ETSIT UPM

dit UPM

Desarrollo de Apps para iOS
Reconocedores de Gestos

IWEB 2015-2016
Santiago Pavón

ver: 2015.10.21

Terminología

- **Gesto**: Secuencia de eventos provocada por varios dedos, y que termina al retirar los dedos, o cuando el sistema lo interrumpe.
- **Evento**: generado al interactuar con la pantalla, e informa de los toques ocurridos.
- **Toque**: representa el contacto de un dedo en la pantalla.

UIResponder

- **UIResponder**:
  - Responde y maneja eventos.
    - de tipo touch, motion, de dispositivos de control remoto.
  - Es una **superclase** de `UIView`, `UIControl`, `UIApplication` `UIViewController`, etc.
- Algunos métodos de `UIResponder`:

```
func touchesBegan(_ touches: NSSet,
        withEvent event: UIEvent)

func touchesMoved(_ touches: NSSet,
        withEvent event: UIEvent)

func touchesEnded(_ touches: NSSet,
        withEvent event: UIEvent)

func touchesCancelled(_ touches: NSSet!,
            withEvent event: UIEvent!)
```

Responder Chain

- **First Responder**:
  - objeto con el que estamos interactuando.
  - es el comienzo de la cadena de respuesta.
- Los eventos avanzan por la cadena de respuesta hasta llegar a `UIWindow`, luego a `UIApplication`, y finalmente se descartan.
  - El avance por la cadena suele romperse cuando el evento es atendido en algún punto intermedio.

Demo: Programar un Swipe

- Hay que sobreescribir los métodos:
  - **`touchesBegan(_,withEvent:)`**
    - Guardamos la posición inicial
  - **`touchedEnded(_,withEvent:)`**
    - Hay que comprobar que:
      - la posición actual no se ha desviado mucho vertical u horizontalmente.
      - ya se ha recorrido una distancia mínima para aceptar el gesto.

(la demo no mira la trayectoria intermedia)

```
import UIKit

let kMinLength: CGFloat = 30
let kMaxError: CGFloat = 5

class ViewController: UIViewController {
    @IBOutlet weak var infoLabel: UILabel!
    var initialPoint: CGPoint?

    override func touchesBegan(touches: NSSet, withEvent event: UIEvent) {
       let touch: UITouch = touches.anyObject() as UITouch
       initialPoint = touch.locationInView(view)
    }

    override func touchesEnded(touches: NSSet, withEvent event: UIEvent) {
        let touch: UITouch = touches.anyObject() as UITouch
        let currentPoint = touch.locationInView(view)

        let diffX: CGFloat = abs(initialPoint!.x - currentPoint.x)
        let diffY: CGFloat = abs(initialPoint!.y - currentPoint.y)

        if diffX >= kMinLength && diffY <= kMaxError {
            // Detectado SWIPE Horizontal - Hacer algo
            infoLabel.text = "Horizontal"
        } else if diffY >= kMinLength && diffX <= kMaxError {
            // Detectado SWIPE Vertical - Hacer algo
            infoLabel.text = "Vertical"
        }
    }
}
```

Reconocedores de Gestos

Reconocedores de Gestos

- Un reconocedor de gestos es un objeto que:
  - Vigila los eventos que ocurren en una determinada view,
  - Y cuando reconoce el gesto (o parte del gesto) que lo caracteriza, ejecuta las targets-actions que tiene configuradas.
- Reconocedores de gestos predefinidos:
  - Discretos:
    - `UITapGestureRecognizer`
    - `UISwipeGestureRecognizer`
  - Continuos:
    - `UIPinchGestureRecognizer`
    - `UIRotationGestureRecognizer`
    - `UIPanGestureRecognizer`
    - `UILongPressGestureRecognizer`
    - `UIScreenEdgePanGestureRecognizer`
- También podemos programar nuestros propios reconocedores de gestos.

UIGestureRecognizer

- **`UIGestureRecognizer`**: Clase base de la que derivan todos los reconocedores de gestos.
  - Es una clase abstracta: no crear instancias de esta clase, sino de sus subclases
- Creación:

```
init(target: AnyObject, action: Selector)
```

  - Cuando se reconozca el gesto, se ejecuta la acción dada del target indicado.
- Añadir nuevos target/action a ejecutar al reconocer el gesto, o quitarlos:

```
func addTarget(_: AnyObject, action: Selector)
func removeTarget(_: AnyObject?, action: Selector)
```

- Información sobre el gesto:

```
func locationInView(_: UIView?) -> CGPoint
func locationOfTouch(_: Int, inView: UIView?) -> CGPoint
func numberOfTouches() -> Int
```

  - Posición del gesto/toque en la view, número de toques del gesto.

- Propiedades:
  - `state`
  - `view`
  - `enabled`
  - `delegate`
  - . . .
- Dependencias entre gestos:

```
func requireGestureRecognizerToFail(_: UIGestureRecognizer)
```

  - Cuando dos gestos empiezan igual hay que indicar cuál se desea reconocer antes.
    - Si falla el primero, reconozco el segundo.
      - Ejemplo: un swipe horizontal debe reconocerse sólo después de que la Z del zorro haya fallado.
- etc…

Target y Acción

- El mensaje de acción enviado a los targets cuando se reconoce un gesto puede:
  - no llevar parámetros

```
func manejaGesto()
```

  - o tomar como parámetro el objeto reconocedor:

```
func manejaGesto(recognizer: UIGestureRecognizer)
```

Añadir el Reconocedor a la UIView

- Supongamos que:
  - Ya hemos creado un objeto reconocedor para algún gesto.
  - Ya le hemos dicho a ese reconocedor que acciones tiene que invocar cuando reconozca ese gesto (usando `init(target:,action:)` o `addTarget(,action:)`
- Sólo falta decir cual es la `UIView` que debe vigilar el reconocedor.
  - **Programáticamente**:
    - Se hace con el método **`addGestureRecognizer`** de `UIView`.
  - **Interface** Builder:
    - Enlazando la propiedad **`gestureRecognizers`** de la UIView con los objetos reconocedores de gestos:
      - Ctrl-Arrastrar desde una view hasta el objeto reconocedor.
  - A partir de este momento, el reconocedor analiza los eventos que ocurren en la `UIView`, y si detecta el gesto que le interesa, ejecuta la acciones programadas.

