UMass Boston Computer Science

CS450 High Level Languages (section 2)

More High-Level Features

Wednesday, September 25, 2024

Logistics

- HW 3 out
  - due: Mon 9/30 12pm (noon) EST

HW 2 review

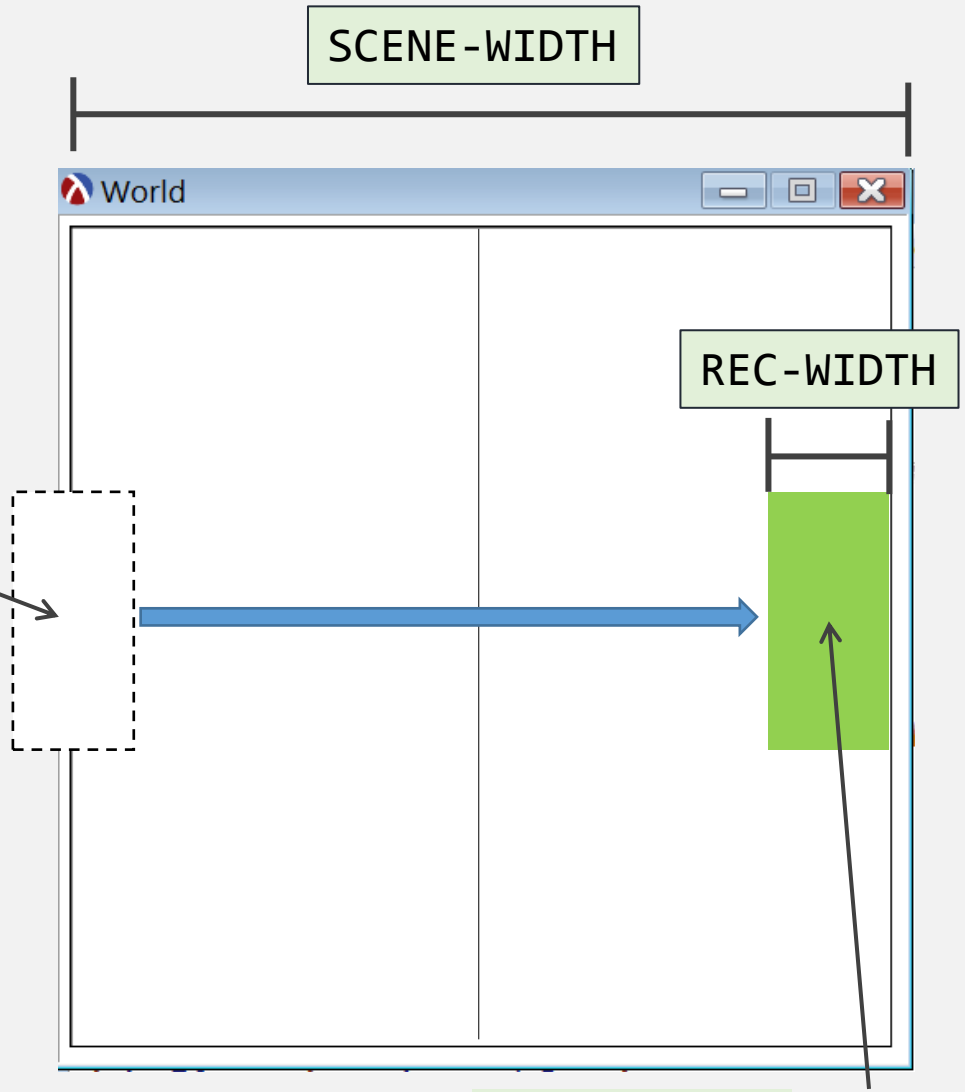
```
;; A WorldState is an Int
;; Interp?
```

```
;; x coord of rec center
```

```
(define INIT-WORLD 0)
```

```
;; render : WorldState -> Image
(define (render w)
  (place-image
   REC-IMG
   ws REC-Y
   EMPTY-SCENE))
```

```
;; render : WorldState -> Image
(define (next-WorldState w)
  (+ w 2))
```



WorldState = SCENE-WIDTH – REC-WIDTH/2

HW 2 review

```
;; A WorldState is an Int
;; Interp:
;; x coord of rec center
;; of lead rec in pair of
;; rec SCENE-WIDTH apart
```

