Chapter 14

Animations with Booleans

The Boolean type allows us to add some new features to our animations. Most obviously, we can use a Boolean as a model, just as we've already learned to use images, numbers, and strings as models. Unfortunately, doing anything interesting with a Boolean model requires *conditionals*, which are introduced in Chapter 15.

However, we can also use Booleans to *stop an animation*. This will give us practice using Booleans before we get to conditionals.

14.1 Stopping animations

Consider the following problem.

Worked Exercise 14.1.1 ***Develop an animation*** *that displays the word "antidisestablishmentarianism" in (say) 18-point blue letters. Every half second, the first letter of the word is removed, so the display becomes "ntidisestablishmentarianism", then "tidisestablishmentarianism",* etc.

Solution: Identify the model: Every second we need to chop one letter off the string, which we know how to do using `substring`, and we know how to convert a string to an image with `text`, so let's use a string as our model.

Identify the necessary handlers: Since the model isn't an image, we'll need a draw handler with the contract

```
; show-string :  string -> image
```

And since the model needs to change every half second, we'll need a tick handler with the contract

```
; chop-string :  string -> string
```

Write the handlers: The `show-string` function is straightforward; here's my answer.

```
; show-string :  string -> image
(check-expect (show-string "") (text "" 18 "blue"))
(check-expect (show-string "hello") (text "hello" 18 "blue"))
(define (show-string model)
  ; model      a string
  (text model 18 "blue"))
```

Once this works, we can go on to the `chop-string` function. Immediately we face a problem: it's not clear what should happen if its argument is the empty string.

```
; chop-string :  string -> string
(check-expect (chop-string "a") "")
(check-expect (chop-string "hello") "ello")
; (check-expect (chop-string "") what to do here?)
```

So we'll change the contract:

```
; chop-string :  non-empty string -> string
```

Now the problematic example is no longer legal, so we can forget about it — at least for now. The function definition looks like

```
(define (chop-string model)
  ; model      a string
  (substring model 1))
```

We test the function, and it works on all of our test cases.

Size and shape of window: If we don't specify dimensions, the animation window will be the size of the initial picture. Will that work? Well, the initial picture is simply the word "antidisestablishmentarianism" in an 18-point font, and as the word gets shorter, the picture can only get smaller, so it'll still fit in the original animation window; we don't need to specify window size.

Call `big-bang`: The initial model is "antidisestablishmentarianism", so ...

```
(big-bang "antidisestablishmentarianism"
          (check-with string?)
          (on-draw show-string)
          (on-tick chop-string 1/2))
```

This works beautifully until the string is reduced to empty, at which point ... **it crashes!** Remember, `chop-string` doesn't work on an empty string. To avoid this crash, we need a way to *stop the animation* before calling `chop-string` on an empty string.

The `picturing-programs` library provides one more kind of "handler" that we didn't see in Chapter 6: a "stopping condition". You'll write (or use) a function with the contract

```
; function-name :  model -> boolean
```

and install it with

```
(stop-when function-name)
```

The animation will stop as soon as the function returns `true`.

In our case, the model is a string, and we want to stop as soon as the string is empty, so here's my definition:

```
; empty-string?  :  string -> boolean
(check-expect (empty-string?  "") true)
(check-expect (empty-string?  "m") false)
(check-expect (empty-string?  "n") false)
(check-expect (empty-string?  "hello") false)
(define (empty-string?  model)
  ; model      a string
  ; ""         a fixed string
  (string=?  model ""))
```

Now we can use this to stop the animation as follows:

```
(big-bang "antidisestablishmentarianism"
          (check-with string?)
          (on-draw show-string)
          (on-tick chop-string 1/2)
          (stop-when empty-string?))
```

The animation works exactly as before, except that when the string is reduced to empty, it stops quietly and peacefully instead of showing an ugly error message. ∎

Note that `empty-string?` is called *after* the draw handler and *before* the tick handler. As a result, you'll see the string get gradually shorter, eventually down to nothing, but then the animation will stop before calling `chop-string` on the empty string (which, as we know, would crash).

