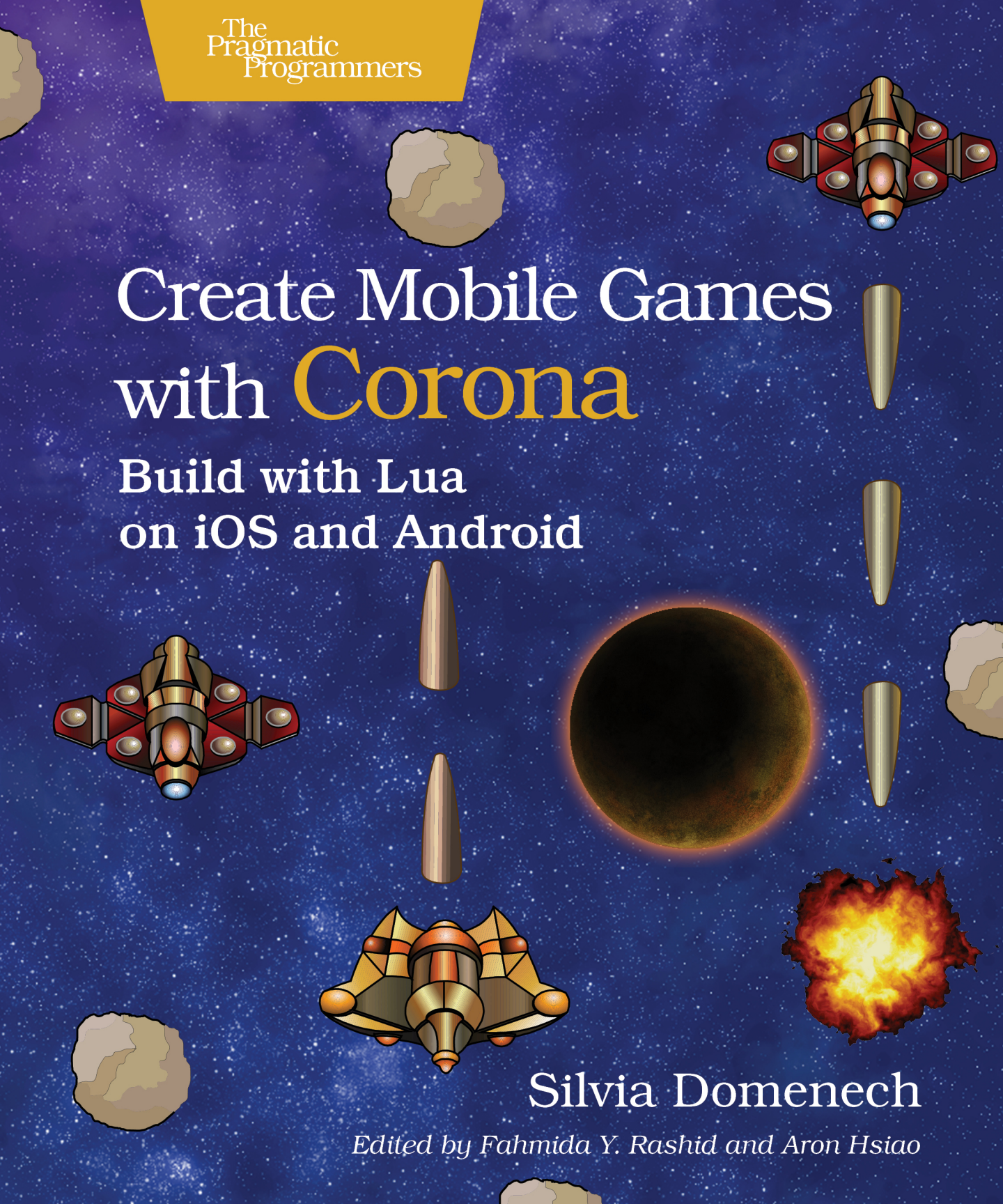
Extracted from:

Create Mobile Games with Corona

Build with Lua on iOS and Android

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina


The Pragmatic Programmers
Create Mobile Games with Corona
Build with Lua on iOS and Android
Silvia Domenech
Edited by Fahmida Y. Rashid and Aron Hsiao

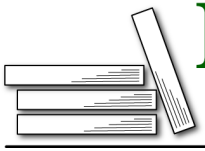
Create Mobile Games with Corona

Build with Lua on iOS and Android

Silvia Domenech

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Fahmida Y. Rashid and Aron Hsiao (editors)
Potomac Indexing, LLC (indexer)
Candace Cunningham (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-937785-57-4
Encoded using the finest acid-free high-entropy binary digits.
Book version: P2.0—January 2014

CHAPTER 1

Hello, Corona!

Greetings, and welcome to *Create Mobile Games with Corona*. In this chapter, you'll learn how to install Corona, build a really small app to ensure the game engine works properly, and learn a bit about game development along the way.

In this book, you'll start learning about Corona from scratch and build several mobile games. If you want to jump ahead, you can install Corona and start working on your first app in Chapter 2, *The Game Loop*, on page ?. Stick around if you want to learn about the various terms we'll be using as we familiarize ourselves with Corona.

1.1 A Word on Game Development

Making games is both a rational activity and a creative endeavor. What does this mean? It means that no matter how smart or skilled we may be, we can't just write an equation for `fun`, implement it in code, and end up with great, immersive gameplay. Technical skills are certainly involved, but making games also requires developers to have fun and express their creative sides. To transform ourselves into serious game developers, in other words, we'll have to imagine entirely fictional worlds, their characters, and their properties and then combine all of them with our technical skills before sharing them with others.

There is no clear scientific approach to creating a game from scratch. On the bright side, this means there's a lot of room in the game-development market for original games. By designing and programming our own games, we can have the same fun that world-building novelists have, along with the thrills that come from writing and debugging a complex program.

90 Percent Development, 90 Percent Polish

Programming games may seem similar to traditional programming, but there are some important differences. Yes, we'll write code and compile and debug programs. As game developers, however, we also have to focus on entertainment. Even if we manage to create a well-coded, well-rendered, bug-free game, players won't play it if it's not fun. On the other hand, players often take to a really fun game even if it has ugly graphics and isn't perfectly implemented behind the scenes.

To be solid game developers, we'll need to work with an eye toward making fun, playable games, not just writing great code. The best way to make sure that this happens is to enjoy what we're doing; if we're not enjoying the games as we make them, players probably won't enjoy our games, either. That's not to say that things never get tough. When coding, we'll encounter persistent bugs and intractable issues; problem solving and the implementation of workarounds are part of the programming cycle, so it's important to remember that game development doesn't end just because an initial prototype is working.

The prototype is just the beginning. For polished games, we'll have to expand our prototypes and convert them into full games and then refine them until we're happy with the results. This is different from nongame programming, in which we might focus on core functionality and bugs. This balancing, polish, and testing stage in a game's life cycle surprises novice game developers with the unexpected workload. If we're not careful, it can postpone launches by many months, so it's important to balance the "second 90 percent" of game development with a healthy dose of practicality. Once we're happy with our games and find them to be fun, it's often a good idea to launch them even if they aren't yet perfect and have bugs. We'll always be able to patch or release successors in the future. Otherwise, the polishing stage can wind up taking longer than the development stage.

Becoming a Game Developer

Many mobile games don't have a long list of contributors and specialists behind them. Since the number of developers tends to be small for mobile games, it's important to know at least something about programming if you want to make mobile games, because you'll sometimes end up doing a bit of everything, including working on the code, design, and art. You'll be in great shape if you've already used other programming languages, but don't worry if you don't have a broad set of programming skills. *Lua*, the language we use with Corona, is easy to learn, and this book will get you up to speed quickly.

Working through this book will develop the core tool set any mobile-game programmer needs.

Beyond gaining basic programming skills, prototyping is the first step in game development. As we gain proficiency with game programming in this book, we'll prototype several games with mechanics that can be used for many future game projects. It's always tempting to polish graphics and gameplay at this point, but our key goal for the prototyping stage will be to create games that work and can be played. The prototype tells us whether a game will ever be fun and identifies key changes that need to be made in subsequent development stages.