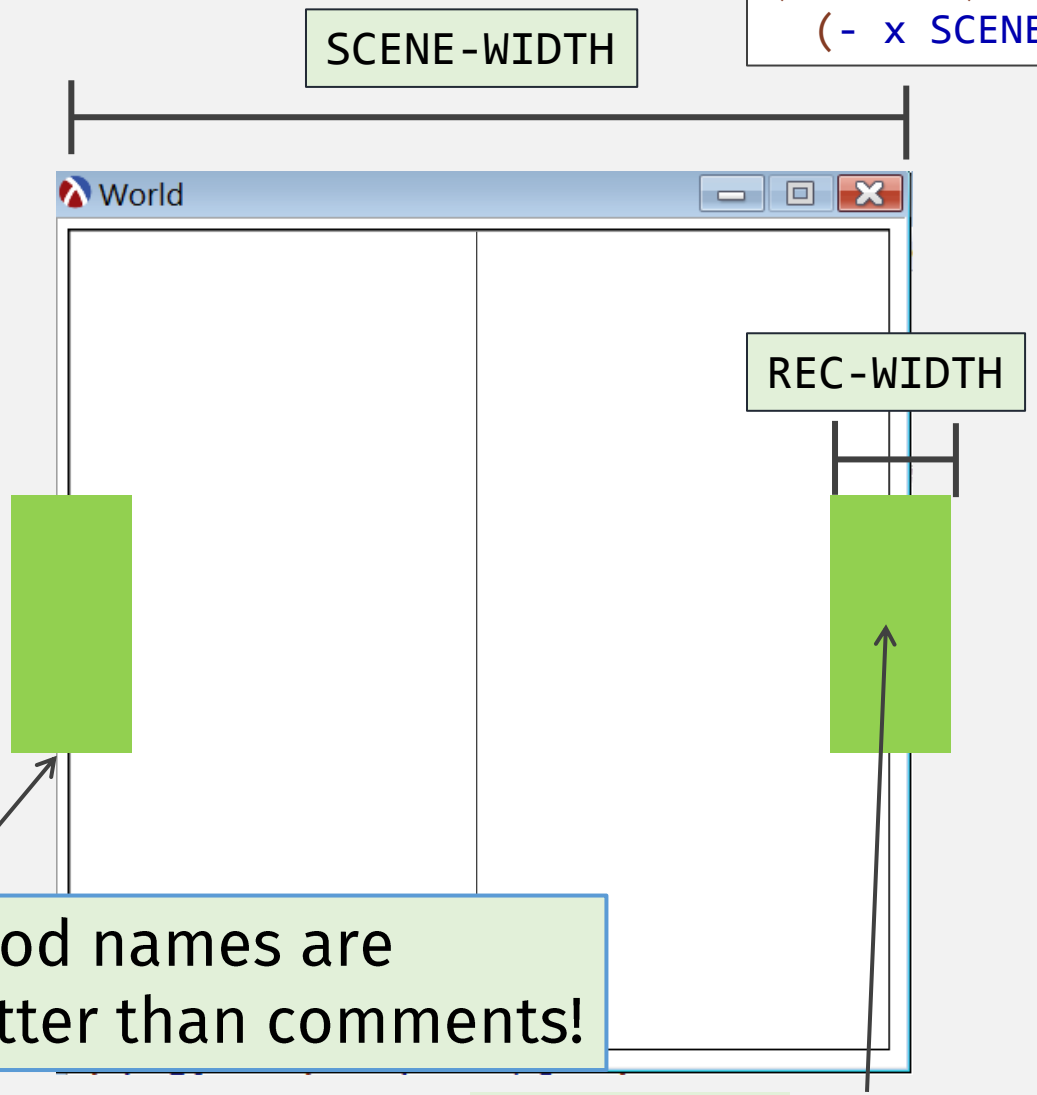
???

```
;; render : WorldState -> Image
(define (render w)
  (place-image
   REC-IMG ;; lead rec
   ws REC-Y
   (place-image
    REC-IMG ;; trail rec
    (lead->trail-rec ws) REC-Y
    EMPTY-SCENE))
  (+ w 2))
```

Readable?

Good names are better than comments!

```
;; lead->trail-rec:
;;   WorldState -> WorldState
(define (lead->trail-rec x)
  (- x SCENE-WIDTH))
```



WorldState = SCENE-WIDTH ???

