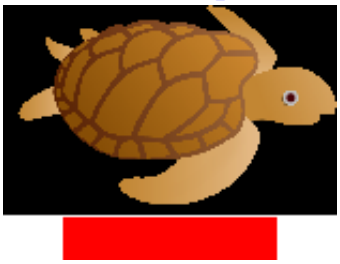
```
;;;;;;;;;;;;;;;;;; VZ proj ;;;;;;;;;;;;;;;;;;
;;;                Yohei Yasukawa              ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; 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) [redacted])
```

```
(check-expect (create-t-fmeter 200) [redacted])
(check-expect (create-t-fmeter 100) [redacted])


; 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 posy dir lighton?))
(define INIT_LBUG (make-lbug 250 50 "left-down" true))
(define INIT_LBUG-1 (make-lbug 249 51 "left-down" true))
;(define INIT_LBUG (make-lbug 250 50 "right-up" true))  ;
For further testing
;(define INIT_LBUG-1 (make-lbug 251 49 "right-up" true)) ;
For further testing

; Definitions for lbug examples
(define lbug-size 100)
(define lbug-edgeL 0)
```

```
(define lbug-edgeR (- BG_WIDTH lbug-size))
(define lbug-edgeT 0)
(define lbug-edgeB (- BG_HEIGHT 75))
(define lbug-ld-10-10 (make-lbug 10 10 "left-down" true))
(define lbug-ld-9-11 (make-lbug 9 11 "left-down" true))
(define lbug-ld-edgeL-10 (make-lbug lbug-edgeL 10
"left-down" true))
(define lbug-rd-edgeL-10 (make-lbug lbug-edgeL 10
"right-down" true))
(define lbug-rd-10-10 (make-lbug 10 10 "right-down" true))
(define lbug-rd-11-11 (make-lbug 11 11 "right-down" true))
(define lbug-rd-edgeR-10 (make-lbug lbug-edgeR 10
"right-down" true))
(define lbug-ld-edgeR-10 (make-lbug lbug-edgeR 10
"left-down" true))
(define lbug-lu-10-10 (make-lbug 10 10 "left-up" true))
(define lbug-lu-9-9 (make-lbug 9 9 "left-up" true))
(define lbug-lu-10-edgeT (make-lbug 10 lbug-edgeT
"left-up" true))
(define lbug-ru-10-edgeT (make-lbug 10 lbug-edgeT
"right-up" true))
(define lbug-ru-10-10 (make-lbug 10 10 "right-up" true))
(define lbug-ru-11-9 (make-lbug 11 9 "right-up" true))
(define lbug-ru-10-edgeB (make-lbug 10 lbug-edgeB
"right-up" true))
(define lbug-lu-10-edgeB (make-lbug 10 lbug-edgeB
"left-up" 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-down")
     (cond
       [(touch-left-wall? current) (flip-horizontally
current)]
       [(touch-bottom-wall? current) (flip-vertically
current)]
       [else (move-left-down current)]
       )]
    [(string=? (lbug-dir current) "right-down")
     (cond
       [(touch-right-wall? current)(flip-horizontally
```

```
current)]
       [(touch-bottom-wall? current) (flip-vertically
current)]
       [else (move-right-down current)]
       )]
    [(string=? (lbug-dir current) "left-up")
     (cond
       [(touch-left-wall? current) (flip-horizontally
current)]
       [(touch-top-wall? current) (flip-vertically
current)]
       [else (move-left-up current)]
       )]
    [(string=? (lbug-dir current) "right-up")
     (cond
       [(touch-right-wall? current) (flip-horizontally
current)]
       [(touch-top-wall? current) (flip-vertically
current)]
       [else (move-right-up current)]
       )]
    ))
(check-expect (lbug-tick lbug-ld-10-10) lbug-ld-9-11)
(check-expect (lbug-tick lbug-ld-edgeL-10)
lbug-rd-edgeL-10)
(check-expect (lbug-tick lbug-rd-10-10) lbug-rd-11-11)
(check-expect (lbug-tick lbug-rd-edgeR-10)
lbug-ld-edgeR-10)


;; 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-ld-edgeL-10) true)
(check-expect (touch-left-wall? lbug-ld-10-10) 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-rd-10-10) false)
(check-expect (touch-right-wall? lbug-rd-edgeR-10) true)

;; touch-top-wall? : LBugStatus -> boolean
; determine if a given lightning bug is touching a wall at
the top
(define (touch-top-wall? current)
  (cond
    [(<= (lbug-posy current) lbug-edgeT) true]
    [else false]
    ))
(check-expect (touch-top-wall? lbug-ru-10-10) false)
(check-expect (touch-top-wall? lbug-ru-10-edgeT) true)

;; touch-bottom-wall? : LBugStatus -> boolean
; determine if a given lightning bug is touching a wall at
the bottom
(define (touch-bottom-wall? current)
  (cond
    [(>= (lbug-posy current) lbug-edgeB) true]
    [else false]
    ))
(check-expect (touch-bottom-wall? lbug-lu-10-edgeB) true)
(check-expect (touch-bottom-wall? lbug-lu-10-10) false)

;; move-left-down : LBugStatus -> LBugStatus
; move a given lightning bug to the left down in 1 px
(define (move-left-down current)
  (light-random
   (make-lbug (- (lbug-posx current) 1) (+ (lbug-posy
current) 1)
              (lbug-dir current) (lbug-lighton? current))
   ))
(check-expect (move-left-down lbug-ld-10-10) lbug-ld-9-11)


;; move-right-down : LBugStatus -> LBugStatus
; move a given lightning bug to the right down in 1 px
(define (move-right-down current)
```

```
  (light-random
   (make-lbug (+ (lbug-posx current) 1) (+ (lbug-posy
current) 1)
              (lbug-dir current) (lbug-lighton? current))))
(check-expect (move-right-down lbug-rd-10-10)
lbug-rd-11-11)


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


;; move-right-up : LBugStatus -> LBugStatus
; move a given lightning bug to the right in 1 px
(define (move-right-up current)
  (light-random
   (make-lbug (+ (lbug-posx current) 1) (- (lbug-posy
current) 1)
              (lbug-dir current) (lbug-lighton? current))))
(check-expect (move-right-up lbug-ru-10-10) lbug-ru-11-9)


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

