



EBook Gratis

APRENDIZAJE sprite-kit

Free unaffiliated eBook created from
Stack Overflow contributors.

#sprite-kit

Tabla de contenido

Acerca de.....	1
Capítulo 1: Empezando con el kit de sprites.....	2
Observaciones.....	2
Versiones.....	2
Examples.....	3
Tu primer juego de SpriteKit (Hello World).....	3
Capítulo 2: Detección de entrada táctil en dispositivos iOS.....	6
Examples.....	6
Detección de toque.....	6
Capítulo 3: Elementos UIKit con SpriteKit.....	7
Examples.....	7
UITableView en SKScene.....	7
Protocolo / Delegado para llamar al método ViewController de un juego desde la escena del.....	8
StackView en SKScene.....	9
Múltiples UIViewController en un juego: cómo saltar de la escena a un viewController.....	11
Guión gráfico :.....	11
GameViewController :.....	12
GameScene :.....	13
Capítulo 4: Física.....	14
Examples.....	14
Cómo eliminar correctamente el nodo en el método didBeginContact (múltiples contactos).....	14
Capítulo 5: Funciones temporizadas en SpriteKit: SKActions vs NSTimers.....	15
Observaciones.....	15
Examples.....	15
Implementando un método que dispara después de un segundo.....	15
Capítulo 6: SKAcción.....	16
Examples.....	16
Crea y ejecuta una simple acción.....	16
Creando una secuencia repetitiva de acciones.....	16
Ejecutar un bloque de código en un SKAction.....	16

Acciones nombradas que pueden ser iniciadas o eliminadas desde otro lugar.....	16
Capítulo 7: SKNode Collision.....	18
Observaciones.....	18
Examples.....	18
Habilitar el mundo de la física.....	18
Habilitar nodo para colisionar.....	18
Contactos de la manija.....	19
Alternativa hizo comenzarContacto.....	20
Simple proyecto de Sprite Kit que muestra colisiones, contactos y eventos táctiles.....	20
Alternativa al manejo de contacto cuando se trata de sprites de múltiples categorías.....	24
Diferencia entre contactos y colisiones.....	24
Manipular las máscaras de bits contactTest y collison para habilitar / deshabilitar contac.....	25
Capítulo 8: SKScene.....	28
Observaciones.....	28
Examples.....	28
Subclasificando SKScene para implementar la funcionalidad principal de SpriteKit.....	28
Crear un SKScene que llena el SKView.....	28
Cree un SKScene que se adapte al SKView.....	29
Cree un SKScene con un SKCameraNode (iOS 9 y posterior).....	30
Capítulo 9: SKSpriteNode (Sprites).....	31
Sintaxis.....	31
Examples.....	31
Añadiendo un Sprite a la escena.....	31
Creando un Sprite.....	31
Subclase SKSpriteNode.....	32
Capítulo 10: SKView.....	34
Parámetros.....	34
Observaciones.....	34
Examples.....	34
Crear un SKView de pantalla completa utilizando Interface Builder.....	34
Visualización de información de depuración.....	35
Crea un pequeño SKView con otros controles usando Interface Builder.....	36

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [sprite-kit](#)

It is an unofficial and free sprite-kit ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official sprite-kit.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con el kit de sprites

Observaciones

SpriteKit es un motor de juego 2D desarrollado por Apple. Proporciona API de alto nivel y una amplia gama de funcionalidades para los desarrolladores. También contiene un motor interno de física.

Está disponible en todas las plataformas de Apple.

- iOS
- Mac OS
- tvOS
- watchOS (> = 3.0)

Nota: Si desea desarrollar utilizando gráficos 3D, necesita usar SceneKit en su lugar.

Los bloques de construcción centrales de SpriteKit son:

- [SKView](#) : una vista en la que se presentan SKScenes.
- [SKScene](#) : una escena 2D que se presenta en un SKView y contiene uno o más SKSpriteNodes.
- [SKSpriteNode](#) : una imagen 2D individual que puede ser animada alrededor de la escena.

Otros bloques de construcción relacionados son:

- SKNode: un nodo más general que se puede usar en una escena para agrupar otros nodos para un comportamiento más complejo.
- SKAction: acciones individuales o grupos de acciones que se aplican a SKNodes para implementar animaciones y otros efectos.
- SKPhysicsBody: permite que la física se aplique a SKNodes para permitir que se comporten de una manera realista, incluida la caída por gravedad, rebotar entre sí y seguir trayectorias balísticas.

[Documentación oficial](#) .

Versiones

iOS 7.0 y posteriores

OS X 10.9 Mavericks y más tarde

watchOS 3.0 y posteriores

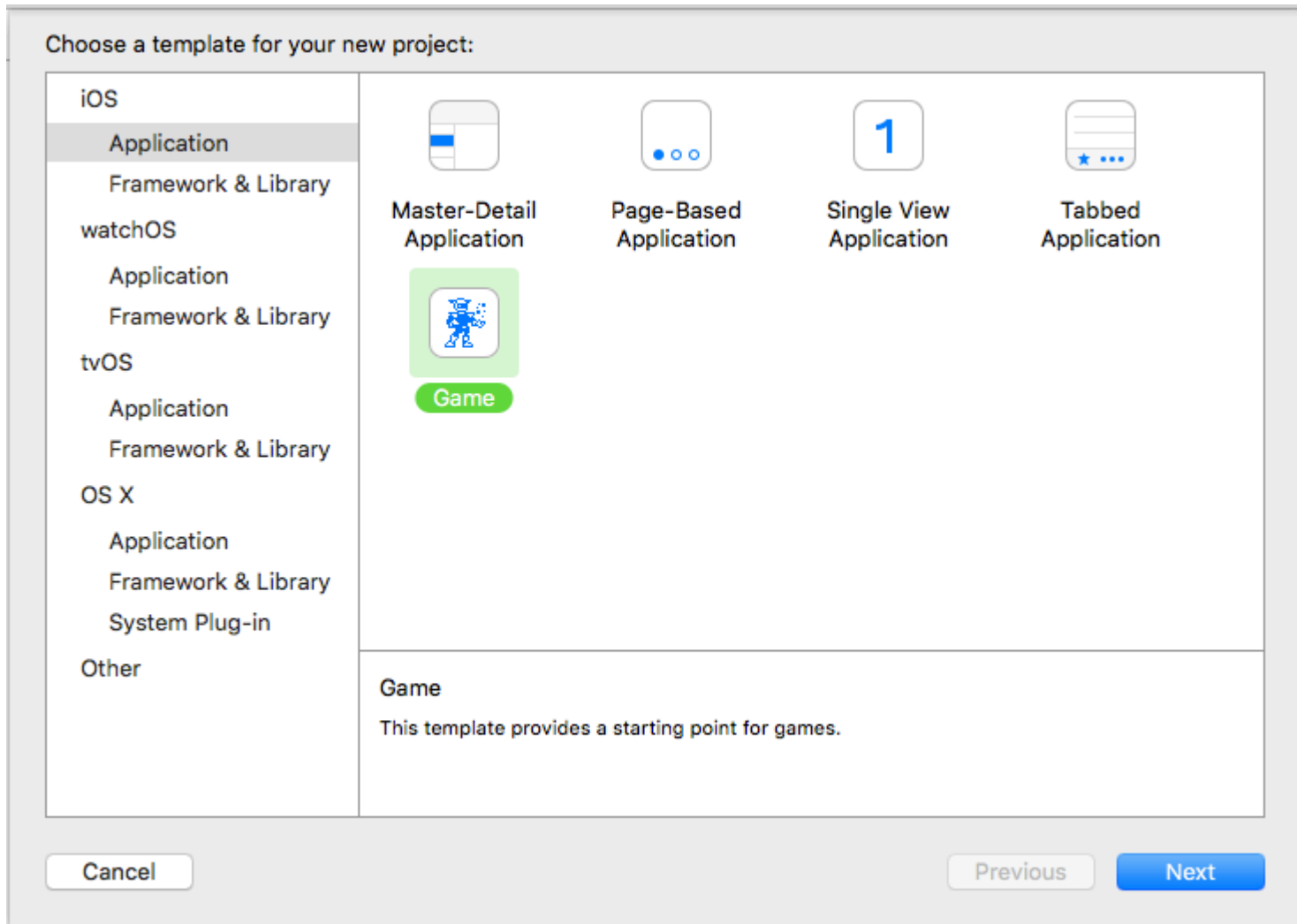
tvOS 9.0 y posteriores

Examples

Tu primer juego de SpriteKit (Hello World)

Abra Xcode y seleccione `Create a new Xcode Project` .

Ahora seleccione `iOS > Application` a la izquierda y `Game` en el área de selección principal.



Presione **Siguiente** .

- Escribe en `Product Name` del `Product Name` el nombre de tu primer gran juego.
- En `Organization Name` el nombre de su empresa (o simplemente su propio nombre).
- `Organisation Identifier` debe contener su nombre de dominio *invertido* (`www.yourdomain.com` convierte en `com.yourdomain`). Si no tienes un dominio, escribe lo que quieras (esto es solo una prueba).
- Luego selecciona `Swift` , `SpriteKit` y `iPhone` .

Choose options for your new project:

Product Name:

Organization Name:

Organization Identifier:

Bundle Identifier:

Language:

Game Technology:

Devices:

Include Unit Tests

Include UI Tests

Presione **Siguiente** .

Seleccione una carpeta de su Mac donde desea guardar el proyecto y haga clic en **Crear** .

Felicidades, creas tu primer juego con SpriteKit! ¡Solo presiona `CMD + R` para ejecutarlo en el simulador!



Lea Empezando con el kit de sprites en línea: <https://riptutorial.com/es/sprite-kit/topic/2956/empezando-con-el-kit-de-sprites>

Capítulo 2: Detección de entrada táctil en dispositivos iOS

Examples

Detección de toque

Puede anular 4 métodos de `SKScene` para detectar el toque del usuario

```
class GameScene: SKScene {  
  
    override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?) {  
    }  
  
    override func touchesMoved(touches: Set<UITouch>, withEvent event: UIEvent?) {  
    }  
  
    override func touchesEnded(touches: Set<UITouch>, withEvent event: UIEvent?) {  
    }  
  
    override func touchesCancelled(touches: Set<UITouch>?, withEvent event: UIEvent?) {  
    }  
  
}
```

Tenga en cuenta que cada método recibe un parámetro de `touches` que (en circunstancias particulares) puede contener más de un solo evento táctil.

