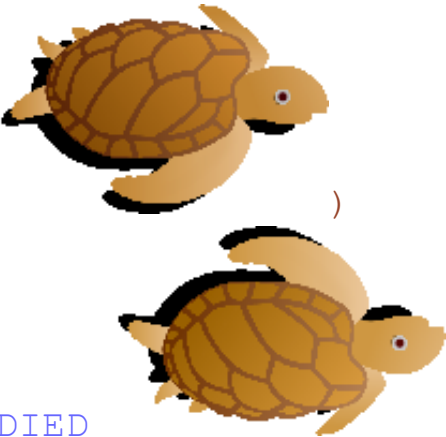


```

;;;;;;;;;;;;; VZ proj ;;;;;;;;;;;;;;
;;;                Yohei Yasukawa                ;;;
;;;;;;;;;;;;;

;; Global Definitions
(define BG_WIDTH 500)
(define BG_HEIGHT 500)
(define BACKGROUND (empty-scene BG_WIDTH BG_HEIGHT))
(define EMPTY_SCENE (empty-scene 0 0))
(define-struct stat (turtle lbug dog))


;;;;;;;;;;;;;
;; Turtle ;;
;;;;;;;;;;;;;
;; Definitions
(define MAX_FTIME 300)



(define T_IMG )



(define T_IMG_DIED )
(define-struct turtle (ftime))
(define INIT_TURTLE (make-turtle MAX_FTIME))
(define INIT_TURTLE-1 (make-turtle (- MAX_FTIME 1)))

;; Definitions for examples
(define turtle-300-ftime (make-turtle 300))
(define turtle-299-ftime (make-turtle 299))
(define turtle-0-ftime (make-turtle 0))
(define turtle-died (make-turtle -1))

;; create-t-fmeter : ftime -> image
; create an image of feed meter by a given time.
(define (create-t-fmeter ftime)
  (rectangle (/ ftime 3) 20 "solid" "red"))
;Examples
(check-expect (create-t-fmeter 300) )

```

```

(check-expect (create-t-fmeter 200) )
(check-expect (create-t-fmeter 100) )

; turtle-tick : TurtleStatus -> TurtleStatus
; calculates the state following the given state if only
time passes
(define (turtle-tick current)
  (cond
    [(< (turtle-ftime current) 0)
     (make-turtle (- 1))]
    [else
     (make-turtle (- (turtle-ftime current) 1))])
  ))
(check-expect (turtle-tick turtle-300-ftime)
turtle-299-ftime)
(check-expect (turtle-tick turtle-0-ftime) turtle-died)
(check-expect (turtle-tick turtle-died) turtle-died)

; turtle-key : TurtleStatus KeyEvent -> TurtleStatus
; calculates the state following the given state if given
key is pressed
(define (turtle-key current key)
  (cond
    [(and (>= (turtle-ftime current) 0) (string=? key "z"))
     (make-turtle MAX_FTIME)]
    [else current]))
(check-expect (turtle-key turtle-300-ftime "z")
turtle-300-ftime)
(check-expect (turtle-key turtle-0-ftime "z")
turtle-300-ftime)
(check-expect (turtle-key turtle-died "z") turtle-died)

; turtle-render : TurtleStatus -> image
; constructs an turtle image representing the given state
(define (turtle-render current)
  (cond
    [(>= (turtle-ftime current) 0)
     (overlay/xy T_IMG
                 0 60
                 (create-t-fmeter (turtle-ftime current)))]
    [else T_IMG_DIED]))

