



POLITÉCNICA

ETSIT
UPM

dit
UPM

Desarrollo de Apps para iOS UIViews

IWEB 2019-2020
Santiago Pavón

ver: 2019.11.03

¿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:

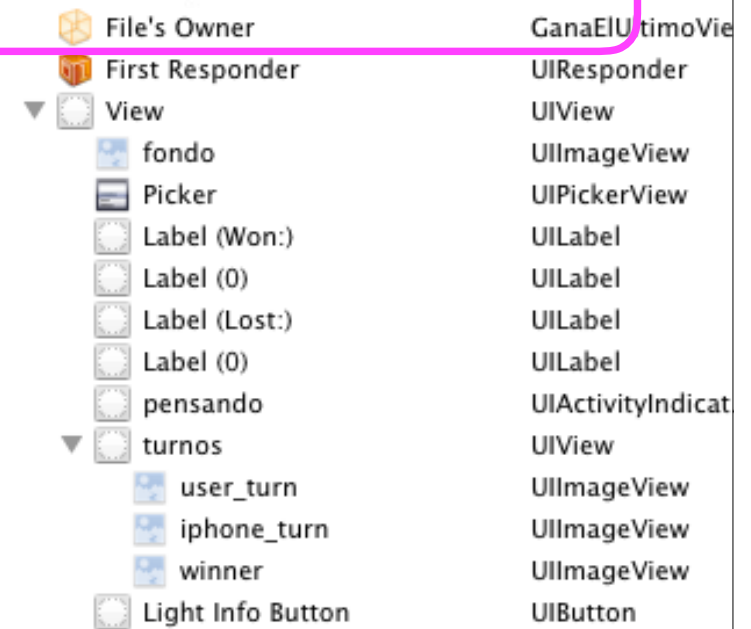
- La view padre:

superview: **UIView?**

- Array con las views hijas:

subviews: **[UIView]**

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



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

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**

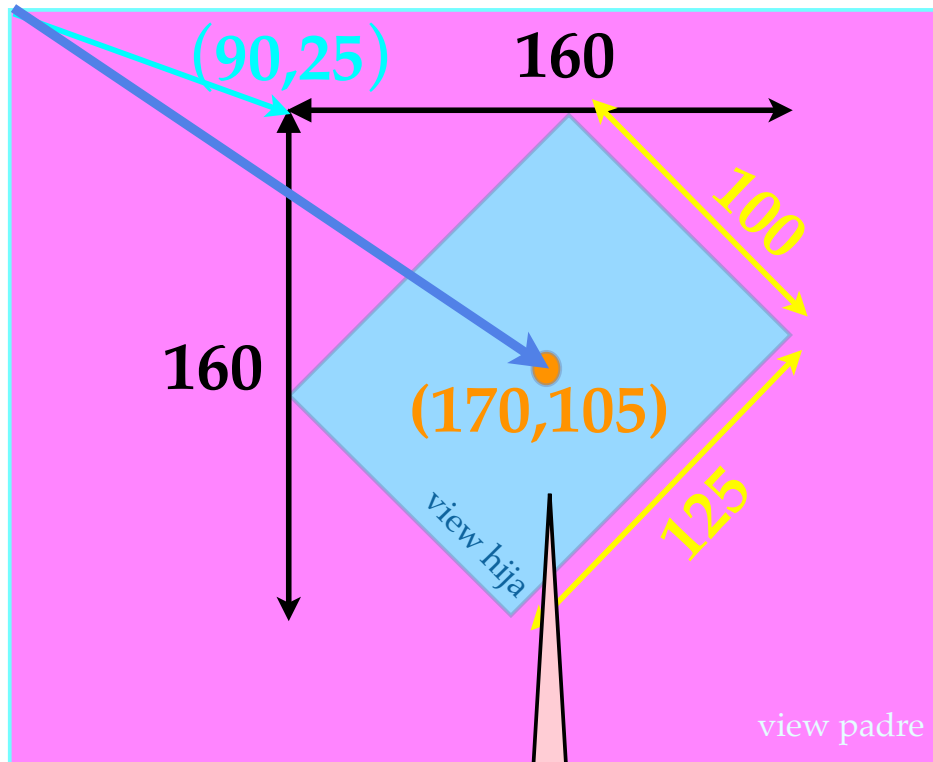
- Es una struct con las propiedades: **origin:CGPoint**, **size:CGSize**.
 - Representa un rectángulo con el origen y tamaño indicados.

- 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)
```

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).



View hija:

bounds = $((0, 0), (100, 125))$

frame = $((90, 25), (160, 160))$

center = $(170, 105)$

Dibujar el Contenido de una UIView

- Podemos crear UIViews personalizadas cuyo contenido lo dibujamos nosotros.
- Para dibujar el contenido 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

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.

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.

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)
```

- 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)
```

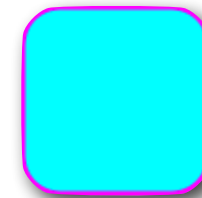
```
path.lineWidth = 3;
```

```
UIColor.magenta.setStroke()
```

```
UIColor.cyan.setFill()
```

```
path.fill()
```

```
path.stroke()
```



- 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()
```



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
o
l
a

- Es mejor usar un objeto UILabel.

