GP Animations

Creating Mobile Apps with Corona SDK

Written and Illustrated by Greg Pugh

Tutorials for Corona SDK from GPAnimationsBlog.com

GP Animations

presents

Creating Mobile Apps

with

Corona SDK

Written & Illustrated by Greg Pugh

Chapter 1

Is this book for you?

I wrote this book in hopes of inspiring more people to dive into the world of mobile application development. At first glance, developing a mobile app can seem like quite a daunting task, but with the right software and a little patience, anyone can create an app.

In this book, we'll cover how to develop apps from scratch using Corona SDK and third-party tools such as Kwik 2 Photoshop plug-in, TexturePacker, and Animo. We'll also briefly discuss recording sound effects and creating artwork to give you the total package of app development.

If this sounds fun and exciting to you, then please continue reading and get ready for an adventure! If it still sounds scary, take a deep breath and I'll walk you through it.

Section 1

Introduction

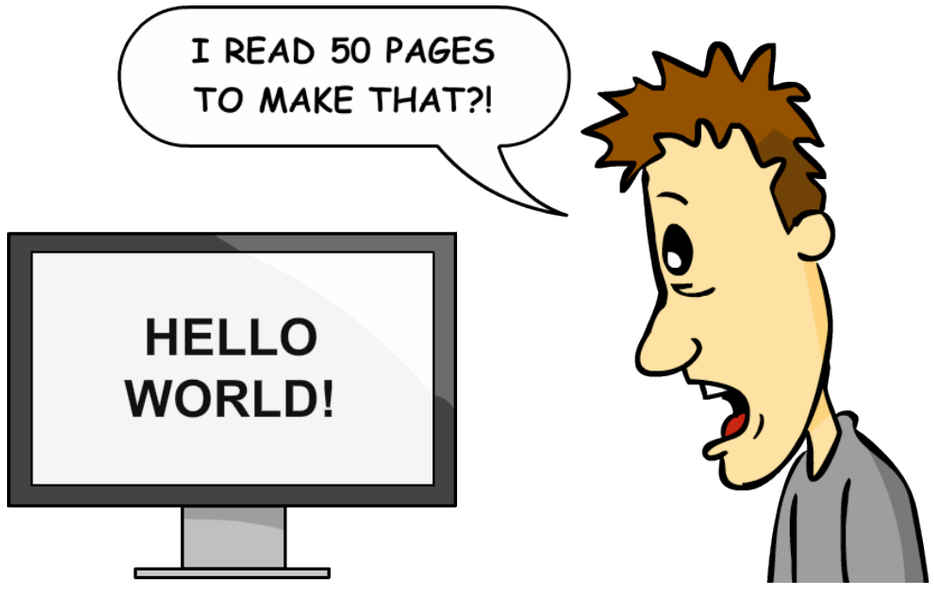
I've probably purchased 50 or more "How to Code" books in my lifetime. Out of those 50, I've finished 0 and I've fallen asleep during many of them. The problem with typical coding books is that they're just too boring for me. They explain the history of the coding language, how it's evolved, and explain everything in technical jargon before they let you make "Hello World" appear on the screen. By that point, I've already lost interest and donated the book to the Salvation Army.

When I read a coding book, I just want to make an app so I can learn from experience. It's difficult for me to imagine what lines of code will do, I want to see them in action. For this reason, I don't want this to be a reference book of Lua code or an encyclopedia of frameworks. This book is intended to give someone with little to no knowledge of Corona SDK the chance to create simple apps to become familiar with mobile application development.

Think of this book as being a goldfish in a plastic bag. I'm just placing the bag in an aquarium to acclimate you to the app world. If you decide you want to live and swim in that world permanently, you can move on to bigger and better books.

I'll also briefly cover creating artwork, sprite sheets, and recording audio, since graphics and sounds are used for the majority of the apps on the market.

With that being said, I'm going to take the next few pages to cover the software that we'll be using as well as other options available. I'll try not to bore you with the details of each piece of software, as there's plenty of documentation on the companies' websites. I just want to give you some options of the tools you can use for your apps.