```
    [(string=? (lbug-dir current) "left-up")
     (light-random (make-lbug (lbug-posx current)
(lbug-posy current)
                              "right-up" (lbug-lighton?
current)))]
    [(string=? (lbug-dir current) "right-up")
     (light-random (make-lbug (lbug-posx current)
(lbug-posy current)
                               "left-up" (lbug-lighton?
current)))]
    ))
(check-expect (flip-horizontally lbug-rd-10-10)
lbug-ld-10-10)
(check-expect (flip-horizontally lbug-ld-10-10)
lbug-rd-10-10)


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

```
lbug-ru-10-10)
(check-expect (flip-vertically lbug-ru-10-10)
lbug-rd-10-10)


;; light-random : LBugStatus -> LBugStatus
; determine if a given lightning bug turns on or off at
random
(define (light-random current)
  (cond
    [(< (random 100) 100) ; NOTE: PLEASE SET 100 WHEN
TESTING!
     (make-lbug (lbug-posx current) (lbug-posy current)
(lbug-dir current) true)]
    [else
     (make-lbug (lbug-posx current) (lbug-posy 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
    [(or (string=? (lbug-dir current) "left-down")
         (string=? (lbug-dir current) "left-up"))
     (create-lbug-image-left current)]
    [(or (string=? (lbug-dir current) "right-down")
         (string=? (lbug-dir current) "right-up"))
     (create-lbug-image-right current)]
    ))

(define (create-lbug-image-left current)
  (cond
    [(boolean=? (lbug-lighton? current) true)
     (place-image LB_LEFT_ON_IMG
                  (+ (lbug-posx current) 50)
                  (+ (lbug-posy current) 50)
                  BACKGROUND)]
    [else
     (place-image LB_LEFT_OFF_IMG
                  (+ (lbug-posx current) 50)
```

```
                    (+ (lbug-posy current) 50)
                    BACKGROUND)]
    ))

(define (create-lbug-image-right current)
  (cond
    [(boolean=? (lbug-lighton? current) true)
     (place-image LB_RIGHT_ON_IMG
                  (+ (lbug-posx current) 50)
                  (+ (lbug-posy current) 50)
                  BACKGROUND)]
    [else
     (place-image LB_RIGHT_OFF_IMG
                  (+ (lbug-posx current) 50)
                  (+ (lbug-posy current) 50)
                  BACKGROUND)]
    ))


;;;;;;;;;
;; Dog ;;
;;;;;;;;;

; Definitions
```

```
(define D_IMG_0
```

