

Praktikum iOS-Entwicklung

Wintersemester 2015/16

Prof. Dr. Linnhoff-Popien

Florian Dorfmeister, Marco Maier



Event Handling – Interaktionen mit dem Display

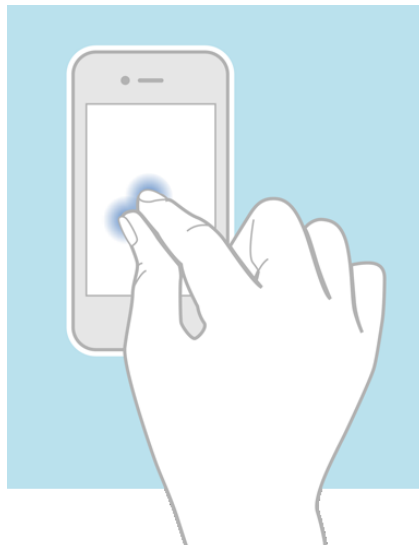
- Touch-Gesten
- Touch-Events
- Selektoren
- Protokolle
- Responder Chain

EVENT HANDLING – INTERAKTIONEN MIT DEM DISPLAY

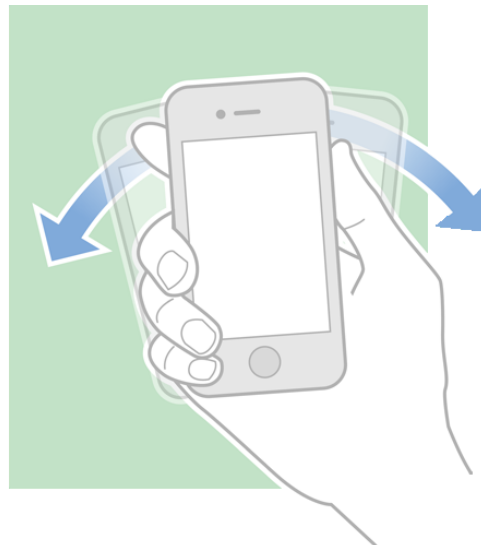
Der Nutzer kann durch Interaktion mit seinem Gerät verschiedene Events in iOS auslösen

Es gibt

- (Multi-)Touch-Events
- Motion-Events
- Remote-Control-Events (zur Bedienung von Multimedia-Komponenten)



Multitouch events



Accelerometer events



Remote control events

Quelle: <http://www.apple.com/>

Events sind Objekte, die an eine Anwendung gesendet werden, um sie über Interaktionen eines Nutzers zu informieren

Events, die durch den Nutzer ausgelöst werden, sind Instanzen der Klasse **UIEvent**

- enthalten Informationen, um auf das Event entsprechend zu reagieren
- haben einen Typ, z.B.
 - touch
 - shaking motion
 - remote control
- und einen Subtyp, z.B. für remote control
 - play
 - pause
 - stop
 - ...

TOUCH-GESTEN

Behandlung einzelner Berührungen der Finger mit dem Display (**Touch-Events**) ist in vielen Fällen viel zu aufwendig

Viele Interaktionen erfolgen nach dem gleichen Schema (Gesten)

- Tippen (Tap)
- Wischen (Swipe)
- Klammern (Pinch)
- Lange Drücken (LongPress)
- ...

→ Verwendung einer "Higher-Level-API" zur Behandlung von Gesten

Geste = Kombinationen von Touch-Events

Intuitive Gesten werden von iOS über "Gesture Recognition" direkt erkannt

- Tippen ([UITapGestureRecognizer](#))
- Pinchen ([UIPinchGestureRecognizer](#))
- Wischen ([UISwipeGestureRecognizer](#))
- Ziehen/Verschieben ([UIPanGestureRecognizer](#))
- ...

Für viele Gesten gibt es in Kombination mit typischen UI-Elementen eine direkte Integration in [UIKit](#):

- Beispiel: [UIScrollView](#)
 - Pinch-Geste → Zoom
 - Wisch-Geste → Scroll

Gesture Recognizer (**UIGestureRecognizer**)

- Dienen der Abstraktion von der komplexen Event-Handling-Logik
- Stellen den bevorzugten Weg zur Behandlung von Touch-Events dar

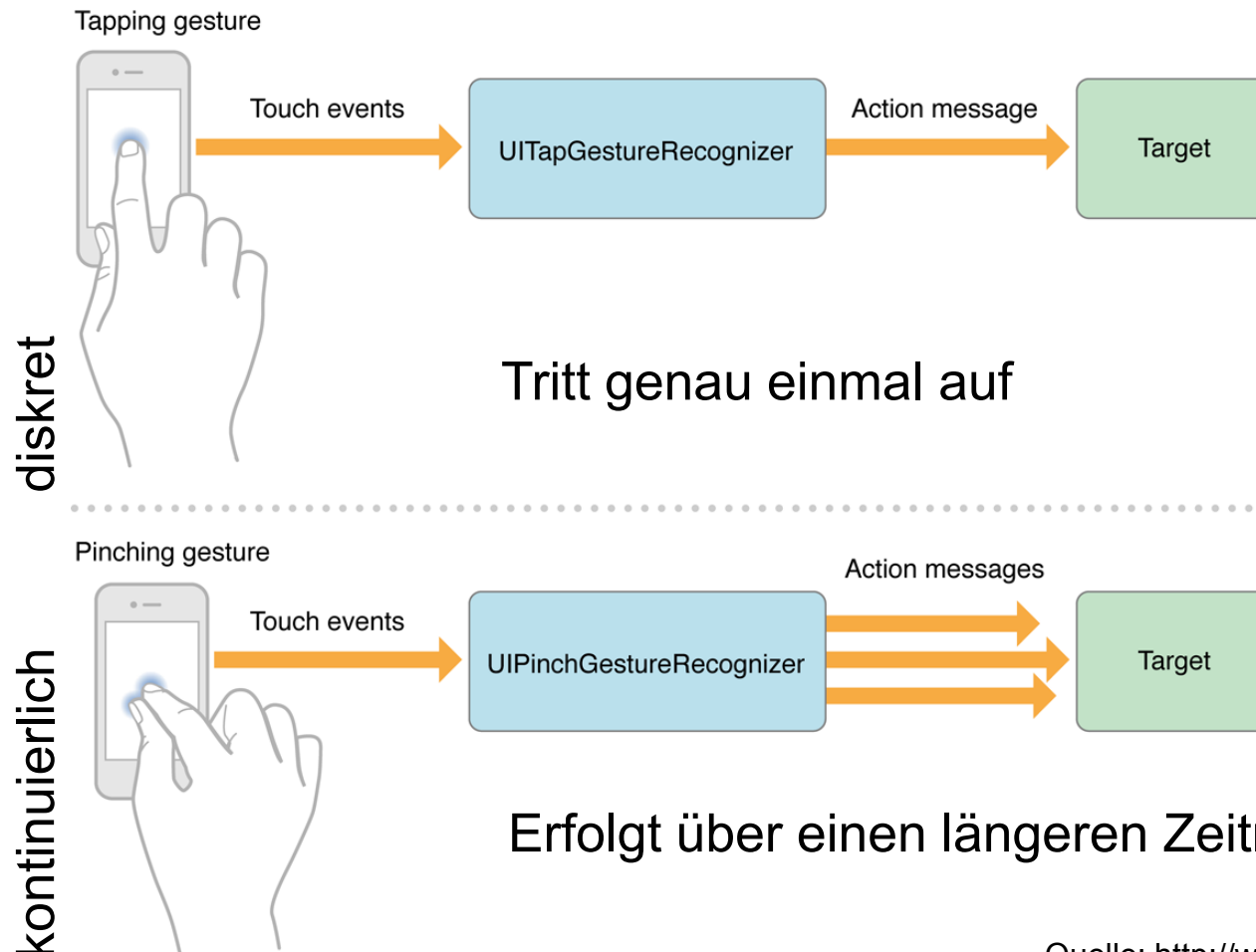
Implementieren eigener Action-Target-Mechanismen:

- Instanziieren eines Gesture Recognizers
- Hinzufügen des Gesture Recognizer zur eigenen View
 - Gesamte View reagiert auf die hinzugefügte Geste

Vorgehensweise:

- Verwendung und Anpassung von **Built-In** Gesture Recognizern, oder
- Implementierung eigener (**Custom**) Gesture Recognizer

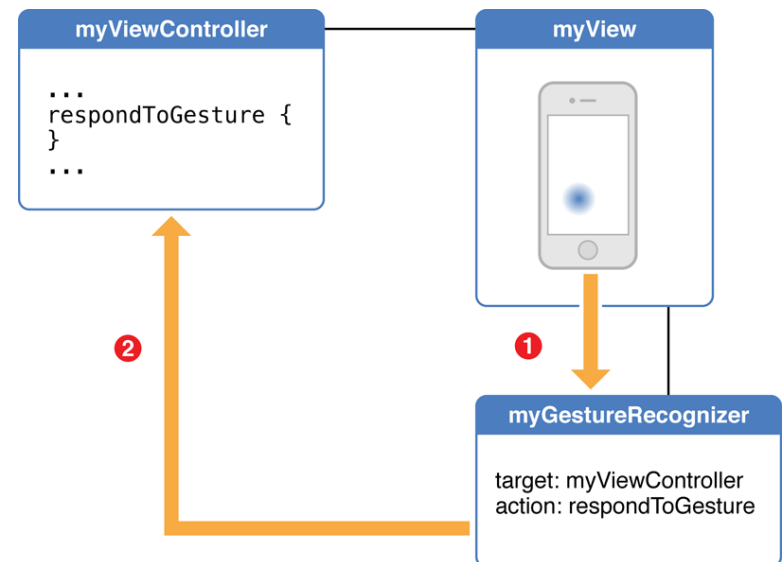
Es existieren **diskrete** und **kontinuierliche** Gesten



