Desarrollo de Apps para iOS
`UIViews`

IWEB 2018-2019
Santiago Pavón

ver: 2018.09.18

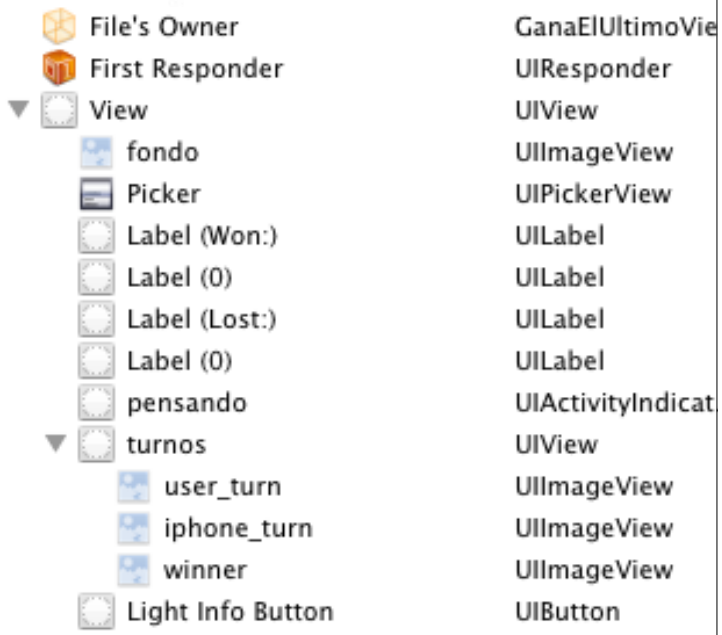
© Santiago Pavón - UPM-DIT

UIView

© Santiago Pavón - UPM-DIT

¿Qué es una UIView?

- Clase base de la que derivan las views usadas para crear un GUI.
- El GUI es una jerarquía (árbol) de views.
- Propiedades para recorrer el árbol:
  - La view padre:
    ```
    superview: UIView?
    ```
  - Array con las views hijas:
    ```
    subviews: [UIView]
    ```



- La raíz de la jerarquía de views es un objeto **`UIWindow`**.

© Santiago Pavón - UPM-DIT

Construir el GUI

- La jerarquía de views puede construirse con el Interface Builder de Xcode.
- O programáticamente:

Inicializador designado de `UIView`.

```
init(frame: CGRect)
```

Crear jerarquía de views.

```
func addSubview(_ view: UIView)
func removeFromSuperview()

func insertSubview(_ view: UIView, at index: Int)
func insertSubview(_ view: UIView, belowSubview siblingSubview: UIView)
func insertSubview(_ view: UIView, aboveSubview siblingSubview: UIView)
```

© Santiago Pavón - UPM-DIT

```
let rect = CGRect(x: 50, y: 50, width: 120, height: 40)

let l = UILabel(frame:rect)

l.text = "Hola mundo"

view.addSubview(l)
```

© Santiago Pavón - UPM-DIT

El contenido de la View

- La view define un área rectangular en el que se puede dibujar.
- Si el contenido sobrepasa el rectángulo, se puede pintar o no (**`clipToBounds`**).
- Se puede controlar su nivel de transparencia (**`alpha`**).
- Puede ser opaca o no (**`isOpaque`**).
- La view entera se puede ocultar (**`isHidden`**).
- Rehacer el contenido de una view cuando cambia su tamaño (**`contentMode`**):
  - reusar imagen cacheada con el contenido actual reajustando tamaño y posición,
  - repintar otra vez el contenido.
- Transformaciones afines (**`transform`**).
- Su color de fondo (**`backgroundColor`**).
- etc.
  - Ver la documentación

© Santiago Pavón - UPM-DIT

Eventos

- Podemos gestionar los eventos que ocurren en la view.
- Se puede configurar para:
  - ignorar los eventos del usuario.
  - decidir si soportar multi-touch.
  - establecer reconocedores de gestos.
  - bloquear el envío de eventos a otras views.
  - etc.

© Santiago Pavón - UPM-DIT

Gestión de Memoria

- Al añadir una view a la jerarquía se retiene (strong), aumentando su contador de retenciones.
  - No es necesario que nosotros retengamos la view con una nueva propiedad.
- Pero al sacar a view de la jerarquía se llama a release, es decir, se disminuye en uno el valor de su contador de retenciones.
  - Cuidado que el contador de retenciones podría llegar a cero en este momento.
    - Podemos evitarlo asignando previamente la view a una variable strong.

