ETSIT UPM

dit UPM

Desarrollo de Apps para iOS
Reconocedores de Gestos

IWEB 2017-2018

Santiago Pavón

ver: 2017.10.10

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: Set<UITouch>,
                  with event: UIEvent?)

func touchesMoved(_ touches: Set<UITouch>,
                  with event: UIEvent?)

func touchesEnded(_ touches: Set<UITouch>,
                  with event: UIEvent?)

func touchesCancelled(_ touches: Set<UITouch>,
                      with 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 sobrescribir los métodos:
  - `touchesBegan(_ touches:, with event:)`
    - Guardamos la posición inicial
  - `touchedEnded(_ touches:, with event:)`
    - 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: Set<UITouch>, with event: UIEvent?) {
       let touch = touches.first
       initialPoint = touch!.location(in: view)
    }

    override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
        let touch = touches.first
        let currentPoint = touch!.location(in: 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: Any?, 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(_ target: Any?, action: Selector)
func removeTarget(_ target: Any?, action: Selector?)
```

- Información sobre el gesto:

```
func location(in view: UIView?) -> CGPoint
func location(ofTouch touchIndex: Int, in view: UIView?) -> CGPoint
var numberOfTouches: Int {get}
```

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

- Más propiedades:
  - `state`
  - `view`
  - `isEnabled`
  - `delegate`
  - . . .
- Dependencias entre gestos:

```
func require(toFail otherGestureRecognizer: 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(_ sender: 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.

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

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

@objc func processTap(_ sender: UITapGestureRecognizer) {
    let pos = sender.location(in: sender.view)
    print("TAP en x=\(pos.x) y=\(pos.y)")
}

// Nota: Con Swift 4 es necesario añadir @objc.
// La inferencia de @objc de Swift 3 se ha deprecado en Swift 4.
```

Reconocedor de taps

Ejecutar en **self** el método **processTap**

Reconocer el gesto en la top view del VC

El reconocedor

pos es donde se pulsó en la view

En este ejemplo sender.view es self.view

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**.
  - Valor **acumulado** desde que empezó el gesto.
- Podemos asignarle un valor.
  - por ejemplo, para resetear o poner a 1 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**.
  - Valor **acumulado** desde que empezó el gesto.
- Podemos asignarle un valor.
  - por ejemplo, para resetear o poner a 0 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 translation(in view: UIView?) -> CGPoint
```

- Distancia que se ha movido el dedo en la view dada.
  - Valor **acumulado** desde que empezó el gesto.

```
func setTranslation(_ translation: CGPoint, in view: UIView?)
```

