Getting Started with Actions

Create, configure, and run actions in SpriteKit.

Overview

You tell nodes to run an instance of SKAction when you want to animate contents of your scene. When the scene processes frames of animation, the actions are executed. Some actions are completed in a single frame of animation, while other actions apply changes over multiple frames of animation before completing. The most common use for actions is to animate changes to a node's properties. For example, you can create actions that move a node, scale or rotate it, or fade its transparency. However, actions can also change the node tree, play sounds, or even execute custom code.

Create and Run an Action

Do the following:

>> Create an action by calling the desired factory method; for example, fade(withDuration:).

>> Optionally configure additional properties, like speed or timingFunction.

>> Pass the action as an argument to a node's run(_:) function to execute it

Control Action Timing

When you run(_:) an action, SpriteKit manages the interpolation. That means that SpriteKit changes the properties of the node over more than one frame rendered by the scene automatically for you. As a result, you control the timing of the animation by configuring:

>> duration, which states how long that action takes to complete in seconds

>> timingMode, which defines the rate at which the animation executes

>> speed, which allows you to adjust the timing of the animation by increasing or decreasing its playback speed.

Create a Reverse Action

Many actions can be reversed, allowing you to create another action object that reverses the effect of that action. For example, if an action object moves a node 20 points to the right of its current position, the reversed action moves the node 20 points to the left. To create a reversed action object, call an action object's `reversed()` method.

Run Code when an Action Completes

The `run(_:completion:)` method is identical to the `run(_:)` method, but after the action completes, your block is called, but only if the action runs to completion. If the action is removed before it completes, the completion handler is never called. The following code fades out a sprite by running a `fadeOut(withDuration:)` action on it and, after the action completes, presents a new scene:

```
let fadeOut = SKAction.fadeOut(withDuration:2)
node.run(fadeOut) {
    skView.presentScene(newScene)
}
```

Reuse Actions

After you create an action, its type cannot be changed, and you have a limited ability to change its properties. SpriteKit takes advantage of this immutable nature to execute the actions efficiently. Because actions are effectively immutable objects, you can run the same action safely on multiple nodes in the tree at the same time. For this reason, if you have an action that is used repeatedly in your game, create a single instance of the action and then reuse it whenever you need a node to execute it.

Drive Game Logic

Because actions can be chained and can wrap custom code, it's possible to drive most of your app's logic using a complex system of actions. But where actions leave more sophisticated behavior to be desired, you might need to override the `SKSceneDelegate` callbacks and drive node behavior from there. For example, if a node needs to be moved in a wholly custom manner, you can adjust its position in `update(_:)` instead of using an action to do so.