Make Games for Playdate with Lua

Brett Chalupa

Make Games for Playdate with Lua

A fun introduction to game programming with Lua and the Playdate SDK

Brett Chalupa

This book is available at https://leanpub.com/playdatebook

This version was published on 2026-05-07

Contents

Introduction

Make Games for Playdate with Lua is a comprehensive guide to creating simple games for the Playdate handheld video game console. This book is perfect for beginners. You'll code, from scratch, a bunch of small games to learn the fundamentals. We'll use the Lua programming language to write our code. If you've written some code before, you'll catch on to Lua quickly! If you haven't coded before, don't worry at all. Lua is great for beginners because of its simplicity and similarity to the English language.

Playdate's simplicity and constraints make it a great learning platform. We'll embrace its limitations in resolution, color, and input to focus on learning how to make fun games. It's truly special to be able to write code and play it on an actual handheld game console within seconds. Playdate's approachability and developer friendliness are unrivaled in the game development space.

Together we'll code games from scratch, learning key concepts and increasing the complexity with each chapter.

Feedback

If you run into any issues or have any feedback, send me an email at brett@brettchalupa.com.

What to Expect

We'll start by displaying some simple text on the screen. Then we'll code our first game—a minimal version of tennis inspired by *Pong*. From there we'll continue to code different games of increasing complexity. In the final two chapters, we'll build a turn-based RPG.

Each chapter will contain a complete project with the code from start to finish explained.

The source code for each chapter can be viewed and downloaded on GitHub at https://github.com/brettchalupa/playdatebook.

For updated links, errata, and supplemental materials, visit the book's webpage at https://pdbook.net.

Files on your operating system are referenced using `/` to separate folders. If you see a file referenced as `source/player.lua`, it means there should be a file named `player.lua` within the folder named `source`. Linux and macOS use `/` while Windows uses `\` to delineate folder paths. For the sake of simplicity, the book uses `/` throughout.

In some of the code examples, there will be `-- snip` if there's code that was removed from the example but would otherwise be in your file. It helps focus the example code and remove the clutter around what's being explained.

```
1  -- snip
2
3  playdate.graphics.drawText("Hello, Playdate", 40, 40)
```

Don't actually type in `-- snip`. It's just to show you there was code from a previous point in the chapter that is unchanged and excluded for brevity.

Getting the Most Out of the Book

The best way to get better at programming is by writing code. The more code you write, the easier it will be to implement your ideas. It takes time to get used to thinking in code, learning the language, and understanding Playdate's SDK.

I encourage you to do four things as you go through the book:

1. Type out the code yourself. Don't copy and paste it. You'll learn the language better by typing it yourself, and it'll become muscle memory.
2. Experiment! It's okay to change the code and deviate from what's in the chapter. Make what you're coding your own.
3. Back up your code. Make copies of your code before you modify it so you have backups. When things are working well, consider zipping it up or duplicating it. Sometimes when coding you can dig yourself into a hole that's difficult to get out of. While this book will not cover version control, if you've heard of Git before, it helps solve this problem by making it easy to back up and track changes to code.

4. Search online to learn more. No book can contain answers to everything. An important aspect of coding games is learning how to learn—being able to search for and apply solutions to problems you're running into. There are imperfections and room for improvement in each chapter, with suggestions for how to take the projects a step further on your own.

Getting Started

The first step is to download the Playdate SDK from the Playdate website. The Playdate SDK contains everything you need to make games for Playdate—example code, a simulator to run your games, a compiler to build your games, a manual called *Inside Playdate* with design guidelines, and more. The Playdate SDK supports Windows, macOS, and Linux.

The Playdate SDK is quite stable, so what's covered in this book should (hopefully) work with newer versions. This book was written and tested against the 3.0 Playdate SDK.

Installing the SDK

How you install the SDK depends on your operating system:

macOS: Open the `.pkg` file you downloaded and follow the installer. It'll install the SDK, the Playdate Simulator, and a command line tool called `pdc` (the Playdate Compiler) that we'll use throughout the book to build our games. The installer takes care of everything for you.

Windows: Extract the downloaded `.zip` and run the `Setup.exe` installer inside it. This installs the SDK, the Simulator, and `pdc`.

Linux: Extract the downloaded `.tar.gz`, move the SDK folder to wherever you'd like to keep it (your home directory works fine), and then run the setup script inside it:

```
1 sudo ./setup.sh
```

This installs `pdc` and the Simulator.

Setting Up the Environment Variable

When you install the SDK, the installer sets an environment variable called `PLAYDATE_SDK_PATH` that points to where the SDK lives on your computer. An environment variable is just a named value that programs on your computer can look up—think of it like a global setting. `pdc` and other tools use `PLAYDATE_SDK_PATH` to find the SDK's files, and we'll use it later to set up autocomplete in our code editor.

The macOS and Windows installers set this up for you automatically. On **Linux**, you'll need to add it yourself. Open your shell configuration file (`~/.bashrc` for Bash or `~/.zshrc` for Zsh) in a text editor and add this line, replacing the path with wherever you put the SDK:

```
export PLAYDATE_SDK_PATH=/home/yourname/PlaydateSDK
```

Save the file and restart your terminal for it to take effect. You can verify it's set by running:

```
echo $PLAYDATE_SDK_PATH
```

If it prints the path to your SDK, you're all set.

The installer also adds the SDK's `bin` folder to your system PATH, which is what lets you run `pdc` from any folder in your terminal. If you ever run `pdc` and get a "command not found" error, it means the SDK's `bin` folder isn't on your PATH. You can fix this by adding the following line to the same shell config file:

```
export PATH=$PATH:$PLAYDATE_SDK_PATH/bin
```

Your Code Editor

The only other thing you need to make games for Playdate is a text editor for writing code. Your text editor is *different* from Microsoft Word or Google Docs. It's a plain text editor that's specifically just for entering characters—your game's code.

If you don't already have a code editor, download Visual Studio Code. It's free and available for Windows, macOS, and Linux.

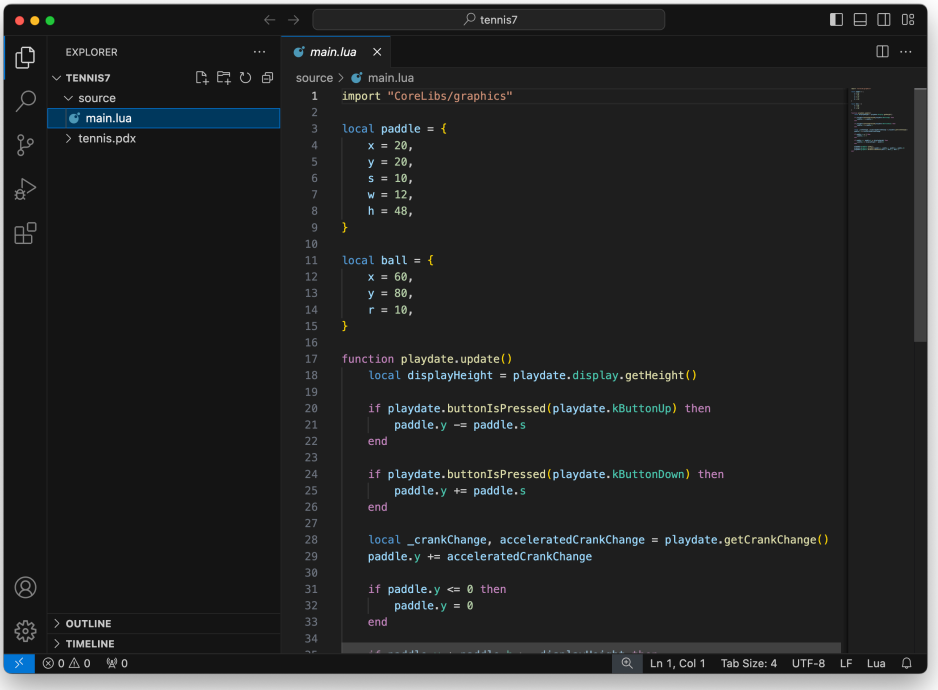

Figure 1. Screenshot of Visual Studio Code

Some other popular code editors are Zed, Sublime Text, Notepad++, and Nova (made for macOS by Panic, the creators of Playdate).