- Resetear o cambiar el valor acumulado.

```
func velocity(in view: 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:

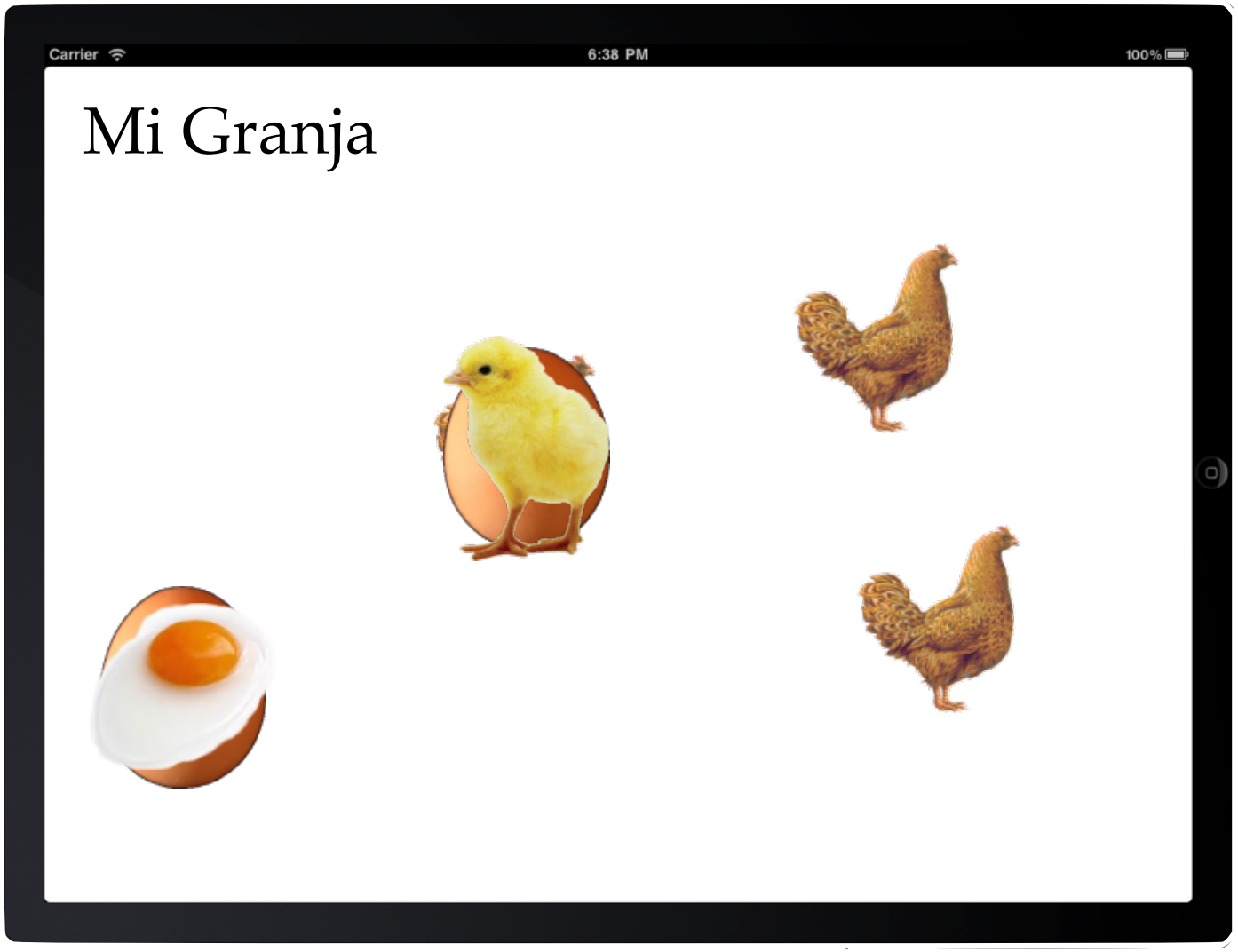
```
var edges: UIRectEdge
```

  - El valor de **edges** es un conjunto de los siguientes valores:
    - **.all**, **.top**, **.left**, **.bottom** y **.right**.
- En iOS 11, se ha creado en UIViewController el método `preferredScreenEdgesDeferringSystemGestures() -> UIRectEdge`
  - para configurar los lados en los que este gesto tiene prioridad sobre los definidos en el sistema.
