Extracted from:

Apple Game Frameworks and Technologies

Build 2D Games with SpriteKit & Swift

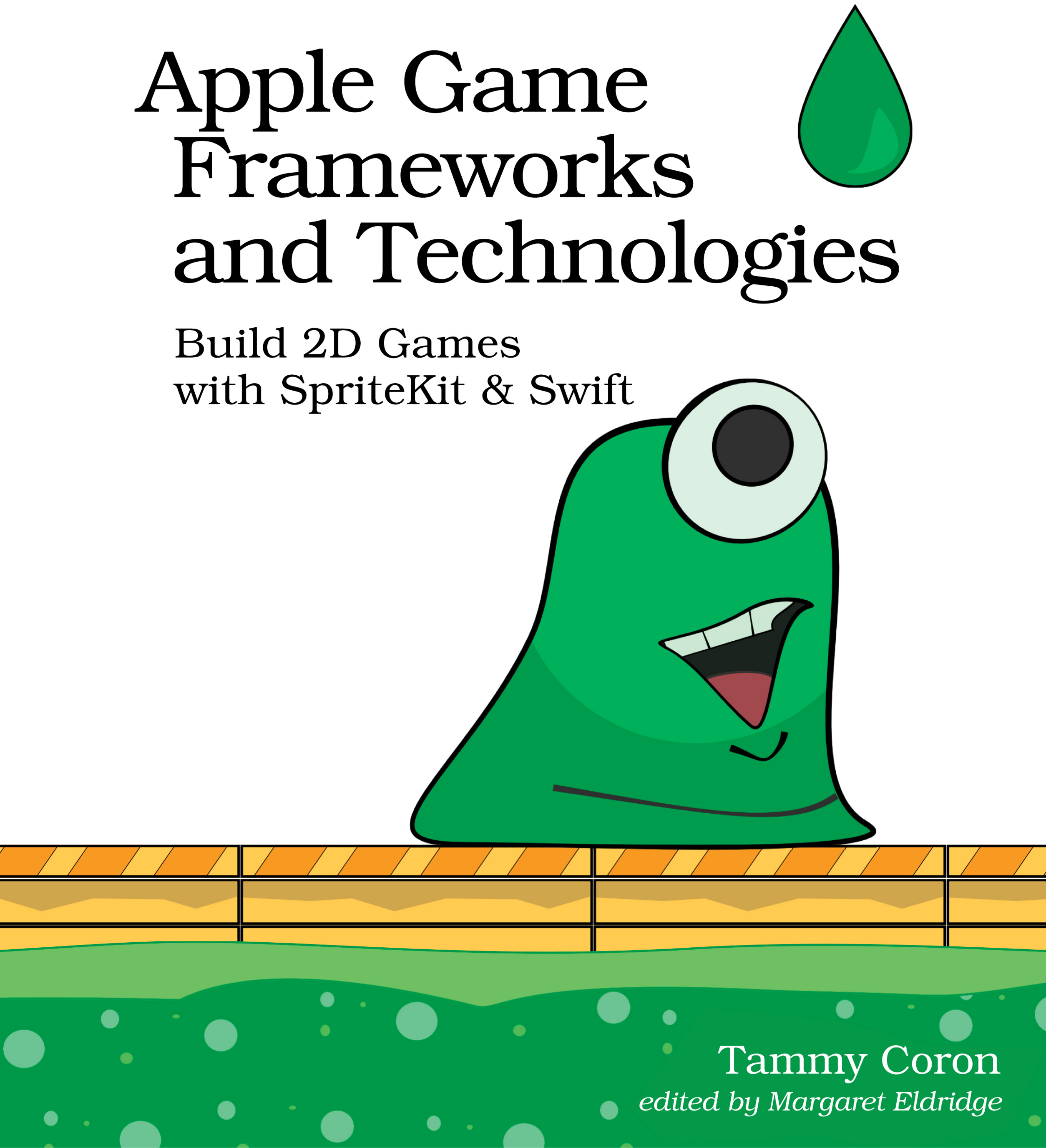
The Pragmatic Bookshelf

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

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.

Load the Textures

Remember, a texture is nothing more than an image, or a visual representation for your sprite. To animate a sprite in SpriteKit, you can load an array of textures and cycle through them. Given this is something you'll do a lot, it makes sense to use an *extension*[2] to load the textures.