```

```
(check-expect (turtle-render turtle-300-ftime)
```



```
    )  
(check-expect (turtle-render turtle-died) T_IMG_DIED)
```

```
;;;;;;;;;;;;;  
;; Lightning Bug ;;  
;;;;;;;;;;;;;
```

```
; Definitions for lbug
```



```
(define LB_LEFT_ON_IMG )
```



```
(define LB_LEFT_OFF_IMG )
```



```
(define LB_RIGHT_ON_IMG )
```



```
(define LB_RIGHT_OFF_IMG )
```

```
(define-struct lbug (posx dir lighton?))
```

```
(define INIT_LBUG (make-lbug 50 "left" true))
```

```
(define INIT_LBUG-1 (make-lbug 49 "left" true))
```

```
; Definitions for lbug examples
```

```
(define lbug-size 100)
```

```
(define lbug-edgeL 0)
```

```
(define lbug-edgeR (- BG_WIDTH lbug-size))
```

```
(define lbug-left-10 (make-lbug 10 "left" true))
```

```
(define lbug-left-9 (make-lbug 9 "left" true))
```

```
(define lbug-left-edgeL (make-lbug lbug-edgeL "left" true))
```

```

(define lbug-right-edgeL (make-lbug lbug-edgeL "right"
true))
(define lbug-right-10 (make-lbug 10 "right" true))
(define lbug-right-11 (make-lbug 11 "right" true))
(define lbug-right-edgeR (make-lbug lbug-edgeR "right"
true))
(define lbug-left-edgeR (make-lbug lbug-edgeR "left" true))

; lbug-tick : LBugStatus -> LBugStatus
; calculates the state following the given state if only
time passes
(define (lbug-tick current)
  (cond
    [(string=? (lbug-dir current) "left")
     (if (touch-left-wall? current)
         (face-opposite current); if touched left wall
         (move-left current)
         )])
    [(string=? (lbug-dir current) "right")
     (if (touch-right-wall? current)
         (face-opposite current); if touched right wall
         (move-right current)
         )])
    ))
(check-expect (lbug-tick lbug-left-10) lbug-left-9)
(check-expect (lbug-tick lbug-left-edgeL) lbug-right-edgeL)
(check-expect (lbug-tick lbug-right-10) lbug-right-11)
(check-expect (lbug-tick lbug-right-edgeR) lbug-left-edgeR)

;; touch-left-wall? : LBugStatus -> boolean
; determine if a given lightning bug is touching a wall on
the left
(define (touch-left-wall? current)
  (cond
    [(<= (lbug-posx current) lbug-edgeL) true]
    [else false]
    ))
(check-expect (touch-left-wall? lbug-left-edgeL) true)
(check-expect (touch-left-wall? lbug-left-10) false)

;; touch-right-wall? : LBugStatus -> boolean

```

```

; determine if a given lightning bug is touching a wall on
the right
(define (touch-right-wall? current)
  (cond
    [(>= (lbug-posx current) lbug-edgeR) true]
    [else false]
  ))
(check-expect (touch-right-wall? lbug-right-10) false)
(check-expect (touch-right-wall? lbug-right-edgeR) true)

;; move-left : LBugStatus -> LBugStatus
; move a given lightning bug to the left in 1 px
(define (move-left current)
  (light-random
    (make-lbug (- (lbug-posx current) 1)
      (lbug-dir current) (lbug-lighton? current))))
(check-expect (move-left lbug-left-10) lbug-left-9)

;; move-right : LBugStatus -> LBugStatus
; move a given lightning bug to the right in 1 px
(define (move-right current)
  (light-random
    (make-lbug (+ (lbug-posx current) 1)
      (lbug-dir current) (lbug-lighton? current))))
(check-expect (move-right lbug-right-10) lbug-right-11)

;; face-opposite : LBugStatus -> LBugStatus
; make a given lightning bug face toward an opposite
direction.
(define (face-opposite current)
  (cond
    [(string=? (lbug-dir current) "left")
      (light-random (make-lbug (lbug-posx current)
        "right" (lbug-lighton?
current))))]
    [(string=? (lbug-dir current) "right")
      (light-random (make-lbug (lbug-posx current)
        "left" (lbug-lighton?
current))))]
  ))
(check-expect (face-opposite lbug-right-10) lbug-left-10)

```

```

(check-expect (face-opposite lbug-left-10) lbug-right-10)

;; light-random : LBugStatus -> LBugStatus
; determine if a given lightning bug turns on or off at
random
(define (light-random current)
  (cond
    [(< (random 100) 50) ; NOTE: PLEASE SET 100 TO ENABLE
TESTING!
      (make-lbug (lbug-posx current) (lbug-dir current)
true)]
    [else
      (make-lbug (lbug-posx current) (lbug-dir current)
false)]
    ))
; omitting check-expects due to random results

; lbug-render : LBugStatus -> image
; constructs an lightning bug image representing the given
state
(define (lbug-render current)
  (cond
    [(string=? (lbug-dir current) "left")
      (cond
        [(boolean=? (lbug-lighton? current) true)
          (place-image LB_LEFT_ON_IMG (+ (lbug-posx current)
50) 100 BACKGROUND)]
        [else
          (place-image LB_LEFT_OFF_IMG (+ (lbug-posx
current) 50) 100 BACKGROUND)]
        )]
    [(string=? (lbug-dir current) "right")
      (cond
        [(boolean=? (lbug-lighton? current) true)
          (place-image LB_RIGHT_ON_IMG (+ (lbug-posx
current) 50) 100 BACKGROUND)]
        [else
          (place-image LB_RIGHT_OFF_IMG (+ (lbug-posx
current) 50) 100 BACKGROUND)]
        )]
    ))