Graphics come after the prototyping stage. After nailing a fun prototype, it's usually time for us to come up with great graphics. On large development teams, there's often an artist in the group, but there are alternatives to this model. There are lots of free and paid resources for game graphics; Appendix 1, *Corona Resources*, on page ?, lists some of these, and we'll use them in various places in our examples. In this book, we won't let a lack of artistic ability prevent us from becoming successful mobile-game developers.

The testing stage follows everything else. We all have friends and family or know dedicated gamers who can play and provide feedback about a game. Paid testers are sometimes an option, but they certainly aren't necessary. The key difference between developing traditional software and games is in this stage. We want to fix gameplay or user-experience issues but not get bogged down trying to uncover or address every possible minor bug or gameplay balance issue. These issues are typically addressed using updates and patches after releasing the game.

After designing, coding, and testing games, we'll release them, instantly becoming mobile-game developers! Games that have a fun underlying concept and work reasonably well will have a lot of gameplays (also called *app sessions*), and the number will increase quickly. After a brief period to rest and regain energy, we can begin the development process again with a new idea for a game.

1.2 Getting Started with Corona

The Corona SDK is different from other programming environments. Instead of a workspace and an embedded debugging system, we'll be using a basic text editor to write code, and a graphics editor to make images. Corona will just be in charge of compiling and executing our games. To get started, we'll need both the Corona application programming interface (API) and a decent

text editor. Over the next few pages, we'll cover the download, installation, and basic usage of Corona, and then we'll discuss some of the common text editors available to you, in case you don't already have a preferred one installed.

Downloading, Installing, and Meeting Corona

Feel free to use the starter edition of Corona as we work through this book. It's free to download and use, and doing so won't limit your ability to follow along with the early projects. You'll be unable to add in-app purchases to your games, but you'll be fine with the rest of the book's projects, though.

To download Corona, visit the Corona SDK website and click the Download link.[1] You'll have to create an account. There's no workaround for this, so an email address and password are needed. Use something memorable because these details will be needed to run the Corona program. After registering, you'll see the page to download the program.

Installing Corona is user-friendly. Open your downloaded file and follow the instructions (click Next most of the time). Once Corona is installed, you'll be prompted for the username and password you created a moment ago.

Once Corona has been installed, we're ready to begin. If you're using Windows, open the Corona simulator. If you're on Mac OS, open the Corona folder in the Applications folder and then open the Corona simulator to open Corona and reach the landing screen. If you want to see the console, open the Corona terminal instead.

The landing screen Corona displays at launch is intuitive. We can create a new project, run a local project in the simulator, monitor a dashboard with our games' statistics (although this feature is only for subscribers), or experiment with the array of demos that ship with Corona. There is also a set of links pointing us to the Corona community and documentation.[2] These links are useful if you want to meet fellow Corona developers or learn about a specific function.

Creating a New Project

Let's create a new project in Corona, just to familiarize ourselves with the development process and the choices available. As you can see in the following screenshot, we're asked to enter an application name and to select the folder

1. Corona Labs: http://www.coronalabs.com/products/corona-sdk/
2. http://www.coronalabs.com

where we want to create our application. Then Corona offers us a choice of templates. We can choose from several options (see the following figure).