Quelle: <http://www.apple.com/>

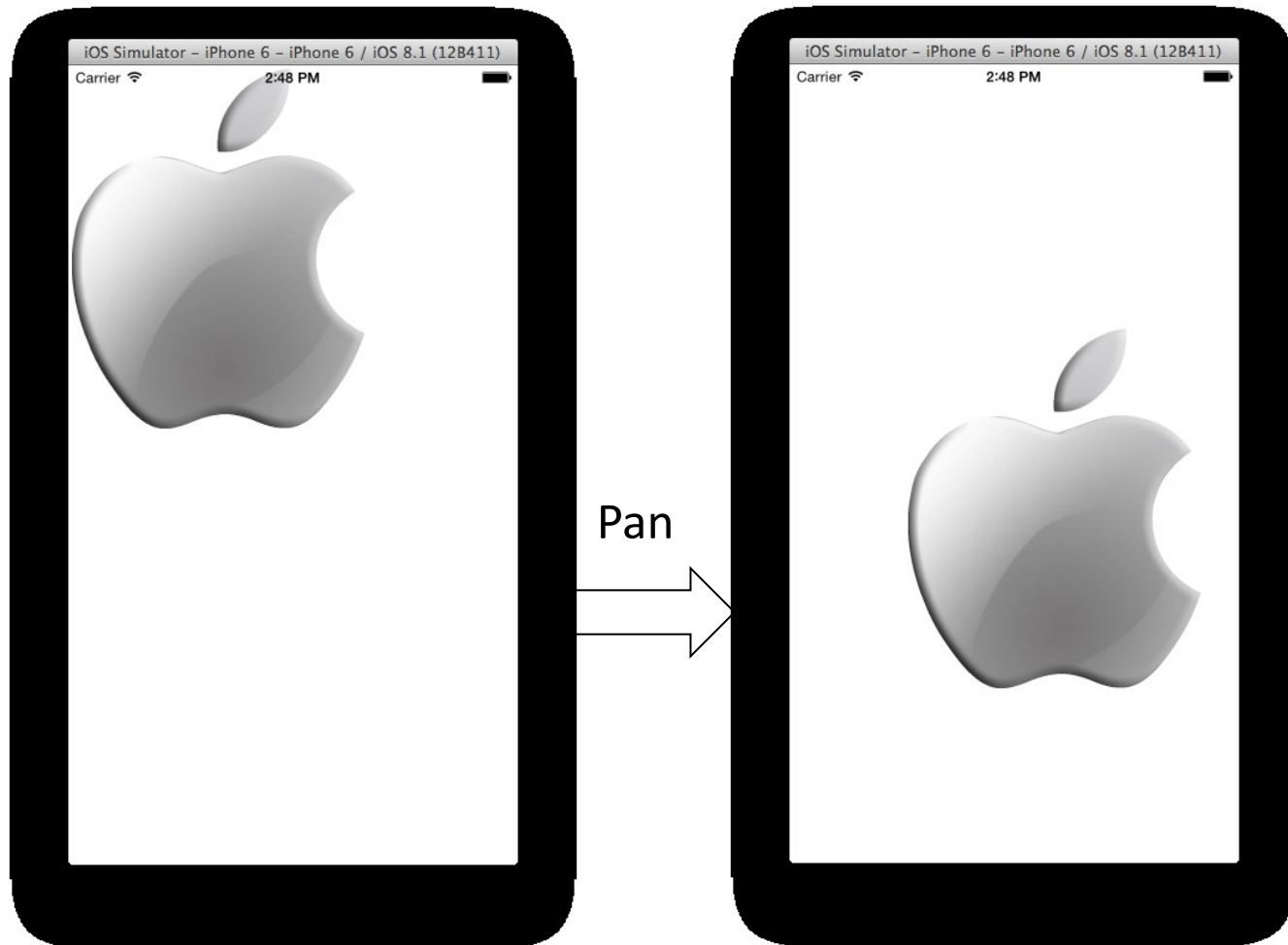
Hinzufügen eines Built-In Gesture Recognizers

- Erzeugen und konfigurieren einer Instanz von **UIGestureRecognizer**
 - Zuweisen eines Targets und einer Action
 - Zuweisen gestenspezifischer Attribute (optional; z.B. **numberOfTapsRequired**)
- Registrieren des Gesture Recognizers mit einer View
- Implementieren der Action-Methode



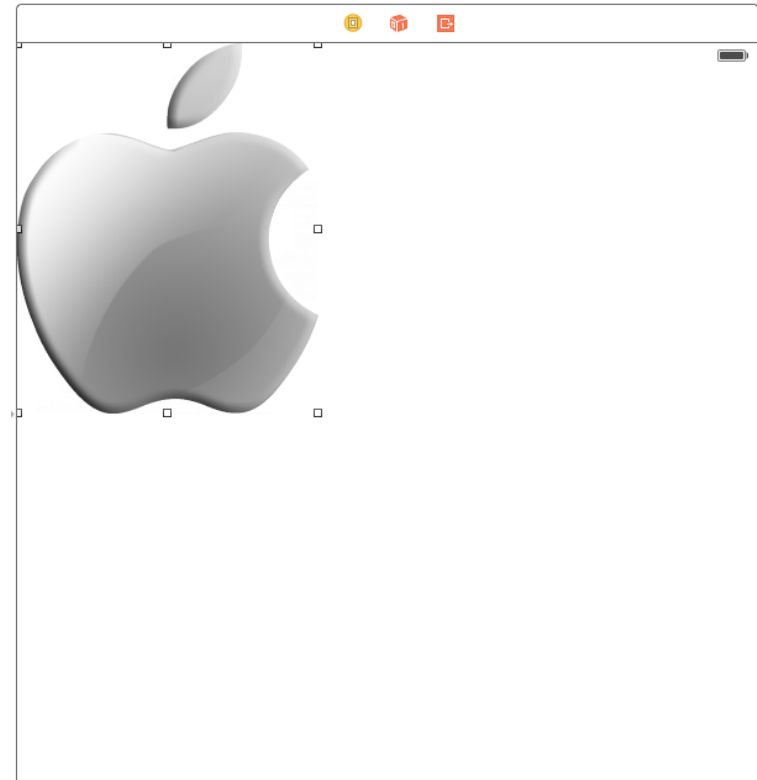
Quelle: <http://www.apple.com/>

Beispiel: ImageTranslationRecognition

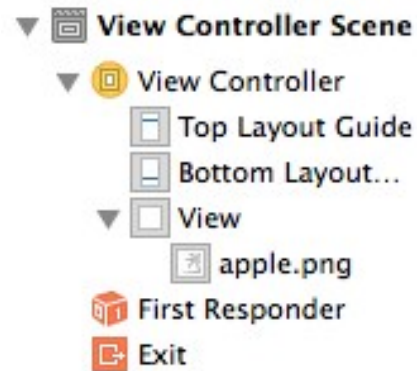


Vorgehensweise

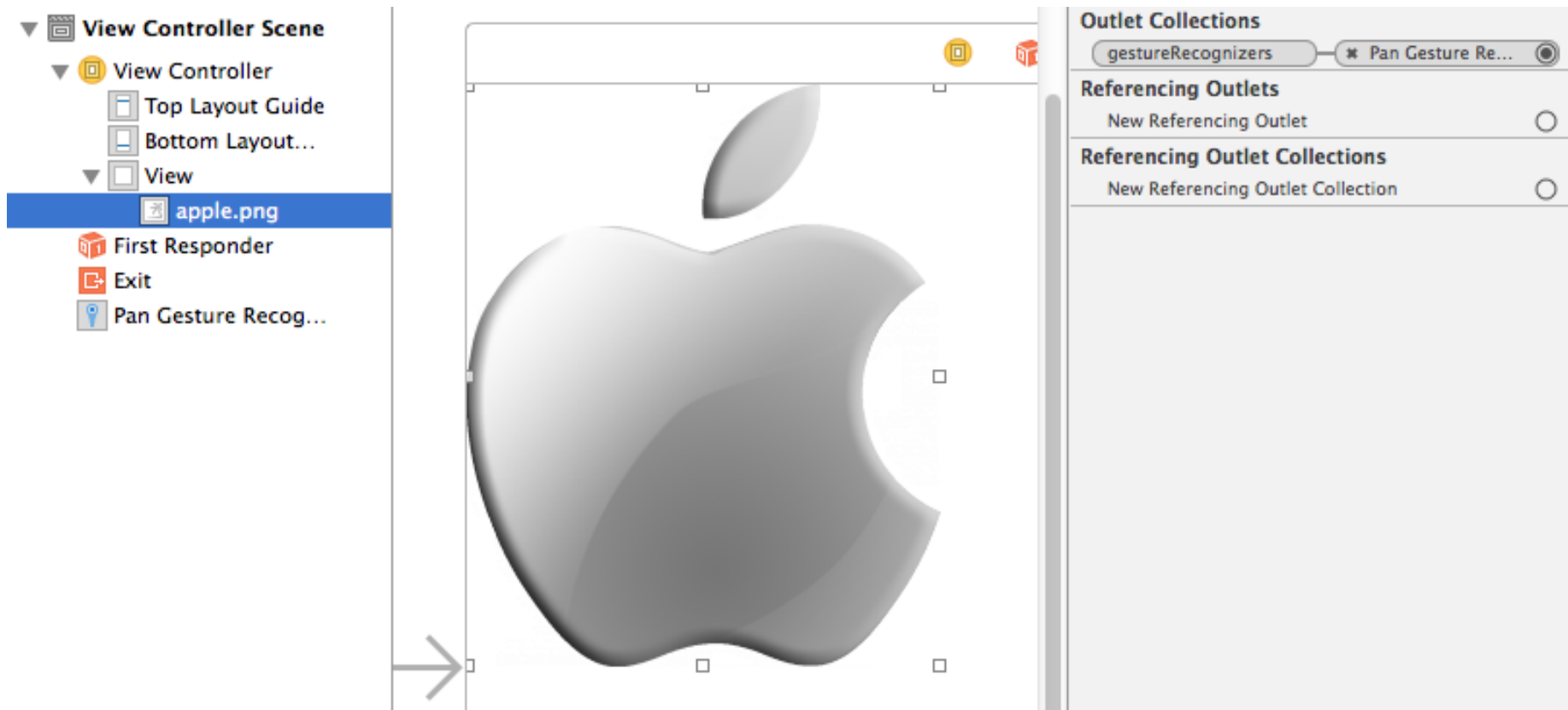
- Erzeugen eines neuen Projekts
"ImageTranslationRecognition" (Template:
Single View Application)
- Im Storyboard:
 - Hinzufügen einer Image View vom Typ
`UIImageView`
- Im Project Navigator:
 - Hinzufügen eines Image (Drag & Drop vom
Finder)
- Unter „Editor“:
 - Size to fit content: Anpassen der Größe der
Image View an das Image



