Day 2: Getting Started with Raylib

Dylan <dylan@programvideogames.com> Tue, Jun 25, 2024 at 12:22 PM
To: brian.w.woodard@gmail.com

Hello there! Welcome back.

By the end of this lesson you will know how to:

- Get a window up and running
- Draw controllable rectangles
- Handle keyboard input

Getting a window up and running

Create a new file called pong.odin.

This time we'll go line by line with a link to the full file available afterwards.

First we'll declare our package: `package main`.

Next we want to import the raylib package that's available in the vendor resource: `import rl "vendor:raylib"`.

Naming the import "rl" makes it a bit easier to type (rather than raylib).

Yet again we need a main procedure to start our program `main :: proc...`

Inside main we'll need to call a couple of procedures from raylib to get it initialised.

`rl.InitWindow(1280, 720, "Pong")` - opens a window that we can use.
`rl.SetTargetFPS(60)` - limits the FPS to 60. This will cap the frame rate to 60 frames per second allowing a consistent movement speed.

Next we will add our main game loop.

The game loop is code that runs every frame.

In Odin there's only one loop keyword and that is `for`.

```
for !rl.WindowShouldClose() {
    rl.BeginDrawing()
    // ...
```

```
        rl.EndDrawing()
}
```

`rl.WindowShouldClose` - a raylib procedure that returns true if the user presses Escape or clicks the close button.

This loop will run until that procedure returns true.

`rl.BeginDrawing, rl.EndDrawing` - all our drawing code goes between these two calls.

At this point we can run the program and see that a window opens and we can press escape to close it.

> Pretty nice! Without using something like raylib this would be 150 lines of code or more.

```
odin run pong.odin -file
```

Drawing a rectangle

It's time to draw a rectangle that we can control with our keyboard.

```
rl.DrawRectangle(100, 100, 180, 30, rl.WHITE)
```

This procedure takes 5 arguments: x, y, width, height, colour.

> Often in 2D the top left is the origin. So, 100, 100 will be 100 pixels left and 100 pixels down.

Run the program again and you should see something like this:

Booyah! I dunno about you, but I'm always excited to see the first triangle or rectangle on the screen.

Moving the rectangle

At this point our rectangle's position is hard coded at 100, 100. Let's go back up to the start of main and create our first variable.

```
main :: proc () {
    pos: i32 = 100
    // ...
```

In Odin, there are a few ways to declare variables.

`:=` - declare a variable without a type to be inferred by the compiler.

`: <type> =` - explicitly declare a typed variable.

`::` - declare a constant. The value must be known at compile time and cannot be changed at run time.

We have explicitly declared a 32-bit integer `i32` because that is the type required by the raylib procedure `rl.DrawRectangle`.

Next, we need a way to detect if a key is pressed to modify it.

Luckily raylib has built-in input handling.

Back inside our game loop, before we draw anything, we will put our input handling code.

```
for !rl.WindowShouldClose() {
    if rl.IsKeyDown(rl.KeyboardKey.A) {
        pos -= 10 // X -= Y is X = X - Y
    }
    if rl.IsKeyDown(rl.KeyboardKey.D) {
        pos += 10
    }

    rl.BeginDrawing()

    rl.DrawRectangle(pos, 100, 180, 30, rl.WHITE)

    rl.EndDrawing()
}
```

Raylib has a few types of built in procedures for input. The IsSomethingDown variety will return true every frame the thing is held down. In this instance, we are checking a keyboard key which is available in the `rl.KeyboardKey` enum.

> An enum is a grouped set of values with an underlying associated number. Usually 0 to N - 1 where N is the number of values in the enum.

Run the program now and you should be able to move the rectangle left and right!

But... There's a problem. The rectangle is *marking* the screen as it moves along. That's not what we want.

To remedy this, we'll add a new procedure call just after the begin drawing procedure: `rl.ClearBackground(rl.BLACK)`

Run the program once more and the problem should be fixed.

Next steps

Note that in our program we have another Input→Processing→Output cycle:

Input: Keyboard keys, state (`pos`), rectangle values directly set (100, 180, 30, ...)

Process: Update `pos`

Output: Draw the rectangle at the new position

These cycles will continue to show up at every stage as we build on this example.

That's all for this lesson. Join me in the next one where we'll introduce another moving object, look at frame rate independence and figure out how to detect if objects are colliding.

If you have any questions, make sure to check out the Discord server.

Here is the full code at the end of this lesson:

```
package main

import rl "vendor:raylib"

main :: proc() {
	pos: i32 = 100

	rl.InitWindow(1280, 720, "Pong")

	for !rl.WindowShouldClose() {
		if rl.IsKeyDown(rl.KeyboardKey.A) {
			pos -= 10
		}
		if rl.IsKeyDown(rl.KeyboardKey.D) {
			pos += 10
		}

		rl.BeginDrawing()

		rl.ClearBackground(rl.BLACK)

		rl.DrawRectangle(pos, 100, 180, 30, rl.WHITE)

		rl.EndDrawing()
	}
}
```