HW 2 review

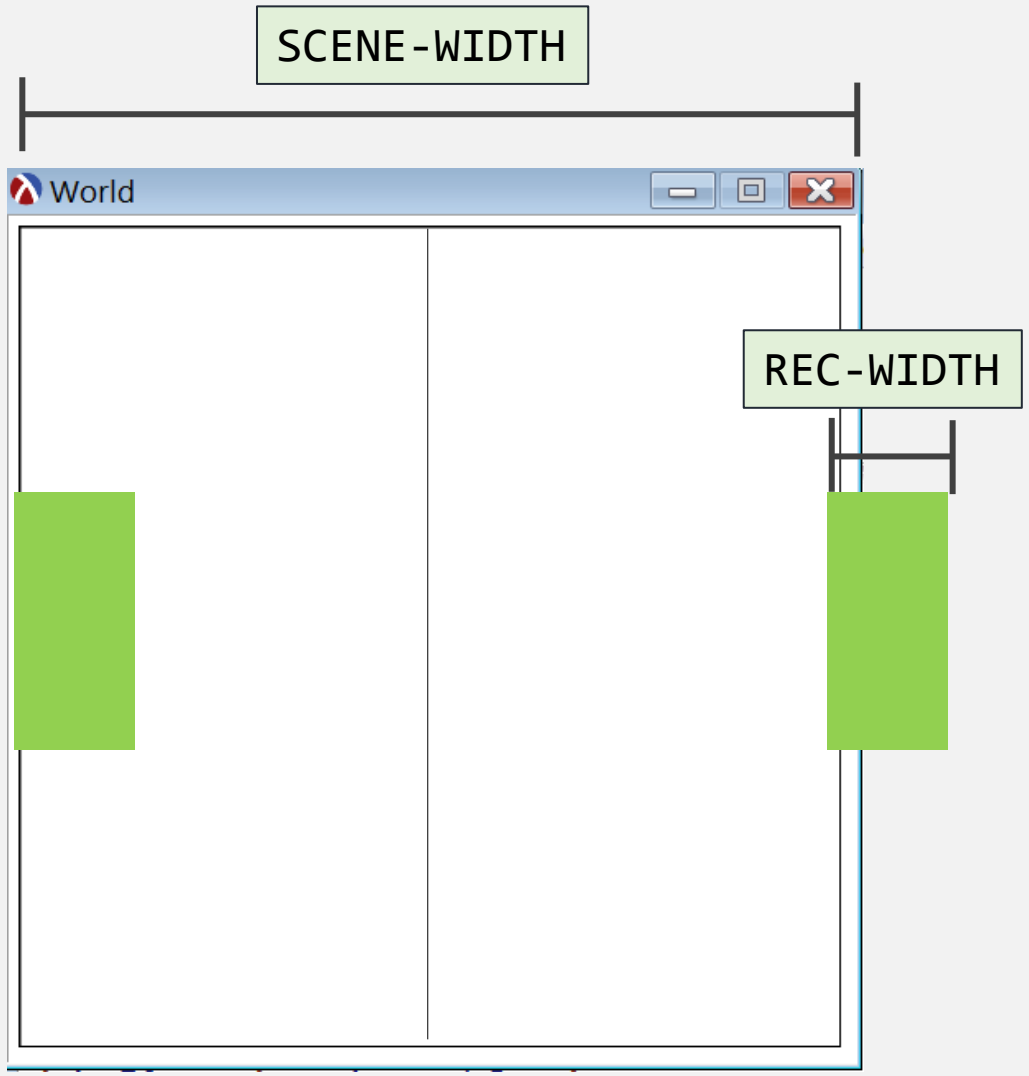
```
;; A WorldState is an Int
;; Interp:
;; x coord of rec center
;; of lead rec in pair of
;; rec SCENE-WIDTH apart
```

```
;; render : WorldState -> Image
(define (render w)
  (place
   REC-I
   ws RE
   (place-imag
    REC-IMG
```

Does this work??

NO! (because we wrote tests)

```
;; render : WorldState -> Image
(define (next-WorldState w)
  (modulo (+ w 2) SCENE-WIDTH))
```



???

WorldState =

SCENE-WIDTH – REC-WIDTH/2 -2

HW 2 review

```
;; A WorldState is an Int
;; Interp:
;; x coord of rec center
;; of lead rec in pair of
;; rec SCENE-WIDTH apart
```

```
;; render : WorldState -> Image
(define (render w)
  (place
   REC-I
   ws RE
   (place-image
     REC-IMG
```

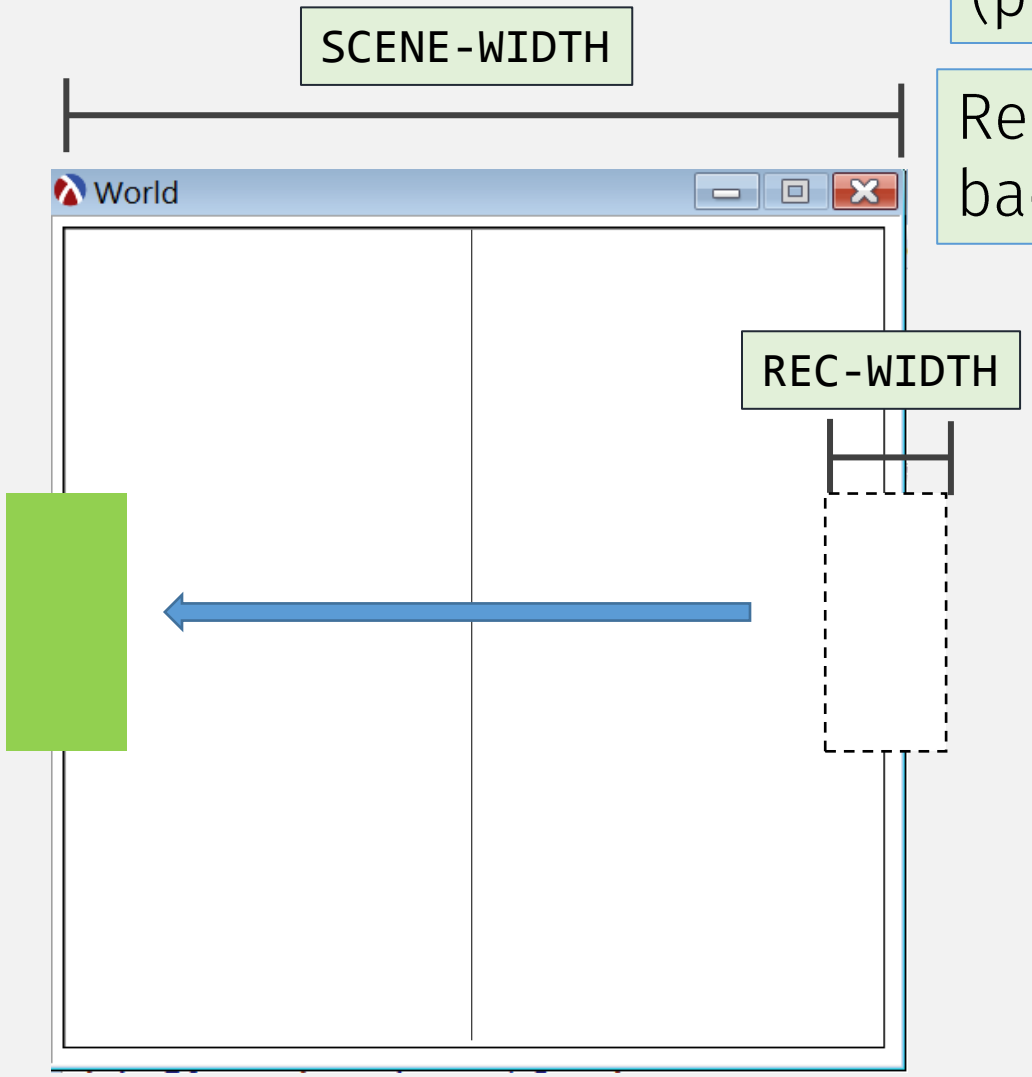
Does this work??