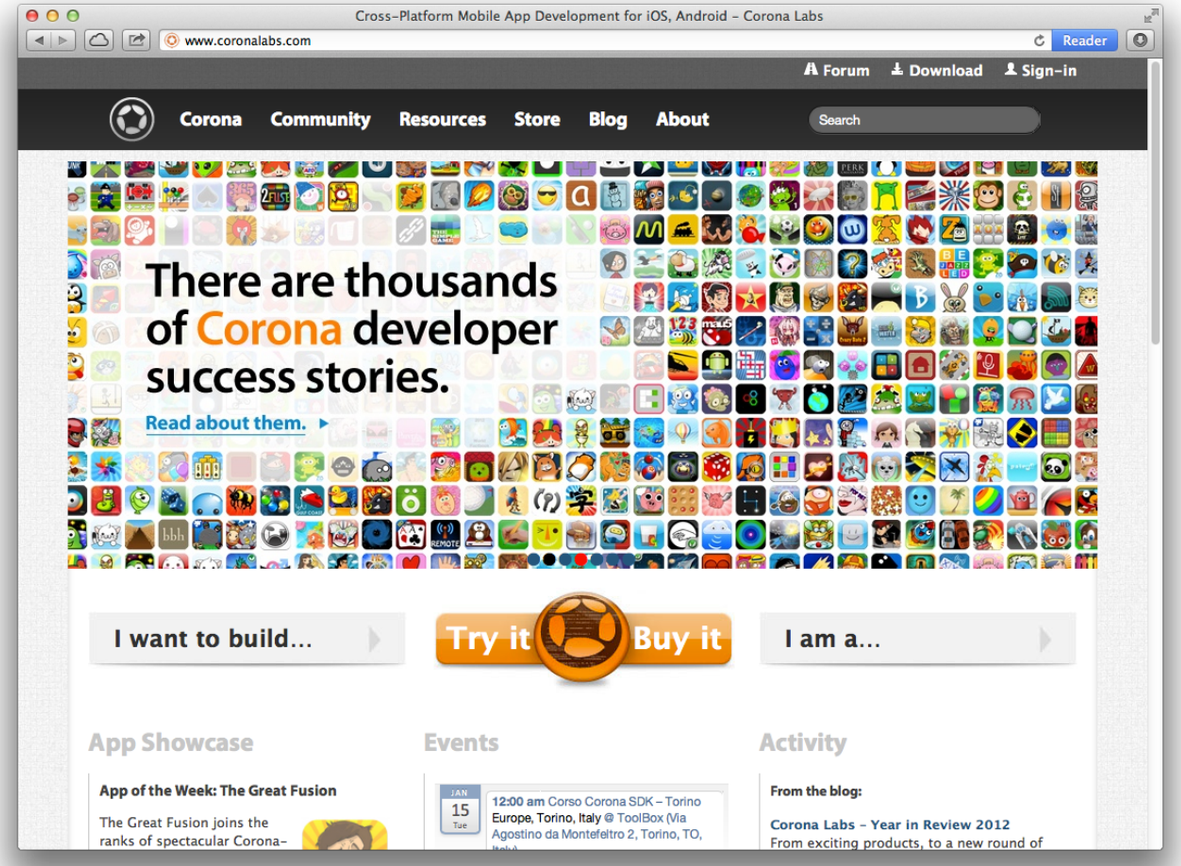
Section 2

What Software Do I Need?

The main piece of software we'll be using is **Corona SDK**. On your computer, head over to *http://www.coronalabs.com* and sign up for a free trial if you're not already a Corona user. The free trial is unlimited and you only have to choose a subscription when you're ready to test on your device and publish to an app store.

You have to download and install the Corona SDK simulator, which is where you'll be testing all of your apps. You don't actually use Corona SDK to write your code, it simply takes what you write and converts it into a visual display of graphics and sounds. It also takes your code and converts it to the file types needed by Amazon, Apple, Barnes and Noble, and Google to host on their app stores. We're not going to worry about that now though, we just need to fill your toolbox with the right tools.

The software you'll use to actually write the code can be found on your computer right now. If you're a PC user, you can use **Notepad** and if you're a Mac user, you can use **TextEdit**. With the Corona SDK free trial installed on your machine, you essentially have everything needed to create an app and it hasn't cost you a penny.



www.CoronaLabs.com

There are paid and other free options available such as **TextWrangler**, **Notepad++**, **TextMate**, and **Sublime Text** that offer some additional features. However, it should be noted that

you ***do not*** need a paid text editing program to create apps. Feel free to test out other text editors, but don't feel like you need to use anything besides good old Notepad or TextEdit. Corona SDK uses Lua code, which like HTML, doesn't care which text editor you use, just as long as you save it as the correct file type.