```

```
;;;;;;;;;;  
;; Dog ;;  
;;;;;;;;;;
```

```
; Definitions
```



```
(define D_IMG_0
```



```
)
```

```
(define D_IMG_1
```



```
(define D_IMG_2
```



```
)
```

```
(define D_IMG_3
```



```
(define D_IMG_4
```



```
)
```

```
(define D_IMG_5
```



```
(define D_IMG_6
```



```
)
```



```
(define D_IMG_8
```



```
)
```



```
(define D_IMG_DIED
```

```
(define D_FEED_INTERVAL 20)
```

```
(define D_PET_INTERVAL 200)
```

```
(define-struct dog (posx posy fullness happiness taildir  
tailpos))
```

```
(define INIT_D_POSX 0)
```

```
(define INIT_D_POSY 0)
```

```
(define MAX_D_FTIME 200)
```

```
(define MAX_D_HTIME 200)
```

```
(define INIT_D_TAILDIR "down")
```

```
(define INIT_D_TAILPOS 0)
```

```
(define INIT_DOG
```

```
(make-dog INIT_D_POSX INIT_D_POSY  
MAX_D_FTIME MAX_D_HTIME  
INIT_D_TAILDIR INIT_D_TAILPOS))
```



```

(define INIT_DOG-1
  (make-dog INIT_D_POSX INIT_D_POSY
    (- MAX_D_FTIME 1) (- MAX_D_HTIME 1) "up" 1))

; Definitions for examples
(define dog-10-full-100-happy (make-dog 0 0 10 100 "down"
0))
(define dog-9-full-99-happy (make-dog 0 0 9 99 "up" 1))
(define dog-0-full-100-happy (make-dog 0 0 0 100 "down" 0))
(define dog-died (make-dog 0 0 -1 99 "up" 1))
(define dog-10-full-0-happy (make-dog 0 0 10 0 "down" 0))
(define dog-left (make-dog 0 0 9 -1 "down" 0))
(define dog-tailup (make-dog 0 0 10 200 "down" 0))
(define dog-taildown (make-dog 0 0 10 200 "down" 9))
(define dog-stop-tailup-1 (make-dog 0 0 10 99 "down" 1))
(define dog-stop-tailup-2 (make-dog 0 0 9 98 "down" 1))
(define dog-0-full (make-dog 0 0 0 0 "down" 0))
(define dog-20-full (make-dog 0 0 20 0 "down" 0))
(define dog-100-full (make-dog 0 0 100 0 "down" 0))
(define dog-120-full (make-dog 0 0 120 0 "down" 0))
(define dog-190-full (make-dog 0 0 190 0 "down" 0))
(define dog-200-full (make-dog 0 0 200 0 "down" 0))
(define dog-0-happy (make-dog 0 0 0 0 "down" 0))
(define dog-100-happy (make-dog 0 0 0 100 "down" 0))
(define dog-200-happy (make-dog 0 0 0 200 "down" 0))
(define dog-tail-down-1 (make-dog 0 0 10 200 "down" 1))
(define dog-tail-down-0 (make-dog 0 0 9 199 "down" 0))
(define dog-tail-up-1 (make-dog 0 0 8 198 "up" 1))
(define dog-tail-up-8 (make-dog 0 0 10 200 "up" 8))
(define dog-tail-up-9 (make-dog 0 0 9 199 "up" 9))
(define dog-tail-down-8 (make-dog 0 0 8 198 "down" 8))

; dog-tick : DogStatus -> DogStatus
; calculates the state following the given state if only
time passes
(define (dog-tick current)
  (cond
    [(and (<= 0 (dog-happiness current)) (< (dog-happiness
current) 100))
      (make-dog (dog-posx current) (dog-posy current)
        (decr-fullness current)
          (decr-happiness current) (dog-taildir
current) (dog-tailpos current))]
    [(or (< (dog-happiness current) 0) (< (dog-fullness
current) 0))
      (make-dog (dog-posx current) (dog-posy current)
        (decr-fullness current)
          (decr-happiness current) (dog-taildir
current) (dog-tailpos current))])

