Learning Cocos2d-JS Game Development

Learn to create robust and engaging cross-platform HTML5 games using Cocos2d-JS

Foreword by
Shun Lin, Cofounder of Cocos2d-x
Huabin LING, Lead Programmer of Cocos2d-JS

Emanuele Feronato
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 “Hello World – A Cross-platform Game”
- A synopsis of the book’s content
- More information on Learning Cocos2d-JS Game Development

About the Author

**Emanuele Feronato** has been studying programming languages since the early 1980s, with a particular interest in game development. He has taught online programming for European Social Fund (ESF), and then founded a web development company in Italy.

As a game developer, Emanuele developed Flash games sponsored by the biggest game portals and his games have been played more than 90 million times. He now ports most of them on mobile platforms and develops HTML5 games, which have been featured in the most important mobile web markets, such as Amazon.

As a writer, he has worked as a technical reviewer for Packt Publishing and published the books *Flash Game Development by Example* and *Box2D for Flash Games*.

His blog, www.emanueleferonato.com, is one of the most visited blogs about indie game development.

First, I would like to thank Packt Publishing for giving me the opportunity to write this book, especially Richard Harvey, Azharuddin Sheikh, and Samantha Gonsalves for helping me improve the book quality.

The biggest thank you obviously goes to my blog readers and my Facebook fans for appreciating my work, giving me the will to write more and more. Also, thank you for playing my games. Hope you enjoy playing them as much as I enjoyed developing them.

Finally, a special thank you to my wife, Kirenia, for her patience while I was writing the book.
Learning Cocos2d-JS
Game Development

Mobile HTML5 casual games are experiencing a golden age these days. Not only famous Flash hits and successful native mobile games, but also original games tailored for mobile browsers are played by millions of players everyday.

With a continuously growing number of mobile devices available to play HTML5 games, and with each one having its own resolution, and display size, and ratio, creating a game version for each device would make you waste a lot of time. That's why there's a new way to make HTML5 games called cross-platform. It means you create a game once, and some magic adapts it to every device capable of running HTML5 content.

This magic is called Cocos2d-JS, which allows you to focus on what you actually love, making games, and takes care of screen resolutions and ratios. Also, you will be using one of the simplest and most widely known languages ever: JavaScript. Last but not least, it's free. You can create the next big game at no cost.

Throughout this book, you will learn how to make games using Cocos2d-JS in the most fun and productive way. We know you hate boring theories, so we did our best to focus the entire book on one single word: action.

Learn Cocos2d-JS with examples taken from the most successful game genres; see how easy it is to create a game once, and then make it run on each device.

What This Book Covers

Chapter 1, Hello World – A Cross-platform Game, will show you how to create a blueprint that you will be using for every game you will make with Cocos2d-JS. This chapter will also guide you through the creation of an environment to create and test your cross-platform games.

Chapter 2, Adding Interactivity – The Making of a Concentration Game, guides you through the creation of one of the most popular games that makes you learn how to create sprites and interact with them both with the mouse and finger.

Chapter 3, Moving Sprites Around the Screen – An Endless Runner, introduces scrolling and sprite movements using motion tweens in the creation of a space-themed endless runner. Particle effects and collision detection complete the game experience.

Chapter 4, Learn about Swipes through the making of Sokoban, shows you how to create a classic puzzle game that you can control with the most intuitive way used these days: swiping on the game.
Chapter 5, *Become a Musical Maestro*, examines the importance of having sound effects and background music in your games. This chapter also shows you how to start and stop sounds and loops by adjusting their volume.

Chapter 6, *Controlling the Game with Virtual Pads*, provides you three different ways to control your games with virtual pads when you build a fruit game. Virtual buttons and analogic pads will be your best friends when you create your next game.

Chapter 7, *Adding Physics to Your Games Using the Box2D Engine*, shows you the magic of physics engines in the creation of the blockbuster Totem Destroyer game, which you will easily build using the Box2D physics engine. You will also learn how to create, destroy, skin, and interact with physics bodies in a realistic physics environment.