Exercise 14.1.2 *Recall the animation of Exercise 10.2.1, which initially displayed an "a", then "ab" a second later, then "abb", and so on.* ***Modify*** *this animation so it stops when the string reaches ten letters long (*i.e. *"abbbbbbbbb").*

Exercise 14.1.3 ***Modify*** *the animation of Exercise 14.1.1 so that it stops when the string is reduced to length 3 or less, rather than when it's reduced to the empty string.* ***Try*** *it with several different initial strings, including some that are already length 3 or less.*

Exercise 14.1.4 ***Modify*** *the animation of Worked Exercise 6.4.1 so that the animation ends when the image displayed (including white space) is wider than the window.*

Exercise 14.1.5 ***Modify*** *the animation of Exercise 6.4.3 so that the animation ends when there's nothing left to show in the window.*

Exercise 14.1.6 ***Modify*** *the animation of Exercise 6.4.4 so that the animation ends when the image (a row of dots) is wider than the window.*

Exercise 14.1.7 ***Modify*** *the animation of Exercise 6.7.2 or 6.7.3 so that the animation ends when the image (including white space) is wider* or taller *than the window.*

Exercise 14.1.8 ***Modify*** *the animation of Exercise 8.3.1 or 8.3.2 so that the animation ends when the circle is wider or taller than the window.*

Hint: Recall that in these animations, the model was a number representing radius; the diameter is twice the radius. Be sure to test the program with a window that's wider than it is tall, or taller than it is wide.

Exercise 14.1.9 ***Modify*** *your animation from Exercise 8.4.2 so that the animation ends when the picture reaches the bottom of the window.*

Exercise 14.1.10 *The animation from Exercise 8.4.3 has a problem: it crashes if you type fast enough to reduce the radius of the dot to less than zero.* ***Modify*** *this animation so it never crashes, but rather stops gracefully when the radius gets too small.*

Exercise 14.1.11 *Your solution to Exercise 8.3.9 probably fills up the progress bar and then just sits there, not making any visible change to the window but not stopping either. (Or, if you did it differently, it may go* past *filling the progress bar, drawing a bar that's more than* 100% *full!)*

Modify *this animation so it stops gracefully when the progress bar reaches* 100%*.*

Exercise 14.1.12 ***Modify*** *the animation of Exercise 10.2.4 so that it counts only up to 30, and then stops.*

14.2 Stopping in response to events

A `stop-when` handler works great when the condition for stopping the animation is easily computed from the model. But in some situations, it's more natural to stop the animation in response to a tick, key, or mouse event, using a built-in function named `stop-with`.

```
; stop-with :  model -> stopping-model
; Returns the same thing it was given, but marked so that
; when big-bang sees this result, it stops the animation.
; The draw handler will be called one last time on this value.
```

Worked Exercise 14.2.1 ***Modify*** *exercise 10.2.5 so that it stops as soon as the user presses a key on the keyboard. (For now it's "any key"; we'll see in Chapter 18 how to tell if it was, say, the "q" key for "quit".)*

Solution: The model is the same as before: a number that starts at 0 and increases by 1 each second. We still need a tick handler (for the "increases by 1 each second"), and we still need a draw handler (to display the number at the appropriate place), but now we also need a key handler.

The key handler's contract is

```
; key handler stop-on-key :  number key -> number
```

Since we're not worried about *what* key the user pressed, we really only need one test case: if the key handler is called at all, it means the user pressed a key and the animation should end, by calling `stop-with` on some model. Whatever argument we give to `stop-with` will be given to the draw handler "one last time" to decide what picture should stay in the animation window after the animation stops. In many cases, we can just give `stop-with` the current model.

```
(check-expect (stop-on-key 24 "whatever")
              (stop-with 24))

(define (stop-on-key model key)
              ; model number
              ; key   whatever this is (ignore)
              (stop-with model)
              )

(big-bang 0
          (check-with number?)
          (on-tick add1 1)
          (on-draw show-num-at-x 500 100)
          (on-key stop-on-key))
```