NO!

How to fix?

Iterate!

```
;; render : WorldState -> Image
(define (next-WorldState w)
  (modulo (+ w 2) SCENE-WIDTH))
```



(possible test)

Rec jumps back too early!

???

WorldState = SCENE-WIDTH

Program Design Recipe

… is iterative!

1. **Data Design**

2. **Function Design**

Function Design Recipe

… is iterative!

1. **Name**
2. **Signature** – types of the **function input(s)** and **output**
3. **Description** – explain (in **English prose**) the **function behavior**
4. **Examples** – show (using `rackunit`) the **function behavior**
5. **Template** – sketch out the **function structure** (using input's **Data Definition**)
6. **Code** – implement the **rest of the function** (arithmetic)
7. **Tests** – check (using `rackunit`) the **function behavior**

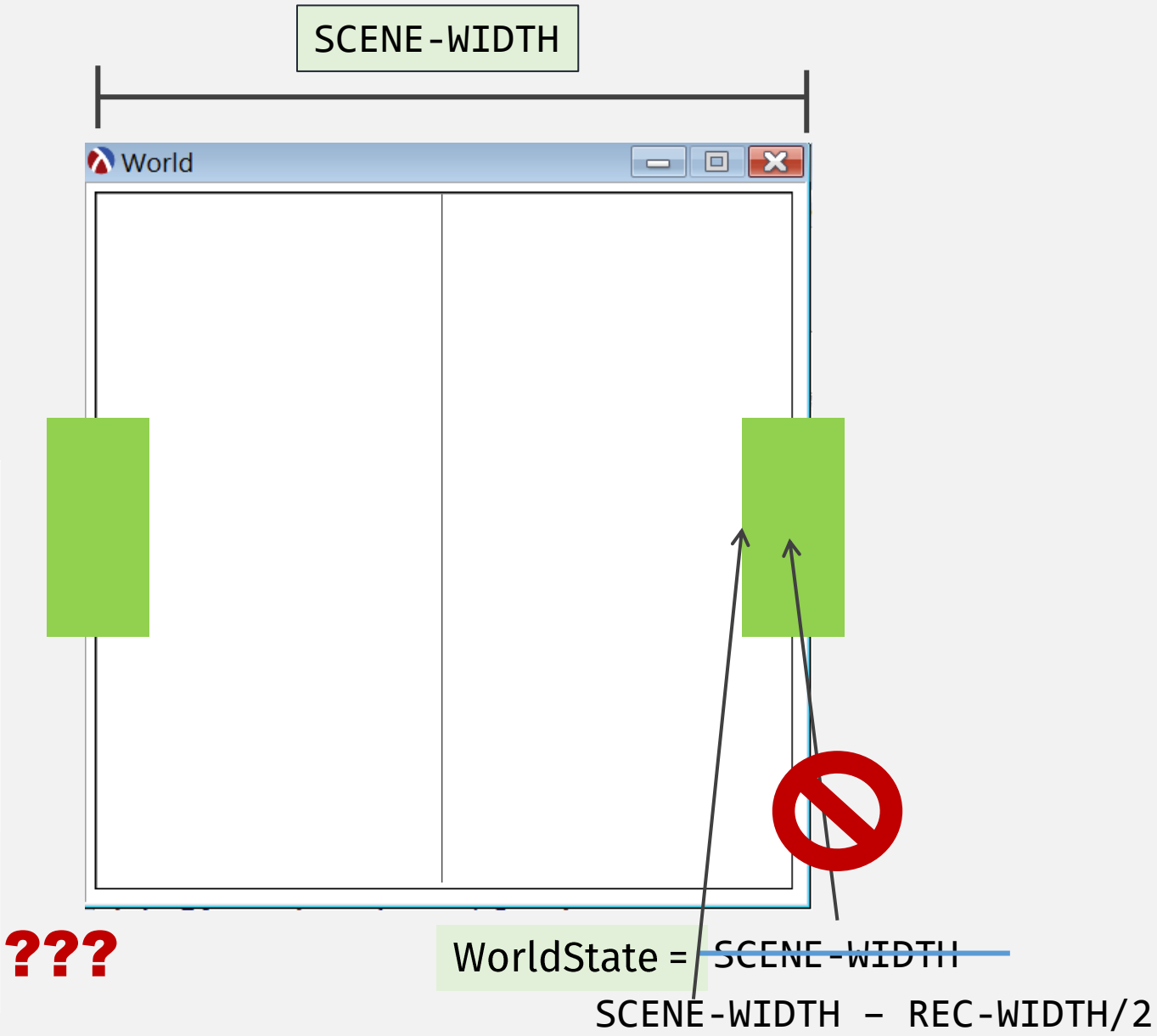