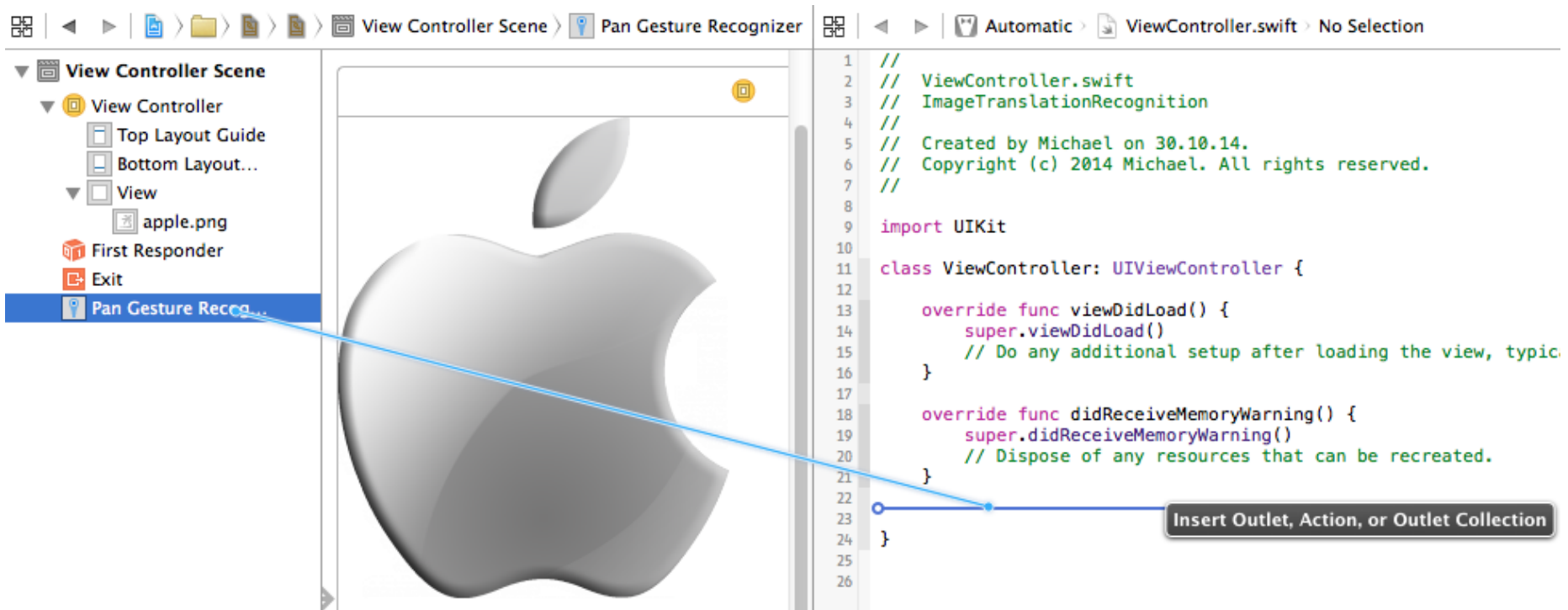
- Von der Objekt Library:
 - Hinzufügen eines Pan Gesture Recognizers vom Typ `UIPanGestureRecognizer` zur Image View (Drag & Drop auf die View)



- Pan Gesture Recognizer ist jetzt als Outlet bei der Image View installiert



■ Erzeugen und Verbinden einer Action-Methode



The screenshot shows the Xcode interface with the 'View Controller Scene' on the left and the 'ViewController.swift' file on the right. The 'Pan Gesture Recognizer' is selected in the left sidebar. A blue line connects the 'Pan Gesture Recognizer' to the 'Insert Outlet, Action, or Outlet Collection' button in the right sidebar. The Swift code in the right pane shows the following structure:

```

1 //
2 // ViewController.swift
3 // ImageTranslationRecognition
4 //
5 // Created by Michael on 30.10.14.
6 // Copyright (c) 2014 Michael. All rights reserved.
7 //
8
9 import UIKit
10
11 class ViewController: UIViewController {
12
13     override func viewDidLoad() {
14         super.viewDidLoad()
15         // Do any additional setup after loading the view, typic
16     }
17
18     override func didReceiveMemoryWarning() {
19         super.didReceiveMemoryWarning()
20         // Dispose of any resources that can be recreated.
21     }
22
23 }
24
25
26

```

- Implementierung der Action-Methode

```
// ViewController.swift

import UIKit

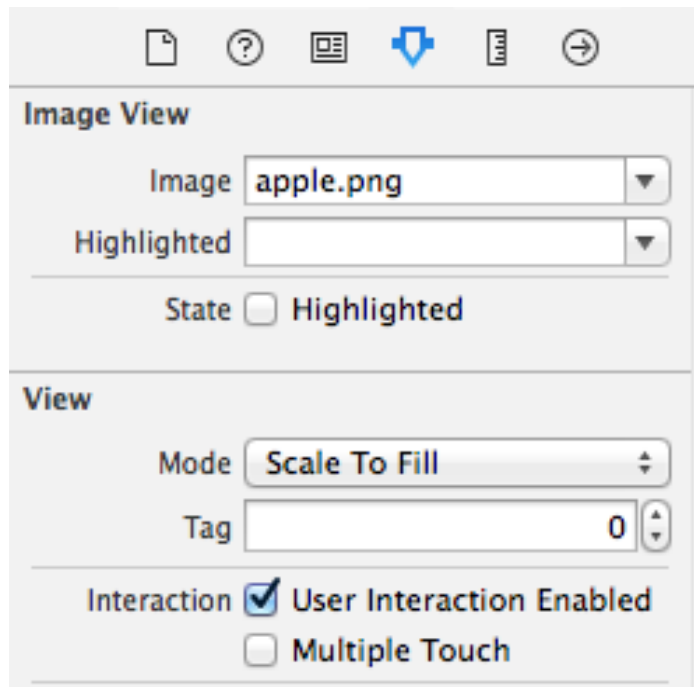
class ViewController: UIViewController {

    @IBAction func handlePan(sender: UIPanGestureRecognizer) {
        // Die Translation liefert hier den absoluten Abstand
        // zu dem Punkt, an dem der Finger auf den Bildschirm traf
        // bezogen auf das Koordinatensystem der übergebenen View
        let translation:CGPoint = sender.translationInView(self.view)

        // Anpassen der Position des Images
        sender.view!.center = CGPoint(x:sender.view!.center.x + translation.x,
                                       y:sender.view!.center.y + translation.y)

        // Zurücksetzen der Translation, um nur relative Positionsveränderungen
        // zur Position im vorhergehenden Aufruf der Action-Methode zu erhalten
        sender.setTranslation(CGPointZero, inView: self.view)
    }
}
```

Im Attributes Inspector: Erlauben von Nutzerinteraktionen mit der Image View



Allgemeine Anmerkungen

- Der Aufruf der Action-Methode erfolgt jedes Mal, wenn die Geste in der entsprechenden View erkannt wird
- Das Referenzieren eines Gesture Recognizer außerhalb der Action-Methode ist über eine Outlet-Connection möglich

Anmerkungen zum [UIPinchGestureRecognizer](#)

- Wenn man den Skalierungsfaktor nicht selbst verändert, wird er absolut berechnet
 - s Skalierungsfaktor
 - d_{Start} Abstand der Finger bei erster Berührung
 - d_{Now} Gegenwärtiger Abstand der Finger

$$\rightarrow s = d_{\text{Now}} / d_{\text{Start}}$$

Initialisierung mit `initWithTarget:action:`

- `target:`
 - Target-Objekt, das in einer Action-Methode auf eine Geste reagiert
 - I.d.R. der zugehörige View Controller
- `action:`
 - Ein **Selektor**
 - Spezifiziert die Action-Methode, die beim Auftreten der Geste ausgeführt werden soll

Registrieren des Gesture Recognizers mit `addGestureRecognizer:` des entsprechenden `UIView`-Objekts

Programmatische Erzeugung erfolgt i.d.R. in `viewDidLoad`

Beispiel:

- die Action-Methode wird als **Selektor** spezifiziert

```
-(void)viewDidLoad {  
    [super viewDidLoad];  
  
    // Erzeugen und Initialisieren einer Tipgeste  
    UITapGestureRecognizer *tapRecognizer =  
        [[UITapGestureRecognizer alloc] initWithTarget:self  
        action:@selector(respondToTapGesture:)];  
  
    // Nur einzelnes Tippen wird als Geste erkannt  
    tapRecognizer.numberOfTapsRequired = 1;  
  
    // Hinzufügen des Gesture Recognizers zur eigenen View  
    [self.view addGestureRecognizer:tapRecognizer];  
  
    // Weitere Initialisierungsschritte  
}
```

Ein Selektor

- identifiziert eine Methode
- ist der Name, der verwendet wird, um eine Methode zu referenzieren, die auf einem Objekt aufgerufen werden soll, bzw.
- ist ein **eindeutiger Bezeichner**, der den Namen einer Methode beim Übersetzen des Quellcodes ersetzt
- entspricht einem dynamischen Zeiger, der – unabhängig von der dazugehörigen Klasse – automatisch die Implementierung einer passenden Methode referenziert

Selektoren besitzen den Typ `SEL`

Es gibt zwei Wege, Selektoren zu referenzieren

- Zur Übersetzungszeit
 - Verwenden der Direktive `@selector`
 - `SEL mySel = @selector(methodName)`
- Zur Laufzeit (wenn Name der Methode zur Übersetzungszeit noch nicht bekannt)
 - Verwenden der Funktion `NSStringFromClass`
 - Der String-Parameter entspricht dem Namen der aufzurufenden Methode
 - `SEL mySel = NSStringFromClass(@"methodName");`

Aufruf einer Methode über einen Selektor mit den Methoden
(aus `NSObject`-Klasse und -Protocol)

- `performSelector:`
- `performSelector:withObject:`
- `performSelector:withObject:withObject:`
- `performSelector:withObject:afterDelay:`
- Bei mehr/anderen Parametern: `NSInvocation`

Beispiel:

```
SEL mySelector = @selector(accelerate);  
  
[myBoat performSelector:mySelector];  
[myCar performSelector:mySelector];
```

Objective-C Verwendet Message Passing!

- Methodenaufrufe über Selektoren sind unsicher!
- Man kann vor dem Aufruf eines Selektors überprüfen, ob das Callee-Objekt eine entsprechende Methode besitzt!

```
CustomClass *myObject = [CustomClass new];
SEL mySel = NSSelectorFromString(@"methodName");

if ([myObject respondsToSelector:@selector(mySel)]) {
    [myObject performSelector:mySel];
}
else {
    // Fehler behandeln...
}
```

In Swift werden Selektoren durch die Struktur **Selector** repräsentiert

- Erzeugung erfolgt mit Hilfe von String Literalen

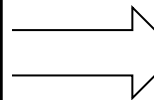
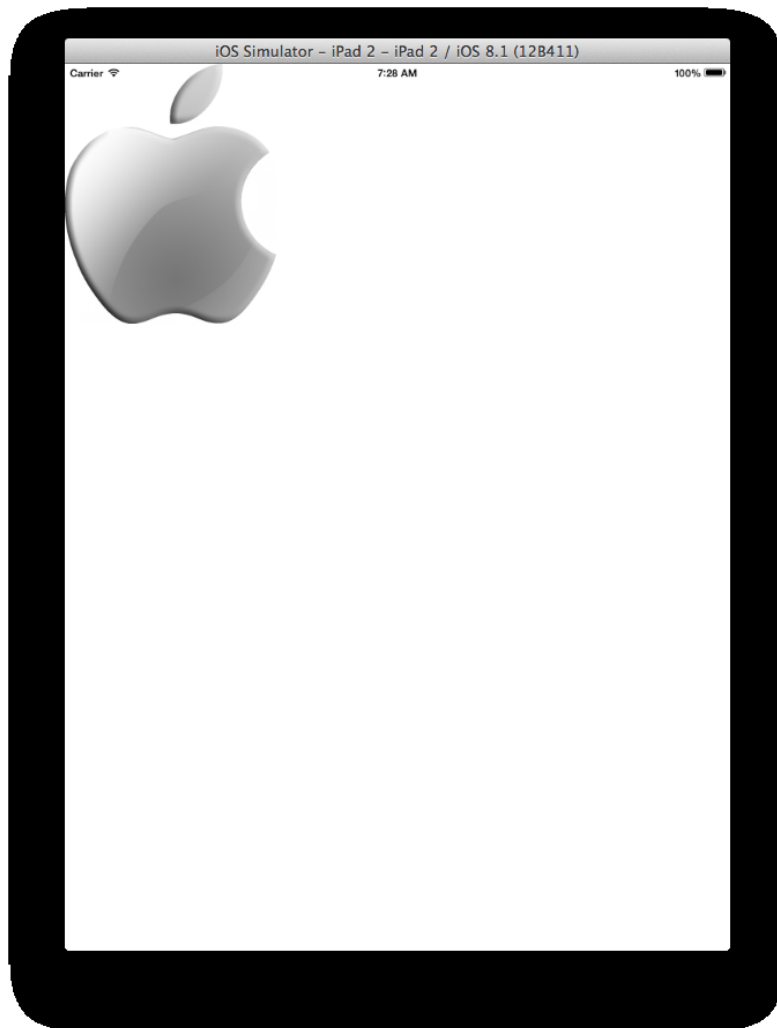
```
let mySel: Selector = "methodName:"
```

