Beginning 2D iOS & tvOS Game Development with Swift 3

Up to date for iOS 10, Xcode 8 & Swift 3

2D Apple Games by Tutorials

By the raywenderlich.com Tutorial Team

Caroline **Begbie**, Mike **Berg**, Michael **Briscoe**,
Ali **Hafizji**, Marin **Todorov**, and Ray **Wenderlich**

2D Apple Games by Tutorials

Caroline Begbie, Mike Berg, Michael Briscoe, Ali Hafizji, Marin Todorov and Ray Wenderlich

Notice of Rights

Notice of Liability

Trademarks

Dedications

"For my mum, who frequently challenges and always inspires me; also for Muffin, my patient and loyal puppy."

— Caroline Begbie

"To my wonderful wife and family, who make it possible to do what I do."

— Mike Berg

"To my father who ignited my love of computers. To my daughters Meghan and Brynne, and all six of my grandchildren. And to Lindsay for all the love and support."

— Michael Briscoe

"To my wife for always being supportive and to our bun in the oven, your parents await your arrival."

— Kauserali Hafizji (a.k.a. Ali)

"To my father."

— Marin Todorov

"To the authors, editors and leads at raywenderlich.com. Teamwork lets you dream bigger!"

— Ray Wenderlich

About the authors

Caroline Begbie is living the dream as an indie iOS developer, educator and explorer. She loves the graphics and animation side of iOS and watches Disney movies "for research."

Mike Berg is a full time game artist who is fortunate enough to work with many different indie game developers from all over the world. When he's not manipulating pixel colors, he loves to eat good food, spend time with his family, play games and be happy.

Michael Briscoe is an independent software developer with over 30 years of programming experience. Learning BASIC on a Commodore 64 way back in 1984, he's been hooked on coding ever since. He enjoys creating simulations and games for all Apple platforms. You can visit his website at skyrocketsoftware.wordpress.com.

Kauserali Hafizji (a.k.a. Ali) is a freelance software developer. He is fortunate to have worked on several large projects. He loves creating software that people use everyday whether it's on the web, watch, phone or tv. A good read, cool dip in the pool and a hot cheesy meal would be the perfect end to his weekend. You can find out more about Ali on his website at: alihafizji.com.

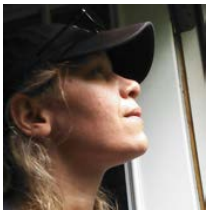
Marin Todorov is a part of Realm and raywenderlich.com. He's also the author of books and apps. You can find out more at www.underplot.com.

Ray Wenderlich is part of a great team — the raywenderlich.com team, a group of over 100 developers and editors from across the world. He and the rest of the team are passionate both about making apps and teaching others the techniques to make them. When Ray's not programming, he's probably playing video games, role playing games, or board games.

About the editors

Kyle Gorlick is the tech editor of this book. Kyle is a software developer currently focused on mobile apps and games. When not developing, he likes to watch basketball and play volleyball. You can learn more about him and what he's working on at kylegorlick.com.

Chris Belanger is the editor of this book. Chris is the Book Team Lead and Lead Editor for raywenderlich.com. If there are words to wrangle or a paragraph to ponder, he's on the case. When he kicks back, you can usually find Chris with guitar in hand, looking for the nearest beach, or exploring the lakes and rivers in his part of the world in a canoe.



Tammy Coron is the final pass editor of this book. Tammy is an independent creative professional and the host of two podcasts — Roundabout: Creative Chaos and Invisible Red. She's also the founder of Just Write Code, a small independent production company based in West Tennessee. Find out more at tammycoron.com.

About the artists

Mike Berg created the artwork for most of the games in this book. Mike is a full time game artist who is fortunate enough to work with many different indie game developers from all over the world. When he's not manipulating pixel colors, he loves to eat good food, spend time with his family, play games and be happy.

Vinnie Prabhu created all the music and sounds for the games in this book. Vinnie is a music composer/software engineer from Northern Virginia who has created music and sound work for concerts, plays and video games. He's also a staff member on OverClocked ReMix, an online community for music and video game fans. You can find Vinnie on Twitter as @palpablevt.

Vicki Wenderlich created many of the illustrations in this book and the artwork for Drop Charge. Vicki is Ray's wife and business partner. She is a digital artist who creates illustrations, game art and a lot of other art or design work for the tutorials and books on raywenderlich.com. She also runs gameartguppy.com, which is a website where she creates free and inexpensive art assets for game developers to use in their games. When she's not making art, she loves hiking, a good glass of wine and attempting to create the perfect cheese plate.

Table of Contents: Overview

Table of Contents: Extended

Introduction

In this book, you'll learn how to make 2D games for iOS, macOS, tvOS and even watchOS. You'll do this using Swift and Apple's built-in 2D game framework: SpriteKit. However, this raises a number of questions:

- **Why SpriteKit?** SpriteKit is Apple's built-in framework for making 2D games. It's easy to learn, especially if you already have some Swift or iOS experience.
- **Why iOS?** For a game developer, there's no better platform. The development tools are well-designed and easy to learn. Plus, the App Store makes it incredibly simple to distribute your game to a massive audience — and get paid for it!
- **Why macOS, tvOS and watchOS?** One of the great things about SpriteKit is that it works on iOS, macOS, tvOS and watchOS. If you get your game running on iOS, it's incredibly easy to get it working on the other platforms too.
- **Why Swift?** Swift is an easy language to learn, especially if you're new to programming.
- **Why 2D?** As impressive as 3D games may be, 2D games are a lot easier to make. The artwork is far less complicated, and programming is faster and doesn't require as much math. All of this allows you, as a developer, to focus on creating killer gameplay.

If you're a beginner, making 2D games is definitely the best way to get started.

If you're an advanced developer, making a 2D game is still much faster than making a 3D game. Since it's not necessarily the case that you earn more money with 3D games, why not go for the easier win? Plus, some people prefer 2D games anyway!

So rest easy — with 2D games and SpriteKit, you're making great choices!

History of this book

Three years ago, we wrote a book named *iOS Games by Tutorials*, covering how to make 2D games with SpriteKit. One year later, we released a second edition fully ported to Swift, as a free update for existing customers. One year after that, we completely revamped it, renamed it and released it as *2D iOS & tvOS Games by Tutorials*. This year, we're doing it again with *2D Apple Games by Tutorials*.

At WWDC 2016, Apple announced a lot of cool new features to both SpriteKit and Xcode, including built-in tile map support and the ability to run SpriteKit on watchOS. Yes, you read that correctly!

These changes were so significant that we decided it would be better to completely revamp the book (again!) — we even included new games!

If you already read *2D iOS & tvOS Games by Tutorials* and you're wondering what's new in this book, here are the highlights:

- **Zombie Conga**: Chapters 1-6 are mostly the same, with some minor updates to support Swift 3 and iOS 10. We also removed the chapter showing how to port the game to tvOS; that's because we added a brand new section, "Section V: Other Platforms", which covers this topic — and more — in greater detail.
- **Cat Nap**: These chapters also remain the same, with some minor updates to support Swift 3 and iOS 10. Just like we did with Zombie Conga, we removed the chapter showing how to port the game to tvOS.
- **Pest Control**: First introduced in *iOS Games by Tutorials*, this new and improved version shows you how to use the new tile maps features in SpriteKit (Chapters 12-13), as well as how to save and load game data (Chapter 14).
- **Drop Charge**: In these chapters, you'll review previous material in the book and learn about simple state machines, particle systems and juice, all updated for Swift 3 and iOS 10. Because Apple released so many new features for GameplayKit this year, we decided it would be too much to properly cover in this book. As such, the chapters dealing with GameplayKit have been removed, and will be covered elsewhere, at a later date.
- **Zombie Piranhas**: These new chapters introduce you to a new game specifically designed to teach you how to work with other Apple platforms like macOS, tvOS and watchOS (Chapters 18-20).
- **CircuitRacer**: These chapters were also updated for Swift 3 and iOS 10. However, since Apple has abandoned iAd, we removed the chapter which covered that topic.

As you can see, it's somewhat of a major overhaul. If you read the book before, but want to read it again and have limited time, the best thing to do is focus on the new games, or the chapters that interest you most.

About this book

This book is something special to us. Our goal at raywenderlich.com is for this to be the best book on 2D game programming you've ever read.