HW 2 review

(Iteration #3)

```
;; A WorldState is an Int
;; Interp:
;; x coord of rec LEFT
;; of lead rec in pair of
;; rec SCENE-WIDTH apart
```

```
;; render : WorldState -> Image
(define (render w)
  (place-image
   REC-IMG
   ws REC-Y
   (place-image
     REC-IMG
```

```
;; render : WorldState -> Image
(define (next-WorldState w)
  (modulo (+ w 2) SCENE-WIDTH))
```



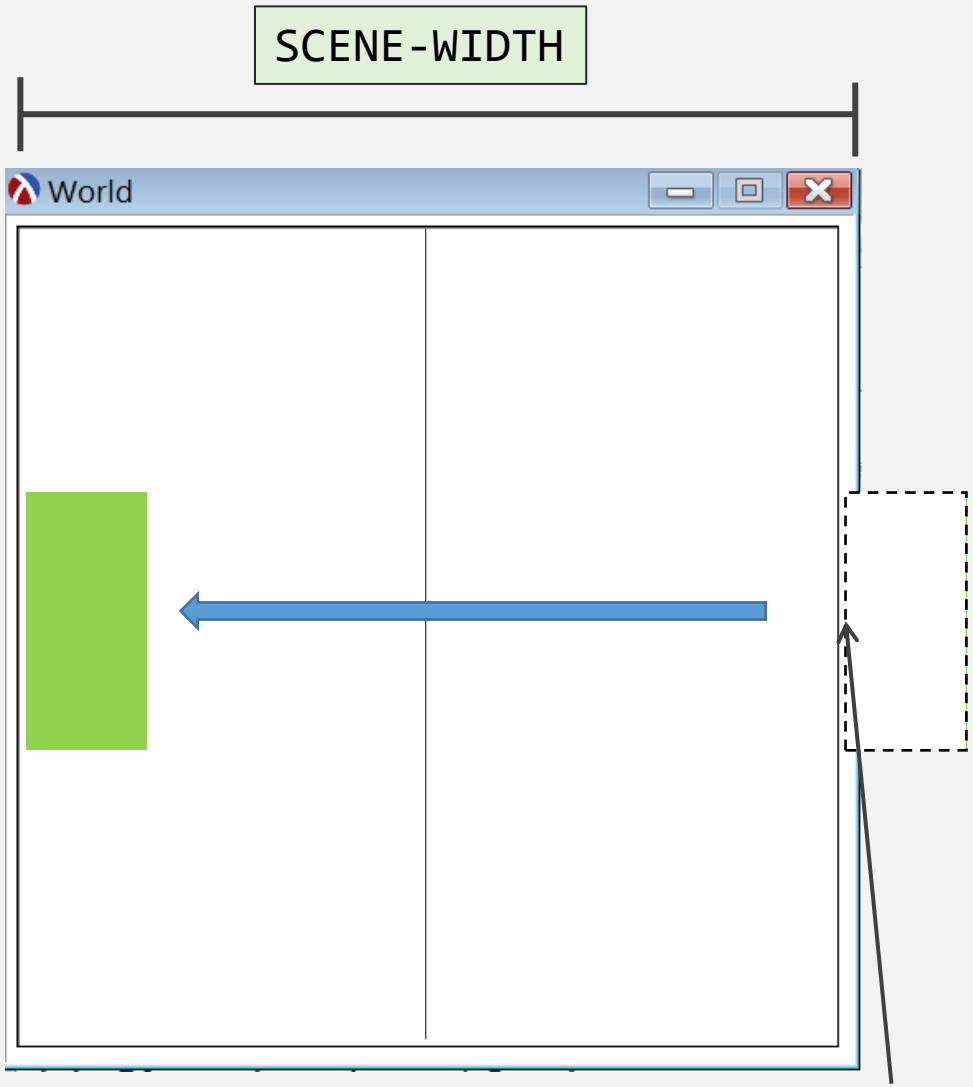
HW 2 review

(Iteration #3)

```
;; A WorldState is an Int
;; Interp:
;; x coord of rec LEFT
;; of lead rec in pair of
;; rec SCENE-WIDTH apart
```

```
;; render : WorldState -> Image
(define (render w)
  (place-image
   REC-IMG
   ws REC-Y
   (place-image
     REC-IMG
```

```
;; render : WorldState -> Image
(define (next-WorldState w)
  (modulo (+ w 2) SCENE-WIDTH))
```