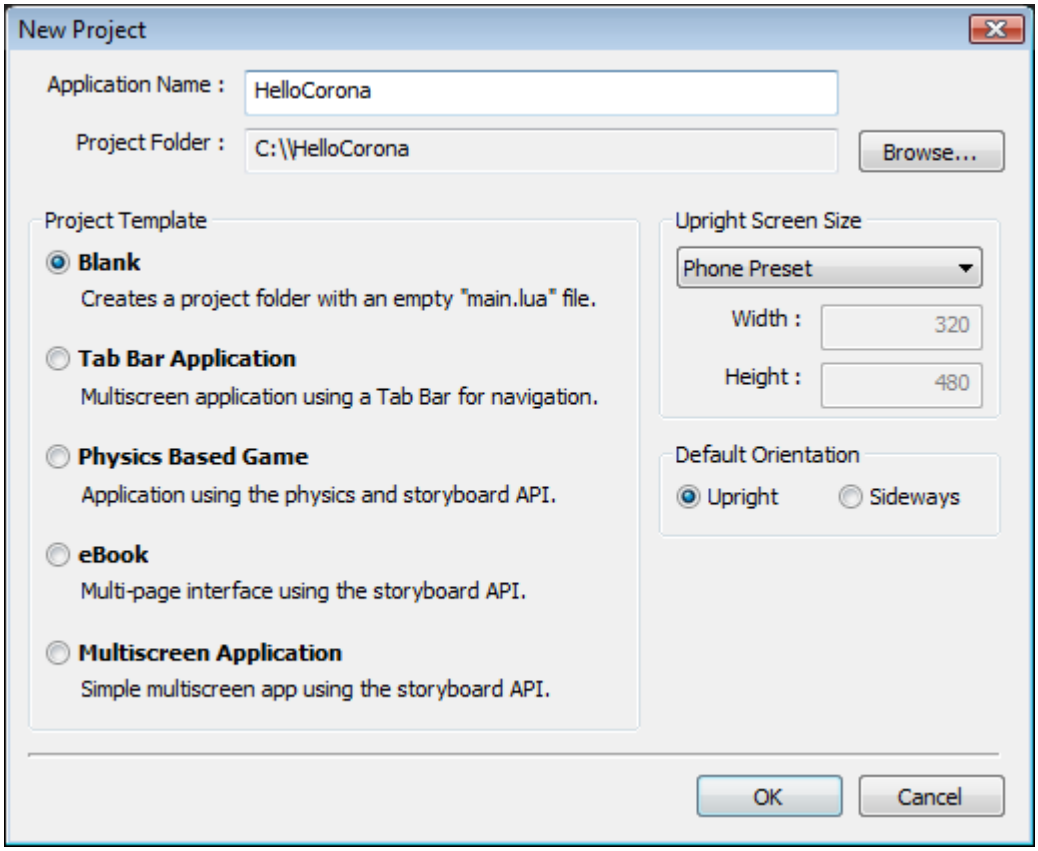


Figure 2—New project creation

- A Blank project template that provides an empty main.lua file. This is the template to use when writing an app from scratch. This can be a good choice for some single-screen games, but it is extra work for anything more complicated, meaning that we won't use it in this book.
- A Tab Bar Application template (called *App* on Mac OS) for creating apps with standard menu bars. Because we're focused on game development, we'll use our own custom menus most of the time, so we won't get much mileage out of this option either.
- A Physics Based Game template (called *Game* on Mac OS) for apps that leverage Corona's Physics API. This option gives us a basic app with physics support by default (though it's also possible to add Physics API support to

projects that didn't start with this template). It also comes with storyboard support (like the multiscreen application) to simplify switching game scenes.

- An eBook template for ebook apps. We won't usually need these when making games.
- A Multiscreen Application template (called *Scene* on Mac OS) for apps that involve multiple views. We'll use this template throughout this book since it gives us an easy way to divide our games into different scenes (menus, levels, and end-game screens).

Though we could go ahead and finish creating a project now, let's back up for the moment and take a quick look at the Corona simulator that will make our work easier as we develop games throughout the book.

The Simulator

The Corona simulator looks just like a mobile device, and you can even change its appearance to match that of several devices on the market. If we develop an iPad game, for example, the simulator can show us the game as it would appear on the larger screen. When we open the simulator, it asks us to choose the project being simulated and starts it using the appropriate device.

As we develop, the console output screen will often be an important development tool. It tends to hide behind the simulator. To see it, move windows around so that it's visible. We'll use the console output screen to monitor any runtime errors. These will happen on a regular basis, particularly as we run new chunks of code for the first time.

