eBook Gratuit

APPRENEZ sprite-kit

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#sprite-kit

Table des matières

À propos

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.

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

Chapitre 1: Premiers pas avec le kit sprite

Remarques

SpriteKit est un moteur de jeu 2D développé par Apple. Il fournit des API de haut niveau et un large éventail de fonctionnalités aux développeurs. Il contient également un moteur physique interne.

Il est disponible sur toutes les plates-formes Apple

- iOS
- macOS
- tvos
- watchOS (> = 3.0)

Remarque: Si vous souhaitez développer en utilisant des graphiques 3D, vous devez utiliser SceneKit à la place.

Les éléments de base de SpriteKit sont:

- SKView : une vue dans laquelle les SKScenes sont présentés.
- SKScene : une scène 2D présentée dans un SKView et contenant un ou plusieurs SKSpriteNodes.
- SKSpriteNode : une image 2D individuelle pouvant être animée autour de la scène.

Les autres éléments constitutifs connexes sont les suivants:

- SKNode: nœud plus général pouvant être utilisé dans une scène pour regrouper d'autres nœuds pour un comportement plus complexe.
- SKAction: un ou plusieurs groupes d'actions appliqués à SKNodes pour implémenter des animations et d'autres effets.
- SKPhysicsBody - permet d'appliquer la physique aux SKNodes pour leur permettre de se comporter de manière réaliste, notamment en tombant sous l'effet de la gravité, en se faisant rebondir et en suivant des trajectoires balistiques.

Documentation officielle

Versions

iOS 7.0 et versions ultérieures

OS X 10.9 Mavericks et plus tard

watchOS 3.0 et plus tard

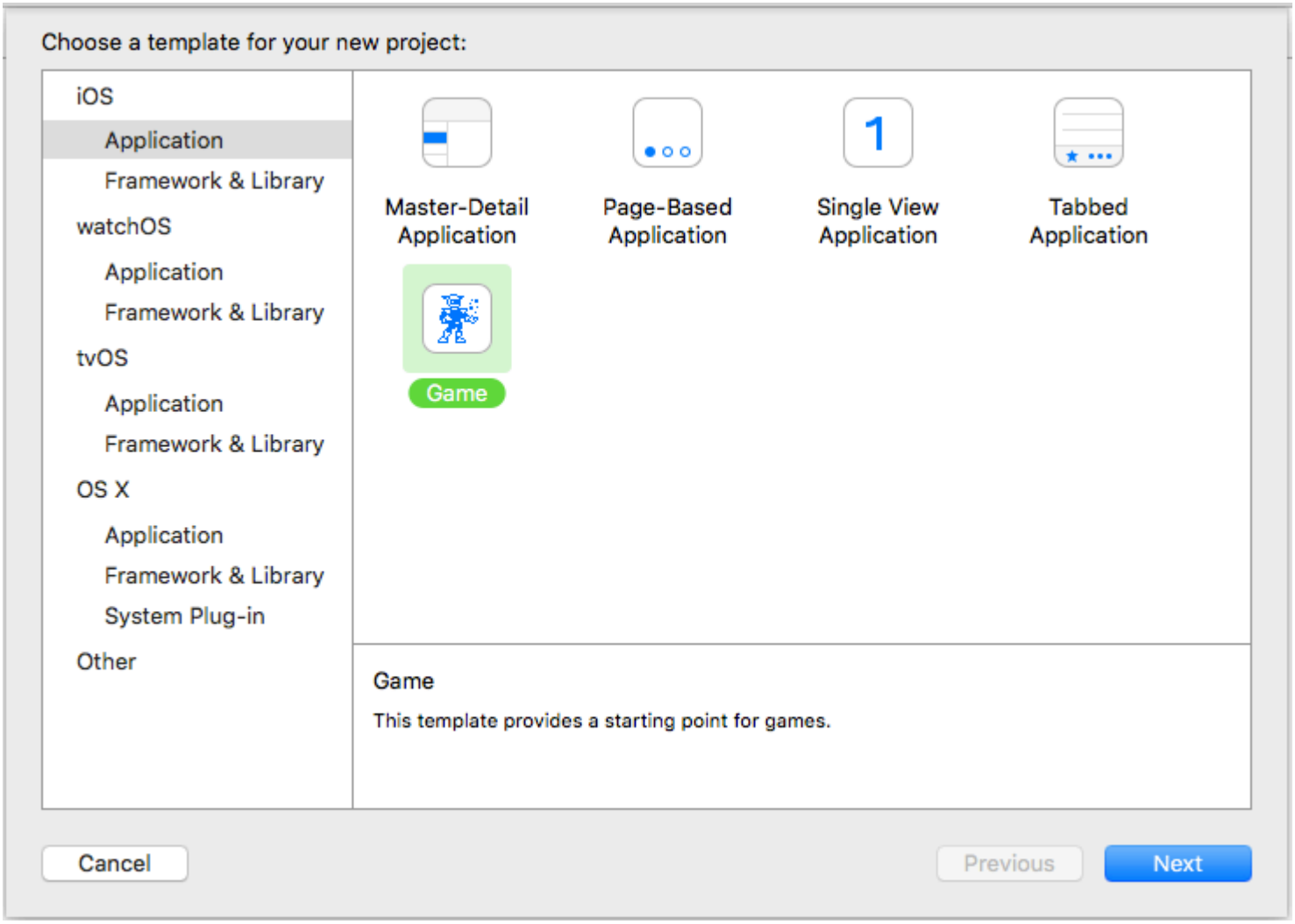
tvOS 9.0 et supérieur

Examples

Votre premier jeu SpriteKit (Hello World)

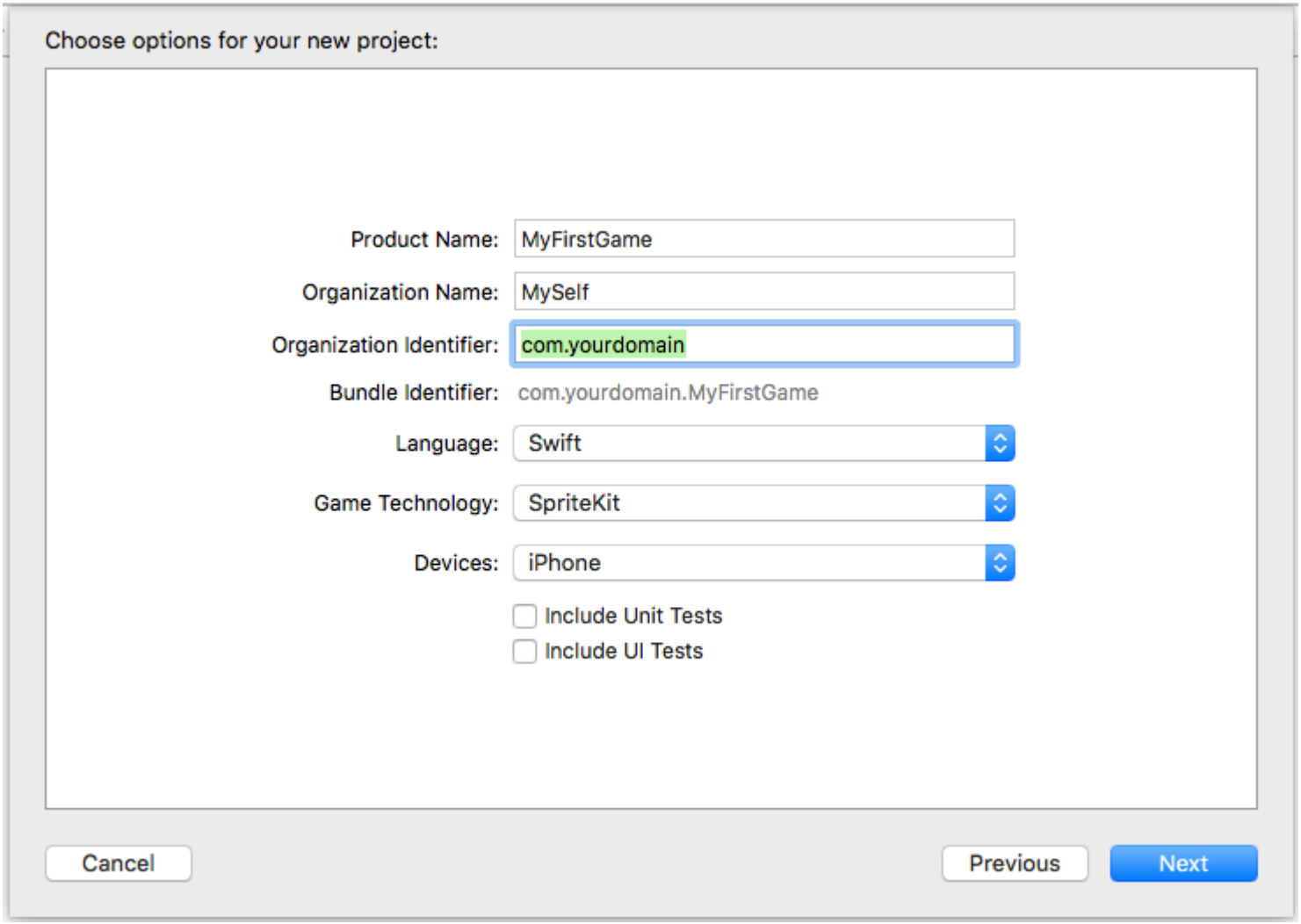
Ouvrez Xcode et sélectionnez `Create a new Xcode Project` .

