Contents

1 Preample

Welcome to **the** most compact yet detailed guide to iOS game programming. You will be guided through absolutely everything you need to know about Cocos2D and SpriteBuilder and 2d game programming in general.

While we will cover the very basics of game programming, such as scene graphs, animations and game loops - Objective-C, the language we will be using throughout the book is not in the scope of things you will learn. When starting this guide, you are expected to have a solid foundation of Objective-C knowledge.

The structure in which you will learn is the following:

- Tools: Get familiar with the very basics of Cocos2D and SpriteBuilder

- Infrastructure: Understand that on a high level a game consists of scenes. Understand how to create scenes and navigation paths through these scenes with Cocos2D and SpriteBuilder

- Action and Movement: Understand how objects in your game can be moved and animated. With Cocos2D and SpriteBuilder

- Interaction: Understand how user interaction can be captured, including Touch interaction and Accelerometer.

- Interobject Interaction: Understand how to use the delightfully integrated Chipmunk physics engine

- Beyond the Basics; Recipes and Best Practices: Once we have the basics, we will look at a ton of recipes and exciting Cocos2D classes, which you can use to create any kind of 2d game. Particle Effects, Custom Drawing, Custom Shaders, Tile Maps, Networking, Audio, cocos2d UI in depth, etc.

1.1 Structure of this book

This book shall function as a learning guide and a reference book. Therefore most examples will be small and self-contained. Instead of builind a game throughout the whole book, you will learn by implementing very small projects that are limited to the material we are currently discussing. That shall give you a better chance of understanding the concepts/code snippets and using them in your original game, instead starting of from an example game you have built in this book.

After we have discussed all the basics and you have a good understanding of the Cocos2D API I will point you to resources that provide example implementations for specific game types.

There are two different ways to read this book. From the front to the beginning, gaining knowledge in logical groups. Or if you aren't a beginner and would like to use this book as an example driven extension of the API reference you can look up pages by Class names or concept names. There is a special glossar in the back of this book.

1.2 Tools used throughout this book

The two main tools we will be using are Cocos2D and SpriteBuilder. Many of the problems that occur during game development can be solved by both of these tools. Wherever it makes sense I will point out both ways, one using only Cocos2D and one using SpriteBuilder. This will allow you to see the advantages of each approach and finally decide which tool you want to use in certain situations for your own games.

1.3 What is a 2D game engine?

2 Introduction to SpriteBuilder and Cocos2D

Now it's time to dive into 2D Game Development! For this chapter I will assume that you haven't written a game with a game engine so I will explain the relevant concepts fairly detailed.

2.1 Installing the software

In general there are two ways to install Cocos2D depending on whether you want to use SpriteBuilder or not. Throughout this book we will be using SpriteBuilder to set up all of our

projects, therefore we will only install SpriteBuilder which will come bundled with the latest version of Cocos2D.

Installing SpriteBuilder is easy, simply open the *App Store* app on your Mac and search for *SpriteBuilder*:

Figure 2.1: Cocos2D Technology Stack

Well done! Later throughout this chapter you will learn how to set up your first project!

2.2 Introduction to Cocos2D

To understand what Cocos2D is, it is helpful to look at the history of game development. The first video games were written in assembler (a very low level programming language) and images were drawn to the screen by manually setting colors for certain pixels. Since then a wealth of frameworks and libraries has been written to make the life of a game developer easier. Throughout this book you will learn how powerful Cocos2D is and how little code you will need for the basic building blocks of your games.

Here are some of the most important features of Cocos2D that make 2D game development a lot easier:

Structure like most game engines Cocos2D demands a specific structure of the code for your game, making most design decisions simple

Scene Graph and Sprites loading an image for a character and adding it to the gameplay scene can happen in two lines of code

Action System a sophisticated action system allows to define

movements of objects and animations

Integrated Physics Engine automatically calculates movements of objects, collisions and more.

There are a whole bunch of more features - but this brief outline shows the most important ones and should give you an idea why almost all game developers these days use game engines.