Exercise 14.2.2 ***Choose*** *a previously-written animation that didn't have a key handler.* ***Modify*** *it so that it stops as soon as the user types a key.*

Exercise 14.2.3 ***Choose*** *a previously-written animation that didn't have a mouse handler.* ***Modify*** *it so that it stops as soon as the user moves or clicks the mouse.*

Common beginner mistake: `stop-with` is intended to be called from within a handler; it is *not* a way of installing handlers, like `on-tick`, `on-key`, or `stop-when`. If you call it as one of the arguments to `big-bang`, as in this example:

```
(big-bang 0
  (check-with number?)
  (on-tick add1)
  (on-draw disk-of-size)
  (stop-with 5)
  )
```

you'll get an error message saying that's not a legal part of a "world description" (remember, `big-bang` "creates the world").

In order to do more interesting things with `stop-with`, *e.g.* stop if the user clicks the mouse in a particular part of the screen, or stop if the user types the "q" key, we'll need the techniques of the next few chapters.

14.3 Review of important words and concepts

For situations in which we want an animation to end as soon as the model meets a certain condition, you can provide a `stop-when` handler — a function from model to Boolean — and as soon as it's true, the animation will end. Note that it is called *after* the draw handler, but *before* the tick, key, and mouse handlers. This means the draw handler has to be written to not crash even if the stopping condition is true, but the tick, key, and mouse handlers may safely assume that the stopping condition is false.

If you want to end the animation in response to a tick, mouse, or key event, it may be more natural to use `stop-with` from within a tick, mouse, or key handler.

Figure 14.1: Event handlers for animations

The big-bang function has the contract

```
; big-bang :  model(start) handler ... -> number
```

tick handlers must have the contract

```
; function-name :  model (old) -> model (new)
```

They are installed with `(on-tick function-name interval)`. The *interval* is the length of time (in seconds) between clock ticks; if you leave it out, the animation will run as fast as it can.

key handlers must have the contract

```
; function-name :  model (old) key -> model (new)
```

The "key" parameter indicates what key was pressed; we'll see how to use it in Chapter 18.

They are installed with `(on-key function-name)`.

mouse handlers must have the contract

```
; function-name :  model (old)
;                  number (mouse-x) number (mouse-y) event
;                  -> model (new)
```

The first parameter is the old model; the second represents the *x coordinate*, indicating how many pixels from the left the mouse is; the third number represents the *y coordinate*, indicating how many pixels down from the top the mouse is; and the "event" tells what happened (the mouse was moved, the button was pressed or released, *etc.*); we'll see in Chapter 18 how to use this.

They are installed with `(on-mouse function-name)`.

draw handlers must have the contract

```
; function-name :  model (current) -> image
```

and are installed with `(on-draw function-name width height)`. (If you leave out the *width* and *height* arguments, the animation window will be the size of the first image.)

An especially simple draw handler, `show-it`, is predefined: it simply returns the same image it was given, and it's useful if you need to specify the width and height of the animation window but don't want to write your own draw handler.

stop handlers must have the contract

```
; function-name :  model (current) -> boolean
```

and are installed with `(stop-when function-name)`.

model type checkers must have the contract

```
; function-name :  anything -> boolean
```

and are installed with `(check-with function-name)`.

If you're using numbers as your model, use `(check-with number?)`, if you're using strings, use `(check-with string?)`, *etc.*

It's a good idea to include a `check-with` clause in every `big-bang` animation to tell Racket (and future readers of your program) what type you think the model should be. If one of your functions returns the wrong type, you'll get a more informative and useful error message that tells you what went wrong.

14.4 Reference: Built-in functions for making decisions in animations

This chapter has introduced two new bits of syntax:

- `stop-when`
- `stop-with`

(Technically, `stop-with` is a function, while `stop-when` is a special form that works only inside `big-bang`.)