Sélectionnez maintenant `iOS > Application` à gauche et `Game` dans la zone de sélection principale.



Appuyez sur **Suivant** .

- Ecrivez dans `Product Name` du `Product Name` le nom de votre premier grand jeu.
- Dans `Organization Name` le nom de votre entreprise (ou simplement votre propre nom).
- `Organisation Identifier` doit contenir votre nom de domaine *inversé* (`www.yourdomain.com` devient `com.yourdomain`). Si vous n'avez pas de domaine, écrivez ce que vous voulez (c'est juste et testez).
- Puis sélectionnez `Swift` , `SpriteKit` et `iPhone` .



Appuyez sur **Suivant** .

Sélectionnez un dossier de votre Mac où vous souhaitez enregistrer le projet et cliquez sur **Créer** .

Félicitations, vous créez votre premier jeu avec SpriteKit! Appuyez simplement sur `CMD` + `R` pour le lancer dans le simulateur!

Lire Premiers pas avec le kit sprite en ligne: https://riptutorial.com/fr/sprite-kit/topic/2956/premiers-pas-avec-le-kit-sprite

Chapitre 2: Détection des entrées tactiles sur les appareils iOS

Examples

Détection du toucher

Vous pouvez remplacer 4 méthodes de `SKScene` pour détecter le toucher de l'utilisateur

```
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?) {

    }
}
```

> Veuillez noter que chaque méthode reçoit un paramètre de `touches` qui (dans des circonstances particulières) peut contenir plusieurs événements tactiles.

Lire Détection des entrées tactiles sur les appareils iOS en ligne: https://riptutorial.com/fr/sprite-kit/topic/3660/detection-des-entrees-tactiles-sur-les-appareils-ios

Chapitre 3: Éléments UIKit avec SpriteKit

Examples

UITableView dans 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()
    }
}
```

Sortie :

Protocole / Délégué pour appeler un jeu La méthode ViewController à partir de la scène de jeu

Exemple de code *GameScene* :

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

Exemple de code *GameViewController* :

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

```
            if let scene = SKScene(fileNamed: "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)
    }
}
```

Sortie :

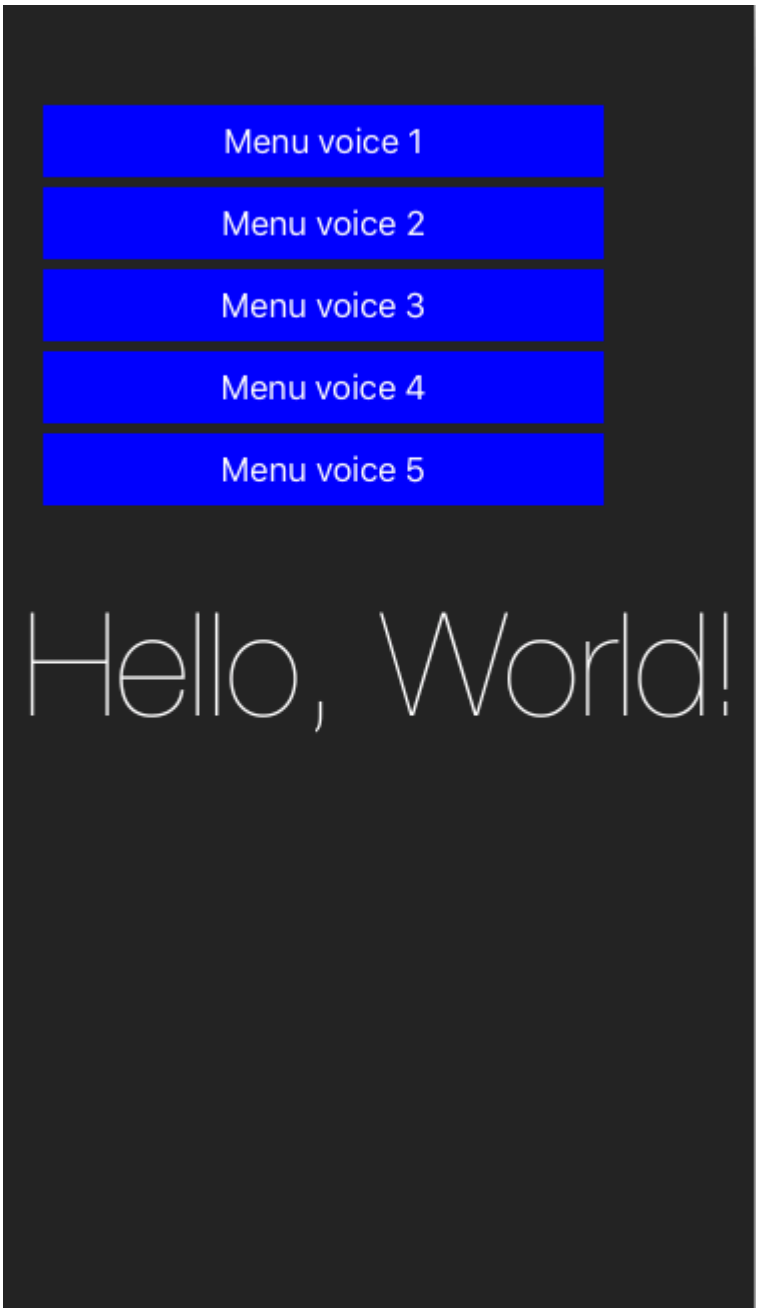
```
inputProperty is:  call game view controller method
```

StackView dans 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)
        }
    }
```

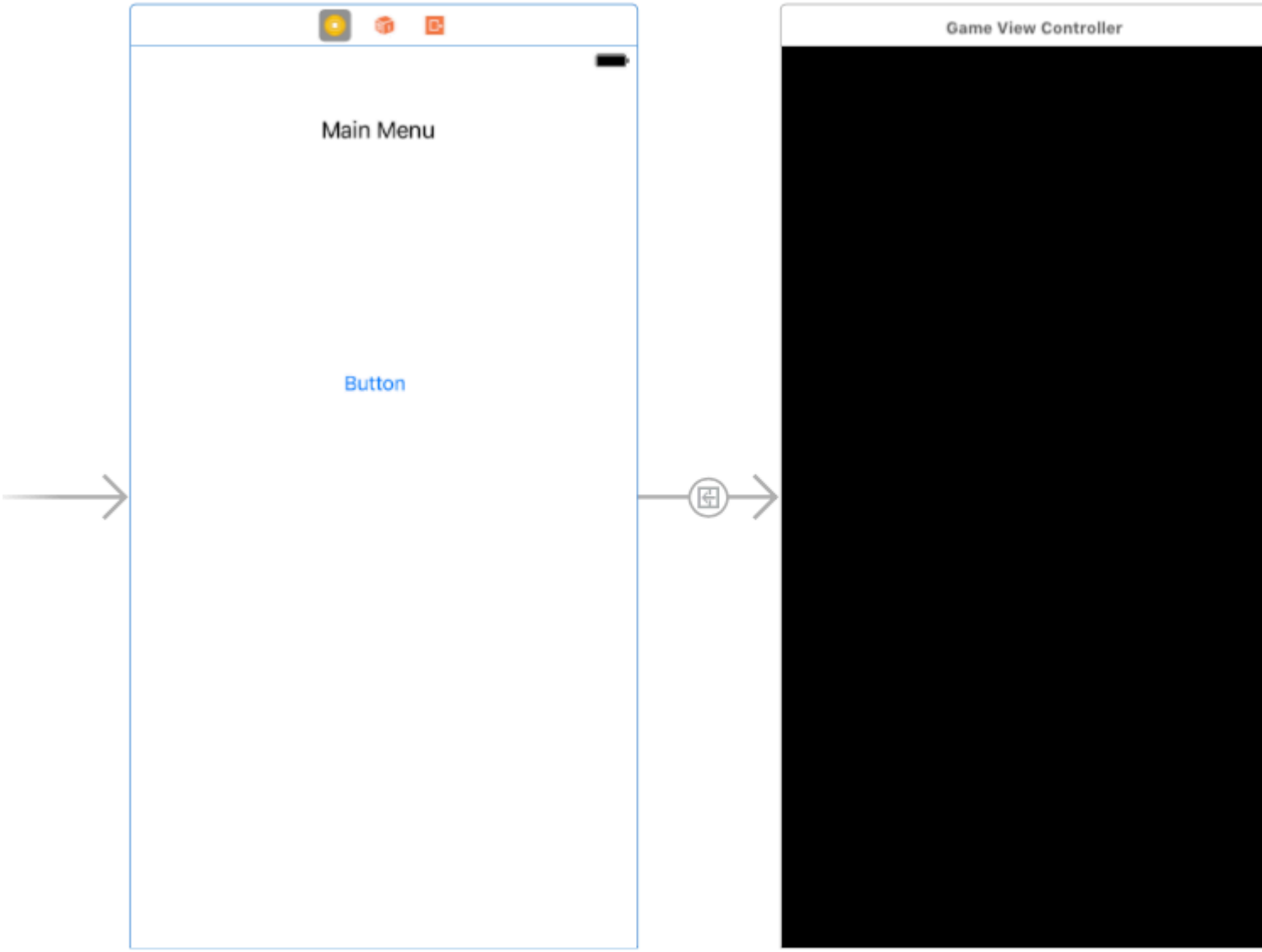
```
    func didTapOnView(_ gestureRecognizer: UIGestureRecognizer) {
        if let index = arrangedSubviews.index(of: gestureRecognizer.view!) {
            delegate?.didTapOnView(at: index)
        }
    }
}
class GameScene: SKScene, StackViewDelegate {
    var gameMenuView = GameMenuView()
    private 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
        }
    }
}
```