Desde Interface Builder

- Pueden añadirse reconocedores de gestos a las escenas de un storyboard o nib.
  - Arrastrándo los objetos reconocedores de gestos desde la librería de objetos hasta los ViewControllers.
- Con el Inspector de Atributos puede ajustarse las propiedades de los reconocedores de gestos.
- Pueden enlazarse los objetos reconocedores con los métodos IBAction existentes para que atiendan los gestos *(target-action)*.
  - O crear directamente nuevos métodos IBAction mediante Ctrl-Arrastrar desde un objeto reconocedor hasta el código de la clase en la que se desea crear el método IBAction.
- Se puede enlazar la propiedad `gestureRecognizers` de las UIView con los objetos reconocedores de gestos que la deben vigilar:
  - Ctrl-Arrastrar desde la view hasta el objeto reconocedor, y seleccionar la propiedad.

Ejemplo: Reconoce Tap

- Un ViewController crea un reconocedor de taps que al reconocer un tap llama a su método **`procesaTap`**.
- Y se lo añade a su top view.

```
func processTap(sender: UITapGestureRecognizer) {
    let pos = sender.locationInView(sender.view)
    print("TAP en x=\(pos.x) y=\(pos.y)")
}

override func viewDidLoad() {
    super.viewDidLoad()

    let tapRec = UITapGestureRecognizer(target: self,
                                        action: "processTap:")
        tapRec.numberOfTapsRequired = 1
        view.addGestureRecognizer(tapRec)
}
```

El reconocedor

¿Dónde pulsé en sender.view?

En este ejemplo sender.view es self.view

Reconocedor de taps

Ejecutar en **self** el método **processTap:**
(: indica que tiene un argumento)

Reconocer el gesto en la top view del VC

El Estado

- La propiedad **`state`** indica en que estado se encuentra el reconocedor.
  - El reconocedor está en el estado **`.Possible`** hasta que empieza a reconocer un gesto.
  - Si el gesto que reconoce es **discreto**, pasa al estado **`.Recognized`** cuando lo reconoce.
  - Si el gesto es **continuo**, pasa al estado **`.Began`**, y después a **`.Changed`**, hasta llegar finalmente a **`.Ended`**.
    - pero esta secuencia puede terminar con **`.Failed`** o **`.Cancelled`**.

Gestos Continuos

- Los reconocedores de gestos continuos llaman a los manejadores (target/action) registrados cada vez que cambia el estado.
- Los manejadores deben comprobar el valor de **`state`** para decidir que deben hacer según el estado.

```
func manejador(sender: UILongPressGestureRecognizer) {

  if sender.state != .Began {return}

  // hacer cosas
}
```

Sólo se ejecuta cuando **comienza** el gesto Long Press

UITapGestureRecognizer

- Reconocer toques (golpe) rápidos.
- Propiedades para configurar el gesto a reconocer:

```
var numberOfTapsRequired: Int
var numberOfTouchesRequired: Int
```

UISwipeGestureRecognizer

- Reconoce un movimiento horizontal o vertical del dedo sobre la view.
- Propiedades para configurar el gesto a reconocer:

```
var direction: UISwipeGestureRecognizerDirection

var numberOfTouchesRequired: Int
```

UIPinchGestureRecognizer

- Reconoce pellizcos con dos dedos.
  - o la separación de los dos dedos.
- Propiedades con información sobre el gesto:

```
var scale: CGFloat
```

- es el valor de escala acumulado.
- puede resetearse para borrar el valor acumulado.

```
var velocity: CGFloat  {get}  // escala/seg
```

UIRotationGestureRecognizer

- Reconoce giros con dos dedos.
- Propiedades:

```
var rotation: CGFloat
```

  - es el valor de rotación acumulado.
  - puede resetearse para borrar el valor acumulado.

```
var velocity: CGFloat  {get}  // radianes/seg
```

UIPanGestureRecognizer

- Reconoce movimientos (Drag) de uno o varios dedos.
- Propiedades para configurar el gesto a reconocer:

```
var minimumNumberOfTouches: Int
var maximumNumberOfTouches: Int
```

- Métodos informativos sobre el gesto:

```
func translationInView(_: UIView) -> CGPoint
```

- Distancia (valor acumulado desde que empezo el gesto) que se ha movido el dedo en la view dada.

```
func setTranslation(_: CGPoint, inView: UIView!)
```

- Resetear o cambiar el valor acumulado.

```
func velocityInView(_: UIView!) -> CGPoint  // puntos/seg
```

UILongPressGestureRecognizer

- Reconoce pulsaciones largas
  - Es un gesto **continuo**.
    - por tanto, invoca los manejadores (target/action) si hay desplazamientos de los dedos tras la pulsación larga.
- Propiedades para configurar el gesto a reconocer:

```
var minimumPressDuration: CFTimeInterval
var numberOfTapsRequired: Int
var numberOfTouchesRequired: Int
var allowableMovement: CGFloat
```

UIScreenEdgePanGestureRecognizer

- Reconoce un gesto Pan que comienza cerca del borde de la pantalla.
  - Algunas aplicaciones usan este gesto para realizar transiciones entre View Controllers.
- Propiedades para configurar el gesto a reconocer:

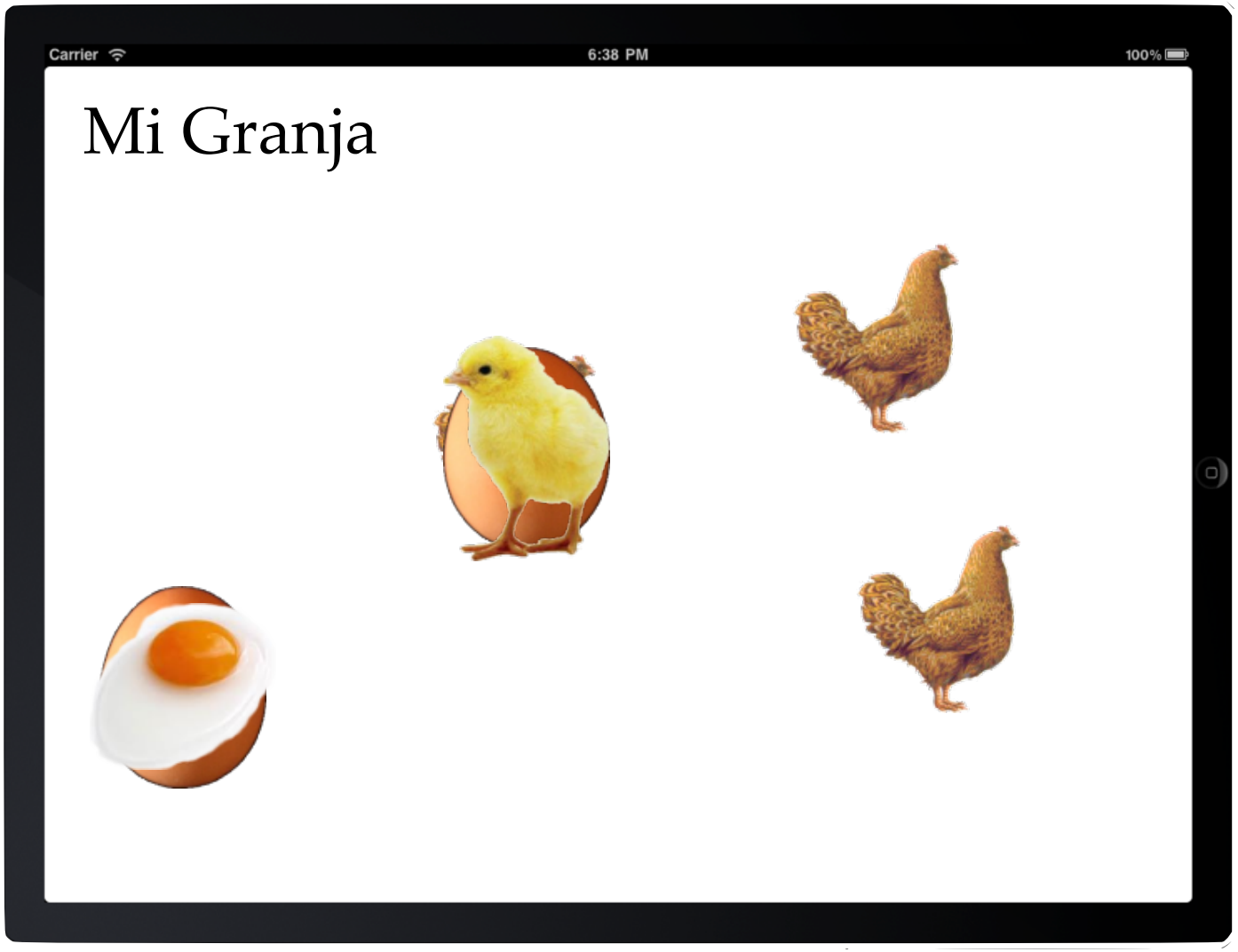
```
var edges: UIRectEdge
```

  - El valor de **edges** es una máscara de bits usando los valores:
    - **.None**, **.All**, **.Top**, **.Left**, **.Bottom** y **.Right**.