- String Literale lassen sich automatisch in Selektoren konvertieren
→ Literale können direkt als Argument übergeben werden, wenn ein Selektor erwartet wird

```
button.addTarget(  
    self, action: "buttonPressed:", forControlEvents: .TouchUpInside)
```

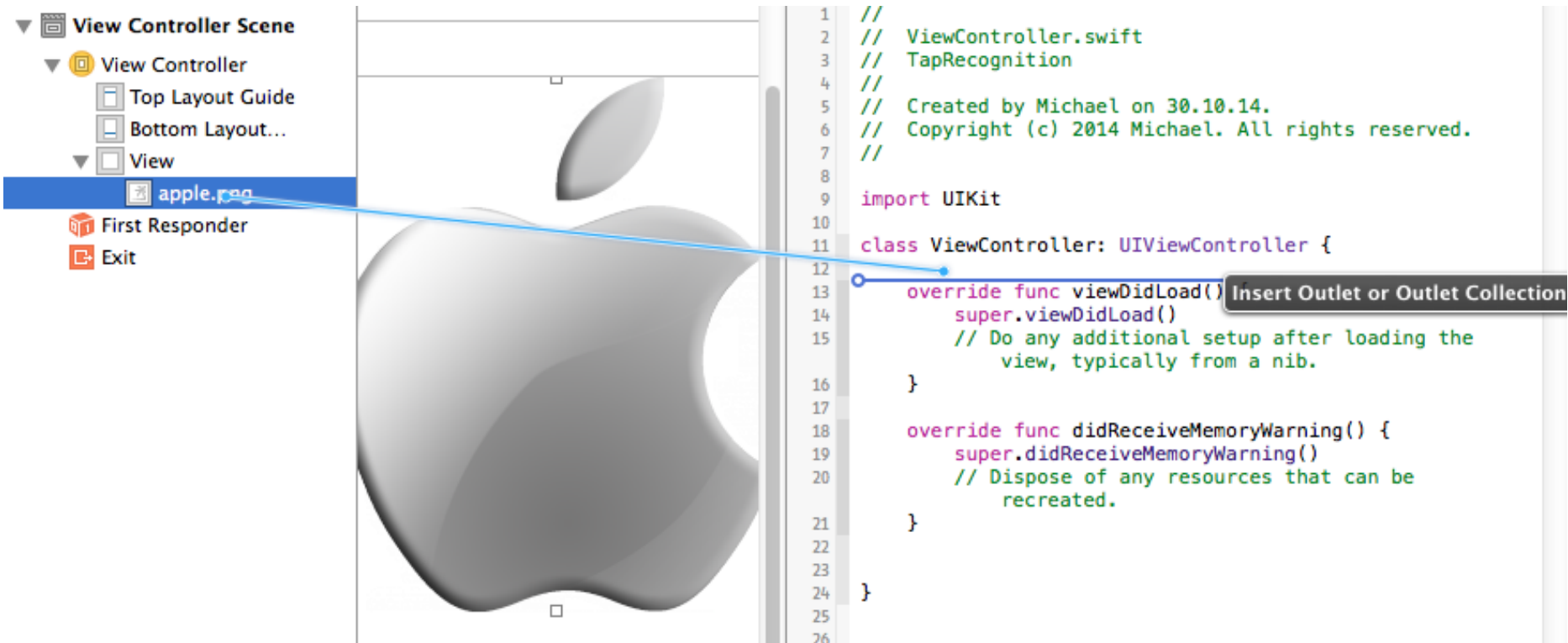
- **Achtung:** Methoden zum Aufruf von Selektoren wie **performSelector:** oder **respondsToSelector:** sind inhärent unsicher
→ Swift bietet diese Methoden nicht an (stattdessen Verwendung von Optionals)

Tippen auf den Bildschirm bewirkt das Verschieben eines Images an die angetippte Stelle



Vorgehensweise

- Analog zum Beispiel ImageTranslationRecognition
- Erzeugen einer Outlet-Connection zur Image View



The screenshot shows the Xcode interface. On the left, the 'View Controller Scene' hierarchy is visible, with 'apple.png' selected under the 'View' category. A blue line connects this asset to the 'viewDidLoad()' method in the 'ViewController.swift' file on the right. A tooltip 'Insert Outlet or Outlet Collection' is displayed near the connection point.

```
1 //
2 // ViewController.swift
3 // TapRecognition
4 //
5 // Created by Michael on 30.10.14.
6 // Copyright (c) 2014 Michael. All rights reserved.
7 //
8
9 import UIKit
10
11 class ViewController: UIViewController {
12
13     override func viewDidLoad() {
14         super.viewDidLoad()
15         // Do any additional setup after loading the
16         // view, typically from a nib.
17     }
18
19     override func didReceiveMemoryWarning() {
20         super.didReceiveMemoryWarning()
21         // Dispose of any resources that can be
22         // recreated.
23     }
24 }
25
26 }
```

Programmatisches Hinzufügen eines **UITapGestureRecognizer** (geht natürlich auch über Interface-Builder...)

```
// ViewController.swift

import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var imageView: UIImageView!

    override func viewDidLoad() {
        // Initialisieren eines UITapGestureRecognizer
        let tgr = UITapGestureRecognizer(target: self, action:handleTap)

        // Registrieren mit der eigenen View
        self.view.addGestureRecognizer(tgr)
    }
}
```

Implementierung der Action-Methode

- Achtung: Nur zwei Signaturen erlaubt:
 - `func handleTap() -> Void`
 - `func handleTap(tgr:UITapGestureRecognizer) -> Void`

```
func handleTap(tgr:UITapGestureRecognizer) -> Void {  
    // Position wo das Event auftrat  
    let location = tgr.locationInView(self.view)  
  
    // Verschieben des Images an die neue Position  
    self.imageView.center = location;  
  
}
```

Eine Instanz von **UIGestureRecognizer** gehört **zu genau einer UIView**-Instanz

- Hinzufügen desselben Gesture Recognizers zu verschiedenen **UIView**-Instanzen nicht möglich!
- Ein wiederholtes Hinzufügen überschreibt das vorhergehende!
- Beispiel:
 - Nur **view2** erhält die Nachricht **tapTapTap**

```
// MyController.m
[...]
```

```
UITapGestureRecognizer *tapGesture =
    [[UITapGestureRecognizer alloc] initWithTarget:self
    action:@selector(tapTapTap:)];
```

```
[self.view1 addGestureRecognizer:tapGesture];
[self.view2 addGestureRecognizer:tapGesture];
```

```
[...]
```

UIGestureRecognizer unterbricht standardmäßig Behandlung von Touch-Events, wenn eine Geste erkannt wurde

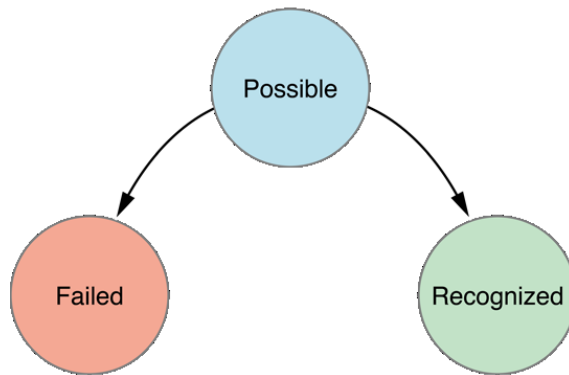
Problem:

- **UIView** mit einer Gestenerkennung erhält möglicherweise nicht alle Touch-Events.

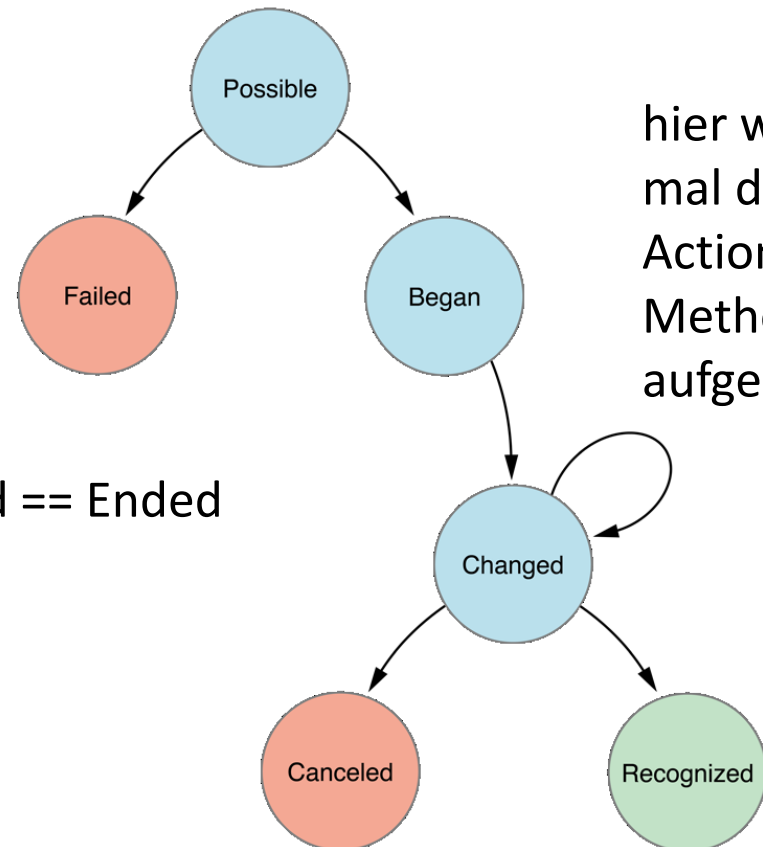
Gesture Recognizer operieren in einer Zustandsmaschine