Lea [Detección de entrada táctil en dispositivos iOS en línea](https://riptutorial.com/es/sprite-kit/topic/3660/deteccion-de-entrada-tactil-en-dispositivos-ios): <https://riptutorial.com/es/sprite-kit/topic/3660/deteccion-de-entrada-tactil-en-dispositivos-ios>

Capítulo 3: Elementos UIKit con SpriteKit

Examples

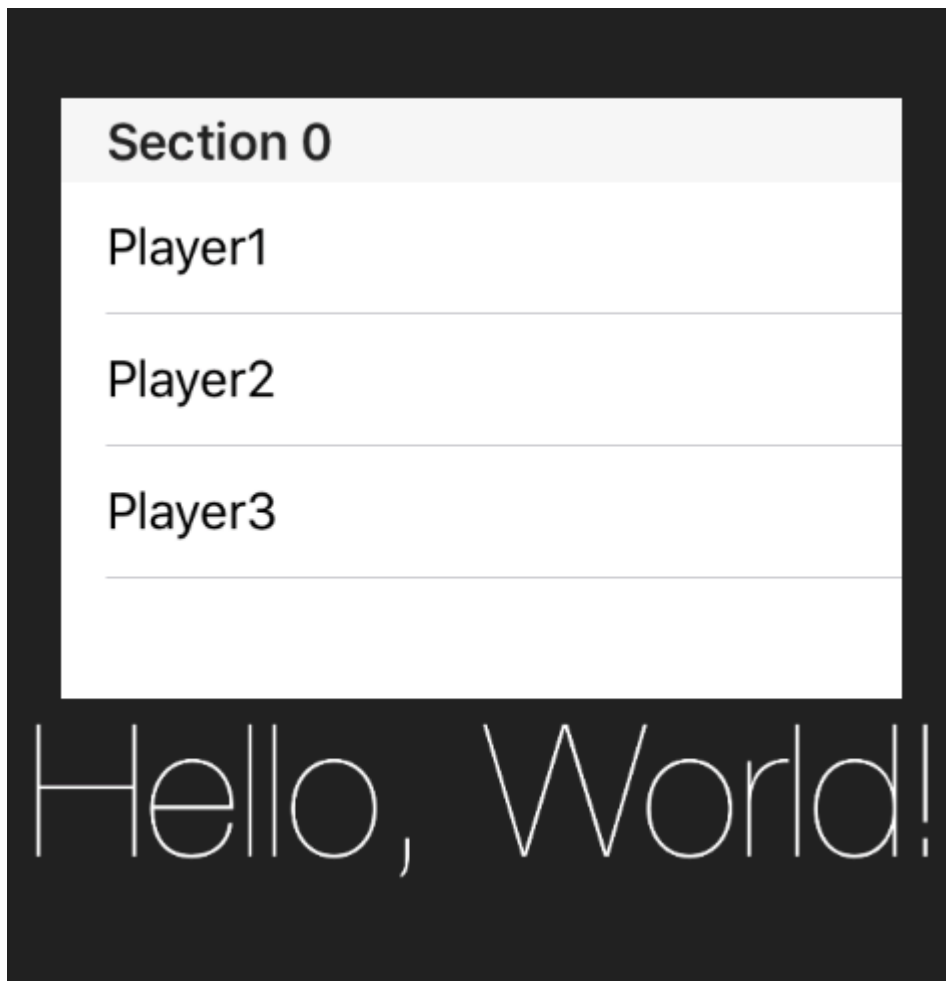
UITableView en SKScene

```
import SpriteKit
import UIKit

class GameRoomTableView: UITableView, UITableViewDelegate, UITableViewDataSource {
    var items: [String] = ["Player1", "Player2", "Player3"]
    override init(frame: CGRect, style: UITableViewStyle) {
        super.init(frame: frame, style: style)
        self.delegate = self
        self.dataSource = self
    }
    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
    // MARK: - Table view data source
    func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return items.count
    }
    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
    UITableViewCell {
        let cell:UITableViewCell = tableView.dequeueReusableCell(withIdentifier: "cell")! as
    UITableViewCell
        cell.textLabel?.text = self.items[indexPath.row]
        return cell
    }
    func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String?
    {
        return "Section \(section)"
    }
    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        print("You selected cell #\(indexPath.row)!")
    }
}

class GameScene: SKScene {
    var gameTableView = GameRoomTableView()
    private var label : SKLabelNode?
    override func didMove(to view: SKView) {
        self.label = self.childNode(withName: "//helloLabel") as? SKLabelNode
        if let label = self.label {
            label.alpha = 0.0
            label.run(SKAction.fadeIn(withDuration: 2.0))
        }
        // Table setup
        gameTableView.register(UITableViewCell.self, forCellReuseIdentifier: "cell")
        gameTableView.frame=CGRect(x:20,y:50,width:280,height:200)
        view.addSubview(gameTableView)
        gameTableView.reloadData()
    }
}
```

Salida :



Protocolo / Delegado para llamar al método ViewController de un juego desde la escena del juego.

Ejemplo de código de GameScene :

```
import SpriteKit
protocol GameControllerDelegate: class {
    func callMethod(inputProperty:String)
}
class GameScene: SKScene {
    weak var viewControllerDelegate:GameViewControllerDelegate?
    override func didMove(to view: SKView) {
        viewControllerDelegate?.callMethod(inputProperty: "call game view controller
method")
    }
}
```

Ejemplo de código de GameController :

```
class GameController: UIViewController, GameControllerDelegate {
    override func viewDidLoad() {
        super.viewDidLoad()
        if let view = self.view as! SKView? {
            // Load the SKScene from 'GameScene.sks'
        }
    }
}
```

```

        if let scene = SKScene(fileName: "GameScene") {
            let gameScene = scene as! GameScene
            gameScene.gameViewControllerDelegate = self
            gameScene.scaleMode = .aspectFill
            view.presentScene(gameScene)
        }
        view.ignoresSiblingOrder = true
        view.showsFPS = true
        view.showsNodeCount = true
    }
}
func callMethod(inputProperty:String) {
    print("inputProperty is: ",inputProperty)
}
}

```

Salida :

```
inputProperty is: call game view controller method
```

StackView en SKScene

```

import SpriteKit
import UIKit
protocol StackViewDelegate: class {
    func didTapOnView(at index: Int)
}
class GameMenuView: UIStackView {
    weak var delegate: StackViewDelegate?
    override init(frame: CGRect) {
        super.init(frame: frame)
        self.axis = .vertical
        self.distribution = .fillEqually
        self.alignment = .fill
        self.spacing = 5
        self.isUserInteractionEnabled = true
        //set up a label
        for i in 1...5 {
            let label = UILabel()
            label.text = "Menu voice \(i)"
            label.textColor = UIColor.white
            label.backgroundColor = UIColor.blue
            label.textAlignment = .center
            label.tag = i
            self.addArrangedSubview(label)
        }
        configureTapGestures()
    }
    required init(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
    private func configureTapGestures() {
        arrangedSubviews.forEach { view in
            view.isUserInteractionEnabled = true
            let tapGesture = UITapGestureRecognizer(target: self, action:
#selector(didTapOnView))
            view.addGestureRecognizer(tapGesture)
        }
    }
}

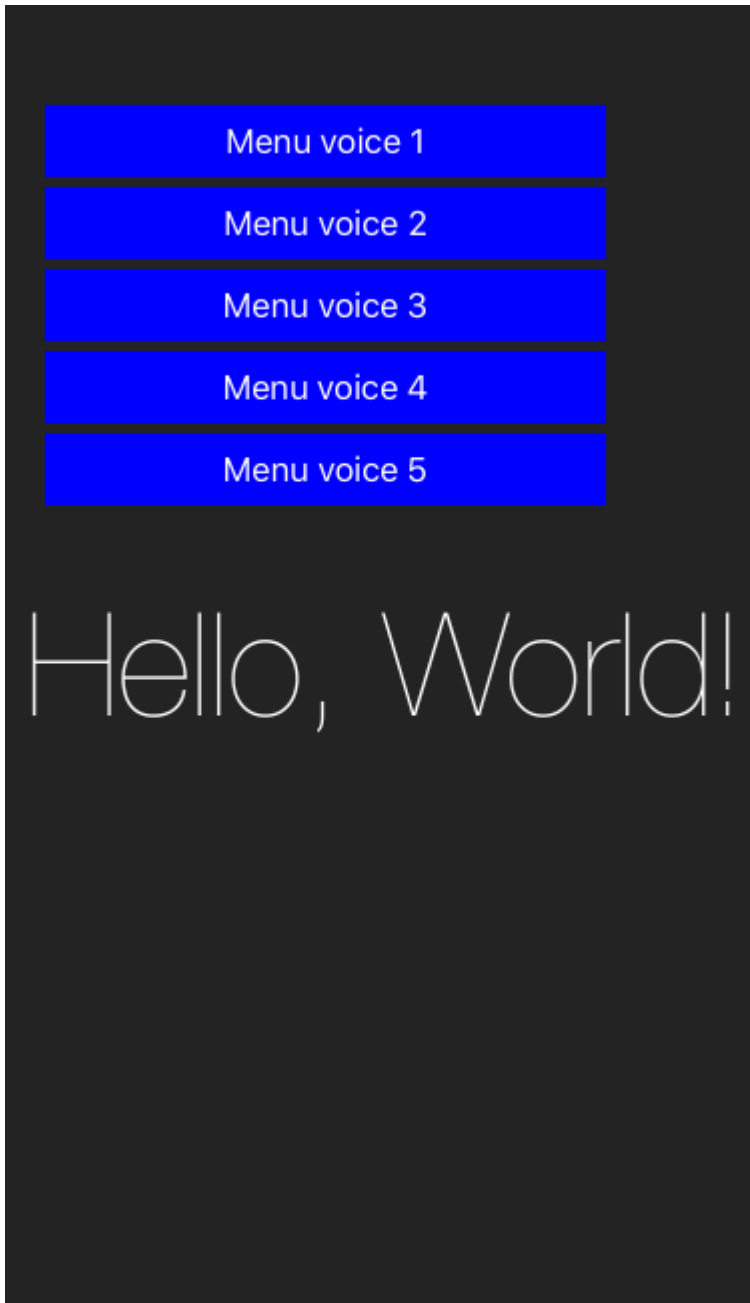
```

```

func didTapOnView(_ gestureRecognizer: UIGestureRecognizer) {
    if let index = arrangedSubviews.index(of: gestureRecognizer.view!) {
        delegate?.didTapOnView(at: index)
    }
}
}
class GameScene: SKScene, StackViewDelegate {
    var gameMenuView = GameMenuView()
    private var label : SKLabelNode?
    override func didMove(to view: SKView) {
        self.label = self.childNode(withName: "//helloLabel") as? SKLabelNode
        if let label = self.label {
            label.alpha = 0.0
            label.run(SKAction.fadeIn(withDuration: 2.0))
        }
        // Menu setup with stackView
        gameMenuView.frame=CGRect(x:20,y:50,width:280,height:200)
        view.addSubview(gameMenuView)
        gameMenuView.delegate = self
    }
    func didTapOnView(at index: Int) {
        switch index {
            case 0: print("tapped voice 1")
            case 1: print("tapped voice 2")
            case 2: print("tapped voice 3")
            case 3: print("tapped voice 4")
            case 4: print("tapped voice 5")
            default:break
        }
    }
}
}

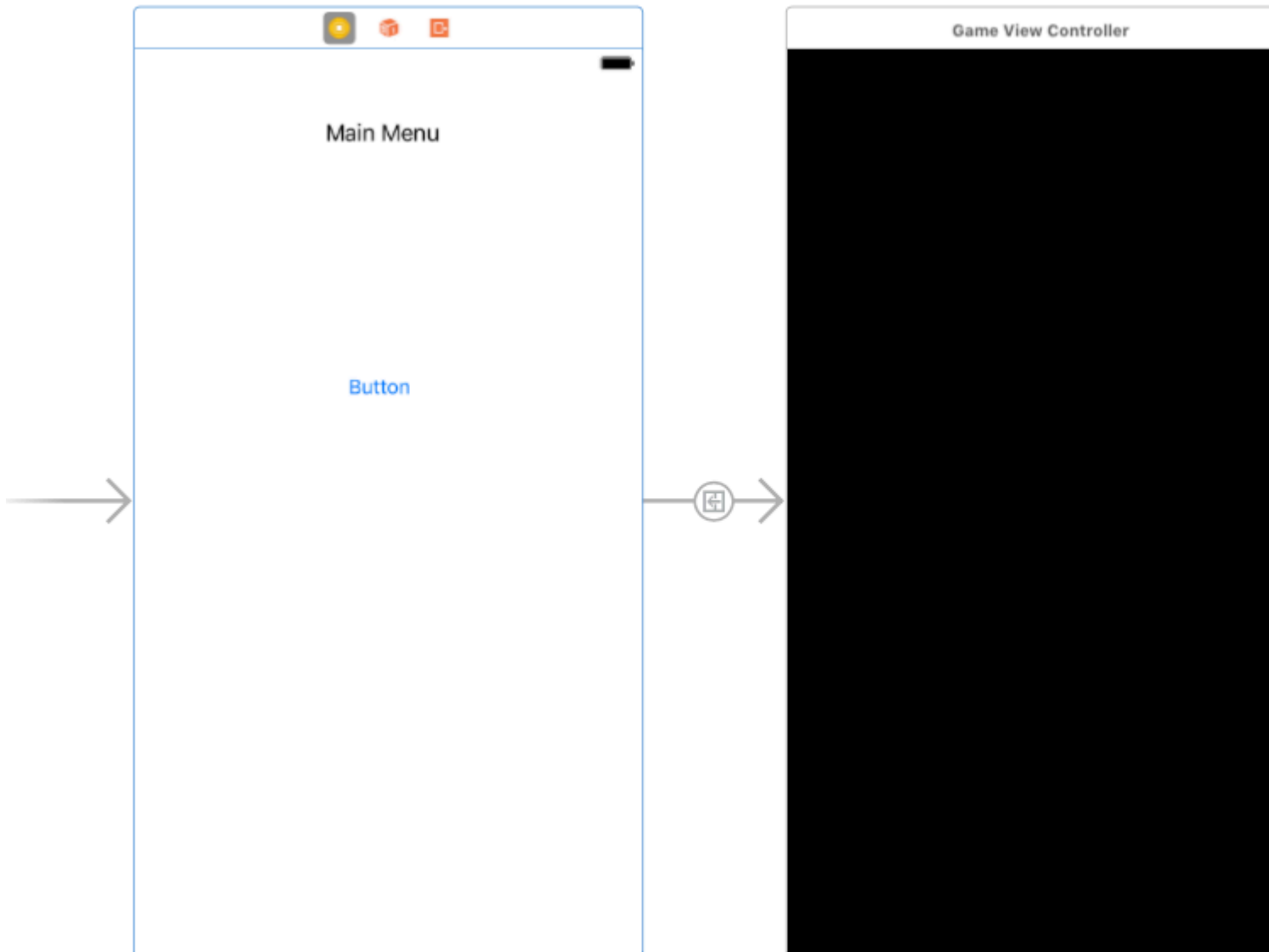
```