Chapter 8, *Adding Physics to Your Games Using the Chipmunk2D Engine*, creates the same Totem Destroyer game, this time using Chipmunk2D rather than Box2D. While the Web is divided by the *Box2D versus Chipmunk2D* war, you will learn both of them, leaving the choice of your favorite physics engine up to you.

Chapter 9, *Creating Your Own Blockbuster Game – A Complete Match 3 Game*, completes the book with the creation of the most successful game genre today: a Match-3 game. At the end of the chapter, you will also find some hints and suggestions on protecting, promoting, marketing, and monetizing your game.
Hello World – A Cross-platform Game

The legend says that the first working script you should do when learning a new language is the classic Hello World printed somewhere on the screen.

This chapter will guide you through the creation of a cross-platform Hello World example, covering these concepts:

- The theory behind the creation of cross platform games
- Cocos2d-JS installation and setup
- A Cocos2d-JS project blueprint
- Scenes, Layers, and Sprites
- Preloading images
- Adding images
- Removing images

By the end of the chapter, you will be able to create a template project to create any kind of Cocos2d-JS cross-platform game that is capable of running on various devices at different resolutions.
Why should I make cross-platform games?

This is a very important question. I asked it to myself a lot of times when HTML5 mobile gaming started to become popular. I was just thinking it was a waste of time to simply care about the different screen resolutions and aspect ratios, so my first HTML5 game was made to perfectly fit my iPad 2 tablet.

When I finally showed it to sponsors, most of them said something like "Hey, I like the game, but unfortunately it does not look that good on my iPhone". "Don't worry", I said, "you'll get the game optimized for iPad and iPhone". Unfortunately, it did not look that good on the Galaxy Note. Neither did it on the Samsung S4.

You can imagine the rest of this story. I found myself almost rewriting the game with a series of if.. then.. else loops, trying to make it look good on any device.

This is why you should make a cross-platform game: To code once and rule them all. Focus on game development and let a framework do the dirty work for you.

What Cocos2d-JS is and how it works

Cocos2d-JS is a free open source 2D game framework. It can help you to develop cross-platform browser games and native applications. This framework allows you to write games in JavaScript. So, if you have already developed JavaScript applications, you don't have to learn a new language from scratch. Throughout this book, you will learn how to create almost any kind of cross-platform game using a familiar and intuitive language.

Requirements to run Cocos2d-JS

Before you start, let's see what software you need to install on your computer in order to start developing with Cocos2d-JS:

- Firstly, you need a text editor. The official IDE for Cocos2d-JS coding is Cocos Code IDE, which you can download for free at http://www.cocos2d-x.org/products/codeide. It features auto completion, code hinting, and some more interesting characteristics to speed up your coding. If you are used to your favorite code editor, that's fine. There are plenty of them, but I personally use PSPad (you can find this at http://www.pspad.com/) on my Windows machine and TextWrangler (you can find this at http://www.barebones.com/products/textwrangler/) on the Mac. They are both free and easy to use, so you can download and have them installed in a matter of minutes.
• To test your Cocos2d-JS projects, you will need to install a web server on your computer to override security limits when running your project locally. I am using WAMP (http://www.wampserver.com/) on my Windows machine, and MAMP (http://www.mamp.info/) on the Mac.

Again, both are free to use as you won't need the PRO version, which is also available for Mac computers. Explaining all the theory behind this is beyond the scope of this book, but you can find all the required information as well as the installation documentation on the official sites.

• If you prefer, you can test your projects directly online by uploading them on an FTP space you own and call them directly from the web. In this case, you don't need to have a web server installed on your computer, but I highly recommend using WAMP or MAMP instead.

• I personally use Google Chrome as the default browser to test my projects. As these projects are meant to be cross-platform games, it should run in the same way on every browser, so feel free to use the browser you prefer.

The latest information about Cocos2d-JS can be found on the official page http://www.cocos2d-x.org/wiki/Cocos2d-JS, while the latest version can be downloaded at http://www.cocos2d-x.org/download.