Cocos2D is built on top of OpenGL ES 2.0. If you have ever written OpenGL code before you know that it takes many lines of code to draw a textured rectangle on the screen. Cocos2D abstracts all of these tasks for you, you can go a very long way without writing any OpenGL code. The following diagram shows which technologies are used by Cocos2D:

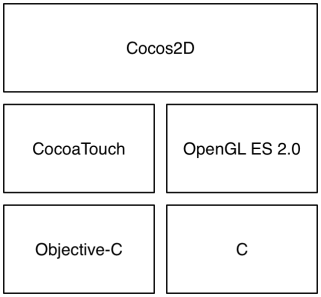


Figure 2.2: Cocos2D Technology Stack

As a Cocos2D developer you define which scenes exist in your game, which objects are part of these scenes and which size, position and appearance these objects have and Cocos2D will use OpenGL to render the scene you have described. In order to provide this functionality Cocos2D consists of many classes - some important ones will be discussed in this chapter. All Cocos2D classes use the *CC* prefix (CCScene, CCNode, etc.).

You have probably realized this already - when working with a 2D game engine for the first time you will be introduced to a whole set of new terminology. Just as a framework to write

desktop applications knows the concept of windows, buttons and mouse clicks a 2D game engine comes with its own set of terms and techniques. We will spent the next sections discussing the important terms in detail.

2.2.1 Scenes

For explaining the important terms and concepts of Cocos2D I will use a top down approach. First we will learn how games that use Cocos2D are structured. On the highest level of structure we have a concept called *scenes*. By default every scene in Cocos2D is full-screen. This means for every screen in your game you will use one scene.

Here's an example from the MakeGamesWithUs game *Deep Sea Fury*: You can see that the game consists of the start scene, the gameplay scene and the game over scene.

A Cocos2D developer creates scenes by using the CCScene class. Another important Cocos2D class for scene handling is CCDirector. The CCDirector class is responsible for deciding which scene is currently active in the game (Cocos2D only allows one active scene at a time). Whenever a developer wants

to display a scene or transition between two scenes he needs to use the CCDirector class.

This means creating and displaying a new scene is a two step process:

1. Create a new instance of CCScene
2. Tell the CCDirector to display this new scene

You will learn a lot more about this down the road, but the important bottom line is: *Scenes are the highest level of structure in your game and a class called CCDirector decides which scene is currently displayed.*

2.2.2 Nodes

Everything that is visible in your Cocos2D game (and a couple of invisible objects) are *nodes*. Nodes are used to structure the content of a scene. Every node can have other nodes as its children. Cocos2D provides a huge amount of different node types. Every node type is a subclass of CCNode.

Most nodes are used to represent an object on the screen (an image, a solid color, an UI element, etc.), a few other nodes

are only used to group other nodes. All nodes have a size, positions and children (and many other properties which are less important for us right now). Here are some of the popular node types of Cocos2D:

CCSprite represents an image or an animated image. Used for characters, enemies, etc.

CCColorNode a node being displayed in one plain color.

CCLabelTTF a node that can represent text in any TTF font.

CCButton a interactive node that can receive touches.

Nodes and their children form a scene graph. The concept of a scene graph isn't unique to Cocos2D it is a common concept of 2D and 3D graphics. A scene graph is a hierarchy of many different nodes.

2.2.3 Scene Graphs

Let's take a look at simple example of a scene graph:

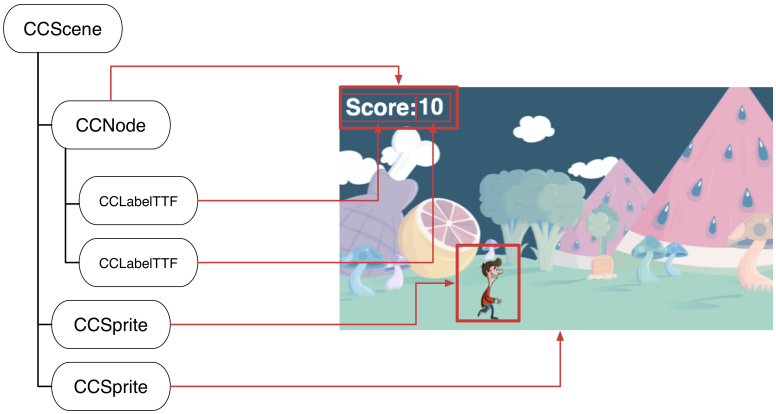


Figure 2.3: Cocos2D Scene Graph

On the left side of the image you can see the node hierarchy. On the first level you have the CCScene. Then we have a CCNode with to children of type CCLabel. This CCNode is the first example of a grouping node. Instances of CCNode don't have any appearance they are solely used to group other nodes. Throughout this book you will learn that it often makes sense to group nodes under certain parent nodes. The main reason is that all children are placed *relative* to their parents.

