Your Own Drawing Functions

This lesson removes earlier scaffolding from working with Reactors, having students brainstorm an original animation of their own and implement it from start-to-finish. This requires them to plan out what kind of data structure they will need, and how it will be drawn and updated.

Product Outcomes	• Students write the draw-state function for a reactor using a single number • Students write the draw-state function for a reactor using a state containing two numbers
Materials	• Single-Number State Starter File • Two-Number State Starter File • Slides are not yet available for this lesson • Printable Lesson Plan (a PDF of this web page)
Prerequisites	• Simple Data Types • Contracts • Simple Inequalities • Piecewise Functions and Conditionals • Compound Inequalities: Solutions & Non-Solutions • Introduction to Data Structures • Structures, Reactors, and Animations • Key Events • Refactoring

Drawing with a Single Number *30 minutes*

Overview

Students practice writing a simple function to draw the state of a Reactor, when that state consists of only a single number.

Launch

The majority of reactive programs you'll write in this course will use data structures consisting of multiple pieces of data, whether that be Numbers, Strings, Images, or Booleans. However, it's not *required* to have a full data structure in order to use a reactor. In fact, we can create an animation based on just a single number!

Open the Single-Number State Starter File file and take a look at the code. Before hitting "Run", can you guess what this code will do?

```
include image
include reactors

# next-state-tick :: Number -> Number
fun next-state-tick(n):
  n + 1
end

# draw-state :: Number -> Image
fun draw-state(n):

  "fix me!"

end

r = reactor:
  init: 1,
  # to-draw: draw-state,
  on-tick: next-state-tick
end

r.interact()
```

Notice how there is no `data` block in this file. Both the `next-state-tick` and the `draw-state` function consume a single number, and the initial value given to the reactor is also a single number (in this case, 1.)

Click "Run". What do you see?

According to the `next-state-tick` function, on every clock tick the state (a single number) will increase by one, which is exactly what happens. Since we didn't tell the reactor how to draw the state (the `to-draw` part of the reactor is commented out), when the reactor runs we see the state of the reactor (a single number) increasing, instead of an animation.

What do you think would happen if we had a reactor with a complete `draw-state` function, but a `next-state-tick` function that never updated the state? (Consuming and producing the same value.)

Reinforce the fact that, although the draw-state and next-state-tick functions work together within a reactor to produce an animation, each function can work without the other. In this example, next-state-tick will update the state even without a function to draw the state.

There are much more interesting things we can display using a number as the state of the reactor, however!

Investigate

Change the `draw-state` function so that it consumes a Number and produces an image. Then, uncomment the `to-draw: draw-state` line in the reactor to see an animation when the program runs!

Encourage students to brainstorm and share ideas for the draw-state function before beginning. Some possible options include:

- Drawing a star of size n (so that it gets larger on each tick)
- Display n as an image using the text function.
- Have students share back the draw-state functions they wrote.

Drawing with Two Numbers *30 minutes*

Overview

This activity turns up the cognitive load: students practice writing a function to draw the state of a Reactor, when that state consists of a structure containing two numbers.

Launch

You've practiced writing a `draw-state` function using a single number as a state, now let's look at something a bit more familiar.

Open the Two-Number State Starter File and take a look at the code.

```
include image
include reactors

data AnimationState:
  | state(
      a :: Number,
      b :: Number)
end

START = state(1, 100)

# next-state-tick :: AnimationState -> AnimationState
fun next-state-tick(s):
  state(s.a + 1, s.b - 1)
end

# draw-state :: AnimationState -> Image
fun draw-state(s):

  "fix me!"

end

r = reactor:
  init: START,
# to-draw: draw-state,
  on-tick: next-state-tick
end

r.interact()
```

This code includes a data structure (called `AnimationState`) containing two numbers as its fields, `a` and `b`. As before, the `draw-state` function is incomplete, and commented out from the reactor.

Based on the `next-state-tick` function defined here, what do you think will happen when you hit "Run"? Discuss with your partner, then try it out!

With only the `next-state-tick` function, we can see the state updating, increasing the first number by 1 and decreasing the second number by 1 each tick.

Investigate

How could you define a `draw-state` function to show something interesting when the program runs? Branstorm with your partner, then change the existing, broken `draw-state` function to consume an `AnimationState` and produce an image. Then, comment out the `to-draw: draw-state` line in the reactor to see an animation when the program runs!

Some possible ideas for this activity:

- Display two shapes of size a and b, which get larger and smaller, respectively, as the reactor runs.
- Make a and b the coordinates of an image, moving down and to the right across a background as the reactor runs.

Synthesize

Have students share back what they brainstormed before beginning, then share the completed draw-state functions they wrote, and the animations they created!