One of my favorite pieces of software to create Corona SDK apps is the **Kwik 2** Photoshop plugin (*http://www.kwiksher.com*). Kwik allows you to use Adobe Photoshop to create apps without having to write code. In fact, I originally created my first children's iPad book app using the traditional method of Objective-C, but when I discovered Kwik, I remade it in a fraction of the time using Kwik. Like most pieces of software, Kwik offers a free trial, so if you're a Photoshop user like myself, it's definitely worth checking out. Since this is a "How to Code" book, I won't cover Kwik too much, but I want to make you aware that it exists and is a great tool for Photoshop users.

If your app is going to rely on animated graphics, you'll probably end up creating what are called "sprite sheets." Sprite sheets are images compiled into a single file that software can recognize and piece together into an animation. There are numerous software packages that create sprite sheets from your artwork, a few I like to use are **TexturePacker** (*http://www.codeandweb.com/texturepacker*), **SpriteHelper** (*http://www.gamedevhelper.com/spritehelper/*), and **Animo** *(http://lanica.co/about/animo/*). Each has its strongpoints and weakness so I encourage you to download their free trials and see which one you like the best.

If you're going to want to record your own sound effects and music, you don't need anything as expensive as Pro Tools, you can use free software like **Sound Recorder** on PC, **GarageBand** on Macintosh or even **Audacity** (*http://audacity.sourceforge.net*), which is free and available for both PC and Mac.

I know reading through this list of software is a lot, but I just wanted to make you aware of the options available to you if you decide to become an app developer. For right now, let's just stick with Corona SDK and any text editor you happen to have on your computer.

Chapter 2

Your First App

You've got your text editor and Corona SDK installed and ready to go. Let's get this party started and make an app!

Section 1

Your First App

Now it's time to build your first Corona SDK application. We'll keep it simple, but more complex than just displaying the words "Hello World!" on the screen. We'll build an iPhone spaceship app where the user can move the spaceship around the screen by pressing on directional arrows. Tap on the image below to watch a movie of the app in action.

I think this is a good starting point because it teaches you many valuable lessons for real-world applications. It covers changing the default phone orientation to landscape mode, hiding the phone's status bar, displaying images, and controlling images by pressing on buttons. These are all things that many game apps have in common and they'll be the first things you learn here.

I've already created the artwork for you so you can focus on the coding for now. You can download all the source files for this book here:

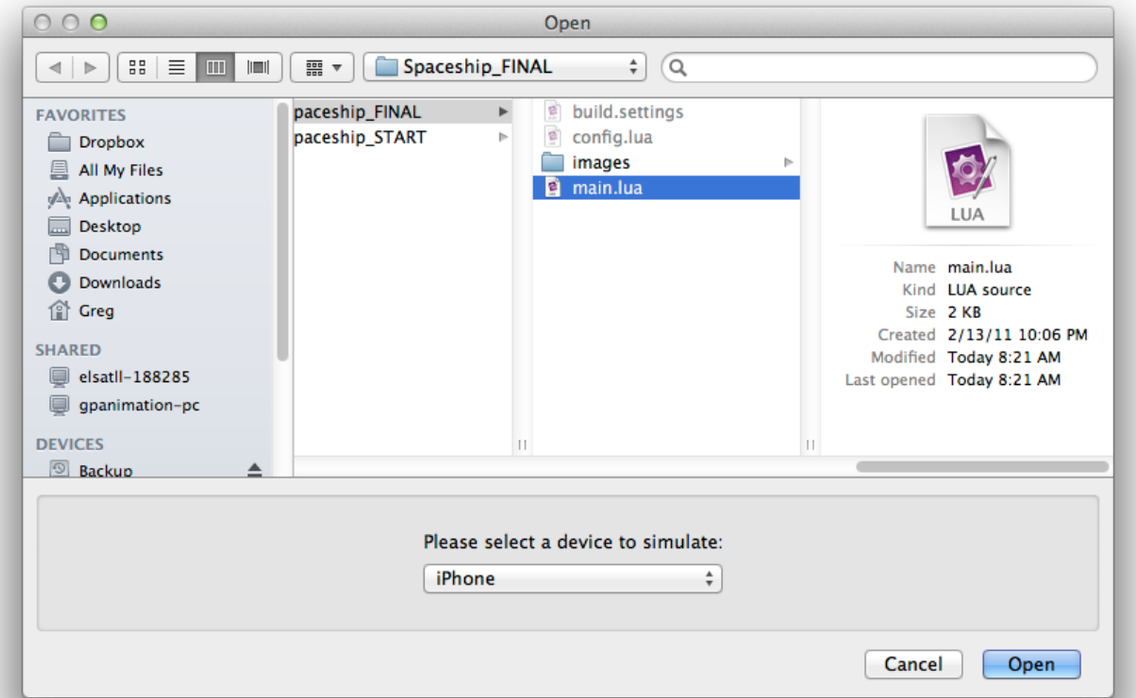
http://www.GPAnimations.com/chapter1.zip

It should be noted that you have to use that link on a computer, if you're reading this on your iPad, you won't be able to unzip it.