Dibujar Imágenes

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

Pintar la imagen en la posición dada.

Crear imagen.

```
img.draw(at: pos)
```

Reescalar imagen y pintarla en el rectángulo dado.

```
img.draw(in: rect)
```

```
img.drawAsPattern(in: rect)
```

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

- Es mejor usar un objeto UIImageView.

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

Inicialización de UIView

- 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 NSCodering.
- **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.

- 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() {  
        // Configuración adicional.  
    }  
  
}
```

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

IB: Previsualizar UIView en Vivo

- 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
```

```
@IBDesignable
class TrajectoryView: UIView {

    @IBInspectable
    var lineWidth : Double = 3.0

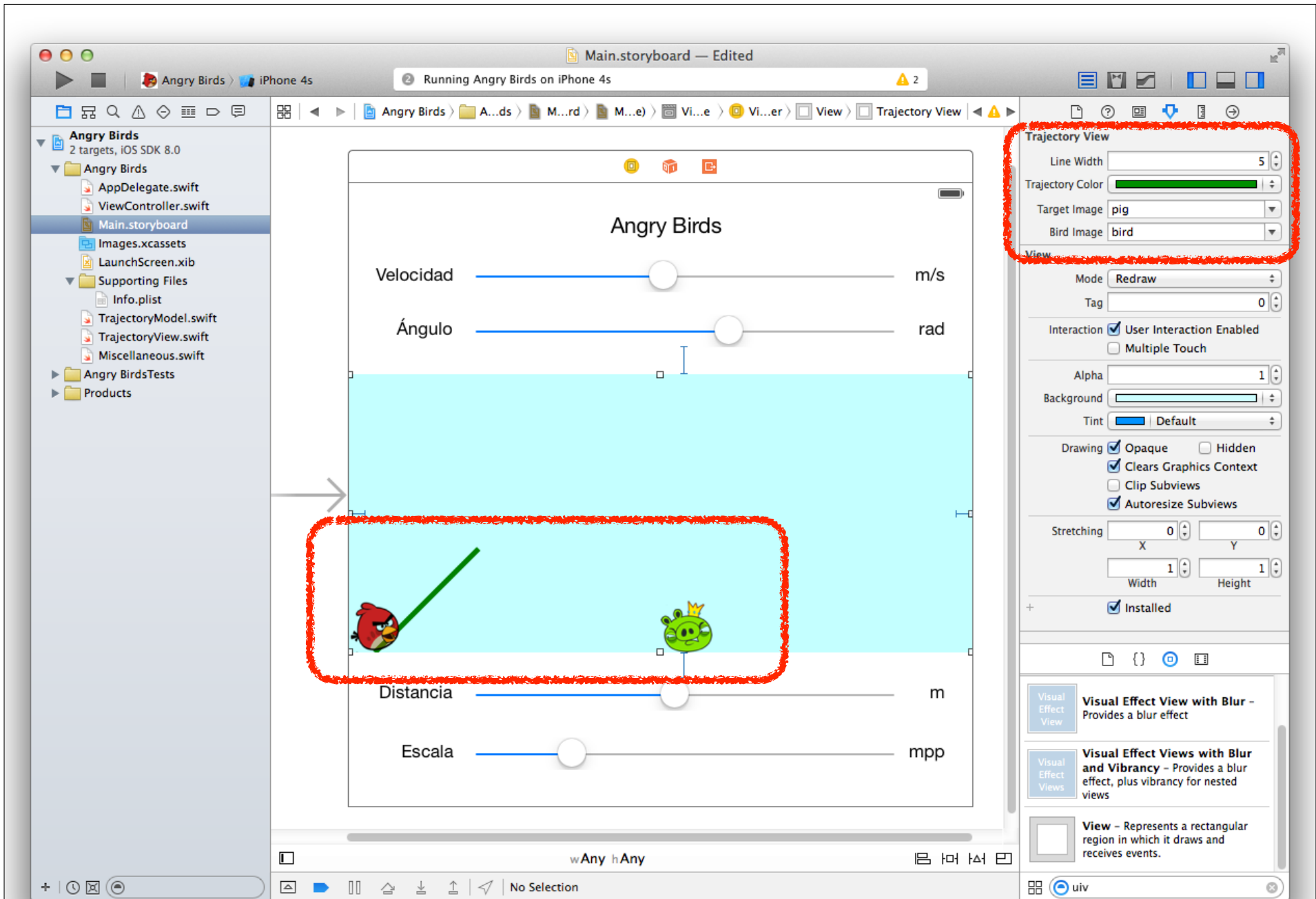
    @IBInspectable
    var trajectoryColor : UIColor = UIColor.redColor()

    @IBInspectable
    var targetImage : UIImage!

    @IBInspectable
    var birdImage : UIImage!

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

    // . . .
```



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.