    - Por ejemplo, para evitar que al hacer swipe desde abajo aparezca el Control Center, y conseguir que se dispare el gesto que hayamos programado en nuestra app.

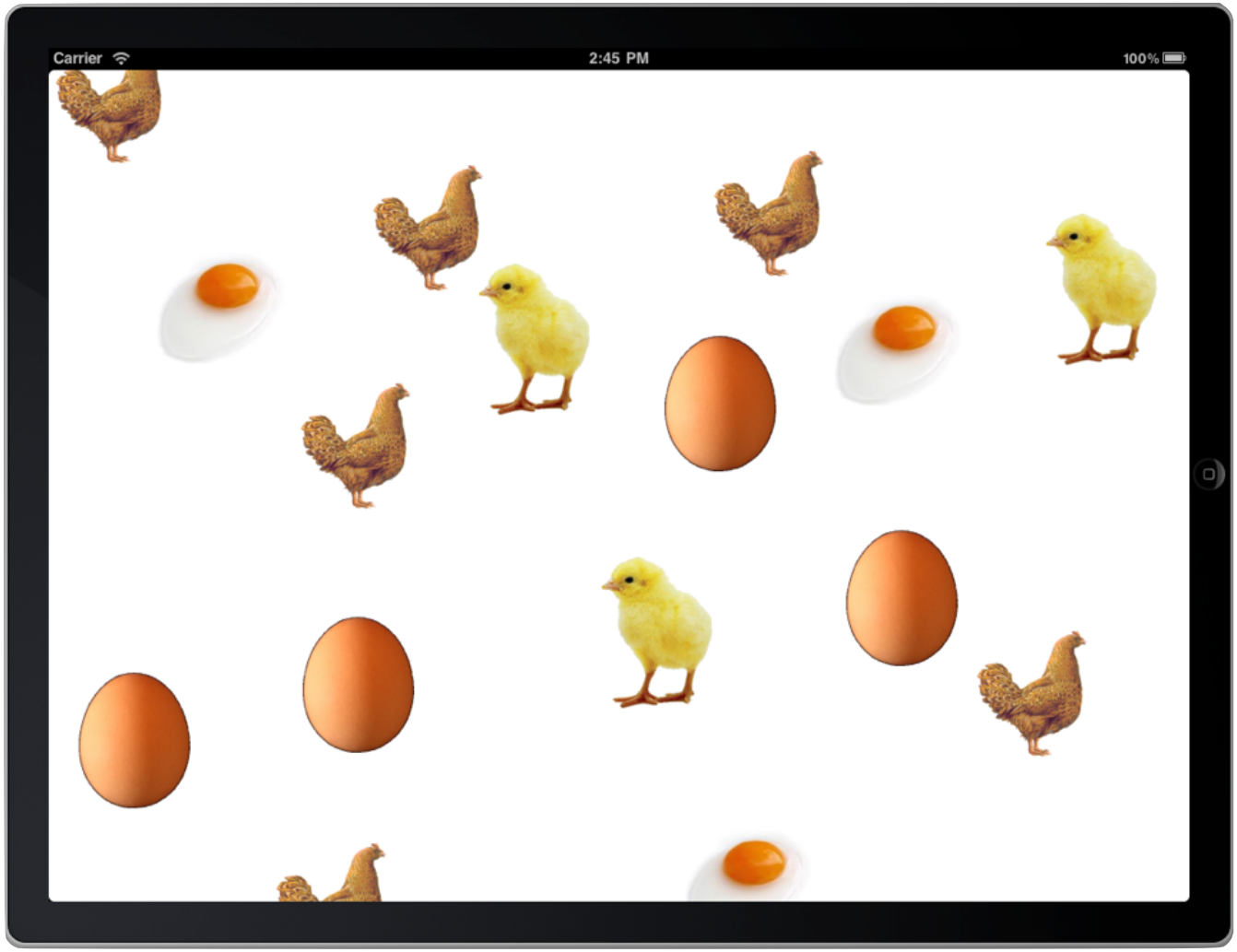
Demo *(Programático)*

La Granja

Demo

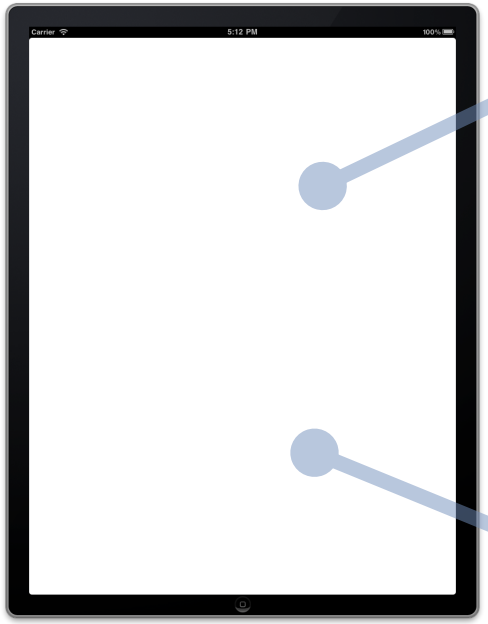


Demo



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: #selector(processRootTap(_:)))
    tapRec.numberOfTapsRequired = 1
    view.addGestureRecognizer(tapRec)

    // Crear reconocedor de SreenEdgePan para borrar todo
    let edgeRec = UIScreenEdgePanGestureRecognizer(
                        target: self,
                        action: #selector(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):