Salida :



Múltiples UIViewController en un juego: cómo saltar de la escena a un viewController

Guión gráfico :



Control de vista inicial : un controlador de vista vacío con un botón para presentar el controlador de juego GameView.

GameViewController : el típico GameViewController de la plantilla Sprite-kit *"Hello World"* .

Objetivo : Quiero presentar el primer viewController de mi juego `SKScene` con la correcta desasignación de mi escena.

Descripción : para obtener el resultado, extendí la clase `SKSceneDelegate` para crear un protocolo/delegate que haga la transición del `GameViewController` al primer controlador inicial (menú principal). Este método puede extenderse a otros controladores de vista de tu juego.

GameViewController :

```
import UIKit
import SpriteKit
class GameViewController: UIViewController, TransitionDelegate {
    override func viewDidLoad() {
        super.viewDidLoad()
        if let view = self.view as! SKView? {
```



```

        if let scene = SKScene(fileName: "GameScene") {
            scene.scaleMode = .aspectFill
            scene.delegate = self as TransitionDelegate
            view.presentScene(scene)
        }
        view.ignoresSiblingOrder = true
        view.showsFPS = true
        view.showsNodeCount = true
    }
}
func returnToMainMenu(){
    let appDelegate = UIApplication.shared.delegate as! AppDelegate
    guard let storyboard = appDelegate.window?.rootViewController?.storyboard else {
return }
    if let vc = storyboard.instantiateInitialViewController() {
        print("go to main menu")
        self.present(vc, animated: true, completion: nil)
    }
}
}
}

```

GameScene :

```

import SpriteKit
protocol TransitionDelegate: SKSceneDelegate {
    func returnToMainMenu()
}
class GameScene: SKScene {
    override func didMove(to view: SKView) {
        self.run(SKAction.wait(forDuration: 2), completion: {[unowned self] in
            guard let delegate = self.delegate else { return }
            self.view?.presentScene(nil)
            (delegate as! TransitionDelegate).returnToMainMenu()
        })
    }
    deinit {
        print("\n THE SCENE \((type(of: self))) WAS REMOVED FROM MEMORY (DEINIT) \n")
    }
}
}

```

Lea Elementos UIKit con SpriteKit en línea: <https://riptutorial.com/es/sprite-kit/topic/8807/elementos-uikit-con-spritekit>

Capítulo 4: Física

Examples

Cómo eliminar correctamente el nodo en el método didBeginContact (múltiples contactos)

```
// PHYSICS CONSTANTS
struct PhysicsCategory {
    static let None          : UInt32 = 0
    static let All           : UInt32 = UInt32.max
    static let player        : UInt32 = 0b1           // 1
    static let bullet        : UInt32 = 0b10          // 2
}

var nodesToRemove = [SKNode]()

// #-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
//MARK: - Physic Contact Delegate methods
// #-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#

func didBegin(_ contact: SKPhysicsContact) {
    var one: SKPhysicsBody
    var two: SKPhysicsBody

    if contact.bodyA.categoryBitMask < contact.bodyB.categoryBitMask {
        one = contact.bodyA
        two = contact.bodyB
    } else {
        one = contact.bodyB
        two = contact.bodyA
    }

    // PLAYER AND BULLET
    if one.categoryBitMask == PhysicsCategory.player && two.categoryBitMask ==
PhysicsCategory.bullet {
        nodesToRemove.append(one.node!) // remove player
        nodesToRemove.append(two.node!) // remove bullet
    }
}

override func didFinishUpdate()
{
    nodesToRemove.forEach(){$0.removeFromParent()}
    nodesToRemove = [SKNode]()
}
}
```

Lea Física en línea: <https://riptutorial.com/es/sprite-kit/topic/8991/fisica>

Capítulo 5: Funciones temporizadas en SpriteKit: SKActions vs NSTimers

Observaciones

¿Cuándo debería usar `SKAction` s para realizar funciones de temporizador? Casi siempre. La razón de esto es porque `Sprite Kit` opera en un intervalo de actualización, y la velocidad de este intervalo se puede cambiar durante la vida útil del proceso utilizando la propiedad de `speed` . Las escenas también se pueden pausar también. Dado que el trabajo de `SKAction` dentro de la escena, cuando modifica estas propiedades, no es necesario modificar las funciones de tiempo. Si su escena está a 0.5 segundos del proceso y la detiene, no necesita detener ningún temporizador y mantener esa diferencia de 0.5 segundos. Se le entrega automáticamente, de modo que, cuando recupere la pausa, continúe el tiempo restante.

¿Cuándo debería usar `NSTimer` s para realizar funciones de temporizador? Siempre que tenga algo que deba cronometrarse fuera del entorno de `SKScene` , y también debe dispararse incluso cuando la escena está en pausa, o debe dispararse a una velocidad constante incluso cuando la velocidad de la escena cambia.

Esta es la mejor opción cuando se trabaja con las dos `UIKit` controles y `SpriteKit` controles. Dado que `UIKit` no tiene idea de lo que sucede con `SpriteKit` , `NSTimer` s se activará independientemente del estado del `SKScene` . Un ejemplo sería que tenemos una `UILabel` que recibe una actualización cada segundo, y necesita datos dentro de su `SKScene` .

Examples

Implementando un método que dispara después de un segundo.

SKACCIÓN:

```
let waitForOneSecond = SKAction.waitForDuration(1) let action = SKAction.runBlock(){action()}\nlet sequence = SKAction.sequence([waitForOneSecond, action]) self.runAction(sequence)
```

NSTimer:

```
NSTimer.scheduledTimerWithTimeInterval(1, target: self, selector: #selector(action), userInfo:\nnil, repeats: false)
```

Lea Funciones temporizadas en SpriteKit: SKActions vs NSTimers en línea:

<https://riptutorial.com/es/sprite-kit/topic/5962/funciones-temporizadas-en-spritekit--skactions-vs-nstimers>

Capítulo 6: SKAcción

Examples

Crea y ejecuta una simple acción

Un ejemplo muy simple sería desvanecer un SKSpriteNode.

En Swift:

```
let node = SKSpriteNode(imageNamed: "image")
let action = SKAction.fadeOutWithDuration(1.0)
node.runAction(action)
```

Creando una secuencia repetitiva de acciones

A veces es necesario realizar una acción en repetición o en una secuencia. Este ejemplo hará que el nodo aparezca y desaparezca un total de 3 veces.

En Swift:

```
let node = SKSpriteNode(imageNamed: "image")
let actionFadeOut = SKAction.fadeOutWithDuration(1.0)
let actionFadeIn = SKAction.fadeInWithDuration(1.0)
let actionSequence = SKAction.sequence([actionFadeOut, actionFadeIn])
let actionRepeat = SKAction.repeatAction(actionSequence, count: 3)
node.runAction(actionRepeat)
```

Ejecutar un bloque de código en un SKAction

Un caso útil es hacer que la acción ejecute un bloque de código.

En Swift:

```
let node = SKSpriteNode(imageNamed: "image")
let actionBlock = SKAction.runBlock({
    //Do what you want here
    if let gameScene = node.scene as? GameScene {
        gameScene.score += 5
    }
})
node.runAction(actionBlock)
```

Acciones nombradas que pueden ser iniciadas o eliminadas desde otro lugar.

A veces, desearía iniciar o eliminar una acción en un nodo específico en un momento determinado. Por ejemplo, es posible que desee detener un objeto en movimiento cuando el usuario toca la pantalla. Esto resulta muy útil cuando un nodo tiene múltiples acciones y solo

desea acceder a una de ellas.

```
let move = SKAction.moveTo(x: 200, duration: 2)
object.run(move, withKey: "moveX")
```

Aquí configuramos la tecla "moveX" para el `move` acción para poder acceder más tarde en otra parte de la clase.

```
override fun touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    object.removeAction(forKey: "moveX")
}
```

Cuando el usuario toque la pantalla, la acción se eliminará y el objeto dejará de moverse.