WorldState = SCENE-WIDTH

HW 2 review

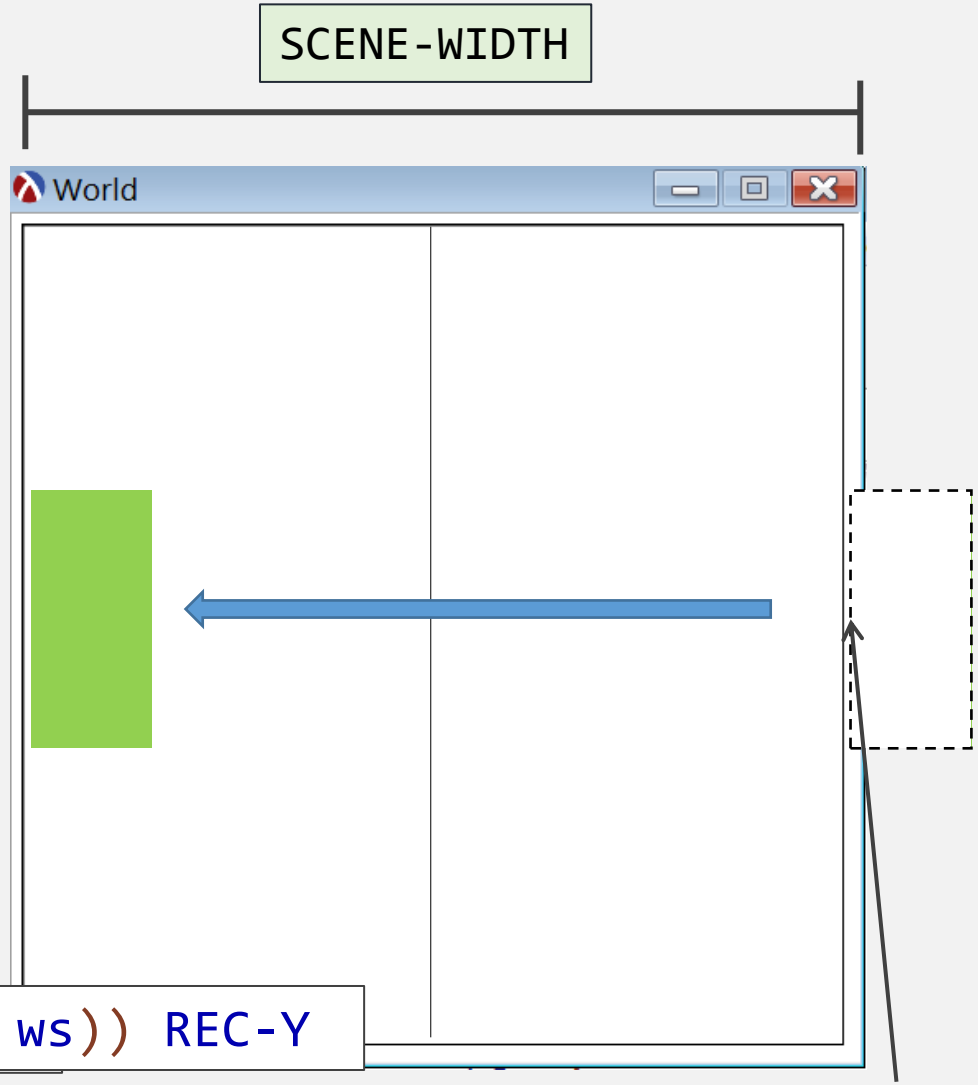
(Iteration #3)

```
;; A WorldState is an Int
;; Interp:
;; x coord of rec LEFT
;; of lead rec in pair of
;; rec SCENE-WIDTH apart
```

```
;; render : WorldState -> Image
(define (render w)
  (place-image
   REC-IMG
   (left->center-x ws) REC-Y
   (place-image
     REC-IMG
     (lead->trail-rec (left->center-x ws)) REC-Y
     EMPTY-SCENE))
```

Readable?

Must be center x!



WorldState = SCENE-WIDTH

Program Design Recipe

… is iterative!