The Demos

Corona also comes bundled with an assortment of premade demonstration programs, each of which showcases one or two important features. As we work through the examples in this book, you'll gain confidence to be able to rip these apart and learn from additional examples. Some of the demos are great apps in their own rights and offer gameplay mechanics similar to those in some popular games. To access the sample projects, click on "Help" in the top menu.

Getting Official Corona Documentation

Like most programming environments, Corona offers extensive documentation. Corona's docs are wonderful learning tools because they usually offer detailed examples. Because the official docs are complete, you can use the official Corona SDK documentation to solve specific problems or fill knowledge gaps after we've finished working through this book.

Access the official Corona docs on the Corona website at www.coronalabs.com.

Choosing a Text Editor or Environment

If you don't already have a favorite text editor, you should consider picking one that highlights Lua code and shows line numbers. Any text editor will do as long as you can save Corona files with the proper extension (.lua). We'll spend most of our development time typing away in a text editor, so it's important that you choose one that feels comfortable to you. There are lots of options for programmers to choose from, but the following are three of the most popular free applications. In case you'd rather use a paid resource designed specifically for Corona, check out the ones listed in Appendix 1, *Corona Resources*, on page ?.

Eclipse

Platform: Mac OS or Windows

Eclipse is a great coding environment for many languages, and it also offers the option to make coding in Lua a lot easier because it highlights variables and color syntax as you code. Because of Lua's dynamic nature, spelling mistakes can lead to a lot of trouble during development, so a text editor that highlights variables can help reduce the number of frustrating mistakes.

Download Eclipse from http://www.eclipse.org/ and the Lua development tools from http://www.eclipse.org/koneki/ldt/.

Notepad++

Platform: Windows

If you're more fond of Notepad-style editors, choose Notepad++. It offers syntax highlighting and line numbers so that you can concentrate on coding rather than text details. It's also lightweight and launches quickly. It's not a development environment in the sense of Eclipse, but I use it to code 90 percent of my Lua programs.

Download Notepad++ from http://notepad-plus-plus.org/.

CodeMAX

Platform: Mac OS or Windows

CodeMAX is hosted at Lua Forge (meaning that it has a close relationship with the Lua community) and comes with syntax highlighting, code completion, and multilanguage support. It's similar to Notepad++, so making a choice between these two is really about aesthetics or selecting the tool layout that seems more intuitive to you.

Download CodeMAX from http://codemax.luaforge.net/.

1.3 Building Our First App

To get the ball rolling and test the Corona simulator at the same time, let's create a simple program that will draw ten rectangles on the screen. Not everything will make sense at this point, but you'll come to understand everything we do here over the next several chapters.

A Test Project

Open the Corona simulator, make a new blank project, and call it something like HelloCorona. This results in a file called main.lua that will hold our code. Navigate to the folder where you decided to create the project, and open the resulting file with your favorite text editor. From now on, we'll use the text editor to type our code, and each time we want to run the program we'll go back to the simulator and open it. You'll have to close the project in the Corona simulator if you want to make changes to some of the images and resources. Otherwise, they're considered "open" by the simulator, and you won't be able to update them.

The main.lua file that we'll be editing has some generated text already. You can either write immediately after it or replace it; both ways will work.

We want to draw some rectangles on the screen, so let's start writing in main.lua. We draw rectangles with Corona's display.newRect() function, so you would enter display.newRect(leftBorder, topBorder, width, height) to create some rectangle-drawing code. We could write something like this ten times to draw ten rectangles, but it's easier to use a loop. Enter for i = 1, 10 do before our rectangle-drawing code. If you've previously coded in other languages, this should be instantly familiar. If you haven't, don't worry, because you'll learn about loops in the next chapter. For now, let's just take a look at the code we've entered.

HelloCorona/main.lua

```
-- Draw 10 rectangles:
for i = 1, 10 do
    -- Draw a new rectangle at a random position,
    --    and with a random size and color
    local newRectangle = display.newRect( math.random(320),
        math.random(480), 10 + math.random(100), 10 + math.random(100) )
    newRectangle:setFillColor( math.random(), math.random(),
        math.random() )
end
```

This code will create ten random rectangles with different sizes and colors.