```
@objc func processRootTap(_ sender: UITapGestureRecognizer) {

    let pos = sender.location(in: sender.view)

    let rect = CGRect(x: pos.x, y: pos.y, width: 1, height: 1)

    let imgv = GHPImageView(frame: rect)

    view.addSubview(imgv)
}

// Nota: Con Swift 4 es necesario añadir @objc.
// La inferencia de @objc de Swift 3 se ha deprecado en Swift 4.
```

Borrar el Fondo

```
@objc 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

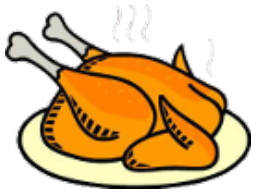
```
// Nota: Con Swift 4 es necesario añadir @objc.
// La inferencia de @objc de Swift 3 se ha deprecado en Swift 4.
```

```
@objc func processRootEdge(_ sender: UIScreenEdgePanGestureRecognizer) {

    if sender.state != .began { return }

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

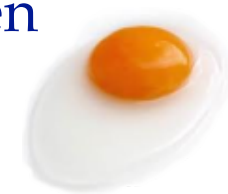
Igual que la versión anterior pero con una animación.

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")
    }
   . . .
```

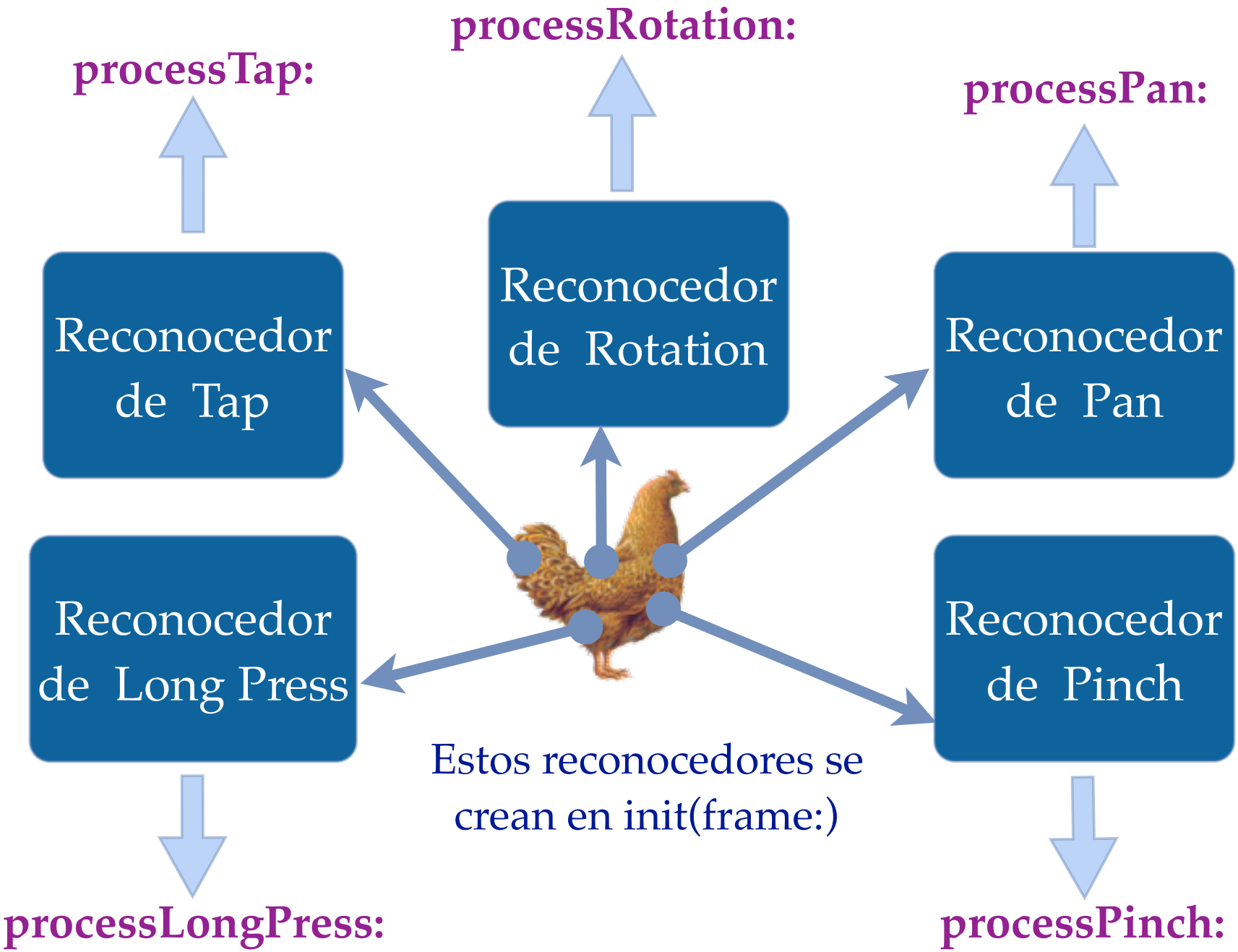
GHPImageView.swift

```
    // 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

- Añadir en init

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

Por defecto UIImageView no atiende eventos

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


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

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


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


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

ProcessTap

