the touched locations in `touchesMoved(_:with:)`. When a user lifts their finger, which triggers `touchesEnded(_:with:)`, this sample counts the horizontal turnings in the touched path, and recognizes the gesture by setting the `state` property to `.ended` if the path has more than two turnings.

```
let count = countHorizontalTurning(touchedPoints: touchedPoints)
state = count > 2 ? .ended : .failed
```

Note that this sample doesn't go further to eliminate the location noise or validate the segments by checking their distance. Real-world apps might consider doing that to improve the gesture recognition accuracy and avoid recognizing false positive gestures.

For more details about custom gesture recognizers, see Implementing a custom gesture recognizer.

See Also

Standard gestures

Handling UIKit gestures

Use gesture recognizers to simplify touch handling and create a consistent user experience.

Coordinating multiple gesture recognizers

Discover how to use multiple gesture recognizers on the same view.

{} Adopting hover support for Apple Pencil

Enhance user feedback for your iPadOS app with a hover preview for Apple Pencil input.

`class UIHoverGestureRecognizer`

A continuous gesture recognizer that interprets pointer movement over a view.

`class UILongPressGestureRecognizer`

A continuous gesture recognizer that interprets long-press gestures.

`class UIPanGestureRecognizer`

A continuous gesture recognizer that interprets panning gestures.

`class UIPinchGestureRecognizer`

A continuous gesture recognizer that interprets pinching gestures involving two touches.

`class UIRotationGestureRecognizer`

A continuous gesture recognizer that interprets rotation gestures involving two touches.

`class UIScreenEdgePanGestureRecognizer`

A continuous gesture recognizer that interprets panning gestures that start near an edge of the screen.

`class` `UISwipeGestureRecognizer`

A discrete gesture recognizer that interprets swiping gestures in one or more directions.

`class` `UITapGestureRecognizer`

A discrete gesture recognizer that interprets single or multiple taps.