Testing Our App

Corona makes it really easy to compile and test apps. We don't need to download anything to an actual mobile device; to run our app, we just have to open it from the Corona simulator's menu. Apps in the simulator run just like on mobile devices. The only difference is that instead of tapping the screen with your finger, you'll use your computer's mouse.

Click the Simulate option on the simulator's starting screen, and navigate to the folder containing the app. To run an app, you just have to open its main code file, called main.lua. If there are several code files in a project, Corona will load them when they're required, so there's no need to open them explicitly in the simulator. Take a look at the following image to see what the simulator looks like.

Figure 3—The Corona simulator

If the app runs without errors, we know that Corona is installed correctly, and we're ready to get to work on other projects. If you face any issues compiling this project, double-check the spelling or take a look at the chapter's code files and try to run them. If you can't compile them either, then you may be facing post-installation issues in Corona. If that is the case, check your installation.

The Stage

If the code worked for you, you already have a running project. It's time to introduce a fundamental concept in Corona development: the stage. As we develop our games in the coming chapters, we'll draw a lot of things on the screen. In Corona, the device's screen is part of a graphical area called the *stage*. Think of the Corona stage the same way you'd think of a theater stage: anything that you add to the stage is intended for the audience (the player).

As is the case in theater, however, some things on the stage (say, at the end, in the wings) may be present and ready for use but not yet be visible to the audience. In Corona, we can do the same thing—draw things on the stage but outside the visible area of the screen. Corona automatically manages the stage and knows which things are on the visible part of the stage and which things are onstage but not yet visible. In this book (and in many other game-development contexts), we call the visible area of the device's screen the *screen bounds*. From now on we'll say that an object is "out of bounds" when it's on the stage but not yet visible to the player (not within the screen bounds).

We've just drawn our first ten rectangles on the stage (and within the screen bounds), as shown in the following image. In the coming chapters, we'll draw much, much more.

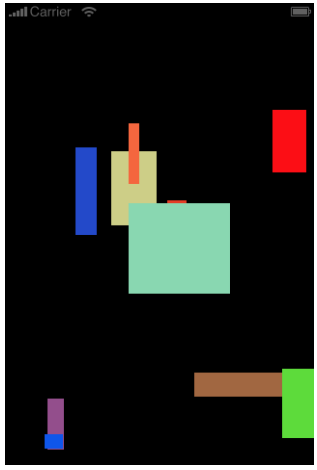


Figure 4—Our first app

Debugging

If the code in the test project didn't produce the expected result, it's time to check the console for error messages. Many other development environments

stop everything whenever incorrect code is encountered. The Corona simulator usually keeps going, so errors and issues go undetected unless we monitor the console while running the project in the simulator. The console is where Corona prints all its error messages. Most of the errors are pretty straightforward English and may reference the specific function or command that is causing trouble. As a result, it's generally easy to tell by looking at an error what type of issue may be causing the problem.

Debugging just requires you to update your code and then run it again in the simulator. Sometimes it works after an easy change, but sometimes it doesn't. Luckily, as as a Corona developer, you'll soon learn to use the console to send yourself test messages while the code is running—everything from in-game data to "Everything is fine if we reached this far" messages—using the `print` function. You can also use the Debugger tool that comes with Corona and that can be used from some development environments like those listed Appendix 1, *Corona Resources*, on page ?.

1.4 What We Covered

Whew! We finished our first chapter. You learned a little about game development in general, installed Corona, ensured that it works, and even started some coding. At this point, you're ready to tackle games.

Not so sure? Feeling a bit overwhelmed? Don't worry. Coding games is easy once you get rolling, but everyone feels intimidated at the beginning. You'll get much more comfortable as we work through some mini-programs in the next few chapters. In *The Game Loop*, we'll start building something called a *game loop*, which we'll turn into a full-fledged game over the two chapters that follow.

And don't forget, the best part of making games is to have fun, so don't let yourself feel stressed. Enjoy the journey!