Customizing the Behavior of a Node

Organize your app's logic and display code with nodes.

Overview

Delegate tasks to specific nodes as a way to organize your app's logic and display code. Learn which customization approach to take based on the particular behavior or look you want to enable.

Subclass a Node to Add Custom Behavior

Subclass SKNode (or one of its subclasses) when you need to customize the node's look or behavior. For example, you might subclass SKNode to implement a custom drawing layer. Or, you might subclass SKSpriteNode to add some AI logic. If you want a node to respond to user input, you must subclass it. SKScene is a subclass of SKNode, and you subclass SKScene to provide a custom look and behavior for your app.

Subclass a Node to Implement Node Archiving

If you add properties to a subclass and that subclass needs to be archived, the NSCoding protocol needs to be implemented on your subclasses.

Use a Shader Instead of Subclassing to do Custom Node Drawing

Unlike views, you cannot subclass SKNode to perform custom drawing. Instead, you use a node subclass that supports SKShader and implement your graphical effects in custom shader source code. Alternatively, you can composite a hierarchy of nodes that collectively effect the look you're going for.

Add Custom Logic to a Node

In many cases, expect to add methods that can be called during the scene's preprocessing and postprocessing steps. Your scene coordinates these steps, but focused node subclasses perform the work.

Handle User Input with a Node

If you want to implement event handling in a node class, you must implement separate event-handling code for iOS and macOS. The SKNode class inherits from NSResponder on macOS and UIResponder on iOS.

Delegate Tasks to a Node

In some app designs, you can rely on the fact that a particular combination of classes is always going to be used together in a specific scene. In other designs, you may want to create classes that can be used in multiple scenes. When two classes are dependent on each other, use delegation to break that dependency. Most often, you do this by defining a delegate on your node and a protocol for delegates to implement. Your scene (or another node, such as the node's parent) implements this protocol. Your node class can then be reused in multiple scenes, without needing to know the scene's class.