© Santiago Pavón - UPM-DIT

Coordenadas

© Santiago Pavón - UPM-DIT

Algunos Tipos **CoreGraphics**

- **`CGFloat`**
  - Es una struct que contiene un número real.
- **`CGPoint`**
  - Es una struct con las propiedades: **`x:CGFloat, y:CGFloat`**.
    - Representa un punto situado en (x,y).
- **`CGSize`**
  - Es una struct con las propiedades: **`width:CGFloat, height:CGFloat`**.
    - Representa un tamaño.
- **`CGRect`**
  - Es una struct con las propiedades: **`origin:CGPoint, size:CGSize`**.
    - Representa un rectángulo con el origen y tamaño indicados.

- Para crear instancias de estas estructuras (*tipos con semántica valor*):

```
let p: CGPoint = CGPoint(x: 0, y: 0)
let s: CGSize  = CGSize(width: 10, height: 20)
let r: CGRect  = CGRect(origin: p, size: s)
let r2: CGRect = CGRect(x: 0, y: 0, width: 10, height: 20)
```

© Santiago Pavón - UPM-DIT

Coordenadas

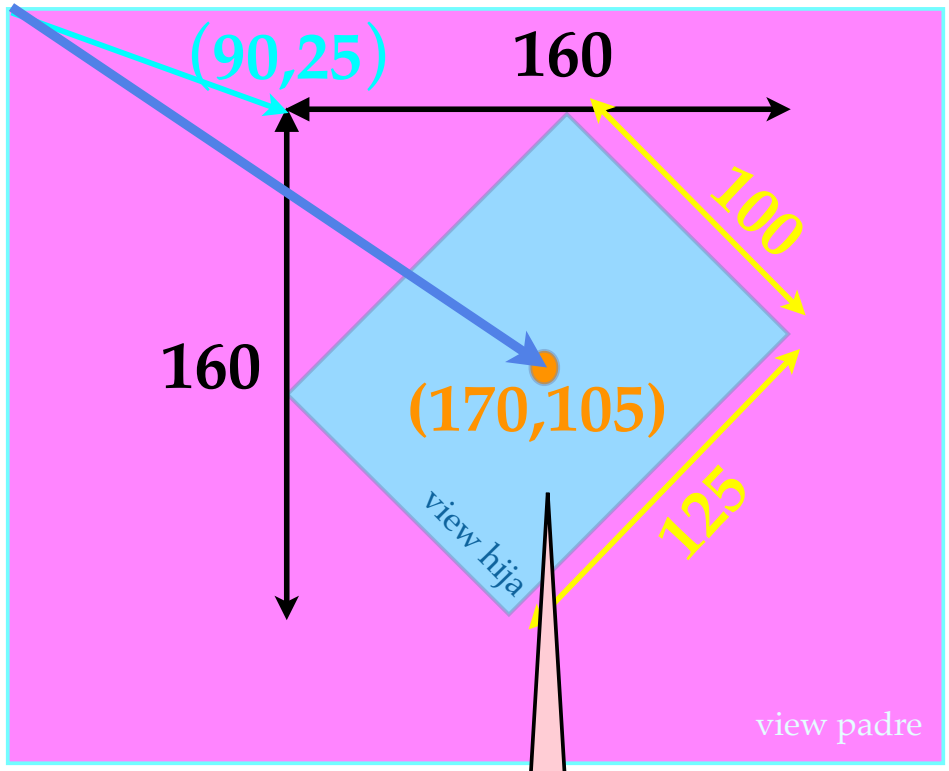
- El origen de coordenadas está situado arriba a la izquierda
- Las unidades son puntos, no píxeles.
  - Normalmente usamos puntos cuando dibujamos.
    - El dibujo se ajusta automáticamente a la mayor resolución.
      - la propiedad **`contentScaleFactor`** contiene número de pixels por punto de la `UIView`.
- Propiedades:

  `bounds:` `CGRect`
  - El origen y tamaño de la view en su sistema de coordenadas.

  `center:` `CGPoint`
  - El centro de la view en el sistema de coordenadas de la view padre.

  `frame:` `CGRect`
  - Rectángulo que contiene completamente a la view en el sistema de coordenadas de su view padre (la view hija puede estar girada).

