

MAU11601
Lecture 05
2026-10-01

AoC 2025 Day 1

I'm not going to try to be super efficient here; this is more about illustrating what we've learned so far.

In particular, we'll try to do this as an interactive graphical programme rather than a batch program.

What kind of events do we have?

- keyboard input, to enter the rotation data
- clock ticks, to move the dial

There's no need for the mouse, so the functions we need to write are the ones for the `to-draw`, `on-key` and `on-tick` parameters to `big-bang`, but not `on-mouse`.

What information should be in our world state?

- the current dial position
- a running count of the number of zeroes seen
- the rotation direction
- the number of clicks remaining in the current rotation
- the part of the number entered so far

The draw function

First we need to define our structure.

```
(define-struct world (points-to count direction clicks-left nsf))
```

We don't need to tell Racket the types, but we should know what they are. We can take them all to be integers. `direction` will be +1 for right and -1 for left, representing the change in `points-to` on each click.

It's good to start with our draw function, so we can do visual debugging. The scene is divided into an unchanging background and a changing foreground.

What belongs in the background?

- circles for the centre (solid) and rim (outline) for the dial
- tick marks for the positions
- numbers?

What belongs in the foreground?

- the arrow
- the running count

The background

We start with the background. We'll build it up step by step.

```
(define r1 5)    (define r2 10)
```

```
...
```

```
(define r8 140) (define r10 200)
```

```
(define b0 (empty-scene (* 2 r10) (* 2 r10)))
```

```
(define b1 (place-image (circle r1 'solid 'black) r10 r10 b0))
```

```
(define b2 (place-image (circle r8 'outline 'black) r10 r10 b1))
```

We define the radii of various circles, in case we want to adjust them later, and then b0, b1 and b2 are scenes with no circles, just the inner circle, and both the inner and outer circles.

Then we define a place-mark function (omitted), such that (place-mark n s) adds the n'th mark to s. once we have that, the full background is

```
(define background (foldl place-mark b2 (range 0 100 1))).
```

(range a b h) gives an arithmetic progression starting from a, finishing before b, with step size h.

The foreground

Our draw function should draw the foreground, the arrow and the running count, on our fixed background.

```
(define (draw w)
  (place-count (world-count w)
               (place-arm (world-points-to w) background)))
```

place-arm is similar to our earlier (omitted) place-mark function. In fact we can define both of them in terms of a more general place-ray function, which draws the portion of a ray from the origin lying between two circles.

```
(define (place-arm n s) (place-ray r2 r3 n s))
(define (place-count n s)
  (place-image (text (number->string n) font-size 'black)
               r10
               (- r10 r9)
               s))
```

Clock ticks

On clock ticks we need to do a number of things:

- decrement the `clicks-left` field, if it's positive.
- update the `points-to` field, incrementing or decrementing according to the `direction` field.
- increment the `count` field if this is the last tick and the arm ends up pointing to 0

We'll ignore the last one for now.

```
(define (tick-update w)
  (if (positive? (world-clicks-left w))
      (make-world (+ (world-points-to w) (world-direction w))
                  (world-count w)
                  (world-direction w)
                  (- (world-clicks-left w) 1)
                  (world-nsf w))
      w))
```

We can, and should, test this before proceeding.

Clock ticks, continued

We can test it just by evaluating it on some possible world states, e.g.

`(tick-update (make-world 50 0 -1 3 0))`, or visually using `big-bang`, e.g.

`(big-bang (make-world 50 0 1 120 0)`

`(name "test") (to-draw draw) (on-tick tick-update))`

We still need to update the count on the last click. One way to do this is to change the else branch of the `if` expression, currently just `w`, to

`(if (zero? (world-clicks-left w))`

`(make-world (world-points-to w)`

`(if (zero? (world-points-to w))`

`(+ (world-count w) 1)`

`(world-count w))`

`0 ; direction`

`-1 ; clicks-left`

`0) ; nsf`

`w)`

Keyboard input

What we do on keyboard input depends on the key pressed.

- On L or R we set `direction` to `-1` or `1` and set `nsf` to `0`.
- On a digit we multiply `nsf` by `10` and add that digit to it.
- On a carriage return we set `clicks-left` to `nsf`.
- Other keys should just be ignored.

We can do all of this with a big `cond` statement.

```
(define (key-update w k)
  (cond [(key=? k "l") (set-direction w -1)]
        [(key=? k "r") (set-direction w 1)]
        [(key=? k "0") (new-digit w 0)]
        ...
        [(key=? k "9") (new-digit w 9)]
        [(key=? k "\r") (make-world ... (world-nsf w) 0)]
        [else w]))
```