Extensions, which are common to many programming languages, are a great way to reuse code, keep things organized, and extend the functionality of an existing class. To keep your code further organized, you'll put the extensions into a separate file.

Create another new file (⌘N) using the iOS Swift File template. Name the file SpriteKitHelper.swift and replace its contents with the following:

```
import Foundation
import SpriteKit

// MARK: - SPRITEKIT EXTENSIONS

extension SKSpriteNode {

}
```

You're now ready to add your first SKSpriteNode extension method. Within the brackets ({}) of the SKSpriteNode extension, add the following code:

```
// Used to load texture arrays for animations
func loadTextures(atlas: String, prefix: String,
                  startsAt: Int, stopsAt: Int) -> [SKTexture] {
  var textureArray = [SKTexture]()
  let textureAtlas = SKTextureAtlas(named: atlas)
  for i in startsAt...stopsAt {
    let textureName = "\(prefix)\(i)"
    let temp = textureAtlas.textureNamed(textureName)
    textureArray.append(temp)
  }

  return textureArray
}
```

This method takes four parameters: an atlas name, a prefix, and the start and stop frame numbers for the animation. It then uses a for-in loop to build and return the array of textures.

Now that you have a convenient way to load textures for your sprite nodes, you can use it to load the textures for Blob's walk cycle.

2. https://docs.swift.org/swift-book/LanguageGuide/Extensions.html

Use the Load Textures Extension

Open the Player.swift file. At the top of the file, below the import statements and above the class definition, add the following block of code:

```
// This enum lets you easily switch between animations
enum PlayerAnimationType: String {
  case walk
}
```

Because you'll add more animation types later, it makes sense to use an *enumeration*[3] using the keyword, enum. An enumeration is a data type you can use to store a set of named values. With the PlayerAnimationType enum, you can easily refer to specific animation types elsewhere in your code using memorable names like walk or run.

First, you need a private property to hold the walk textures. Inside the Player class below the line that reads // MARK: - PROPERTIES, add the following code:

```
// Textures (Animation)
private var walkTextures: [SKTexture]?
```

Next, you need to add the init() and init(coder:) methods. You may get some errors while adding these methods, but you can ignore them because they'll disappear once you've added all of the code.

Below the line that reads // MARK: - INIT, add the following block of code:

```
init() {

  // Set default texture
  let texture = SKTexture(imageNamed: "blob-walk_0")

  // Call to super.init
  super.init(texture: texture, color: .clear, size: texture.size())

  // Set up animation textures
  self.walkTextures = self.loadTextures(atlas: "blob", prefix: "blob-walk_",
                                        startsAt: 0, stopsAt: 2)

  // Set up other properties after init
  self.name = "player"
  self.setScale(1.0)
  self.anchorPoint = CGPoint(x: 0.5, y: 0.0) // center-bottom
}

required init?(coder aDecoder: NSCoder) {
  fatalError("init(coder:) has not been implemented")
}
```

3. https://docs.swift.org/swift-book/LanguageGuide/Enumerations.html

The init() method creates an SKTexture object using the first blob-walk image. It then makes a call to super.init.

After calling super.init, the init() method calls your new extension method and passes in the name of the atlas, the prefix for the images, and the start and end numbers of the image names for this animation. To better understand how this extension method works, consider the following code (but don't add it to your project):

```
// Create the array of textures
self.walkTextures = [SKTexture(imageNamed: "blob-walk_0"),
                     SKTexture(imageNamed: "blob-walk_1"),
                     SKTexture(imageNamed: "blob-walk_2")]
```

This code creates the walkTextures array in-line rather than by calling the extension method like the following code does:

```
self.walkTextures = self.loadTextures(atlas: "blob", prefix: "blob-walk_",
                                      startsAt: 0, stopsAt: 2)
```

Either way is acceptable, but with the extension method, you're able to simplify and reuse your code with other sprite nodes, making use of the *DRY (Don't Repeat Yourself) principle*.

The init() method also sets some additional properties like the name, scale, and anchorPoint for the player sprite node.

The second method you added is init(coder:). The init(coder:) method is a required method; it's used when initializing a sprite from a scene file. You're not using scene files yet, so there's not much you need to do with this method besides include it.