```

```

current) 0))
  current]
  [(string=? (dog-taildir current) "down")
   (taildown current)]
  [(string=? (dog-taildir current) "up")
   (tailup current)]
  ))
; Examples
(check-expect (dog-tick dog-10-full-100-happy)
dog-9-full-99-happy)
(check-expect (dog-tick dog-0-full-100-happy) dog-died)
(check-expect (dog-tick dog-10-full-0-happy) dog-left)
(check-expect (dog-tick dog-stop-tailup-1)
dog-stop-tailup-2)

;; taildown : DogStatus -> DogStatus
; move tail position to down
(define (taildown current)
  (cond
    [(= (dog-tailpos current) 0)
     (make-dog (dog-posx current) (dog-posy current)
(decr-fullness current)
      (decr-happiness current) "up" 1)]
    [else
     (make-dog (dog-posx current) (dog-posy current)
(decr-fullness current)
      (decr-happiness current) "down" (-
(dog-tailpos current) 1))])
  ))
(check-expect (taildown dog-tail-down-0) dog-tail-up-1)
(check-expect (taildown dog-tail-down-1) dog-tail-down-0)

;; tailup : DogStatus -> DogStatus
; move tail position to up
(define (tailup current)
  (cond
    [(= (dog-tailpos current) 9)
     (make-dog (dog-posx current) (dog-posy current)
(decr-fullness current)
      (decr-happiness current) "down" 8)]
    [else
     (make-dog (dog-posx current) (dog-posy current)
(decr-fullness current)
      (decr-happiness current) "up" (+

```

```

(dog-tailpos current) 1))]
  ))
(check-expect (tailup dog-tail-up-8) dog-tail-up-9)
(check-expect (tailup dog-tail-up-9) dog-tail-down-8)

;; decr-fullness : DogStatus -> dog-fullness
; decrement a fullness by a given status
(define (decr-fullness current)
  (cond
    [(< (dog-fullness current) 0)
     -1]
    [else
     (- (dog-fullness current) 1)]
  ))
(check-expect (decr-fullness dog-20-full) 19)
(check-expect (decr-fullness dog-died) -1)

;; decr-happiness : DogStatus -> dog-happiness
; decrement a happiness by a given status
(define (decr-happiness current)
  (cond
    [(< (dog-happiness current) 0)
     -1]
    [else
     (- (dog-happiness current) 1)]
  ))
(check-expect (decr-happiness dog-100-happy) 99)
(check-expect (decr-happiness dog-left) -1)

; dog-key : DogStatus KeyEvent -> DogStatus
; calculates the state following the given state if given
key is pressed
(define (dog-key current key)
  (cond
    [(or (< (dog-fullness current) 0) (< (dog-happiness
current) 0))
     current]
    [(string=? key "m") (feed-dog current)]
    [(string=? key "n") (pet-dog current)]
    [else current])) ; This code won't be executed.
; Examples

```

```

(check-expect (dog-key dog-0-full "m") dog-20-full)
(check-expect (dog-key dog-died "m") dog-died)
(check-expect (dog-key dog-0-happy "n") dog-200-happy)
(check-expect (dog-key dog-left "n") dog-left)

;; feed-dog : DogStatus -> DogStatus
; calculate how much fullness a dog gets in one feed by a
given status
(define (feed-dog current)
  (cond
    [(<= (dog-fullness current) (- MAX_D_FTIME
D_FEED_INTERVAL))
      (make-dog (dog-posx current) (dog-posy current)
        (+ (dog-fullness current)
D_FEED_INTERVAL)
          (dog-happiness current)
          (dog-taildir current) (dog-tailpos
current)))]
    [else
      (make-dog (dog-posx current) (dog-posy current)
        MAX_D_FTIME
          (dog-happiness current)
          (dog-taildir current) (dog-tailpos
current)))]
    ))
(check-expect (feed-dog dog-100-full) dog-120-full)
(check-expect (feed-dog dog-190-full) dog-200-full)

;; pet-dog : DogStatus -> DogStatus
; calculate how much happiness a dog gets in one pet by a
given status
(define (pet-dog current)
  (cond
    [(<= (dog-happiness current) (- MAX_D_HTIME
D_PET_INTERVAL))
      (make-dog (dog-posx current) (dog-posy current)
        (dog-fullness current)
          (+ (dog-happiness current)
D_PET_INTERVAL)
          (dog-taildir current) (dog-tailpos
current)))]
    [else