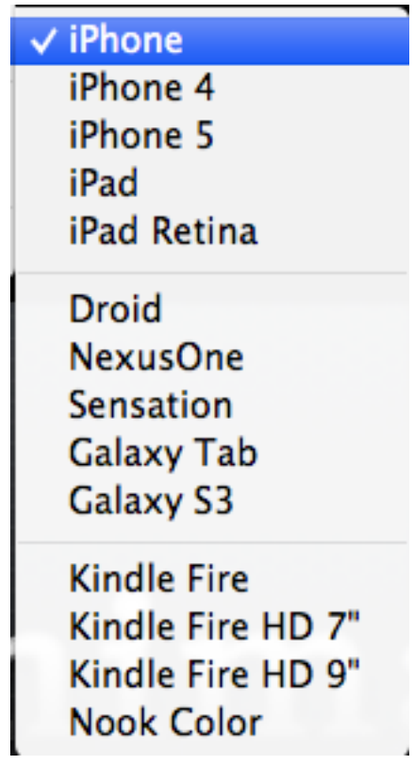
When you download the *chapter1.zip* file, you can unzip it and see the folders of the projects we'll be making in this book. Right now you're interested in the **Spaceship** folder, which contains **Spaceship_START** and **Spaceship_FINAL**. We'll be using Spaceship_START, but if you want to quickly see the final product, you can open Spaceship_FINAL.

To open the final product, start the Corona SDK Simulator, which will be in the Corona SDK folder in your Applications folder if you're on a Mac or in your Start bar menu if you're on a PC.

Once the Corona SDK Simulator is running, go to **File > Open** and browse for your **Spaceship_FINAL** folder and open **main.lua**. Note that the Corona Simulator can only open files named *main.lua*, but more on that later.



Now you should now see your Spaceship game in action! If it looks a little blurry and is running on a different device than an iPhone, that's not a big deal. In the Corona simulator, navigate to **Window > View As > iPhone**. This will also give you an opportunity to see all of the other devices that the Corona SDK supports.



To the right, you can see all of the supported devices at the time of this writing, which is using *Corona SDK daily build version 2013.999*. Once you are a Corona Subscriber, you have access to Daily Builds, which are regular updates to the Corona Simulator that fix reported bugs and issues.

However, daily builds may introduce new bugs, so you might feel more comfortable using the latest stable release. Regardless of what version you're using, by the time of this writing, your supported device list should look very similar to the image here.

Building Your App from Stage 1

Now that you see firsthand what you'll be creating, let's start from scratch. Close the Corona Simulator for now and open your text editor of choice, whether it be Notepad, TextEdit, or one that I mentioned earlier.

Start a new document and save it as **config.lua** in your **Spaceship_START** folder (*be sure to save it with .lua, not .txt as is the common default for many text editors*). Type the following text into **config.lua** and save:

```
application =
{
	content =
	{
		width = 320,
		height = 480,
		scale = "letterbox",
		fps = 60,
	},
}
```

All you've done here is set up some parameters for the game. You set the initial width and height, which is the size of an iPhone screen (320x480). When scaled up to other devices, it will "letterbox", or add black bars to the sides to compensate for different screen dimensions. The app will run at 60 frames-per-second, which will make the spaceship appear as it's moving very smoothly across the screen.

Close your config.lua file and start a new document in your text editor. This time, save it as **build.settings** in your **Spaceship_START folder**, being sure to use *.settings* instead of *.txt*. Now type the following code and save your file:

```
settings =
{
	orientation =
	{
		default = "landscapeRight",
		supported =
			{
				"landscapeRight", "landscapeLeft"
			},
	},
}
```

With this chunk of code, you've told the app that you want it to open and run in landscape mode instead of the traditional portrait mode. Now close *build.settings* and start a new document. Name the document **main.lua** and save it in the **Spaceship_START folder**.

