Coordinating Multiple Gesture Recognizers

On This Page

Gesture recognizers track incoming touch events separately, but UIKit normally allows the recognition of only one gesture at a time on a single view. Recognizing only one gesture at a time is usually preferable because it prevents user input from triggering more than one action at a time. However, this default behavior can introduce unintended side effects. For example, in a view that contains both pan and swipe gesture recognizers, swipes are never recognized. Because the pan gesture recognizer is continuous, it always recognizes its gesture before the swipe gesture recognizer, which is discrete.

To prevent the unintended side effects of the default recognition behavior, you can tell UIKit to recognize gestures in a specific order using a delegate object. UIKit uses the methods of your delegate object to determine whether a gesture recognizer must come before or after other gesture recognizers. For example, your delegate can tell UIKit that a swipe gesture recognizer must fail before a pan gesture recognizer is allowed to act. Your delegate can also tell UIKit that two gestures can be recognized simultaneously.

Preferring One Gesture Over Another

Use a gesture recognizer delegate object to determine the order in which gestures are recognized in your views. For any two gesture recognizers involved in a potential conflict, only one needs an associated delegate object, and that object must conform to the `UIGestureRecognizerDelegate` protocol. In your delegate, implement the methods you need for the appropriate resolution. The best way to see how to use these methods is with a few examples.

Listing 6-1 shows how to recognize tap and double-tap gestures in the same view. The view in this example has two `UITapGestureRecognizer` objects, one of which is configured to require two taps. Normally, the single-tap gesture would always be recognized before the double-tap gesture, but you can use the `gestureRecognizer:shouldRequireFailureOfGestureRecognizer:` method to reverse that behavior. The implementation of this method prevents the single-tap gesture from being recognized until after the double-tap gesture recognizer explicitly reaches the failed state, which happens when the touch sequence contains only one tap.

Listing 6-1 Recognizing a single tap only after the failure of a double tap

```
func gestureRecognizer(_ gestureRecognizer: UIGestureRecognizer,
                       shouldRequireFailureOf otherGestureRecognizer:
  UIGestureRecognizer) -> Bool {
    // Don't recognize a single tap until a double-tap fails.
    if gestureRecognizer == self.tapGesture &&
        otherGestureRecognizer == self.doubleTapGesture {
        return true
    }
    return false
}
```

Listing 6-2 shows how to recognize a swipe gesture before a pan gesture. In this case, the delegate object is attached to the swipe gesture recognizer and implements the `gestureRecognizer:shouldBeRequiredToFailByGestureRecognizer:` method. The logic of this method prevents the pan gesture from being recognized until the swipe gesture fails.

Listing 6-2 Delaying the recognition of a pan gesture until after a swipe fails

```
func gestureRecognizer(_ gestureRecognizer: UIGestureRecognizer,
                       shouldBeRequiredToFailBy otherGestureRecognizer:
  UIGestureRecognizer) -> Bool {
    // Do not begin the pan until the swipe fails.
    if gestureRecognizer == self.swipeGesture &&
        otherGestureRecognizer == self.panGesture {
        return true
    }
    return false
}
```

Listing 6-3 shows an alternative way to configure the dependency between a swipe and a pan gesture. Instead of attaching a delegate object to the swipe gesture, this example attaches the delegate to the pan

gesture. Because of the change, the delegate must now implement the `gestureRecognizer:shouldRequireFailureOfGestureRecognizer:` method and require the swipe to fail. The end results are the same as those from Listing 6-2.