There are a lot of game programming books out there, and many of them are quite good, so this might be a lofty goal. But here's what we've done to try to accomplish it:

- **Learn by making games**: Other books teach the high-level concepts and show code snippets, but many leave you on your own to put together a complete, functioning game. In this book, you'll learn by making six games in a variety of genres — games that are actually fun. Our hope is that you can and will reuse techniques or code from these games to make your own games.
- **Learn by challenges**: Every chapter in this book includes some challenges at the end that are designed to help you practice what you've learned. Following a tutorial is one thing, but applying it yourself is quite another. The challenges in this book take off the training wheels and push you to solidify your knowledge by grappling with a problem on your own. Because we're not mean, we also provide the solutions to our challenges. But try not to look soon, or you might spoil the fun. =]
- **Focus on polish**: The key to making a hit game is polish — adding loads of well-considered details that set your game apart. Because of this, we've put our money where our mouths are and invested in a top-notch artist and sound designer to create resources for the games in this book. We've also included a chapter all about polishing your game with special effects — otherwise known as adding "Juice" — which we think you'll love.
- **High-quality tutorials**: Our site is known for its high-quality programming tutorials, and we've put a lot of time and care into the tutorials in this book to make them equally valuable, if not more so. Each chapter has been put through a rigorous multi-stage editing process — resulting in some chapters being rewritten several times! We've strived to ensure that each chapter contains great technical content while also being fun and easy to follow.

After you finish reading this book, please let us know if you think we were successful in meeting these goals. You can email Ray anytime at ray@raywenderlich.com.

We hope you enjoy the book, and we can't wait to see what games you make on your own!

iOS game development: a history

As you'll see, it's easy to make games with SpriteKit — but it wasn't always so. In the early days of iOS, your only option was to make your game with OpenGL ES, which (along with Metal) is the lowest-level graphics API available on the platform. OpenGL ES is notoriously difficult to learn, and it was a big barrier to entry for many beginning game developers.

After a while, third-party developers released some game frameworks on top of OpenGL, the most popular of which was called Cocos2D — in fact, several of us wrote a book on the subject! Many of the games at the top of the App Store charts were made with Cocos2D, and many developers can say that Cocos2D was their entry point into the world of game development.

Cocos2D was a great framework, but it wasn't written or supported by Apple. Because of this, there were often problems when new versions of iOS were released, or with integrating other Apple APIs into the system.

To resolve this, with iOS 7 Apple released a new framework for making 2D games: SpriteKit. Its API is very similar to Cocos2D, with similar types for the sprites, actions and scenes that Cocos2D developers know and love, so fans of the older framework will have no trouble getting up to speed. SpriteKit also has a few extra bells and whistles, like support for playing videos, making shapes and applying special image effects.

The SpriteKit API is well-designed and easy to use, especially for beginners. Best of all, you can use it knowing that it's fully supported by Apple and heavily optimized to make 2D games on iOS — and now with support for macOS, tvOS and watchOS, it makes it the clear choice for Apple 2D game development.

From here on out, if you want to make a 2D game on iOS, macOS, tvOS and watchOS, we definitely recommend using SpriteKit rather than other game frameworks. There's one big exception: if you want to make a cross platform game (i.e. for Android, Windows, etc). SpriteKit is an Apple-only API so it will be more challenging to port your game from SpriteKit to other platforms than using other options such as Unity. If you're interested in learning Unity, please check out our newly released book, *Unity Games by Tutorials*, which you can order here: https://www.raywenderlich.com/store/unity-games-by-tutorials.

If you just want to make something simple for Apple platforms only, SpriteKit is the way to go. So let's get you up to speed with SpriteKit!

What you need

To follow along with the tutorials in this book, you need the following:

- **A Mac running OS X El Capitan or later**. This is so you can install the latest version of the required development tool: Xcode.
- **Xcode 8.0 or later**. Xcode is the main development tool for Apple platforms. You need to use Xcode 8.0 or later in this book. You can download the latest version of Xcode for free from the Apple developer site: https://developer.apple.com/xcode/download/
- **An iPhone or iPad running iOS 10 or later, and a paid membership to the iOS development program [optional]**. For most of the chapters in the book, you can run your code on the iOS 10 Simulator that comes with Xcode. However, there are a few chapters later in the book that require a device for testing. Also note that SpriteKit performs better on physical devices than it does in the Simulator, so your frame rates will appear lower than expected when running your game in the Simulator.
- **An Apple TV [optional]**: You do not need an Apple TV since you can work with the Apple TV simulator, but it's definitely handy to test with a physical remote — plus awesome to see your games on the big screen!
- **An Apple Watch [optional]**: Just like you do not need an Apple TV, you also do not need an Apple Watch; using the simulator is perfectly acceptable.

If you don't have the latest version of Xcode installed, be sure to do that before continuing with the book.

Who this book is for

This book is for beginning to advanced iOS developers. Wherever you fall on that spectrum, you'll learn a lot from this book!

This book does require some basic knowledge of Swift. If you do not know Swift, you can still follow along with the book because all of the instructions are in step-by-step format. However, there will likely be parts that are confusing due to gaps in your knowledge. Before beginning this book, you might want to go through our Swift Apprentice series, which covers the basics of Swift development:

- www.raywenderlich.com/store

How to use this book

There are two ways to use this book, depending on whether you are a complete beginner to Apple game development or an advanced developer with knowledge of other 2D game frameworks.

If you are a complete beginner

If you're a complete beginner to Apple game development, the best way to read this book is from cover to cover. We have arranged the chapters to introduce the material in the most logical manner to build up your skills one layer at a time.

If you are an advanced developer

If you're an advanced developer with knowledge of other 2D game frameworks, you'll have an easier time adapting to SpriteKit, as the core concepts and syntax will look very familiar.

Our suggestion is to skim through the early chapters and focus more on the later, more advanced chapters, or where you have a particular interest.

Don't worry — you can jump right into any chapter in the book, because we'll always have a starter project waiting for you!

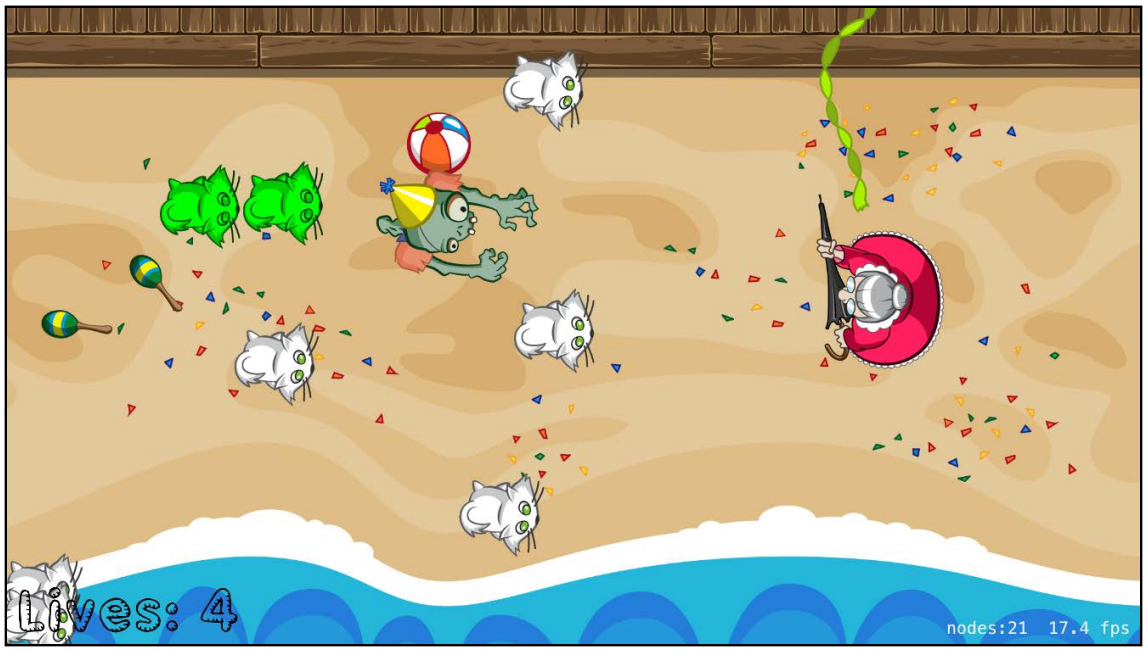
What's ahead: an overview

2D Apple Games by Tutorials is split into six sections, moving from beginning to advanced topics. In each section, you'll create a complete mini-game, from scratch! The book also includes a bonus chapter at the end that we think you'll enjoy.

Take a look at what's ahead!

Section I: Getting started

This section covers the basics of making 2D games with SpriteKit. These are the most important techniques, the ones you'll use in almost every game you make. By the time you reach the end of this section, you'll be ready to make your own simple game.