Lea SKAcción en línea: <https://riptutorial.com/es/sprite-kit/topic/6229/skaccion>

Capítulo 7: SKNode Collision

Observaciones

Los determinantes de la colisión del Sprite Kit y el procesamiento de eventos de contacto son los ajustes de relación, creados por usted, de `categoryBitMask`, `collisionBitMask` y `contactTestBitMask` para cada uno de sus tipos de objetos interactivos. Al establecer racionalmente estos servicios al servicio de los resultados deseados de los contactos y las colisiones, usted determina qué tipos pueden colisionar e informar de los contactos con otros, y evita la colisión no deseada, el contacto y la sobrecarga de procesamiento de la física.

Para cada tipo de 'entidad' puede configurar los tres:

1. `categoryBitMask` : una categoría específica para este tipo de nodo
2. `collisionBitMask` : un diferenciador de colisión, puede ser diferente del anterior
3. `contactTestBitMask` : un diferenciador de contactos, puede ser diferente de los dos anteriores

Los pasos generales para implementar colisiones y contactos son:

1. establecer tamaño corporal físico, forma y (a veces) masa
2. agregue las BitMasks necesarias para su tipo de nodo de la categoría, colisión y contacto arriba
3. configura la escena como un delegado de contacto que le permite verificar e informar sobre colisiones y contactos
4. Implementar controladores de contactos y cualquier otra lógica pertinente para eventos de física.

Examples

Habilitar el mundo de la física

```
// World physics
self.physicsWorld.gravity = CGVectorMake(0, -9.8);
```

Habilitar nodo para colisionar

En primer lugar, establecemos la categoría de nodo.

```
let groundBody: UInt32 = 0x1 << 0
let boxBody: UInt32 = 0x1 << 1
```

Luego agregue el nodo Tipo de tierra y el nodo Tipo de caja

```
let ground = SKSpriteNode(color: UIColor.cyanColor(), size: CGSizeMake(self.frame.width, 50))
ground.position = CGPointMake(CGRectGetMidX(self.frame), 100)
ground.physicsBody = SKPhysicsBody(rectangleOfSize: ground.size)
```

```

ground.physicsBody?.dynamic = false
ground.physicsBody?.categoryBitMask = groundBody
ground.physicsBody?.collisionBitMask = boxBody
ground.physicsBody?.contactTestBitMask = boxBody

addChild(ground)

// Add box type node

let box = SKSpriteNode(color: UIColor.yellowColor(), size: CGSizeMake(20, 20))
box.position = location
box.physicsBody = SKPhysicsBody(rectangleOfSize: box.size)
box.physicsBody?.dynamic = true
box.physicsBody?.categoryBitMask = boxBody
box.physicsBody?.collisionBitMask = groundBody | boxBody
box.physicsBody?.contactTestBitMask = boxBody
box.name = boxId

let action = SKAction.rotateByAngle(CGFloat(M_PI), duration:1)

box.runAction(SKAction.repeatActionForever(action))

self.addChild(box)

```

Contactos de la manija

Establecer escena como delegado

```

//set your scene as SKPhysicsContactDelegate

class yourScene: SKScene, SKPhysicsContactDelegate

self.physicsWorld.contactDelegate = self;

```

Luego, tiene que implementar una u otra de las funciones de contacto: la función opcional `comido` `Comenzar (contacto :)` y / o el método opcional de fondos de fondos (`contacto :`) para completar su lógica de contacto, por ejemplo, como

```

//order

let bodies = (contact.bodyA.categoryBitMask <= contact.bodyB.categoryBitMask) ?
(A:contact.bodyA,B:contact.bodyB) : (A:contact.bodyB,B:contact.bodyA)

//real handler
if ((bodies.B.categoryBitMask & boxBody) == boxBody){
    if ((bodies.A.categoryBitMask & groundBody) == groundBody) {
        let vector = bodies.B.velocity
        bodies.B.velocity = CGVectorMake(vector.dx, vector.dy * 4)

    }else{
        let vector = bodies.A.velocity
        bodies.A.velocity = CGVectorMake(vector.dx, vector.dy * 10)

    }
}

```

Alternativa hizo comenzarContacto

Si está utilizando categorías simples, y cada cuerpo de física pertenece a una sola categoría, entonces esta forma alternativa de `didBeginContact` puede ser más legible:

```
func didBeginContact(contact: SKPhysicsContact) {
    let contactMask = contact.bodyA.categoryBitMask | contact.bodyB.categoryBitMask

    switch contactMask {

    case categoryBitMask.player | categoryBitMask.enemy:
        print("Collision between player and enemy")
        let enemyNode = contact.bodyA.categoryBitMask == categoryBitMask.enemy ?
        contact.bodyA.node! : contact.bodyB.node!
        enemyNode.explode()
        score += 10

    case categoryBitMask.enemy | categoryBitMask.enemy:
        print("Collision between enemy and enemy")
        contact.bodyA.node.explode()
        contact.bodyB.node.explode()

    default :
        //Some other contact has occurred
        print("Some other contact")
    }
}
```

Simple proyecto de Sprite Kit que muestra colisiones, contactos y eventos táctiles.

Aquí hay un simple `Sprite-Kit GameScene.swift`. Cree un nuevo proyecto `SpriteKit` vacío y reemplace el `GameScene.swift` con esto. Luego construir y ejecutar.

Haga clic en cualquiera de los objetos en la pantalla para hacer que se muevan. Verifique los registros y los comentarios para ver cuáles chocan y cuáles entran en contacto.

```
//
// GameScene.swift
// bounceTest
//
// Created by Stephen Ives on 05/04/2016.
// Copyright (c) 2016 Stephen Ives. All rights reserved.
//

import SpriteKit

class GameScene: SKScene, SKPhysicsContactDelegate {

    let objectSize = 150
    let initialImpulse: UInt32 = 300 // Needs to be proportional to objectSize

    //Physics categories
    let purpleSquareCategory: UInt32 = 1 << 0
```



```

let redCircleCategory:      UInt32 = 1 << 1
let blueSquareCategory:    UInt32 = 1 << 2
let edgeCategory:          UInt32 = 1 << 31

let purpleSquare = SKSpriteNode()
let blueSquare = SKSpriteNode()
let redCircle = SKSpriteNode()

override func didMove(to view: SKView) {

    physicsWorld.gravity = CGVector(dx: 0, dy: 0)

    //Create an boundary else everything will fly off-screen
    let edge = frame.insetBy(dx: 0, dy: 0)
    physicsBody = SKPhysicsBody(edgeLoopFrom: edge)
    physicsBody?.isDynamic = false //This won't move
    name = "Screen_edge"

    scene?.backgroundColor = SKColor.black

    //          Give our 3 objects their attributes

    blueSquare.color = SKColor.blue
    blueSquare.size = CGSize(width: objectSize, height: objectSize)
    blueSquare.name = "shape_blueSquare"
    blueSquare.position = CGPoint(x: size.width * -0.25, y: size.height * 0.2)

    let circleShape = SKShapeNode(circleOfRadius: CGFloat(objectSize))
    circleShape.fillColor = SKColor.red
    redCircle.texture = view.texture(from: circleShape)
    redCircle.size = CGSize(width: objectSize, height: objectSize)
    redCircle.name = "shape_redCircle"
    redCircle.position = CGPoint(x: size.width * 0.4, y: size.height * -0.4)

    purpleSquare.color = SKColor.purple
    purpleSquare.size = CGSize(width: objectSize, height: objectSize)
    purpleSquare.name = "shape_purpleSquare"
    purpleSquare.position = CGPoint(x: size.width * -0.35, y: size.height * 0.4)

    addChild(blueSquare)
    addChild(redCircle)
    addChild(purpleSquare)

    redCircle.physicsBody = SKPhysicsBody(circleOfRadius: redCircle.size.width/2)
    blueSquare.physicsBody = SKPhysicsBody(rectangleOf: blueSquare.frame.size)
    purpleSquare.physicsBody = SKPhysicsBody(rectangleOf: purpleSquare.frame.size)

    setUpCollisions()

    checkPhysics()
}

func setUpCollisions() {

    //Assign our category bit masks to our physics bodies
    purpleSquare.physicsBody?.categoryBitMask = purpleSquareCategory
    redCircle.physicsBody?.categoryBitMask = redCircleCategory
    blueSquare.physicsBody?.categoryBitMask = blueSquareCategory
    physicsBody?.categoryBitMask = edgeCategory // This is the edge for the scene itself
}

```

```

// Set up the collisions. By default, everything collides with everything.

redCircle.physicsBody?.collisionBitMask &= ~purpleSquareCategory // Circle doesn't
collide with purple square
purpleSquare.physicsBody?.collisionBitMask = 0 // purpleSquare collides with nothing
// purpleSquare.physicsBody?.collisionBitMask |= (redCircleCategory |
blueSquareCategory) // Add collisions with red circle and blue square
purpleSquare.physicsBody?.collisionBitMask = (redCircleCategory) // Add collisions
with red circle
blueSquare.physicsBody?.collisionBitMask = (redCircleCategory) // Add collisions with
red circle

// Set up the contact notifications. By default, nothing contacts anything.
redCircle.physicsBody?.contactTestBitMask |= purpleSquareCategory // Notify when red
circle and purple square contact
blueSquare.physicsBody?.contactTestBitMask |= redCircleCategory // Notify when
blue square and red circle contact

// Make sure everything collides with the screen edge and make everything really
'bouncy'
enumerateChildNodes(withName: "//shape*") { node, _ in
    node.physicsBody?.collisionBitMask |= self.edgeCategory //Add edgeCategory to the
collision bit mask
    node.physicsBody?.restitution = 0.9 // Nice and bouncy...
    node.physicsBody?.linearDamping = 0.1 // Nice and bouncy...
}

//Lastly, set ourselves as the contact delegate
physicsWorld.contactDelegate = self
}

func didBegin(_ contact: SKPhysicsContact) {
    let contactMask = contact.bodyA.categoryBitMask | contact.bodyB.categoryBitMask

    switch contactMask {
    case purpleSquareCategory | blueSquareCategory:
        print("Purple square and Blue square have touched")
    case redCircleCategory | blueSquareCategory:
        print("Red circle and Blue square have touched")
    case redCircleCategory | purpleSquareCategory:
        print("Red circle and purple Square have touched")
    default: print("Unknown contact detected")
    }
}

override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {

    for touch in touches {
        let touchedNode = selectNodeForTouch(touch.location(in: self))

        if let node = touchedNode {
            node.physicsBody?.applyImpulse(CGVector(dx:
CGFloat(arc4random_uniform(initialImpulse)) - CGFloat(initialImpulse/2), dy:
CGFloat(arc4random_uniform(initialImpulse)) - CGFloat(initialImpulse/2)))
            node.physicsBody?.applyTorque(CGFloat(arc4random_uniform(20)) - CGFloat(10))
        }
    }
}
}