Setting Up Zed for Playdate

I use the Zed editor, a free and open source text editor. If you decide to use it, you can follow these steps to set up Lua and the Playdate SDK with autocomplete.

Setting Up Visual Studio Code for Playdate

If you're using Visual Studio Code, there's a one-time setup that'll make coding games for Playdate easier and give you autocomplete for Playdate's SDK functions. When you start typing `playdate.up` it'll suggest `playdate.update()` and so on. It's not required, but it's nice to have.

First, install the **Lua** extension by sumneko (also called LuaLS). Open Visual Studio Code, click the Extensions icon in the left sidebar (or press ⌘⇧X), search

for “Lua”, and install the one by sumneko. This gives you syntax highlighting, error checking, and autocomplete for Lua.

Next, you need to tell the Lua extension where the Playdate SDK lives so it knows about all the Playdate functions. In your project folder, create a `.vscode` folder and a `settings.json` file inside it:

```
.vscode/
  settings.json
```

Add the following to `.vscode/settings.json`:

```
{
  "Lua.workspace.library": ["${env:PLAYDATE_SDK_PATH}/CoreLibs"],
  "Lua.runtime.nonstandardSymbol": ["+=", "-=", "*=", "/="]
}
```

The first setting points the Lua extension to the Playdate SDK’s core libraries so it can provide autocomplete and documentation for Playdate functions. The second setting tells the extension that Playdate’s Lua supports shorthand operators like `+=` and `-=`, which standard Lua doesn’t have. Without it, you’ll see false error squiggles in your code.

This uses the `PLAYDATE_SDK_PATH` environment variable we set up earlier. If for some reason autocomplete isn’t working, make sure that variable is set correctly. You can also replace `${env:PLAYDATE_SDK_PATH}` with the actual path to your SDK, like `"/Users/yourname/Developer/PlaydateSDK"` on macOS or `"C:/Users/yourname/Documents/PlaydateSDK"` on Windows.

You’ll want to add a `.vscode/settings.json` to each new Playdate project you create. It’s a small step, but it makes a big difference when you’re writing code.

Next Steps

Because the Playdate SDK includes the Playdate Simulator, you don’t actually need the game console to make games for it. You’ll be able to test drive the SDK and experiment without spending any money. But I’d recommend buying a Playdate as you’ll want to test your game on it and play others’ games. The

processor of the Playdate is much slower than your computer's. That means if you write slow code, it may run fine on the simulator but be unplayable on the actual Playdate. You'll want to test early and test often on your device if you've got one.

Once you've got the Playdate SDK installed, your environment variable set, and a text editor ready to go, you're all set to start making a Playdate game!

Hello, Playdate!

A common tradition when learning programming is to output some simple text to the screen that says "Hello, World!" While it is a bit trivial, it's a great starting place because you get to familiarize yourself with some key concepts of a given language or programming toolkit. We'll learn about how to structure your games for Playdate, write some basic Lua, position text on the screen, build your game, and run it on the simulator and device.

On your computer, create a new folder called `helloPlaydate` and then a folder within it called `source`.

Then open your project folder in your text editor and create a `main.lua` file within the `source` folder.

`source/main.lua` is the primary entrypoint into your game. While larger games will use multiple files to organize everything, for our initial simple games, we'll write all of our code in `source/main.lua`.

Type the following code into `source/main.lua`:

```
1  function playdate.update()
2    playdate.graphics.drawText("Hello, Playdate!", 40, 40)
3  end
```

This is the simplest form of a Playdate game. `playdate.update()` is a special function that gets called once every frame. By default, Playdate games run at 30 frames per second (FPS). Virtually all games run in a loop, processing 30, 60, or even 120 times per second. Movement in games appears smooth to the human eye because the refresh cycle is so quick.

The game loop runs over and over and over while the game is active, handling input, updating the world, playing audio, and drawing graphics to the screen. Your game reacts to the passage of time via the update loop that runs every frame.

Any code within the function we defined gets called each frame. The code in our function says to draw the text `"Hello, Playdate!"` at the x position of 40 pixels and the y position of 40 pixels. Functions have parameters that are

separated by commas. So the first parameter is the text we want to be drawn. The second is the x coordinate where the text should be placed. And the third is the y coordinate.

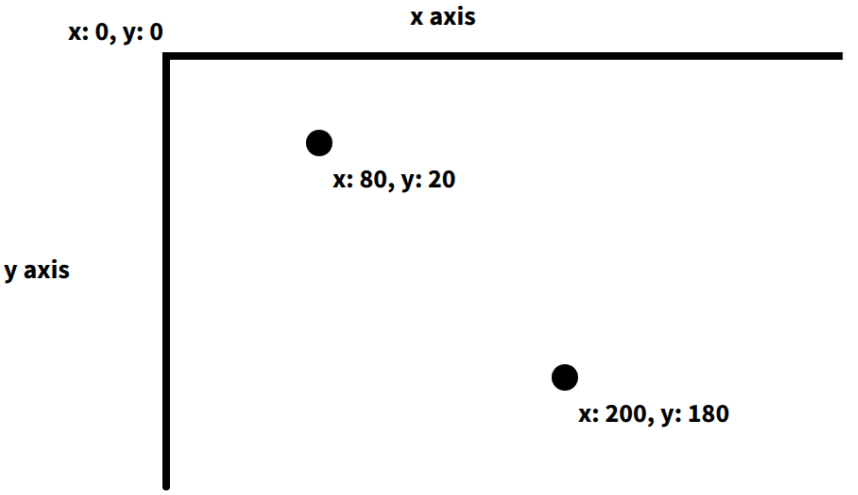

Figure 2. Diagram showing screen coordinates with the upper-left being 0, 0 for the x axis and y axis

Let's run our game, and then we'll dive more into coordinates, as they're a key part of game development.

When you installed the Playdate SDK, a command line program called `pdc` was installed. `pdc` stands for Playdate Compiler, and it's what takes your game's source code and assets and builds your game into a file that a Playdate console can play.

In Visual Studio Code, click the **Terminal** top menu dropdown and select **New Terminal**. This will open up a terminal within Visual Studio Code in your current folder. Type this command and press :

```
pdc source HelloPlaydate.pdx
```

That command says: use the Playdate Compiler (`pdc`) to build our source folder for our game and output it as a file named `HelloPlaydate.pdx`.

In Visual Studio Code's Explorer side menu, right-click on the newly created `HelloPlaydate.pdx` and reveal it in Finder (macOS), Explorer (Windows), or your file manager (Linux). Double-click `HelloPlaydate.pdx` in your file manager and the game will launch in the Playdate Simulator, rendering the text we typed in.