Throughout this section, you'll create an action game named Zombie Conga, where you take the role of a happy-go-lucky zombie who just wants to party!

There are six chapters in this section; they are:

1. **Chapter 1, Sprites**: Get started by adding your first sprites to the game: the background and the zombie.
2. **Chapter 2, Manual Movement**: You'll make the zombie follow your touches around the screen and get a crash-course in basic 2D vector math.
3. **Chapter 3, Actions**: You'll add cats and crazy cat ladies to the game, as well as basic collision detection and gameplay.
4. **Chapter 4, Scenes**: You'll add a main menu to the game, as well as win and lose scenes.
5. **Chapter 5, Camera**: You'll make the game scroll from left to right, and finally, add the conga line itself.
6. **Chapter 6, Labels**: You'll add a label to show the zombie's lives and the number of cats in his conga line.

Section II: Physics and nodes

In this section, you'll learn how to use the built-in 2D physics engine included with SpriteKit. You'll also learn how to use special types of nodes that allow you to play videos and create shapes in your game.

In the process, you'll create a physics puzzle game named Cat Nap, where you take the role of a cat who has had a long day and just wants to go to bed.

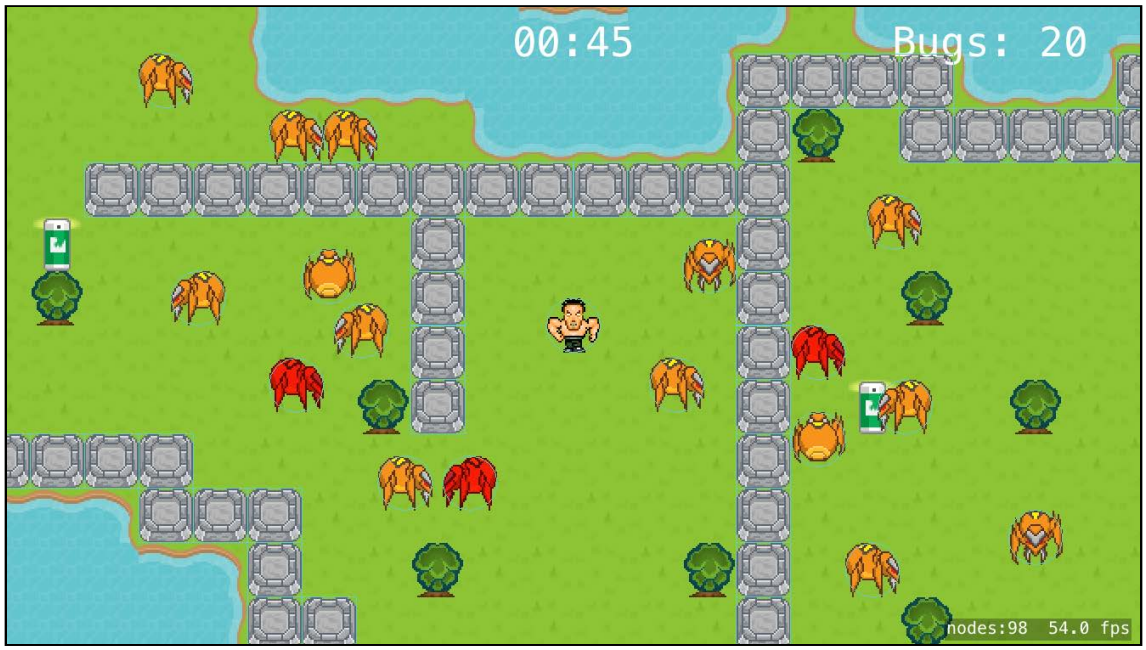
There are five chapters in this section; they are:

7. **Chapter 7, Scene Editor**: You'll begin by creating the first level of the game. By the end, you'll have a better understanding of Xcode's level designer, better known as the scene editor.
8. **Chapter 8, Beginning Physics**: In this chapter, you're going to take a little detour in order to learn the basics of creating physics simulations for your games. As a bonus, you'll learn how to prototype games inside an Xcode playground.
9. **Chapter 9, Intermediate Physics**: You'll learn about physics-based collision detection and create custom classes for your SpriteKit nodes.
10. **Chapter 10, Advanced Physics**: You'll add two more levels to the game as you learn about interactive bodies, joints between bodies, composed bodies and more.
11. **Chapter 11, Crop, Video and Shape Nodes**: You'll add special new blocks to

Cat Nap while learning about additional types of nodes that allow you to do amazing things, like play videos, crop images and create dynamic shapes.

Section III: Tile Maps

In this section, you'll learn about tile maps in SpriteKit and how to save and load game data.



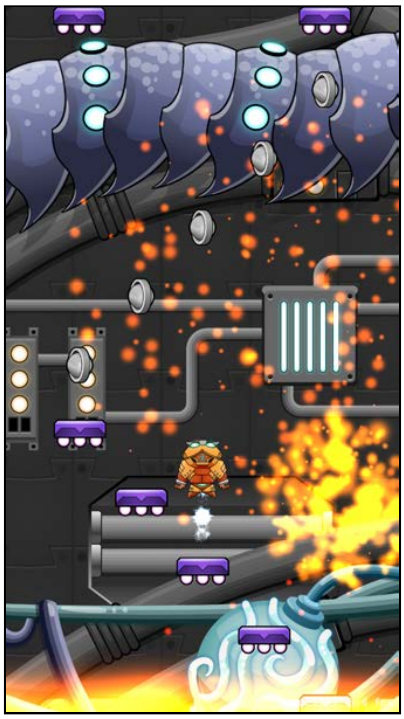
In the process, you'll create a game named Pest Control, where you take control of a vigorous, impossibly ripped he-man named Arnie. Your job is to lead Arnie to bug-fighting victory by squishing all those pesky bugs.

There are three chapters in this section; they are:

12. **Chapter 12, Beginning Tile Maps**: You'll learn the basics about tile maps in SpriteKit by creating a tile set and a background tile map.
13. **Chapter 13, Intermediate Tile Maps**: You'll take things a step further by learning how to access tile maps in code and how to create a tile map with randomly placed tiles.
14. **Chapter 14, Saving and Loading Games**: You'll finish up by adding a winning end state and a heads-up display. You'll also learn how to automatically save the game when you leave it and then reload it from where you left off.

Section IV: Juice

In this section, you'll learn how to take a good game and make it great by adding a ton of special effects and excitement — also known as "juice."



In the process, you'll create a game named Drop Charge, where you're a space hero with a mission to blow up an alien space ship — and escape with your life before it explodes. To do this, you must jump from platform to platform, collecting special boosts along the way. Just be careful not to fall into the red hot lava!

There are three chapters in this section; they are:

15. **Chapter 15, Making Drop Charge**: You'll put together the basic gameplay using the scene editor and code, flexing the SpriteKit muscles you've developed working through previous chapters.
16. **Chapter 16, Particle Systems**: You'll learn how to use particle systems to create amazing special effects.
17. **Chapter 17, Juice Up Your Game**: You'll trick out your game with music, sound, animation, more particles and other special effects, experiencing for yourself the benefits of mastering the details.

Section V: Other Platforms

In this section, you'll learn how to leverage your iOS knowledge to build games for the other Apple Platforms: macOS, tvOS and watchOS.

In the process, you'll create a game named Zombie Piranhas. In this game, your goal is to catch as many fish as possible without hooking a zombie — because we all know what happens when zombies are around.

There are three chapters in this section; they are:

18. **Chapter 18, macOS Games**: You'll take a complete iOS game and add a target for macOS. Along the way, you'll learn some of the differences between the platforms, such as windows and mouse and keyboard events.
19. **Chapter 19, tvOS Games**: Building from Chapter 18, you'll add another target for tvOS. You'll learn concepts such as Focus and parallax icons, Top Shelf and working with the Apple TV Remote.
20. **Chapter 20, watchOS Games**: Lastly, you'll add a target for watchOS, and you'll learn about gestures, the Digital Crown and Haptic Feedback. You'll also discover some of the design considerations when working with a small device.

Section VI: Advanced Topics

In this section, you'll learn some APIs other than SpriteKit that are good to know when making games for the Apple platforms. In particular, you'll learn how to add Game Center leaderboards and achievements into your game. You'll also learn how to use the ReplayKit API.

In the process, you'll integrate these APIs into a top-down racing game named Circuit Racer, where you take the role of an elite race car driver out to set a world record — which wouldn't be a problem if all this debris wasn't on the track!

