# 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 2025-11-12

# 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 be able to play it on an actual handheld game console within seconds. Playdate's approachability and developer friendliness is unrivaled in the game development space.

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

## Early Access

*Make Games for Playdate with Lua* is in Early Access. This means I'm still actively writing the book and adding more content regularly. It's not polished, and it's not finished. There will be typos, as the book has not yet been closely proofread and edited. New chapters will be added regularly and issues will be fixed.

If you run into any issues or have any feedback, send me an email at playdatebook@fastmail.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 and utilities together. I've got lots of ideas for what to cover, like the classic *Snake* to a minimal *Vampire Survivors* clone to a fishing game with the crank.

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.

Files on your operating system are referenced using the `/` to represent the folder. If you see reference to a file in this format: `source/player.lua`, it means within the folder named `source` there should

be a file named `player.lua`. Linux and macOS use `/` but Windows uses `\` to delineate folder paths, but 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[1]. The Playdate SDK contains everything you need to make games for Playdate—example code, a simulator to run your

[1] https://play.date/dev/

games, a manual called *Inside Playdate* with design guidelines, and more. The Playdate SDK supports Windows, macOS, and Linux.

The only other thing you need right now to make games for Playdate is a text editor for writing code. Your text editor is *different* than 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[2]. It's free, available on Windows, macOS, and Linux. There are some helpful extensions for Visual Studio Code for making games with Playdate.