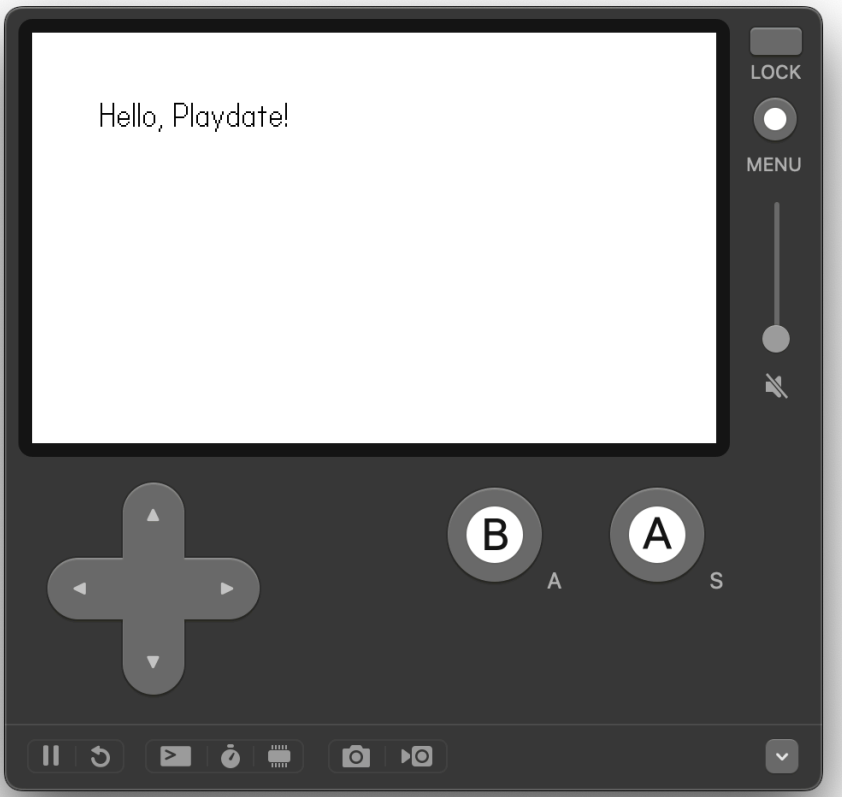

Figure 3. HelloPlaydate - Simulator Screenshot

If you've got a Playdate console, plug it into your computer and unlock it. In the Playdate Simulator, click the **Device** menu option and select **Upload Game to Device**. This will copy and run the game on your console!

Games you upload to your device are listed on the Home screen in the **Sideloaded** section. So you'll be able to play your games even when your device is not connected to your computer.

We've got our game running on the Playdate Simulator and the device in just three lines of code and one command–not bad! These fundamentals of structuring a project and running your game are essential, and the Playdate team has made it as easy as possible.

Try changing the text around and see what happens. Make your game draw `"Hello, World"` instead.

Counting Frames

Understanding the passage of time and the game loop is really important, so let's illustrate that with a simple example building upon our `helloPlaydate` game.

Update `source/main.lua` to be the following:

```
local updates = 0

function playdate.update()
  updates += 1

  playdate.graphics.clear()
  playdate.graphics.drawText("Hello, Playdate", 40, 40)
  playdate.graphics.drawText("Updates: " .. updates, 40, 60)
end
```

Compile and run the game (command: `pdc source HelloPlaydate.pdx`).

You'll see the number of updates rapidly increase. It increments by 30 every second. We created a local variable called `updates` to keep track of how many times the function `playdate.update()` is called. A variable is a programming construct that stores data we can read and change over time. A common variable in a game would be the player's health. As they take damage, their health decreases.

At the beginning of the `playdate.update()` function we add 1 to the `updates` variable. `updates += 1` is shorthand for `updates = updates + 1`. The `+=` operator takes the existing variable's value and adds the right-hand side to it.

`playdate.graphics.clear()` clears the screen. Without it, each frame overwrites the previous and the text doesn't clear from the previous screen.

This wasn't a problem in our initial `helloPlaydate` game because the text wasn't moving or changing. But because `updates` is incrementing and changing, we must clear the screen each frame.

Finally, the second `drawText` call says to draw the string `"Updates: "` and append the `updates` variable to it using the `..` operator. We draw it at the same x position so that it is aligned with the `"Hello, Playdate"` text but draw it 20 pixels lower at y position 60. This makes it so that the two pieces of text don't overlap each other.

While our `updates` variable isn't too interesting, we've used it as a way to learn about variables and better understand how the game loop runs over and over.

Moving the Text

Let's use player input to change variables and move the text around the screen using the d-pad. Similar to how we tracked the number of updates, we'll introduce two new variables for the text's `x` and `y` position.

Update the `source/main.lua` code to be the following:

```
local updates = 0
local x = 40
local y = 40

function playdate.update()
  updates += 1

  playdate.graphics.clear()
  playdate.graphics.drawText("Hello, Playdate", x, y)
  playdate.graphics.drawText("Updates: " .. updates, 40, 60)
end
```

If you compile and run your game in the Playdate Simulator, you'll notice nothing has changed from before. Great! That was our goal.

We've *refactored* our code—which means we've adjusted the code without changing the functionality. By putting the `x` and `y` positions of where we draw the `"Hello, Playdate"` text into variables, they'll be easy to change.

Now in each loop of the game, we'll need to check to see if the player is pressing any of the d-pad buttons. If they are, then we'll change the `x` and `y` position of where the text is drawn.

Update `source/main.lua` to the following:

```
local updates = 0
local x = 40
local y = 40

function playdate.update()
  updates += 1

  if playdate.buttonIsPressed(playdate.kButtonUp) then
    y -= 2
  end
  if playdate.buttonIsPressed(playdate.kButtonDown) then
    y += 2
  end
  if playdate.buttonIsPressed(playdate.kButtonLeft) then
    x -= 2
  end
  if playdate.buttonIsPressed(playdate.kButtonRight) then
    x += 2
  end

  playdate.graphics.clear()
  playdate.graphics.drawText("Hello, Playdate", x, y)
  playdate.graphics.drawText("Updates: " .. updates, 40, 60)
end
```

The new code in `playdate.update()` checks whether the d-pad button for each direction (up, down, left, and right) is pressed. If it is, then we change our variable accordingly. For moving up, we subtract 2 pixels from the `y` variable. For moving down, we add 2 pixels to the `y` variable. And so on. `-=` works just like `+=` but with subtraction instead of addition.

Conditional checks using `if ... then ... end` are a fundamental part of programming. We check whether some condition is true and act on it. In these cases, we check for a button being pressed. A nice aspect of Lua is that it reads like the English language. If the up button is pressed, then subtract from the y position.

Try changing the value of 2 and seeing what happens.

The text will move around faster or slower because it's moving more or fewer pixels each frame.

Bonus: Refactor the code to extract the speed into a variable so that you only need to change it in one place rather than four.

Player Input to Change Greeting

Let's add one more feature to our greeting learning exercise: the ability to change who we're greeting when the A button is pressed. We'll check for input and then select a random name from a list and update what we're drawing.

```
local updates = 0
local x = 40
local y = 40
local names = { "Playdate", "Goku", "Bulma", "Piccolo" }
local name = names[1]

function playdate.update()
  updates += 1

  -- snip: movement code hidden

  if playdate.buttonJustPressed(playdate.kButtonA) then
    name = names[math.random(#names)]
  end

  playdate.graphics.clear()
  playdate.graphics.drawText("Hello, " .. name, x, y)
  playdate.graphics.drawText("Updates: " .. updates, 40, 60)
end
```

A handful of new concepts are present in the code above. We've got a new variable named `names` with a bunch of strings between curly braces. This is known as a table. A table is a way to group together a collection of different data.

Then we've got the `name` variable, which is assigned to the value at index `1` of the `names` table, which in this case is `"Playdate"`.

We check to see if `playdate.kButtonA` is pressed. But notice we use `playdate.buttonJustPressed` instead of `buttonIsPressed`. There's a subtle but important difference between the two: `buttonJustPressed` returns false if the button is held down, meaning it needs to be pressed again to select another random name. If we used `buttonIsPressed` and held down the A button, the name would keep changing each game loop. (Give it a try and see!)