Cocos2d-JS is updated quite frequently, but at the time of writing, the latest stable release is v3.1. Although new releases always bring some changes, all examples included in this book should work fine with any release marked as 3.x as there aren't huge changes in the roadmap.

You will notice the download file is a ZIP file that is greater than 250 MB. Don't worry. Most of the content of the package is made by docs, graphic assets, and examples, while the only required folder, at the moment, is the one called cocos2d-html5.
The structure of your Cocos2d-JS project

Every HTML5 game is basically a web page with some magic in it; this is what you are going to create with Cocos2d-JS: a web page with some magic in it.

To perform this magic, a certain file structure needs to be created, so let's take a look at a screenshot of a folder with a Cocos2d-JS project in it:

This is what you are going to build; to tell you the truth, this is a picture of the actual project folder I built for the example to be explained in this chapter, which is placed in the WAMP localhost folder on my computer. It couldn't be any more real.

So, let's take a look at the files to be created:

- **cocos2d-html5:** This is the folder you will find in the zip archive.
- **index.html:** This is the web page that will contain the game.
- **main.js:** This is a file required by Cocos2d-JS with the Cocos2d-JS function calls to make the game start. You will create this within the next few minutes.
- **project.json:** This is a *JavaScript Object Notation (JSON)* with some basic configurations. This is what you need to make your game run. Well, almost, because the actual game will be placed in the *src* folder. Let's see a few other things first.
Hello Cross-World

The time has come, the boring theory has ended, and we can now start coding our first project. Let's begin!

1. Firstly, create a page called index.html in the root of the game folder and write this HTML code:

```html
<!DOCTYPE html>
<head>
  <title>My Awesome game</title>
  <script src="cocos2d-html5/CCBoot.js" type="text/javascript"></script>
  <script src="main.js" type="text/javascript"></script>
</head>
<body style="padding:0;margin:0;background-color:#000000;">
</body>
</html>
```

There's nothing interesting in it as it is just plain HTML. Let's take a closer look at these lines to see what is going on:

```html
<script src="cocos2d-html5/CCBoot.js ">
</script>
```

Here, I am including the Cocos2d-JS boot file to make the framework start:

```html
<script src="main.js"></script>
```

From the preceding line, this is where we call the script with the actual game we are going to build. Next, we have the following code:

```html
<canvas id="gameCanvas"></canvas>
```

This is the canvas we will use to display the game. Notice here that the canvas does not have a width and height, as they will be defined by the game itself.
2. Next is the creation of `main.js`: the only file we will call from our main `index.html` page. This is more of a configuration file rather than the game itself, so you won't code anything that is game-related at the moment. However, the file you are going to build will be the blueprint you will be using in all your Cocos2d-JS games.

The content of `main.js` is as follows:

```javascript
cc.game.onStart = function(){
    cc.view.setDesignResolutionSize(320, 480, cc.ResolutionPolicy.SHOW_ALL);
    cc.director.runScene(new gameScene());
};
cc.game.run();
```

Don't worry about the code at the moment; it looks a lot more complicated than it really is. At the moment, the only line we have to worry about is the one that defines the resolution policy.

One of the most challenging tasks in cross-platform development is to provide a good gaming experience, no matter what browser or what device the game is running on. However, the problem here is that each device has its own resolution, screen size, and ratio.

Cocos2d-JS allows us to handle different resolutions in a similar way web designers do when building responsive design. At the moment, we just want to adapt the game canvas to fit the browser window while targeting the most popular resolution, which is 320x480 (portrait mode). That's what this line does:

```javascript
cc.view.setDesignResolutionSize(320, 480, cc.ResolutionPolicy.SHOW_ALL);
```

Using these settings, you should be pretty sure that your game will run on every device, although you will be working in a low resolution.

