Chapter 5

Game Building Blocks

The game DoodleDrop in the previous chapter was written to be easy to understand if you’re new to cocos2d. If you’re a more experienced developer, though, you probably noticed that there is no separation of code; everything is in just one file. Clearly, this doesn’t scale, and if you’re going to make bigger, more exciting games than DoodleDrop, you’ll have to find a suitable way to structure your code. Otherwise, you might end up with one class driving your game’s logic. The code size can quickly grow to thousands of lines, making it hard to navigate and tempting to change anything from anywhere, very likely introducing subtle and hard-to-find bugs.

Each new project demands its own code design. In this chapter, I’ll introduce you to some of the building blocks for writing more complex cocos2d games. The code foundation laid out in this chapter will then be used to create the side-scrolling shooter game we’ll be building in the next few chapters.

Working with Multiple Scenes

The DoodleDrop game had only one scene and one layer. More complex games will surely need several scenes and multiple layers. How and when to use them will become second nature for you. Let’s see what’s involved.

Adding More Scenes

The basics still apply. In Listings 4-1 and 4-2 in the previous chapter, I outlined the basic code needed to create a scene. Adding more scenes is a matter of adding more classes built on that same basic code. It’s when you’re transitioning between scenes that things get a little more interesting. There’s a set of three methods in CCNode that are called on each node object in the current scene hierarchy when you’re replacing a scene via the CCDirector replaceScene method.

The onEnter and onExit methods get called at certain times during a scene change, depending on whether a CCTransitionScene is used. You must always call the super
implementation of these methods to avoid input problems and memory leaks. Take a look at Listing 5–1, and note that all of these methods call the super implementation.

**Listing 5–1. The onEnter and onExit Methods**

```swift
-(void) onEnter
{
    // Called right after a node's init method is called.
    // If using a CCTransitionScene: called when the transition begins.
    [super onEnter];
}

-(void) onEnterTransitionDidFinish
{
    // Called right after onEnter.
    // If using a CCTransitionScene: called when the transition has ended.
    [super onEnterTransitionDidFinish];
}

-(void) onExit
{
    // Called right before node's dealloc method is called.
    // If using a CCTransitionScene: called when the transition has ended.
    [super onExit];
}
```

**NOTE:** If you don't make the call to the super implementation in the onEnter methods, your new scene may not react to touch or accelerometer input. If you don't call super in onExit, the current scene may not be released from memory. Since it's easy to forget this and the resulting behavior doesn't lead you to realize that it may be related to these methods, it's important to stress this point. You can see this behavior in the ScenesAndLayer01_WithBugs project.

These methods are useful whenever you need to do something in any node (CCNode, CCLayer, CCScene, CCSprite, CCLabelTTF, and so on) right before a scene is changed or right after. The difference from simply writing the same code in a node's init or dealloc method is that the scene is already fully set up during onEnter, and it still contains all nodes during onExit.

This can be important. For example, if you perform a transition to change scenes, you may want to pause certain animations or hide user interface elements until the transition finishes. Here's the sequence in which these methods get called, based on the logging information from the ScenesAndLayers02 project:

1. scene: OtherScene
2. init: <OtherScene = 066B2130 | Tag = -1>
3. onEnter: <OtherScene = 066B2130 | Tag = -1>