If the button is just pressed, then we change the variable `name` to be a random entry in the `names` table. The `math.random(#names)` code says: give us a random index between 1 and the number of entries in the `names` table.

Sometimes the name won't change because of the luck of the draw—the random index landed on the same name again.

Finally, we use the `name` variable when we draw our greeting using string concatenation: `playdate.graphics.drawText("Hello, " .. name, x, y)`.

Bonus #1: Add some more names to the `names` table.

Bonus #2: Make pressing the B button reset the `updates` counter.

A Gentle Introduction

Drawing, moving, and changing some text may seem a bit basic, but I promise you that these concepts are absolutely fundamental to making games. We went over the game loop, functions, variables, conditionals, and tables. These core data structures and control flow mechanisms will be used over and over again. And we got our little greeting game working on the Playdate Simulator and the console itself.

Next we'll move on to coding our first actual game! We'll make a single-player tennis game where you move your paddle with the crank.

Tennis

Let's build a simple game of single-player tennis inspired by the classic *Pong*. We'll code a paddle that we can move up and down the screen with the d-pad or crank. When the ball hits the top, right, and bottom of the screen, it'll bounce off. If it goes past our paddle and off the left side of the screen, it'll be game over. We'll keep track of how many hits we get. And to add a little difficulty to our game, we'll make the ball get a bit faster over time.

Displaying the Paddle

Start off by creating a new folder called `tennis` with a `source` folder that contains `source/main.lua`.

Put the following into `source/main.lua`:

```
1  function playdate.update()
2    playdate.graphics.fillRect(36, 80, 12, 48)
3  end
```

`playdate.graphics.fillRect` draws a rectangle on the screen and fills it black (the default fill color). The first parameter is the `x` position, the second is the `y` position, the third is the width, and the fourth is the height. Our code says to draw a rectangle at 36 pixels in, 80 pixels down, with a width of 12 pixels and a height of 48 pixels.

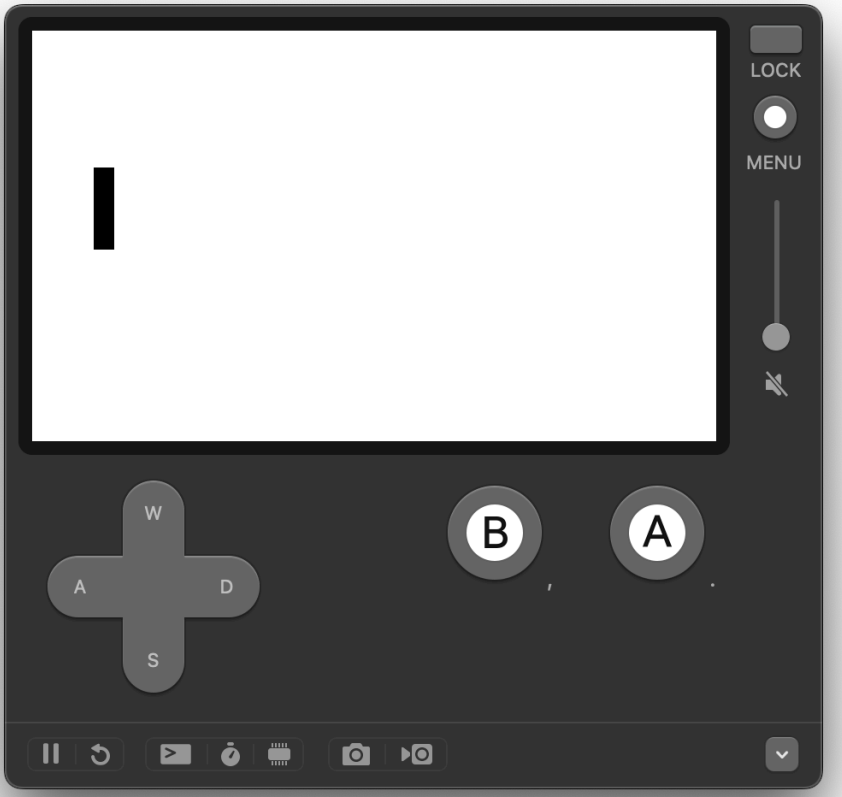

Figure 4. Filled rectangle displayed on the Playdate Simulator

Let's make it so we can move the paddle. This won't be too different from moving text around the screen. But this time let's use a table with key-value pairs to keep track of the paddle position and make our data structure clearer.

Lua is quite flexible in that it allows key-less tables, like we saw with the list of names, `local names = { "Playdate", "Goku", "Bulma", "Piccolo" }`. But Lua also allows us to specify keys and values to easily access and change them. We'll see this in action, but here's an example of the alternative way of using a table:

```
local player = {
  name = "Oolong",
  health = 10,
  level = 5
}
```

The values of `player` can be accessed with dot syntax (`player.name` which returns the string `"Oolong"`) or with bracket syntax (`player["health"]` which returns the number `10`).

Let's update `source/main.lua` to introduce a `paddle` table that has an `x` and `y` position:

```
local paddle = {
  x = 36,
  y = 80,
}

function playdate.update()
  playdate.graphics.fillRect(paddle.x, paddle.y, 12, 48)
end
```

We've refactored our code to keep track of the paddle's position in a table named `paddle`. Very convenient.

Bonus: Put the paddle's width and height into the `paddle` table and reference them when drawing the paddle.

Moving the Paddle

Let's move the paddle. We'll start with the d-pad and then support the crank. Much like with moving the text, we'll check for up and down input on the d-pad.

Update `source/main.lua` to be:

```
local paddle = {
  x = 36,
  y = 80,
  s = 10,
}

function playdate.update()
  if playdate.buttonIsPressed(playdate.kButtonUp) then
    paddle.y -= paddle.s
  end

  if playdate.buttonIsPressed(playdate.kButtonDown) then
    paddle.y += paddle.s
  end

  playdate.graphics.clear()
  playdate.graphics.fillRect(paddle.x, paddle.y, 12, 48)
end
```

Our expanded code moves the paddle up and down by `paddle.s`—a new entry in our table that represents the paddle's speed. Since it's defined in just one place, it's easy to change the value to find a speed that *feels* right.

We also had to make sure we clear the screen every update with `playdate.graphics.clear()`.

Did you notice that the paddle can be moved off the screen? Penalty! Out of bounds! Well, no, not really. But a bad player experience. Let's make it so that the paddle can't go off the Playdate's screen.

We'll add two new conditionals after the button checks. The first checks whether the paddle's `y` position is less than or equal to `0`, and if it is, sets it to `0`. The second checks whether the height of the paddle plus its `y` position is greater than or equal to `240`, which is how many pixels tall the Playdate's screen is.

Let's code that up in `source/main.lua`:

```
local paddle = {
  x = 36,
  y = 80,
  s = 10,
  w = 12,
  h = 48,
}

function playdate.update()
  if playdate.buttonIsPressed(playdate.kButtonUp) then
    paddle.y -= paddle.s
  end

  if playdate.buttonIsPressed(playdate.kButtonDown) then
    paddle.y += paddle.s
  end

  if paddle.y <= 0 then
    paddle.y = 0
  end

  if paddle.y + paddle.h >= 240 then
    paddle.y = 240 - paddle.h
  end

  playdate.graphics.clear()
  playdate.graphics.fillRect(paddle.x, paddle.y, paddle.w, paddle.h)
end
```

Now our paddle stops at the top and bottom of the screen. We use `paddle.y + paddle.h` (rather than just `paddle.y`) for the bottom check because `paddle.y` is the top of the rectangle. Similarly, `paddle.x` represents the left side of the rectangle, not the center.