21. **Chapter 21, Game Center Achievements**: Enable Game Center for your game and award the user achievements for accomplishing certain feats.
22. **Chapter 22, Game Center Leaderboards**: Set up various leaderboards for your game and track and report the player's scores.
23. **Chapter 27, ReplayKit**: You'll learn how to allow players to record and share videos of their games with ReplayKit.

Section VI: Bonus chapter

And that's not all — on top of the above, we included a bonus chapter about making your own game art:

29. **Chapter 29, Making Art for Programmers**: If you liked the art in these mini-games and want to learn how to either hire an artist or make some art of your own, look no further than this chapter! This chapter guides you through drawing a cute cat in the style of this book with Illustrator.

Book source code and forums

This book comes with complete source code for each of the chapters — it's shipped with the PDF. Some of the chapters have starter projects or other required resources that are also included, and you'll definitely want to have these on hand as you go through the book.

We've set up an official forum for the book at raywenderlich.com/forums. This is a great place to ask any questions you have about the book or about making games with SpriteKit, or to submit any errors you may find.

Book updates

Since you purchased the PDF version of this book, you get free access to any updates we may make to the book!

The best way to get update notifications is to sign up for our monthly newsletter. This includes a list of the tutorials that came out on raywenderlich.com that month, any important news like book updates or new books, and a list of our favorite development links for that month. You can sign up here:

- www.raywenderlich.com/newsletter

License

By purchasing 2D Apple Games by Tutorials, you acquire the following license:

- You are allowed to use and/or modify the source code provided with 2D Apple Games by Tutorials in as many games as you want, with no attribution required.
- You are allowed to use and/or modify all art, music and sound effects that are included with 2D Apple Games by Tutorials in as many games as you want, but must include this attribution line somewhere inside your game: "Artwork/sounds: from 2D Apple Games by Tutorials book, available at http://www.raywenderlich.com."
- The source code included in 2D Apple Games by Tutorials is for your personal use only. You are NOT allowed to distribute or sell the source code in 2D Apple Games by Tutorials without prior authorization.
- This book is for your personal use only. You are NOT allowed to sell this book without prior authorization, or distribute it to friends, co-workers or students — they would need to purchase their own copy.

All materials provided with this book are provided on an "as-is" basis, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software.

All trademarks and registered trademarks appearing in this guide are the property of their respective owners.

Acknowledgements

We would like to thank many people for their assistance in making this book possible:

- **Our families**: For bearing with us during this hectic time as we worked all hours of the night to get this book ready for publication!
- **Everyone at Apple**: For developing an amazing 2D game framework and other helpful APIs for games, for constantly inspiring us to improve our apps and skills, and for making it possible for many developers to have their dream jobs! Special thanks for the Apple TV dev kits as well. :]
- **Ricardo Quesada**: Ricardo is the lead developer of Cocos2D, which got many of us into making games. SpriteKit seems to draw quite a bit of inspiration from Cocos2D, so Ricardo deserves "mad props" for that as well.
- And most importantly, **the readers of raywenderlich.com and you**! Thank you so much for reading our site and purchasing this book. Your continued readership and support is what makes this all possible!

Section I: Getting Started

This section covers the basics of making 2D games with SpriteKit. These are the most important techniques, the ones you'll use in almost every game you make. By the time you reach the end of this section, you'll be ready to make your own simple game.

Throughout this section, you'll create an action game named Zombie Conga, where you take the role of a happy-go-lucky zombie who just wants to party!

1 Chapter 1: Sprites

By Ray Wenderlich

Now that you know what SpriteKit is and why you should use it, it's time to try it out for yourself!

The first minigame you'll build in this book is named Zombie Conga. Here's what it will look like when you're finished:

In Zombie Conga, you take the role of a happy-go-lucky zombie who wants to party!

Luckily, the beach town you occupy has an overly abundant cat population. You simply need to bite them and they'll join your zombie conga line.

But watch out for crazy cat ladies! These wizened warriors in red dresses won't take kindly to anyone stealing their beloved cats and will do their best to make the zombie rest in peace — permanently.

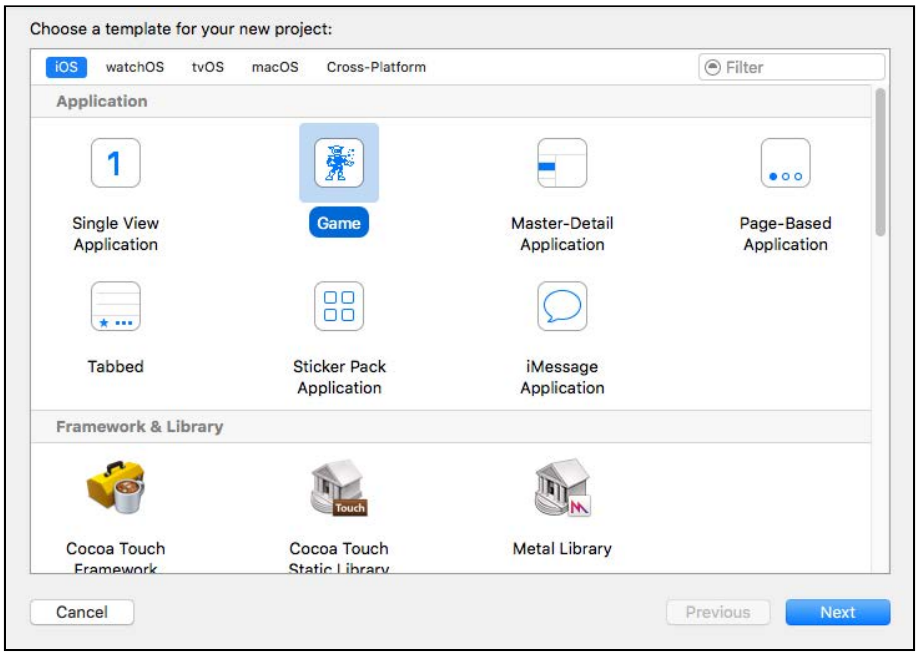
You will build this game across the next six chapters, in stages:

1. **Chapter 1, Sprites**: You are here! Get started by adding your first sprites to the game: the background and the zombie.
2. **Chapter 2, Manual Movement**: You'll make the zombie follow your touches around the screen and get a crash course in basic 2D vector math.
3. **Chapter 3, Actions**: You'll add cats and crazy cat ladies to the game, as well as basic collision detection and gameplay.
4. **Chapter 4, Scenes**: You'll add a main menu to the game, as well as win and lose scenes.
5. **Chapter 5, Camera**: You'll make the game scroll from left to right, and finally, add the conga line itself.
6. **Chapter 6, Labels**: You'll add labels to show the zombie's number of lives and the number of cats in his conga line.

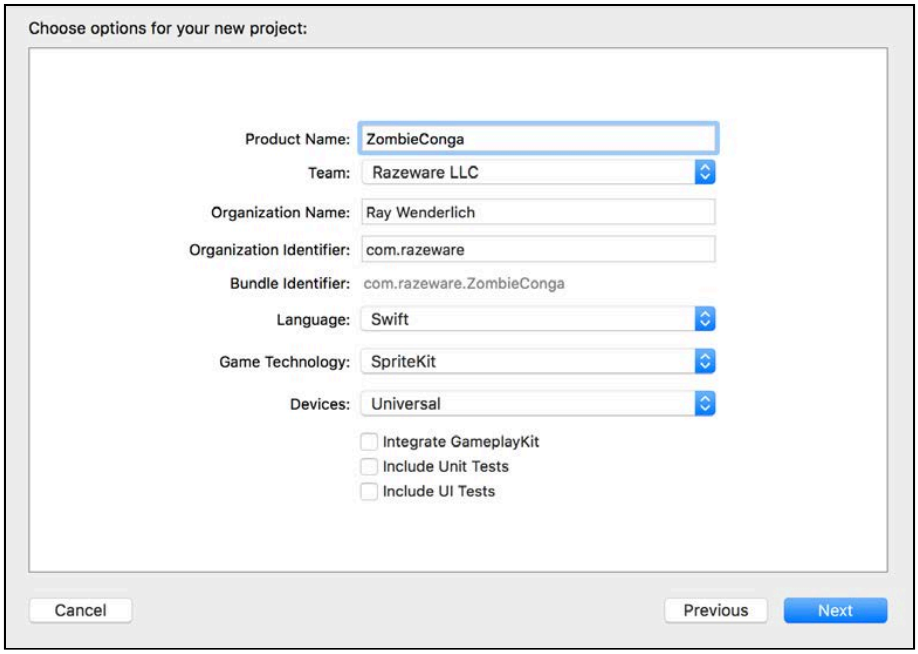
Time to get this conga started!