© Santiago Pavón - UPM-DIT



View hija:

```
bounds = ((0,0),(100,125))
frame  = (90,25),(160,160))
center = (170,105)
```

© Santiago Pavón - UPM-DIT

`UIView`: Dibujar

© Santiago Pavón - UPM-DIT

Alternativas para Dibujar

- Podemos usar:
  - **Quartz 2D**
    - Pertenece al framework Core Graphics.
      - Consultar: *Quartz 2D Programming Guide*.
      - API con funciones C adaptadas para que tengan un estilo *Swiftero*.
    - El framework UIKit proporciona varios envoltorios para facilitar su uso.
      - `UIColor`, `UIFont`, `UIBezierPath`, ...
  - **OpenGL ES**
  - **Metal**

© Santiago Pavón - UPM-DIT

Dibujar con Quartz 2D

- Para hacer dibujos personales en la pantalla deberemos crear una subclase de **`UIView`**.
- Para dibujar hay que redefinir el método:

```
func draw(_ rect: CGRect)
```

  - Incluyendo en él las sentencias de pintado.
  - El rectángulo pasado como argumento indica la zona que hay que repintar.
- Nunca llamaremos directamente a **`draw`**
  - Cuando haya que repintar el contenido de la UIView, llamaremos a uno de estos métodos:

```
func setNeedsDisplay()
func setNeedsDisplay(_ rect: CGRect)
```

    - Y el sistema llamará a `draw` en el siguiente ciclo de pintado.

© Santiago Pavón - UPM-DIT

Path

- Un path es un dibujo creado con líneas rectas y curvas, arcos, elipses y rectángulos.
  - Puede ser abierto o cerrado, relleno o no, con líneas sólidas o discontinuas, etc.
- Como se usa:
  - Crear un path.
  - Establecer el punto inicial de pintado.
  - Añadimos tramos al path (líneas, arcos, ...).
    - Al final, podemos (opcional) cerrar el path.
  - Definir el color/tipo/patrón de las líneas, el color del relleno, zona de clip, etc...
  - Acabamos ordenando que se pinte el path creado.
    - En ese momento se pintan las líneas que lo forman, el relleno o ambos.
- Un path también puede guardarse para reusarlo varias veces.

© Santiago Pavón - UPM-DIT

UIKit Envoltorios

- UIKit proporcionan métodos útiles para simplificar el código de pintado que hay que programar:
  - Establecer el color de las líneas y de relleno:

```
UIColor.blue.set()
```

  - Establecer el color de las líneas:

```
UIColor.green.setStroke]()
```

  - Establecer el color de relleno:

```
UIColor.red.setFill()
```

  - Crear un color personalizado:

```
let c = UIColor(red:0.2, green:0.1, blue:0.8, alpha:1.0)
```

© Santiago Pavón - UPM-DIT

- Establecer el font:

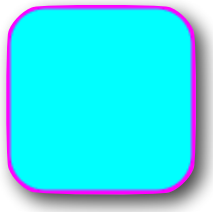
```
let font1 = UIFont.systemFont(ofSize: 20)

let font2 = UIFont.systemFontSize

let font3 = UIFont(name: "HelveticaNeue-Thin", size: 24)
```

- Dibujar un rectángulo, relleno o solo el contorno:

```
let rect = CGRect(x: 10, y: 10, width: 100, height: 100)

UIRectFill(rect)
UIRectFrame(rect)
```

- Dibujar un path (rectángulo redondeado):

```
let path = UIBezierPath(roundedRect: rect, cornerRadius: 20)

path.lineWidth = 3;
UIColor.magenta.setStroke()
UIColor.cyan.setFill()
path.fill()
path.stroke()
```



© Santiago Pavón - UPM-DIT

- Dibujar un path (una copa) con UIBezierPath:

```
let path = UIBezierPath()

path.move(to: CGPoint(x: 10, y: 10))

path.addArc(withCenter: CGPoint(x: 90, y: 10),
            radius: 80,
            startAngle: CGFloat(M_PI),
            endAngle: CGFloat(M_PI_2),
            clockwise: false)
path.addLine(to: CGPoint(x: 95, y: 180))
path.addLine(to: CGPoint(x: 40, y: 200))
path.addLine(to: CGPoint(x: 170, y: 200))
path.addLine(to: CGPoint(x: 115, y: 180))
path.addLine(to: CGPoint(x: 120, y: 90))
path.addArc(withCenter: CGPoint(x: 120, y: 10),
            radius: 80,
            startAngle: CGFloat(M_PI_2),
            endAngle: CGFloat(0),
            clockwise: false)
path.close()

path.lineWidth = 3
UIColor.magenta.setStroke()
UIColor.cyan.setFill()
path.fill()
path.stroke()
```

© Santiago Pavón - UPM-DIT

Dibujar Texto

```
let text = "hola"

let attrs = [NSAttributedStringKey.font: UIFont.systemFont(ofSize: 28)]


let pos = CGPoint(x: 10, y: 10)

text.draw(at: pos, withAttributes: attrs)
```

hola

```
let rect = CGRect(x: 20, y: 40, width: 30, height: 200)

text.draw(in: rect, withAttributes: attrs)
```

h
ol
a

- Es mejor usar un objeto UILabel.