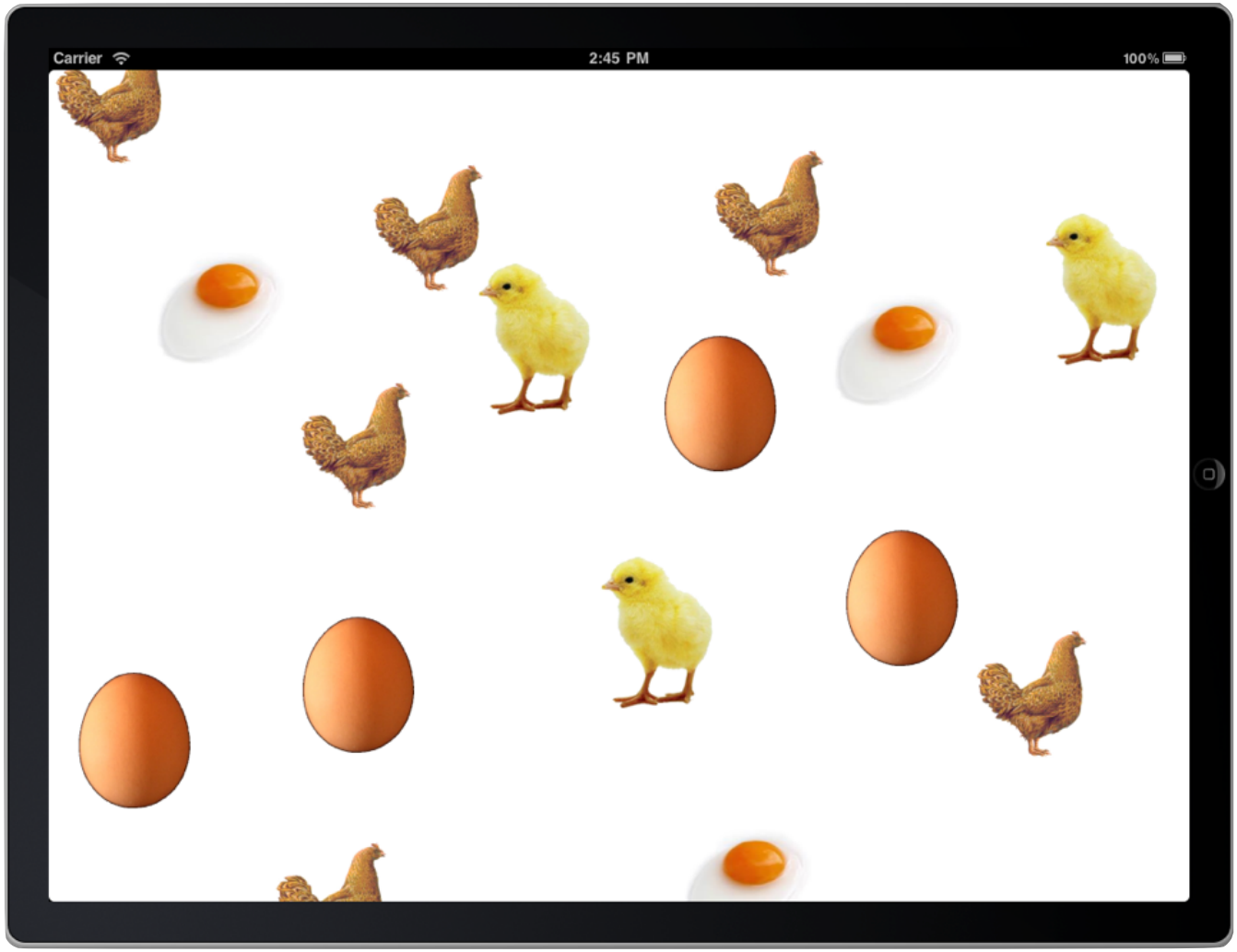
Demo *(Programático)*

La Granja

Demo

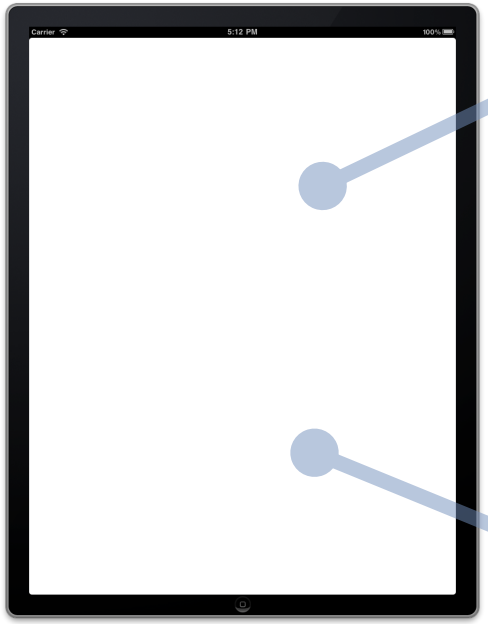


Demo



Reconocedores para el Fondo

- Los creo en **`viewDidLoad`** de ViewController.
  1. Reconocedor de un gesto **Tap** en el fondo
     - Llama a **`processRootTap`**
       - crea una **GHPImageView** (muestra una gallina)
  2. Reconocedor de un gesto **ScreenEdgePan** en el fondo
     - Llama **`processRootEdge`**
       - borra la pantalla
- (el fondo es la **`self.view`** del ViewController)



Reconocedor de Tap

processRootTap

Reconocedor de ScreenEdgePan

processRootEdge

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Crear reconocedor de Tap para crear gallina
    let tapRec = UITapGestureRecognizer(
                                    target: self,
                                    action: "processRootTap:")
    tapRec.numberOfTapsRequired = 1
    view.addGestureRecognizer(tapRec)

    // Crear reconocedor de SreenEdgePan para borrar todo
    let edgeRec = UIScreenEdgePanGestureRecognizer(
                                    target: self,
                                    action: "processRootEdge:")
    edgeRec.edges = .Left
    view.addGestureRecognizer(edgeRec)
}
```

Manejador para crear gallina inicial

- **processRootTap** crea la **GHPView** que muestra una gallina, y la añade a **self.view** (el fondo):

```
func processRootTap(sender: UITapGestureRecognizer) {

    let pos = sender.locationInView(sender.view)

    let rect = CGRectMake(pos.x, pos.y, 1, 1)

    let imgv = GHPImageView(frame: rect)

    view.addSubview(imgv)
}
```

Borrar el Fondo

```
func processRootEdge(sender: UIScreenEdgePanGestureRecognizer) {

    if sender.state != .Began { return }

    for subview in view.subviews {
        if subview is GHPImageView {
            subview.removeFromSuperview()
        }
    }
}
```

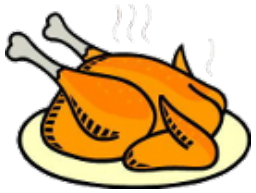
Quitamos todas las subviews de tipo GHPImageView de self.view

```
func processRootEdge(sender: UIScreenEdgePanGestureRecognizer) {

    if sender.state != .Began { return }

    UIView.transitionWithView(view,
        duration: 0.5,
        options: .TransitionCrossDissolve,
        animations: { for subview in self.view.subviews {
                          if subview is GHPImageView {
                              subview.removeFromSuperview()
                          }
                      }
                    },
        completion: nil
        )
    }
```

Igual que la versión anterior pero con una animación.

La Clase GHPImageView

- Es una clase propia que deriva de **UIImageView**
  - ampliada con una propiedad para indicar su estado:
    - .HEN
    - .EGG
    - .CHICKEN
    - .FRIED
    - .ROAST



  - al cambiar el valor de la propiedad se cambia la imagen mostrada.

```swift
import UIKit

enum GHPState {
    case HEN
    case EGG
    case CHICKEN
    case FRIED
    case ROAST
}

class GHPImageView: UIImageView {

    // Estado que indica la imagen a mostrar
    var ghpState: GHPState = .HEN {
        didSet {
            updateGHPStateImage()
        }
    }

    // Inicializador
    override init(frame: CGRect) {
        super.init(frame: frame)
        updateGHPStateImage()
    }

    required init(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
   . . .
```

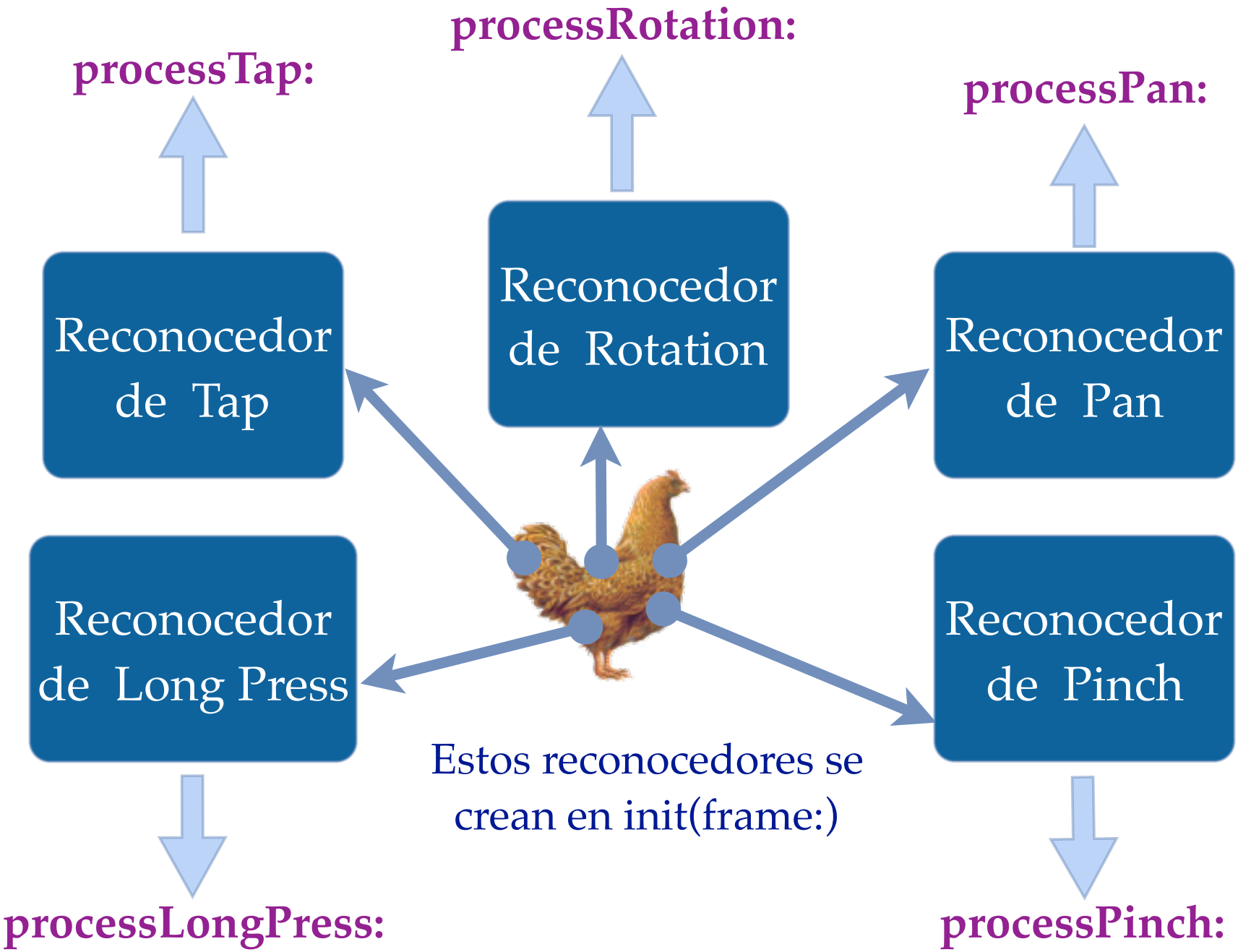
GHPImageView.swift