- mehr dazu im *"Event Handling Guide for iOS"*

State transitions for discrete gestures



State transitions for continuous gestures



hier wird jedes
mal die
Action-
Methode
aufgerufen

Achtung: Recognized == Ended

Quelle: <http://www.apple.com/>

Testen des gegenwärtigen Zustands über die Konstanten

- **UIGestureRecognizerStatePossible** (Default State)
 - Es wurden eventuell Touch-Events registriert, aber noch keine Geste erkannt
- **UIGestureRecognizerStateBegan**
 - Es wurde ein Touch-Event empfangen und eine **kontinuierliche** Geste erkannt
→ Löst eine Action-Nachricht aus
- **UIGestureRecognizerStateChanged**
 - Es wurden Touch-Events erkannt, die eine Veränderung in der **kontinuierlichen** Geste signalisieren
→ Löst eine Action-Nachricht aus
- **UIGestureRecognizerStateEnded**
 - Es wurden Touch-Events erkannt, die das Ende einer (**kontinuierlichen**) Geste signalisieren
→ Löst eine Action-Nachricht aus
→ Reset auf **UIGestureRecognizerStatePossible**

Testen des gegenwärtigen Zustands über die Konstanten

- `UIGestureRecognizerStateFailed`

- Es wurde eine Sequenz von Touch-Events empfangen, die aber nicht als Teil der gegenwärtigen Geste erkannt werden konnten
 - **Keine** Action-Nachricht
 - Reset auf `UIGestureRecognizerStatePossible`

- `UIGestureRecognizerStateRecognized`

- Es wurden eine Sequenz von Touch-Events als Geste erkannt
 - Löst eine Action-Nachricht aus
 - Reset auf `UIGestureRecognizerStatePossible`

Hinweise:

- Die Konstanten sind als `enum` realisiert.
- In Swift erfolgt der Zugriff jedoch über die Member Values (z.B. `UIGestureRecognizerState.Began`)
- In Swift existiert keine Konstante `.Recognized`

Beispiel: Drag & Drop:

- Realisierung normalerweise mit LongPress und Pan auf ein UI-Objekt.
- LongPress zum Starten des Drag-Vorgangs, Pan zum Bewegen
- Die gesamte Pan-Geste **liegt vollständig innerhalb** der LongPress-Geste!

Problem:

- Standardmäßig wird das parallele Erkennen mehrerer Gesten unterdrückt
- Hier muss aber Pan während eines LongPress erkannt werden
- LongPress gibt Touch-Events nicht weiter
→ Pan kann nicht erkannt werden!

Lösung:

- Implementierung des [UIGestureRecognizerDelegate](#) Protokolls

Die Klasse eines Objekts unterscheidet sich häufig von dessen **Rolle** in einem System.

Beispiel

- Die Klasse eines Objekts ist `NSMutableArray` aber seine Rolle in der Anwendung ist eine Warteschleife (für Druckaufträge).

Spezifikation von Rollen erfolgt in Objective-C / Swift über Protokolle (ähnlich zu Interfaces in Java).

Protokolle

- stellen eine Alternative zur Vererbung dar
- können von beliebigen Klassen implementiert werden
- ermöglichen es Objekten schwach assoziierter Klassen miteinander zu kommunizieren

Protokolle spezifizieren die zu implementierenden Methoden

Methoden können als **verpflichtend** oder **optional** spezifiziert werden

Beispiel: **UIGestureRecognizerDelegate**-Protokoll

```
// Dieses Protokoll ist konform zum NSObject Protokoll
@protocol UIGestureRecognizerDelegate <NSObject>

// die folgenden Methoden MÜSSEN implementiert werden
@required
[...]

// die folgenden Methoden KÖNNEN implementiert werden
@optional
-(BOOL)gestureRecognizer:(UIGestureRecognizer *)gr
    shouldRecognizeSimultaneouslyWithGestureRecognizer:
        (UIGestureRecognizer *)o;

[...]

@end
```

Protokolle können ebenfalls als **verpflichtend** oder **optional** spezifiziert werden

- Dazu muss man es mit dem Attribut `@objc` markieren

```
@objc protocol SomeDelegate {  
    // die folgenden Methoden MÜSSEN implementiert werden  
    func doSomeCalculation(count: Int) -> Int  
  
    // die folgenden Methoden KÖNNEN implementiert werden  
    optional func doSomeOtherStuff() -> Void  
[...]  
  
@end
```

In Swift existieren viele weitere Möglichkeiten zur Definition von Protokollen und zur Restriktion ihrer Anwendung

- Siehe:
https://developer.apple.com/library/ios/documentation/swift/conceptual/Swift_Programming_Language/Protocols.html

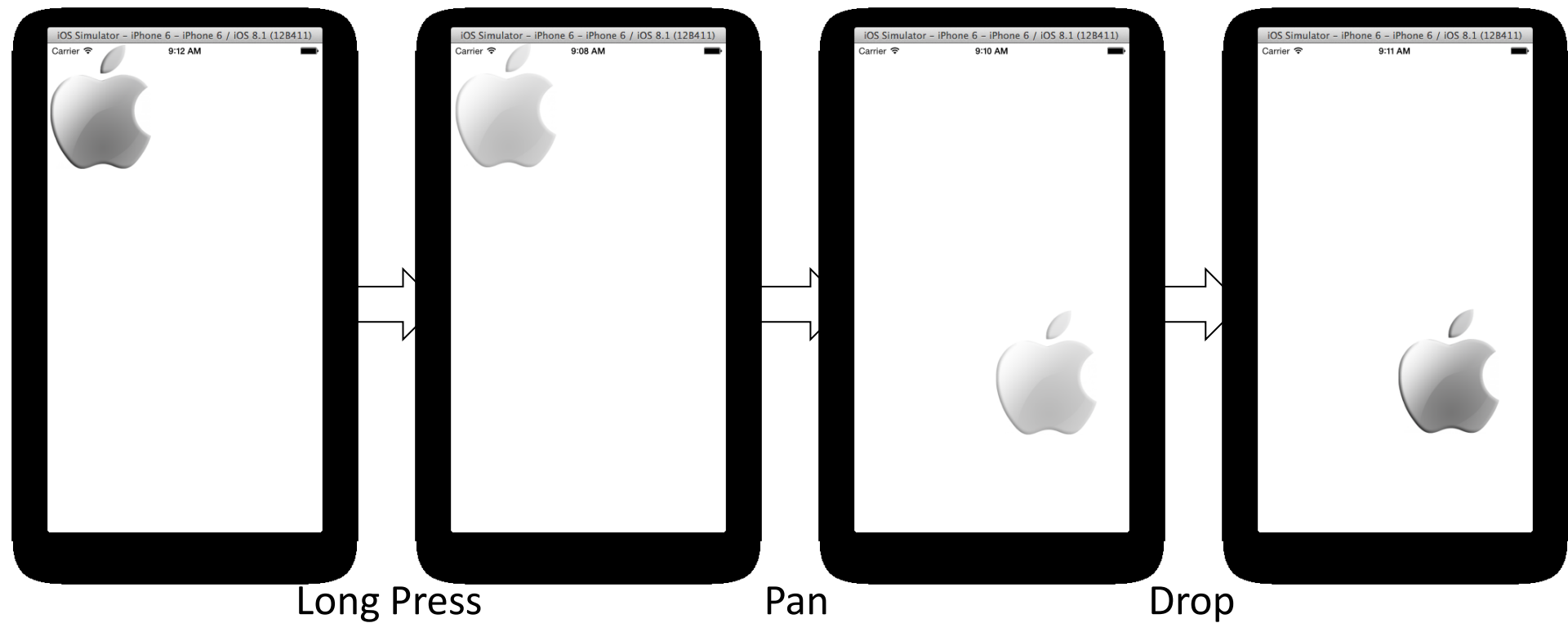
Problem:

- Parallele Erkennung von LongPress und Pan

Lösung:

- Implementierung des UIGestureRecognizerDelegate Protokolls

Beispiel: DragAndDrop



Vorgehensweise

- Analog zum Beispiel "ImageTranslationRecognition"
- Kennzeichnung des View Controller als Delegate

```
// ViewController.swift

import UIKit

class ViewController: UIViewController, UIGestureRecognizerDelegate {
    [...]
}
```

Erzeugen notwendiger Outlets und Properties

```
// ViewController.swift

import UIKit

class ViewController: UIViewController, UIGestureRecognizerDelegate {

    @IBOutlet weak var imageView: UIImageView!
    var panGR: UIPanGestureRecognizer!
    var longPressGR: UILongPressGestureRecognizer!

    [...]

}
```

Initialisierung der Gesture Recognizer

```
// ViewController.swift

import UIKit

class ViewController: UIViewController, UIGestureRecognizerDelegate {
    [...]

    override func viewDidLoad() {
        // Initialisieren/Registrieren eines UILongPressGestureRecognizer
        self.longPressGR = UILongPressGestureRecognizer(
            target:self, action:"handleLongPress:")
        self.imageView.addGestureRecognizer(self.longPressGR)

        // Initialisieren/Registrieren eines UIPanGestureRecognizer
        self.panGR = UIPanGestureRecognizer(target: self, action: "handlePan:")
        self.imageView.addGestureRecognizer(self.panGR)

        // Setzen des UILongPressGestureRecognizer als Delegate
        self.longPressGR.delegate = self
    }
}
```

Gibt die Protokoll-Methode **true** zurück, können die beiden übergebenen **UIGestureRecognizer** parallel zueinander Gesten erkennen.

```
// ViewController.swift

import UIKit

class ViewController: UIViewController, UIGestureRecognizerDelegate {
    [...]

    func gestureRecognizer(
        UIGestureRecognizer,
        shouldRecognizeSimultaneouslyWithGestureRecognizer:
        UIGestureRecognizer) -> Bool {
        // Erlaube parallele Verarbeitung von Pan während LongPress
        // Achtung: Dies erlaubt auch den LongPress während Pan!
        return true
    }
}
```

Implementierung des LongPress-Handlers

```
// ViewController.swift

import UIKit

class ViewController: UIViewController, UIGestureRecognizerDelegate {
    [...]
    func handleLongPress(gestureRecognizer:UILongPressGestureRecognizer) {
        // Visualisiere, dass jetzt ein Drag & Drop passiert
        if gestureRecognizer.state == UIGestureRecognizerState.Began
        || gestureRecognizer.state == UIGestureRecognizerState.Changed {
            self.imageView.alpha = 0.5
        }
        else {
            // Visualisiere, dass Drag & Drop zu Ende ist
            self.imageView.alpha = 1.0
        }
    }
}
```

Implementierung des Pan-Handlers

```
// ViewController.swift

import UIKit

class ViewController: UIViewController, UIGestureRecognizerDelegate {

    [...]

    func handlePan(gestureRecognizer:UIPanGestureRecognizer) {
        // Führe Pan nur aus, wenn gerade ein LongPress aktiv ist
        if self.longPressGR.state == UIGestureRecognizerState.Began
        || self.longPressGR.state == UIGestureRecognizerState.Changed {
            // Verschiebe Image an die Position, wo das Event auftrat
            self.imageView.center =
                gestureRecognizer.locationInView(self.view)
        }
    }
}
```

TOUCH-EVENTS

Gesture Recognizer können nicht immer zur Behandlung von Touch-Events verwendet werden

- Oft ergibt sich kein Vorteil, wenn man das Erkennen eines Touches von der assoziierten Aktion entkoppelt
- Stehen die Inhalte einer View in direktem Zusammenhang mit den erkannten Touches, bietet sich auch deren direkte Behandlung im entsprechenden **UIView**-Objekt an
- Beispiel: Zeichenprogramm
 - Linie soll sich verändern, während sich der Finger bewegt

Wie und wo kommt man an diese Events?