© Santiago Pavón - UPM-DIT

Dibujar Imágenes

```
if let img = UIImage(named: "fondo.png") {
    img.draw(at: pos)
    img.draw(in: rect)
    img.drawAsPattern(in: rect)
}
```

Crear imagen.

Pintar la imagen en la posición dada.

Reescalar imagen y pintarla en el rectángulo dado.

Pintar la imagen repetida (tiled) para llenar el rectángulo dado.

- Es mejor usar un objeto `UIImageView`.

© Santiago Pavón - UPM-DIT

Cambios de tamaño

- Cuando cambia el tamaño de una view:
  - Si el valor de **`contentMode`** es **`UIViewContentMode.redraw`**, se redibuja su contenido otra vez.
  - Para otro valor de **`contentMode`** se repinta el contenido usando un bitmap cacheado que contiene la imagen actual.
    - Se reescala y se pinta el bitmap si el valor de **`contentMode`** es **`.scaleToFill`**, **`.scaleAspectFit`**, **`.scaleAspectFill`**.
    - O se dibuja el bitmap en la posición indicada si el valor de **`contentMode`** es:

      `.center .left .right .top .right`

      `.bottomLeft .bottomRight .topLeft .topRight`

© Santiago Pavón - UPM-DIT

Crear UIImages

- Usando un fichero en el main bundle de la aplicación:

```
let img = UIImage(named:"fondo.png")
```

- Usando el path de un fichero:

```
let paths = NSSearchPathForDirectoriesInDomains(.documentDirectory,
                                                .userDomainMask, true)
let docs = paths[0] as NSString
let filePath = docs.appendingPathComponent("mundo.jpg")
let img = UIImage(contentsOfFile: filePath)
```

- Desde un buffer de bytes:

```
if let url = URL(string: "http://www.sitio.es/~fotos/face.jpg"),
   let data = try? Data(contentsOf: url, options: []) {
    let img = UIImage(data: data)
}
```

- Dibujándola en un contexto gráfico:

```
UIGraphicsBeginImageContext(CGSize(width: 200, height: 200))
// dibujar aqui usando funciones CGContext o UIBezierPath
let img = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()
```

© Santiago Pavón - UPM-DIT

Crear JPEG y PNG

- El contenido de una `UIImage` puede guardarse en un `Data` en formato JPEG o PNG.

```
func UIImageJPEGRepresentation(_ image: UIImage,
                               _ compressionQuality: CGFloat)
                                    -> Data?

func UIImagePNGRepresentation(_ image: UIImage)
                                    -> Data?
```

© Santiago Pavón - UPM-DIT

Contexto Gráfico

- El pintado se hace usando un contexto gráfico.
- Independiza del elemento sobre el que se pinta
  - pantalla, imagen, impresora, PDF, …
- Se obtiene usando el método:

```
func UIGraphicsGetCurrentContext() -> CGContext?
```

  - Cada vez que se llama a `draw` se crea automáticamente un nuevo contexto gráfico.
    - Ese contexto sólo es válido para la actual llamada a `draw`.
    - No se puede guardar para usarlo más tarde.

© Santiago Pavón - UPM-DIT

Dibujar una Copa

```
override func draw(_ rect: CGRect) {
   let context = UIGraphicsGetCurrentContext()!

    context.beginPath()
    context.move(to: CGPoint(x: 10, y: 10))
    context.addArc(center: CGPoint(x: 90, y: 10),
                   radius: 80,
                   startAngle: CGFloat(M_PI),
                   endAngle: CGFloat(M_PI_2),
                   clockwise: true)
    context.addLine(to: CGPoint(x: 95, y: 180))
    context.addLine(to: CGPoint(x: 40, y: 200))
    context.addLine(to: CGPoint(x: 170, y: 200))
    context.addLine(to: CGPoint(x: 115, y: 180))
    context.addLine(to: CGPoint(x: 120, y: 90))
    context.addArc(center: CGPoint(x: 120, y: 10),
                   radius: 80,
                   startAngle: CGFloat(M_PI_2),
                   endAngle: CGFloat(0),
                   clockwise: true)
    context.closePath()

    context.setLineWidth(3)
    context.setStrokeColor(UIColor.magenta.cgColor)
    context.setFillColor(UIColor.cyan.cgColor)
    context.drawPath(using: .fillStroke)
}
```