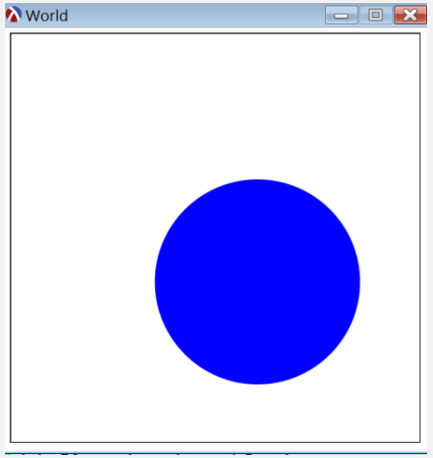
1. **Data Design**

2. **Function Design**

Last Time

```
;; A Coord is a Real
;; Represents x or y position on big-bang canvas
```

```
;; A WorldState is a (make-world [x : Coord] [y : Coord])
;; where:
;; x : represents x coordinate of ball center
;; y : represents y coordinate of ball center
```



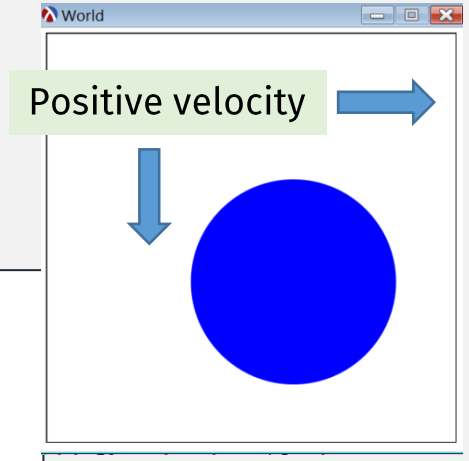

```
;; mouse-handler : WorldState Coordinate Coordinate MouseEvent -> WorldState
;; Sets the WorldState to be the current mouse location
(define (mouse-handler w x y evt)
  (world x y))
```

Let's allow ball to move on its own

```
(check-equal? (mouse-handler (world 1 1) 0 0 "button-down")
              (world 0 0))
```

```
;; A Velocity (Vel) is an Int in [0,10)
;; represents pixels/tick
;; positive = down or right
;; negative = up or left
```

```
;; A WorldState is a
;;  (make-world [x : Coord][y : Coord][xv : Vel][yv : Vel]
;; where:
;; x : represents x coordinate of ball center
;; y : represents y coordinate of ball center
;; xv : velocity in x direction
;; yv : velocity in y direction
```




```
;; WorldState TEMPLATE
;; world-fn : WorldState -> ???
```

```
(define (world-fn w)
  .... (world-x w) .... (world-y w) ....
  .... (world-xvel w) .... (world-yvel w) ....)
```

(extract pieces of compound data … to be combined with arithmetic)

```
;; A WorldState is a
;;  (make-world [x : Coord][y : Coord][xv : Vel][yv : Vel]
;; where:
;; x : represents x coordinate of ball center
;; y : represents y coordinate of ball center
;; xv : velocity in x direction
;; yv : velocity in y direction
```

```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (make-world
    (+ (world-x w) (world-xv w))
    (+ (world-y w) (world-yv w))
    (world-xv w)
    (world-yv w)))
```

```
(check-equal?
  (next-world
    (make-world 2 2 1 1))
  (make-world 3 3 1 1)))
```

Add velocity to pos

Repeated code
(not that bad, but let's see some ways to remove it)

let

```
;; A WorldState is a
;;  (make-world [x : Coord][y : Coord][xv : Vel][yv : Vel]
;; where:
;; x : represents x coordinate of ball center
;; y : represents y coordinate of ball center
;; xv : velocity in x direction
;; yv : velocity in y direction
```

```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (let ([x (world-x w)]
        [y (world-y w)]
        [xv (world-xv w)]
        [yv (world-yv w)])
    (make-world (+ x xv) (+ y yv) xv yv)))
```

Extract all compound data pieces first, before doing “arithmetic”

```
(let ([id val-expr] ...) body ...+)
```

Defines new local variables

in scope only in the body

Local variables **shadow** previously defined vars

Internal **defines** (equiv to **let**)

```
;; A WorldState is a
;;  (make-world [x : Coord][y : Coord][xv : Vel][yv : Vel]
;; where:
;; x : represents x coordinate of ball center
;; y : represents y coordinate of ball center
;; xv : velocity in x direction
;; yv : velocity in y direction
```

```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (define x (world-x w))
  (define y (world-y w))
  (define xv (world-xv w))
  (define yv (world-yv w))
  (make-world (+ x xv) (+ y yv) xv yv)))
```

Extract all compound data pieces first, before doing “arithmetic”

(is there an easier way to do this?)

Pattern Matching!

```
;; A WorldState is a
;;  (make-world [x : Coord][y : Coord][xv : Vel][yv : Vel]
;; where:
;; x : represents x coordinate of ball center
;; y : represents y coordinate of ball center
;; xv : velocity in x direction
;; yv : velocity in y direction
```

```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (match-define (world x y xv yv) w)



  (make-world (+ x xv) (+ y yv) xv yv)))
```