Ein **Touch** entspricht dem Berühren oder Bewegen eines Fingers über den Bildschirm

Eine Geste besteht aus einem oder mehreren Touches

Ein Touch wird als Instanz der Klasse **UITouch** repräsentiert

Beispiel Pinch:

- Besteht aus zwei Touches
- Zwei Finger auf dem Bildschirm, die sich aufeinander zu bzw. voneinander weg bewegen

Touches werden über **Events** kommuniziert

- Ein Event beinhaltet **alle** **UITouch**-Objekte, die innerhalb einer **Multitouch**-Sequenz auftreten

Eine **Multitouch-Sequenz**

- startet mit dem Berühren des ersten Fingers und
- endet mit dem Entfernen des letzten Fingers vom Bildschirm

Wenn sich ein Finger bewegt, werden **UITouch**-Objekte gekapselt in einem **UIEvent**-Objekt gesendet (Typ **UIEventTypeTouches**)

Ein **UITouch**-Objekt

- verfolgt genau einen Finger (= Tracking)
- dauert (bzw. existiert) exakt so lange, wie die Multitouch-Sequenz

Während der Multitouch-Sequenz kümmert sich iOS um das Update der Attribute des **UITouch**-Objekts

- Man muss **UITouch**-Objekte nicht „zwischenspeichern“
- Man **sollte** das auch nie probieren!
 - Werden vom OS verwaltet (ARC, etc.)
- Falls nötig: Kopien von Werten erzeugen oder schwache Referenz

Stadium und Ort von Touches verfolgen

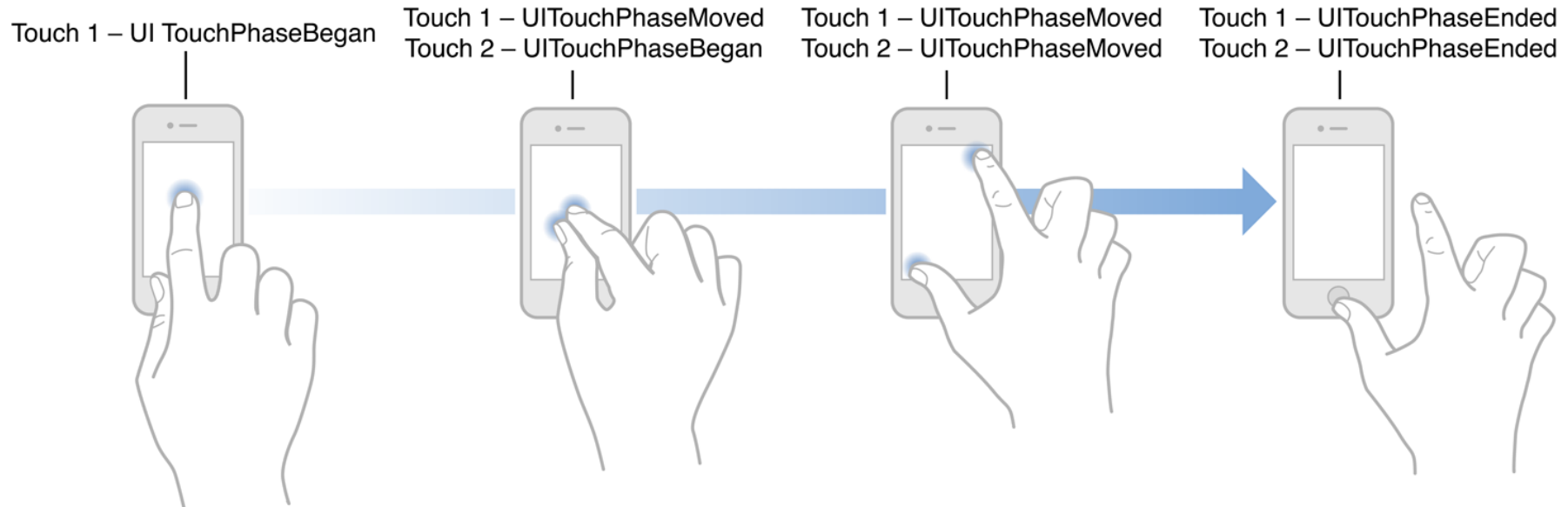
Für jeden Berührungspunkt (Finger) wird ein **UITouch**-Objekt mit folgenden Properties erzeugt

- **phase**: Began, Moved, Ended bzw. Cancelled
- **locationInView**: Position in einer View
- **previousLocationInView**: Vorhergehende Position in der View
- **timeStamp**: Zeitpunkt des Auftretens bzw. der letzten Veränderung
- Seit iOS 9 **force** und **maximumPossibleForce** für 3D Touch

Location steht auf drei Arten zur Verfügung:

- Als Referenz auf das **UIWindow**-Objekt, in dem der Touch auftrat
- Als Referenz auf das **UIView**-Objekt innerhalb dieses **UIWindow**-Objekts
- Als Position innerhalb dieses/eines **UIView**-Objekts

Beispiel



Achtung:

Quelle: <http://www.apple.com/>

- iOS kümmert sich selbst um die Interpretation einer Berührung des Bildschirms und der Transformation in ein **UITouch**-Objekt
- Kein eigener Code an dieser Stelle notwendig!

Klassen, deren Instanzen über Events informiert werden wollen, müssen die **UIResponder**-Schnittstelle implementieren

- **UIResponder** ist die Elternklasse von
 - **UIView**, **UIViewController**, **UIControl**, **UIApplication** und **UIWindow**
- **UIResponder** bietet Methoden zur Behandlung von Touches an
 - `touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event`
(Finger "trifft" auf den Bildschirm)
 - `touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event`
(Finger bewegt sich auf dem Bildschirm)
 - `touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event`
(Finger verlässt den Bildschirm)
 - `touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event`
Wenn eine Multitouch-Sequenz durch ein System-Event abgebrochen wird (z.B. eingehender Anruf)

Jede der Methoden entspricht einer der **Touch-Phasen**

- Began, Moved, Ended bzw. Cancelled

Treten für eine dieser Phasen **neue bzw. veränderte UITouch**-Objekte auf, wird die entsprechende **touch**-Methode aufgerufen

touch-Methoden Parameter:

- **NSSet**:
 - Ein **NSSet**-Objekt, das alle **neuen bzw. veränderten UITouch**-Objekte der entsprechenden Phase enthält
- **UIEvent**:
 - Ein **UIEvent**-Objekt, das **alle UITouch**-Objekte der entsprechenden Multitouch-Sequenz enthält

Beispiel:

- Ein `UITouch`-Objekt `obj` wechselt von der Began- in die Moved-Phase
→ Aufruf von `touchesMoved:withEvent:`
 - `NSSet touches` beinhaltet nun dieses `UITouch`-Objekt `obj` und alle anderen `UITouch`-Objekte, die sich ebenfalls in der Moved-Phase befinden
 - `UIEvent event` beinhaltet **alle** `UITouch`-Objekte der Multitouch-Sequenz
- Es gilt also:
 - `event.allTouches.count >= touches.count`

Ein paar weitere Hinweise:

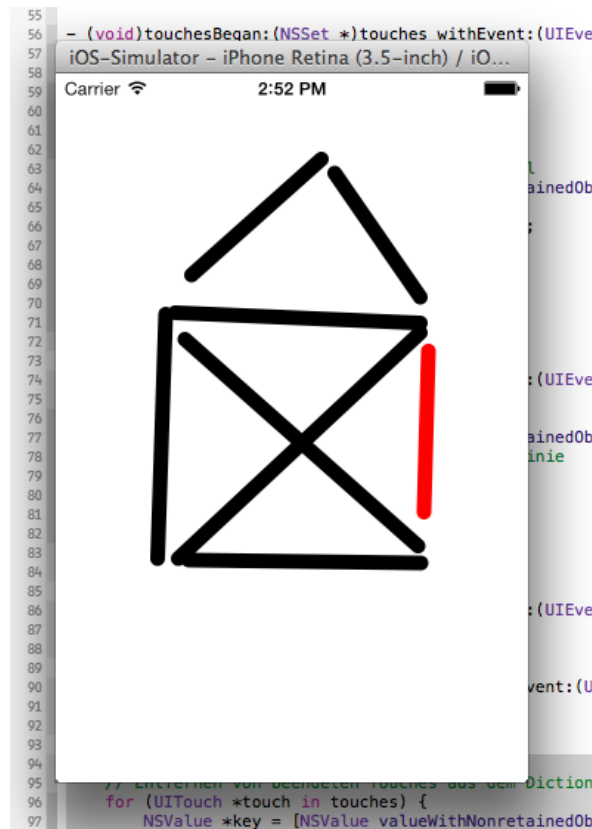
- Entfernt sich der Finger vom Bildschirm, wird das zugehörige `UITouch`-Objekt ein letztes Mal aktualisiert.
- `touchesEnded:withEvent:` `NSSet` enthält das `UITouch`-Objekt, dessen dazugehöriger Finger den Bildschirm nun **nicht mehr** berührt.
- Nach Ausführung von `touchesEnded:withEvent:` werden die übergebenen `UITouch`-Objekte gelöscht.
- Starten mehrere Berührungen gleichzeitig auf demselben `UIView`-Objekt, enthält das `NSSet` beim Aufruf von `touchesBegan:withEvent:` mehrere Elemente. (Das Zeitfenster für Gleichzeitigkeit ist sehr klein...)