Getting started

Start Xcode and select **File > New > Project...** from the main menu. Select the **iOS/Application/Game** template and click **Next**.

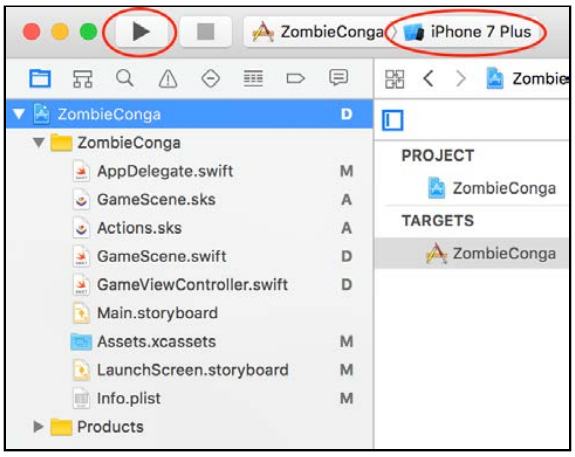


Enter **ZombieConga** for the Product Name, choose **Swift** for Language, **SpriteKit** for Game Technology, **Universal** for Devices and click **Next**.

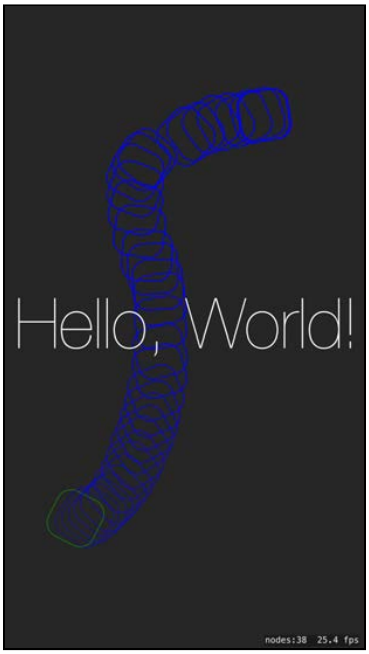


Select somewhere on your hard drive to save your project and click **Create**. At this point, Xcode will generate a simple SpriteKit starter project for you.

Take a look at what SpriteKit made. In Xcode's toolbar, select the iPhone 7 Plus and click **Play**.



After a brief splash screen, you'll see a single label that says, "Hello, World!". When you drag your mouse across the screen, the text will bounce and some spinning rounded rectangles will appear.



In SpriteKit, a single object called a **scene** controls each "screen" of your app. Scenes are represented by SpriteKit's `SKScene` class.

Right now, this app just has a single scene: `GameScene`. Open **GameScene.swift** and you'll see the code that displays the label and the spinning rounded rectangles.

It's not important to understand this code quite yet — you're going to remove it all and build your game one step at a time.

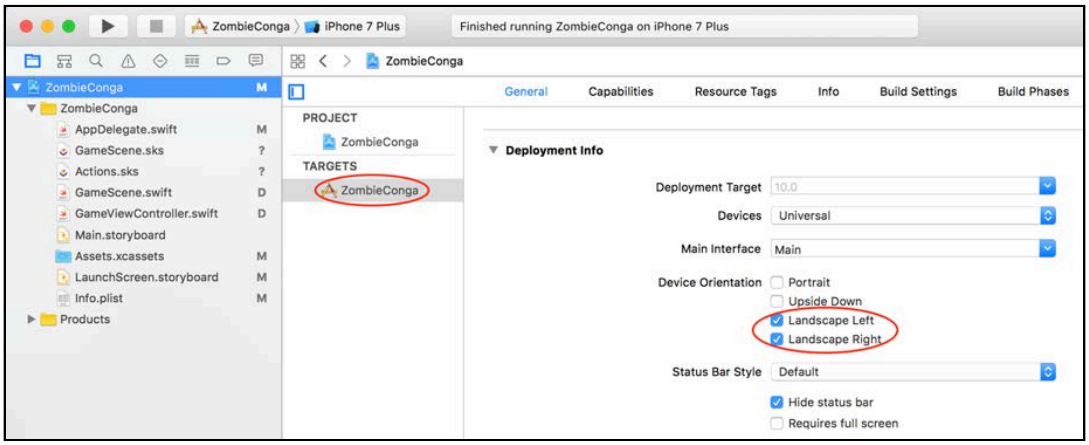
For now, delete everything in **GameScene.swift** and replace it with the following:

```
import SpriteKit

class GameScene: SKScene {
  override func didMove(to view: SKView) {
    backgroundColor = SKColor.black
  }
}
```

Note that `GameScene` is a subclass of `SKScene`. SpriteKit calls the method `didMove(to:)` before it presents your scene in a view; it's a good place to do some initial setup of your scene's contents. Here, you simply set the background color to black.

Zombie Conga is designed to run in landscape mode, so you need configure the app for this. Select the **ZombieConga** project in the project navigator and then select the **ZombieConga** target. Go to the **General** tab and make sure only **Landscape Left** and **Landscape Right** are checked:



The SpriteKit project template automatically creates a file named **GameScene.sks**. You can edit this file with Xcode's built-in scene editor to lay out your game scene visually. Think of the scene editor as a simple Interface Builder for SpriteKit.

You'll learn all about the scene editor in Chapter 7, "Scene Editor", but you won't be using it for Zombie Conga as it will be easier and more instructive to create the sprites programmatically instead.

Control-click **GameScene.sks**, select **Delete** and then select **Move to Trash**. While you're at it, also delete **Actions.sks**, another file you won't need for this game.

Since you're no longer using these files, you'll have to modify the template code appropriately.

Open **GameViewController.swift** and replace the contents with the following:

```
import UIKit
import SpriteKit

class GameViewController: UIViewController {
  override func viewDidLoad() {
    super.viewDidLoad()
    let scene =
      GameScene(size:CGSize(width: 2048, height: 1536))
    let skView = self.view as! SKView
    skView.showsFPS = true
    skView.showsNodeCount = true
    skView.ignoresSiblingOrder = true
    scene.scaleMode = .aspectFill
    skView.presentScene(scene)
  }

  override var prefersStatusBarHidden: Bool {
    return true
  }
}
```

Previously, the view controller loaded the scene from **GameScene.sks**, but now it creates the scene by calling an initializer on `GameScene` instead.

Notice that when you create the scene, you pass in a hard-coded size of **2048x1536** and set the scale mode to `aspectFill`.

This is a good time for a quick discussion about how this game is designed to work as a universal app.

Universal app support

> **Note**: This section is optional and for those who are especially curious. If you're eager to get coding as soon as possible, feel free to skip to the next section, "Adding the art".

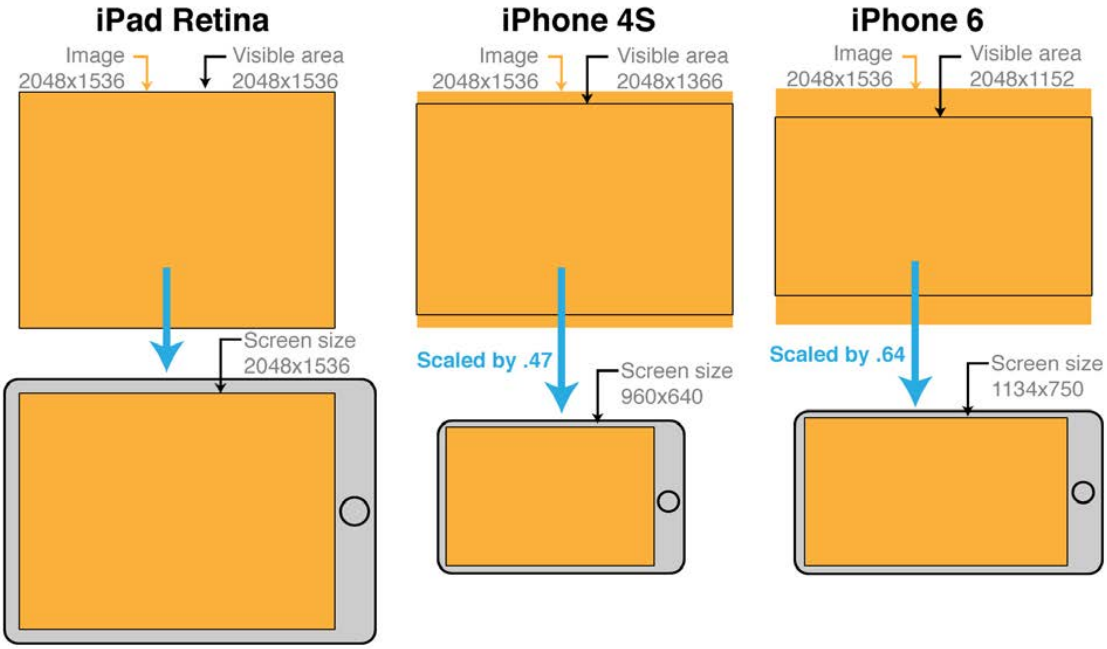
We've designed all the games in this book as universal apps, which means they will work on the iPhone and the iPad.

The scenes for the games in this book have been designed with a base size of 2048x1536, or reversed for portrait orientation, with the scale mode set to aspect fill.

Aspect fill instructs SpriteKit to scale the scene's content to fill the entire screen, even if SpriteKit needs to cut off some of the content to do so.

This results in your scene appearing as-is on the iPad Retina (excluding the 12.9-inch iPad Pro), which has a resolution of 2048x1536, but as scaled/cropped on the iPhone to fit the phone's smaller size and different aspect ratio.

Here are a few examples of how the games in this book will look in landscape orientation on different devices, moving from smallest to largest aspect ratio:



- **iPad Retina [4:3 or 1.33]**: Displayed as-is to fit the 2048x1536 screen size.
- **12.9-inch iPad Pro [4:3 or 1.33]**: Aspect fill will scale the 2048x1536 visible area by 1.33 to fit the 2732x2048 screen.
- **iPad non-Retina [4:3 or 1.33]**: Aspect fill will scale a 2048x1536 visible area by 0.5 to fit the 1024x768 screen.
- **iPhone 4s [3:2 or 1.5]**: Aspect fill will scale a 2048x1366 visible area by 0.47 to fit the 960x640 screen.
- **iPhone 5 [16:9 or 1.77]**: Aspect fill will scale a 2048x1152 visible area by 0.56 to fit the 1136x640 screen.
- **iPhone 6/7 [16:9 or 1.77]**: Aspect fill will scale a 2048x1152 visible area by 0.64 to fit the 1334x750 screen.
- **iPhone 6/7 Plus [16:9 or 1.77]**: Aspect fill will scale a 2048x1152 visible area by 0.93 to fit the 1920x1080 screen.

Since aspect fill will crop the scene on the top and bottom for iPhones, we've designed the games in this book to have a main "playable area" that is guaranteed

to be visible on all devices. Basically, the games will have a 192-point margin on the top/bottom in landscape and the left/right in portrait, in which you should avoid putting essential content. We'll show you how to visualize this later in the book.

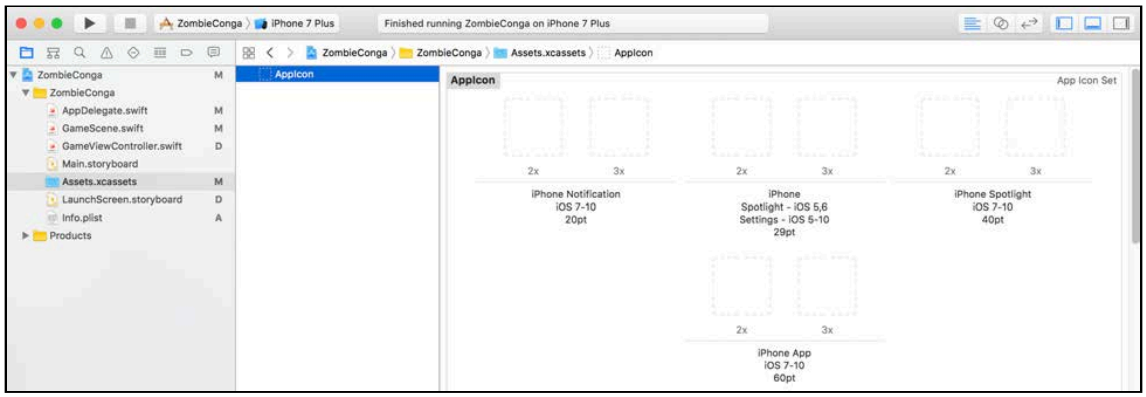
Note that you'll only use one set of art in this book: the art to fit the base size of 2048x1536. The art will be displayed as-is on iPad Retina, upscaled on the 12.9-inch iPad Pro and downscaled on other devices.

> **Note**: The downside of this approach is that the art will be bigger than necessary for some devices, such as the iPhone 4s, thereby wasting texture memory and space. The pro of this approach is that the game stays nice and simple and works well on all devices.
>
> An alternate approach would be to add different images as needed for each device and scale factor (i.e. iPad@1x, iPad@2x, iPhone@2x, iPhone@3x), leveraging the power of Xcode's asset catalogs. However, that would require generating a large set of images, so we will keep things simple for now.

Adding the art

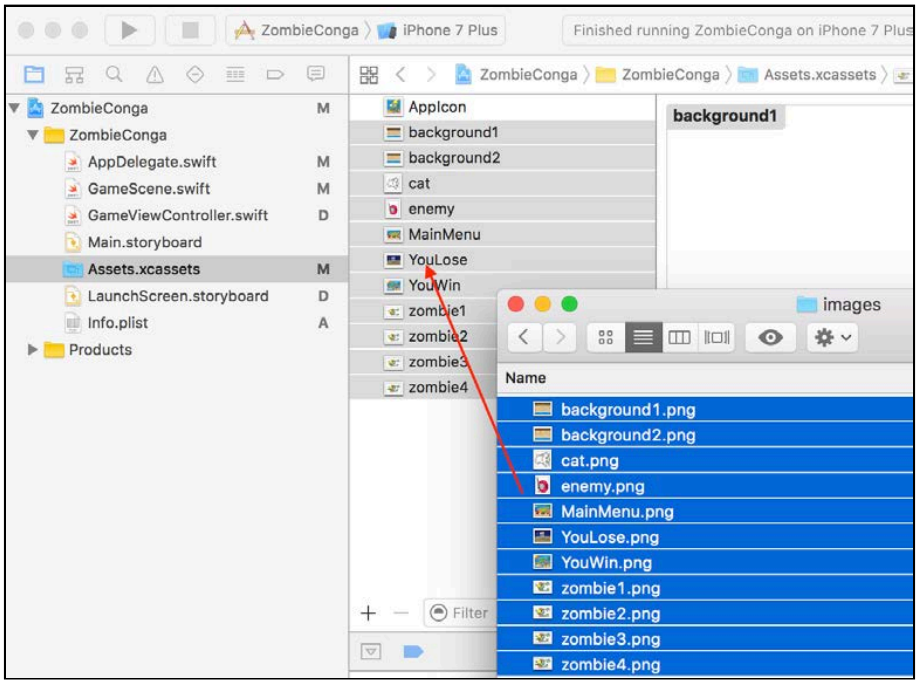
Next, you need to add the game art to the project.

In Xcode, open **Assets.xcassets**, select the **Spaceship** entry and press the Delete key to remove it — unfortunately, this is not a game about space zombies! :] At this point, only **AppIcon** will remain:



With **AppIcon** selected, drag the appropriate icon from **starter/resources/icons** into each slot:

Then, drag all the files from **starter/resources/images** into the left sidebar:



By including your images in the asset catalog, Xcode will automatically build **texture atlases** containing these images and use them in your game, which will automatically increase performance.

Launch screen

> **Note**: This is another optional section, as it won't have any impact on gameplay; it simply adds a "nice-to-have" feature that you'd typically want in a game. If you'd rather get straight to coding, feel free to skip to the next section, "Displaying a sprite".

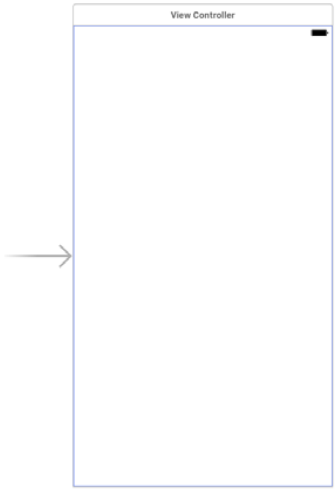
There's one last thing you should do to get this game started on the right foot: configure the launch screen.

The launch screen is what iOS displays when your app is first loading, which usually takes a few seconds. Your app actually has a launch screen already. When you launched your app earlier, you may have noticed the brief, blank white screen. That was it!

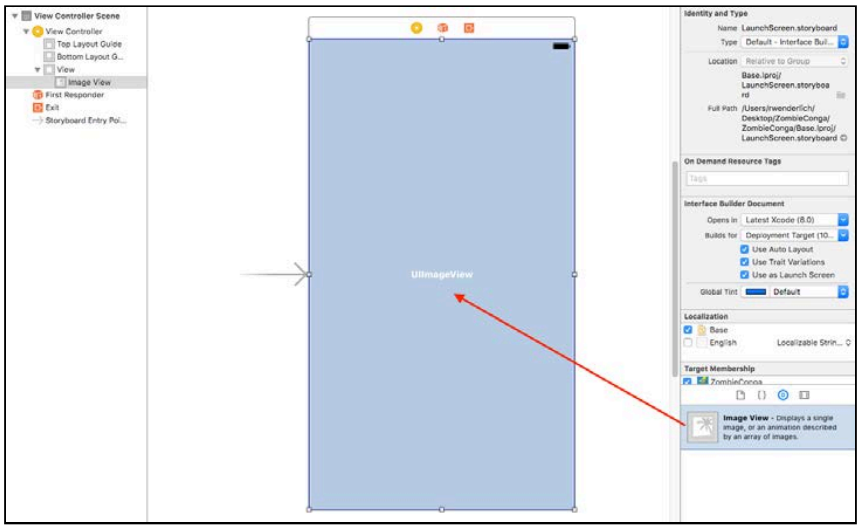
A launch screen should give the player the impression that your app is starting quickly — the default white screen, needless to say, does not. For Zombie Conga, you'll show a splash screen with the name of the game.

In iOS, apps have a special **launch screen** file; this is basically a storyboard, **LaunchScreen.storyboard** in this project, that you can configure to present something onscreen while your app is loading. The advantage of this over the old method of just displaying an image is that you can use Auto Layout for finer control of the appearance of the screen on different devices.

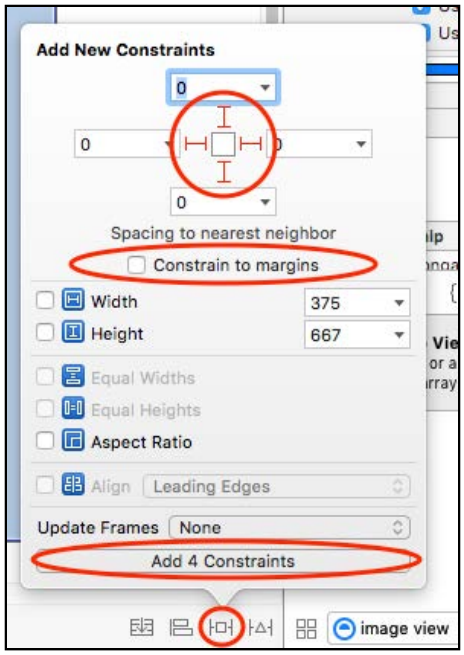
Time to try this out. Open **LaunchScreen.storyboard**. You'll see the following:



From the Object Library in the Utilites Area, drag an **Image View** into the view and resize it to fill the entire area:



Next, you need to set the image view so that it always has the same width and height as its containing view. To do this, make sure the image view is selected and then click the **Pin** button in the lower right — it looks like a TIE fighter from Star Wars. In the Add New Constraints screen, click the four light-red lines so that the image view is pinned to each edge. Make sure that **Constrain to margins** isn't checked and that all values are set to 0, then click **Add 4 Constraints**:



With the image view still selected, make sure the Attributes Inspector is showing — it's the fourth tab on the right. Set the Image to **MainMenu** and set the Content Mode to **Aspect Fill**:

Build and run your app again, and if necessary, rotate your simulator to the right with **Hardware > Rotate Right**. This time, you'll see a brief Zombie Conga splash screen:

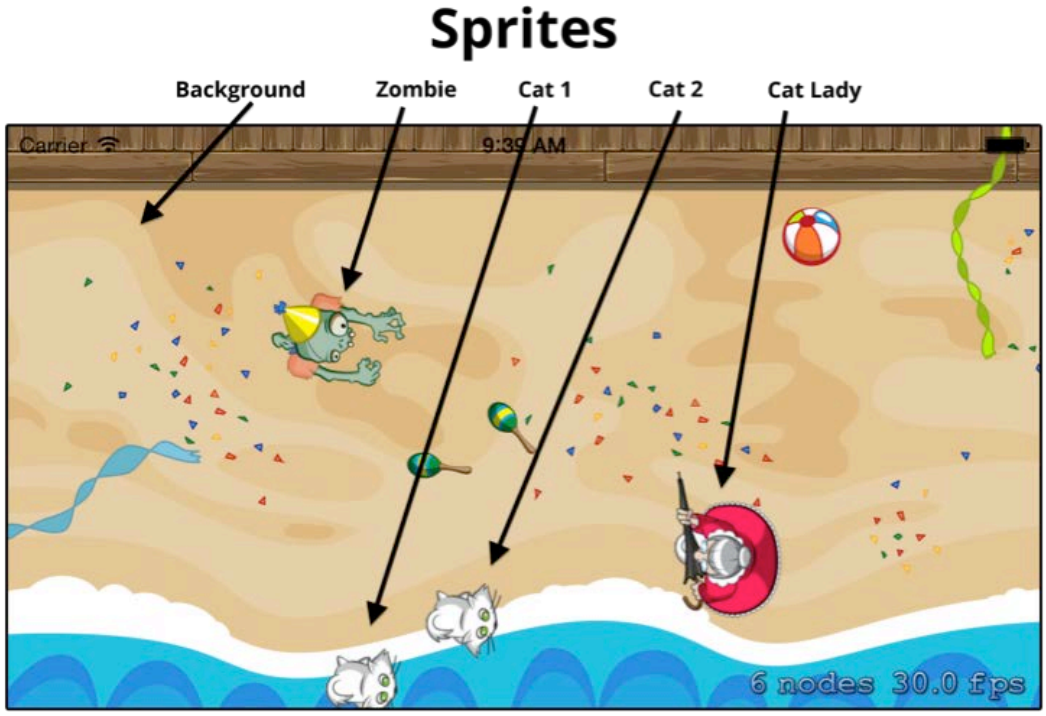
Which will be quickly followed by a (mostly) blank, black screen:

This may not look like much, but you now have a starting point upon which to build your first SpriteKit game.

Move on to the next task, which also happens to be one of the most important and common when making games: displaying an image on the screen.

Displaying a sprite

When making a 2D game, you usually put images on the screen representing your game's various elements: the hero, enemies, bullets and so on. Each of these images is called a **sprite**.



SpriteKit has a special class named `SKSpriteNode` that makes it easy to create and work with sprites. This is what you'll use to add all your sprites to the game.

Creating a sprite

Open **GameScene.swift** and add this line to `didMove(to:)`, right after you set the background color:

```
let background = SKSpriteNode(imageNamed: "background1")
```

You don't need to pass the image's extension; SpriteKit will automatically determine that for you.

Build and run, ignoring the warning for now. Hmm, you still see a blank screen — what gives?

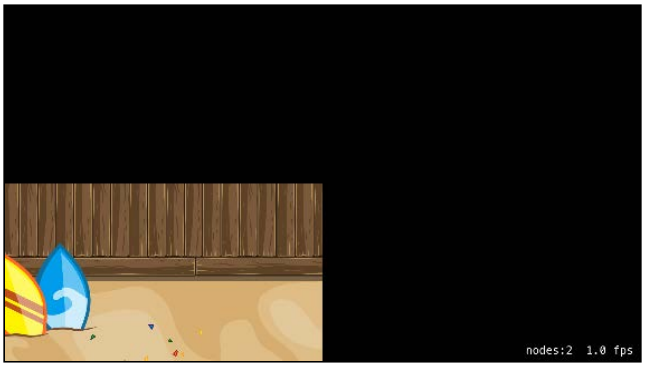
Adding a sprite to the scene

You haven't done anything wrong — it's just that a sprite won't show up onscreen until you add it as a child of the scene, or as one of the scene's descendant **nodes**.

To do this, add this line of code right after the previous line you added:

```
addChild(background)
```

You'll learn about nodes and scenes later. For now, build and run again, and you'll see part of the background appear in the bottom left of the screen:



Obviously, that's not quite what you want. To get the background in the correct spot, you have to set its position.

Positioning a sprite

By default, SpriteKit positions sprites so they are centered at (0, 0), which in SpriteKit represents the bottom left. Note that this is different from the UIKit coordinate system in iOS, where (0, 0) represents the top left.

Try positioning the background somewhere else by setting the position property. Add this line of code right before calling `addChild(background)`:

```
background.position = CGPoint(x: size.width/2, y: size.height/2)
```

Here, you position the background to the center of the scene. Even though this is a single line of code, there are four important things to understand:

1. The type of the `position` property is `CGPoint`, which is a simple structure that has `x` and `y` components:

```
public struct CGPoint {
  public var x: CGFloat
  public var y: CGFloat
  // ...
}
```

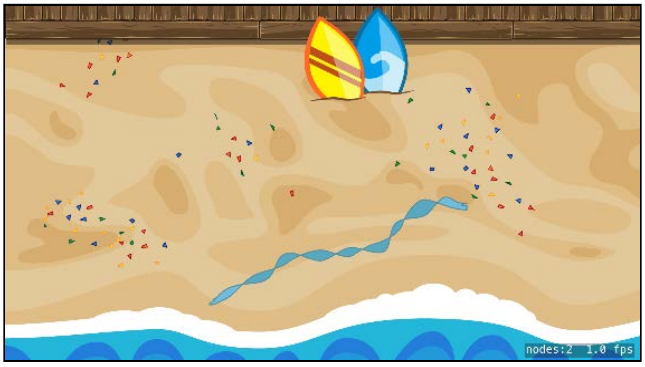
2. You can easily create a new `CGPoint` with the initializer shown above.

3. Since you're writing this code in an `SKScene` subclass, you can access the size of the scene at any time with the `size` property. The `size` property's type is `CGSize`, which is a simple structure like `CGPoint` that has width and height components.

```
public struct CGSize {
  public var width: CGFloat
  public var height: CGFloat
  // ...
}
```

4. A sprite's position is within the coordinate space of its parent node, which in this case is the scene itself. You'll learn more about this in Chapter 5, "Camera".

Build and run, and your background will be fully visible:



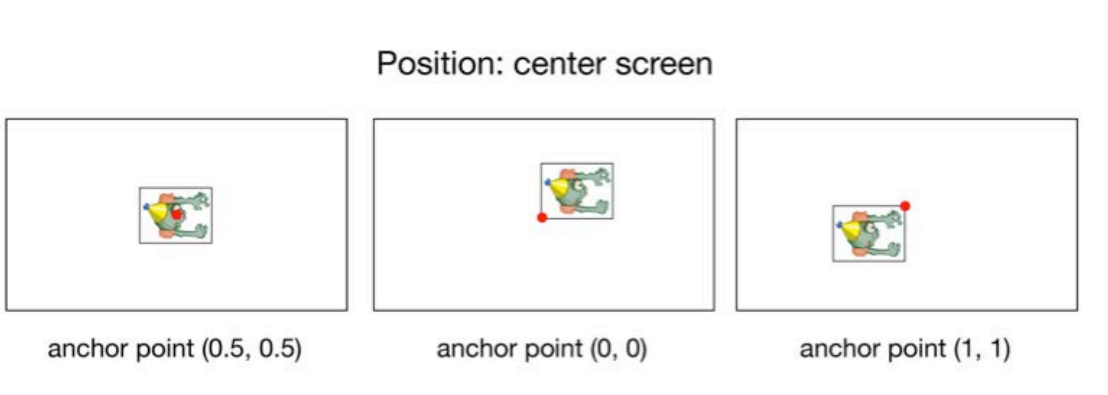
Note: You may notice you can't see the entire background on iPhone devices — parts of it overlap on the top and bottom. This is by design, so the game will work on both the iPad and the iPhone, as discussed in the "Universal app support" section earlier in this chapter.

Setting a sprite's anchor point

Setting the position of the background sprite means setting the *center* of the sprite to that position.

This explains why you could only see the upper-right portion of the sprite earlier. Before you set the position, the position defaulted to (0, 0), which placed the center of the sprite in the lower-left corner of the screen.

You can change this behavior by setting a sprite's anchor point. Think of the anchor point as "the spot within a sprite that you pin to a particular position". Here's an illustration showing a sprite positioned at the center of the screen, but with different anchor points:



To see how this works, find the line that sets the background's position to the center of the scene and replace it with the following:

```
background.anchorPoint = CGPoint.zero
background.position = CGPoint.zero
```

`CGPoint.zero` is a handy shortcut for (0, 0). Here, you set the anchor point of the sprite to (0, 0) to pin the lower-left corner of the sprite to whatever position you set — in this case, also (0, 0).

Build and run, and the image is still in the right spot:

This works because now you're pinning the lower-left corner of the background image to the lower-left corner of the scene.

Here, you changed the anchor point of the background for learning purposes. However, usually you can leave the anchor point at its default of (0.5, 0.5), unless you have a specific need to rotate the sprite around a particular point — an example of which is described in the next section.

So, in short: when you set the position of a sprite, by default you are positioning the center of the sprite.

Rotating a sprite

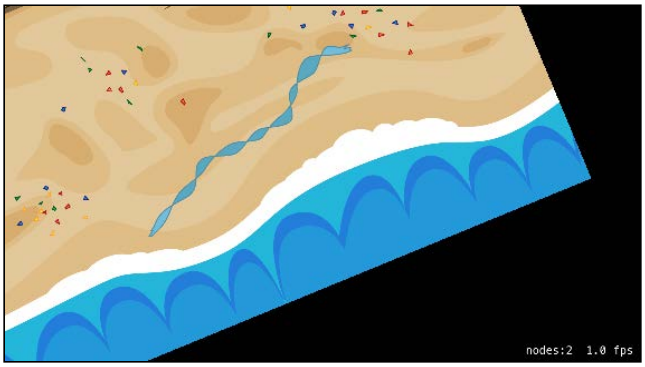
To rotate a sprite, you simply set its `zRotation` property. Try it out on the background sprite by adding this line right before the call to `addChild(_:)`:

```
background.zRotation = CGFloat(M_PI) / 8
```

Rotation values are in radians, which are units used to measure angles. This example rotates the sprite π / 8 radians, which is equal to 22.5 degrees. Also notice that you convert `M_PI`, which is a `Double`, into a `CGFloat`. You do this because `zRotation` requires a `CGFloat` and Swift doesn't automatically convert between types like some other languages do.

> **Note**: I don't know about you, but I find it easier to think about rotations in degrees rather than in radians. Later in the book, you'll create helper routines to convert between degrees and radians.

Build and run, and check out your rotated background sprite:



This demonstrates an important point: Sprites are rotated about their anchor points. Since you set this sprite's anchor point to (0, 0), it rotates around its bottom-left corner.

> **Note**: Remember that on the iPhone, the bottom-left of this image is actually offscreen! If you're not sure why this is, refer back to the "Universal app support" section earlier in this chapter.

Try rotating the sprite around the center instead. Replace the lines that set the anchor point and position with these:

```
background.anchorPoint = CGPoint(x: 0.5, y: 0.5) // default
background.position = CGPoint(x: size.width/2, y: size.height/2)
```

Build and run, and this time the background sprite will have rotated about the center:

This is all good to know! But for Zombie Conga, you don't want a rotated background, so comment out that line:

```
// background.zRotation = CGFloat(M_PI) / 8
```

If you're wondering when you might want to change the anchor point in a game, imagine you're creating a character's body out of different sprites, one each for the head, torso, left arm, right arm, left leg and right leg:

If you wanted to rotate these body parts at their joints, you'd have to modify the anchor point for each sprite, as shown in the diagram above.

But again, usually you should leave the anchor point at default unless you have a specific need, like the one shown here.

Getting the size of a sprite

Sometimes when you're working with a sprite, you want to know how big it is. A sprite's size defaults to the size of the image. In SpriteKit, the class representing this image is called a texture.

Add these lines after the call to `addChild(_:)` to get the size of the background and log it to the console:

```
let mySize = background.size
print("Size: \(mySize)")
```

Build and run, and in your console output, you'll see something like this:

```
Size: (2048.0, 1536.0)
```

Sometimes it's useful to get the size of a sprite programmatically, as you do above, instead of hard-coding numbers. Your code will be much more robust and adaptable.

Sprites and nodes

Earlier, you learned that to make a sprite appear onscreen you need to add it as a child of the scene, or as one of the scene's descendant **nodes**. This section will delve more deeply into the concept of nodes.

Everything that appears on the screen in SpriteKit derives from a class named `SKNode`. Both the scene class (`SKScene`) and the sprite class (`SKSpriteNode`) derive from `SKNode`.