```

```

    (make-dog (dog-posx current) (dog-posy current)
      (dog-fullness current)
        MAX_D_HTIME
        (dog-taildir current) (dog-tailpos
current)))])
  ))
  (check-expect (pet-dog dog-0-happy) dog-200-happy)
  (check-expect (pet-dog dog-100-happy) dog-200-happy)

```

```

; dog-render : DogStatus -> image
; constructs an image representing the given state
(define (dog-render current)
  (overlay/xy (create-dog-image current)
    0 70
    (create-meters current)
    ))
(check-expect (dog-render dog-10-full-100-happy)

```



```

(check-expect (dog-render dog-died)

```

```

;; create-dog-image : current -> image
; create an dog image with tail down or tail up by a given
status.
(define (create-dog-image current)
  (cond
    [(< (dog-fullness current) 0) D_IMG_DIED]
    [(< (dog-happiness current) 0) EMPTY_SCENE]
    [(= (dog-tailpos current) 0) D_IMG_0]
    [(= (dog-tailpos current) 1) D_IMG_1]

```

```

[ (= (dog-tailpos current) 2) D_IMG_2]
[ (= (dog-tailpos current) 3) D_IMG_3]
[ (= (dog-tailpos current) 4) D_IMG_4]
[ (= (dog-tailpos current) 5) D_IMG_5]
[ (= (dog-tailpos current) 6) D_IMG_6]
[ (= (dog-tailpos current) 7) D_IMG_7]
[ (= (dog-tailpos current) 8) D_IMG_8]
[ (= (dog-tailpos current) 9) D_IMG_9]
))
(check-expect (create-dog-image dog-tailup) D_IMG_0)
(check-expect (create-dog-image dog-taildown) D_IMG_9)

```



```

(check-expect (create-dog-image dog-died) )
(check-expect (create-dog-image dog-left) EMPTY_SCENE)

```

```

;; create-meters : DogStatus -> image
; create/disappear a feed meter and happiness meter by a
given status,
; and put them into one image.
(define (create-meters current)
  (cond
    [(or (< (dog-fullness current) 0) (< (dog-happiness
current) 0))]
    EMPTY_SCENE]
    [else
     (overlay/xy (create-d-fmeter (dog-fullness current))
                 0 30
                 (create-hmeter (dog-happiness current))
                 )]
  ))

```



```

(check-expect (create-meters dog-10-full-100-happy) )
(check-expect (create-meters dog-left) EMPTY_SCENE)



```

```

;; create-d-fmeter : ftime -> image
; create an image of feed meter by a given time.
(define (create-d-fmeter ftime)
  (rectangle (/ ftime 2) 20 "solid" "red"))



```

`;Examples`

```
(check-expect (create-d-fmeter 200) )
(check-expect (create-d-fmeter 100) )
```

```
;; create-hmeter : htime -> image
; create an image of happiness meter by a given time.
(define (create-hmeter htime)
  (rectangle (/ htime 2) 20 "solid" "orange"))
```

`;Examples`

```
(check-expect (create-hmeter 200) )
(check-expect (create-hmeter 100) )
```

```
;;;;;;;;;;
;; Main ;;
;;;;;;;;;;
```

`;; Definitions for examples`

```
(define INIT_STAT
  (make-stat INIT_TURTLE INIT_LBUG INIT_DOG))
(define INIT_STAT-1
  (make-stat INIT_TURTLE-1 INIT_LBUG-1 INIT_DOG-1))
```

```
;; main-tick : Status -> Status
; calculates the state following the given state if only
time passes
```

```
(define (main-tick current)
  (make-stat
    (turtle-tick (stat-turtle current))
    (lbug-tick (stat-lbug current))
    (dog-tick (stat-dog current))
  ))
(check-expect (main-tick INIT_STAT) INIT_STAT-1)
```

`;; main-key : Status -> KeyEvent`

```
; calculates the state following the given state if given
key is pressed
```

```
(define (main-key current kevent)
  (cond
```