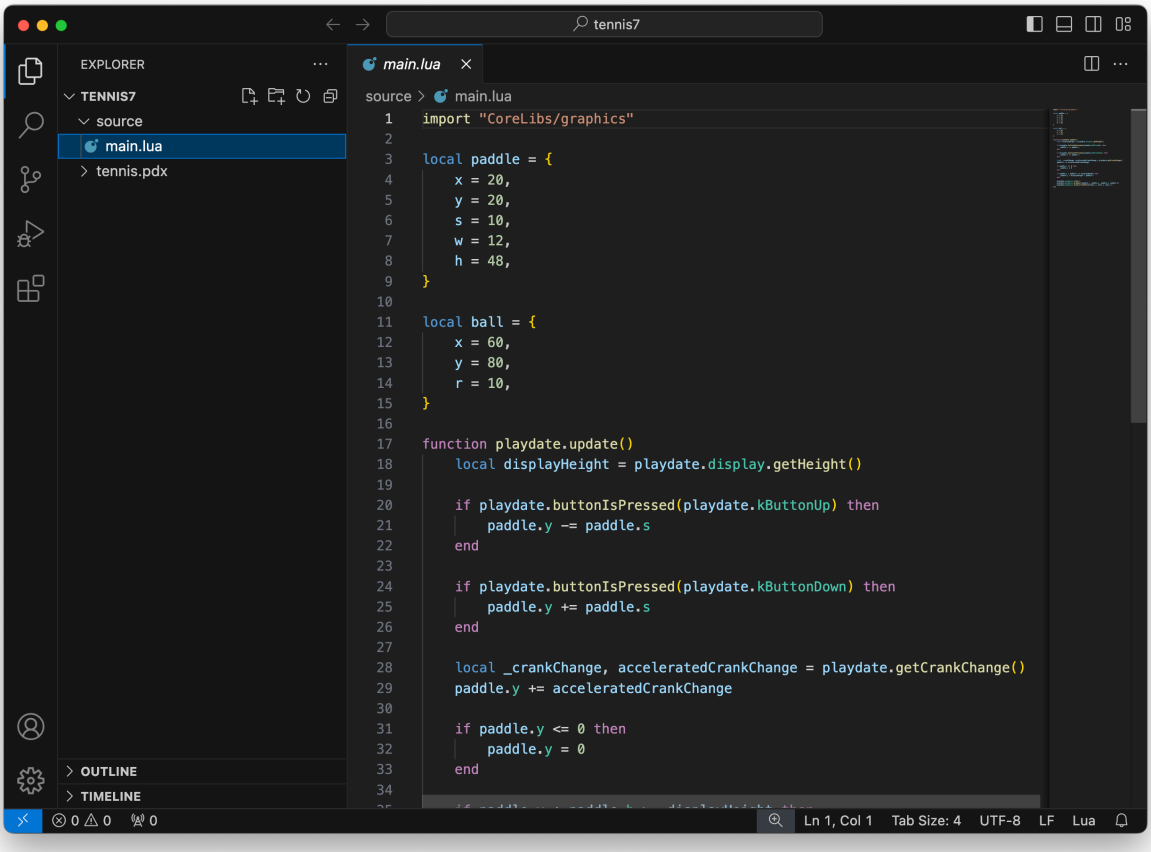


**Figure 1. Screenshot of Visual Studio Code**

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

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

[2] https://code.visualstudio.com/

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.

Once you've got the Playdate SDK installed and a text editor, you're ready to start making a Playdate game!

-# Part 1: The Basics

# Hello, Playdate!

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

## Counting Frames

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

## Moving the Text

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

## Player Input to Change Greeting

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

## A Gentle Introduction

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

# Tennis

Let's code 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 it. 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 little 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 fill 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.