© Santiago Pavón - UPM-DIT

Push y Pop del Contexto Gráfico

- Si se realizan modificaciones al estado del contexto gráfico, pero no se quiere perder el estado actual:
  - puede guardarse el estado actual del contexto gráfico con:

```
context.saveGState()
```

  - realizar las modificaciones que se deseen en el contexto gráfico
  - y recuperar el estado inicial con:

```
context.restoreGState()
```

- Ejemplo de uso:
  - `draw` invoca un método auxiliar que cambia el estado del contexto para hacer ciertos dibujos.
  - Este método hace un push al principio y un pop al final para que los cambios del estado sean locales a él.

© Santiago Pavón - UPM-DIT

```
override func draw(_ rect: CGRect) {
    drawAxis()
    drawObject()
}

private func drawAxis() {

    let context = UIGraphicsGetCurrentContext()

    // Guardar el estado del contexto actual
    context.saveGState()

    // Realizo cambios en el estado del contexto
    context.setLineWidth(1)
    context.setStrokeColorWithColor(UIColor.red.cgColor)

    // Dibujar aqui

    // Recupero el estado inicial
    context.restoreGState()
}

private func drawObject() {
    // . . .
}
```

Los cambios realizados por este método en el contexto gráfico no afectan a siguiente método

© Santiago Pavón - UPM-DIT

Inicialización de UIView

© Santiago Pavón - UPM-DIT

Inicialización

- Existen varios inicializadores de UIView y sus subclases:

  `init(frame: CGRect)`

  - Es el inicializador que llamamos (*típicamente*) explícitamente nosotros al crear un objeto UIView programáticamente.
  - Este inicializador los redefiniremos en nuestras clases derivadas de UIView para añadir las sentencias de configuración que necesitemos para las nuevas vistas.

  `init?(coder: NSCoder)`

  - Es el inicializador que se llama al reconstruir un objeto serializado.
    - Es decir, es el inicializador que se llama al cargar las views desde un fichero storyboard o nib.
  - Especificado en el protocolo NSCoding.
  - **IMPORTANTE**: Cuando se usa este inicializador, no se invoca el otro.
  - Si tenemos que realizar alguna configuración personalizada adicional, podemos hacerla en el método **`awakeFromNib`**.
    - Este método se invoca en todos los objetos cargados de un fichero NIB o storyboard.

© Santiago Pavón - UPM-DIT

- Patrón típico para no repetir el mismo código de configuración en el método **`init(frame:)`** y en el método **`awakeFromNib`**:

```
class MyView: UIView {

    override init(frame: CGRect) {
        super.init(frame: frame)
        setup()
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
    }

    override func awakeFromNib() {
        super.awakeFromNib()
        setup()
    }

    private func setup() {
        // Configuracion adicional.
    }
}
```

Así, el método **setup** se ejecutará siempre, independientemente de como creemos el objeto UIView

© Santiago Pavón - UPM-DIT

Threads

© Santiago Pavón - UPM-DIT

Main Thread

- La manipulación de la interface de usuario debe hacerse sólo desde el main thread de la aplicación.
- Los métodos de UIView (y de otras clases de UIKit) sólo pueden invocarse desde el main thread.
- Únicamente la creación de los objetos UIView puede hacerse desde otro thread.

