Chapter 10

Animations with arbitrary models

10.1 Model and view

In Chapter 8, we saw that an animation involves two pieces of information: a *model* (either an image or a number) and a *view* of that model (always an image). In fact, things are more flexible than that: the *model* can be of *any data type at all*, as long as you're consistent within a given animation. The details are in Figure 10.1.

Note that all the examples of chapters 6 and 8 still work. But now we can also use strings as models, and as we learn more data types in future chapters we'll be able to use those types as models too.

10.2 Design recipe for an animation, version 3

Our design recipe for an animation is now as in Figure 10.2.

Exercise 10.2.1 ***Write an animation*** *that initially displays the letter "a" in 18-point green type, and each second adds a "b" onto the end. So after one second it'll say "ab"; after two seconds "abb";* etc.

Hint: For this animation, your "model" should be a string, and your draw handler will involve the `text` function.

Exercise 10.2.2 ***Add a mouse handler*** *to the previous animation: every time the mouse is moved or clicked, one character will be chopped off the beginning of the string.*

Exercise 10.2.3 ***Write an animation*** *that initially displays the word `"cat"`. Each second, it inserts the letters `"xyz"` in the middle (*i.e. *between the first half and the second half) of the current word.*

Hint: It may be useful to write a "helper" function `insert-in-middle` that takes two strings, and inserts one of them into the middle of the other.

Figure 10.1: Event handlers for animations with arbitrary models

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.

To specify the model type , use `(check-with type-checker)`, where *type-checker* is a function that checks whether something is of a specified type, *e.g.* `image?`, `number?`, or `string?`, depending on what type you've chosen for this animation's model.

Figure 10.2: Design recipe for an animation, version 3

1. **Identify what handlers you'll need** (check-with, draw, tick, mouse, and/or key).

   - You should always have a `check-with` handler.
   - If your animation needs to change at regular intervals, you'll need a tick handler.
   - If your animation needs to respond to mouse movements and clicks, you'll need a mouse handler.
   - If your animation needs to respond to keyboard typing, you'll need a key handler.
   - You always need a draw handler. If your "model" is simply the image you want to show in the animation window, you can use `show-it`; otherwise you'll need to write your own.

2. **Decide what type** a "model" is and what it means.

   The model type should be something that you can easily update in response to events, and also something from which you can figure out what to show on the screen. Choosing an image as the model usually makes the draw handler easy to write, but may make the other handlers more difficult.

   For example, if your response to events is easily described by arithmetic, you probably want a numeric model. If it's easily described by image operations, you probably want an image model. If it's easily described by string operations, you probably want a string model.

   If you decide to use something other than an image as the model, you'll definitely need to write a draw handler.

3. **Write the contracts** for the handlers (using Figure 10.1). Again, the names of the functions are up to you, but once you've chosen a type for your model, the contracts must be exactly as in Figure 10.1.

4. **Develop** each of these functions, following the usual design recipe for each one. Don't go on to the next one until the previous one passes all of its test cases.

5. **Decide** on the initial value of the model.

6. **Decide on the width and height** (if the draw handler doesn't produce something of the right size).

7. **Decide on the time interval between "ticks"** (if you have a tick handler).

8. **Call** `big-bang` with the initial picture and handlers (specified using `check-with`, `on-draw`, `on-tick`, `on-mouse`, and `on-key`). See whether it works.

The following exercise is a step towards the "digital clock" we described earlier:

Exercise 10.2.4 ***Develop an animation*** *that displays a digital counter, in 18-point blue numerals. It should start at 0 and increase by 1 each second.*

Hint: Since the change every second is a *numeric* change — adding 1 — you should use a number as the model. But to display it on the screen, you'll need to turn the number into an image. Have you written a function that does this?

Exercise 10.2.5 ***Develop an animation*** *that displays a number that starts at 0 and increases by 1 each second, while* simultaneously *moving one pixel to the right each second. So, for example, after 24 seconds you should see the decimal number 24, 24 pixels from the left edge of the window.*

Exercise 10.2.6 ***Develop an animation*** *that, at all times, shows the mouse's coordinates as an ordered pair in the animation window. For example, if the mouse were currently 17 pixels from the left and 43 pixels down from the top, the screen would show (17, 43)*

10.3 Review of important words and concepts

Interactive programs are generally written following the *model/view framework*: the model changes in response to events, and the view is computed from the model. The model in an animation may be of *any data type you choose*, as long as you pick a type and stick to it consistently for all the relevant handlers.

10.4 Reference

There are no new built-in functions or syntax rules in this chapter, but some previously-defined functions have broader contracts than you knew about before; see Figure 10.1.