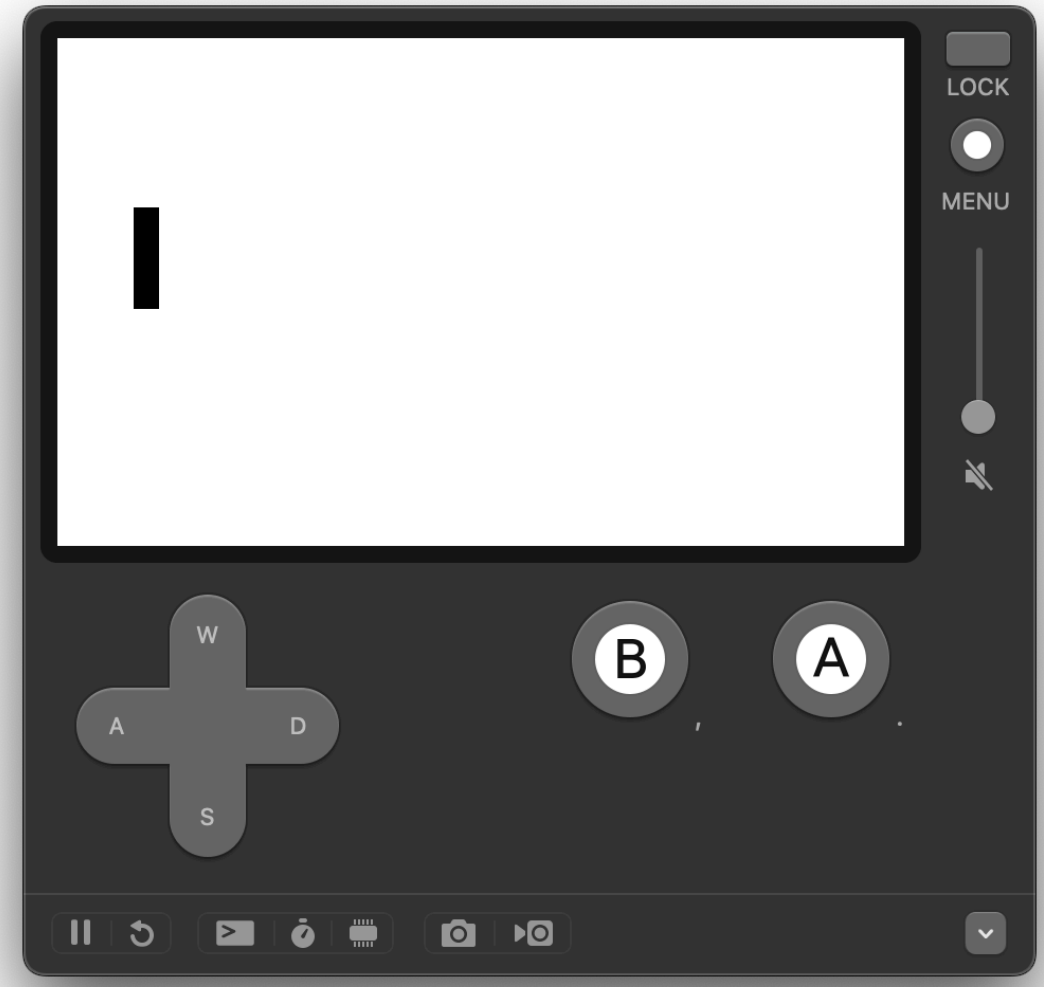


**Figure 2. Filled rectangle displayed on the Playdate Simulator**

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 to keep track of the paddle position and use key-value pairs to make our data structure more clear.

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

```lua
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:

```lua
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` variable and reference it from the variable 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 the up and down input on the d-pad.

Update `source/main.lua` to be:

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

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

```
11
12      if playdate.buttonIsPressed(playdate.kButtonDown) then
13          paddle.y += paddle.s
14      end
15
16      playdate.graphics.clear()
17      playdate.graphics.fillRect(paddle.x, paddle.y, 12, 48)
18  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. By having it live in just one place, it's easy to change the value to find a speed that *feels* right. Try changing `s` around to find a speed that feels good to you.

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 outside the bounds of the screen of the Playdate.

We'll introduce a condition within our checks for buttons being pressed. For up input, we'll check to see if the paddle's `y` position is less than or equal to `0`, and if it is, then we'll set it to `0`. For down input, we'll check to see if the height of the paddle plus its `x` position is greater than or equal to `240`, which is how many pixels tall the Playdate is.

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

```
1   local paddle = {
2       x = 36,
3       y = 80,
4       s = 10,
5       w = 12,
6       h = 48,
7   }
8
9   function playdate.update()
10      if playdate.buttonIsPressed(playdate.kButtonUp) then
11          paddle.y -= paddle.s
12      end
13
14      if playdate.buttonIsPressed(playdate.kButtonDown) then
15          paddle.y += paddle.s
16      end
17
18    if paddle.y <= 0 then
19      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 have to check that the paddle's `y` position plus its height is less than the screen height because `paddle.y` is the top of the rectangle. And the `x` position is the left side.

[diagram illustrating the positioning and where things are drawn]

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()
```

```
23      playdate.graphics.fillRect(paddle.x, paddle.y, paddle.w, paddle.h)
24  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 have turning it move the paddle up and down. Cranking forward will move the paddle down, while cranking backwards will move 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` to between the d-pad input checking and the boundary checking:

```
1   -- snip
2   if playdate.buttonIsPressed(playdate.kButtonUp) then
3       paddle.y -= paddle.s
4   end
5
6   if playdate.buttonIsPressed(playdate.kButtonDown) then
7       paddle.y += paddle.s
8   end
9
10  local _crankChange, acceleratedCrankChange = playdate.getCrankChange()
11  paddle.y += acceleratedCrankChange
12
13  if paddle.y <= 0 then
14      paddle.y = 0
15  end
16
17  if paddle.y + paddle.h >= displayHeight then
18      paddle.y = displayHeight - paddle.h
19  end
20  -- 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 backwards returns a negative value. Just what we need.

(debugging with `print` to see the crank values)

**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 that 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 the `r` radius of the circle 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 nor 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 our tennis game. They responded 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` below the paddle boundary checking and above the drawing, 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.

```lua
-- 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. `2` pixels per update felt too slow, so it's been increased to `4`.

`d` was also added to the `ball` table and it represents the horizontal *direction*. While it's not super clear, that's okay, it'll end up going away when we introduce angles. But for this section, let's stick with `d`. The value defaults to `1` which means *move right*. `-1` means *move left*.

We set a variable for the `displayWidth` just like we did for `displayHeight` but call `getWidth()` instead. We need for checking the boundary of the ball.

When we change the `ball.x`, we use `ball.s` instead of the magic number from the last section. And we multiple it by `-1` to change the direction. `*=` means multiple the value on the left by the value on the right and assign that new value to it. Multiplying the `-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`. By multiplying `ball.s` by `-1`, it subtracts `4` from `ball.x` in future loops. To make the ball move right again, multiply it by `-1` to turn it positive. This directional modifier is quite powerful and a regular staple of game programming.