) (define D_IMG_1

)

```
(define D_IMG_2
```

) (define D_IMG_3

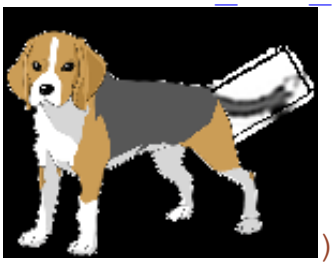

)

(define D_IMG_4 ) (define D_IMG_5

)

(define D_IMG_6 ) (define D_IMG_7

)

(define D_IMG_8 ) (define D_IMG_9

)

```
(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 300)
(define INIT_D_POSY 300)
(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))
     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]))
; 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)
```

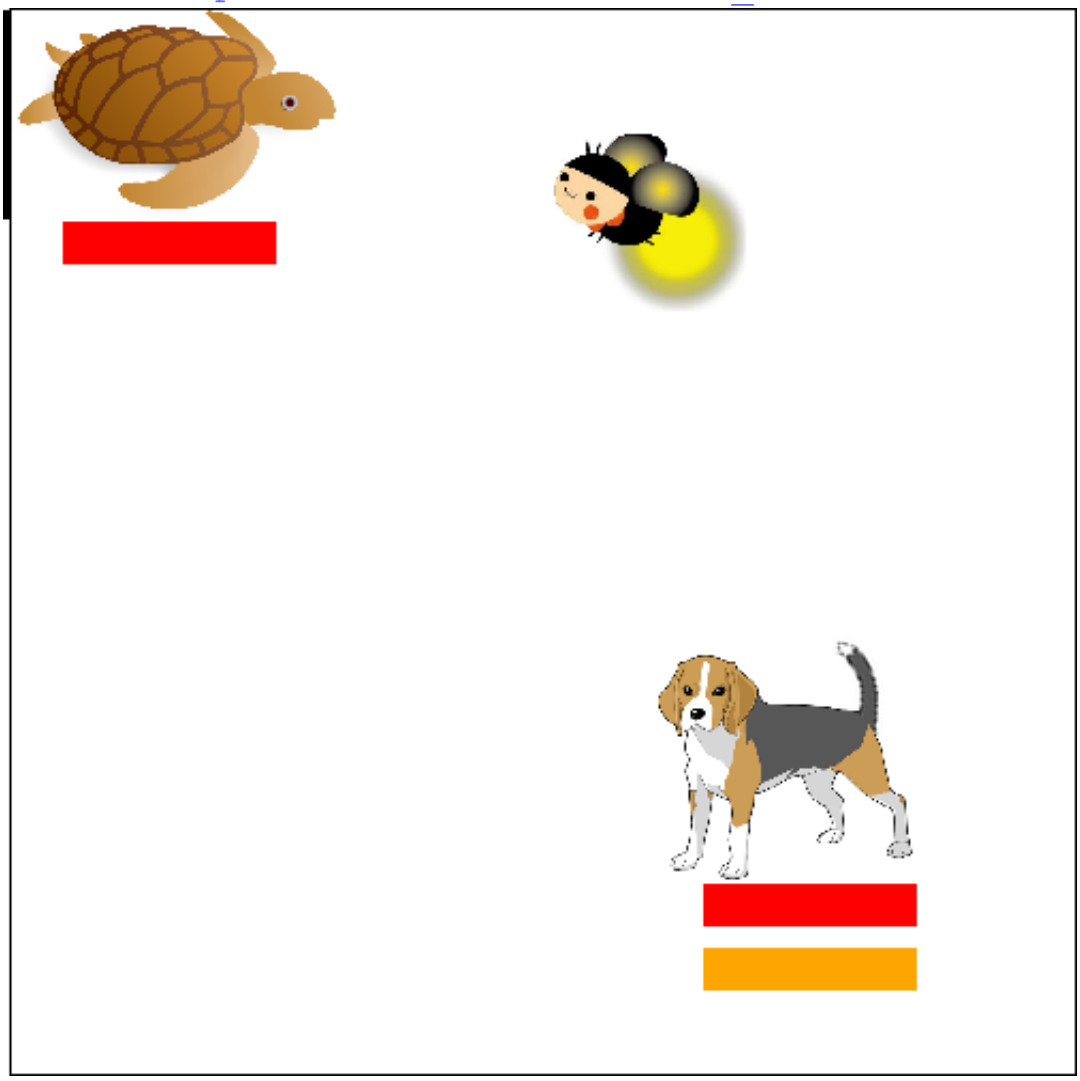
```
)
(check-expect (create-hmeter 100)
```

```
)



;;;;;;;;;;
;; 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))]
    [(and (string=? kevent " ")
          (>= (dog-fullness (stat-dog current)) 0)
          (>= (dog-happiness (stat-dog current)) 0))
     (dogjump4lbug 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)

;; dogjump4lbug : Status -> Status
```

```
; have a dog jump toward a lightning bug by a given state.
(define (dogjump4lbug current)
  (make-stat (stat-turtle current) (stat-lbug current)
             (make-dog (calc-jump-x current) (calc-jump-y
current)
                       (dog-fullness (stat-dog current))
                       (dog-happiness (stat-dog current))
                       (dog-taildir (stat-dog current))
                       (dog-tailpos (stat-dog current))
                       )
             ))

; offset to jumping to a lightning bug
(define dogjump-offset 3)

;; calc-jump-x : Status -> dog-posx