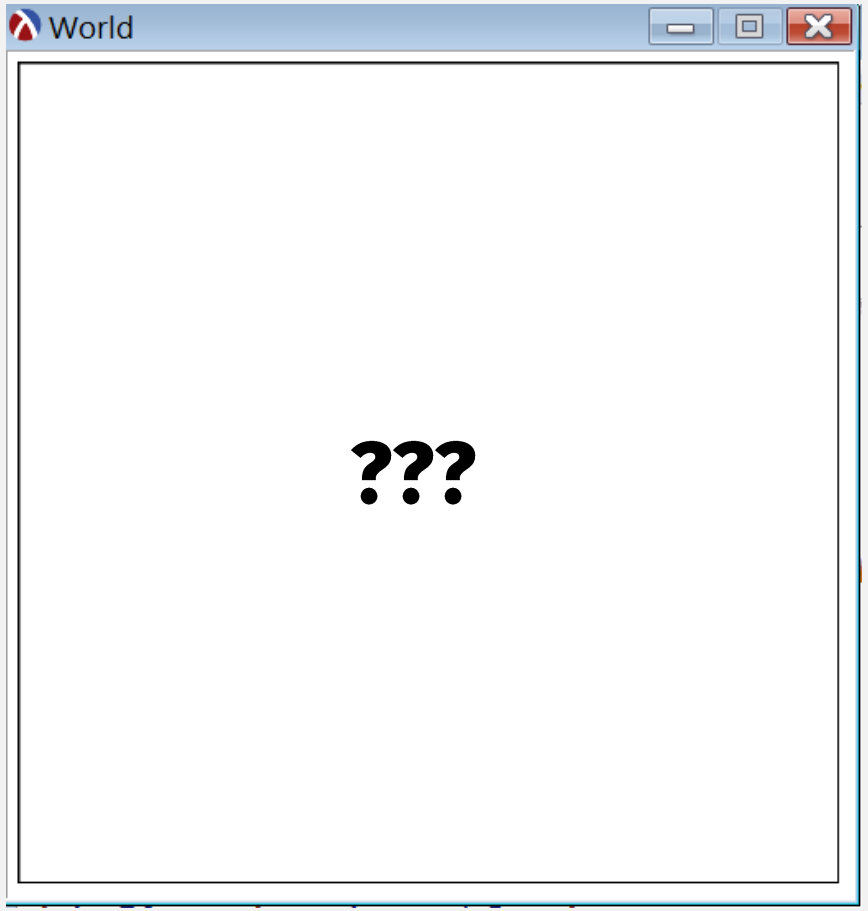
Extract all compound data pieces, at the same time!

Make it bounce?

```
;; A WorldState is a
;;  (make-world [x : Coord][y : Coord][xv : Vel][yv : Vel]
;; where:
;; x : represents x coordinate of ball center
;; y : represents y coordinate of ball center
;; xv : velocity in x direction
;; yv : velocity in y direction
```

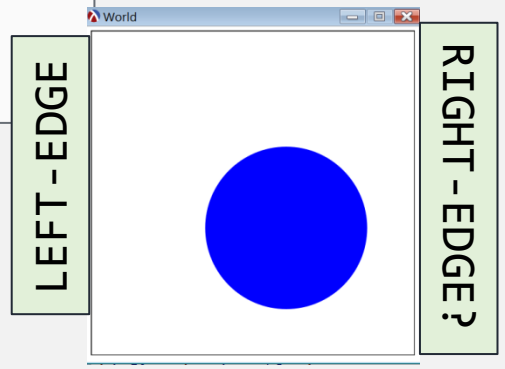
```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (match-define (world x y xv yv) w)




  (make-world (+ x xv) (+ y yv) xv yv)))
```

Let’s see what our animation looks like …



Make it bounce?

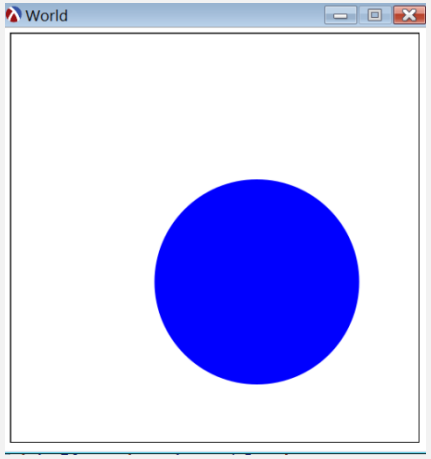
```
;; A WorldState is a
;;  (make-world [x : Coord][y : Coord][xv : Vel][yv : Vel]
;; where:
;; x : represents x coordinate of ball center
;; y : represents y coordinate of ball center
;; xv : velocity in x direction
;; yv : velocity in y direction
```




```
;; next-world : WorldState -> WorldState
;; Computes the next ball pos
(define (next-world w)
  (match-define (world x y xv yv) w)
  (define new-xv ???)

  (make-world (+ x xv) (+ y yv) new-xv yv)))
```

In-class exercise: more `big-bang` practice

- Create a `big-bang` program with a “ball”
- Design `WorldState` so it can
  - move in both x and y directions …
  - And have x and y velocities!
- Make the ball bounce when it hits a “wall”



<u>Submitting</u>

1. File: `in-class-09-25-<Lastname>-<Firstname>.rkt`
2. Join the in-class team: `cs450f24/teams/in-class`
3. Commit to repo: `cs450f24/in-class-09-25`
   - (May need to `merge/pull + rebase` if someone `push`es before you)