```
    // En funcion del estado se usa una imagen distinta
    private func updateGHPStateImage() {

        switch ghpState {
        case .HEN:
            image = UIImage(named: "gallina")
        case .EGG:
            image = UIImage(named: "huevo")
        case .CHICKEN:
            image = UIImage(named: "pollo")
        case .FRIED:
            image = UIImage(named: "frito")
        case .ROAST:
            image = UIImage(named: "asado")
        }

        if image != nil {
            bounds.size = image!.size
        }
    }
}
```

GHPImageView.swift

Reconocedores de Gestos de GHPImageView

- Reconocedor de un gesto **Tap**
  - Llama a **processTap** para cambiar de gallina a huevo, de huevo a pollo, y de huevo a gallina.
- Reconocedor de un gesto **Pan**
  - Llama a **processPan** para mover la imagen.
- Reconocedor de un gesto **Pinch**
  - Llama a **processPinch** para reescalar la imagen.
- Reconocedor de un gesto **Rotation**
  - Llama a **processRotation** para girar la imagen.
- Reconocedor de un gesto **Long Press**
  - Llama a **processLongPress** para cambiar de huevo a huevo frito, y de gallina a pollo asado.



Crear Reconocedores

```
// Necesito enterarme de los eventos del usuario
userInteractionEnabled = true

// Crear reconocedor de Tap para ir de gallina a huevo, a pollo, y a gallina
let tapRec = UITapGestureRecognizer(target: self, action: "processTap:")
tapRec.numberOfTapsRequired = 1
addGestureRecognizer(tapRec)

// Crear reconocedor de Long Press para freir huevo
let lpRec = UILongPressGestureRecognizer(target: self, action: "processLongPress:")
addGestureRecognizer(lpRec)

// Crear reconocedor de Pan para mover imagen view
let panRec = UIPanGestureRecognizer(target: self, action: "processPan:")
addGestureRecognizer(panRec)

// Crear reconocedor de Pinch para escalar imagen view
let pinchRec = UIPinchGestureRecognizer(target: self, action: "processPinch:")
addGestureRecognizer(pinchRec)

// Crear reconocedor de Rotation para girar imagen view
let rotationRec = UIRotationGestureRecognizer(target:self,action:"processRotation:")
addGestureRecognizer(rotationRec)
```

Por defecto UIImageView no atiende eventos

ProcessTap

```
func processTap(sender: UITapGestureRecognizer) {

    switch ghpState {
    case .HEN:
        ghpState = .EGG
    case .EGG:
        ghpState = .CHICKEN
    case .CHICKEN:
        ghpState = .HEN
    case .FRIED:
        break
    case .ROAST:
        break
    }
}
```

Pasar de gallina a huevo

Pasar de huevo a pollo

Pasar de pollo a gallina

ProcessLongPress

```
func processLongPress(sender: UILongPressGestureRecognizer) {

    if sender.state != .Began { return }

    switch ghpState {
    case .HEN:
        ghpState = .ROAST
    case .EGG:
        ghpState = .FRIED
    case .CHICKEN:
        break
    case .FRIED:
        break
    case .ROAST:
        break
    }
}
```

Long Press es un gesto continuo. Solo me interesa el comienzo.

Asar gallina

Freír el huevo

ProcessLongPress

```
func processLongPress(sender: UILongPressGestureRecognizer) {

    if sender.state != .Began { return }

    UIView.transitionWithView(self,
            duration: 0.5,
            options: .TransitionFlipFromLeft,
            animations: {
               switch self.ghpState {
                case .HEN:
                    self.ghpState = .ROAST
                case .EGG:
                    self.ghpState = .FRIED
                case .CHICKEN:
                    break
                case .FRIED:
                    break
                case .ROAST:
                    break
                }
        }, completion: nil)
}
```

Igual que la versión anterior pero con una animación.

ProcessPan

```
func processPan(sender: UIPanGestureRecognizer) {
    let pos = sender.translationInView(sender.view!)
    transform = CGAffineTransformTranslate(transform, pos.x, pos.y)
    sender.setTranslation(CGPointZero, inView: sender.view)
}
```

Translación en la vista donde ocurrió el gesto

Resetear el valor en el reconocedor

ProcessPinch

```
func processPinch(sender: UIPinchGestureRecognizer) {

    let factor = sender.scale

    transform = CGAffineTransformScale(transform, factor, factor)