Sortie :



Multiple UIViewController dans un jeu: comment sauter de la scène à un viewController

Storyboard :



Initial viewController : un viewController vide avec un bouton pour présenter le GameViewController

GameViewController : le GameViewController typique du modèle de kit Sprite *"Hello World"* .

But : Je veux présenter le premier viewController de mon jeu `SKScene` avec la désallocation correcte de ma scène.

Description : Pour obtenir le résultat, j'ai étendu la classe `SKSceneDelegate` pour créer un `protocol/delegate` qui effectue la transition entre `GameViewController` et le premier contrôleur initial (menu principal). Cette méthode pourrait être étendue à d'autres contrôleurs de vue de votre jeu.

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(fileNamed: "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")
    }
}
```

Lire Éléments UIKit avec SpriteKit en ligne: https://riptutorial.com/fr/sprite-kit/topic/8807/elements-uikit-avec-spritekit

Chapitre 4: Fonctions temporelles dans SpriteKit: SKActions vs NSTimers

Remarques

Quand devriez-vous utiliser `SKAction` s pour exécuter les fonctions de minuterie? Presque toujours. La raison en est que `Sprite Kit` fonctionne sur un intervalle de mise à jour et que la vitesse de cet intervalle peut être modifiée pendant toute la durée de vie du processus en utilisant la propriété `speed` . Les scènes peuvent également être interrompues. Depuis que `SKAction` travaille dans la scène, lorsque vous modifiez ces propriétés, vous n'avez pas besoin de modifier vos fonctions temporelles. Si votre scène dure 0,5 seconde et que vous suspendez la scène, vous n'avez pas besoin d'arrêter les minuteries et de conserver cette différence de 0,5 seconde. Il vous est donné automatiquement, de sorte que lorsque vous ne le faites pas, le temps restant continue.

Quand devriez-vous utiliser `NSTimer` pour effectuer des fonctions de minuterie? Chaque fois que vous avez quelque chose qui doit être chronométré en dehors de l'environnement `SKScene` , et doit également être déclenché même lorsque la scène est en pause, ou doit se déclencher à un rythme constant même lorsque la vitesse de la scène change.

Ceci est préférable lorsque vous travaillez avec les contrôles `UIKit` et les contrôles `SpriteKit` . Comme `UIKit` n'a aucune idée de ce qui se passe avec `SpriteKit` , `NSTimer` se déclenche quel que soit l'état du `SKScene` . Un exemple serait que nous ayons un `UILabel` qui reçoit une mise à jour toutes les secondes et qui a besoin de données provenant de votre `SKScene` .

Examples

Implémenter une méthode qui se déclenche après une seconde

SKAction:

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

NSTimer:

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

Lire Fonctions temporelles dans SpriteKit: SKActions vs NSTimers en ligne: https://riptutorial.com/fr/sprite-kit/topic/5962/fonctions-temporelles-dans-spritekit--skactions-vs-nstimers

Chapitre 5: La physique

Examples

Comment supprimer correctement le noeud dans la méthode didBeginContact (plusieurs contacts)

```
// 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]()
    }
}
```

Lire La physique en ligne: https://riptutorial.com/fr/sprite-kit/topic/8991/la-physique

Chapitre 6: SKAction

Examples

Créer et exécuter un SKAction simple

Un exemple très simple serait de supprimer un SKSpriteNode.

En Swift:

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

Création d'une séquence d'actions répétée

Parfois, il est nécessaire de faire une action sur une répétition ou une séquence. Cet exemple rendra le nœud en fondu et sortira au total 3 fois.

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)
```

Exécution d'un bloc de code dans un SKAction

Un cas utile est que l'action exécute un bloc de code.

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)
```

Actions nommées pouvant être lancées ou supprimées ailleurs.

Parfois, vous souhaitez démarrer ou supprimer une action sur un nœud spécifique à un moment donné. Par exemple, vous pouvez vouloir arrêter un objet en mouvement lorsque l'utilisateur appuie sur l'écran. Cela devient très utile lorsqu'un nœud a plusieurs actions et que vous

souhaitez uniquement accéder à l'un d'entre eux.

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

Ici, nous définissons la clé "moveX" pour le `move` action afin d'y accéder ultérieurement dans une autre partie de la classe.

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

Lorsque l'utilisateur touche l'écran, l'action sera supprimée et l'objet cessera de bouger.

Lire SKAction en ligne: https://riptutorial.com/fr/sprite-kit/topic/6229/skaction

Chapitre 7: SKNode Collision

Remarques

Les déterminants de la collision Sprite Kit et du traitement des événements de contact sont les paramètres de relation que vous avez créés, `categoryBitMask` , `collisionBitMask` et `contactTestBitMask` pour chacun de vos types d'objet en interaction. En les définissant rationnellement en fonction des résultats escomptés des contacts et des collisions, vous déterminez quels types peuvent entrer en collision et informer des contacts avec d'autres, et éviter les collisions, les contacts et le traitement physique indésirables.

Pour chaque type d'entité, vous pouvez définir tous les trois:

1. `categoryBitMask` : une catégorie spécifique à ce type de noeud
2. `collisionBitMask` : un différenciateur de collision, peut être différent du précédent
3. `contactTestBitMask` : un différenciateur de contact, peut être différent des deux `contactTestBitMask`

Les étapes générales pour implémenter des collisions et des contacts sont les suivantes:

1. définir la taille physique du corps, la forme et (parfois) la masse
2. ajouter les BitMasks nécessaires pour votre type de noeud à partir de la catégorie, de la collision et du contact ci-dessus
3. mettre en scène en tant que délégué de contact lui permettant de vérifier et d'informer des collisions et des contacts
4. mettre en œuvre des gestionnaires de contacts et toute autre logique pertinente pour les événements de physique

Examples

Activer le monde de la physique

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

Activer le nœud pour entrer en collision

Tout d'abord, nous définissons la catégorie de nœud

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

Ajoutez ensuite le nœud du type de sol et le nœud du type de boîte.

```
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)
```

Gérer les contacts

Définir la scène en tant que délégué

```
//set your scene as SKPhysicsContactDelegate

class yourScene: SKScene, SKPhysicsContactDelegate

self.physicsWorld.contactDelegate = self;
```

Ensuite, vous devez implémenter l'une ou l'autre des fonctions de contact: option func didBegin (contact :) et / ou facultatif fund didEnd (contact :) méthode pour remplir votre logique de contact, par exemple comme

```
//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)

   }
}
```

Alternative aBeginContact

Si vous utilisez des catégories simples, avec chaque corps physique appartenant à une seule catégorie, alors cette forme alternative de didBeginContact peut être plus lisible:

```
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")
}
}
```

Projet Simple Sprite Kit montrant des collisions, des contacts et des événements tactiles.

Voici un simple Sprite-Kit GameScene.swift. Créez un nouveau projet SpriteKit vide et remplacez-le par GameScene.swift. Ensuite, construisez et exécutez.

Cliquez sur l'un des objets à l'écran pour les faire bouger. Vérifiez les journaux et les commentaires pour voir ceux qui entrent en collision et ceux qui entrent en contact.