So if we would want to move the scoreboard of the example above to the top right corner we would only have to move the parent node instead of both child nodes. As you can imagine this becomes even more relevant in games that have ten or more entries in their scoreboard.

Structuring Nodes

Always group nodes that logically belong together under one parent node. That will save you a lot of time when you change the layout of your scene.

The other two objects in the scene graph are simpler. One represents the background image the other one the main character.

For some games scene graphs can get very complex and include hundreds of different nodes. The key takeaways for now are:

1. Every node in Cocos2D can have children
2. A hierarchy of nodes is called a scene graph

3. Children of nodes are placed relative to their parents - often it is useful to group nodes that are moved together under one parent

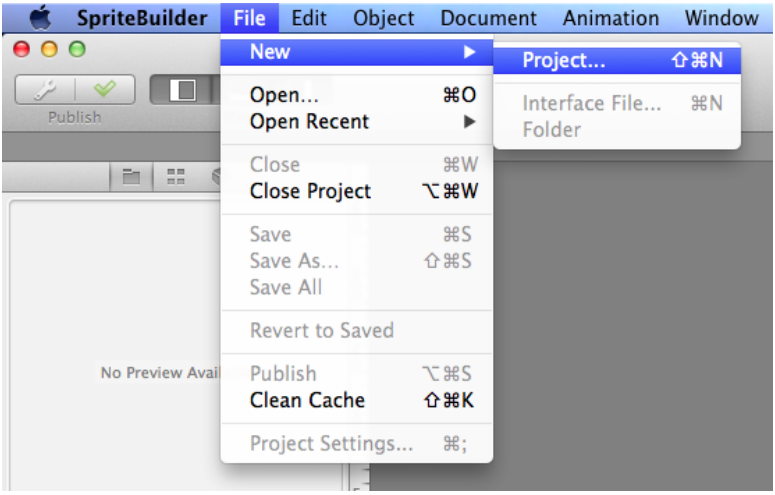
Now that you have a basic understanding of how games are structured in Cocos2D we will take a look at a second tool which we will be using throughout this book: SpriteBuilder.

2.3 Introduction to SpriteBuilder

You have learned the fundamentals of the game engine we will use. Now we will take a look at a tool called SpriteBuilder which we will use to create the majority of our game content. The main purpose of SpriteBuilder is to provide a visual editor for the creation of scenes, animations and more. For most games you will create some basic mechanics in code (enemy movement, score mechanism, etc.) but you will create the most of your game content in SpriteBuilder since it is a lot easier to create levels, menus and other scenes in an editor that provides you with a live preview instead of putting these scenes together in code.

2.3.1 Creating a first project

To dive into the features of SpriteBuilder we will create our first project! Create a new project by opening SpriteBuilder and selecting *File* > *New* > *Project...*:



SpriteBuilder will ask for a name and a location for the new project. Name it *HelloSB*. After you create the project the folder structure should look similar to this:

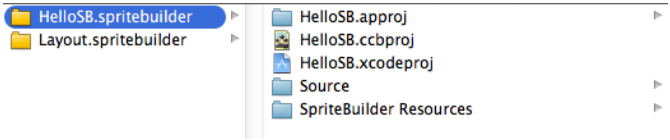


Figure 2.4: SpriteBuilder project folder structure

Every SpriteBuilder project is contained in a *.spritebuilder* folder. Within this folder all the files of the SpriteBuilder project are stored - along with an Xcode project.