    sender.scale = 1
}
```

Factor de escala indicado por el gesto

Resetear el valor en el reconocedor

ProcessRotation

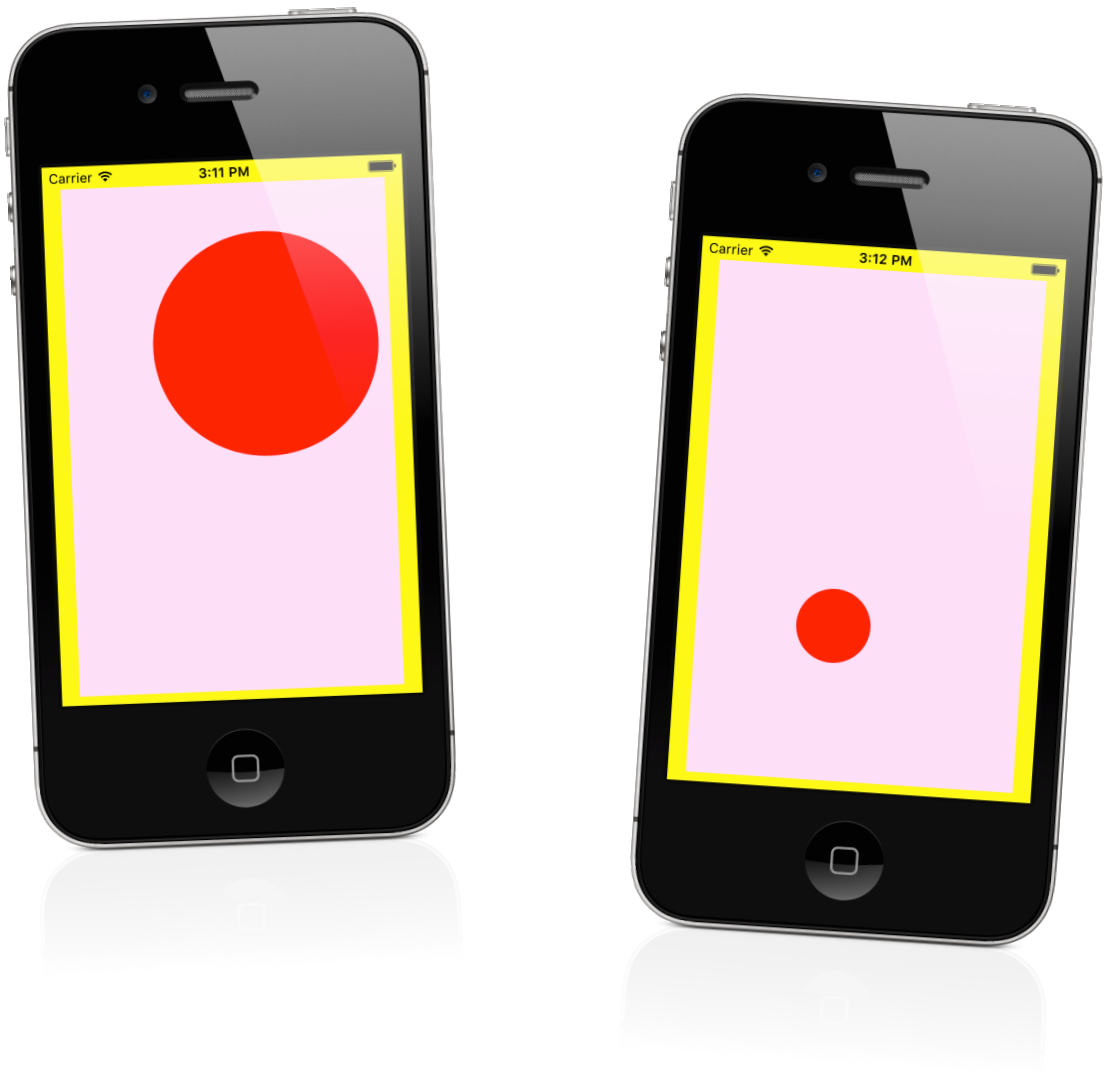
```
func processRotation(sender: UIRotationGestureRecognizer) {
    let angle = sender.rotation
    transform = CGAffineTransformRotate(transform, angle)
    sender.rotation = 0
}
```

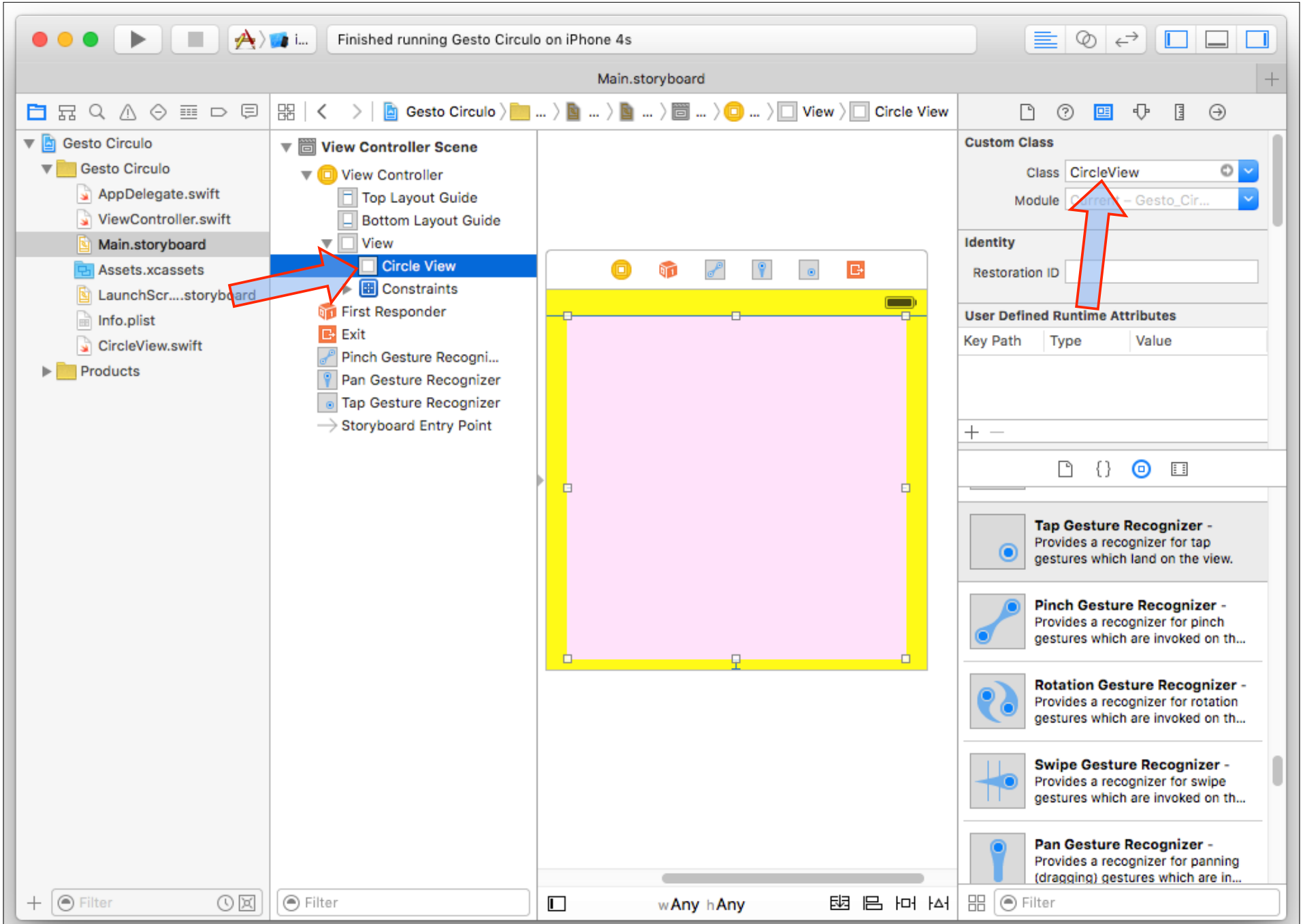
Rotación indicada por el gesto

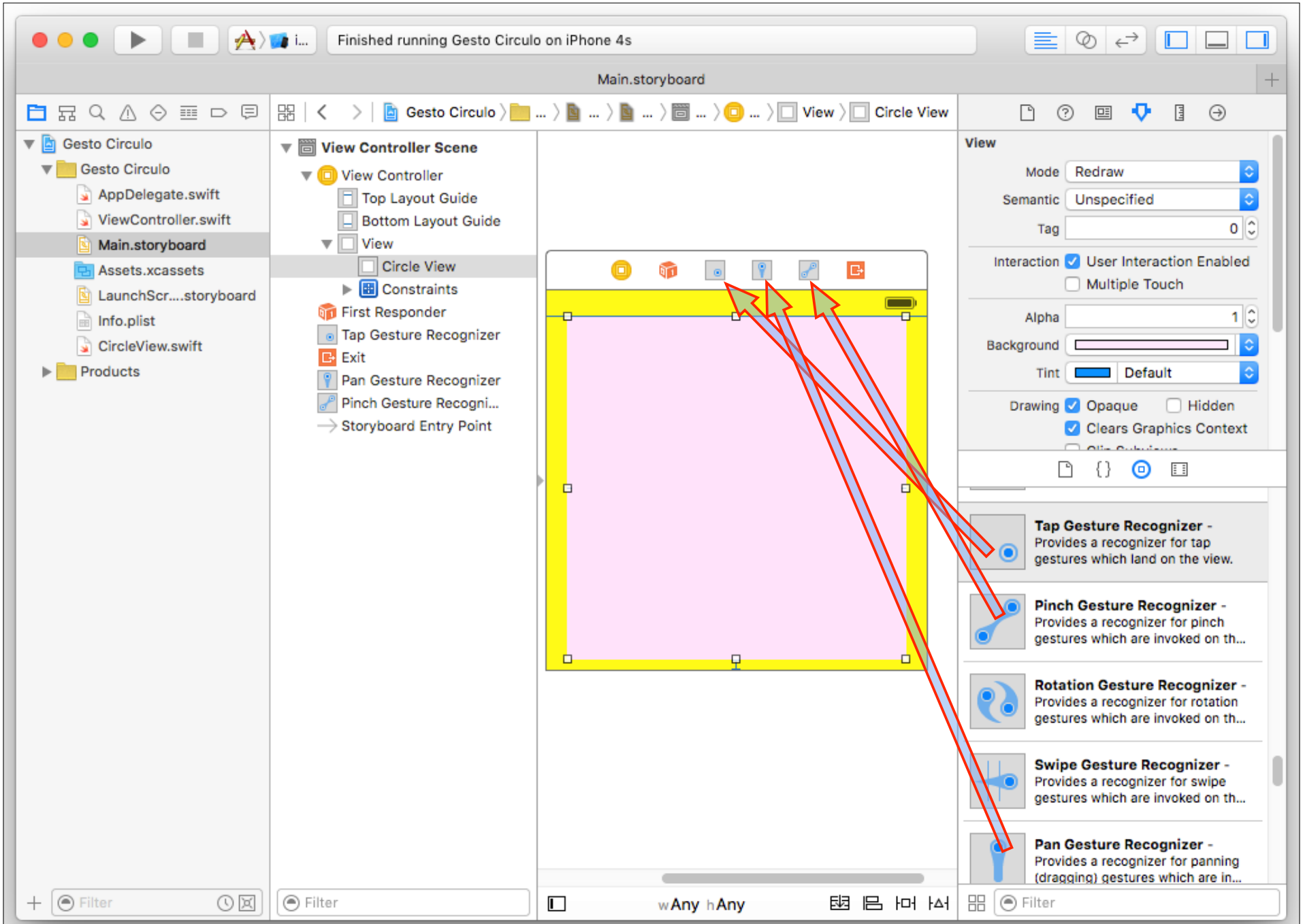