Viele weitere wichtige Details im *"Event Handling Guide for iOS"*

Voraussetzungen zur Behandlung von Touch-Events durch eine View

- Die entsprechende Klasse muss von (einer Unterklasse von) **UIResponder** erben
- Die entsprechende Klasse muss die **UIResponder**-Schnittstelle (**touch**-Methoden) implementieren
- Die Property **userInteractionEnabled** muss auf **YES** bzw. **true** gesetzt sein
- Die mit dem **UIResponder**-Objekt assoziierte View (View selbst bzw. das assoziierte **UIViewController**-Objekt) muss sichtbar sein (nicht **transparent**; nicht **hidden**)

Einfaches Zeichenprogramm, das das parallele Zeichnen von Linien erlaubt
(Beispiel basiert auf "*The Big Nerd Ranch Guide*")



Vorgehensweise

- Erzeugen eines neuen Projekts "DrawLines" (Template: Single View Application)
- Erzeugen einer Klasse `Line` zur Speicherung von Linien (erbt von `NSObject`)
- Erzeugen einer Klasse `DrawLinesView` zur Anzeige der Zeichnung (erbt von `UIView`)

Achtung:

- Dieses Beispiel dient nur zur Verdeutlichung des Touch-Konzepts (keine saubere Trennung zwischen Modell und View)

Implementierung der Klasse `Line`

```
// Line.h

#import <Foundation/Foundation.h>

@interface Line : NSObject
@property (nonatomic) CGPoint begin;
@property (nonatomic) CGPoint end;
@end
```

```
// Line.m

#import "Line.h"

@implementation Line

@end
```

Hinzufügen benötigter Datenstrukturen zur Klasse `DrawLinesView`

```
// DrawLinesView.h

#import <UIKit/UIKit.h>

@interface DrawLinesView : UIView

// zur Verwaltung "unfertiger" Linien
@property NSMutableDictionary *pendingLines;

// zur Speicherung gezeichneter Linien
@property NSMutableArray *finishedLines;

// zum Löschen aller Linien
-(void)clearAll;
@end
```

Initialisierung der Datenstrukturen und Konfiguration der View

```
// DrawLinesView.m

-(instancetype)initWithFrame:(CGRect)frame {
    self = [super initWithFrame:frame];
    if (self) {
        self.pendingLines = [NSMutableDictionary new];
        self.finishedLines = [NSMutableArray new];
        self.backgroundColor = [UIColor whiteColor];
        // Standard ist NO (d.h. alle Touches außer dem Ersten würden
        // ignoriert
        self.multipleTouchEnabled = YES;
    }
    return self;
}
```

Implementierung der `drawRect`-Methode von `DrawLinesView`

```
// DrawLinesView.m
-(void)drawRect:(CGRect)rect {
    CGContextRef ctx = UIGraphicsGetCurrentContext();
    CGContextSetLineWidth(ctx, 10.0);
    CGContextSetLineCap(ctx, kCGLineCapRound);
    // fertige Linien werden schwarz
    [[UIColor blackColor] set];
    for(Line *line in self.finishedLines) {
        CGContextMoveToPoint(ctx, line.begin.x, line.begin.y);
        CGContextAddLineToPoint(ctx, line.end.x, line.end.y);
        CGContextStrokePath(ctx);
    }
    // "unfertige" Linien werden rot
    [[UIColor redColor] set];
    for(NSValue *v in self.pendingLines) {
        Line *line = [self.pendingLines objectForKey:v];
        CGContextMoveToPoint(ctx, line.begin.x, line.begin.y);
        CGContextAddLineToPoint(ctx, line.end.x, line.end.y);
        CGContextStrokePath(ctx);
    }
}
```

Implementierung der `clearAll`-Methode von `DrawLinesView`

```
// DrawLinesView.m

- (void)clearAll {
    [self.pendingLines removeAllObjects];
    [self.finishedLines removeAllObjects];
    [self setNeedsDisplay];
}
```

Implementierung von `touchesBegan:withEvent`

```
// DrawLinesView.m
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    for (UITouch *touch in touches) {
        // Double Tap soll alles löschen
        if(touch.tapCount > 1) {
            [self clearAll];
            return;
        }
        // Verwende (schwache) Referenz auf UITouch-Objekt als Schlüssel
        NSValue *key = [NSValue valueWithNonretainedObject:touch];
        // Erzeugen einer Linie
        CGPoint p = [touch locationInView:self];
        Line *newLine = [Line new];
        newLine.begin = p;
        newLine.end = p;
        self.pendingLines[key] = newLine;
    }
}
```

Implementierung von `touchesMoved:withEvent`

```
// DrawLinesView.m

-(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    // Update der ausstehenden Linien
    for (UITouch *touch in touches) {
        // Verwende (schwache) Referenz auf UITouch-Objekt als Schlüssel
        NSValue *key = [NSValue valueWithNonretainedObject:touch];
        // Dereferenzieren der entsprechenden Linie
        Line *line = self.pendingLines[key];
        // Update des Endpunkts
        line.end = [touch locationInView:self];
    }
    [self setNeedsDisplay];
}
```

Eine Linie soll fertiggestellt werden, wenn

- der Finger sich vom Display bewegt (`touchesEnded:withEvent:`)
- ein System-Event die Anwendung unterbricht (`touchesCancelled:withEvent:`)

Hier werden zur Vereinfachung in beiden Fällen die Linie fertigstellen

```
// DrawLinesView.m
```

```
-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {  
    [self endTouches:touches];  
}
```

```
-(void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event {  
    [self endTouches:touches];  
}
```

Implementierung von `endTouches:touches`

```
// DrawLinesView.m

-(void)endTouches:(NSSet*)touches {
    // Entfernen von beendeten Touches aus dem Dictionary
    for (UITouch *touch in touches) {
        // Verwende (schwache) Referenz auf UITouch-Objekt als Schlüssel
        NSValue *key = [NSValue valueForKey:@"touch"];
        // Dereferenzieren der entsprechenden Linie
        Line *line = self.pendingLines[key];
        // falls ein Double Tap vorliegt, ist line == nil!
        if(line) {
            [self.finishedLines addObject:line];
            [self.pendingLines removeObjectForKey:key];
        }
    }
    [self setNeedsDisplay];
}
```

Initialisierung der View im View Controller

```
// DrawLinesView.m

#import "DrawLinesViewController.h"
#import "DrawLinesView.h"

@implementation DrawLinesViewController

-(void)loadView {
    self.view = [[DrawLinesView alloc] initWithFrame:CGRectZero];
}

@end
```

Anmerkung zur Klasse `NSValue`:

- `NSValue`-Objekte dienen als einfache Container für einzelne C bzw. Objective-C Datentypen.
- Erlaubt sind
 - Skalare Typen (z.B. int, float, und char)
 - Pointer
 - Strukturen
 - Objektreferenzen
- `NSValue`-Instanzen dienen u.a. dazu, solche Datentypen z.B. Instanzen der Klassen `NSArray` oder `NSSet` hinzuzufügen (Elemente solcher Collections müssen Objekte sein!)
- `NSValue`-Objekte sind stets unveränderlich (immutable)

Anmerkung zur Klasse `NSValue`:

- `+(NSValue *)valueWithNonretainedObject:(id)anObject`
 - Die Methode liefert eine Instanz vom Typ `NSValue`, die das Objekt `anObject` beinhaltet.
 - Großer Vorteil: Es wird dabei eine **schwache Referenz** auf `anObject` angelegt!
 - `anObject` kann somit als Schlüssel innerhalb des Dictionarys verwendet werden, ohne dass die Collection das Objekt besitzt

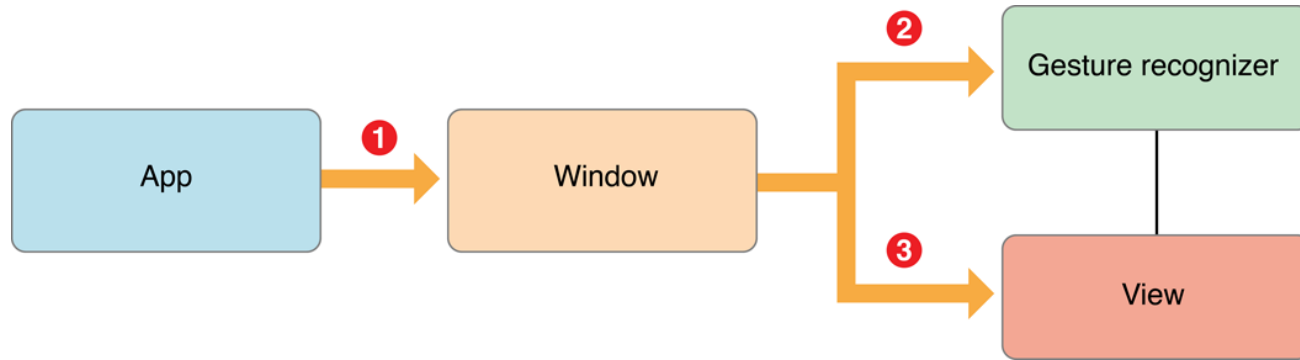
TOUCH EVENTS VS. GESTURE RECOGNIZER

Man kann Touch-Events über Gesture Recognizer oder direkt über den entsprechenden Responder behandeln

Was wenn beide Mechanismen Anwendung finden?

Wer wird zuerst über Touches informiert?

Standardpfad der Auslieferung von Touch-Events



Quelle: <http://www.apple.com/>

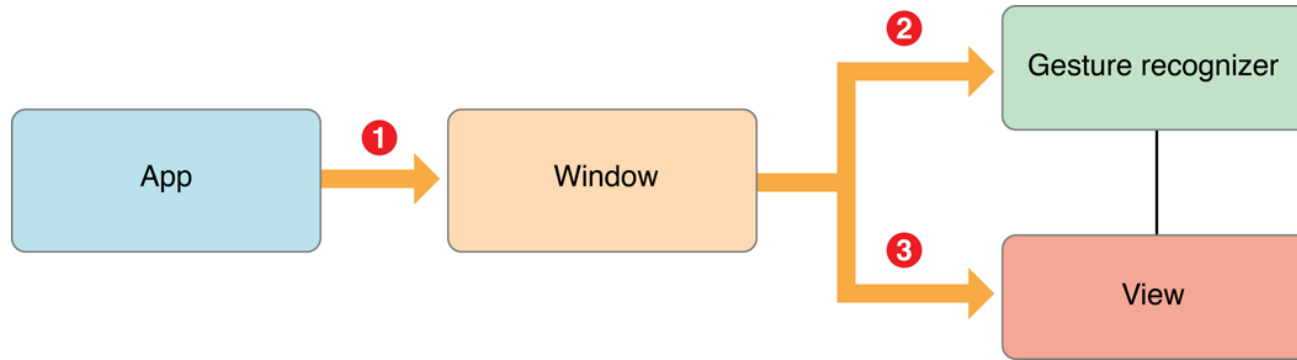
UITouch-Objekte werden zuerst an die registrierten Gesture Recognizer einer View (Superview) gesendet

Erst dann gelangen sie zur View selbst!