There's an aspect of this code that I don't love, and it's `240`. When coding, a number that's present without any context to its meaning is known as a **magic number**. It's difficult to know what `240` means at first glance. While we know it's the height of the Playdate's screen, it's not *obvious*. Our goal is to write code that's easy to understand. We could make a variable, `local screen_height = 240` and reference that. But we've got another option that's a little more futureproof. Playdate provides an API to get the height of the display: `playdate.display.getHeight()`.

```
-- snip
local displayHeight = playdate.display.getHeight()

function playdate.update()
  if playdate.buttonIsPressed(playdate.kButtonUp) then
    paddle.y -= paddle.s
  end

  if playdate.buttonIsPressed(playdate.kButtonDown) then
    paddle.y += paddle.s
  end

  if paddle.y <= 0 then
    paddle.y = 0
  end

  if paddle.y + paddle.h >= displayHeight then
    paddle.y = displayHeight - paddle.h
  end

  playdate.graphics.clear()
  playdate.graphics.fillRect(paddle.x, paddle.y, paddle.w, paddle.h)
end
```

This introduces a variable `displayHeight` that's set dynamically from the Playdate SDK and removes the magic number.

Let's whip the crank out and use it to move the paddle up and down. Cranking forward will move the paddle down, while cranking backward will move it up. The Playdate SDK provides the `playdate.getCrankChange()` function, which returns two values: the degrees of angle change since the last time the function was called and an accelerated change value based on how fast the player is moving the crank. We'll use the second value, the accelerated change, since it feels better.

Add the code to `source/main.lua` between the d-pad input checks and the boundary checks:

```lua
-- snip
if playdate.buttonIsPressed(playdate.kButtonUp) then
  paddle.y -= paddle.s
end

if playdate.buttonIsPressed(playdate.kButtonDown) then
  paddle.y += paddle.s
end

local _crankChange, acceleratedCrankChange = playdate.getCrankChange()
paddle.y += acceleratedCrankChange

if paddle.y <= 0 then
  paddle.y = 0
end

if paddle.y + paddle.h >= displayHeight then
  paddle.y = displayHeight - paddle.h
end
-- snip
```

The first return value, `_crankChange`, is prefixed with an underscore to signify that it's not used. Then we take the `acceleratedCrankChange` variable and add it to `paddle.y`. This works because cranking forward returns a positive value and cranking backward returns a negative value. Just what we need.

Bonus #1: Swap `acceleratedCrankChange` for `_crankChange` when adding to `paddle.y` and see how that feels.

Bonus #2: How would you increase or decrease the crank change to refine the paddle movement?

Displaying the Ball

Similar to how we drew a rectangle for the paddle, we'll draw a circle to represent the ball.

At the very top of `source/main.lua`, we need to import the graphics core library. We'll also set up a `ball` table variable for tracking the ball's data.

```
import "CoreLibs/graphics"

local paddle = {
  -- snip
}

local ball = {
  x = 220,
  y = 80,
  r = 10,
}
```

Then at the bottom of the `playdate.update()` function, call `playdate.graphics.fillCircleAtPoint`:

```
playdate.graphics.clear()
playdate.graphics.fillRect(paddle.x, paddle.y, paddle.w, paddle.h)
playdate.graphics.fillCircleAtPoint(ball.x, ball.y, ball.r)
```

`r` represents the radius of the circle in pixels.

Bonus: Adjust `ball.r` to find a size that seems appropriate.

Refactoring `playdate.graphics`

Let's take a brief moment to refactor our calls to `playdate.graphics` to follow the best practices suggested by the Playdate SDK.

We'll introduce a constant value called `gfx` to make our code a bit more concise. It also makes our code slightly faster. Win-win!

Update the top of `source/main.lua` to add the following line below the graphics import:

```
import "CoreLibs/graphics"

local gfx <const> = playdate.graphics
```

`<const>` means that the value is *constant*. It shouldn't change or be reassigned.

Then in `playdate.update()` replace `playdate.graphics` with `gfx`:

```
gfx.clear()
gfx.fillRect(paddle.x, paddle.y, paddle.w, paddle.h)
gfx.fillCircleAtPoint(ball.x, ball.y, ball.r)
```

Moving the Ball

We've coded two moving objects already—the text from Chapter 1 and the paddle in *Tennis*. They respond to user input via the d-pad and crank. The ball in *Tennis* will move automatically. Instead of checking for player input, we'll just modify the ball's position in each update of the game loop.

In `source/main.lua`, between the paddle boundary checks and the drawing code, increase the ball's `x` position by 2 pixels every loop:

```
-- snip
if paddle.y + paddle.h >= displayHeight then
  paddle.y = displayHeight - paddle.h
end

ball.x += 2

gfx.clear()
-- snip
```

Run the game and see what happens. The ball moves to the right. Going, going, gone! It disappears off the screen. It's still moving, deep into space. We just can't see it.

Bonus: Refactor the code to put the ball's speed of 2 into the `ball` table.

Bounce Off the Wall

Much like how we check for the top and bottom of the screen to stop moving the paddle, we'll check to see if the ball has hit the right side of the screen. If it has, then we'll change its direction.

```
-- snip

local ball = {
  x = 220,
  y = 80,
  r = 10,
  s = 4,
}

local displayHeight = playdate.display.getHeight()
local displayWidth = playdate.display.getWidth()

function playdate.update()
  -- snip

  ball.x += ball.s

  if ball.x + ball.r >= displayWidth then
    ball.x = displayWidth - ball.r
    ball.s *= -1
  end

  if ball.x <= ball.r then
    ball.x = ball.r
    ball.s *= -1
  end

  -- snip
end
```

Quite a few changes have been made. First, `s` was added to the `ball` table to represent its speed. Two pixels per update felt too slow, so it's been increased to 4.

The direction is encoded in the sign of `ball.s`—positive means *move right*, negative means *move left*. While combining speed and direction into one variable isn't ideal, that's okay – it'll change when we introduce angles.

We set a variable for `displayWidth` just like we did for `displayHeight` but call `getWidth()` instead. We need it for checking the ball's boundaries.

When we change `ball.x`, we use `ball.s` instead of the magic number from the last section. And we multiply `ball.s` by `-1` to change the direction. `*=` means multiply the value on the left by the value on the right and assign that new value to it. Multiplying by `-1` changes the sign of the direction.

If the ball is moving right, we're increasing its `x` value, so `ball.s` is positive. But if we want the ball to move to the left, we need to subtract `ball.s` from

`ball.x`. Multiplying `ball.s` by `-1` causes it to subtract 4 from `ball.x` in future loops. To make the ball move right again, multiply it by `-1` to turn it positive. This sign-flipping technique is quite powerful and a regular staple of game programming.

Then we check if the ball's `x` position plus its radius (`r`) is greater than or equal to the width of the screen. If it is, we set the ball's position to the rightmost edge and flip the direction by multiplying `ball.s` by `-1` so it moves to the left in the next call to `playdate.update()`.

Similarly, for the left side of the screen, we check if `ball.x` is less than or equal to the ball's radius (`r`). If it is, we set `ball.x` to `ball.r` and flip the direction so the ball moves to the right.

The boundary checking for the ball is slightly different from our paddle as the origin point of the circle is in the center, not the upper left. We need to factor that into our logic. Try changing `if ball.x <= ball.r then` to `if ball.x <= 0 then` and see what happens. (And then undo it because it's not what we want moving forward.)

While it's fun to watch the ball move back and forth, it flies right through our paddle. Let's make it so that when the ball overlaps with the paddle, it changes direction.