SpriteBuilder and Xcode

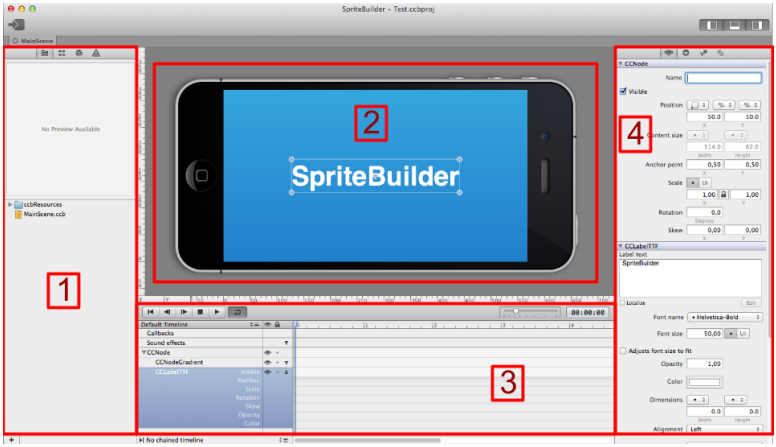
SpriteBuilder will create an Xcode project for every new project you create! The Xcode project will automatically contain the newest version of Cocos2D - very handy.

Later on you will learn more about how the SpriteBuilder project and the Xcode project work together. The general rule is that all code will be part of the Xcode project and most content creation will happen in the SpriteBuilder project.

2.3.2 The Editor

When you have created your first SpriteBuilder project you will see that the SpriteBuilder UI gets enabled. Let's take a look at the different parts of the editor to get a better understanding of SpriteBuilder.

The SpriteBuilder interface is divided into 4 main sections:



1. *Resource/Component Browser:* Here you can see the different resources and scenes you have created or added to your project. You can also select different types of Nodes

and drag them into your scene.

2. *Stage:* The stage will preview your current scene. Here you can arrange all of the Nodes that belong to a scene.

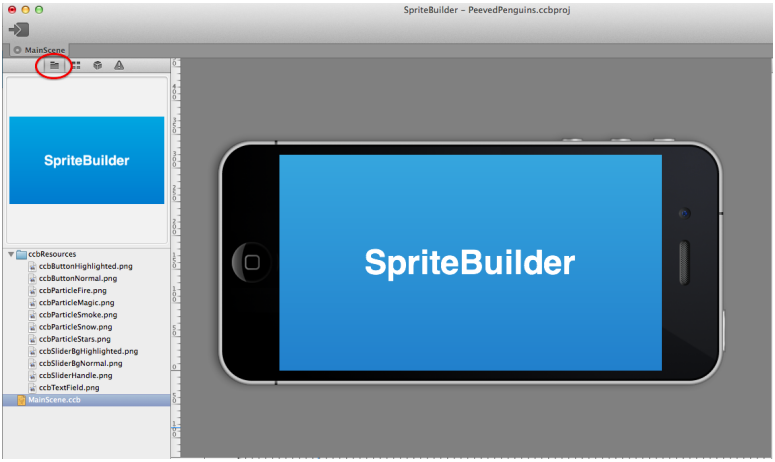
3. *Timeline:* The timeline is used to create animations within SpriteBuilder.

4. *Detail View:* Once you select a node in your scene, this detail view will display a lot of editable information about that node. You can modify positions, content (the text of a label, for example) and physics properties.

Let's take a closer look at some of the most important views.

File View

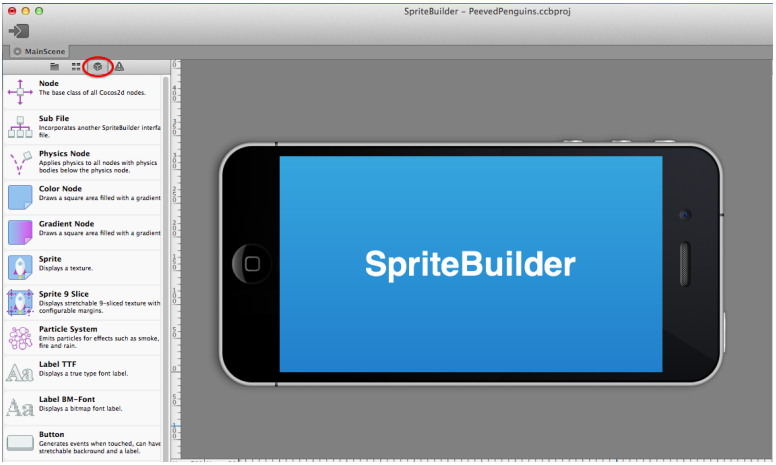
The first tab in the resource/component browser represents the *File View*. It lists all the *.ccb* files and resources that are part of the SpriteBuilder project:



In this view you can add new resources and restructure your project's folder hierarchy.