```
//
//  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 collion and contctTest
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 collisonmask 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 collisonmask 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)")}
                }
            }
        }
    }
}
```

Alternative au traitement des contacts avec les sprites multi-catégories

```
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:()

}
```

Différence entre contacts et collisions

Dans Sprite-Kit, il y a le concept de **collision** qui fait référence au moteur physique SK qui interagit avec la façon dont les objets physiques interagissent lorsqu'ils entrent en collision, par exemple ceux qui rebondissent sur les autres.

Il a également le concept de **contacts** , qui est le mécanisme par lequel votre programme est informé lorsque 2 objets physiques se croisent.

Les objets peuvent entrer en collision mais ne pas générer de contacts, générer des contacts sans entrer en collision, ou entrer en collision et générer un contact (ou ne pas interagir ni interagir du tout)

Les collisions peuvent également être unilatérales, c'est-à-dire que l'objet A peut entrer en collision (rebondir) avec l'objet B, tandis que l'objet B continue comme si rien ne s'était passé. Si vous voulez que deux objets se renversent, ils doivent tous les deux être contraints d'entrer en collision avec l'autre.

Les contacts ne sont toutefois pas à sens unique; si vous voulez savoir quand l'objet A a touché (contacté) l'objet B, il suffit de configurer la détection de contact sur l'objet A par rapport à l'objet

B. Il n'est pas nécessaire de configurer la détection de contact sur l'objet B pour l'objet A.

Manipulation des masques de bit contactTest et collison pour activer / désactiver le contact et les collisions spécifiques.

Pour cet exemple, nous utiliserons 4 corps et ne montrerons que les 8 derniers bits des masques de bits pour plus de simplicité. Les 4 corps sont 3 SKSpriteNodes, chacun avec un corps physique et une limite:

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

Notez que le corps physique "bord" est le corps physique de la scène, pas un nœud.

Nous définissons 4 catégories uniques

```
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
```

Chaque corps de physique se voit attribuer les catégories auxquelles il appartient:

```
        //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 de la propriété collisionBitMask d'un corps est défini sur 1, alors il se heurte (rebondit) à tout corps ayant un "1" à la même position dans son objet categoryBitMask. De même pour contactTestBitMask.

À moins d'indication contraire, tout se heurte à tout le reste et aucun contact n'est généré (votre code ne sera pas notifié lorsque quelque chose entrera en contact avec quelque chose d'autre):

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

Chaque bit dans chaque position est '1', donc quand on le compare à n'importe quel autre categoryBitMask, Sprite Kit trouvera un '1' et une collision se produira. Si vous ne voulez pas que ce corps entre en collision avec une certaine catégorie, vous devrez définir le bon bit dans le collisonBitMask sur '0'

et son `contactTestbitMask` est défini sur tous les `0` s:

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

Identique à collisionBitMask, sauf inversé.

Les contacts ou les collisions entre les corps peuvent être **désactivés** (laissant le contact ou la collision existants inchangés) en utilisant:

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

Nous avons logiquement le masque de bit de collision de ET nodeA avec l'inverse (l'opérateur logique, l'opérateur ~) du masque de catégorie de nodeB pour "désactiver" ce bitMask du bit nodeA. par exemple pour empêcher le cercle rouge d'entrer en collision avec le carré violet:

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

qui peut être raccourci à:

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

Explication:

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

redCircle n'entre plus en collision avec des corps avec une catégorie de .... 0001 (purpleSquare)

Au lieu de désactiver les bits individuels dans le collsionsbitMask, vous pouvez le définir directement:

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

c'est-à-dire `blueSquare.physicsBody?.collisionBitMask = (....00000010 OR ....00000001)`

qui équivaut à `blueSquare.physicsBody?.collisionBitMask = ....00000011`

blueSquare n'entrera en collision qu'avec des corps avec une catégorie ou ..01 ou ..10

Les contacts ou les collisions entre les 2 corps peuvent être activés (sans affecter les contacts existants ou collisions) à tout moment en utilisant:

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

Nous avons logiquement le bitMask de AND redCircle avec le masque de bits de la catégorie purpleSquare pour "activer" ce bit dans le bitMask de redcircle. Cela laisse les autres bits du bitMas de redCircel non affectés.

Vous pouvez vous assurer que chaque forme «rebondit» sur un bord de l'écran comme suit:

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

Remarque:

Les collisions peuvent être unilatérales, c'est-à-dire que l'objet A peut entrer en collision (rebondir) avec l'objet B, tandis que l'objet B continue comme si rien ne s'était passé. Si vous voulez que deux objets se rejoignent, ils doivent tous les deux se heurter à l'autre:

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

Les contacts ne sont toutefois pas à sens unique; si vous voulez savoir quand l'objet A a touché (contacté) l'objet B, il suffit de configurer la détection de contact sur l'objet A par rapport à l'objet B. Il n'est pas nécessaire de configurer la détection de contact sur l'objet B pour l'objet A.

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

Nous n'avons pas besoin de `redcircle.physicsBody?.contactTestBitMask= blueSquareCategory`

Utilisation avancée:

Non couvert ici, mais les corps physiques peuvent appartenir à plusieurs catégories. Par exemple, nous pourrions configurer notre jeu comme suit:

```
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
```

Chaque corps de physique se voit attribuer les catégories auxquelles il appartient:

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

leurs categorybitMasks sont maintenant:

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

Cela affectera la façon dont vous manipulez les champs de bits. Cela peut être utile (par exemple) d'indiquer qu'un corps physique (par exemple une bombe) a changé (par exemple, il pourrait avoir la capacité «super» qui est une autre catégorie, et vous pouvez vérifier qu'un certain objet

Lire SKNode Collision en ligne: https://riptutorial.com/fr/sprite-kit/topic/6261/sknode-collision

Chapitre 8: SKScene

Remarques

SKScene représente une scène unique dans une application SpriteKit. Un SKScene est présenté dans un SKView . SKSpriteNodes sont ajoutés à la scène pour implémenter les sprites réels.

Les applications simples peuvent avoir un SKScene unique contenant tout le contenu SpriteKit. Des applications plus complexes peuvent avoir plusieurs SKScen présentés à des moments différents (par exemple, une scène d'ouverture pour présenter les options de jeu, une deuxième scène pour implémenter le jeu lui-même et une troisième scène pour présenter les résultats de Game Over).

Examples

Sous-classement de SKScene pour implémenter la fonctionnalité SpriteKit principale

La fonctionnalité SpriteKit peut être implémentée dans une sous-classe de SKScene. Par exemple, un jeu peut implémenter la fonctionnalité principale du jeu dans une sous-classe SKScene appelée 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 fonctionnalité secondaire pourrait alors être implémentée dans des sous-classes des SKSpriteNodes utilisées dans la scène (voir Sous-classement de SKSpriteNode).

Créer un SKScene qui remplit le SKView

Un cas simple pour créer un SKScene qui remplit exactement le SKView. Cela évite d'avoir à considérer la mise à l'échelle de la vue pour l'ajuster ou à définir une caméra pour afficher une partie de la scène.

Le code suivant suppose qu'un SKView appelé skView existe déjà (par exemple, tel que défini dans Créer un SKView en plein écran à l'aide d'Interface Builder) et qu'une sous-classe de SKScene appelée GameView a été définie:

En Swift :

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

skView.presentScene(scene)
```

Cependant, si SKView peut changer de taille (par exemple, si l'utilisateur fait pivoter son appareil et que cela entraîne l'étirement de la vue en raison de ses contraintes), le SKScene ne sera plus compatible avec SKView. Vous pouvez gérer cela en redimensionnant le SKScene chaque fois que SKView change de taille (par exemple, dans la méthode didChangeSize).

Créez un SKScene adapté à SKView

Un SKScene a un paramètre **scaleMode** qui définit la manière dont il changera de taille pour qu'il corresponde au SKView qu'il présente dans SKView s'il n'a pas la même taille et / ou la même forme.

Il y a quatre options pour scaleMode:

- **AspectFit** : la scène est mise à l'échelle (mais pas étirée) jusqu'à ce qu'elle rentre dans la vue. Cela garantit que la scène n'est pas déformée, mais il peut y avoir certaines zones de la vue qui ne sont pas couvertes par la scène si la scène n'a pas la même forme que la vue.
- **AspectFill** : la scène est mise à l'échelle (mais pas étirée) pour remplir complètement la vue. Cela garantit que la scène n'est pas déformée et que la vue est complètement remplie, mais que certaines parties de la scène peuvent être tronquées si la scène n'a pas la même forme que la vue.
- **Remplir** : la scène est mise à l'échelle (et si nécessaire étirée) pour remplir complètement la vue. Cela garantit que la vue est complètement remplie et qu'aucune de vos scènes n'est recadrée mais que la scène sera déformée si la scène n'a pas la même forme que la vue.
- **ResizeFill** : la scène n'est pas du tout dimensionnée, mais sa taille est modifiée pour s'adapter à la taille de la vue.

Le code suivant suppose qu'un SKView appelé skView existe déjà (par exemple, tel que défini dans Créer un SKView plein écran à l'aide d'Interface Builder) et qu'une sous-classe de SKScene appelée GameView a été définie, puis utilise le scaleMode **AspectFill** :

Dans Swift 3 :

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