```

```

// Return the sprite where the user touched the screen
func selectNodeForTouch(_ touchLocation: CGPoint) -> SKSpriteNode? {

    let touchedNode = self.atPoint(touchLocation)
    print("Touched node is \(touchedNode.name)")
    //         let touchedColor = getPixelColorAtPoint(touchLocation)
    //         print("Touched colour is \(touchedColor)")

    if touchedNode is SKSpriteNode {
        return (touchedNode as! SKSpriteNode)
    } else {
        return nil
    }
}

//MARK: - Analyse the collision/contact set up.
func checkPhysics() {

    // Create an array of all the nodes with physicsBodies
    var physicsNodes = [SKNode]()

    //Get all physics bodies
    enumerateChildNodes(withName: "//.") { node, _ in
        if let _ = node.physicsBody {
            physicsNodes.append(node)
        } else {
            print("\(node.name) does not have a physics body so cannot collide or be
involved in contacts.")
        }
    }

    //For each node, check it's category against every other node's collision and contactTest
bit mask
    for node in physicsNodes {
        let category = node.physicsBody!.categoryBitMask
        // Identify the node by its category if the name is blank
        let name = node.name != nil ? node.name! : "Category \(category)"

        let collisionMask = node.physicsBody!.collisionBitMask
        let contactMask = node.physicsBody!.contactTestBitMask

        // If all bits of the collisionmask set, just say it collides with everything.
        if collisionMask == UInt32.max {
            print("\(name) collides with everything")
        }

        for otherNode in physicsNodes {
            if (node.physicsBody?.dynamic == false) {
                print("This node \(name) is not dynamic")
            }

            if (node != otherNode) && (node.physicsBody?.isDynamic == true) {
                let otherCategory = otherNode.physicsBody!.categoryBitMask
                // Identify the node by its category if the name is blank
                let otherName = otherNode.name != nil ? otherNode.name! : "Category
\(\otherCategory)"

                // If the collisionmask and category match, they will collide
                if ((collisionMask & otherCategory) != 0) && (collisionMask != UInt32.max)
{
                    print("\(name) collides with \(\otherName)")
                }
            }
        }
    }
}

```

```

    }
    // If the contactMask and category match, they will contact
    if (contactMask & otherCategory) != 0 {print("\(name) notifies when
contacting \(otherName)")}
    }
}
}
}
}
}

```

Alternativa al manejo de contacto cuando se trata de sprites de múltiples categorías

```

let bodies = (contact.bodyA.categoryBitMask <= contact.bodyB.categoryBitMask) ?
(A:contact.bodyA,B:contact.bodyB) : (A:contact.bodyB,B:contact.bodyA)

switch (bodies.A.categoryBitMask,bodies.B.categoryBitMask)
{
    case let (a, _) where (a && superPower): //All we care about is if category a has a super
power
        //do super power effect
        fallthrough //continue onto check if we hit anything else
    case let (_, b) where (b && superPower): //All we care about is if category b has a super
power
        //do super power effect
        fallthrough //continue onto check if we hit anything else
    case let (a, b) where (a && groundBody) && (b && boxBody): //Check if box hit ground
//boxBody hit ground
    case let (b, _) where (b && boxBody): //Check if box hit anything else
//box body hit anything else
    default:()
}
}

```

Diferencia entre contactos y colisiones.

En Sprite-Kit, existe el concepto de **colisiones** que se refiere al motor de física de SK que maneja cómo interactúan los objetos de la física cuando chocan, es decir, cuáles rebotan sobre otros.

También tiene el concepto de **contactos**, que es el mecanismo por el cual su programa se informa cuando se cruzan 2 objetos de física.

Los objetos pueden chocar pero no generar contactos, generar contactos sin chocar, o chocar y generar un contacto (o no hacer nada y no interactuar en absoluto)

Las colisiones también pueden ser de un solo lado, es decir, el objeto A puede chocar (rebotar) el objeto B, mientras que el objeto B continúa como si nada hubiera sucedido. Si quieres que 2 objetos reboten entre sí, debes decirles que colisionen con el otro.

Sin embargo, los contactos no son unilaterales; Si desea saber cuándo el objeto A tocó (contactó) el objeto B, es suficiente configurar la detección de contactos en el objeto A con respecto al objeto B. No tiene que configurar la detección de contactos en el objeto B para el objeto A.

Manipular las máscaras de bits `contactTest` y `collison` para habilitar / deshabilitar contactos y colisiones específicos.

Para este ejemplo, usaremos 4 cuerpos y mostraremos solo los últimos 8 bits de las máscaras de bits para simplificar. Los 4 cuerpos son 3 `SKSpriteNodes`, cada uno con un cuerpo de física y un límite:

```
let edge = frame.insetBy(dx: 0, dy: 0)
physicsBody = SKPhysicsBody(edgeLoopFrom: edge)
```

Tenga en cuenta que el cuerpo físico de 'borde' es el cuerpo físico de la escena, no un nodo.

Definimos 4 categorías únicas.

```
let purpleSquareCategory: UInt32 = 1 << 0 // bitmask is ...00000001
let redCircleCategory: UInt32 = 1 << 1 // bitmask is ...00000010
let blueSquareCategory: UInt32 = 1 << 2 // bitmask is ...00000100
let edgeCategory: UInt32 = 1 << 31 // bitmask is 10000...00000000
```

A cada cuerpo de física se le asignan las categorías a las que pertenece:

```
//Assign our category bit masks to our physics bodies
purpleSquare.physicsBody?.categoryBitMask = purpleSquareCategory
redCircle.physicsBody?.categoryBitMask = redCircleCategory
blueSquare.physicsBody?.categoryBitMask = blueSquareCategory
physicsBody?.categoryBitMask = edgeCategory // This is the edge for the scene itself
```

Si un bit en `collisionBitMask` de un cuerpo se establece en 1, entonces colisiona (rebota) cualquier cuerpo que tenga un '1' en la misma posición en su `categoryBitMask`. Del mismo modo para `contactTestBitMask`.

A menos que especifique lo contrario, todo choca con todo lo demás y no se generan contactos (no se notificará a su código cuando nada entre en contacto con cualquier otra cosa):

```
purpleSquare.physicsBody.collisonBitMask = 11111111111111111111111111111111 // 32 '1's.
```

Cada bit en cada posición es '1', por lo que, en comparación con cualquier otra categoría de `BitMask`, Sprite Kit encontrará un '1' por lo que se producirá una colisión. Si no desea que este cuerpo colisione con una determinada categoría, deberá establecer el bit correcto en `collisonBitMask` en '0'

y su `contactTestbitMask` se establece en todos los 0 s:

```
redCircle.physicsBody.contactTestBitMask = 00000000000000000000000000000000 // 32 '0's
```

Igual que para `collisionBitMask`, excepto revertido.

Los contactos o las colisiones entre cuerpos se pueden **desactivar** (dejando el contacto existente o la colisión sin cambios) usando:

```
nodeA.physicsBody?.collisionBitMask &= ~nodeB.category
```

En forma lógica, la máscara de bits de colisión de AND nodeA con el inverso (NOT lógico, el operador ~) de la máscara de bits de la categoría de nodeB para "desactivar" la bitMask de bitAnd de ese bit. Por ejemplo, para evitar que el círculo rojo colisione con el cuadrado púrpura:

```
redCircle.physicsBody?.collisionBitMask = redCircle.physicsBody?.collisionBitMask & ~purpleSquareCategory
```

que se puede acortar a:

```
redCircle.physicsBody?.collisionBitMask &= ~purpleSquareCategory
```

Explicación:

```
redCircle.physicsBody.collisionBitMask = 11111111111111111111111111111111
purpleSquareCategory = 00000000000000000000000000000001
~purpleSquareCategory = 11111111111111111111111111111110
11111111111111111111111111111111 & 11111111111111111111111111111110 =
11111111111111111111111111111110
redCircle.physicsBody.collisionBitMask now equals 11111111111111111111111111111110
```

redCircle ya no colisiona con cuerpos con una categoría de ... 0001 (purpleSquare)

En lugar de desactivar bits individuales en collisionsbitMask, puede establecerlo directamente:

```
blueSquare.physicsBody?.collisionBitMask = (redCircleCategory | purpleSquareCategory)
```

es decir, blueSquare.physicsBody?.collisionBitMask = (...00000010 OR ...00000001)

que es igual a blueSquare.physicsBody?.collisionBitMask = ...00000011

blueSquare solo chocará con cuerpos de una categoría o ..01 o ..10

Contactos o colisiones entre 2 cuerpos pueden estar activada (**sin** afectar a los contactos existentes o colisiones) en cualquier punto usando:

```
redCircle.physicsBody?.contactTestBitMask |= purpleSquareCategory
```

Lógicamente AND redCircle's bitMask con la máscara de bits de purpleSquare para "activar" ese bit en bitMask de redcircle. Esto deja cualquier otro bit en los bitMas de redCircel no afectado.

Puede asegurarse de que cada forma "rebote" en un borde de la pantalla de la siguiente manera:

```
// Make sure everything collides with the screen edge
enumerateChildNodes(withName: "/*") { node, _ in
    node.physicsBody?.collisionBitMask |= self.edgeCategory //Add edgeCategory to the
collision bit mask
}
```

Nota:

Las colisiones pueden ser de un solo lado, es decir, el objeto A puede chocar (rebotar) el objeto B, mientras que el objeto B continúa como si nada hubiera sucedido. Si desea que 2 objetos reboten entre sí, debe decirles a ambos que colisionen con el otro:

```
blueSquare.physicsBody?.collisionBitMask = redCircleCategory
redCircle.physicsBody?.collisionBitMask = blueSquareCategory
```

Sin embargo, los contactos no son unilaterales; Si desea saber cuándo el objeto A tocó (contactó) el objeto B, es suficiente configurar la detección de contactos en el objeto A con respecto al objeto B. No tiene que configurar la detección de contactos en el objeto B para el objeto A.

```
blueSquare.physicsBody?.contactTestBitMask = redCircleCategory
```

No necesitamos `redCircle.physicsBody?.contactTestBitMask = blueSquareCategory`

Uso avanzado:

No se cubre aquí, pero los cuerpos físicos pueden pertenecer a más de una categoría. Por ejemplo, podríamos configurar nuestro juego de la siguiente manera:

```
let squareCategory: UInt32 = 1 << 0 // bitmask is ...00000001
let circleCategory: UInt32 = 1 << 1 // bitmask is ...00000010
let blueCategory: UInt32 = 1 << 2 // bitmask is ...00000100
let redCategory: UInt32 = 1 << 3 // bitmask is ...00001000
let purpleCategory: UInt32 = 1 << 4 // bitmask is ...00010000
let edgeCategory: UInt32 = 1 << 31 // bitmask is 10000...0000000
```

A cada cuerpo de física se le asignan las categorías a las que pertenece:

```
//Assign our category bit masks to our physics bodies
purpleSquare.physicsBody?.categoryBitMask = squareCategory | purpleCategory
redCircle.physicsBody?.categoryBitMask = circleCategory | redCategory
blueSquare.physicsBody?.categoryBitMask = squareCategory | blueCategory
```

su `categoryBitMasks` ahora son:

```
purpleSquare.physicsBody?.categoryBitMask = ...00010001
redCircle.physicsBody?.categoryBitMask = ...00001010
blueSquare.physicsBody?.categoryBitMask = ...00000101
```

Esto afectará cómo manipulas los campos de bits. Puede ser útil (por ejemplo) para indicar que un cuerpo de física (por ejemplo, una bomba) ha cambiado de alguna manera (por ejemplo, podría haber ganado la habilidad 'super' que es otra categoría, y usted puede verificar que un determinado objeto (una madre alienígena)

Lea [SKNode Collision en línea](https://riptutorial.com/es/sprite-kit/topic/6261/sknode-collision): <https://riptutorial.com/es/sprite-kit/topic/6261/sknode-collision>

Capítulo 8: SKScene

Observaciones

SKScene representa una sola escena en una aplicación SpriteKit. Un SKScene se 'presenta' en un [SKView](#) . [SKSpriteNodes](#) se agregan a la escena para implementar los sprites reales.