Node Library

The third tab in the left view is the Node Library:



This panel shows you all available node types you can use to construct your gameplay scenes and menus. You will drag these nodes from this view to the stage in the center to add them to your scenes.

Inspector

The first tab of the Detail View (the right panel) is the Inspector. Once you have selected an object on your stage you can use this panel to modify many of its properties, like position

and color:

Code Connections

The second tab on the right panel let's you manage code connections for your selected node. As mentioned previously the entire code for your games will be written as part of the Xcode project. This view allows you to create connections between the Xcode project and the SpriteBuilder project. For example you can set a custom Objective-C class for a node or you can select a method in your code that shall be called once a button

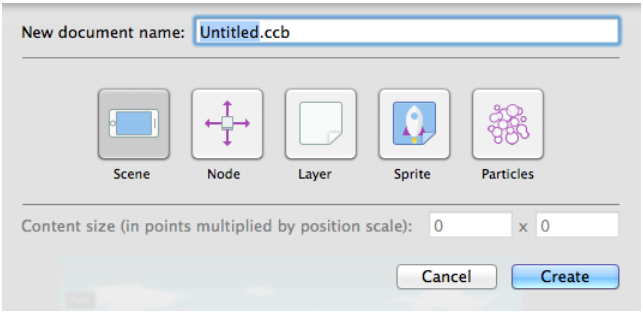
in your scene is tapped.

Code connections will be discussed in detail later on.

2.3.3 CCB Files

CCB Files are the basic building blocks of your SpriteBuilder project. Every scene in your game that is created with SpriteBuilder is represented by one CCB File. However CCB Files are not only used to create entire scenes - they are used to create any kind of scene graph. SpriteBuilder provides different kinds of templates depending on which type of scene graph

you want to create. You get an overview of the available CCB File templates when you create a new one, by selecting *New > File...* from the *File* menu in SpriteBuilder:



These are the different templates briefly explained:

Scenes will fill the full screen size of the device.

Nodes used primarily for grouping functionality, don't have a size.

Layers are nodes with a content size. This is useful, for instance, when creating levels or contents for scroll views.

Sprites used to create (animated) characters, enemies, etc.

Particles is used to design particle effects.

You will get a good understanding when to use which type of CCB File once we get started with our example projects. The key takeaway is that CCB Files are used by SpriteBuilder to store an entire scene graph including size, positions and many other properties of all the nodes that you have added.

2.3.4 How SpriteBuilder and Xcode work together

I have mentioned how SpriteBuilder and Xcode integrate a couple of times briefly. In order to be a well versed and efficient SpriteBuilder game developer it is very important to understand the details of this cooperation.

When creating a SpriteBuilder project, SpriteBuilder will create and maintain a corresponding Xcode project. In SpriteBuilder will you create multiple CCB Files that describe the content of the scenes in your game. You will also add the resources that you want to use in your game and set up code connections to interact with the code in your Xcode project. Xcode will be the place where you add code to your project and where you run the actual game.

Since Xcode is the tool that actually compiles and runs your game it needs to know about all the scenes and resources that are part of your SpriteBuilder project. Therefore SpriteBuilder has a **publish** functionality, provided by a button in the top left corner of the interface:

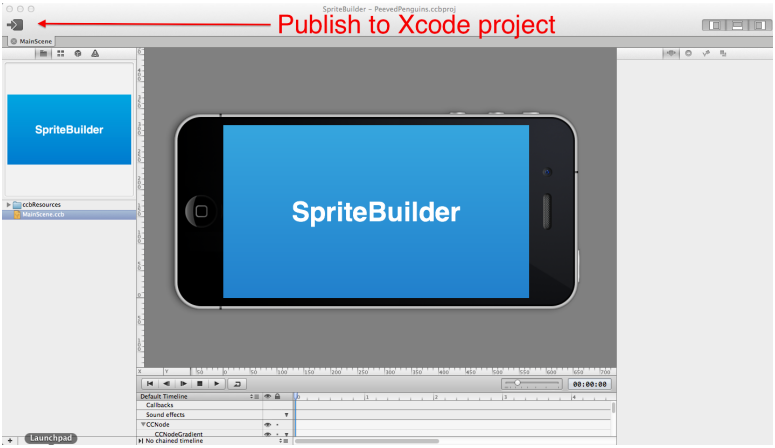


Figure 2.5: Use the publish button to update your Xcode project with the latest changes in your SpriteBuilder project.

Using that button, you publish your changes in your SpriteBuilder project to your Xcode project. Whenever you changed your SpriteBuilder project and want to run it you should hit this button before building the Xcode project.

Here's a diagram that visualizes how SpriteBuilder and Xcode work together:

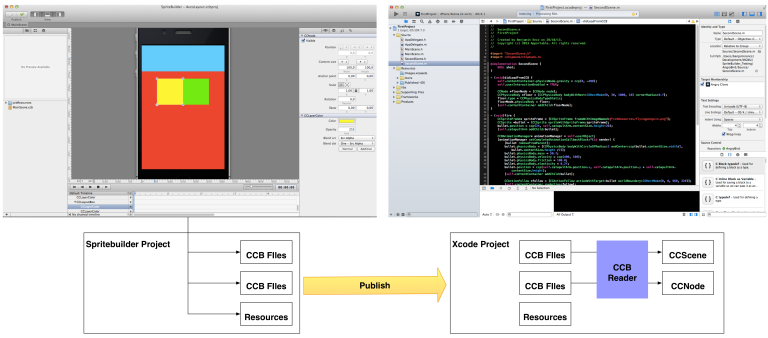


Figure 2.6: SpriteBuilder creates and organizes a Xcode project for you. Adding all the resources and scenes you have created.

CCB Files created in SpriteBuilder store a scene graph; the

hierarchy and positions of your nodes. When publishing a SpriteBuilder project the CCB Files and all other project resources are copied to your Xcode project. When running the project in Xcode a class called CCBReader will parse your CCB Files and create the according CCNode subclasses to reconstruct the scene graph you have designed in SpriteBuilder.

If you would use Cocos2D without SpriteBuilder you would manually create instances of CCNode, CCSprite, etc. in code and add children to these nodes - essentially building the entire scene graph in code.

When using SpriteBuilder the CCBReader class will build this scene graph for you, based on the information stored in the CCB Files that you created in SpriteBuilder.

Another important part of information contained in CCB Files that we have not discussed in detail yet are *Code Connections*.

2.3.5 Code Connections

Code connections are used to create links between your scenes in SpriteBuilder and your code in Xcode. There are three basic

types of code connections:

Custom Classes are an important information for the CCBReader. As mentioned previously the CCBReader builds the scene graph by creating different nodes based on the information in your CCB File. By default it will create an instance of CCSprite for every sprite you added in SpriteBuilder an instance of CCNode for every node you added, etc. Often however you will want to add custom behaviour to a node (for example a movement pattern for an enemy). Then you will have to use the *Custom Class* property to tell the CCBReader which class it should instantiate instead of the default one. Whichever class you enter here needs to be a subclass of the default class (e.g. a subclass of CCSprite). You will learn how to use this feature in the final project of this chapter!

Variable Assignments If you have assigned a *Custom Class* you can use variables assignments to retrieve references to different nodes in the scene. For example a character might want a reference to its right arm node (a child of the character node) in order to move it.

Callbacks are only available to UI elements like buttons and sliders. They allow you to decide which method should be called on which class once a button is pressed.

Now you should have an idea about what code connections are used for and which kinds exist. We will discuss the details of all types when we use them as part of our example projects.

2.4 A first SpriteBuilder project

You have already created the SpriteBuilder project called *HelloSB*. Now we will start adding some content to it. During this exercise we will create a small first project that will teach you all of the following:

- Creating scenes in SpriteBuilder
- Creating code connections