Resetear el valor en el reconocedor

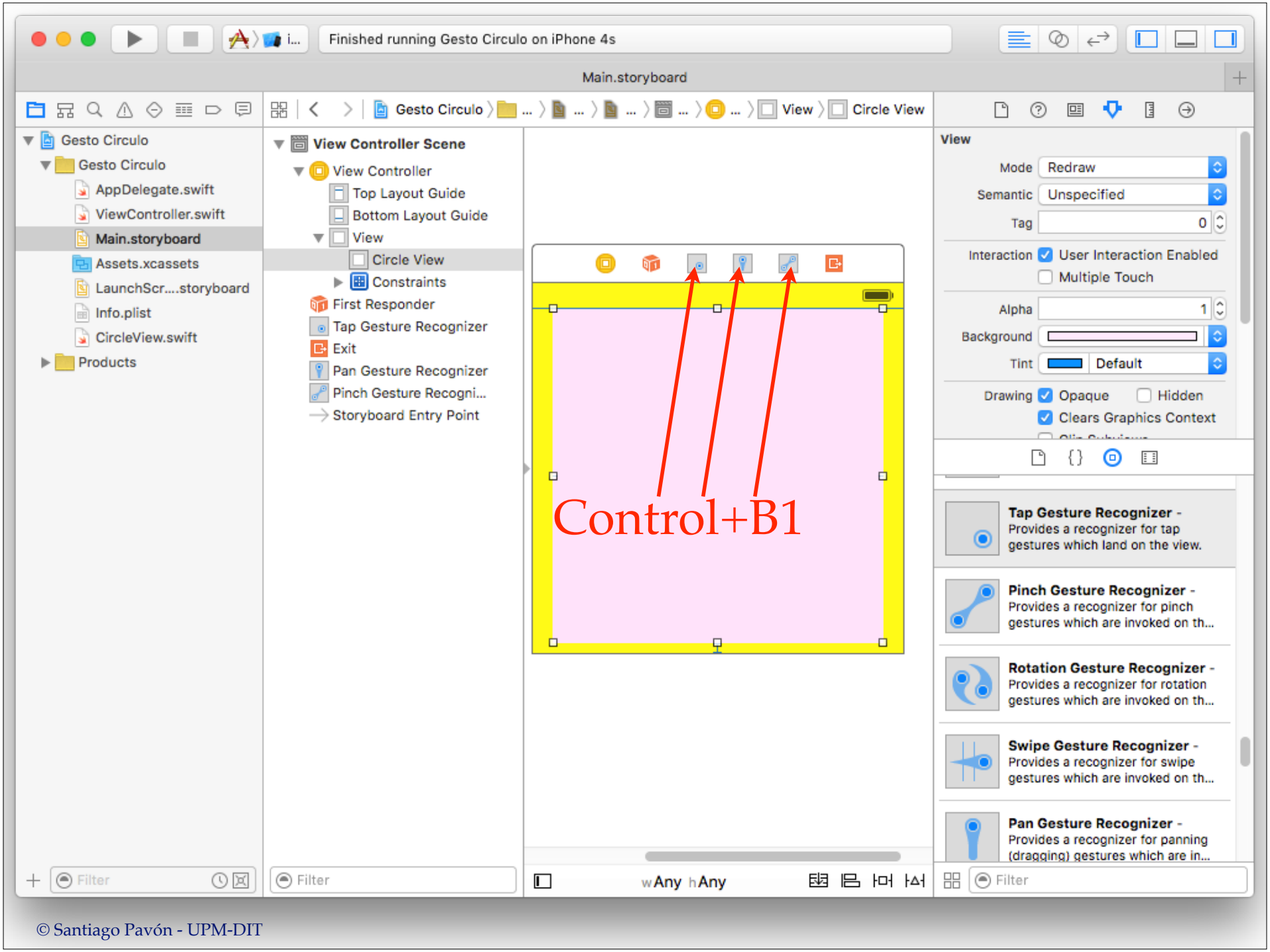
Demo *(IB)*

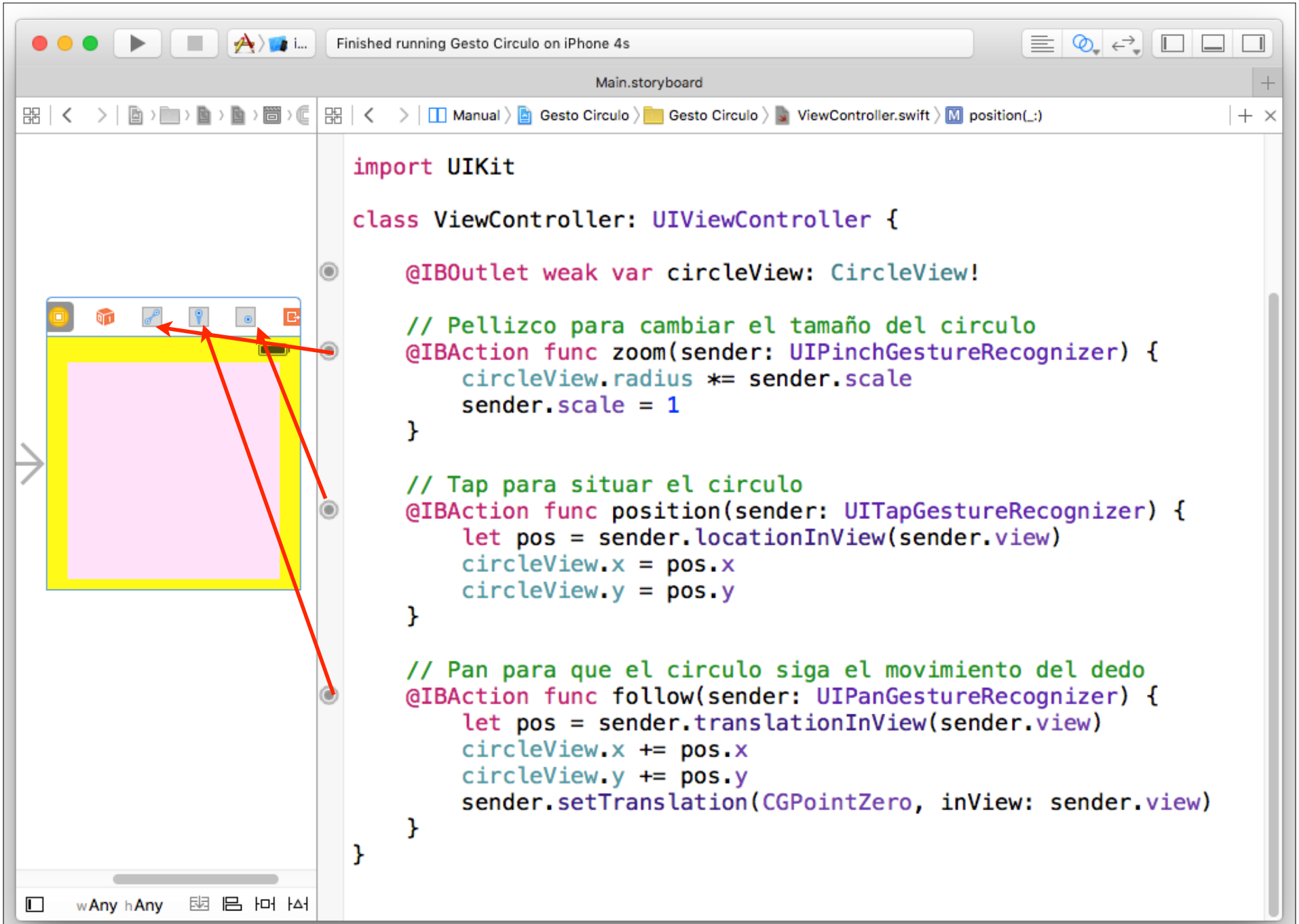
Círculo












```swift
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var circleView: CircleView!

    // Pellizco para cambiar el tamaño del circulo
    @IBAction func zoom(sender: UIPinchGestureRecognizer) {
        circleView.radius *= sender.scale
        sender.scale = 1
    }

    // Tap para situar el circulo
    @IBAction func position(sender: UITapGestureRecognizer) {
        let pos = sender.locationInView(sender.view)
        circleView.x = pos.x
        circleView.y = pos.y
    }

    // Pan para que el circulo siga el movimiento del dedo
    @IBAction func follow(sender: UIPanGestureRecognizer) {
        let pos = sender.translationInView(sender.view)
        circleView.x += pos.x
        circleView.y += pos.y
        sender.setTranslation(CGPointZero, inView: sender.view)
    }
}
```