Main.lua files are the only files the Corona Simulator will open. However, it understands to look for config.lua and build.settings files when it opens main.lua. So it's as if it says to itself "*Is there any certain way I should run this main.lua file? Oh yes, open it as the config and build files tell me.*" In our case, it's going to want to

open *main.lua* in landscape mode with content at 60 frames per second, 320x480 dimensions, and in letterbox format.

Open the Corona Simulator again, but this time when you navigate to **File > Open**, choose the *main.lua* file in the **Spaceship_START folder**. Now you'll see your super-exciting app in all its glory.

Yes, I know it's far from spectacular, but there is a lot happening. Usually the simulator would open in portrait mode, but thanks to you, it opens in landscape mode. It's also ready to display content to the parameters you specified in your *config.lua* file, we just can't see it yet since there's nothing on the screen.

What's the Status?

The first thing you can do is hide the status bar that shows the battery life, time, service provider, etc. It's not going to be needed for this app, so we might as well get rid of it. The beauty of Lua code is that it's one of the most simple of the programming languages. You can generally type things in terms you'll understand as you read through your code. Type the following into your **main.lua** file and save:

```
-- Hide the status bar
display.setStatusBar(display.HiddenStatusBar);
```

When you saved your file, the simulator most likely refreshed automatically and removed the status bar. If it didn't, select the simulator and press *Ctrl+R* or *Cmd+R* to manually refresh it.

The first line of code you typed with the double dashes (--) is a comment code. The simulator won't read these lines, they're just helpful to you. I recommend using comment code a lot, as it will be a great help when you come back to a project at a later date. Plus, if you're working collaboratively, your colleagues will appreciate you saying what each chunk of code does. Now you know that the line of code to follow hides the status bar.

The second line refers to the status bar on the display, and tells that status bar to be hidden. That's all there is to it. Simple and straightforward, just how I like my code, which is why I love Lua code and Corona SDK.

Varying Variables

Now it's time to set up a couple variables. We'll be referencing the center of the screen a couple times in the app, so setting variables with those coordinates will be easier than writing them out each time. Type the following into your *main.lua* file and save:

```
-- Variables to Determine Center of Screen
_W = display.contentWidth / 2;
_H = display.contentHeight / 2;
```

Here we set _W to be equal to the width of the display divided by 2. In other words, _W is equal to the horizontal middle of the screen. We also set _H equal to the vertical middle of the screen.

Now you'll see why we created _W and _H. Under the code you've just typed, type the following into *main.lua* and save:

```
-- Place background image
local background = display.newImage ("images/
background.png");

background.x = _W;
background.y = _H;
```

Here we placed the *background.png* file that is contained in the *images folder* inside *Spaceship_START* on the screen. We positioned it in the center of the screen using our newly created _W and _H variables. An alternative method of using _W and _H is:

```
background.x = display.contentWidth / 2;
background.y = display.contentHeight / 2;
```

However, that would get tiresome after awhile, so it's easier and shorter to use _W and _H. Now your simulator should look like this:

Adding the Spaceship and Directional Pad

Now that you know how to make images appear on the screen, let's add the spaceship image to the center of our app. Type the following into *main.lua* after the background code and save:

```
-- Place ship image
local ship = display.newImage ("images/spaceship.png");
ship.x = _W;
ship.y = _H;
```

Your simulator will now look like this:

You can also set images to be at exact coordinates that you specify. X coordinates refer to horizontal placements and Y coordinates refer to vertical placements. As an example, we'll place the arrow keys at precise X and Y coordinates. Under the spaceship code, type the following into *main.lua* and save:

```
-- Place arrows on screen
local upArrow = display.newImage ("images/upArrow.png");
upArrow.x = 70;
upArrow.y = 230;

local downArrow = display.newImage ("images/downArrow.png");
downArrow.x = 70;
downArrow.y = 290;

local leftArrow = display.newImage ("images/leftArrow.png");
leftArrow.x = 30;
leftArrow.y = 260;

local rightArrow = display.newImage ("images/rightArrow.png");
rightArrow.x = 110;
rightArrow.y = 260;
```

Your simulator should now contain all of the artwork needed for the app.

You'll notice for all of the images that we're placing on the screen, we first say "local" then create a name for the object. Not declaring "local" defaults it to "global". Declaring local variables speeds up the app and is generally good practice if your variables don't need to share data. Think of it this way, would it be faster to find someone if you were told their location was somewhere on the planet Earth, or if you were told their location was in Atlanta, Georgia on Peachtree Avenue? It's also good practice to name your variables something that makes sense, so try to avoid naming a spaceship variable something like "woodenDesk" to avoid confusion.