Bounce Off the Paddle

To bounce the ball off the paddle, we'll check to see if the circular ball overlaps the rectangular paddle. This is where we introduce some non-trivial math into our game. Don't worry – I'll break it down and explain it.

We'll also introduce our first custom function to encapsulate the logic. Our `playdate.update()` function is getting a bit long and unwieldy, so by putting code into functions, we can keep it clear and focused.

```lua
-- snip
function playdate.update()
  -- snip

  ball.x += ball.s

  if circleOverlapsRect(ball, paddle) then
    ball.s *= -1
  end

  -- snip
end

function circleOverlapsRect(circle, rect)
  -- Find the closest point in the rectangle to the circle's center
  local closestX = math.max(rect.x, math.min(circle.x, rect.x + rect.w))
  local closestY = math.max(rect.y, math.min(circle.y, rect.y + rect.h))

  -- Calculate the distance between the circle's center and the closest point
  local distance = math.sqrt((closestX - circle.x)^2 + (closestY - circle.y)^2)

  -- If the distance is less than or equal to the radius, there's overlap
  return distance <= circle.r
end
```

Right below where we update the ball's `x` position, we'll add a conditional check using a new function: `circleOverlapsRect`. We pass in the `ball` (our circle) as the first parameter and then the `paddle` (our rectangle) as the second parameter. If the circle does overlap the rectangle, meaning the ball hit our paddle, we'll flip the ball's direction by changing the sign of its speed.

The `circleOverlapsRect` function takes a `circle` and `rect` as arguments. Comments starting with `--` walk through the logic.

Let's break down how we determine `closestX` and `closestY`. These calculations find the point within the `rect` that's closest to the `circle` by comparing the center of the circle to the bounding box of the `rect`.

The distance formula is then used to determine the distance between the closest point in the `rect` and the center of the `circle`. If the distance is less than or equal to the circle's radius (`circle.r`), then the circle is overlapping the rectangle.

Refactor into Functions

Our code in `playdate.update()` is getting complex and unwieldy. Let's refactor the code we've got into separate functions that we call from within `playdate.update()`. Putting code into functions serves three key purposes:

1. Functions help us reuse code rather than repeating it multiple times.
2. Functions break our code down into smaller parts that are easier to understand.
3. Functions self-document our code by grouping related lines under a meaningful name.

Our current `source/main.lua` looks like this:

```
1  import "CoreLibs/graphics"
2
3  local gfx <const> = playdate.graphics
4
5  local paddle = {
6    x = 36,
7    y = 80,
8    s = 10,
9    w = 12,
10   h = 48,
11 }
12
13 local ball = {
14   x = 220,
15   y = 80,
16   r = 10,
17   s = 6,
18 }
19
20 local displayHeight = playdate.display.getHeight()
21 local displayWidth = playdate.display.getWidth()
22
23 function playdate.update()
24   if playdate.buttonIsPressed(playdate.kButtonUp) then
25     paddle.y -= paddle.s
26   end
27
28   if playdate.buttonIsPressed(playdate.kButtonDown) then
29     paddle.y += paddle.s
30   end
31
```

```
  local _crankChange, acceleratedCrankChange = playdate.getCrankChange()
  paddle.y += acceleratedCrankChange

  if paddle.y <= 0 then
    paddle.y = 0
  end

  if paddle.y + paddle.h >= displayHeight then
    paddle.y = displayHeight - paddle.h
  end

  ball.x += ball.s

  if ball.x + ball.r >= displayWidth then
    ball.x = displayWidth - ball.r
    ball.s *= -1
  end

  if ball.x <= ball.r then
    ball.x = ball.r
    ball.s *= -1
  end

  if circleOverlapsRect(ball, paddle) then
    ball.s *= -1
  end

  gfx.clear()
  gfx.fillRect(paddle.x, paddle.y, paddle.w, paddle.h)
  gfx.fillCircleAtPoint(ball.x, ball.y, ball.r)
end

function circleOverlapsRect(circle, rect)
  -- Find the closest point in the rectangle to the circle's center
  local closestX = math.max(rect.x, math.min(circle.x, rect.x + rect.w))
  local closestY = math.max(rect.y, math.min(circle.y, rect.y + rect.h))

  -- Calculate the distance between the circle's center and the closest point
  local distance = math.sqrt((closestX - circle.x)^2 + (closestY - circle.y)^2)

  -- If the distance is less than or equal to the radius, there's overlap
  return distance <= circle.r
end
```

I see three functions we can extract:

1. Moving the paddle: `movePaddle()`
2. Moving the ball: `moveBall()`
3. Drawing our shapes: `draw()`

Here's what our refactored code looks like:

```
import "CoreLibs/graphics"

local gfx <const> = playdate.graphics

local paddle = {
  x = 36,
  y = 80,
  s = 10,
  w = 12,
  h = 48,
}

local ball = {
  x = 220,
  y = 80,
  r = 10,
  s = 6,
}

local displayHeight = playdate.display.getHeight()
local displayWidth = playdate.display.getWidth()

function playdate.update()
  movePaddle()
  moveBall()

  if circleOverlapsRect(ball, paddle) then
    ball.s *= -1
  end

  draw()
end

function draw()
  gfx.clear()
  gfx.fillRect(paddle.x, paddle.y, paddle.w, paddle.h)
  gfx.fillCircleAtPoint(ball.x, ball.y, ball.r)
end

function movePaddle()
  if playdate.buttonIsPressed(playdate.kButtonUp) then
    paddle.y -= paddle.s
  end

  if playdate.buttonIsPressed(playdate.kButtonDown) then
    paddle.y += paddle.s
  end

```

```
49    local _crankChange, acceleratedCrankChange = playdate.getCrankChange()
50    paddle.y += acceleratedCrankChange
51
52    if paddle.y <= 0 then
53      paddle.y = 0
54    end
55
56    if paddle.y + paddle.h >= displayHeight then
57      paddle.y = displayHeight - paddle.h
58    end
59  end
60
61  function moveBall()
62    ball.x += ball.s
63
64    if ball.x + ball.r >= displayWidth then
65      ball.x = displayWidth - ball.r
66      ball.s *= -1
67    end
68
69    if ball.x <= ball.r then
70      ball.x = ball.r
71      ball.s *= -1
72    end
73  end
74
75  function circleOverlapsRect(circle, rect)
76    -- Find the closest point in the rectangle to the circle's center
77    local closestX = math.max(rect.x, math.min(circle.x, rect.x + rect.w))
78    local closestY = math.max(rect.y, math.min(circle.y, rect.y + rect.h))
79
80    -- Calculate the distance between the circle's center and the closest point
81    local distance = math.sqrt((closestX - circle.x)^2 + (closestY - circle.y)^2)
82
83    -- If the distance is less than or equal to the radius, there's overlap
84    return distance <= circle.r
85  end
```

All we've done is move our code into separate functions and then call them within `playdate.update()`. The code is a lot easier to reason about now.

The call to `circleOverlapsRect` stays in `playdate.update()` because it's simple enough to leave there. It wouldn't fit naturally into `movePaddle()` or `moveBall()` since it deals with both of them. If collision detection gets more complicated, we could extract it into its own function.

Now that our code is cleaned up a bit, let's make the game more exciting by introducing angles so the ball bounces all over the screen.