Then we check if the ball's `x` position plus it's radius (`r`) are greater than or equal to the width of the screen. If it is, we set the ball's position to the furthest right edge and change the direction to `-1` so it moves to the left in the next game loop call of `playdate.update()`.

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

The boundary checking for the ball is slightly different than our padddle as the origin point of the circle is in the center, not the upper left. This means we need to factor this 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 that when the ball overlaps with the paddle, it changes direction.

## Bounce Off the Paddle

To bounce the ball off of the paddle, we'll check to see if the circular ball overlaps the rectangular paddle. This is where we introduce non-trivial math into our game in the form of trigonometry. Don't worry if you don't know much trig, 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 unweildy, so by putting code into functions, we can keep it clear and focused.

```
1  -- snip
2  function playdate.update()
3      -- snip
4
5      ball.x += ball.s
6
7      if circleOverlapsRect(ball, paddle) then
8          ball.s *= -1
9      end
10
11     -- snip
12 end
13
14 function circleOverlapsRect(circle, rect)
15     -- Find the closest point in the rectangle to the circle's center
16     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 with the new function we're adding: `circleOverlapsRect`. We pass in the `ball` (our circle) as the first parameter and then the `paddle` (our rectange) 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 by .

The `circleOverlapsRect` function takes a `circle` and `rect` as arguments. There are some comments that start with `--` to explain the code a bit.

Let's breakdown the determination of `closestX` and `closestY`. It checks to see what point within the `rect` is 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 it means 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 which are easier to understand.
3. Functions self-document our code by grouping lines together into something with a name.

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

```
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()
    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

    ball.x += ball.s
```

```
44
45      if ball.x + ball.r >= displayWidth then
46          ball.x = displayWidth - ball.r
47          ball.s *= -1
48      end
49
50      if ball.x <= ball.r then
51          ball.x = ball.r
52          ball.s *= -1
53      end
54
55      if circleOverlapsRect(ball, paddle) then
56          ball.s *= -1
57      end
58
59      gfx.clear()
60      gfx.fillRect(paddle.x, paddle.y, paddle.w, paddle.h)
61      gfx.fillCircleAtPoint(ball.x, ball.y, ball.r)
62  end
63
64  function circleOverlapsRect(circle, rect)
65      -- Find the point to the circle center within the rectangle
66      local closestX = math.max(rect.x, math.min(circle.x, rect.x + rect.w))
67      local closestY = math.max(rect.y, math.min(circle.y, rect.y + rect.h))
68
69      -- Distance between the circle center and the closest point
70      local distance = math.sqrt((closestX - circle.x)^2 + (closestY - circle.y)^2)
71
72      -- If the distance is less than or equal to the radius, there's overlap
73      return distance <= circle.r
74  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
```

```lua

    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()
    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
end

function circleOverlapsRect(circle, rect)
    -- Find the point to the circle center within the rectangle
    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))

    -- Distance between the circle 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
```

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

The call to the `circleOverlapsRect` function did not get moved into its own function because it's simple enough to just stay put. It wouldn't fit naturally into `movePaddle()` or `moveBall()` since it deals with both of them. If collision detection gets more complicated, then it might make sense to move it into its own function.

Now that our code is cleaned up a bit, let's make the game more exciting. Let's introduce angles to make the ball bounce all over the screen.

## Angles

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

```
1  -- snip
2
3  local ball = {
4      x = 220,
5      y = 80,
6      r = 10,
7      s = 6,
8      a = 195, -- degrees
9  }
10
11 -- snip
12
13 function playdate.update()
14     movePaddle()
15     moveBall()
16
17     if circleOverlapsRect(ball, paddle) then
18         ball.a = math.random(160, 200) - ball.a
19     end
20
21     draw()
22 end
23
24 function draw()
25     -- snip
26 end
```

```
27
28  function movePaddle()
29      -- snip
30  end
31
32  function moveBall()
33      local radians = math.rad(ball.a)
34      ball.x += math.cos(radians) * ball.s
35      ball.y += math.sin(radians) * ball.s
36
37      if ball.x + ball.r >= displayWidth then
38          ball.x = displayWidth - ball.r
39          ball.a = 180 - ball.a
40      end
41
42      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 to your Playdate to play test it if you haven't yet. It's starting to get fun!