Let's Get a Move On!

Now comes the tricky and fun part, actually make the app have interactions and movement. It might get a little confusing, but don't worry, I'm not going to make you memorize every detail of why the code does what it does. Type the following into *main.lua* and save:

```
-- Set up speed
local shipMoveX = 0;
local shipMoveY = 0;
local speed = 6;

local function stopShip (event)
if event.phase =="ended" then
        shipMoveX = 0;
        shipMoveY = 0;
        end
end

-- When no arrow is touched, stop movement
Runtime:addEventListener("touch", stopShip);
```

You won't notice any difference in the simulator so don't worry if it looks and acts the same as before. Here we set up some initial variables called shipMoveX, shipMoveY and speed. As you can imagine, shipMoveX has to do with the ship's movement along the X coordinate, shipMoveY is its movement on the Y coordinate and speed is how fast it moves.

In the next chunk of code, you set up your first function. Functions carry out a specific task, which in this case, stops the spaceship from moving. It's good practice to name functions accordingly just like variable names, so we named ours *stopShip*. The function tells our app that if the user isn't touching the arrow buttons then the spaceship should stop moving.

Since we have a function that stops the ship from moving, we should have one that starts the movement as well. Let's call this function *moveShip*. In your *main.lua* file, type the following code and save:

```
-- Move the spaceship
local function moveShip (event)
	ship.x = ship.x + shipMoveX;
	ship.y = ship.y + shipMoveY;
end

-- Start movement
Runtime:addEventListener("enterFrame", moveShip);
```

Wait a second, the simulator refreshed, but the spaceship still doesn't move when you press the arrow keys, what gives? That's because all we've done is set up the functionality of the buttons, we haven't actually applied them to anything yet. This new function just tells the app to move the spaceship along the X and Y coordinates, it doesn't say when to do it.

Now let's make the spaceship respond to arrow button touches. Type the following code into your *main.lua* file and save:

```
function upArrow:touch()
	shipMoveX = 0;
	shipMoveY = -speed;
end
upArrow:addEventListener("touch", up);

function downArrow:touch()
	shipMoveX = 0;
	shipMoveY = speed;
end
downArrow:addEventListener("touch", down);

function leftArrow:touch()
	shipMoveX = -speed;
	shipMoveY = 0;
end
leftArrow:addEventListener("touch",left);

function rightArrow:touch()
	shipMoveX = speed;
	shipMoveY = 0;
end
rightArrow:addEventListener("touch",right);
```

Now your spaceship flies around the screen on command! Let's break down one of the functions so we can see how it works.

function upArrow:touch()
Here we create a new function named upArrow and prepare it for a touch event.

shipMoveX = 0;
Since the up arrow won't move the ship side-to-side, the X coordinate can be set to 0.

shipMoveY = -speed
The up arrow makes the ship move on the Y coordinate and we want it to travel up. In the simulator, to make an object move up,

you give it a negative value, hence the reason for -speed. If you remember, we set speed to equal 6, this number can be changed to anything you desire.

```
upArrow:addEventListener("touch", up)
```

This is where we actually apply the function to the upArrow button.

If you've been playing around with your new app, you may have noticed that the spaceship is able to fly off the screen, that might be a problem!

What we can do to solve this is build "fake" walls. I say fake because we're not actually going to program walls to contain the spaceship, we're just going to tell the app to keep it on the screen based on X and Y coordinates. You could very well program walls and physics, but that will be in a later project. Type the following code into *main.lua* and save:

```
-- Keep spaceship boxed in by fake walls
local function fakeWalls (event)

	if ship.x < 50 then
		ship.x = 50
	end

	if ship.x > 430 then
		ship.x = 430
	end

	if ship.y > 270 then
		ship.y = 270
	end

	if ship.y < 50 then
		ship.y = 50
	end

	if speed < 0 then
		speed = 0
	end
end
Runtime:addEventListener("enterFrame", fakeWalls);
```

Congratulations, you just built your first working Corona app! Let's continue on our journey and build more apps to expand your knowledge of Corona SDK and Lua.

Thanks for reading!

This concludes the free preview of GP Animations presents Creating Mobile Apps with Corona SDK.

This preview is to gauge the amount of interest from readers like yourself. If you'd like to see the completed version on iBooks, let us know by leaving a comment on www.GPAnimationsBlog.com.