```
    scene.scaleMode = .aspectFill

    skView.presentScene(scene)
```

Créer un SKScene avec un SKCameraNode (iOS 9 et versions ultérieures)

Vous pouvez placer un SKCameraNode dans un SKScene pour définir quelle partie de la scène est affichée dans SKView. Pensez au SKScene comme à un monde en 2D avec une caméra flottant au-dessus: le SKView montre ce que la caméra «voit».

Par exemple, la caméra pourrait être attachée au sprite du personnage principal pour suivre l'action d'un jeu défilant.

Le SKCameraNode a quatre paramètres qui définissent quelle partie de la scène est affichée:

- **position** : c'est la position de la caméra dans la scène. La scène est rendue pour placer cette position au milieu du SKView.
- **xScale** et **yScale** : définissent le zoom de la scène dans la vue. Gardez ces deux valeurs identiques pour éviter de déformer la vue. Une valeur de 1 signifie qu'il n'y a pas de zoom, que les valeurs inférieures à 1 font un zoom avant (rendent les images-objets plus grandes) et les valeurs supérieures à 1 font un zoom arrière (rendent les images-objets plus petites).
- **zRotation** : définit la rotation de la vue dans la vue. Une valeur de zéro ne sera pas une rotation. La valeur est en radians, donc une valeur de Pi (3.14 ...) fera pivoter la vue à l'envers.

Le code suivant suppose qu'un SKView appelé skView existe déjà (par exemple, tel que défini dans Créer un SKView plein écran à l'aide d'Interface Builder) et qu'une sous-classe de SKScene appelée GameView a été définie. Cet exemple définit simplement la position initiale de la caméra, vous devrez déplacer la caméra (de la même manière que vous le feriez avec d'autres SKSpriteNodes dans la scène) pour faire défiler votre vue:

Dans 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)
```

Lire SKScene en ligne: https://riptutorial.com/fr/sprite-kit/topic/4519/skscene

Chapitre 9: SKSpriteNode (Sprites)

Syntaxe

- commod init (imageNamed name: String) // Crée un SKSpriteNode à partir d'une image nommée dans le catalogue d'actifs
- var position: propriété CGPoint // SKNode, héritée par SKSpriteNode. La position du nœud dans le système de coordonnées des parents.
- func addChild (_ node: SKNode) // Méthode SKNode, héritée par SKScene. Utilisé pour ajouter un SKSpriteNode à la scène (également utilisé pour ajouter SKNodes à d'autres SKNodes).

Examples

Ajouter un sprite à la scène

Dans SpriteKit, un Sprite est représenté par la classe `SKSpriteNode` (qui hérite de `SKNode`).

Tout d'abord, créez un nouveau projet Xcode basé sur le modèle SpriteKit, comme décrit dans Votre premier jeu SpriteKit .

Créer un sprite

Vous pouvez maintenant créer un SKSpriteNode à l'aide d'une image chargée dans le dossier `Assets.xcassets` .

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

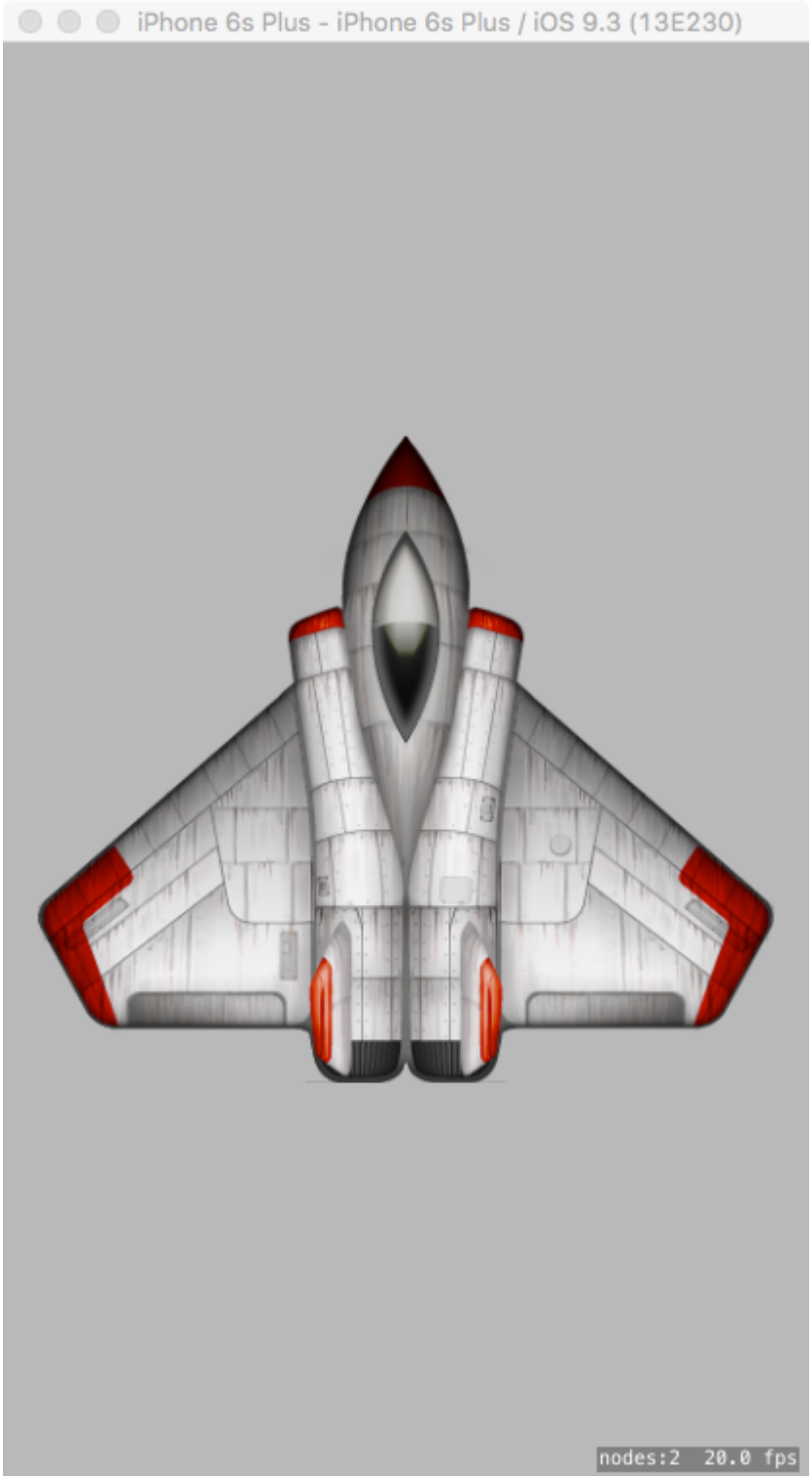
`Spaceship` est le nom de l'élément d'image dans les Assets.xcassets.

Une fois le sprite créé, vous pouvez l'ajouter à votre scène (ou à tout autre nœud).

Ouvrez `GameScene.swift` , supprimez tout son contenu et ajoutez ce qui suit

```
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)
    }
}
```

Maintenant, appuyez sur `CMD` + `R` dans Xcode pour lancer le simulateur.



Sous-classement SKSpriteNode

Vous pouvez sous- `SKSpriteNode` et définir votre propre type d'image-objet.

```
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)
    }
}
```

Lire SKSpriteNode (Sprites) en ligne: https://riptutorial.com/fr/sprite-kit/topic/3001/skspritenode--sprites-

Chapitre 10: SKView

Paramètres

Paramètre	Détails
montreFPS	Affiche le nombre d'images par seconde en cours dans la vue.
afficheNodeCount	Affiche le nombre de nœuds SKNodes actuellement affichés dans la vue.
montrePhysique	Affiche une représentation visuelle des SKPhysicsBodys dans la vue.
montreFields	Affiche une image représentant les effets des champs physiques dans la vue.
showDrawCount	Affiche le nombre de passes de dessin nécessaires au rendu de la vue.
afficheQuadCount	Affiche le nombre de rectangles requis pour afficher la vue.

Remarques

Un SKView est une sous-classe de UIView utilisée pour présenter des animations 2D SpriteKit.

Un SKView peut être ajouté à Interface Builder ou par programmation de la même manière que les UIViews «normales». Le contenu SpriteKit est ensuite présenté dans SKView dans un SKScene.