There are some major changes in the code with how we're handling the ball 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 right and down a little bit. (`180` would be straight to the right.) 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 the `ball.a` to a random value between `160` and `200` degrees minus

`ball.a`. This turns the ball around and gives it a little bit of random variance to prevent the ball 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:

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

Since the ball is moving at an angle, both its `x` and `y` position need to change based on that angle. More 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 insead. The foundations of this formula is using the unit circle to determine angular 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 side of the screen. For the left and right side 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 the `ball.a` 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 will make the game more challenging the better we get at it.

## Speeding Up the Ball

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

```
-- snip

if circleOverlapsRect(ball, paddle) then
    ball.a = math.random(160, 200) - ball.a
    ball.s += 1
end

-- snip
```

It quickly gets out of hand and the ball moves way too fast. So let's set a max upper speed by adding `max_s` to our `ball` table and checking to see if `ball.s` is beyond it.

```
-- 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 fun about using the crank to hit the ball around! We're starting to get our first glimpses at the fun of joy making games.

## Score

Let's keep track of the score for each round and reset it to zero if the ball hits the left wall. We'll also reset the ball to its initial position, speed, and angle when the let wall is hit. We need a new variable, `score`, that we'll set and draw on the screen. We'll also keep track of the initial values we want to track for reseting the game.

```lua
-- 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

```

```
44  function movePaddle()
45      -- snip
46  end
47
48  function moveBall()
49      -- snip
50
51      if ball.x <= ball.r then
52          resetGame()
53      end
54
55      -- snip
56  end
57
58  function resetGame()
59      score = 0
60      ball.x = ball.initX
61      ball.y = ball.initY
62      ball.s = ball.initS
63      ball.a = ball.initA
64  end
65
66  -- snip
```

There's nothing too new here. Lots of using the fundamentals we already learned. Except for the assigning of `ball.initX`, `ball.initY`, etc. Basically what the code does is that after we create the `ball`, we take the values and assign them to a separate entry in the table. Because `ball.x`, `ball.y`, etc. are going to change throughout the game. We want to be able to easily reset to those initial values.

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.

```lua
-- 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

        -- snip
    end

    draw()
end

function draw()
    -- snip
end

function movePaddle()
    -- snip
end

function moveBall()
    -- snip

    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")
```

```
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. And we use that in the new `playSFX` function, which takes a note and plays it on 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. And the sound when the ball hits the paddle is also different.

**Bonus**: Similar to how we greeted a random name in the first chapter, make 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 a whole bunch of variables, and made something that's a little fun. Nice work.

Here's the finished code for this chapter:

```
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,
    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()
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 point to the circle center within the rectangle
    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))

    -- Distance between the circle 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
```

```
130 end
```

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

## Bonus Ideas

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

1. 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.
2. Add multiple balls that spawn after reaching certain score thresholds!
3. Add collectibles to aim for or even bricks to break.
4. Add pinball-style bumpers that the ball can hit.
5. Make the ball appear larger when it hits the paddle or a wall.
6. Shake the paddle when it's hit by the ball.
7. There's a small bug—sometimes the ball and paddle can get stuck on each other. How would you separate them when they collide based on the position of the ball when it hits the paddle?
8. 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!
9. Adding a mute setting would be nice!
10. Rework the paddle's movement to use acceleration and factor its current velocity into the angle the ball is bounced off at (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.

## Bonus Ideas

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.

## Bonus Ideas

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.

## Bonus Ideas

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.

## Bonus Ideas

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.

## Bonus Ideas

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.

## Leveling Up

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.

## Save Data

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

## Boss Battle

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

## Bonus Ideas

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 3

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

## Bonus Ideas

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.

# 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.

### 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.