Also, have a look at this line:

```javascript
cc.director.runScene(new gameScene());
```

Basically, a Cocos2d-JS game is made by a scene where the game itself runs. There can be more scenes in the same game. Imagine a scene with the title screen, a scene with the game over screen, and a scene with the game itself. At the moment, you only have one scene called `gameScene`. Remember this name because you are going to use it later.
3. Following this, the next required blueprint file you are going to build is project.json, which has some interesting settings. Let’s take a look at the file first:

```
{
   "debugMode" : 0,
   "showFPS" : false,
   "frameRate" : 60,
   "id" : "gameCanvas",
   "renderMode" : 0,
   "engineDir":"cocos2d-html5/",

   "modules" : ["cocos2d"],

   "jsList" : [  
      "src/gamescript.js"
   ]
}
```

What do these lines mean? Let's see them one by one:

- **debugMode**: This is the object key that determines the level of debug warnings. It has a range from 0 to 6. Leave it at 0 at the moment since the project is very simple and we won't make any errors.
- **showFPS**: This object can be **true** or **false**; it shows or hides the FPS meter on the screen.
- **frameRate**: This object sets the frame rate of your game. Set it to 60 to have a smooth game.
- **id**: This is the DOM element that is required to run the game. Do you remember you gave your canvas the gameCanvas id? Here you are.
- **engineDir**: This is the folder where Cocos2d-JS is installed.
- **modules**: This object engines the modules to load. At the moment, we only need the basic Cocos2d library.
- **jsList**: This is an array with the files used in the game. This means we are going to create our game in src/gamescript.js.

4. Finally, we arrive at the game script itself. This is the one that will contain the actual game, gamescript.js, which at the moment is just a plain declaration of the game scene:

```
var gameScene = cc.Scene.extend({
   onEnter:function () {
      this._super();
   }
});
```
console.log("my awesome game starts here");
}
});

Here, you want to save everything and call index.html page from your localhost (refer to your WAMP or MAMP docs) in your browser. If you now open the developer console, you should see:

my awesome game starts here

Congratulations! This means you have successfully managed to create a Cocos2d-JS template file to build your future games.

Let's build our first mini game at once!

**Preloading and adding images**

In this example, I am using a 64x64 PNG image representing a target, as shown in the following figure:

![Target Image](image.png)

You are obviously free to use whatever image you prefer.

When you load a web page, in most cases, the page is loaded and shown before all images are loaded. This might sound okay on a web page because readers won't mind if they have to wait a couple of seconds before an image is showed, but this definitively can't happen in a game. This means our images need to be preloaded, and Cocos2d-JS can easily handle this. The steps on how to preload images in your game are as follows:

1. This is the first time you add this line to the `project.json` file:

    ```json
    {
        "debugMode" : 0,
        "showFPS" : false,
        "frameRate" : 60,
        "id" : "gameCanvas",
        "renderMode" : 0,
    ```

---
"engineDir":"cocos2d-html5/",
"modules" : ["cocos2d"],

"jsList" : [
   "src/loadassets.js",
   "src/gamescript.js"
]
}

This means you are going to create another file called `loadassets.js` in the same `src` folder where you just created `gamescript.js`.

This is the content of `loadassets.js`:

```javascript
var gameResources = [
   "assets/target.png"
];
```

An array called `gameResources` stores the assets to preload. So, you should create a folder called `assets` and place the `target.png` image inside this folder.

To keep the project organization clear, I am going to place all game assets in a folder called `assets`.

2. Now that Cocos2d-JS is aware which images need to be preloaded, we only need to tell the game that it has to preload them before the scene starts, so we need to add a couple of lines to `main.js`:

```javascript
cc.game.onStart = function(){
   cc.view.setDesignResolutionSize(320, 480, cc.ResolutionPolicy.SHOW_ALL);
   cc.LoaderScene.preload(gameResources, function () {
      cc.director.runScene(new gameScene());
   }, this);
};
cc.game.run();
```

The `cc.LoaderScene.preload` constructor will preload scene resources taken from the `gameResources` array defined in `loadassets.js`. All puzzle pieces match perfectly.
3. Finally, let's add the target to the game by rewriting the `gamescript.js` file:

```javascript
var gameScene = cc.Scene.extend({
    onEnter:function () {
        this._super();
        var gameLayer = new game();
        gameLayer.init();
        this.addChild(gameLayer);
    }
});
var game = cc.Layer.extend({
    init:function () {
        this._super();
        var target = cc.Sprite.create("assets/target.png");
        this.addChild(target,0);
    }
});
```