Listing 6-3 An alterr

On This Page

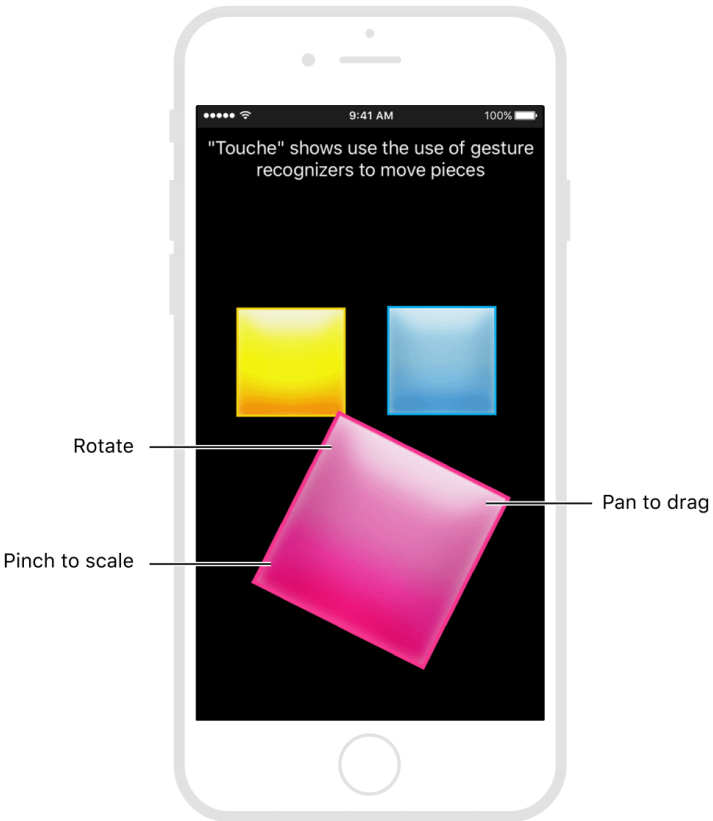
```
func gestureRecognizer(_ gestureRecognizer: UIGestureRecognizer,
                       shouldRequireFailureOf otherGestureRecognizer:
  UIGestureRecognizer) -> Bool {
    if gestureRecognizer == self.panGesture &&
        otherGestureRecognizer == self.swipeGesture {
        return true
    }
    return false
}
```

The ability to attach a delegate to any gesture recognizer lets you create complex dependency chains among your gestures. For more information about regulating gesture recognition, see *UIGestureRecognizerDelegate Protocol Reference*.

Allowing the Simultaneous Recognition of Multiple Gestures

There are times when it makes sense to allow the simultaneous recognition of multiple gestures. Figure 6-1 shows an app that allows users to drag, scale, and rotate three colored views onscreen. Each view maintains its own unique set of pan, pinch, and rotate gesture recognizers, and it is possible for all three of a view's gesture recognizers to perform their actions simultaneously.

Figure 6-1 Dragging, scaling, and rotating an object



To allow a gesture recognizer to operate simultaneously with other gestures, assign a delegate object that implements the `gestureRecognizer:shouldRecognizeSimultaneouslyWithGestureRecognizer:` method to it. UIKit calls this method for pairs of gesture recognizers attached to the same view. Returning true allows both gestures to process events simultaneously.

Listing 6-4 shows the `gestureRecognizer:shouldRecognizeSimultaneouslyWithGestureRecognizer:` method from the app shown in Figure 6-1. This method returns true when the gesture recognizers are attached to the

same view. If the gesture recognizers are attached to different views, or if one of the objects is a long press gesture recognizer, this method returns false.

Listing 6-4 Determining when to handle gestures simultaneously

```
func gestureRecognizer(_ gestureRecognizer: UIGestureRecognizer,
                       shouldRecognizeSimultaneouslyWith otherGestureRecognizer:
  UIGestureRecognizer)
    -> Bool {
        // If the gesture recognizer's view isn't one of the squares, do not
        // allow simultaneous recognition.
        if gestureRecognizer.view != self.yellowView &&
            gestureRecognizer.view != self.cyanView &&
            gestureRecognizer.view != self.magentaView {
            return false
        }

        // If the gesture recognizers are on diferent views, do not allow
        // simultaneous recognition.
        if gestureRecognizer.view != otherGestureRecognizer.view {
            return false
        }

        // If either gesture recognizer is a long press, do not allow
        // simultaneous recognition.
        if gestureRecognizer is UILongPressGestureRecognizer ||
            otherGestureRecognizer is UILongPressGestureRecognizer {
            return false
        }

        return true
}
```

On This Page

For more information about the sample, see *Handling Touches Using Responder Methods and Gesture Recognizers*.

Attaching Gesture Recognizers to UIKit Controls

Gesture recognizers attached to your views do not impact the ability of UIKit controls to handle events. Events occurring within the bounds of a control are handled by the control first, giving the control a chance to call its action method. Specifically, UIKit controls call their action method in the following situations:

- A single finger single tap occurs on a `UIButton`, `UISwitch`, `UIStepper`, `UISegmentedControl`, or `UIPageControl` object.
- A single finger swipe occurs on the knob of a `UISlider` object, in a direction parallel to the slider.
- A single finger pan occurs on the knob of a `UISwitch` object, in a direction parallel to the switch.

To handle any of the preceding gestures before the control calls its action method, install your gesture recognizer on the control itself. Gesture recognizers handle touch events before the views to which they are attached. As a result, installing a gesture recognizer directly on a control prevents that control from calling its action method.

> IMPORTANT
>
> Always consult the platform-specific human interface guidelines before changing the default behavior of standard controls. For more information, see *iOS Human Interface Guidelines* or *Apple TV Human Interface Guidelines*.