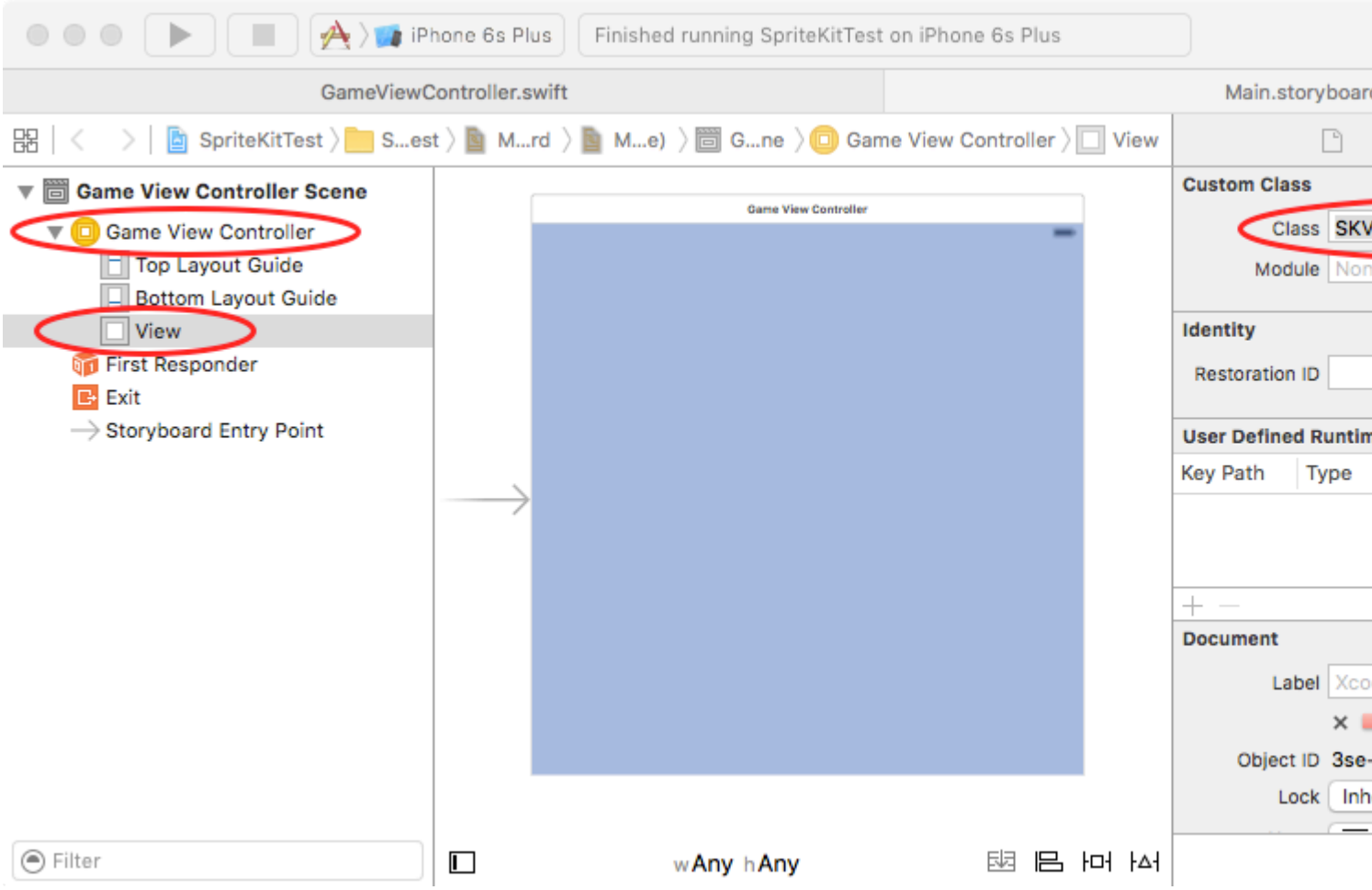
Voir aussi Référence de la classe SKView à partir de la documentation Apple.

Examples

Créer un SKView en plein écran à l'aide d'Interface Builder

Un cas d'utilisation typique de SpriteKit est l'endroit où SKView remplit tout l'écran.

Pour ce faire, dans Interface Builder de Xcode, créez d'abord un ViewController normal, puis sélectionnez la vue contenue et modifiez sa **classe** de **UIView** à **SKView** :



Dans le code du View Controller, dans la méthode viewDidLoad, saisissez un lien vers ce SKView en utilisant self.view:

En Swift:

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

(L'énoncé de garde protège contre l'erreur théorique que la vue n'est pas une vue SKView.)

Vous pouvez ensuite l'utiliser pour effectuer d'autres opérations telles que la présentation d'un SKScene:

En Swift:

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

Affichage des informations de débogage

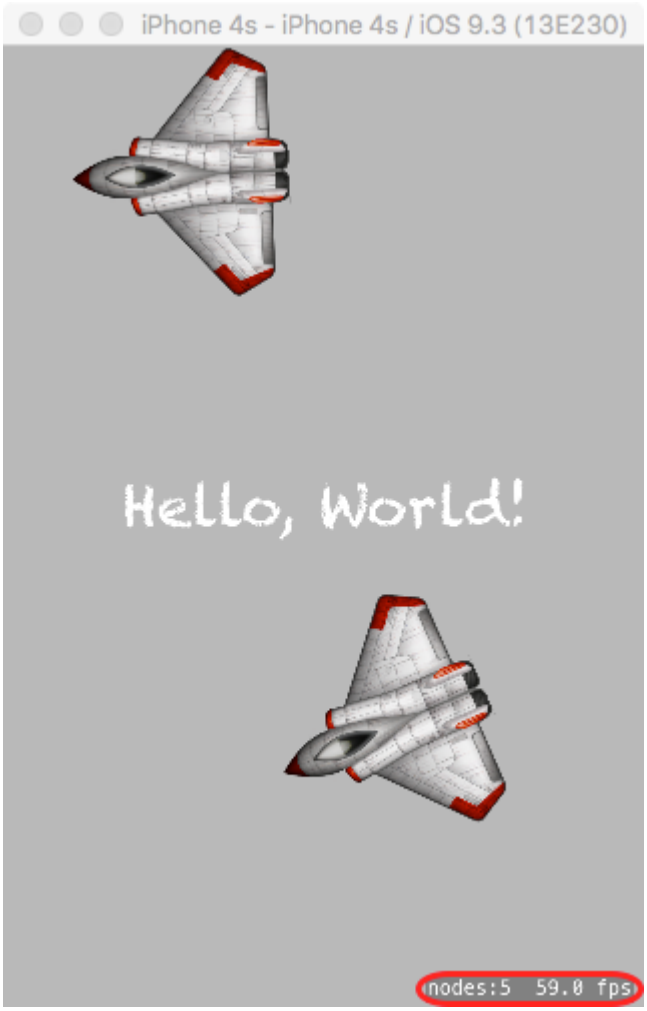
La fréquence d'images actuelle (en FPS, Frames Per Second) et le nombre total de SKNodes dans la scène (nodeCount, chaque sprite est un SKNode mais les autres objets de la scène sont également des SKNodes) peuvent être affichés dans le coin inférieur droit de la vue. .

Celles-ci peuvent être utiles lorsqu'elles sont activées (définies sur true) pour le débogage et l'optimisation de votre code, mais elles doivent être désactivées (définies sur false) avant de soumettre l'application à l'AppStore.

En Swift:

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

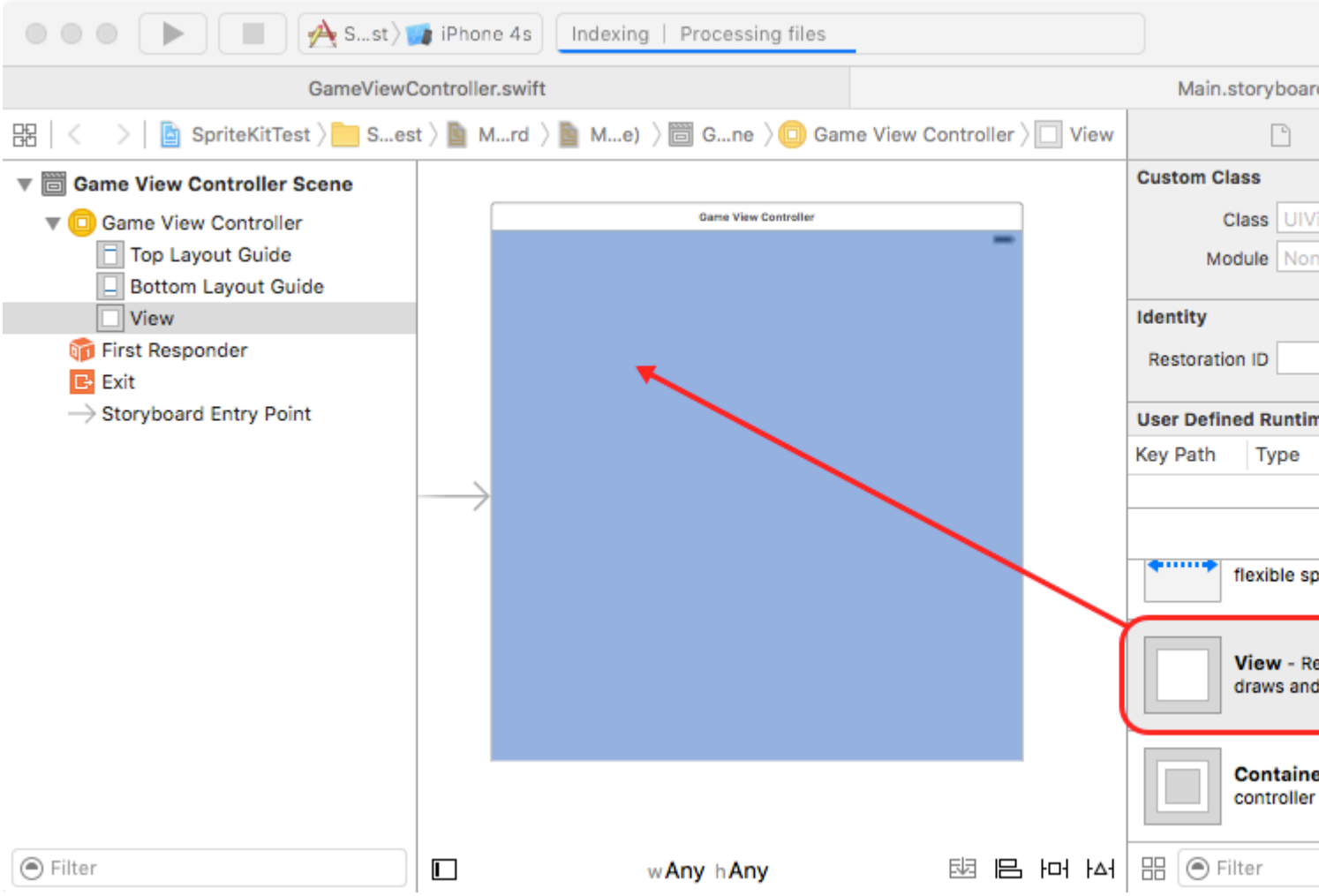
Résultat:



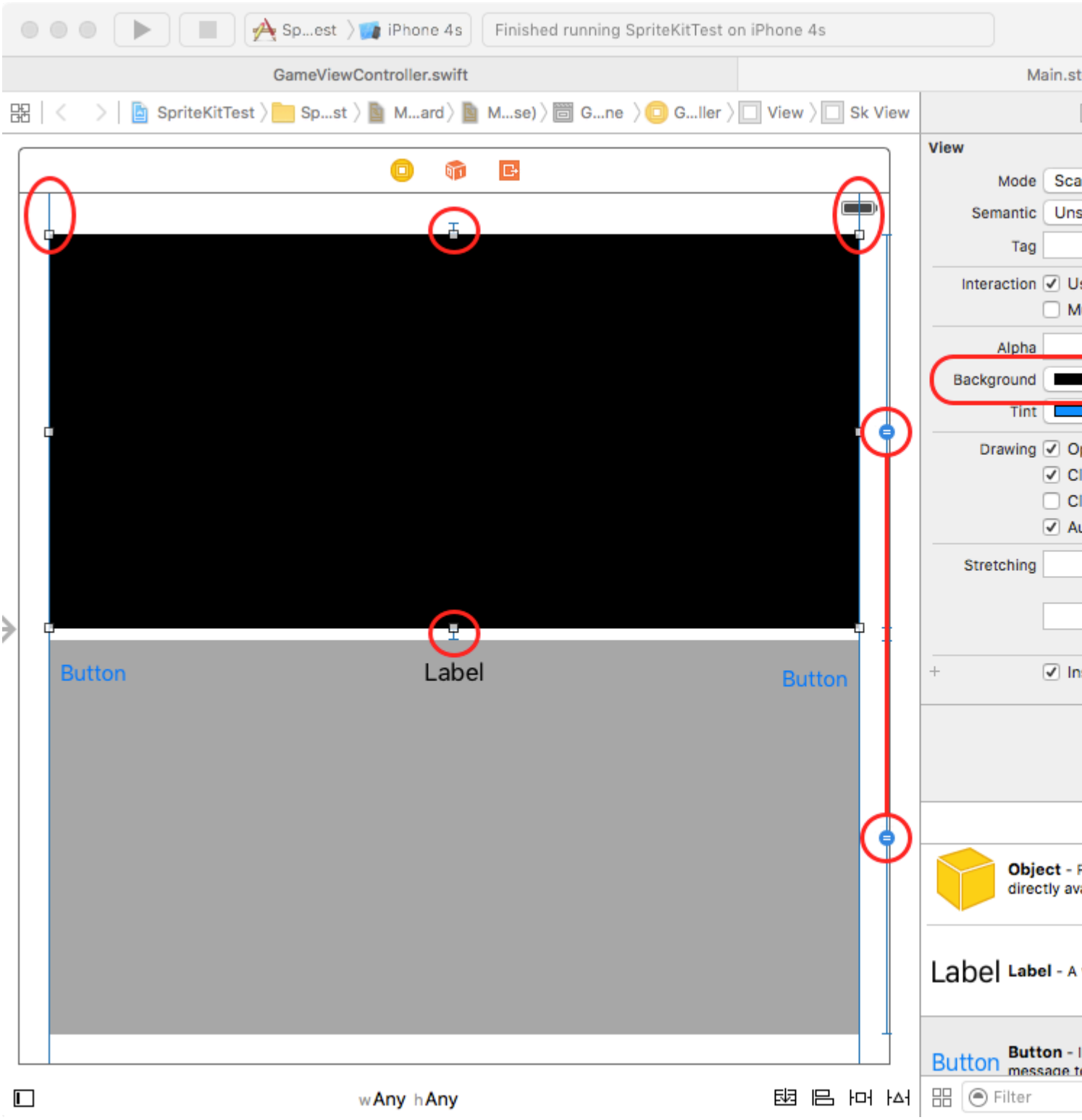
Créer un petit SKView avec d'autres contrôles à l'aide d'Interface Builder

Un SKView n'a pas besoin de remplir tout l'écran et peut partager de l'espace avec d'autres contrôles de l'interface utilisateur. Vous pouvez même avoir plus d'un SKView affiché en même temps si vous le souhaitez.

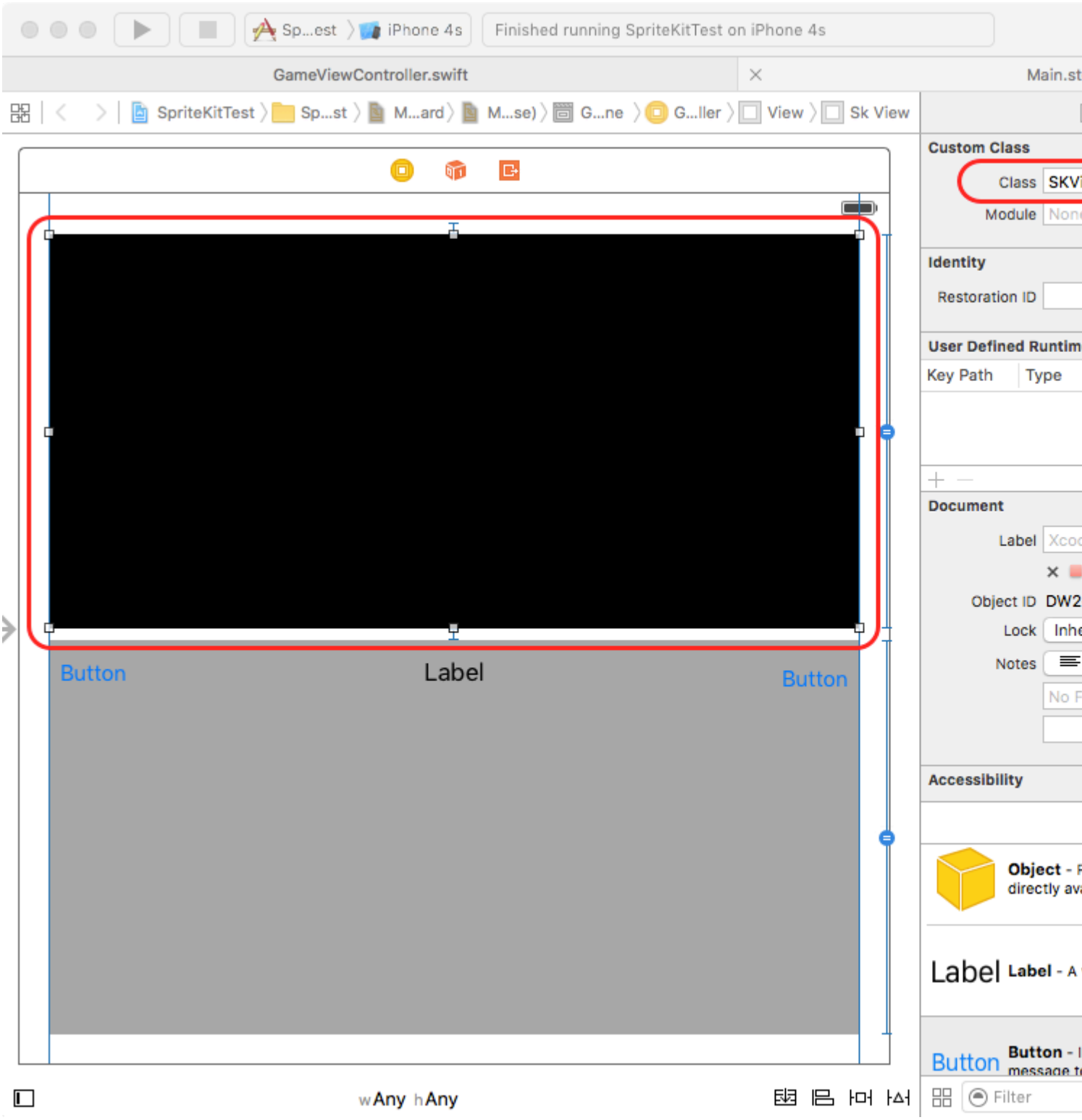
Pour créer un SKView plus petit, entre autres, avec Interface Builder, créez d'abord un ViewController normal, puis faites glisser une nouvelle vue sur le contrôleur de vue:



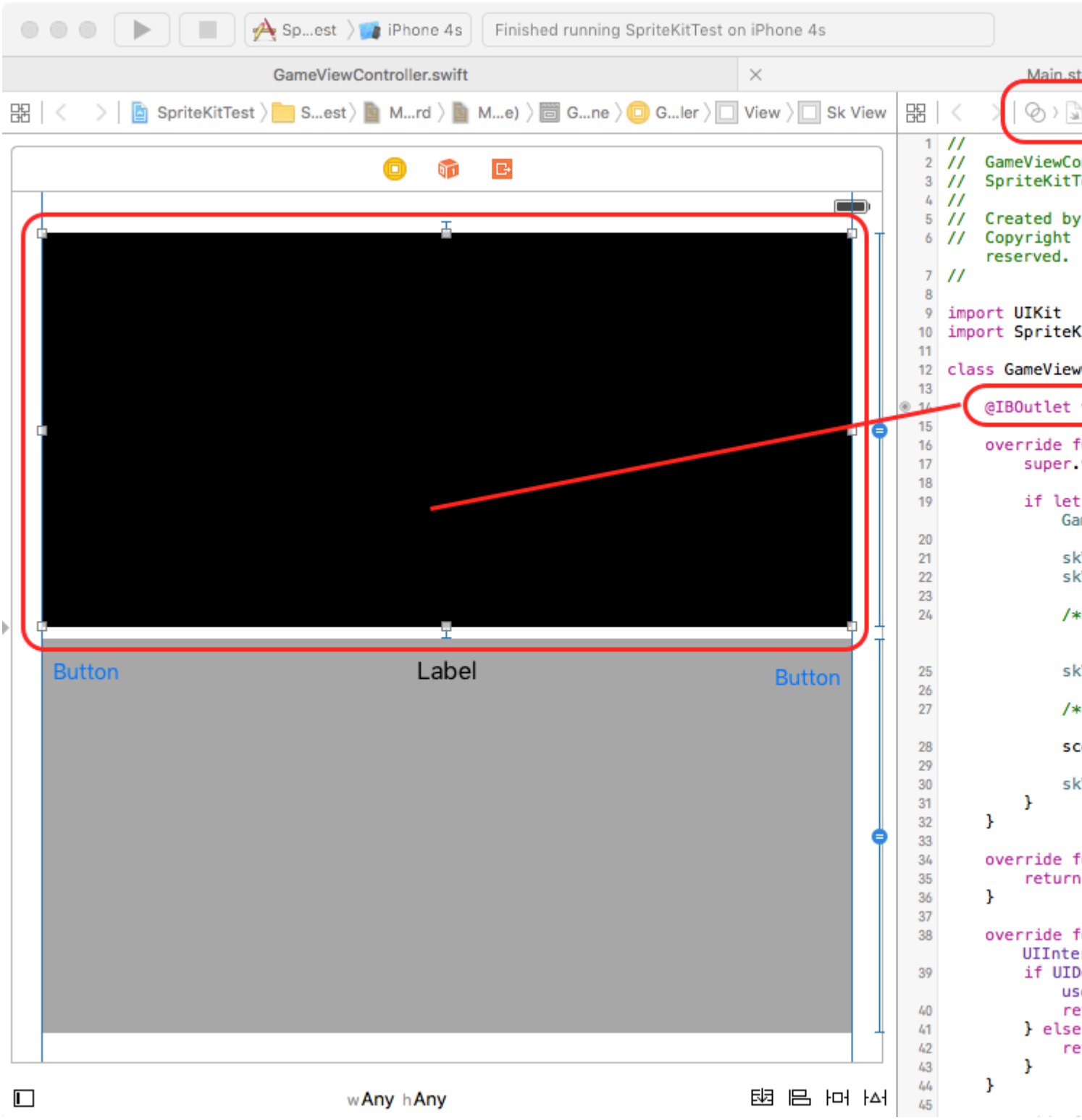
Il peut être utile de définir la couleur de cette vue sur autre chose que du blanc (ici, le noir est utilisé) afin qu'elle soit visible plus clairement dans Interface Builder (cette couleur ne sera pas affichée sur l'application finale). Ajoutez d'autres contrôles (un UIView, deux boutons et une étiquette sont présentés ici à titre d'exemples) et utilisez des contraintes comme d'habitude pour les afficher à l'écran:



Sélectionnez ensuite la vue que vous souhaitez utiliser comme SKView et changez sa classe en SKView:



Ensuite, à l'aide de l'éditeur assistant, faites glisser le curseur depuis ce SKView vers votre code et créez une sortie:

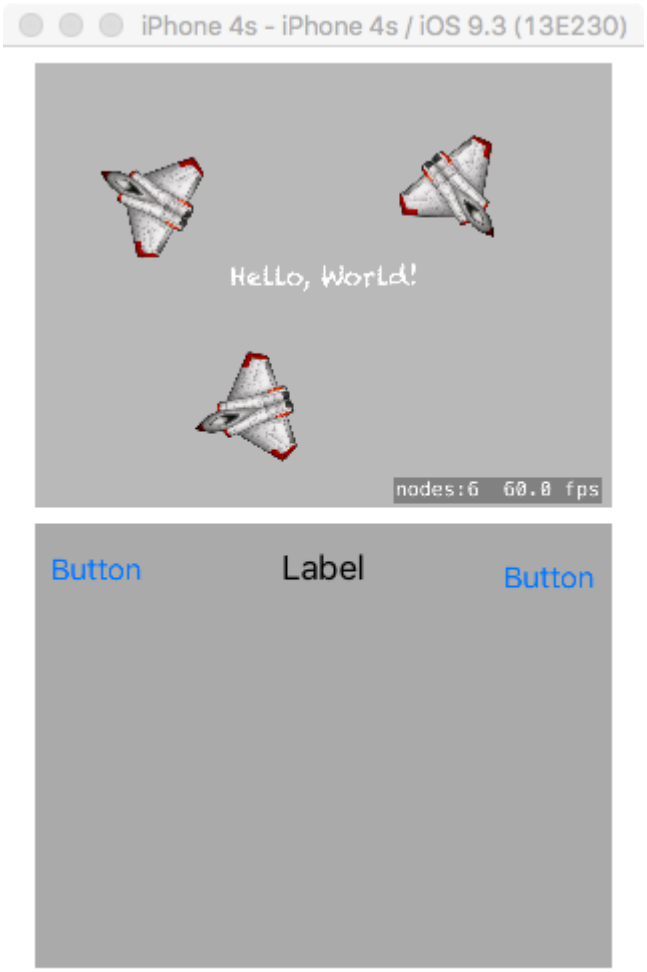


Utilisez cette prise pour présenter votre SKScene.

En Swift:

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

Résultat (basé sur l'exemple Hello World):



Lire SKView en ligne: https://riptutorial.com/fr/sprite-kit/topic/3572/skview

Crédits

S. No	Chapitres	Contributeurs
1	Premiers pas avec le kit sprite	Ali Beadle, Community, Luca Angeletti
2	Détection des entrées tactiles sur les appareils iOS	Knight0fDragon, Luca Angeletti
3	Éléments UIKit avec SpriteKit	Alessandro Ornano, Knight0fDragon
4	Fonctions temporelles dans SpriteKit: SKActions vs NSTimers	Knight0fDragon
5	La physique	Alessandro Ornano
6	SKAction	Abdou023, Kendel
7	SKNode Collision	Chen Wei, Confused, Knight0fDragon, RamenChef, Steve Ives
8	SKScene	Ali Beadle
9	SKSpriteNode (Sprites)	Ali Beadle, Knight0fDragon, Luca Angeletti
10	SKView	Ali Beadle, Kendel