```
@objc func processTap(_ sender: UITapGestureRecognizer) {

    switch ghpState {
    case .hen:
        ghpState = .egg
    case .egg:
        ghpState = .chicken
    case .chicken:
        ghpState = .hen
    case .fried:
        break
    case .roast:
        break
    }
 }
// Nota: Con Swift 4 es necesario añadir @objc.
// La inferencia de @objc de Swift 3 se ha deprecado en Swift 4.
```

Pasar de gallina a huevo

Pasar de huevo a pollo

Pasar de pollo a gallina

ProcessLongPress

```
@objc 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
    }
}
// Nota: Con Swift 4 es necesario añadir @objc.
// La inferencia de @objc de Swift 3 se ha deprecado en Swift 4.
```

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

Asar gallina

Freír el huevo

ProcessLongPress

```
@objc func processLongPress(sender: UILongPressGestureRecognizer) {

    if sender.state != .Began { return }

    UIView.transition(with: 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

```
@objc func processPan(_ sender: UIPanGestureRecognizer) {
    let pos = sender.translation(in: sender.view!)
    transform = transform.translatedBy(x: pos.x, y: pos.y)
    sender.setTranslation(CGPoint.zero, in: sender.view)
}
```

Translación en la vista donde ocurrió el gesto

Resetear el valor en el reconocedor

```
// Nota: Con Swift 4 es necesario añadir @objc.
// La inferencia de @objc de Swift 3 se ha deprecado en Swift 4.
```

ProcessPinch

```
@objc func processPinch(_ sender: UIPinchGestureRecognizer) {
    let factor = sender.scale
    transform = transform.scaledBy(x: factor, y: factor)
    sender.scale = 1
}
```

Factor de escala indicado por el gesto

Resetear el valor en el reconocedor

```
// Nota: Con Swift 4 es necesario añadir @objc.
// La inferencia de @objc de Swift 3 se ha deprecado en Swift 4.
```

ProcessRotation

```
@objc func processRotation(_ sender: UIRotationGestureRecognizer) {
    let angle = sender.rotation
    transform = transform.rotated(by: angle)
    sender.rotation = 0
}
```

Rotación indicada por el gesto

Resetear el valor en el reconocedor

```
// Nota: Con Swift 4 es necesario añadir @objc.
// La inferencia de @objc de Swift 3 se ha deprecado en Swift 4.
```

Demo *(IB)*

Círculo