Las aplicaciones simples pueden tener un solo SKScene que contiene todo el contenido de SpriteKit. Las aplicaciones más complejas pueden tener varias escenas SKS que se presentan en diferentes momentos (p. Ej., Una escena inicial para presentar las opciones del juego, una segunda escena para implementar el juego en sí y una tercera escena para presentar los resultados de 'Game Over').

Examples

Subclasificando SKScene para implementar la funcionalidad principal de SpriteKit

La funcionalidad SpriteKit se puede implementar en una subclase de SKScene. Por ejemplo, un juego puede implementar la funcionalidad principal del juego dentro de una subclase de SKScene llamada GameScene.

En Swift :

```
import SpriteKit

class GameScene: SKScene {

    override func didMoveToView(view: SKView) {
        /* Code here to setup the scene when it is first shown. E.g. add sprites. */
    }

    override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?) {
        for touch in touches {
            let location = touch.locationInNode(self)
            /* Code here to respond to a user touch in the scene at location */
        }
    }

    override func update(currentTime: CFTimeInterval) {
        /* Code here to perform operations before each frame is updated */
    }
}
```

La funcionalidad secundaria podría luego implementarse en las subclases de los SKSpriteNodes que se usan dentro de la escena (ver [Subclase de SKSpriteNode](#)).

Crear un SKScene que llene el SKView

Un simple caso de uso es crear un SKScene que llene exactamente el SKView. Esto evita la necesidad de considerar escalar la vista para ajustar o configurar una cámara para mostrar una parte de la escena.

El siguiente código asume que ya existe un SKView llamado skView (por ejemplo, como se define en [Crear un SKView de pantalla completa utilizando Interface Builder](#)) y se ha definido una subclase de SKScene llamada GameView:

En Swift :

```
let sceneSize = CGSizeMake(skView.frame.width, skView.frame.height)
let scene = SKScene(size: sceneSize)

skView.presentScene(scene)
```

Sin embargo, si el SKView puede cambiar de tamaño (por ejemplo, si el usuario gira su dispositivo y esto hace que la vista se estire debido a sus limitaciones), el SKScene ya no se ajustará al SKView. Puede administrar esto cambiando el tamaño de SKScene cada vez que SKView cambie de tamaño (por ejemplo, en el método didChangeSize).

Cree un SKScene que se adapte al SKView

Un SKScene tiene un parámetro **scaleMode** que define cómo cambiará su tamaño para que se ajuste al SKView que se presenta en el SKView si no tiene el mismo tamaño y / o forma.

Hay cuatro opciones para scaleMode:

- **AspectFit** : la escena se escala (pero no se estira) hasta que se ajusta a la vista. Esto garantiza que la escena no se distorsione, pero puede haber algunas áreas de la vista que no están cubiertas por la escena si la escena no tiene la misma forma que la vista.
- **AspectFill** : la escena se escala (pero no se estira) para llenar la vista completamente. Esto garantiza que la escena no esté distorsionada y que la vista esté completamente llena, pero algunas partes de la escena pueden recortarse si la escena no tiene la misma forma que la vista.
- **Rellenar** : la escena se escala (y, si es necesario, se estira) para completar la vista por completo. Esto asegura que la vista esté completamente llena y que ninguna parte de su escena esté recortada, pero la escena se distorsionará si la escena no tiene la misma forma que la vista.
- **ResizeFill** : la escena no se escala en absoluto, sino que se cambia su tamaño para ajustarse al tamaño de la vista.

El siguiente código asume que ya existe un SKView llamado skView (por ejemplo, como se define en [Crear un SKView de pantalla completa usando Interface Builder](#)) y se definió una subclase de SKScene llamada GameView y luego se usa el modo de **escala AspectFill** :

En Swift 3 :

```
let sceneSize = CGSize(width:1000, height:1000)
let scene = GameScene(size: sceneSize)
```

```
scene.scaleMode = .aspectFill

skView.presentScene(scene)
```

Cree un SKScene con un SKCameraNode (iOS 9 y posterior)

Puede colocar un SKCameraNode en un SKScene para definir qué parte de la escena se muestra en el SKView. Piense en el SKScene como un mundo en 2D con una cámara flotando sobre él: el SKView mostrará lo que "ve" la cámara.

Por ejemplo, la cámara podría estar conectada al sprite del personaje principal para seguir la acción de un juego de desplazamiento.

El SKCameraNode tiene cuatro parámetros que definen qué parte de la escena se muestra:

- **posición** : esta es la posición de la cámara en la escena. La escena se renderiza para colocar esta posición en el centro de SKView.
- **xScale** y **yScale** : definen cómo se amplía la escena en la vista. Mantenga estos dos valores iguales para evitar distorsionar la vista. Un valor de 1 significa que no hay zoom, los valores menores que uno se acercarán (hará que los sprites parezcan más grandes) y los valores superiores a 1 se alejarán (los sprites aparecerán más pequeños).
- **zRotación** : esto define cómo se gira la vista en la vista. Un valor de cero será sin rotación. El valor está en radianes, por lo que un valor de Pi (3.14 ...) girará la vista al revés.

El siguiente código asume que ya existe un SKView llamado skView (por ejemplo, como se define en [Crear un SKView de pantalla completa utilizando Interface Builder](#)) y se ha definido una subclase de SKScene llamada GameView. En este ejemplo, solo se establece la posición inicial de la cámara, debe mover la cámara (de la misma manera que lo haría con otros SKSpriteNodes en la escena) para desplazarse por su vista:

En Swift 3 :

```
let sceneSize = CGSize(width:1000, height:1000)
let scene = GameScene(size: sceneSize)
scene.scaleMode = .aspectFill

let camera = SKCameraNode()
camera.position = CGPointM(x:500, y:500)
camera.xScale = 1
camera.yScale = 1
camera.zRotation = 3.14
scene.addChild(camera)
scene.camera = camera

skView.presentScene(scene)
```

Lea SKScene en línea: <https://riptutorial.com/es/sprite-kit/topic/4519/skscene>

Capítulo 9: SKSpriteNode (Sprites)

Sintaxis

- conveniencia `init (nombre de imageNamed: String) // Crear un SKSpriteNode a partir de una imagen con nombre en el catálogo de activos`
- `var position: CGPoint // propiedad SKNode, heredada por SKSpriteNode. La posición del nodo en el sistema de coordenadas de los padres.`
- `func addChild (_ node: SKNode) // Método SKNode, heredado por SKScene. Se usa para agregar un SKSpriteNode a la escena (también se usa para agregar SKNodes a otros SKNodes).`

Examples

Añadiendo un Sprite a la escena

En SpriteKit, un Sprite está representado por la clase `SKSpriteNode` (que se hereda de `SKNode`).

En primer lugar, cree un nuevo proyecto de Xcode basado en la plantilla SpriteKit como se describe en [Su primer juego SpriteKit](#).

Creando un Sprite

Ahora puede crear un `SKSpriteNode` utilizando una imagen cargada en la carpeta `Assets.xcassets`.

```
let spaceship = SKSpriteNode(imageNamed: "Spaceship")
```

`Spaceship` es el nombre del elemento de imagen en `Assets.xcassets`.

Después de que se haya creado el sprite, puede agregarlo a su escena (o a cualquier otro nodo).

Abra `GameScene.swift`, elimine todo su contenido y agregue lo siguiente

```
class GameScene: SKScene {
    override func didMoveToView(view: SKView) {
        let enemy = SKSpriteNode(imageNamed: "Spaceship")
        enemy.position = CGPoint(x:self.frame.midX, y:self.frame.midY)
        self.addChild(enemy)
    }
}
```

Ahora presione `CMD + R` en Xcode para iniciar el simulador.



Subclase SKSpriteNode

Puede `SKSpriteNode` subclase de `SKSpriteNode` y definir su propio tipo de sprite.

```
class Hero: SKSpriteNode {
  //Use a convenience init when you want to hard code values
  convenience init() {
    let texture = SKTexture(imageNamed: "Hero")
    self.init(texture: texture, color: .clearColor(), size: texture.size())
  }
}
```

```
//We need to override this to allow for class to work in SpriteKit Scene Builder
required init?(coder aDecoder: NSCoder) {
    super.init(coder:aDecoder)
}

//Override this to allow Hero to have access all convenience init methods
override init(texture: SKTexture?, color: UIColor, size: CGSize)
{
    super.init(texture: texture, color: color, size: size)
}
}
```

Lea SKSpriteNode (Sprites) en línea: <https://riptutorial.com/es/sprite-kit/topic/3001/skspritenode--sprites->

Capítulo 10: SKView

Parámetros

Parámetro	Detalles
showsFPS	Muestra un recuento de la velocidad de fotogramas actual en fotogramas por segundo en la vista.
showsNodeCount	Muestra un recuento del número actual de SKNodes que se muestran en la vista.
muestraFísica	Muestra una representación visual de SKPhysicsBodys en la vista.
showsFields	Muestra una imagen que representa los efectos de los campos de física en la vista.
showsDrawCount	Muestre un recuento del número de pases de dibujo necesarios para representar la vista.
showsQuadCount	Muestra un recuento del número de rectángulos necesarios para representar la vista.

Observaciones

Un SKView es una subclase de UIView que se utiliza para presentar animaciones SpriteKit 2D.

Un SKView se puede agregar a Interface Builder o programáticamente de la misma manera que las UIViews "normales". El contenido de SpriteKit se presenta en el SKView en un SKScene.