; calculate XCoord of a dog by a give status
(define (calc-jump-x current)
  (cond
    [(< (dog-posx (stat-dog current)) (lbug-posx
(stat-lbug current)))
     (+ (/ (abs (- (dog-posx (stat-dog current))
                   (lbug-posx (stat-lbug current))))
dogjump-offset)
          (dog-posx (stat-dog current)))]
    [else
     (- (dog-posx (stat-dog current))
        (/ (abs (- (dog-posx (stat-dog current))
                   (lbug-posx (stat-lbug current))))
dogjump-offset)
        )]
    ))

;; calc-jump-y : Status -> dog-posx
; calculate YCoord of a dog by a give status
(define (calc-jump-y current)
  (cond
    [(< (dog-posy (stat-dog current)) (lbug-posy
(stat-lbug current)))
     (+ (/ (abs (- (dog-posy (stat-dog current))
                   (lbug-posy (stat-lbug current))))
dogjump-offset)
        (dog-posy (stat-dog current)))]
    [else
```

```
      (- (dog-posy (stat-dog current))
         (/ (abs (- (dog-posy (stat-dog current))
                    (lbug-posy (stat-lbug current))))
dogjump-offset)
         )]
     ))

;; main-render : Status -> Status
; constructs an whole image representing the given state
(define (main-render current)
  (overlay/xy (lbug-render (stat-lbug current))
              75 50
              (overlay/xy (turtle-render (stat-turtle
current))
                          (dog-posx (stat-dog current))
                          (dog-posy (stat-dog current))
                          (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)
```