If you developed Flash games using **AS3 (ActionScript 3)**, you will find Cocos2d-JS assets hierarchy familiar to display objects. If you are new to this, allow me to explain what happens:

1. Like all frameworks that deal with graphic resources, Cocos2d-JS has hierarchy rules. On the top of such a hierarchy, we find the `Scene` object. Each scene contains some game logic; think about a main menu scene, a game scene, and a game over scene.

2. Each scene contains one or more `Layer` objects; layers define which content should be at the top of other content. In a real-world example, a level background is in the bottom-most layer, player and enemies will be created in a layer above the background, and game information such as score and remaining lives are placed on the topmost layer.

3. Finally, all layers can have one or more `Sprite` objects, which are the graphic assets themselves such as the player, the enemies, or in this case, the target.

4. To summarize, the code means that once `gameScene` is executed, create and add the `game` layer, and in this layer, add the `target` sprite.
It’s time to test the project by calling the `index.html` file, and the following screenshot is what you should get:

![Screenshot](image.png)

Although it’s just a basic project, there are several things to take note of:

- Images are preloaded and a default loading screen is shown. This means the preloader works.
- Although our project is set to work at 320x480, the game stretches to fill the browser completely, thanks to the resolution policy set before.
- Images have their registration point in the center of the image, whereas most frameworks have their image registration point in the upper-left corner.
- The origin (0,0) of the scene takes place in the lower-left corner, while most frameworks have their origin in the upper-left corner.

To top it all, you were able to create your first project. To change the target position and place it in the middle of the screen, just use the `setPosition` method that changes `gamescript.js` this way:

```javascript
var gameScene = cc.Scene.extend({
  onEnter:function () {
    this._super();
    var gameLayer = new game();
    gameLayer.init();
  }
});
```
Hello World – A Cross-platform Game

```javascript
var game = cc.Layer.extend({
  init:function () {
    this._super();
    var target = cc.Sprite.create("assets/target.png");
    this.addChild(target,0);
    target.setPosition(160,240);
  }
});
```

Test the project and you will see the target image in the middle of the screen.

Removing images and changing the background color

Now you know how to add images you might also be interested in knowing how to remove them. It's really intuitive: you added images with the `addChild` method, so you are going to remove them with the `removeChild` method.

Moreover, we will change the background color by adding an actual background layer, which covers the entire scene with a solid color.

There are just a couple of lines to add to `gamescript.js`:

```javascript
var gameScene = cc.Scene.extend({
  onEnter:function () {
    this._super();
    var gameLayer = new game();
    gameLayer.init();
    this.addChild(gameLayer);
  }
});
var backgroundLayer;
var game = cc.Layer.extend({
  init:function () {
    this._super();
    backgroundLayer = cc.LayerColor.create(new cc.Color(40,40,40,255),
    320, 480);
    this.addChild(backgroundLayer);
  }
});
```
var target = cc.Sprite.create("assets/target.png");
backgroundLayer.addChild(target, 0);
target.setPosition(160, 240);
setTimeout(function() {
    backgroundLayer.removeChild(target);
}, 3000);

In the preceding code, `backgroundLayer` is a new layer that will be filled with a new color with the RGBA format (in this case, a full opaque dark grey), which will also contain the target image.

After three seconds since its creation, the target is removed from `backgroundLayer` with the `removeChild` method.

**Summary**

In this chapter, you learned how to install, configure, and run your first Cocos2d-JS project. You also learned how to place images on the screen.

Placing more instances of the same object will be one of the topics covered in the next chapter, where you will also create your first game, so no looking yet!

Test yourself with an exercise by trying to put 10 targets on the screen at random positions.
Where to buy this book

You can buy Learning Cocos2d-JS Game Development from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.