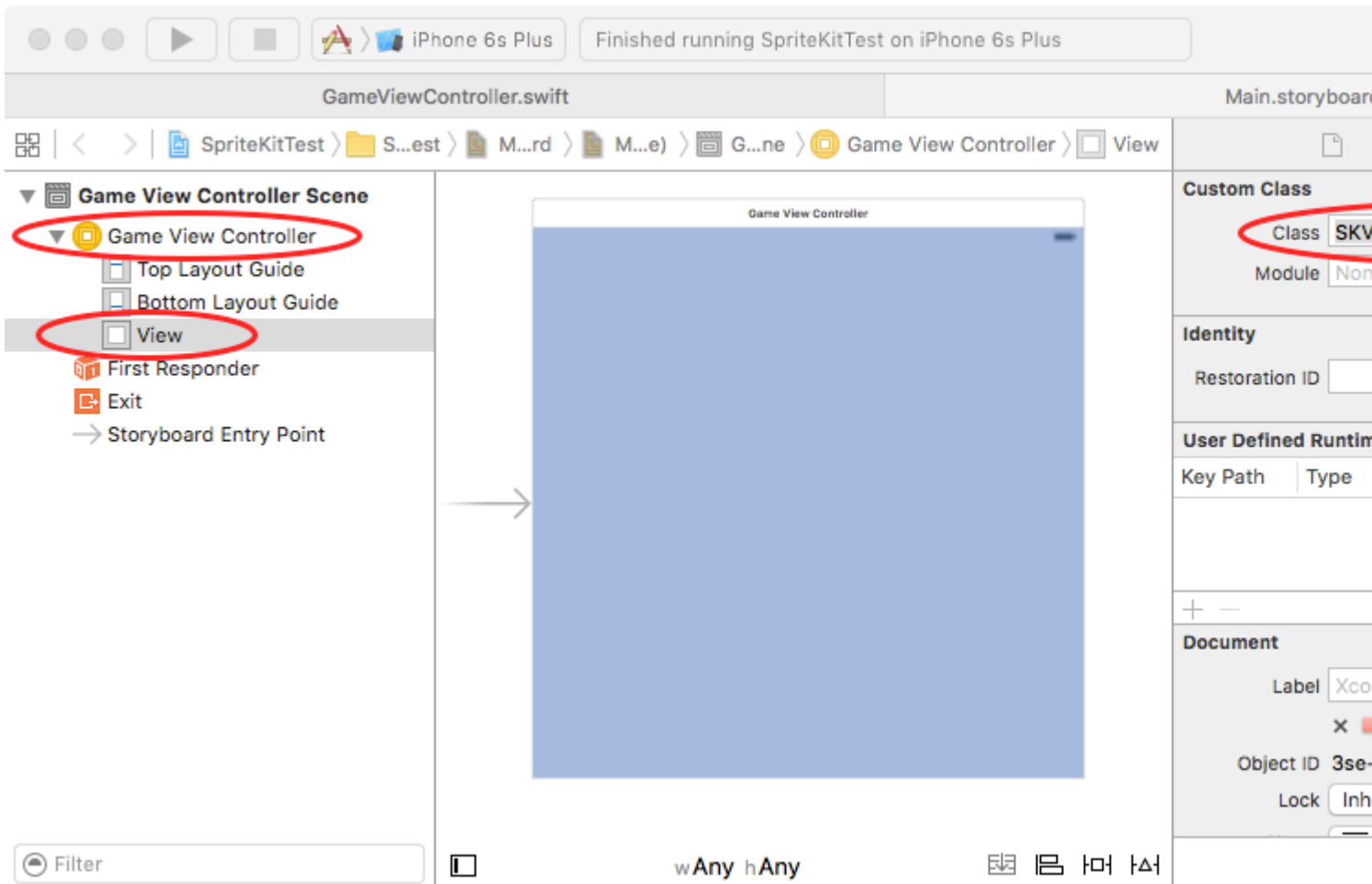
Ver también [Referencia de clase SKView](#) de la documentación de Apple.

Examples

Crear un SKView de pantalla completa utilizando Interface Builder

Un caso de uso típico para SpriteKit es donde el SKView llena toda la pantalla.

Para hacer esto en el Generador de Interfaz de Xcode, primero cree un ViewController normal, luego seleccione la vista contenida y cambie su **Clase** de **UIView** a **SKView** :



Dentro del código del Controlador de vista, en el método `viewDidLoad`, tome un enlace a este `SKView` usando `self.view`:

En Swift:

```
guard let skView = self.view as? SKView else {
    // Handle error
    return
}
```

(La declaración de protección aquí protege contra el error teórico de que la vista no es un `SKView`).

Luego puede usar esto para realizar otras operaciones, como presentar un `SKScene`:

En Swift:

```
skView.presentScene(scene)
```

Visualización de información de depuración

La velocidad de cuadros actual (en FPS, cuadros por segundo) y el número total de `SKNodes` en la escena (`nodeCount`, cada sprite es un `SKNode` pero otros objetos en la escena también son `SKNodes`) se pueden mostrar en la esquina inferior derecha de la vista .

Estos pueden ser útiles cuando están activados (configurados como verdaderos) para depurar y optimizar su código, pero deben estar desactivados (configurados como falsos) antes de enviar la aplicación a la AppStore.

En Swift:

```
skView.showsFPS = true  
skView.showsNodeCount = true
```

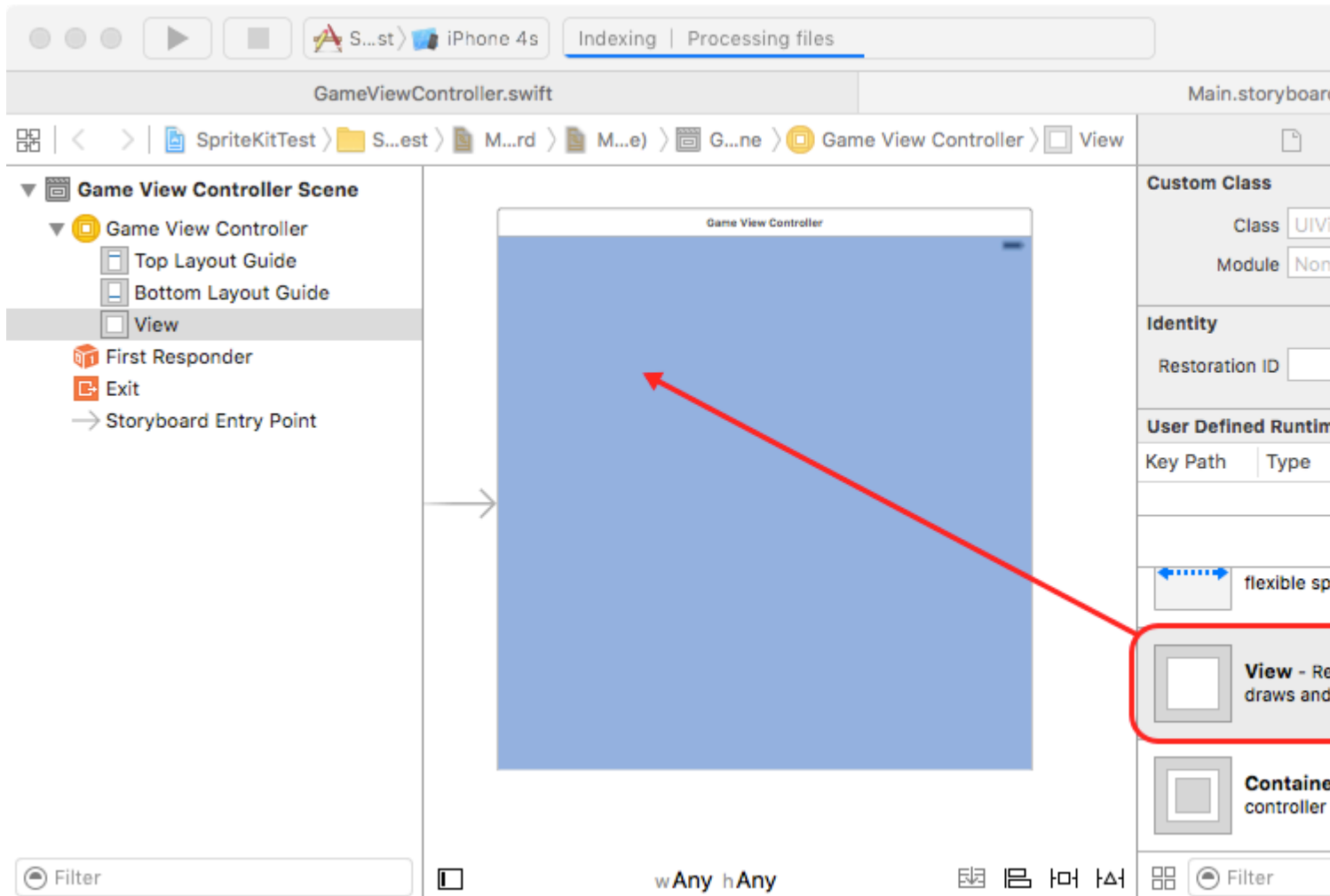
Resultado:



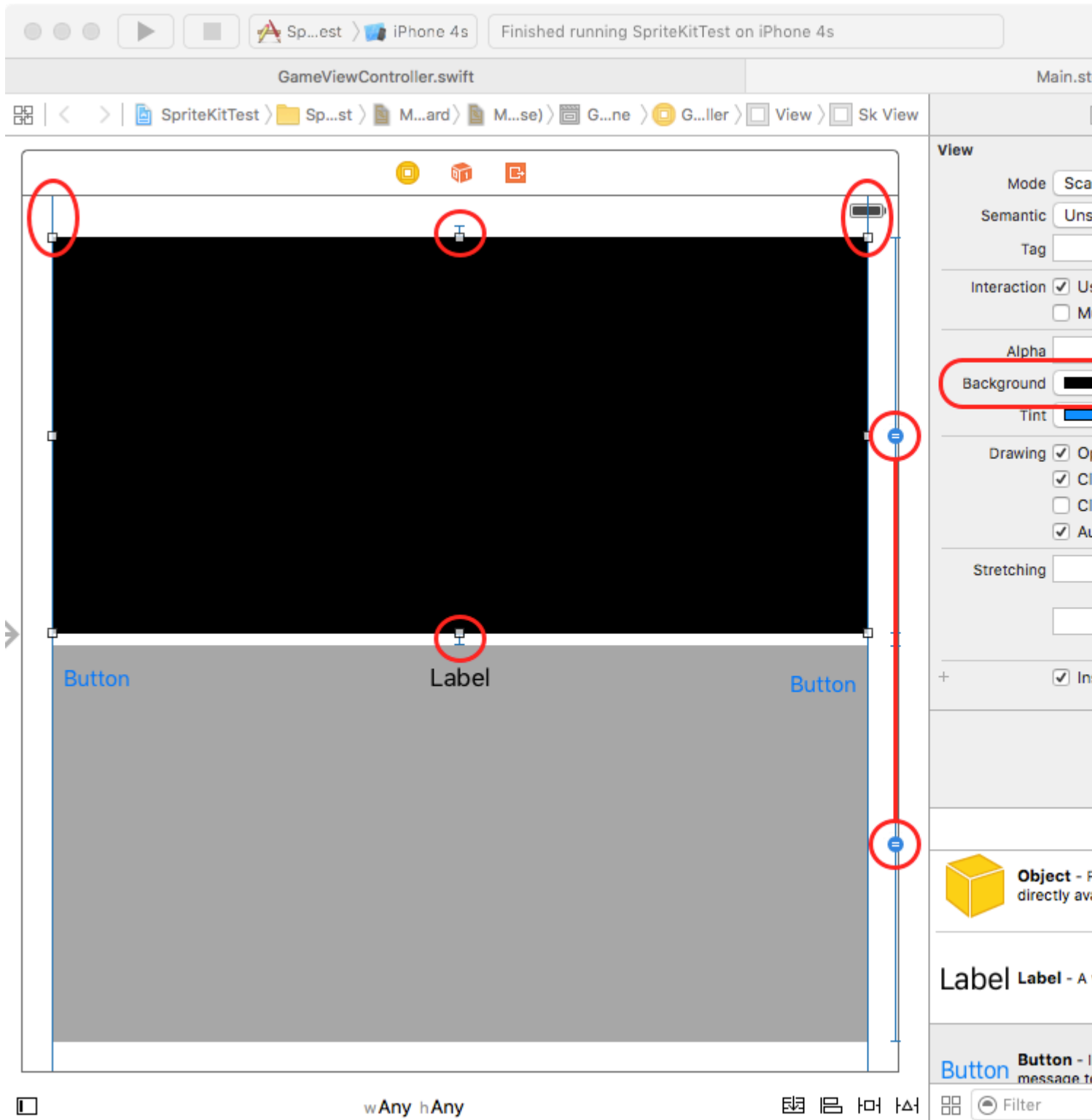
Crea un pequeño SKView con otros controles usando Interface Builder

Un SKView no necesita llenar toda la pantalla y puede compartir espacio con otros controles de la interfaz de usuario. Incluso puede tener más de un SKView mostrado a la vez si lo desea.