Angles

Our ball is just moving along the x-axis, which isn't very exciting. We want it to move all over the screen. This means, yes, more math! We need to keep track of the ball's angle and combine it with our speed to determine its velocity.

```
-- snip

local ball = {
  x = 220,
  y = 80,
  r = 10,
  s = 6,
  a = 195, -- degrees
}

-- snip

function playdate.update()
  movePaddle()
  moveBall()

  if circleOverlapsRect(ball, paddle) then
    ball.a = math.random(160, 200) - ball.a
  end

  draw()
end

function draw()
  -- snip
end

function movePaddle()
  -- snip
end

function moveBall()
  local radians = math.rad(ball.a)
  ball.x += math.cos(radians) * ball.s
  ball.y += math.sin(radians) * ball.s

  if ball.x + ball.r >= displayWidth then
    ball.x = displayWidth - ball.r
    ball.a = 180 - ball.a
  end

  if ball.x <= ball.r then
```

```
43      ball.x = ball.r
44      ball.a = 180 - ball.a
45    end
46
47    if ball.y + ball.r >= displayHeight then
48      ball.y = displayHeight - ball.r
49      ball.a *= -1
50    end
51
52    if ball.y <= ball.r then
53      ball.y = ball.r
54      ball.a *= -1
55    end
56  end
57
58  function circleOverlapsRect(circle, rect)
59    -- snip
60  end
```

Play your game on the Playdate Simulator and upload it to your Playdate to playtest if you haven't yet. It's starting to get fun!

There are some major changes to how we're handling the ball's movement.

First, we introduce `ball.a` to represent the ball's angle of movement. It's in degrees and initially set to `195`, which is to the left and slightly up. (`180` would be straight to the left.) We'll use the angle when determining how to change the `x` and `y` position of the ball based on its speed.

In `playdate.update()` we set `ball.a` to a random value between `160` and `200` degrees, minus the current `ball.a`. This turns the ball around and gives it a little random variance to prevent it from getting deadlocked in a straight line.

`draw()`, `movePaddle()`, and `circleOverlapsRect()` remain unchanged. But `moveBall()` gets entirely overhauled.

Let's break down the two key parts:

```
1  local radians = math.rad(ball.a)
2  ball.x += math.cos(radians) * ball.s
3  ball.y += math.sin(radians) * ball.s
```

Since the ball is moving at an angle, both its `x` and `y` positions need to change based on that angle. Trigonometry!

We convert the ball's angle to radians, which we need for working with the sine (`math.sin()`) and cosine (`math.cos()`) functions. We calculate the change in the `x` position based on the cosine of the angle times the ball's speed. The `y` position is similar but we use sine instead. This formula uses the unit circle to convert an angle and a speed into a velocity.

After we change the `x` and `y` position, we check to see if the ball has collided with the top, bottom, left, and right sides of the screen. For the left and right sides along the `x` axis, we subtract the current `ball.a` from `180` to flip the horizontal movement. For the top and bottom collisions along the `y` axis, we multiply `ball.a` by `-1` to keep the horizontal direction the same but change the vertical direction.

Let's make it so the ball gets faster each time it hits the paddle. This makes the game more challenging as we get better at it.

Speeding Up the Ball

To make the ball go faster, all we need to do is increase `ball.s` inside our `if circleOverlapsRect(ball, paddle) then` block in `playdate.update()`.

```
1  -- snip
2
3  if circleOverlapsRect(ball, paddle) then
4    ball.a = math.random(160, 200) - ball.a
5    ball.s += 1
6  end
7
8  -- snip
```

It quickly gets out of hand and the ball moves way too fast. So let's set a max speed by adding `max_s` to our `ball` table and only incrementing `ball.s` while it's below `max_s`.

```
-- snip

local ball = {
  x = 220,
  y = 80,
  r = 10,
  s = 6,
  max_s = 12,
  a = 195, -- degrees
}

-- snip

function playdate.update()
  movePaddle()
  moveBall()

  if circleOverlapsRect(ball, paddle) then
    ball.a = math.random(160, 200) - ball.a

    if ball.s < ball.max_s then
      ball.s += 1
    end
  end

  draw()
end

-- snip
```

There's something really satisfying about using the crank to hit the ball around! We're starting to get our first glimpses of the fun and joy of making games.

Score

Let's keep track of the score and reset everything when the ball hits the left wall. We need a new variable, `score`, that we'll set and draw on the screen. We'll also save the ball's initial position, speed, and angle so we can restore them when resetting.

```
-- snip

local ball = {
  x = 220,
  y = 80,
  r = 10,
  s = 6,
  max_s = 12,
  a = 195, -- degrees
}

ball.initX = ball.x
ball.initY = ball.y
ball.initS = ball.s
ball.initA = ball.a

local score = 0

local displayHeight = playdate.display.getHeight()
local displayWidth = playdate.display.getWidth()

function playdate.update()
  movePaddle()
  moveBall()

  if circleOverlapsRect(ball, paddle) then
    ball.a = math.random(160, 200) - ball.a

    score += 1

    if ball.s < ball.max_s then
      ball.s += 1
    end
  end

  draw()
end

function draw()
  -- snip
  gfx.drawText("Score: " .. score, displayWidth - 100, 20)
end

function movePaddle()
  -- snip
end

function moveBall()
  -- snip

```

```
  if ball.x <= ball.r then
    resetGame()
  end

  -- snip
end

function resetGame()
  score = 0
  ball.x = ball.initX
  ball.y = ball.initY
  ball.s = ball.initS
  ball.a = ball.initA
end

-- snip
```

There's nothing too new here – just using the fundamentals we already learned, with the addition of `ball.initX`, `ball.initY`, and so on. After we create the `ball` table, we copy its starting values into separate entries because `ball.x`, `ball.y`, etc. are going to change throughout the game. Saving these initial values lets us easily reset the ball to its starting state.

We're almost done with *Tennis*. Let's polish it up a little bit more.

Sound Effects

Our game is feeling a bit quiet, don't you think? The Playdate SDK offers a lot of options for playing sound effects, from samples to files to an included synth. We'll add some simple MIDI sound effects to our game whenever the ball hits anything.

```
-- snip
local synth = playdate.sound.synth.new(playdate.sound.kWaveSine)

function playdate.update()
  movePaddle()
  moveBall()

  if circleOverlapsRect(ball, paddle) then
    ball.a = math.random(160, 200) - ball.a
    playSFX("C4")

    score += 1
```

```
13
14      -- snip
15    end
16
17    draw()
18  end
19
20  function draw()
21    -- snip
22  end
23
24  function movePaddle()
25    -- snip
26  end
27
28  function moveBall()
29    -- snip
30
31    if ball.x + ball.r >= displayWidth then
32      playSFX("E4")
33      ball.x = displayWidth - ball.r
34      ball.a = 180 - ball.a
35    end
36
37    if ball.x <= ball.r then
38      playSFX("F3")
39      resetGame()
40    end
41
42    if ball.y + ball.r >= displayHeight then
43      playSFX("D4")
44      ball.y = displayHeight - ball.r
45      ball.a *= -1
46    end
47
48    if ball.y <= ball.r then
49      playSFX("A4")
50      ball.y = ball.r
51      ball.a *= -1
52    end
53  end
54
55  function resetGame()
56    -- snip
57  end
58
59  function playSFX(note)
60    synth:playMIDINote(note, 1, 0.25)
61  end
62
```

```
63  -- snip
```

We create a new `synth` variable using the `kWaveSine` constant. We use it in the new `playSFX` function, which takes a note and plays it through the synth for a quarter of a second. The second parameter, `1`, is the volume of the sound effect. You can see that for each wall the ball hits, we play a different note. This is more pleasing than repeating the same sound over and over. The sound when the ball hits the paddle is also different.

Bonus: Similar to how we picked a random name to greet in the first chapter, create a table of music notes and randomly select one when the ball hits the paddle.

Final Code

That's *Tennis*! Our version is complete, at least for now. We learned a whole lot in this chapter. We wrote our own functions, drew shapes, implemented angular velocity, created and changed plenty of variables, and made something that's a little fun. Nice work.