With these two methods in place, you're ready to add the player to the scene.

Add the Player to the Scene

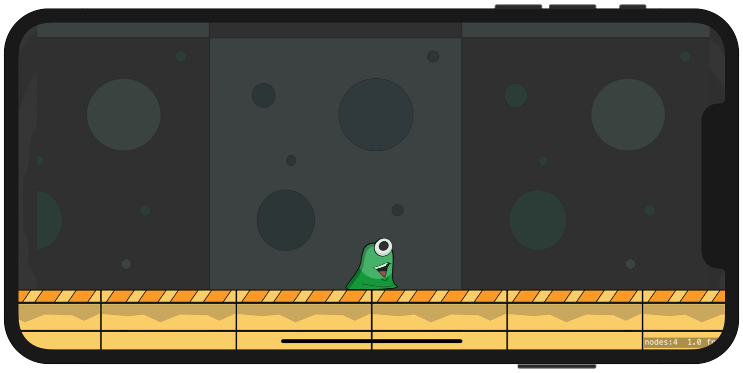
Adding the player to the scene is a lot like adding the background and foreground.

Start by opening the GameScene.swift file. At the end of the didMove(to:) method and below the code block that sets up the foreground, add the following code:

```
// Set up player
let player = Player()
player.position = CGPoint(x: size.width/2, y: foreground.frame.maxY)
addChild(player)
```

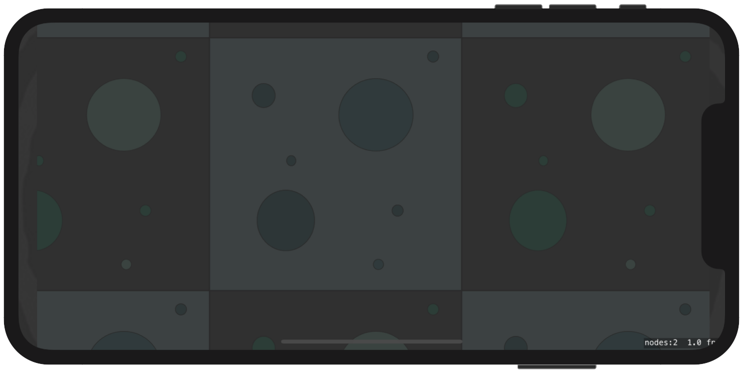
This block of code initializes an instance of the Player class and sticks that instance into a local variable. It then sets the position of the sprite node to the center of the scene and directly on top of the foreground node using the maxY property on foreground.frame. The maxY property returns the maximum y-value of a node, which in this case is the top of the foreground node.

Build and run the project.



So far, everything looks as expected, but there's a potential problem. Can you guess what it is?

No spoilers, but I'll give you a hint: it has to do with the render order. To see what I mean, move the background set-up code to the end of the didMove(to:) method after adding the player. Now, build and run the project again.



Notice how the foreground and player are missing. Well, technically, they're not missing—you just can't see them anymore because they're behind the background node. To fix this problem, you first need to learn how to control the render order using a node's z-position.

Control Render Order with Z-Position

When you add nodes to the scene, you're building a node tree, which you briefly read about in Chapter 1, Creating Scenes with Sprites and Nodes, on page ?. The order in which you add a node to the scene determines how it's rendered. In this case, you added the background node last, which places it on top of the other nodes.

Here's how it works:

- The scene renders itself, clearing its contents to its background color.
- The scene renders the foreground node, the player node, and finally, the background node.

You could build your scenes with this process in mind, but that can get complicated as you add and remove nodes. Luckily, you're able to change how things render by using a node's z-position. You can think of the z-position as the node's depth setting within the scene.

When you use z-positions to set up your nodes, the node tree gets rendered differently. Here's how it works:

- The global z-position for each node is calculated by recursively adding its z-position to its parent's z-position.
- The drawing order is determined by the node's z-position and is ordered from lowest to highest.
- If two nodes share the same z-position, parent nodes are rendered first, followed by siblings and their children. The child nodes are rendered in the order in which they appear in the parent's `children` array.

The default z-position for a node is 0. Setting this to a higher number brings it closer to the top. So, a z-position of 10, for example, is on top of a node whose z-position is set to 5.