© Santiago Pavón - UPM-DIT

Visualización en Vivo de la UIView en Storyboard

© Santiago Pavón - UPM-DIT

Visualizar UIView

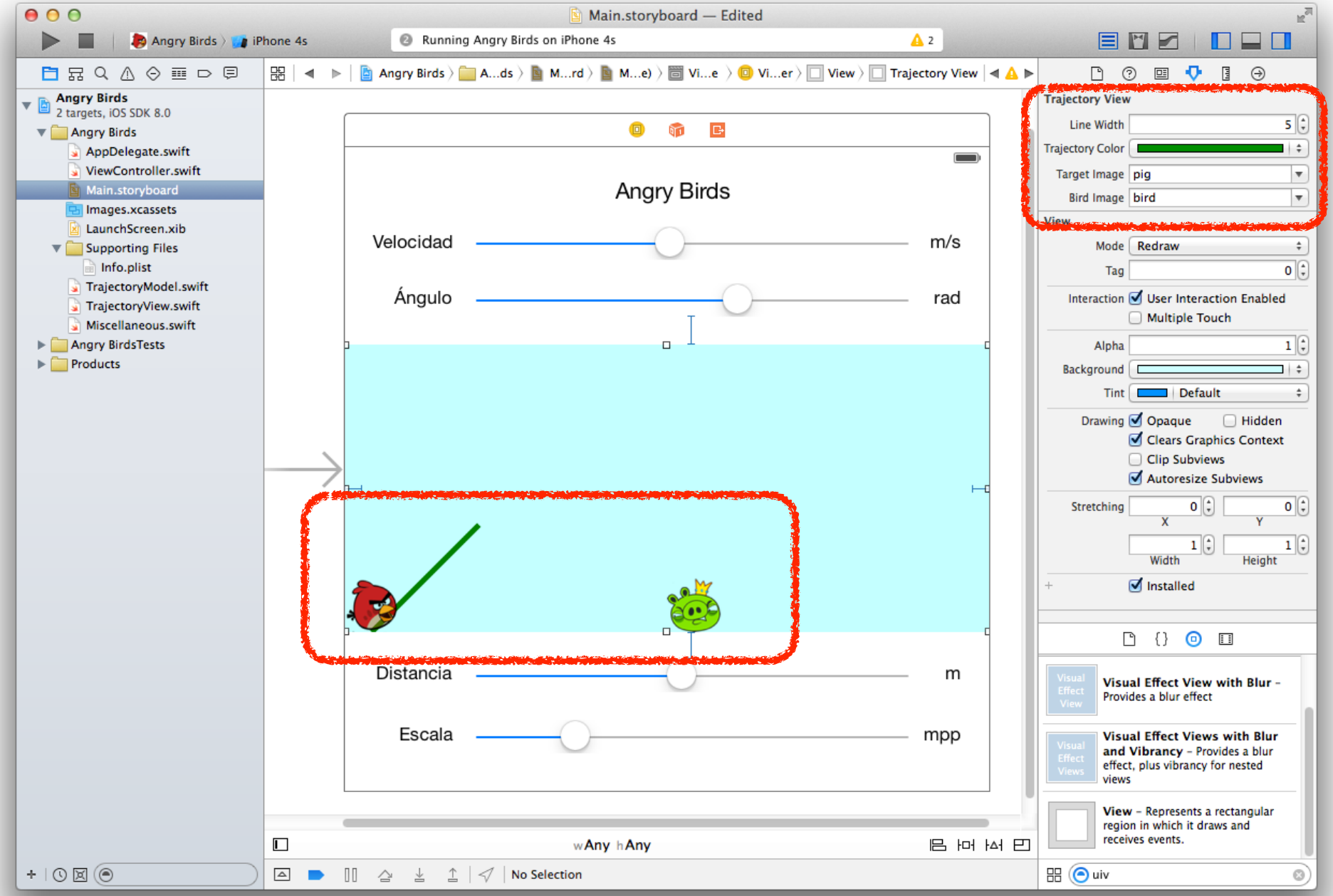
- Para visualizar en vivo el contenido de una UIView personalizada en el storyboard, marcar la clase con **`@IBDesignable`**.
- Los atributos que se deseen ajustar desde el inspector de atributos deben marcarse con **`@IBInspectable`**.
- Si es necesario realizar una configuración especial para que la visualización en el IB sea la adecuada, sobrescribir el método **`prepareForInterfaceBuilder`**.
- También puede incluirse código condicional para que solo se ejecute cuando el IB está presentado la view. Usando la directiva:

```
#if TARGET_INTERFACE_BUILDER
#else
#endif
```

© Santiago Pavón - UPM-DIT

```
@IBDesignable
class TrajectoryView: UIView {

    @IBInspectable
    var lineWidth : Double = 3.0

    @IBInspectable
    var trajectoryColor : UIColor = UIColor.redColor()

    @IBInspectable
    var targetImage : UIImage!

    @IBInspectable
    var birdImage : UIImage!

    // . . .
    override func prepareForInterfaceBuilder() {
        dataSource = FakeDataSource()

    }

    // . . .
```

© Santiago Pavón - UPM-DIT



© Santiago Pavón - UPM-DIT

© Santiago Pavón - UPM-DIT