Here's the finished code for this chapter:

```
1   import "CoreLibs/graphics"
2
3   local gfx <const> = playdate.graphics
4
5   local paddle = {
6     x = 36,
7     y = 80,
8     s = 10,
9     w = 12,
10    h = 48,
11  }
12
13  local ball = {
14    x = 220,
15    y = 80,
16    r = 10,
17    s = 6,
18    max_s = 12,
19    a = 195, -- degrees
20  }
21
```

```
ball.initX = ball.x
ball.initY = ball.y
ball.initS = ball.s
ball.initA = ball.a

local score = 0

local displayHeight = playdate.display.getHeight()
local displayWidth = playdate.display.getWidth()
local synth = playdate.sound.synth.new(playdate.sound.kWaveSine)

function playdate.update()
  movePaddle()
  moveBall()

  if circleOverlapsRect(ball, paddle) then
    ball.a = math.random(160, 200) - ball.a
    playSFX("C4")

    score += 1

    if ball.s < ball.max_s then
      ball.s += 1
    end
  end

  draw()
end

function draw()
  gfx.clear()
  gfx.fillRect(paddle.x, paddle.y, paddle.w, paddle.h)
  gfx.fillCircleAtPoint(ball.x, ball.y, ball.r)
  gfx.drawText("Score: " .. score, displayWidth - 100, 20)
end

function movePaddle()
  if playdate.buttonIsPressed(playdate.kButtonUp) then
    paddle.y -= paddle.s
  end

  if playdate.buttonIsPressed(playdate.kButtonDown) then
    paddle.y += paddle.s
  end

  local _crankChange, acceleratedCrankChange = playdate.getCrankChange()
  paddle.y += acceleratedCrankChange

  if paddle.y <= 0 then
    paddle.y = 0
```

```
  end

  if paddle.y + paddle.h >= displayHeight then
    paddle.y = displayHeight - paddle.h
  end
end

function moveBall()
  local radians = math.rad(ball.a)
  ball.x += math.cos(radians) * ball.s
  ball.y += math.sin(radians) * ball.s

  if ball.x + ball.r >= displayWidth then
    playSFX("E4")
    ball.x = displayWidth - ball.r
    ball.a = 180 - ball.a
  end

  if ball.x <= ball.r then
    playSFX("F3")
    resetGame()
  end

  if ball.y + ball.r >= displayHeight then
    playSFX("D4")
    ball.y = displayHeight - ball.r
    ball.a *= -1
  end

  if ball.y <= ball.r then
    playSFX("A4")
    ball.y = ball.r
    ball.a *= -1
  end
end

function resetGame()
  score = 0
  ball.x = ball.initX
  ball.y = ball.initY
  ball.s = ball.initS
  ball.a = ball.initA
end

function playSFX(note)
  synth:playMIDINote(note, 1, 0.25)
end

function circleOverlapsRect(circle, rect)
  -- Find the closest point in the rectangle to the circle's center
```

```
122   local closestX = math.max(rect.x, math.min(circle.x, rect.x + rect.w))
123   local closestY = math.max(rect.y, math.min(circle.y, rect.y + rect.h))
124
125   -- Calculate the distance between the circle's center and the closest point
126   local distance = math.sqrt((closestX - circle.x)^2 + (closestY - circle.y)^2)
127
128   -- If the distance is less than or equal to the radius, there's overlap
129   return distance <= circle.r
130 end
```

130 lines of code. That's respectable for our first Playdate game. If you have someone to share your Playdate with, show them what you made!

Extra Credits

There's a lot you could do to expand *Tennis* into something more fun. Here are some ideas to take it to the next level:

- Make the game two-player by introducing another paddle. Player 1 controls the left paddle with the d-pad and Player 2 controls the right paddle with the crank.
- Add multiple balls that spawn after reaching certain score thresholds!
- Add collectibles to aim for or even bricks to break.
- Add pinball-style bumpers that the ball can hit.
- Make the ball appear larger when it hits the paddle or a wall.
- Shake the paddle when it's hit by the ball.
- There's a small bug—sometimes the ball and paddle can get stuck on each other. How would you use the ball's position when it hits the paddle to separate them?
- In future chapters we'll learn about saving data. Make it so that the high score is saved between play sessions. When a new high score is reached, modify the score text to let the player know!
- Adding a mute setting would be nice!
- Rework the paddle's movement to use acceleration and factor its current velocity into the angle at which the ball bounces (e.g., if the paddle hits the ball while the paddle is moving up, the ball should also move up and to the right).

What's Next

Tennis was a lot to soak in. Let's take a breather by coding up a simple little utility for our Playdate: a clock.

Clock

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Drawing the Time

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Respecting Preferences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Padding Zeroes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Lowering the Framerate

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Showing Power Details

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Custom Font

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Finished Source Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Extra Credits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

What's Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Snake

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Moving on the Grid

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Control the Snake

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Spawning Apples

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Adding Snake Pieces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Don't Spawn an Apple on the Snake

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Game Over

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Restart the Game

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

High Score

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Reset High Score

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Finished Source Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Extra Credits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

What's Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Soaring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Controlling the Bird

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Spawning Trees

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Crank Indicator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Scenes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Refactoring Gameplay Reset

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Introducing a State Table

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Breaking Our Code Into Multiple Files

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Extra Credits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

What's Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Sokoban

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Moveable Player

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Pushing a Box

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Target Spot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Counting Steps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Preparing for Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Multiple Boxes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Parsing Levels

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Level Select

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Extra Credits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

What's Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Dungeon Crawler - Part 1

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Drawing Player from a Tilemap

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Designing Levels with Tiled and Rendering Them

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Player Movement with Camera

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Map Collision Detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Talking to NPCs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Extra Credits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

What's Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Dungeon Crawler - Part 2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Organizing Our Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Multiple Maps with Stairs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Random Encounters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Combat

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Defining Enemies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Setting Up for Combat

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Initializing Combat

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Drawing the Combat Screen

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Navigating the Menu

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

The Message Queue

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Updating and Drawing Combat

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Player Stats & Enemy Attacks Back

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Death

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Extra Credits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

What's Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Playdate by Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Text

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Use a Font

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Audio

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Play a Sound Effect from WAV File

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Creating a Sound Effect Table Module

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Capturing Gameplay Footage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Release Steps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Outro

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Start Small

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Releasing Your Playdate Games

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Continue to Learn

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Backing Up Your Games

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Beyond Playdate

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Game Development Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Assets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Stay in Touch

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Thanks & Credit

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Playdate Cheatsheet

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Display

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Draw Text

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Import Files

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Input Handling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Sprites

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Graphics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Images

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Timers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Sound

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Save Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Animation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

System Menu

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Lifecycle Callbacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Coroutines in playdate.update()

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Common Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Game State Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Simple Animation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Collision Detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Camera/Scrolling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Quick Constants Reference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Lua Cheatsheet

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Variables

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Variable Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Multiple Assignment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Local Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Anonymous Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Functions with Multiple Returns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Variable Arguments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Tables

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Create Tables

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Access Table Values

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Add to a Table

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Remove from Table

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Iterate Through Tables

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Table Length

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Strings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

String Creation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

String Concatenation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

String Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

String Formatting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

String Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Control Flow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

If Statements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Logical Operators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Operators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Arithmetic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Comparison

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Assignment Shortcuts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Common Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Default Values

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Table as Object

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Error Handling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Module Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Useful Built-in Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Math

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Important Playdate Notes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Type Checking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Conversion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Coroutines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Basic Coroutine Usage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Coroutine Status

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Passing Values

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Common Coroutine Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Playdate-Specific Coroutine Usage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Style Guide

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Lua Styles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.

Project Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/playdatebook.