Die Auslieferung von **UITouch**-Objekten an die View wird dazu von der **UIWindow**-Instanz verzögert

→ Recognizer können Touch zuerst auswerten

Standardpfad der Auslieferung von Touch-Events



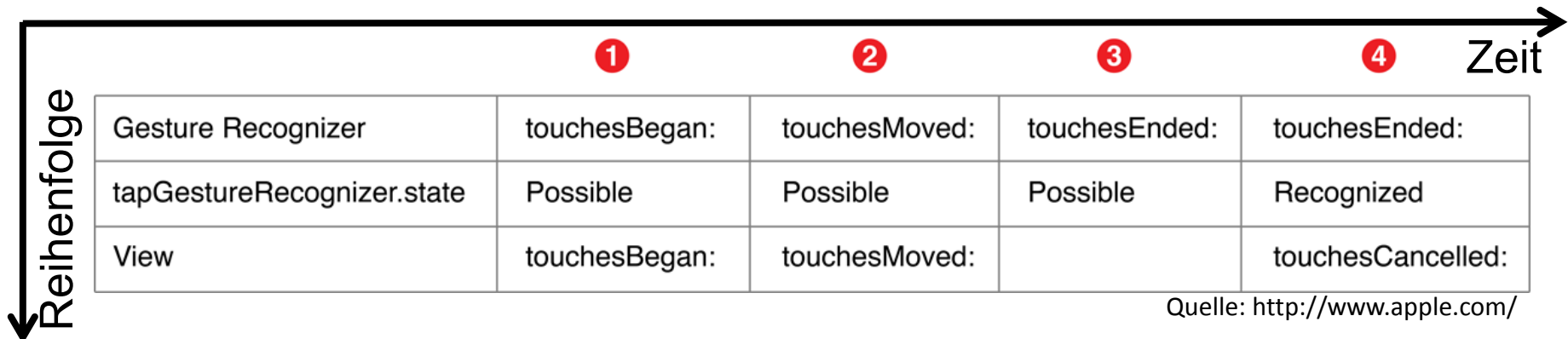
Quelle: <http://www.apple.com/>

Wird eine Geste erkannt, werden keine (weiteren) **UITouch**-Objekte an die View gesendet

Etwaige vorhergehende Touch-Events, die die View schon erhalten hat, werden als ungültig gekennzeichnet

Beispiel:

- Diskrete Geste, die sich durch einen Zwei-Finger-Tap definiert



1. **UIWindow**-Instanz sendet zwei **UITouch**-Objekte an Gesture Recognizer (GR). GR kann noch keine Geste erkennen → Weiterleitung der **UITouch**-Objekte an die View
2. **UIWindow**-Instanz sendet zwei **UITouch**-Objekte an GR. GR kann immer noch keine Geste erkennen → Weiterleitung der **UITouch**-Objekte an die View
3. **UIWindow**-Instanz sendet ein **UITouch**-Objekt an GR. GR hat noch nicht genug Info, um Geste komplett zu erkennen, **ABER UIWindow**-Instanz leitet **UITouch**-Objekte **NICHT** an die View weiter
4. **UIWindow**-Instanz sendet zweites **UITouch**-Objekt an GR. GR erkennt die Geste → View erklärt vorhergehende Touches als ungültig.
(Wenn Gestenerkennung hier fehlschlägt, werden die beiden letzten **UITouch**-Objekt an die View weitergereicht)

Einflussnahme auf die Abhandlung von Touch-Events erfolgt über Anpassung entsprechender Properties in den registrierten Gesture Recognizern

delaysTouchesBegan

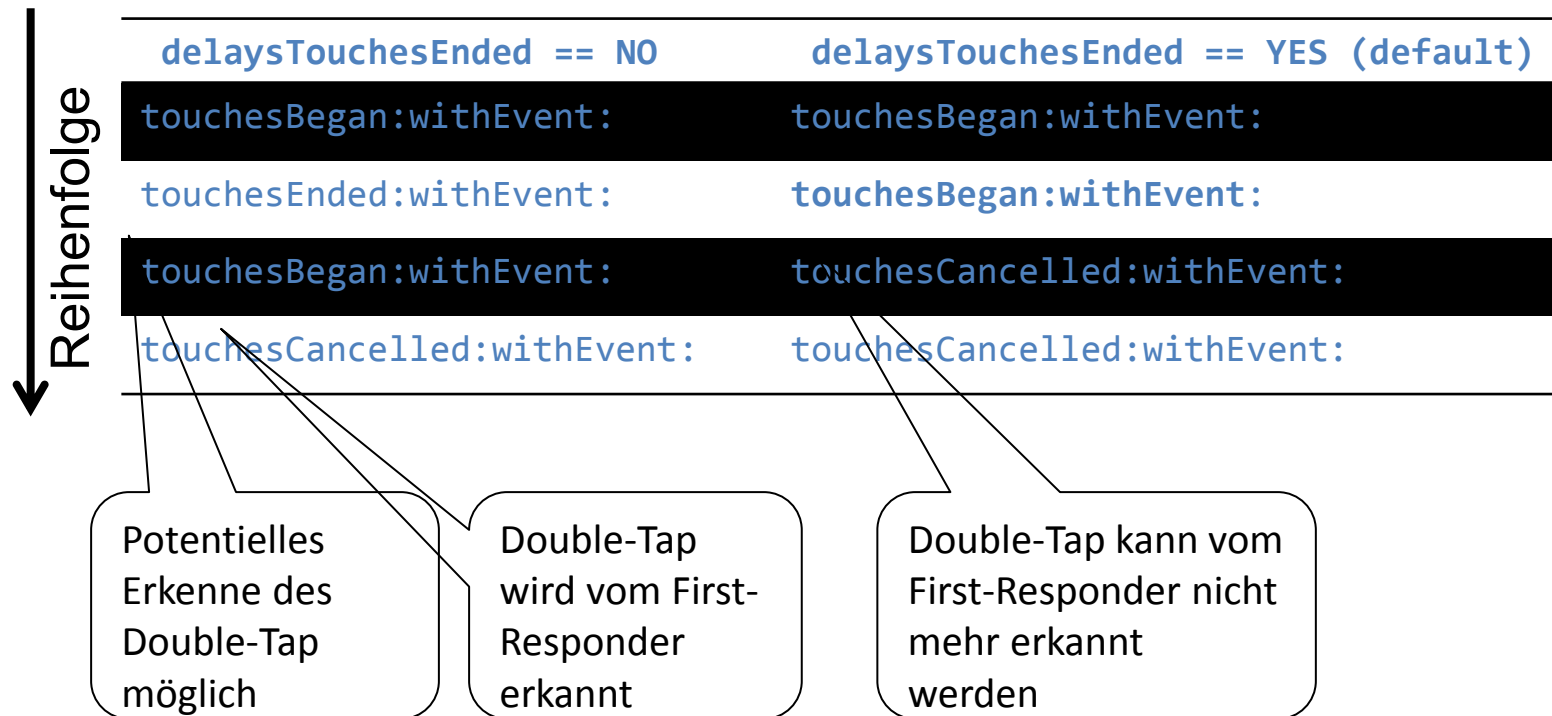
- Falls **YES**: Verhindert die Auslieferung von **UITouch**-Objekten in der Begin-Phase. Stellt sicher, dass keinerlei UITouch-Objekt vor Erkennung einer Geste an die View gesendet wird (default: **NO**)
- Kann zu schlechter Antwortzeit der UI führen!

delaysTouchesEnded

- Falls **YES**: Stellt sicher, dass die View keine Actions ausführt, die ein GR später u.U. noch abbrechen muss.
- Falls **NO**: **UITouch**-Objekten in der Ended-Phase werden direkt an die View weitergereicht und parallel mit dem Gesture Recognizer verarbeitet (default: **YES**)

Beispiel zu `delaysTouchesEnded`:

- Erkennen eines Double-Taps im First-Responder bei gleichzeitig installiertem `UITapGestureRecognizer` (`numberOfTapsRequired == 2`)
- Reihenfolge eingehender Nachrichten bei der First-Responder-View



Ignorieren von **UITouch**-Objekten mit **ignoreTouch:forEvent:**

- Z.B. falls Gesture Recognizer einen Touch analysiert, der nicht Teil seiner Geste ist
- Aufruf (auf sich selbst) bewirkt direkte Weitergabe des **UITouch**-Objekts an die View

Der Checkmark Gesture Recognizer

- Beispiel basiert auf adaptiertem Quellcode von: <http://www.apple.com/>
- Das Zeichnen eines Checkmarks (Häkchen) soll als Geste erkannt werden
- Implementierung als diskrete Geste (Zustände: Possible, Failed, Ended == Recognized)



Vorgehensweise

- Erzeugen einer Klasse, die von `UIGestureRecognizer` erbt
- Importieren des Headers `UIGestureRecognizerSubclass.h`
- Überschreiben der Methoden
 - `-(void)reset;`
 - `-(void)touchesBegan:(NSSet *)touches
withEvent:(UIEvent *)event;`
 - `-(void)touchesMoved:(NSSet *)touches
withEvent:(UIEvent *)event;`
 - `-(void)touchesEnded:(NSSet *)touches
withEvent:(UIEvent *)event;`
 - `-(void)touchesCancelled:(NSSet *)touches
withEvent:(UIEvent *)event;`

```
// CheckmarkGestureRecognizer.h
```

```
#import <UIKit/UIKit.h>
```

```
#import <UIKit/UITapGestureRecognizerSubclass.h>
```

```
@interface CheckmarkGestureRecognizer : UITapGestureRecognizer
```

```
-(void)reset;
```

```
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
```

```
-(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
```

```
-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;
```

```
-(void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event;
```

```
@end
```

```
// CheckmarkGestureRecognizer.m

#import "CheckmarkGestureRecognizer.h"

@interface CheckmarkGestureRecognizer()
@property (nonatomic) BOOL strokeUp;
@property (nonatomic) CGPoint midPoint;
@end

@implementation CheckmarkGestureRecognizer

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesBegan:touches withEvent:event];
    if ([touches count] != 1) {
        self.state = UIGestureRecognizerStateFailed;
        return;
    }
    // Setze den Mittelpunkt auf den Startpunkt
    self.midPoint = [touches.anyObject locationInView:self.view];
}
```

```

-(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesMoved:touches withEvent:event];
    if (self.state == UIGestureRecognizerStateFailed) return;
    CGPoint nowPoint = [touches.anyObject locationInView:self.view];
    CGPoint prevPoint = [touches.anyObject previousLocationInView:self.view];
    // Sind wir noch auf dem Weg nach rechts unten?
    if (!self.strokeUp) {
        // Setze den Mittelpunkt, falls noch auf dem Weg nach rechts unten
        if (nowPoint.x >= prevPoint.x && nowPoint.y >= prevPoint.y) {
            self.midPoint = nowPoint;
        }
        // Ab jetzt interessiert nur noch das Ende der Geste
        else if (nowPoint.x >= prevPoint.x && nowPoint.y < prevPoint.y) {
            self.strokeUp = YES;
        }
        else {
            self.state = UIGestureRecognizerStateFailed;
        }
    }
}

```

```
-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesEnded:touches withEvent:event];
    if ((self.state == UIGestureRecognizerStatePossible) && self.strokeUp) {
        CGPoint nowPoint = [touches.anyObject locationInView:self.view];
        // Liegt der Endpunkt rechts oberhalb des Mittelpunkts?
        if(nowPoint.x >= self.midPoint.x && nowPoint.y < self.midPoint.y) {
            self.state = UIGestureRecognizerStateRecognized;
        }
        else {
            self.state = UIGestureRecognizerStateFailed;
        }
    }
}
```

```

-(void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesCancelled:touches withEvent:event];
    self.midPoint = CGPointZero;
    self.strokeUp = NO;
    self.state = UIGestureRecognizerStateFailed;
}

-(void)reset {
    [super reset];
    self.midPoint = CGPointZero;
    self.strokeUp = NO;
}
@end

```

Ergebnis:

