Extracted from:

Apple Game Frameworks and Technologies

Build 2D Games with SpriteKit & Swift

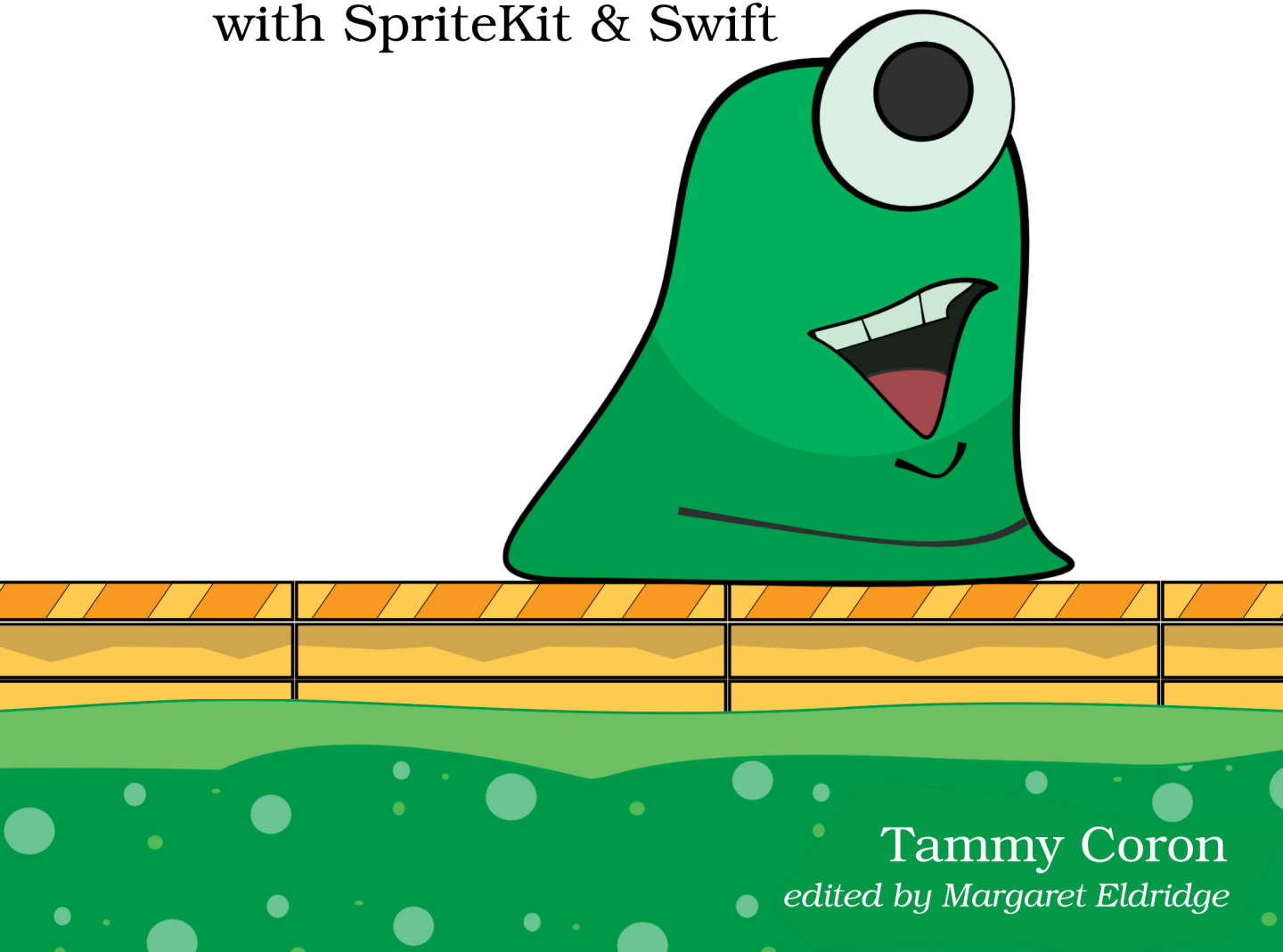
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Apple Game Frameworks and Technologies

Build 2D Games with SpriteKit & Swift



Tammy Coron

edited by Margaret Eldridge

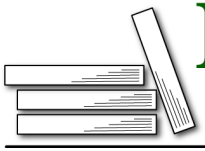
Apple Game Frameworks and Technologies

Build 2D Games with SpriteKit & Swift

Tammy Coron

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

The team that produced this book includes:

CEO: Dave Rankin
COO: Janet Furlow
Managing Editor: Tammy Coron
Development Editor: Margaret Eldridge
Copy Editor: Katharine Dvorak
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics
Founders: Andy Hunt and Dave Thomas

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-784-3
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—April 2021

To my children, Travis and Jake, and to my husband, Bill: I love you guys so very, very much. Life wouldn't be the same without you.

Preface

Why SpriteKit? Well, that's a good question—and one that I get asked quite often. For those who ask, my response is always the same: "Why *not* SpriteKit? It has everything you need to build great games."

With SpriteKit and the rest of the Apple game frameworks and technologies—like GameplayKit, Game Center, Xcode, and Swift—you have access to the tools you need to create high-performance, power-efficient games that work across the entire Apple ecosystem.

What's especially neat about SpriteKit is that it's built on top of Metal,[1] a robust Apple framework that provides near-direct access to the *graphics processing unit (GPU)*. Because SpriteKit leverages Metal, it's possible to tap into your game's full graphics and compute potential (using Metal) while also providing a simpler programming interface (using SpriteKit).

Another benefit of using SpriteKit, especially if you're already familiar with Xcode and Swift—even more so if you're new to game development—is that the learning curve isn't as steep as something like Unity or Unreal Engine. Don't get me wrong, Unity and Unreal Engine are phenomenal tools and well worth learning—I've used them both and will continue to do so—but there's something so beautifully simple, yet strangely powerful about SpriteKit.

So, is SpriteKit the best tool for making games? Well, maybe. It all depends on what you're trying to do and why you're doing it.

Suppose you have a fantastic idea for an exciting new game you want to make, but you're not sure what tools are best for making it, so you ask me for some advice. My first question for you would be, "Why do you want to make a game?" But I wouldn't stop there. I'd also ask you questions like:

- What kind of game are you making?
- What genre? Action, adventure, simulation, role-playing?

1. https://developer.apple.com/documentation/metal

- Is it a 2D game or a 3D game?
- What about making an augmented reality (AR) or virtual reality (VR) version? Do you have plans to do that?
- Are you looking to make a prototype, a full game, or both?
- Where do you want to publish your game?
- How much time do you want to spend developing your game?
- What about money and other resources? Are you funding the project yourself?
- Are you working with other developers or is this a solo project?
- Do you have any prior experience with programming or game development?

Without knowing *what* you want to do, it's a little difficult to suggest *how* to do it. For me, deciding what to use to build a game is rarely a difficult decision—for 2D games, I almost always start with SpriteKit. If SpriteKit isn't powerful enough or doesn't have a way to include a specific feature I want in my game, then I move on to something else. Most often, that's Unity.

So far, the only thing I can't do with the Apple game frameworks and technologies is create games for other platforms—so, if you want to build games for Android, Windows, or any other device outside of the Apple platform, this book may not be right for you. Then again, it's a great place to start if you're new to game development.

Who Should Read This Book

You should read this book, that's who! Yes, I realize I just made a very bold statement—heck, I even went against my own rule about using exclamation points (sorry about that).

So, how can I make such a bold statement? How do I know *you* should read this book? Simple. You're reading *this* page *right now,* so evidently you're curious what this book is all about. There's a reason for that—maybe even more than one.

Perhaps you think game development will be fun. Maybe you've tried it before but got so lost in the learning that you gave up, and now you're ready to try again. Or perhaps you're already familiar with Apple game frameworks and technologies and you're ready to see what you can *really do* with it.

Whatever your reasons were (and are) for picking up this book and reading this page (and hopefully all the ones that come next), thank you. Thank you for taking that first step, scary as it may be, to trying something new.

What Are Apple Game Frameworks and Technologies

Apple game frameworks and technologies[2] refers to a suite of related tools and application programming interfaces (APIs) that you can use to build games for the Apple platforms. This suite includes:

- ARKit
- Metal
- SceneKit
- SpriteKit
- ReplayKit
- GameplayKit
- Model I/O
- Game Center
- Game Controller
- On-Demand Resources
- Apple Arcade

Some of these frameworks and technologies listed are covered in this book.

What's in This Book

In this book, you'll build three exciting games: Gloop Drop—a new twist on a classic arcade game; Val's Revenge—a roguelike dungeon crawler; and Hog Dice—a social player versus player dice game.

Although the primary focus of this book is on building games in Xcode using SpriteKit and Swift, you'll also discover how to use other Apple game frameworks and technologies like GameplayKit and Game Center. You'll learn how to add pathfinding, artificial intelligence (AI), complex rule systems, and social player versus player features to your games.

But the learning (and fun) doesn't stop there. With the bonus chapters, you'll get to dip a proverbial toe or two into the waters of monetization. You'll find out what it takes to include in-app purchases and third-party ads with your games. (Seriously, who doesn't want to earn some extra money doing something they love, right?)

2. https://developer.apple.com/games

What's Not in This Book

Not many people want to read an 800+ page book while trying to learn something new—and even if they did, how many of those 800 pages would stick in their brains afterward?

There's a lot to Apple game frameworks and technologies, and although I eventually want to cover all of it, doing so in a single book is a terrible idea. The goal of learning (and teaching) shouldn't be about who makes it to the finish line first. Instead, it should be about the journey. With that thought in mind, I took a considerable amount of time deciding what to include in this book and what to save for another day.

So there is no confusion, misunderstanding, or disappointment for a lack of coverage, please understand that the following Apple game frameworks and technologies are *not covered* in this book:

- ARKit
- Metal
- SceneKit
- ReplayKit
- Game Controller
- On-Demand Resources
- Apple Arcade

How to Read This Book

If you're new to game development or Apple game frameworks and technologies, your best bet is to read this book cover-to-cover. The chapters and sample projects were carefully crafted and their order well planned. In a sense, these chapters tell a story and take you on a journey—some would say a hero's journey. (I don't want to spoil the surprise, but you, my friend, are the hero in this book. I'm just your guide.)

If reading something cover-to-cover isn't your style, don't worry; reading this book linearly is not mandatory. If there's a specific topic that interests you—let's say you want to know how to create and use tile maps or add physics to your games—no problem; every chapter in this book includes a starter project, so you can jump right in at any point without having to worry. I also include an ending project for every chapter, so if you get stuck, you can use those projects as a reference.

Conventions Used in This book

For the most part, this book uses standard conventions when it comes to programming in Swift. However, if you ask a group of 10 programmers how to create a method that returns some value, you're bound to get back 12 or so different solutions.

Is this difference in technique a bad thing? No, not at all. Having different choices is a good thing. If there was only one way for you to apply physics within your games, it might limit the types of games you can build—and making games shouldn't limit your imagination—it should help it flourish.

Don't misunderstand my words. There are rules you must (or at least should) follow and certain best practices and design guidelines[3] developers need to use—I encourage you to read them—but don't lose sight of your goal.

Game development should be fun, and how you code is an art unto itself. As developers, we all code differently. We have certain patterns we favor and stylistic choices we make. For instance, I use the prefix `setup` rather than `setUp` when naming methods that "set up" stuff. Is this wrong? Well, technically, it breaks the camel case of naming conventions. Set up (the verb) is two words, not one (as in setup, the noun or adjective).

The point is, as you work through the examples in this book, you'll more than likely come across some code or solution that you either 1) may have done differently yourself, or 2) had been taught to do another way. Try not to get distracted by these nuances—the diversity in code makes us all grow and discover new ways of doing things.

Your suggestions on how to improve the book's projects are welcome. Better yet, I encourage you to play around and find new ways of doing things. Remember, this is your journey; I'm simply your guide.

Online Resources

Several resources are available on the interwebs about game development, fewer about SpriteKit. For Apple-related resources and help, your first stop should always be the official Apple documentation.[4]

3. https://swift.org/documentation/api-design-guidelines
4. https://developer.apple.com/documentation/technologies

If you're new to Xcode[5] and Swift[6] development, you may want to read through the documentation for each.

Finally, if you find any errors in this book, need access to its source code, or want to discuss what you've read within its pages, you can find out more by visiting the book's website.[7]

Next Steps

I'm incredibly excited that you're taking this journey into the world of game development using the Apple game frameworks and technologies. I'm even more excited that you're bringing this book along with you.

Sure, you may not be fighting fire-breathing dragons or protecting some far off land from a band of evil forest dwellers, but it's still going to be a lot of fun. So, strap yourself in, because the adventure is about to begin.

5. https://developer.apple.com/documentation/xcode
6. https://swift.org/documentation
7. https://pragprog.com/titles/tcswift