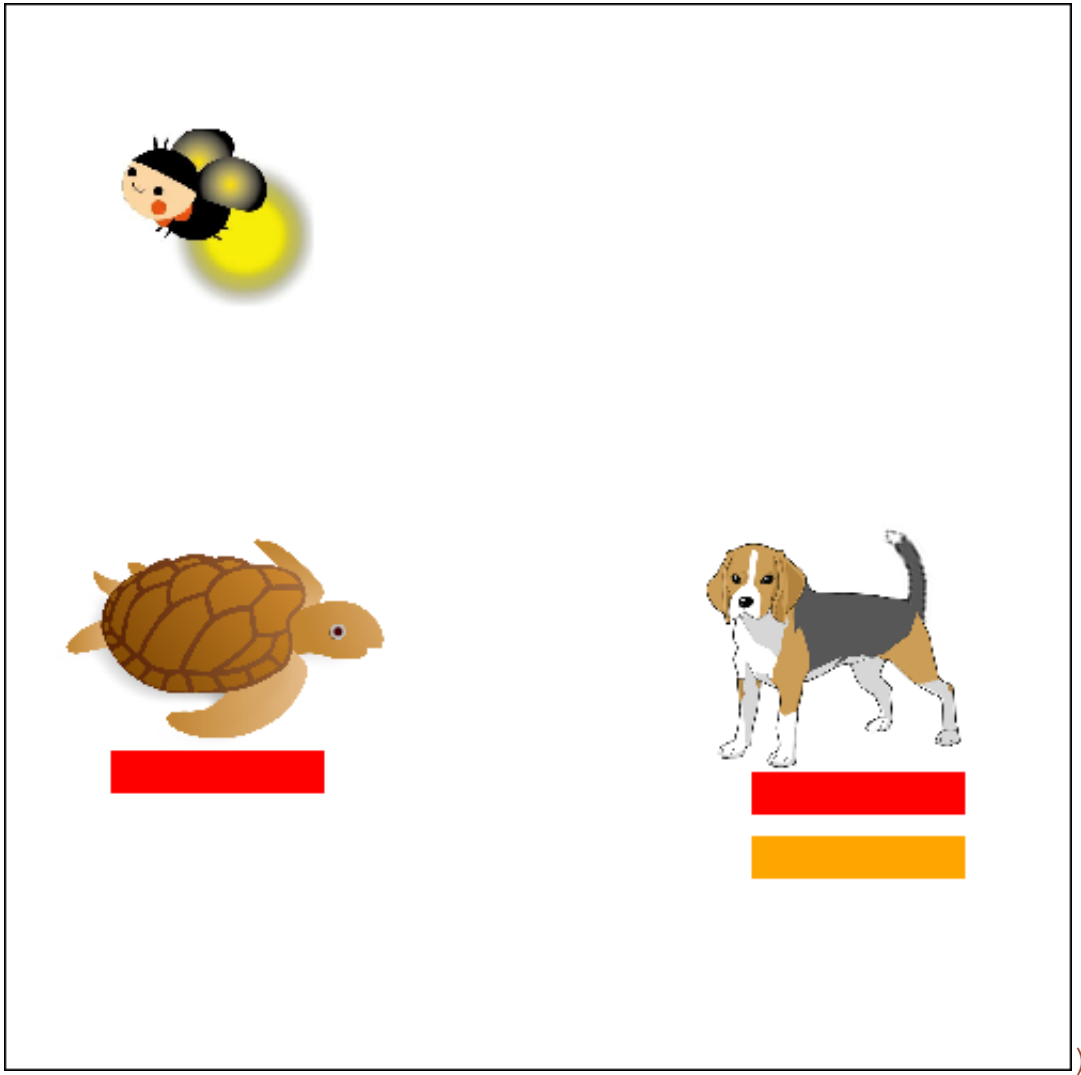
```

[ (or (string=? kevent "m") (string=? kevent "n"))
  (make-stat (stat-turtle current)
              (stat-lbug current)
              (dog-key (stat-dog current) kevent))]
[ (string=? kevent "z")
  (make-stat (turtle-key (stat-turtle current) kevent)
              (stat-lbug current)
              (stat-dog current))]
[else current]
))
(check-expect (main-key INIT_STAT "z") INIT_STAT)
(check-expect (main-key INIT_STAT "m") INIT_STAT)
(check-expect (main-key INIT_STAT "n") INIT_STAT)

;; main-render : Status -> Status
; constructs an whole image representing the given state
(define (main-render current)
  (overlay/xy (lbug-render (stat-lbug current))
              100 300
              (overlay/xy (turtle-render (stat-turtle
current))
                          300 0
                          (dog-render (stat-dog current))))
  ))
(check-expect (main-render INIT_STAT)

```





```
(define (main current)
  (big-bang current
    (on-tick main-tick)
    (on-key main-key)
    (to-draw main-render)
  ))
```

```
(main INIT_STAT)
```