Although you can set the z-position for each node using hard-coded numbers throughout your code, it's best to use an enum. Using an enum makes it easier to maintain your code as you build more advanced scenes.

Open the `SpriteKitHelper.swift` file and add the following code to the top of this file, below the import statements:

```
// MARK: - SPRITEKIT HELPERS

// Set up shared z-positions
enum Layer: CGFloat {
  case background
  case foreground
  case player
}
```

This code creates a new Layer enum that you can use for ordering the different nodes. Because player is last in the list, it has the highest number, which means any nodes that use this value will appear on the top-most layer.

Now that you have an enum available for setting a z-position value, you can use it to set the node's zPosition property.

Open the Player.swift file, and inside the init() method, add the following line after setting the anchorPoint:

```
self.zPosition = Layer.player.rawValue
```

Here, you're using one of the enum values you set up earlier to set the node's zPosition property. Because this is the player node, you want it to appear on top of everything else.

Next, open the GameScene.swift file, and inside the didMove(to:) method, update the code for both the background and foreground nodes.

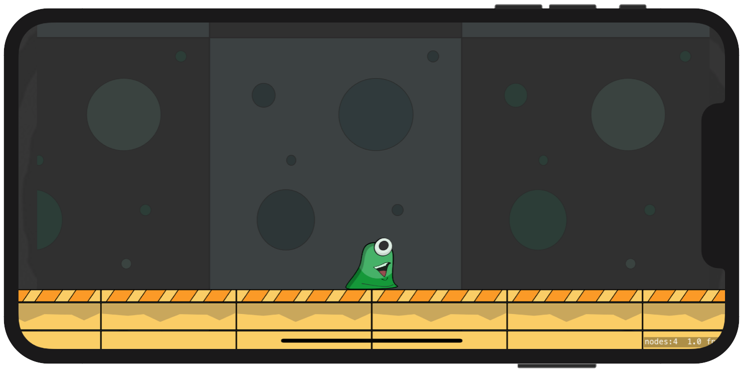
For the background node, below the line of code that sets its anchorPoint, add the following:

```
background.zPosition = Layer.background.rawValue
```

For the foreground node, below the line of code that sets its anchorPoint, add the following:

```
foreground.zPosition = Layer.foreground.rawValue
```

Build and run the project to confirm everything is working as expected.



Although you can keep the code as it is now—remember, in Add the Player to the Scene, on page 9, you swapped the order in which you're adding nodes to the scene—it's better to put everything back the way it was. For reference, the didMove(to:) method should look like this:

```
override func didMove(to view: SKView) {

  // Set up background
  let background = SKSpriteNode(imageNamed: "background_1")
  background.anchorPoint = CGPoint(x: 0, y: 0)
  background.zPosition = Layer.background.rawValue
  background.position = CGPoint(x: 0, y: 0)
  addChild(background)

  // Set up foreground
  let foreground = SKSpriteNode(imageNamed: "foreground_1")
  foreground.anchorPoint = CGPoint(x: 0, y: 0)
  foreground.zPosition = Layer.foreground.rawValue
  foreground.position = CGPoint(x: 0, y: 0)
  addChild(foreground)

  // Set up player
  let player = Player()
  player.position = CGPoint(x: size.width/2, y: foreground.frame.maxY)
  addChild(player)
}
```

Build and run the project again to make sure things continue to look as expected.

A Word about Render Order

In Chapter 1, Creating Scenes with Sprites and Nodes, on page ?, you may recall a property named ignoresSiblingOrder. When ignoresSiblingOrder is set to its default value of false, nodes within a scene are sorted and rendered in a deterministic order: parents before children, and then siblings in the order in which they appear in the node tree.

In contrast, when the ignoresSiblingOrder property is set to true, the render order is based entirely on a node's z-position. The default zPosition property value of a node is 0. While setting ignoresSiblingOrder = true offers an increase in performance, you must ensure that each node has its zPosition property set. In cases where two nodes share the same z-position, their render order is arbitrary and can change.

For most SpriteKit games, leaving ignoresSiblingOrder = false is recommended unless performance is an issue.

With your player image resources added and your base methods set up, you have everything in place to animate Blob's walk cycle.