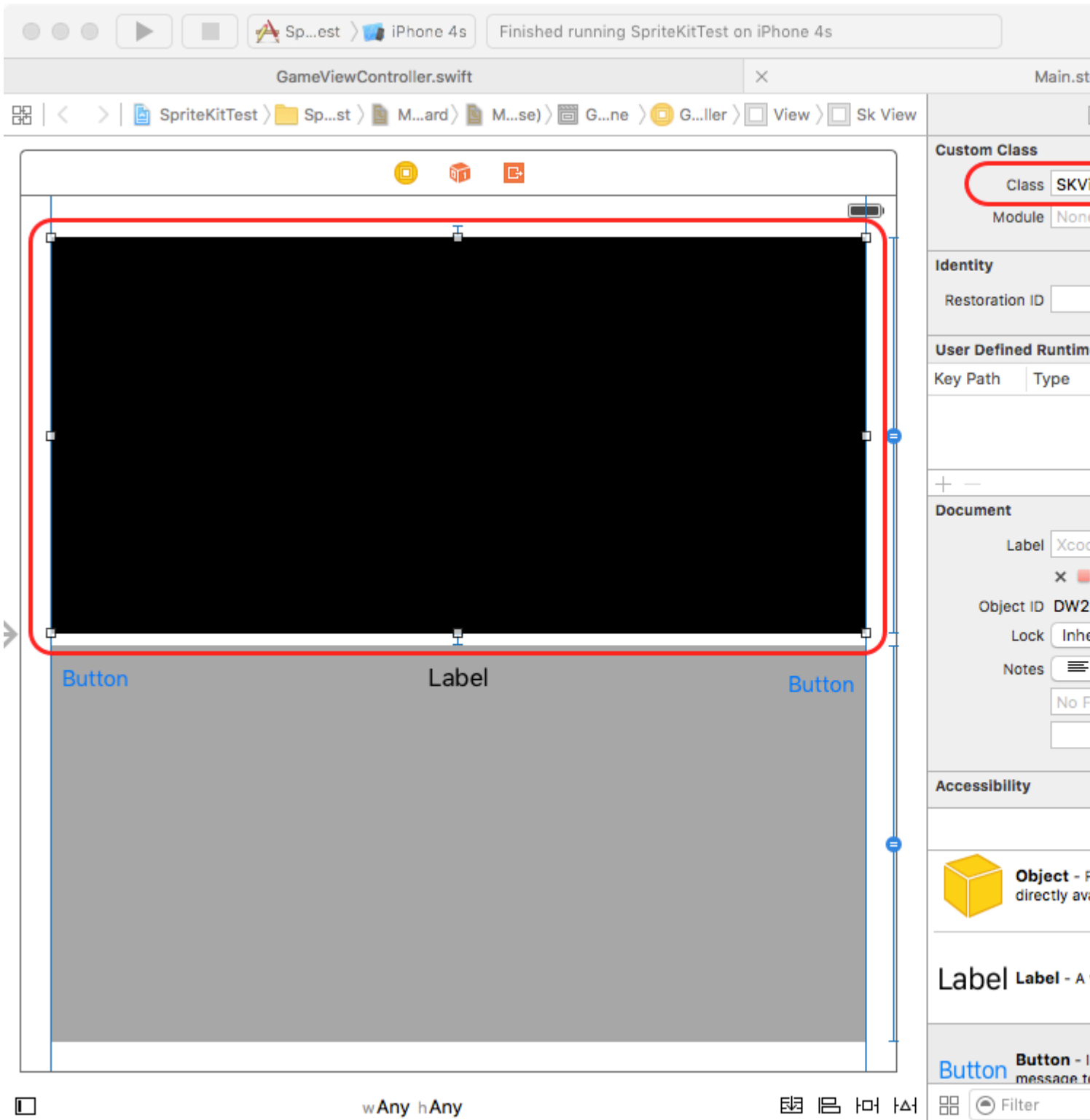
Para crear un SKView más pequeño entre otros controles con Interface Builder, primero cree un ViewController normal, luego arrastre y suelte una nueva vista en el controlador de vista:



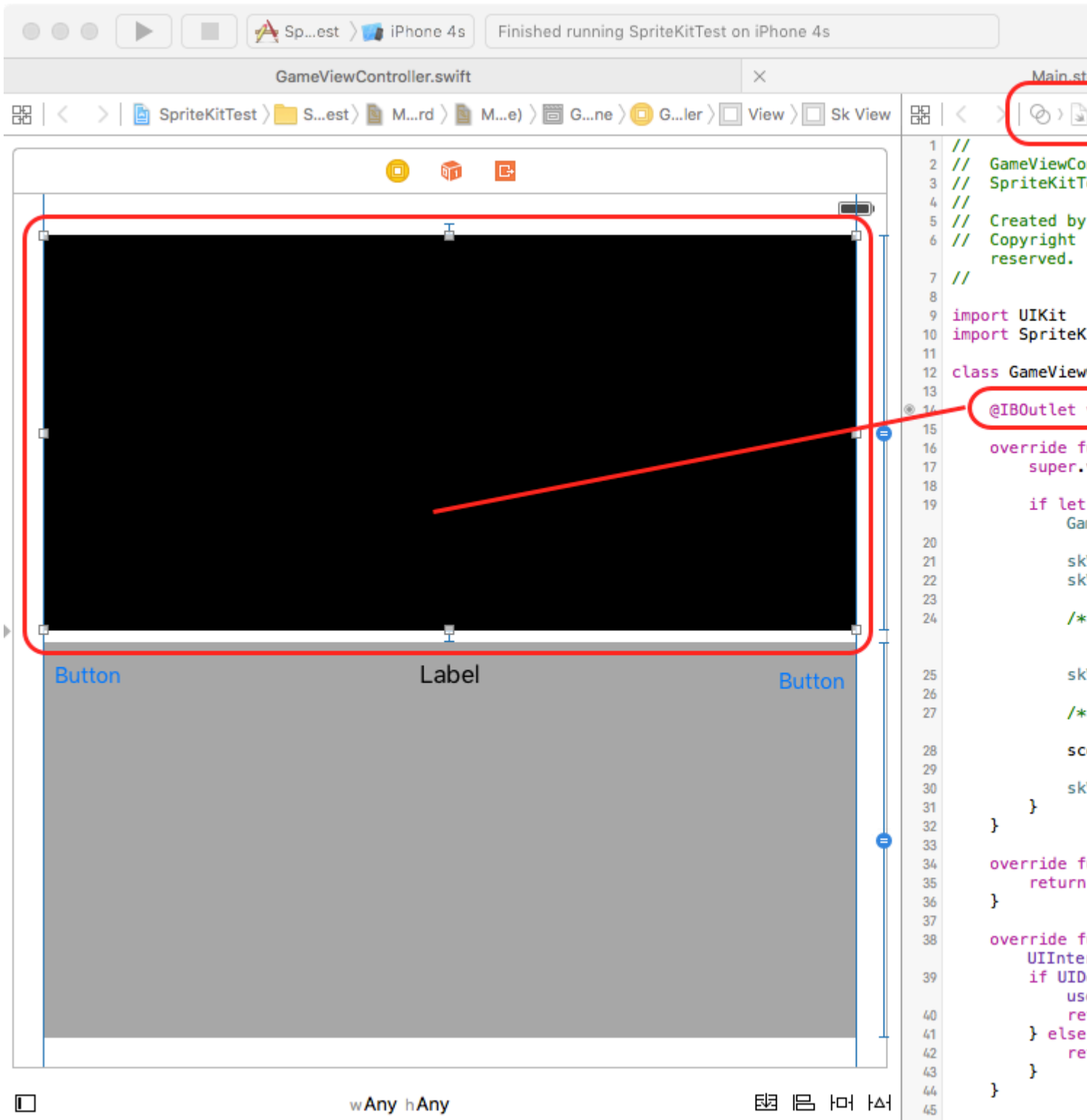
Puede ser útil establecer el color de esta vista en algo que no sea el blanco (aquí se usa el negro) para que se pueda ver más claramente en Interface Builder (este color no se mostrará en la aplicación final). Agregue otros controles (una UIView, dos botones y una etiqueta se muestran aquí como ejemplos) y use las restricciones como de costumbre para colocarlas en la pantalla:



Luego seleccione la vista que desea que sea un SKView y cambie su clase a SKView:



Luego, utilizando el editor asistente, arrastre y arrastre desde este SKView a su código y cree un Outlet:



Utilice esta salida para presentar su SKScene.

En Swift:

```
skView.presentScene(scene)
```

Resultado (basado en el ejemplo de [Hello World](#)):



Lea SKView en línea: <https://riptutorial.com/es/sprite-kit/topic/3572/skview>

Creditos

S. No	Capítulos	Contributors
1	Empezando con el kit de sprites	Ali Beadle , Community , Luca Angeletti
2	Detección de entrada táctil en dispositivos iOS	KnightOfDragon , Luca Angeletti
3	Elementos UIKit con SpriteKit	Alessandro Ornano , KnightOfDragon
4	Física	Alessandro Ornano
5	Funciones temporizadas en SpriteKit: SKActions vs NSTimers	KnightOfDragon
6	SKAcción	Abdou023 , Kendel
7	SKNode Collision	Chen Wei , Confused , KnightOfDragon , RamenChef , Steve Ives
8	SKScene	Ali Beadle
9	SKSpriteNode (Sprites)	Ali Beadle , KnightOfDragon , Luca Angeletti
10	SKView	Ali Beadle , Kendel